Information Retrieval

Isa-Ali Kirca

Contents

1 Week 1: Recap IR0 **2**
- 1.1 Document Parsing . . . 4
- 1.2 Tokenizing . . . 4
- 1.3 Stopping . . . 4
- 1.4 Stemming . . . 5
- 1.5 Phrases and N-grams . . . 5
- 1.6 Ranking with indexes . . . 6
- 1.7 Slides IR0 recap . . . 7

2 Week 2: Evaluation **9**
- 2.1 Test collections . . . 9
- 2.2 Metrics . . . 10

3 Week 3: Document representation and matching **13**
- 3.1 Query analysis . . . 13
- 3.2 Query processing . . . 13
- 3.3 Termbased Retrieval . . . 13
- 3.4 Semantic-based Retrieval . . . 16
- 3.5 Semantic retrieval: Neural Models for ranking . . . 19

4 Week 4: Learning to Rank (LTR) and Interactions **22**
- 4.1 Preliminaries and Goal . . . 22
- 4.2 Offline LTR . . . 23
- 4.3 Pointwise approach . . . 23
- 4.4 Pairwise approach . . . 24
- 4.5 Listwise approach . . . 27
- 4.6 ListNet and ListMLE . . . 28
- 4.7 User interactions . . . 29

1 Week 1: Recap IR0

After gathering the text we want to search, the next step is to decide whether it should be modified or restructured to simplify searching. The types of changes that are made at this stage are called **text transformation** or, more often, **text processing**.

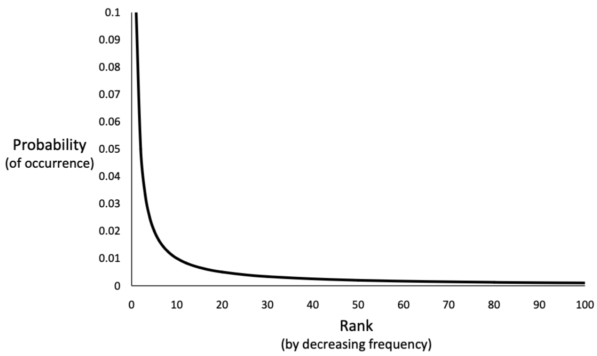
Goal of **text processing**: convert the many forms in which words can occur into more consistent index terms.

Index terms	representation of the content of a document used for searching.
Tokenization	words are split apart. Process of forming words from the sequence of characters in a document.
Stopping	some words may be ignored entirely in order to make query processing more effective and efficient.
Stemming	allow similar words (like "run" and "running") to match each other.

Statistical models of **word occurrences** are very **important** in information retrieval, and are used in many of the core components of search engines, such as the ranking algorithms, query transformation, and indexing techniques. One of the most obvious features of text is that the **distribution of word frequencies** is very **skewed**.

Zipf's Law: the frequency of the rth most common word is inversely proportional to r or, alternatively, the rank of a word times its frequency (f) is approx. a constant (k): $r \cdot f = k$.

But we want the probability of occurrence of a word, which is the frequency of the word divided by the total number of word occurrences in the text. In this case, Zipf's law is: $r \cdot P_r = c$, where P_r is the probability of occurrence for the rth ranked word, and c is a constant.



Hapax Legomena: words that occur once in a text corpus or book.

Heaps' law: Relationship between size of corpus and size of vocabulary.

$$v = k \cdot n^{\beta}$$

v = vocabulary size for a corpus of size n words
k and β = parameters that vary for each collection.

Heaps' law predicts that the number of new words will increase very rapidly when the corpus is small and will continue to increase indefinitely, but slower for larger corpora.

Word occurrence statistics can also be used to estimate the size of the results from a web search. A **result** is any document (or web page) that contains all of the query words.

If we assume that words occur ***independently*** of each other, then the probability of a document containing all the words in the query is simply the **product of the probabilities** of the individual words occurring in a document. For example, if there are three query words a, b, and c, then:

$P(a \cap b \cap c) = P(a) \cdot P(b) \cdot P(c)$	$P(a \cap b \cap c)$ = joint probability, or the probability that all three words occur in a document $P(a), P(b), P(c)$ = are the probabilities of each word occurring in a document.

A search engine will always have access to the number of documents that a word occurs in (f_a, f_b, and f_c), and the number of documents in the collection (N), so these probabilities can easily be estimated as $P(a) = f_a/N, P(b) = f_b/N$, and $P(c) = f_c/N$:

$f_{abc} = N \cdot f_a/N \cdot f_b/N \cdot f_c/N = (f_a \cdot f_b \cdot f_c)/N^2$ where f_{abc}: estimated size of result set.

But, this assumption does **not** lead to good estimates for result size, especially for **combinations** of **three words**. The problem is that the words in these combinations do not occur independently of each other. If we see the word "fish" in a document, for example, then the word "aquarium" is more likely to occur in this document than in one that does not contain "fish".

Better estimates are possible if **word co-occurrence** information is also available. Obviously, this would give exact answers for two-word queries. For longer queries, we can improve the estimate by not assuming independence. In general, for three words:

$P(a \cap b \cap c) = P(a \cap b) \cdot P(c \mid (a \cap b))$	$P(a \cap b)$ = probability that the words a and b co-occur in a document $P(a), P(b), P(c)$ = probability that the word c occurs in a document given that the words a and b occur in the document.

These estimates are much better than the ones produced assuming independence, but they are **still too low**.

Skipped the last few paragraphs of 4.2, because I don't think this is very important.

1.1 Document Parsing

Document parsing involves recognizing the content and structure of text document.

Metadata is information about a document that is not part of the text content. Metadata content includes document attributes such as date and author, and, most importantly, the **tags** that are used by **markup languages** to identify document components. Most popular are HTML and XML.

The **parser** uses the **tags** and other metadata recognized in the document to interpret the document's structure based on the syntax of the markup language (**syntactic analysis**) and to produce a representation of the document that includes both the structure and content.

1.2 Tokenizing

Tokenizing is the process of forming words from the sequence of characters in a document. Some examples of issues involving tokenizing that can have significant impact on the effectiveness of search are:

- **Small words**: can be important in combination with other words. For example, master, world war II.
- **Hyphenated and non-hyphenated forms of words**
- **Special characters**: important part of the tags, URLs, code, and other important parts of documents that must be correctly tokenized.
- **Capitalized words**: can have different meaning from lowercase words. For example, "Bush" and "Apple".
- **Apostrophes**
- **Numbers, including decimals**: for example, nokia 3250, quicktime 6.5 pro.
- **Periods can occur in numbers, abbreviations** (e.g., "I.B.M.", "Ph.D."), URLs, ends of sentences, and other situations.

Text processing for queries **must** be the **same** as that used for documents. Otherwise, many of the index terms will simply not match the corresponding terms.

1.3 Stopping

Human language is filled with function words: words that have little meaning apart from other words. The most popular—"the," "a," "an," "that," and "those"—are determiners. These words are part of how we describe nouns in text, and express concepts like location or quantity. Prepositions, such as "over," "under," "above," and "below," represent relative position between two nouns.

In information retrieval, these function words have a second name: **stopwords**. We call them stopwords because text processing stops when one is seen, and they are thrown out. **Throwing out** these words **decreases index size**, **increases retrieval efficiency**, and generally **improves retrieval effectiveness**.

Constructing a stopword list must be done with **caution**. Removing too many words will **hurt** retrieval effectiveness.

If storage space requirements allow, it is best to at least index all words in the documents. If stopping is required, the stopwords can always be removed from queries. If keeping stopwords in an index is not possible because of space requirements, as few as possible should be removed in order to maintain maximum flexibility.

1.4 Stemming

Stemming, also called **conflation**, is a component of text processing that captures the relationships between different variations of a word. More precisely, **stemming** **reduces** the different forms of a word that occur because of ***inflection*** (e.g., plurals, tenses) or ***derivation*** (e.g., making a verb into a noun by adding the suffix -ation) to a common stem.

There are two basic types of **stemmers**: algorithmic and dictionary-based. An algorithmic stemmer uses a small program to decide whether **two words** are **related**, usually based on knowledge of word suffixes for a particular language. By contrast, a dictionary-based stemmer has no logic of its own, but instead **relies** on **pre-created dictionaries** of related terms to store term relationships.

Algorithmic stemmers:

- **Suffix-s stemmer**: assumes any word ending in the letter "s" is plural. Cakes → cake.
- **Porter stemmer**: consists of a number of steps, each containing a set of rules for removing suffixes. At each step, the rule for the longest applicable suffix is executed.

Dictionary (Hybrid)-based stemmers:

- **Krovetz stemmer**: makes constant use of a dictionary to check whether the word is valid. The Krovetz stemmer has the additional advan- tage of producing stems that, in most cases, are full words, whereas the Porter stemmer often produces stems that are word fragments.

Incorporating language-specific stemming algorithms is one of the most important aspects of customizing, or **internationalizing**, a search engine for multiple languages.

1.5 Phrases and N-grams

A **phrase** is equivalent to a simple **noun phrase**. This is often restricted even further to include just sequences of nouns, or adjectives followed by nouns. Phrases defined by these criteria can be identified using a **part-of-speech (POS) tagger**. A POS tagger marks the words in a text with labels corresponding to the part-of-speech of the word in that context. Taggers are based on statistical or rule-based approaches and are trained using large corpora that have been manually labeled.

N-gram: any sequence of n words.
Unigram: single words.
Bigrams: sequences of two words.
Unigram: sequences of three words.

The more frequently a word **N-gram** occurs, the **more likely** it is to correspond to a **meaningful phrase in the language**. N-grams of all lengths form a Zipf distribution, with a few common phrases occurring very frequently and a large number occurring with frequency 1. In fact, the rank-frequency data for n-grams (which includes single words) fits the Zipf distribution better than words alone.

1.6 Ranking with indexes

Unsorted arrays are slow to search, and sorted arrays are slow at insertion. By contrast, hash tables and trees are fast for both search and insertion. These structures are more complicated than arrays, but the speed difference is compelling.

Text search is very different from traditional computing tasks, so it calls for its own kind of data structure, the **inverted index**.

1.7 Slides IR0 recap

Outline text analysis:

- **Statistical properties of written text** (Zipf's law and Heaps' law)
- **Text analysis pipeline**
    1. Remove white-spaces and punctuation
    2. Convert terms to lower-case
    3. Remove stop-words
        - Frequency-based:
            * Set a frequency threshold f
            * Remove words with the frequency higher than f
        - Dictionary-based:
            * Create a dictionary of stop-words
            * Remove words that occur in this dictionary
    4. Convert terms to their stems
    5. Deal with phrases
    6. Apply language-specific processing rules
- **Stemming**
    - Algorithmic: Porter-stemmer
    - Dictionary-based
        * Large dictionary of related words
        * Semi-automatic: run → running, runs, runned, runly
        * New-words problem
    - Hybrid: produces words, **not stems**. Comparable effectiveness with the Porter stemmer
        * check the word in a dictionary, **if found**, leave it as is
        * if not found, apply algorithmic stemming (remove suffixes)
        * check the dictionary again
        * if not found, apply rules to modify the ending
- **Phrases**
    1. detect noun phrases using a part-of-speech tagger
        - sequences of nouns
        - adjectives followed by nouns
    2. Detect phrases at the query processing time
       Use index with word positions
    3. Use frequent **n-grams**, e.g., bigrams and trigrams

Outline indexing:

1. **Data structures**: Web Graph, Forward index, Page attribute file, Inverted index
2. **Inverted index**
   a) Dictionary
      - Each entry contains
        - Number of pages containing the term
        - Pointer to the start of the inverted list
        - Other meta-data about the term
      - B+ tree, hash table

   b) Inverted lists
      - Document identifiers
      - Frequencies
      - Positions
      - Weights
3. **Constructing an index**
   - Simple indexer
     Problems of this indexer:
     a) In-memory:
        - Two-pass index
        - One-pass index with merging

     b) Single-threaded
        - Distributed indexing (MapReduce)
4. **Updating an index**
   Strategies:
   - No merge (low index maintenance cost, high query processing cost)
   - Incremental update
   - Immediate merge (always a single index)
   - Lazy merge (trade-off between index maintenance and query processing cost)

   Page deletions:
   - Maintain identifiers of deleted documents in memory, access during query processing
   - Garbage collection (e.g., during index merging)

2 Week 2: Evaluation

Evaluation is the key to making progress in building better search engines.

What is different about evaluation in IR? Many possible queries (cannot be enumerated) and documents (most cannot be judged) and expect good "zero-shot" performance.

- **Boolean retrieval**: considers only whether a document is relevant
  - Does not estimate a **degree** of relevance (i.e., no ranking)
  - Acceptable for some use cases, especially recall oriented
  - Limitations for large result sets
- **Ranked retrieval**: sorts (ranks) documents by estimated relevance
  - Relevance score estimated for each document
  - Common setting (e.g., Web search)
  - No expectation that user consumes entire ranked list
  - Evaluation metrics characterize different trade-offs

Outline offline evaluation:

2.1 Test collections

- **Components of test collections**
  - **Test documents**: repr. for appl. in terms of the nr., size, and type.
  - **Test queries**: Queries from pot. users (query log). More is better, at least 50.
  - **Ground Truth/Relevance judgements**: Search result relevant?
    * Where to get: users, independent judges, crowdsourcing.
    * How many: more the better. More judged queries, fewer judgements per query. Multiple judges.
    * Graded relevance: 4 - perfect, 3 - excellent, 2 - good, 1 - fair, 0 - bad.
  - **But, impossible to obtain judgements for all documents. So:** **Depth-k pooling**: Produces a large number of judgments for each query
    1. consider multiple search systems (by participants)
    2. consider top-k results from each system
    3. remove duplicates
    4. present documents to judges in a random order
  - still **incomplete**.

- **Multiple assessors**
  - **Inter-assessor agreement**, Cohen's kappa coefficient:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \qquad P(E) = \text{Expected chance agreement}$$

  - Values:
    - > 0.8: high
    - 0.67 - 0.8: acceptable
    - < 0.67: low
  - For more than two assessors, average pair-wise coefficients

- **Evaluation campaigns**
  - Text REtrieval Conference (TREC)
  - Cross-Language Education and Function (CLEF)
  - NII Test Collections for IR (NTCIR)

2.2 Metrics

- **Unranked evaluation**
  - **Precision**: is the fraction of retrieved items that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  - **Recall**: is the fraction of relevant items that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

	Relevant	**Non-relevant**
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

  - **F-measure**

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{\left(\beta^2 + 1\right) PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

  - **F1-measure** $(\alpha = 0.5, \beta^2 = 1)$

$$F_1 = \frac{2PR}{P + R}$$

  - **Ranking of items is not taken into account!**

- **Ranked evaluation**
  - **Precision** at rank k $\quad P@k = \frac{\#(\text{relevant items at k})}{k}$
  - **Recall** at rank k $\quad R@k = \frac{\#(\text{relevant items at k})}{\#(\text{relevant items})}$
  - **Reciprocal rank (RR)** $\quad RR = \frac{1}{\text{rank of first relevant item}}$
  - **Average precision (AP)** $\quad AP = \frac{\sum_{d \in rel} P@k_d}{\#(\text{ relevant items })}$
  - **Average over multiple queries**
    * mean $P@k$
    * MRR
    * mean $R@k$
    * MAP
  - **User search behavior is not taken into account!**

- **User-oriented evaluation**
  - **Discounted cumulative gain (DCG)**
    * Graded relevance $\quad R_k \in \{0, 1, 2, 3, 4\}$
    * Cumulative Gain: $\quad CG = \sum_{k=1}^{N} (2^{R_k} - 1)$
    * Gain is discounted by rank: $\quad D(k) = \frac{1}{\log(k+1)}$
    * Discounted cumulative gain: $\quad DCG = \sum_{k=1}^{N} \frac{2^{R_k}-1}{\log(k+1)}$
    * Normalized DCG: $\quad NDCG = \frac{DCG}{DCG_{ideal}}$
  - **Rank-biased precision (RBP)**
    * view next item with probability $\quad \theta$
    * stop with probability $\quad 1 - \theta$
    * Probability of looking at rank k $\quad P(\text{look at } k) = \theta^{k-1}$
    * Average number of examined items

$$\begin{aligned}\text{Avg. exam} &= \sum_{k=1}^{\infty} k \cdot P(\text{look at } k) \cdot P(\text{stop at } k) \\ &= \sum_{k=1}^{\infty} k \cdot \theta^{k-1} \cdot (1 - \theta) \quad = \quad \frac{1}{1 - \theta}\end{aligned}$$

    * Utility at rank k $\quad U@k = P(\text{look at } k) \cdot R_k = \theta^{k-1} \cdot R_k$
    * Avg. utility of all results

$$RBP = \frac{\sum_{k=1}^{N} U@k}{\text{Avg. exam}} = (1 - \theta) \cdot \sum_{k=1}^{N} \theta^{k-1} \cdot R_k$$

    * θ is usually close to 1

- **Expected reciprocal rank (ERR)**
  * Reciprocal rank $RR = \frac{1}{\text{rank of first relevant item}}$
  * If an item is relevant (R_k) then stop
  * Otherwise $(1 - R_k)$, continue with probability θ
  * Probability of looking at rank k $P(\text{look at } k) = \prod_{i=1}^{k-1} ((1 - R_i) \cdot \theta)$
  * Probability of reciprocal rank $= \frac{1}{k}$ $P\left(RR = \frac{1}{k}\right) = R_k \cdot \prod_{i=1}^{k-1} ((1 - R_i) \cdot \theta)$
  * Expected reciprocal rank

$$
\begin{aligned}
ERR &= \sum_{k=1}^{N} \frac{1}{k} \cdot P(RR = \frac{1}{k}) \\
&= \sum_{k=1}^{N} \frac{1}{k} \cdot \theta^{k-1} \cdot R_k \cdot \prod_{i=1}^{k-1} (1 - R_i)
\end{aligned}
$$

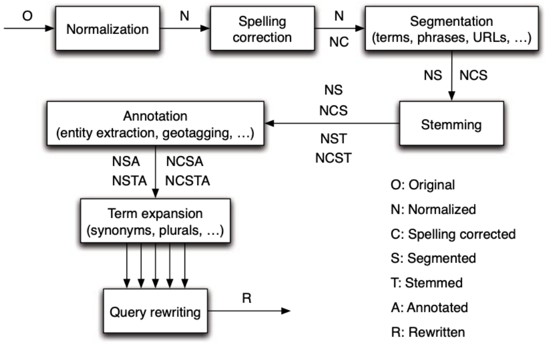
  * θ is usually close to 1

3 Week 3: Document representation and matching

3.1 Query analysis

Pipeline: should be the same as document processing pipeline.

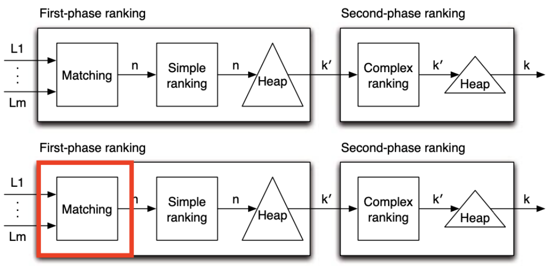
- Normalization
- Spelling correction
- Segmentation
- Stemming
- Term expansion, etc..



3.2 Query processing

Practical considerations:

- Conjunctive mode (AND)
- Document-at-a-time
- A score is usually computed as a linear combination of query-dependent and query-independent scores



3.3 Termbased Retrieval

1. **Vector space model**
   - Document as vector: binary occurrence of a term in a document (each document is a vector)
   - Match using Cosine similarity

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

   Where: V is the size of the vocabulary.
   - **Instead** of using the binary occurrence, we can use these weights:
     - **Term frequency**:
       * Raw term frequency: $tf(t,d)$
       * Long term frequency: $\begin{cases} 1 + \log tf(t,d) & \text{if } tf(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$

- **TF-IDF**:
  - Inverse document frequency (1): $idf(t) = \log \frac{N}{df(t)}$
  - Inverse document frequency (2): $\max\left\{0, \log \frac{N - df(t)}{df(t)}\right\}$

    Where:

$$
\begin{aligned}
df(t) &= \text{document frequency of term t} \\
N &= \text{total number of documents in a collection}
\end{aligned}
$$

  - **TF-IDF**:

$$\text{TF-IDF}(t, d) = \text{tf}(t, d) \cdot \text{idf}(t)$$

2. **Language modelling (LM) in IR**:
   - **Method**

     Statistical language model is a probability distr. over sequences of words.
     - given a **sequence** of **length m**
     - a LM assigns probability $P(w_1, \cdots, w_m)$ to this sequence
     - Unigram LM: $P(w_1, \cdots, w_m) = P(w_1) \cdots P(w_m)$
     - Bi-gram LM: $P(w_1, \ldots, w_m) = P(w_1) P(w_2 \mid w_1) \ldots P(w_m \mid w_{m-1})$
     - Documents as distributions (1):
       - **Unigram LM**: $P(t \mid M_d) = \frac{tf(t,d)}{dl(d)}$
       - $dl(d)$ is total number of terms in the document (length of document)
       - this is the maximum likelihood estimation
       - a document is a multinomial distribution over words
       - if some vocabulary terms do not in document d, then $P(t \mid M_d) = 0$
       - addressed by smoothing
     - How to match the distributions (using query likelihood model QLM) (2)?
       - **Likelihood** of a doc given a query: $P(d \mid q) = \frac{P(q|d)P(d)}{P(q)}$
       - **Prior distr.** over queries P(q) does
         not affect matching a particular query: $P(d \mid q) \overset{\text{rank}}{=} P(q \mid d)P(d)$
       - Usually, the **prior distr** over docs P(d)
         is assumed to be uniform: $P(d \mid q) \overset{rank}{=} P(q \mid d) = P(q \mid M_d)$
       - "Bag of words" assumption:
         terms are independent: $P(q \mid M_d) = \prod_{t \in q} P(t \mid M_d) = \prod_{t \in q} \frac{tf(t,d)}{dl(d)}$
       - Match using **KL-divergence**

$$KL(M_d \| M_q) = \sum_{t \in V} P(t \mid M_q) \log \frac{P(t \mid M_q)}{P(t \mid M_d)}$$

- **Smoothing** (3)
  - **Jelinek-Mercer smoothing**

$$\begin{aligned} P_s(t \mid M_d) &= \lambda P(t \mid M_d) + (1-\lambda) P(t \mid M_c) \\ &= \lambda \frac{tf(t,d)}{dl(d)} + (1-\lambda)\frac{cf(t)}{cl} \end{aligned}$$

Where: $cf(t)$ = collection frequency of term t
cl = collection length

  - **Dirichlet smoothing**
    * A **unigram language model** can be seen as a multinomial distr. over words $\mathcal{L}_d(n_1, \ldots, n_k \mid p_1, \ldots, p_k)$:
      - $n_i = tf(t_i, d)$
      - $p_i = P(t_i \mid M_d)$
    * The **conjugate prior** for **multinomial** is the Dirichlet distr. $P_{\text{prior}}\left(p_1, \ldots, p_k; \alpha_1^{pr}, \ldots, \alpha_k^{pr}\right)$:
      - $\alpha_i^{pr} = \mu P(t_i \mid M_c)$
      - μ is a smoothing parameter $\left(\lambda = \frac{dl}{dl+\mu}\right)$
    * The **posterior** is the Dirichlet distr. with parameters

$$\alpha_i^{po} = n_i + \alpha_i^{pr} = \text{tf}(t_i, d) + \mu P(t_i \mid M_c)$$

    * **Dirichlet smoothing**:

$$P_s(t \mid M_d) = \frac{tf(t_i, d) + \mu P(t_i \mid M_c)}{dl(d) + \mu}$$

3. **BM25**

$$BM25 = \sum_{t \in q} \log\left[\frac{N}{df(t)}\right] \cdot \frac{(k_1+1) \cdot tf(t,d)}{k_1 \cdot \left[(1-b) + b \cdot \frac{dl(d)}{dl_{avg}}\right] + tf(t,d)}$$

- k_1, b - parameters
- $dl(d)$ - length doc d
- dl_{avg} – avg. doc length

- $k_1 = 0 \rightarrow$ summation of idfs
- $k_1 = \infty \rightarrow$ summation of tf-idfs
- $b = 0 \rightarrow$ no normalization by doc length
- $b = 1 \rightarrow$ doc length full effect
- $tf \rightarrow$ compensates (if it's too large for example)

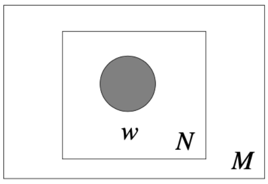
BM25 for long queries

$$BM25 = \sum_{t \in q} \log\left[\frac{N}{df(t)}\right] \cdot \frac{(k_1+1) \cdot tf(t,d)}{k_1 \cdot \left[(1-b) + b \cdot \frac{dl(d)}{dl_{avg}}\right] + tf(t,d)} \cdot \frac{(k_3+1)\, tf(t,q)}{k_3 + tf(t,q)}$$

3.4 Semantic-based Retrieval

1. **Topic modelling**

Unigram language model

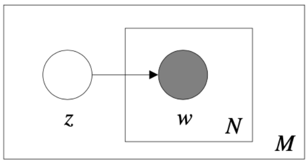


$$W_{ij} \sim \text{Mult}(d_i)$$
$$i \in \{1, \cdots, M\}$$
$$j \in \{1, \cdots, N_i\}$$

- The circle is a random variable.
- Shaded circle is observed, empty is not observed.
- RV is occurrence of word

So: we have a collection of M documents, each document has a length of N. On every nth position the word w may either occur or not.

Mixture of unigrams

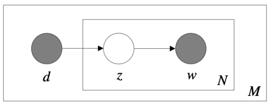


$$z_i \sim \text{Mult}(\theta)$$
$$w_{ij} \sim \text{Mult}\left(\phi_{z_i}\right)$$

“Arts”	“Budgets”	“Children”
NEW	MILLION	CHILDREN
FILM	TAX	WOMEN
SHOW	PROGRAM	PEOPLE
MUSIC	BUDGET	CHILD
MOVIE	BILLION	YEARS

This time: we first added an unobserved hidden variable z (topic). Now we pick a topic M times. From that topic we sample N times. Topics are for example as depicted in the right figure above: ”arts”, ”budgets” and ”children”.

Probabilistic latent semantic analysis (pLSA)



$$z_{ij} \sim \text{Mult}(\theta_i)$$
$$w_{ij} \sim \text{Mult}\left(\phi_{z_{ij}}\right)$$

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

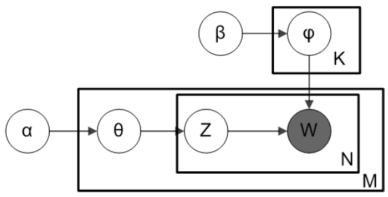
In this case: we assume that every word in the document comes from a certain topic. The document is observed and we have M documents. For every position in the document (that is observed) we randomly sample a topic from the multinomal distr. (which is not known). From that topic we sample a word.

So the **probability of a word given a document**:

$$P(w \mid d) = \sum_{z} P\left(w \mid \phi_z\right) P\left(z \mid \theta_d\right)$$

This is **better** than before, because we are semantically matching. Before we were matching exactly (terms in a query and terms in a document). If the query term did not occur in the document, we used smoothing. **Now**, even if the query term does not occur in the document, but the document is in general about the topic of the query, we still want this document to be ranked **high**.

Latent Dirichlet allocation (LDA)



a) Choose $\theta_i \sim \operatorname{Dir}(\alpha)$, where $i \in \{1, \ldots, M\}$

b) Choose $\phi_k \sim \operatorname{Dir}(\beta)$, where $k \in \{1, \ldots, K\}$

c) For each position j, where $j \in \{1, \ldots, N_i\}$

  i. Choose a topic $z_{ij} \sim \operatorname{Mult}(\theta_i)$

  ii. Choose a word $w_{ij} \sim \operatorname{Mult}(\phi_{z_{ij}})$

This one is basically the same as pLSA, but with priors added.

Estimating LDA: expectation-maximization [**BEYOND IR1!!**]

- **E-step**: define the expected value of the log-likelihood function, with respect to the current estimates of the parameters $\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)}$:

$$Q\left(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)}\right) = \mathrm{E}_{\mathrm{Z}|\mathrm{W}, \boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)}}[\log L(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathrm{W}, \mathrm{Z})]$$

- **M-step**: find the parameters that maximize this quantity

$$\boldsymbol{\theta}^{(t+1)}, \boldsymbol{\phi}^{(t+1)} = \underset{\boldsymbol{\theta}, \boldsymbol{\phi}}{\arg\max} Q\left(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)}\right)$$

- Repeat until convergence

2. **Latent semantic indexing/analysis**

- **Singular Value Decomposition (SVD)**
  - $C = U\Sigma V^T$: $m \times n$ (term document)
  - U: $m \times m$ unitary matrix
  - Σ: diag. $m \times n$ with singular values
  - V^T: $n \times n$ unitary matrix
- **Low-rank approximation**

$$C = U\Sigma V^T = \sum_{i=1}^{\min(m,n)} \sigma_i \vec{u}_i \vec{v}_i^T \approx \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T = U_k \Sigma_k V_k^T$$

- **Latent semantic indexing/analysis**

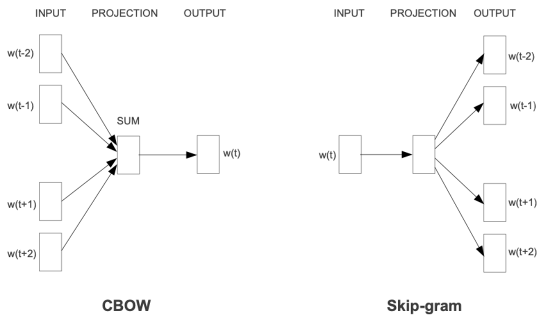
$$\overset{\substack{C \\ (\mathbf{d}_j) \\ \downarrow}}{(\mathbf{t}_i^T) \rightarrow \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix}} = \overset{U_k}{(\hat{\mathbf{t}}_i^T) \rightarrow \begin{bmatrix} \begin{bmatrix} \mathbf{u}_1 \end{bmatrix} & \dots & \begin{bmatrix} \mathbf{u}_k \end{bmatrix} \end{bmatrix}} \cdot \overset{\Sigma_k}{\begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_k \end{bmatrix}} \cdot \overset{\substack{V_k^T \\ (\hat{\mathbf{d}}_j) \\ \downarrow}}{\begin{bmatrix} [\ \mathbf{v}_1\ ] \\ \vdots \\ [\ \mathbf{v}_k\ ] \end{bmatrix}}$$

$$d_j = U_k \Sigma_k \hat{d}_j \Longrightarrow \hat{d}_j = \Sigma_k^{-1} U_k^T d_j$$

- **Documents as vectors**:
  - Given a collection of documents, perform SVD and low-rank approximation to obtain Σ_k and U_k
  - Given a document and a query, represent them as a vectors in the obtained "semantic" vector space $\hat{d} = \Sigma_k^{-1} U_k^T d$ $\quad \hat{q} = \Sigma_k^{-1} U_k^T q$
  - Match the obtained "semantic" vector representations $\hat{d}$ and $\hat{q}$ using cosine similarity

3. **Neural models**
   - **Word embeddings**

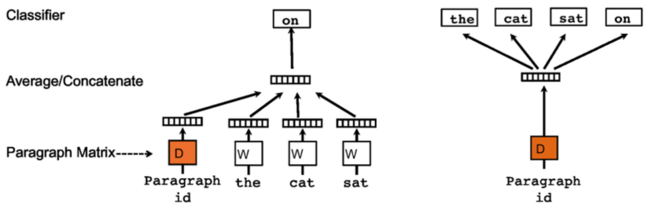
Word2Vec



Documents as vectors

- Compute word embeddings
- Given a document and a query, compute their vector representations as average word embeddings (AWEs)
- Match using cosine similarity

   - **Document embeddings**
     - **Paragraph vector**



Paragraph vector

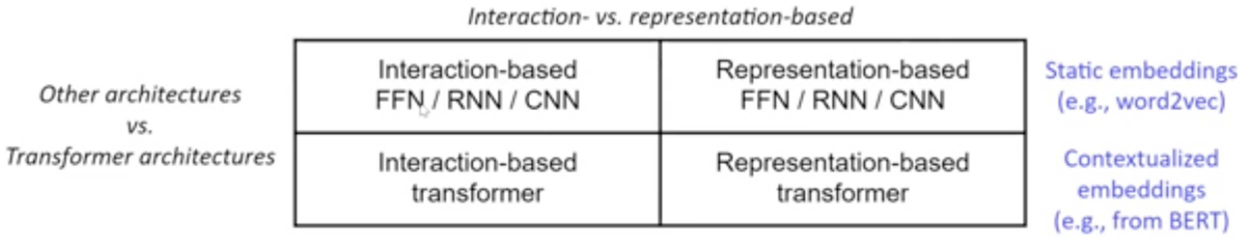
- At every step of stochastic gradient descent, sample a fixed-length context from a random paragraph
- Compute the error gradient from the network
- Use the gradient to update parameters

Documents as vectors

- Compute document embeddings
- Given a query
  a) Fix the word matrix W
  b) Add a (random) column to the document matrix D corresponding to the query repr.
  c) Update D using gradient descent
  d) Get the vector repr. of the query from the updated matrix D
- Match using cosine similarity

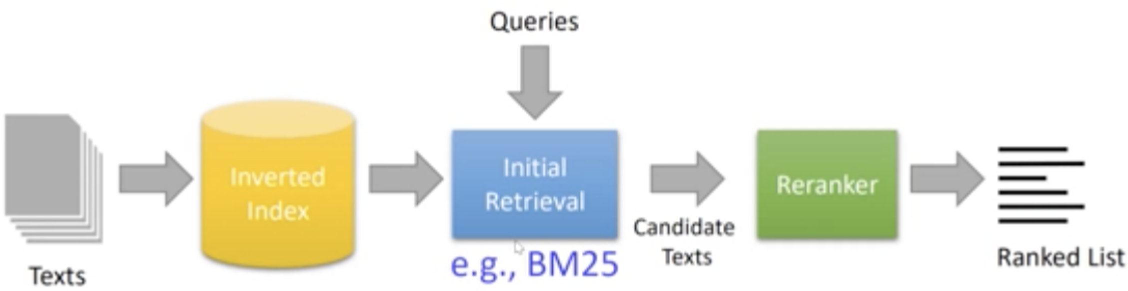
3.5 Semantic retrieval: Neural Models for ranking

Taxonomy of approaches:

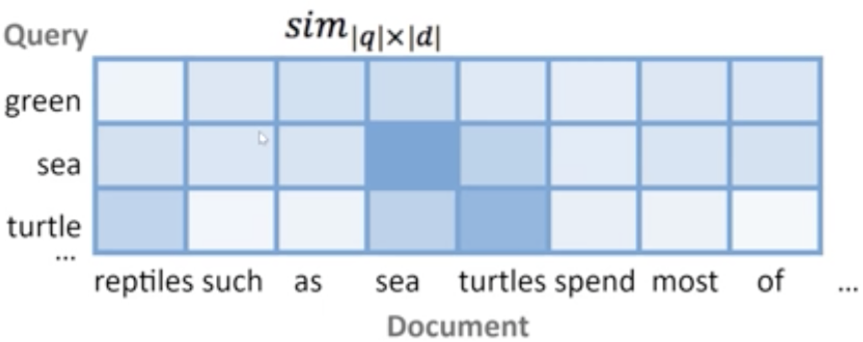


Soft matching enabled by **dense representations** (of terms, phrases, documents, ...).

Reranking:



Many Interaction based approaches consume a *similarity matrix* to predict a document's relevance (query-doc similarity matrix):



State-of-the-art method KNRM:
Idea: kernel pooling to characterize soft matches.

1. Create similarity matrix ("translation matrix")
2. Apply Gaussian kernels to each row (query term)

$$K_k(M_i) = \sum_j \exp(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2})$$

   Where:
   M_{ij} is the similarity score.
   μ is hyperparameter (place where histogram is centered at).
   σ is hyperparameter (how much can be included in that histogram).

3. Sum kernel scores across query terms

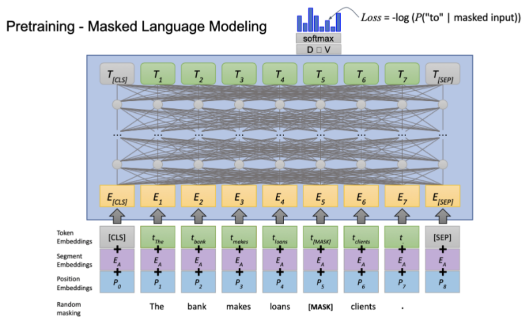
$$\vec{K}(M_i) = \{K_1(M_i), \ldots, K_K(M_i)\} \quad \phi(M) = \sum_{i=1}^{n} \log \vec{K}(M_i)$$

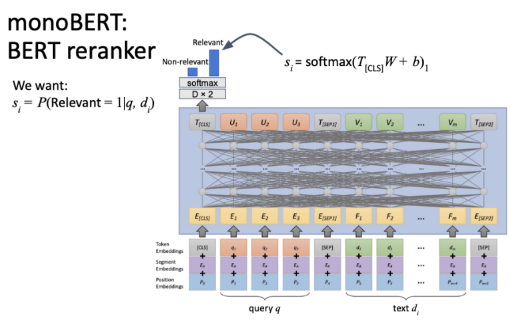
4. Compute document with FFN

$$f(q, d) = \tanh(w^t \phi(M) + b)$$

   Note: embeddings are trained along with model.

Transformer (interaction) based approaches:
BERT's pretraining ingredients:
Transformer (encoder only) with lots of parameters + lots of text + lots of computation.



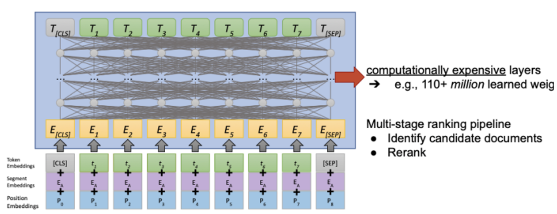


Loss: $L = -\sum_{j \in J_{\text{pos}}} \log(s_j) - \sum_{j \in J_{\text{neg}}} \log(1 - s_j)$

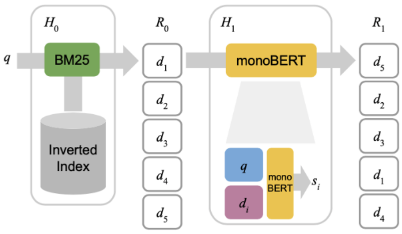
Judgments

Judgments, BM25

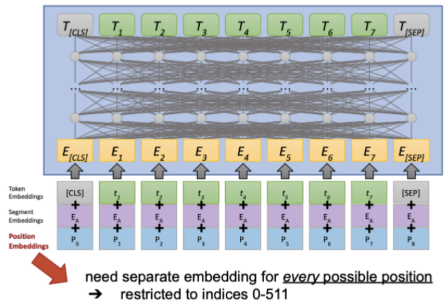
BERT's Limitations



BERT's Limitations: Reranking Pipeline



BERT's Limitations

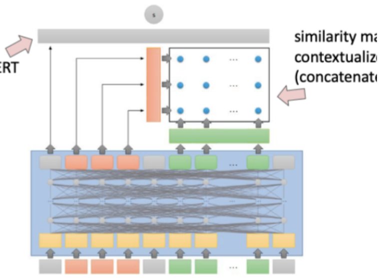


Cannot input entire documents

- what do we input?
- & how do we label it?

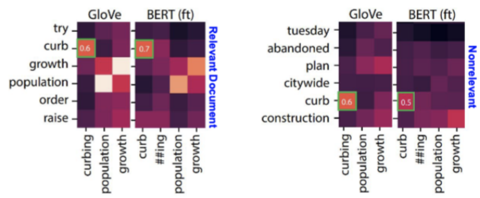
CEDR: Leveraging Contextualized Embeddings

interaction-based pre-BERT model (e.g., KNRM)



similarity ma
contextualiz
(concatenat

CEDR: Leveraging Contextualized Embeddings



Transformer (representation) based approaches:

We do not need to match exactly anymore.

Sparse Representations

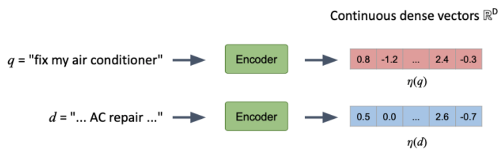
Estimate the relevance of text d to a query q:

q = "fix my air conditioner"
d = "... AC repair ..."

$$\mathrm{BM25}(q,d) = \sum_{\boxed{t \in q \cap d}} \log \frac{N - \mathrm{df}(t) + 0.5}{\mathrm{df}(t) + 0.5} \cdot \frac{\mathrm{tf}(t,d) \cdot (k_1 + 1)}{\mathrm{tf}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{l_d}{L}\right)}$$

Terms need to match exactly

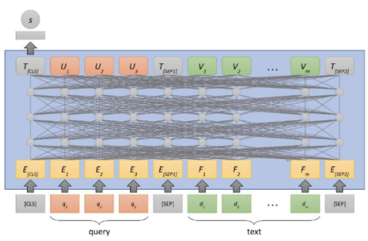
Document-level Dense Representations



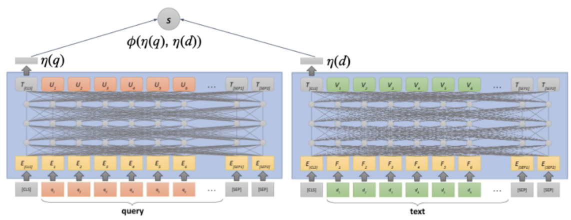
ϕ is a similarity function (e.g., inner product or cosine similarity)

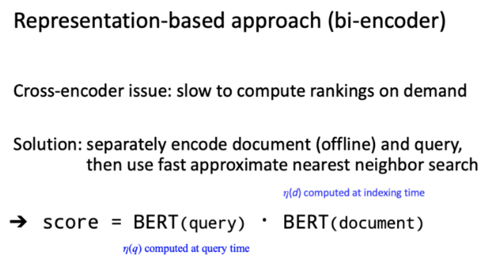
$\phi(\eta(q), \eta(d)) \rightarrow$ measures how relevant d is to q

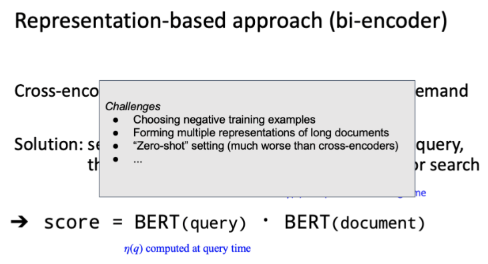
Interaction-based approach (cross-encoder)

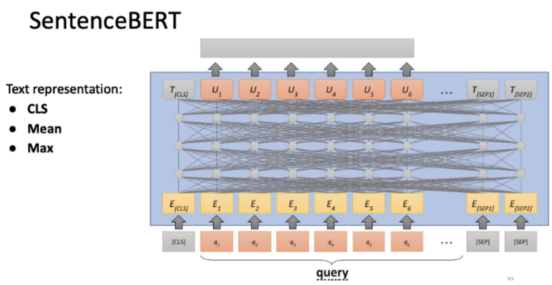


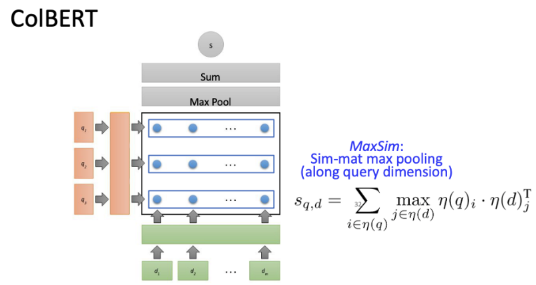
Representation-based approach (bi-encoder)











4 Week 4: Learning to Rank (LTR) and Interactions

4.1 Preliminaries and Goal

Representation:
Represent the document and query in a format that a ML model can use: a numerical vector $\vec{x} \in \mathbb{R}^n$.

Prediction:
Then a ranking model $f : \vec{x} \to \mathbb{R}$ is optimized to score each document-query combination so that relevant documents are scored higher.
In mathematical terms: f maps a vector to a real-valued score.

Features:
Traditionally features are hand-crafted to encode IR insights, nowadays we also have *deep learned* features.

They can be categorized as:

- **Document-only** or static features (e.g., document length)
- **Document-Query-combination** or dynamic features (e.g., BM25)
- **Query-only** features (e.g., query length)

Models can be trained on different data:

- **Offline or Supervised** LTR: learn from annotated data.
  - Expensive and time consuming.
  - Provides ground truth
- **Online/Counterfactual** LTR: learn from user interactions.
  - Virtually free and easy to obtain. Hard to interpret.

4.2 Offline LTR

Data is obtained by:

1. Pay some humans to be annotators (and train them to be good annotators).
2. Collect a set of queries
3. Preselect a large (not too large) set of documents per query.
4. Show document-query pairs to annotators.
5. Annotators rate every document-query pair on their relevance (e.g. on a scale from 0 to 4).

Goal:
We have:

- Feature representation of document-query pairs: $\vec{x}_{q,d} \in \mathbb{R}$.
- Labels indicating the relevance of document-query pairs: $y_{q,d} \in [0, 4]$

And we want:

- A function $f : \vec{x} \to \mathbb{R}$ that scores documents.
- To get the best ranking by sorting according to $f(\vec{x})$.

How to find f?

4.3 Pointwise approach

Regression-based or classification-based approaches are popular.

Regression loss:
Given $\langle q, d \rangle$ predict the value of $y_{q,d}$.

E.g., square loss for binary or categorical labels:

$$\mathcal{L}_{\text{Squared}}\ (q, d, y_{q,d}) = \|y_{q,d} - f\left(\vec{x}_{q,d}\right)\|^2$$

where $y_{q,d}$ is the one-hot representation or the actual value of the label.

Classification loss:
Given $\langle q, d \rangle$ predict the class $y_{q,d}$.

E.g., Cross-Entropy with Softmax over categorical labels Y:

$$\mathcal{L}_{\text{CE}}\left(q, d, y_{q,d}\right) = -\log\left(p\left(y_{q,d} \mid q, d\right)\right) = -\log\left(\frac{e^{\sigma \cdot s_{y_{q,d}}}}{\sum_{y \in Y} e^{\sigma \cdot s_y}}\right)$$

where $s_{y_{q,d}}$ is the model's score for label $y_{q,d}$.

Issues with **pointwise approaches**:

- **Class imbalance**: many irrelevant documents and very few relevant documents.
- **Query level feature normalization needed**: distr. of features differs greatly per query

But, these can be overcome.

The fundamentally wrong part is:
Ranking is not a regression or classification problem.

A document-level loss does not work for raking problems because document scores should not be considered independently (pointwise methods do not directly optimize ranking quality).

4.4 Pairwise approach

Instead of looking at document-level, consider pairs of documents.

$$P\left(d_i \succ d_j\right) = f\left(\vec{x}_i, \vec{x}_j\right)$$

Do not change the model to take document pairs as input (would be quadratic in complexity: $O\left(N^2\right)$, during inference).

The scoring model remains unchanged: $f(\vec{x_i}) = s_i$, but the loss function is based on document pairs:

$$\mathcal{L}_{\text{pairwise}} = \sum_{d_i \succ d_j} \phi\left(s_i - s_j\right)$$

Thus we still score documents and then order according to scores.

Pairwise loss functions:
Pairwise loss minimizes the average number of inversions in ranking:
$d_i \succ_q d_j$ but d_j is ranked higher than d_i

Generally the following form:

$$\mathcal{L}_{\text{pairwise}} = \phi\left(s_i - s_j\right)$$

where ϕ can be:

- **Hinge function**: $\phi(z) = \max(0, 1 - z)$
- **Exponential function**: $\phi(z) = e^{-z}$
- **Logistic function**: $\phi(z) = \log(1 + e^{-z}$
- etc.

RankNet (using σ instead of γ, following assignment notation):

is a pairwise loss function - popular choice for training neural LTR models.

For a given query, each pair of documents D_i and D_j with differing labels is chosen, and each such pair (with feature vectors x_i and x_j) is presented to the model, which computes the scores $s_i = f\left(x_i\right)$ and $s_j = f\left(x_j\right)$. Let $D_i \triangleright D_j$ denote the event that D_i should be ranked higher than D_j. The two outputs of the model are mapped to a learned probability that D_i should be ranked higher than D_j via a sigmoid function:

Predicted probabilities:

$$P_{ij} = P(D_i \triangleright D_j) \equiv \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$
$$P_{ji} = P(D_i \triangleleft D_j) \equiv \frac{1}{1 + e^{-\sigma(s_j - s_i)}}$$

Desired probabilities:

$$\bar{P}_{ij} = 1$$
$$\bar{P}_{ji} = 0$$

Computing **cross-entropy** between $\bar{P}$ and P:

$$\begin{aligned}
\mathcal{L}_{\text{RankNet}} &= -\bar{P}_{ij} \log\left(P_{ij}\right) - \bar{P}_{ji} \log\left(P_{ji}\right) \\
&= -\log\left(P_{ij}\right) \\
&= \log\left(1 + e^{-\sigma(s_i - s_j)}\right)
\end{aligned}$$

Factorization RankNet: let $S_{ij} \in \{-1, 0, 1\}$ indicate the preference between d_i and d_j.

Predicted probabilities:

$$\bar{P}\left(d_i \succ d_j\right) = \frac{1}{2}\left(1 + S_{ij}\right)$$

Desired probabilities:

$$P\left(d_i \succ d_j\right) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$

The cross-entropy loss is then:

$$\mathcal{L}_{ij} = \frac{1}{2}\left(1 - S_{ij}\right)\sigma\left(s_i - s_j\right) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right)$$

We can also consider a **sped-up version** of the RankNet:
First we need the derivative w.r.t. s_i:

$$\frac{\delta \mathcal{L}_{ij}}{\delta s_i} = \sigma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \right) = -\frac{\delta \mathcal{L}_{ij}}{\delta s_j}$$

We can further factorize this loss so that:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \frac{\delta \mathcal{L}_{ij}}{\delta s_i}\frac{\delta s_i}{\delta w} + \frac{\delta \mathcal{L}_{ij}}{\delta s_j}\frac{\delta s_j}{\delta w} = \sigma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \right)\left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w} \right)$$

The factorized **cross entropy loss**:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \sigma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \right)\left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w} \right)$$

We choose λ so that:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \lambda_{ij}\left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w} \right)$$

where:

$$\lambda_{ij} = \sigma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \right)$$

These lambdas act like **forces** pushing pairs of documents apart or together.

On document level the same can be done (will copy-paste directly from the paper, since the slides didn't help me):

Let I denote the set of pairs of indices $\{i, j\}$, for which we desire D_i to be ranked differently from D_j (for a given query). I must include each pair just once, so it is convenient to adopt the convention that I contains pairs of indices $\{i, j\}$ for which $D_i \triangleright D_j$, so that $S_{ij} = 1$ (which simplifies the notation considerably, and we will assume this from now on). Note that since RankNet learns from probabilities and outputs probabilities, it does not require that the documents are labeled; it just needs the set I, which could also be determined by gathering pairwise preferences.

Now we introduce the λ_i (one λ_i for each document: note that the λ 's with one subscript are sums of the λ 's with two). To compute λ_i (for document D_i), we find all j for which $\{i, j\} \in I$ and all k for which $\{k, i\} \in I$. For the former, we increment λ_i by λ_{ij}, and for the latter, we decrement λ_i by λ_{ki}. For example, if there were just one pair with $D_1 \triangleright D_2$, then $I = \{\{1, 2\}\}$, and $\lambda_1 = \lambda_{12} = -\lambda_2$. In general, we have:

$$\lambda_i = \sum_{j:\{i,j\}\in I} \lambda_{ij} - \sum_{j:\{j,i\}\in I} \lambda_{ij}$$

Issues with **pointwise approaches**:

- **RankNet based on virtual probabilities**: $P(d_i \succ d_j)$
  In reality the ranking model does not follow these probabilites.

But, not a big deal.

The fundamentally wrong part is:
Not every document pair is equally important.

It is Much more important to get the correct ordering of top documents than of the bottom documents (top 5 more important than order of documents after position 10).

4.5 Listwise approach

The **fundamental problem** with the approaches so far is that they did **not optimize ranking quality directly**.
A **LTR method** should **directly optimize the ranking metric** we care about (from simple to more complex):

- Simple:

$$\text{precision}(R) = \frac{1}{|R|} \sum_{R_i} \text{relevance } (R_i)$$

- Complex (e.g., discounted cumulative gain):

$$DCG(R) = \sum_{R_i} \frac{2^{\text{relevance}(R_i)} - 1}{\log(i+1)}$$

These metrics are non-continuous and non-differentiable.

Due to strong position-based discounting in IR measures, **errors at higher ranks** are much more problematic than at lower ranks.

LambdaRank:

Multiply actual gradients with the change in NDCG by swapping the rank positions of the two documents:

$$\lambda_{LambdaRank} = \lambda_{RankNet} \cdot |\Delta\text{NDCG}|$$

Works also with other metrics, e.g. $|\ \Delta$ Precision $|$.

Empirically, LambdaRank was shown to directly optimize IR metrics. Theoretically was shown, that LambdaRank optimizes a lower bound on certain IR metrics.

4.6 ListNet and ListMLE

Create a probabilistic model for ranking, which is differentiable.

Sample documents from a Plackett-Luce distribution:

$$P\left(d_i\right)=\frac{\phi\left(s_i\right)}{\sum_{d_j \in D} \phi\left(s_j\right)}$$

For instance, $\phi(s_i)=e^{s_i}$:

$$P(d_i)=\frac{e^{s_i}}{\sum_{d_j \in D} e^{s_j}}$$

According to the Luce model, given four items $\{d_1, d_2, d_3, d_4\}$ the probability of observing a particular rank-order, say $[d_2, d_1, d_4, d_3]$, is given by:

$$P(\pi \mid s)=\frac{\phi\left(s_2\right)}{\phi\left(s_1\right)+\phi\left(s_2\right)+\phi\left(s_3\right)+\phi\left(s_4\right)} \cdot \frac{\phi\left(s_1\right)}{\phi\left(s_1\right)+\phi\left(s_3\right)+\phi\left(s_4\right)} \cdot \frac{\phi\left(s_4\right)}{\phi\left(s_3\right)+\phi\left(s_4\right)}$$

where π is a particular permutation and ϕ is a transformation (e.g., linear, exponential, or sigmoid) over the score s_i corresponding to item d_i.

ListNet

Compute the probability distribution over all possible permutations based on model score and ground-truth labels. The loss is then given by the KL-divergence between these two distributions. This is computationally very costly, computing permutations of only the top-K items makes it slightly less prohibitive.

ListMLE

Compute the probability of the ideal permutation based on the ground truth. However, with categorical labels more than one permutation is possible which makes this difficult.

Recap learning to rank:

Ranking is very important in places were search or recommendation is involved. Methods should scale to large collections and work fast enough to help users. Search engines use large numbers of signals/features.

Pointwise approach:

- Predict the relevance per item, simple but very naive.
- Ignores that ordering of items is what matters.

Pairwise approach:

- Loss based on document pairs, minimize the number of incorrect inversions.
- Ignores that not all document pairs have the same impact.
- Often used.

Listwise approach:

- Tries to optimize for IR metrics, but they are not differentiable.
- Approximations by heuristics, bounding or probabilistic approaches to ranking.
- Best approach out of three.

4.7 User interactions

Why are user interactions important?

- Evaluate IR systems
- Improve IR systems

Models of user search interactions:

- Click models
- Models of mouse hovering
- Models of time between user actions

Outline:

1). **Basic click models**

1a). **Position based model**:

Suppose we have the search results above and we observe the 2 clicks. How do we **model** this? **First**, we randomly choose a result out of the 5, let's say we choose the 3rd one. We assume that the user first **examines** the **snippet**. This is called the **probability of examination**: $P_{exam}(3)$. This depends on the *position*.

Secondly, if we like and think it is a good result for the **query**, we make another decision. Namely, we decide whether we are **attracted** or not (by the snippet given our query). This is called **the probability of attractiveness**: $P_{attr}(qd_3)$ (does not depend on examination).

From lecture: So see it as tossing 2 types of coins. First, you toss a coin from the **set of examination coins**, thereafter you toss a coin from the **set of attractive coins**. So you **click only** if you **examine and find it attractive**. Same applies for all 5 results $\{P_{exam}(1), P_{attr}(qd_1) \dots P_{exam}(5), P_{attr}(qd_5)\}$.

Position-based model: examination:

Terminology:

- **Examination** = reading a **snippet**
- E_r - binary random variable denoting examination of a snippet at rank r

Position-based model (PBM):

- Examination depends on rank: $P(E_r = 1) = \gamma_r$

So all probabilities of examination will be replaced with γ, as following for example 5 results: $\{\gamma_1, P_{attr}(qd_1) \ldots \gamma_5, P_{attr}(qd_5)\}$. If the number of results is 100, then we would get 100 γ's: $\{\gamma_1, \ldots, \gamma_{100}\}$.

Position-based model: attractiveness:

Terminology:

- **Attractiveness** = a user wants to click on a document after examining it's **snippet**
- A_{qd} - binary random variable showing whethet document d is attractive to a user, given query q

Position-based model (PBM):

- Attractive depends on a query-document pair: $P(A_{qd} = 1) = \alpha_{qd}$

So all probabilities of attractiveness will be replaced with α, as following for example 5 documents: $\{\gamma_1, \alpha_{qd_1} \ldots \gamma_5, \alpha_{qd_5}\}$. If the number of documents is 100, then we would get 100 α's: $\{\alpha_{qd_1}, \ldots, \alpha_{qd_{100}}\}$.

Position-based model: Summary

- Probability of examination: $P(E_{r_d} = 1) = \gamma_d$
- Probability of attractiveness: $P(A_{qd} = 1) = \alpha_{qd}$

- Probability of **click**: $P(C_d = 1) = P(E_{r_d} = 1) \cdot P(A_{qd} = 1) = \gamma_d \cdot \alpha_{qd}$

1b). **Cascade model**:

1. Start from first document
2. Examine documents one by one
3. If click, then stop
4. Otherwise, continue

Again we click iff we examine and find it attractive: $E_r = 1$ and $A_{d_r} = 1 \Leftrightarrow C_r = 1$.
The probability of attractiveness stays the same, but there are some changes (see below):

$$P(A_{d_r} = 1) = \alpha_{qd_r}$$
$$\underbrace{P(E_1 = 1)}_{\text{start from first}} = 1$$
$$\underbrace{P(E_r = 1 \mid E_{r-1} = 0)}_{\text{examine one by one}} = 0$$
$$\underbrace{P(E_r = 1 \mid C_{r-1} = 1)}_{\text{if click, then stop}} = 0$$
$$\underbrace{P(E_r = 1 \mid E_{r-1} = 1, C_{r-1} = 0)}_{\text{otherwise, continue}} = 1$$

Basic click models summary:

- Position-based model (PBM):
  + examination and attractiveness
  – examination of a document at rank r does not depend on examinations and clicks above r
- Cascade model (CM):
  + cascade dependency of examination at r on examinations and clicks above r
  – only one click is allowed

2). **Estimation**

Parameter estimation:

- Maximum likelihood estimation
- Expectation-maximization
  1. Set parameters to some initial values
  2. Repeat until convergence
     - **E-step**: derive the expectation of the likelihood function
     - **M-step**: maximize this expectation

EM update rules for PBM: attractiveness

$$\alpha_{qd}^{(t+1)} = \frac{1}{|S_{qd}|} \sum_{s \in S_{qd}} \left(c_d^{(s)} + \left(1 - c_d^{(s)}\right) \frac{\left(1-\gamma_r^{(t)}\right)\alpha_{qd}^{(t)}}{1-\gamma_r^{(t)}\alpha_{qd}^{(t)}} \right)$$

t: iteration
S_{qd}: search sessions iniated by query q and containing doc u
$c_d^{(s)}$: observed click on doc u in search sessions s

EM update rules for PBM: examination

$$\gamma_r^{(t+1)} = \frac{1}{|S|} \sum_{s \in S} \left(c_d^{(s)} + \left(1 - c_d^{(s)}\right)_k \frac{\gamma_r^{(t)}\left(1-\alpha_{qd}^{(t)}\right)}{1-\gamma_r^{(t)}\alpha_{qd}^{(t)}} \right)$$

3). Applications

What can we get after estimation of a click model?

Full probability - probability that a user clicks on a document at rank r: $P(C_r = 1)$

Conditional probability - probability that a user clicks on a document at rank r given previous clicks $P(C_r = 1 \mid C_1, \ldots, C_{r-1})$

Click model's output	Application
Full click probabilities	Model-based metrics
Conditional click probabilities	User simulation
Parameter values	Ranking

Model-based metrics:

Utitility-based metric:

$$\text{uMetric } = \sum_{r=1}^{n} P\left(C_r = 1\right) \cdot U_r$$

Effort-based metric:

$$\text{eMetric } = \sum_{r=1}^{n} P\left(S_r = 1\right) \cdot F_r$$

Dynamic Bayesian network model (DBN)

$$\begin{aligned} P\left(A_r = 1\right) &= \alpha_{qd_r} \\ P\left(E_1 = 1\right) &= 1 \\ P\left(E_r = 1 \mid S_{r-1} = 1\right) &= 0 \\ P\left(E_r = 1 \mid S_{r-1} = 0\right) &= \gamma \\ P\left(S_r = 1 \mid C_r = 0\right) &= 0 \\ P\left(S_r = 1 \mid C_r = 1\right) &= \sigma_{qd_r} \\ P\left(S_r = 1\right) &= ? \end{aligned}$$

Expected reciprocal rank (ERR, second term last part from DBN):

$$\begin{aligned} ERR &= \sum_r \frac{1}{r} \cdot P(S_r = 1) \\ &= \sum_r \frac{1}{r} \cdot R_{qd_r} \cdot \prod_{i=1}^{r-1} (\gamma \cdot (1 - R_{qd_i})) \end{aligned}$$

Dynamic Bayesian network model (DBN)

$$\begin{aligned} P\left(S_r = 1\right) &= P\left(S_r = 1 \mid C_r = 1\right) \cdot P\left(C_r = 1\right) \\ &= \sigma_{qd_r} \cdot P\left(C_r = 1\right) \\ &= \sigma_{qd_r} \cdot \alpha_{qd_r} \cdot P\left(E_r = 1\right) \\ &= \sigma_{qd_r} \cdot \alpha_{qd_r} \cdot \prod_{i=1}^{r-1} \left(\gamma \cdot \left(1 - \sigma_{qd_i} \cdot \alpha_{qd_i}\right)\right) \\ &= R_{qd_r} \cdot \prod_{i=1}^{r-1} \left(\gamma \cdot \left(1 - R_{qd_i}\right)\right) \quad \textbf{(ERR)}. \end{aligned}$$