Contents

1 Foundations

1.1 Basics

- User has an *Information Need* that we are tying to satisfy
- Want to identify *relevant documents*.

Challenges

- How to decide if a document is relevant?
- How to decide *how* relevant a document is?
- Finding relevant documents in a large corpus?

Basic Approach • Go through documents and save statistics on documents (e.g. *term frequency, document frequency*). • Use these to determine relevance to a query. • Can do this at *index time* instead of at query time.

Basic Definitions

- *Document* – An enclosed "package" of information, .e.g. a web page, a text file, an article, ...
- *Collection* – A set of documents (assumed to be static)
- *Relevance* – Does the document satisfy the users information need?
- *Token* – Part of text that is considered a single linguistic element (i.e. a word or an abbreviation or a number)
- *Term* – Equivalence class of tokens as defined by e.g. a common stem.

Boolean Retrieval Model We only consider whether a term appears in a document or not. Hence: • save document-term incidence matrix (infeasible) • Use AND, OR, NOT to search for combinations of query terms.

This is not flexible enough and the incidence matrix is infeasible to construnct, hence...

Inverted Index Data structure that holds

- Statistics per each document for each term
  (inverted: Term $\mapsto$ Posting List with frequencies)
  - *Document frequency*: How many document this term is contained in.
  - *Term frequency*: How often this term appears in this document
- Document metadata
  - *Document length*, *average document length*
  - Name, location on disk, full text, ...

Data Structures for inverted index (Dictionary)

- **HashMaps**
  - Fast lookups
  - No easy way to find minor variants
  - No prefix search
  - Potentially need to rehash
  - No sorting or sorted sequential access
- **Trees** (binary tree, B-Tree)
  - Can easily find prefixes (helps with wildcard queries)
  - Slower lookup
- Finite State Transducers

The two most basic parts in an IR engine are:

1. **Indexing** – Inverted Index can be constructed iteratively, document-by-document.
2. **Query Processing** – Retrieve statistics from Inverted Index for all query tokens, then rank documents.

1.2 Indexing

Basic Steps

1. **Collect Documents**, Register document metadata
2. **Tokenization** – equivalence classes can be defined by
   - Stemming
   - Case Folding
   - Removal of Hyphens, Dots, Apostrophes, ...
   - Reduction to semantic symbols
3. **Stop Word Removal**
4. Upsert term into **Term Dictionary**
5. Upsert term into **Posting List** for according document (Have a mapping from Dictionary Entry to Posting List)

Tokenization Split input string in single words / units of linguistic information. **Challenges**:

- Keep Abbreviations, Dates, Numbers as a single token.
- Distinguish Abbreviations from words (e.g. *C.A.T.* vs *cat*)
- Apostrophes (*l'hospital* – one or two tokens?)
- Hyphenations (*Versicherungs-Gesellschaft, Hewlett-Packard*)
- Names (e.g. TU Wien), *Phrases* (i.e. *Queen of England, San Francisco, figure of speech*)
- Compound words
- Mixture of LTR, RTL text
- Umlauts, other language-specific things like delimitation of a semantic token in chinese or japanese.
- Mixture of different character sets
- Handling of synonyms

Stemming Reduce tokens to a common root before indexing (language-specific), e.g. by removing suffixes according to some rules. Tradeoff between precision and cost.

Stop Word Removal

- **Motivation**: • Smaller data volume • stop words dont have much semantic content.
- **Challenges**:
  - Good compression techniques mean there is little actual cost for keeping stop words.
  - Needed for • Phrases • When semantics are important (e.g. *flights to London*)

Combination Schemes Can improve index by taking into account phrases that appear very often together (e.g. *TU Wien*, *United Kingdom*)

Handling Phrase Queries Want to be able to search for *Phrases* like e.g. *TU Wien*. Entered e.g. by putting parts of query string in quotes. Possible approaches:

1. **Biword indexes** – Index by not one token but pairs. Can then query longer phrases by AND on pairs.
   - Can have false positives (potentially returns more than wanted)!
   - Much larger index
2. **Positional Indexes** – • For each term, store the position it appears in the file (token index). • Get doc:pos lists for each term in phrase from dictionary • merge these by document • then find the ones where terms are succeeding (or other, for proximity search)
   - Larger index size

Handling Wildcard Queries Approaches

- For a query *foo*qux*, find results for *foo** and separately for **qux* (*maintain backwards B-Tree*), then intersect results and use positional indices to check. (Need multiple trees)
- *Permuterm Index*: Index each term in seperate entries (e.g. *hello* goes to *helloX*, *elloXh*, *lloXhe*, ...) where X is a special character. • Then rotate query term so that wildcard is on end. $\rightarrow$ larger index size.
- *k-gram indices* Keep mapping from k-grams of characters to dictionary terms that match each k-gram, then *mon** can be queried by *Xm* AND *mo* AND *on*
  - Again, false positives – must filter again against queries
  - Less expensive compared to permuterm approach

Spelling Correction Can apply to indexing or query processing. *Basic assumption:* Have a dictionary with "correct" spellings (could also be just all terms of corpus). *Objective*: Find the word in the dictionary that is "closest" to the entered word. Distance measures:

- **(weighted) edit distance** – given a query, enumerate all strings within a given edit distance of query term, intersect these with correct words from dictionary. Show these as suggestions. Calculating the edit distance to every dictionary term is expensive – use...
- **n-gram overlap** – keep n-gram index, then consider all words with a certain number of overlapping n-grams as alternatives (as score, use *Jaccard coefficient*)
- **Context-sensitive for phrase queries** – Fix one word at a time, suggest the alternative with the most hits.

Soundex reduce tokens into phonetic equivalents.

2 Efficient Text Processing

Ideas:

- Avoid garbage collection (e.g • use array pool • manage memory manually)
- Avoid manipulations on strings (substring, split, ...) – Strings are immutable, so this causes the creation and allocation of new objects.
- Use memory-mapped files.
- Avoid creating objects and work directly with byte buffers instead.
- Use int/long when possible
- Interpret strings (charbuffers) as longs for comparison (e.g. in stop word removal)
- Use multithreading
- Measure!

3 Scoring Documents

3.1 Basics

Disadvantages of Boolean Search (retrieve all documents matching boolean query)

- Often result in too many or too few results
- Writing queries with the intended breadth can be hard
  AND produces high precision but low recall; OR gives low prevision but high recall
- Output is not really ranked
- All terms are equally weighted

Assume *free text queries*, that is: query is not a semantically string combination of operators and expression but just a string.

Bag of Words : Idea that order of terms is not important for ranking documents.

Term Frequency $tf(t,d)$ How often does this term appear in this document? Commonly used in combination with a logarithm: tf-weight $= \log(1 + tf(t,d))$. Logarithm has kind of a dampening effect (extremely large values are not as large).

Document Frequency $df(t)$ How many documents does this term appear in?

Idea: Rare terms are more informative than frequent terms – should have a high score. Hence inverse fraction and use log-dampening: *Inverse document frequency* $idf(t) = \log(\frac{n}{df(t)})$

tf-idf weight

$$tf.idf(t,d) = \log(1 + tf(t,d)) \cdot \log(\frac{n}{df(t)})$$

- $tf(t,d)$ increases with the number of terms in the document
- $idf(t)$ increases with the rarity of the term in the collection

We define the score of a document d w.r.t a query q:

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

Problem: It's a sum: Longer document → Potentially higher term frequency in document → Higher score

3.2 Vector Space Model

Vector Space model

- Each document is represented a vector with a component for each term.
- Can use $tf.idf(t,d)$ as weight for a component.
- Query is also considered a Vector
- Need a distance function
  - Euclidean distance is bad because: It's a sum of differences of components. Thus the most relevant document would be one that is exactly equal to the query. Any longer document would receive a worse score, even if it still only contained query terms.
  - Use *cosine similarity* instead (angle between vectors).

Performance considerations

- Only want to find top k documents, not order entire collection.
- General approach: Identify set of candidates of which most are in top k. Returning the top k of *these* is assumed to suffice.
- Only consider
  - docs containing at least one query term
  - docs containing many query terms
  - high-idf query terms (i.e. rare terms, do not consider e.g. stop words, those appear in many documents)
- Consider static quality score $g(d)$ (authority) – citations, bookmarks, PageRank, ...; use this in computation of score (e.g. linear combination with relevance)
- Order all postings by g (static, common ordering, independent of query). Then, top-scoring documents are likely to appear earlier in postings traversal.
- In time-bound applications, we can stop traversing early.

3.3 Probabilistic Model

We are working with semantically imprecise items, i.e. different terms mean the same thing, different documents are relevant to some degree to a query, etc. Can we try to use mathematical probability to model this uncertainty?

$\rightarrow$ *Probability of relevance to query* $P(rel \mid d, q)$. Rank by decreasing probability of relevance.

3.3.1 Binary Independence Model

Assumptions

- Binary term-document incidence vectors
- Assume independence between terms
- Relevance of document is independent of relevance of other documents
- Assuming that relevant documents are a small fraction of the collection, statistics for nonrelevant documents can be approximated by statistics for the entire collection.

3.3.2 BM25

Maybe check IR book?

3.4 Language Model

Based on the contents of a document, come up with an abstract (generative) model of it. Then, compute the probability that the query was generated by this model.

Take a finite state automaton as a deterministic language model. For simplicity, assume a unigram language model, i.e. a FSA with a single state. Transitions from/to that state generate different words with different probability.

Under term independence, the probability that a query string is generated is then the product of the probabilities of its terms.

$P(q \mid d)$ is the probability that q is generated by d. From Bayes' Rule we have

$$P(d \mid q) = \frac{P(q \mid d)P(d)}{P(q)}$$

where $P(q)$ is independent of document (ignored), $P(d)$ is assumed to be the same for all d. So, we can equivalently rank according to $P(q \mid d)$ (the probability that a query q would be observed as a random sample from the document model of d, called M_d).

Under independence assumption, we have

$$P(q \mid M_d) = \prod P(q_i \mid M_d)$$

which is equivalent to

$$P(q \mid M_d) = \prod_{t \in q} P(t \mid M_d)^{tf(t,q)}$$

(only need to figure out probabilities for occurence of single terms).

Now need to figure out $P(t \mid M_d)$, i.e. the probability that a (single) term t is generated by M_d. Can estimate with $P(t \mid M_d) \approx \frac{tf(t,d)}{|d|}$.

Problem: Need to avoid zeroes in product. Assume a non-occuring term has the non-zero probability equal to occuring by chance in the collection model M_c:

$$P(t \mid M_c) = \frac{cf(t)}{T}$$

where T is total number of tokens in collection. — Then, use this as a linear combination with $P(t \mid M_d)$.

$$P(t \mid d) = \lambda P(t \mid M_d) + (1 - \lambda)P(t \mid M_c)$$

- λ large, more focus on document model: Tends to retrieve documents containing all query words (because value of product must be high).
- λ small, more focus on collection model: More weight if term is popular in collection, independent of document to be ranked.

Summary (*Mixture Model*):

$$P(q \mid d) \approx \prod_{q_i \in q} \lambda P(q_i \mid M_d) + (1 - \lambda)P(q_i \mid M_c)$$

4 Evaluation

Motivation

- Need to find reliable comparison measures for IR systems
- Large amounts of documents
- Human subjectivity involved (judge relevance)

Kinds of evaluation:

- Efficiency (technical concerns, time and space)
- Effectiveness (Quality of results, harder to measure)
- User studies
- *intrinsic* (consider and measure retrieved set) vs. *extrinsic* (evaluate in entire usage-context)

Things to measure (Cleverdon)

- Coverage – How well can system cover my needs?
- Time lag
- Presentation
- User effort required
- Recall
- Precision

Precision How accurate is the result that we received?

$$\text{Precision} = \frac{\text{relevant retrieved docs}}{\text{total } \textit{retrieved} \text{ docs}}$$

Recall How much of the available information did we actually retrieve?

$$\text{Recall} = \frac{\text{relevant retrieved docs}}{\text{total } \textit{relevant} \text{ docs}}$$

→ *F-Measure* combines both into a single measure. Loses expressive power though.

Precision-Recall curve A precision-recall curve shows the relationship between precision (= positive predictive value) and recall (= sensitivity) for every possible cut-off.

With increasing allowed size of retrieved set, plot Precision vs. Recall.

With increasing size, recall will most likely improve and precision will fall.

Construction

1. Define cutoffs of results size
2. For each cutoff, for each query, measure precision and recall

Methods

- Transform into step function by taking as value the maximum of previously found precisions.
- Average points from different queries / rankings

Challenges

- **Graded Relevance** – Doesn't take into account the order of the retrieved documents and that some documents could in fact be more relevant than others
- **Cutoff Value** – Both P and R are related to some result set size, i.e. a cutoff value.

4.1 Handling Cutoff Value

Different approaches:

r-Precision Fix cutoff value at number of relevant documents. Precision at the r-th position where r is the number of relevant documents.

Reciprocal Rank

$$RR = \frac{1}{\text{rank of first relevant document}}$$

Mean Average Precision Average precision at ranks of relevant documents.

$$AP = \frac{\sum P(i) \cdot rel(i)}{R}$$

where R is number of relevant documents, i is the rank, k number of retrieved documents, $P(i)$ precision at rank and $rel(i)$ boolean indicator if i is relevant. — Then take mean of this over all queries.

4.2 Handling graded relevance

Cumulative Gain Let $\text{rel}(d, q)$ be a nonnegative (and nonbinary!) relevance score for i. Then

$$CG = \sum_i rel(i, q)$$

Problem: How to judge relevance?

Discounted Cumulative Gain Gain is accumulated starting at the top, Gain summands diminish with larger index by scaling down with log. Thus, a system that returns more relevant documents at a higher rank receives a higher score.

Problem: Both CG and DCG depend on the number of relevant documents per topic! So, they cannot be used to compare performance across topics! — Can normalise by dividing with *Ideal DCG* (value for optimal return set).

Rank-biased precision Alternative to CG variants.

$$\begin{aligned} RBP(n) &:= (1-p) \cdot \sum_i rel(i) \cdot p^{i-1} \\ &= \sum_i rel(i) \cdot (1-p) \cdot p^{i-1} \end{aligned}$$

p can be see as the probability of moving to the next result.

4.3 Properties of effectiveness metrics

1. **Boundedness**
2. **Monotonicity**
3. **Convergence** – score decreases if a relevant document is replaced with a non-reported less-relevant document.
4. **Top-weightedness** – score decreases if a relevant document is swapp with a reported less-relevant document.
5. **Localization** – score for top k can be computed using only the top k results.
6. **Completeness** – A score can be calculated even if the query has no relevant documents.
7. **Realizability** – provided that the collection has at least one relevant document, it is possible for the score at depth k to be maximal.

4.4 Statistical Validity

Naturally, all experimental results must be statistically meaningful. We test against the null hypothesis that there is no difference between some two systems.

p-Value Likelihood that we observe a result based on the distribution implied by the null hypothesis (as such "just a coincidence"). Commonly assumed that hypothesis is valid if $p <= 0.05$.

Increasing statistical validity – By increasing the number of topics covered by the system. Then, we can feel more confident that average scores and therefore their differences between systems are meaningful/significant.

4.5 Determining Relevance

Ideal: Manual assession of all documents. **Problem**: Highly subjective, highly different results between different judges (Agreement rarely above 80%)

Infeasible because collections are very large. Instead: Only look at the sets of returned documents by the systems that are tested. **Assumption**: If a document is not returned by any tested system it is surely not relevant.

Pooling Use union of top k documents from each run as *pool*, give this to humans for relevance assessment.

Advantages:

- Fewer documents have to be manually assessed

Disadvantages:

- Have to pick size of pool (k) adequately so that meaningful but still feasible. Usually $k = 100$.
- Gives no real statement about how many documents have been found at all (some might not have been returned by any tested system)
- If a system does find a relevant document but ranks it lower than k, it is not taken into account.

Solve some of these problems by...

Pooling with randomized sampling Add to the pool some documents that are randomly sampled from the entire retrieved set (Use stratified sampling to get more higher-ranking ones)

Problem : Have to take into account that some documents are not ranked (not manually assessed).

BPref Measure for the quality of a ranking, takes unjudged documents into account. Summe über jd. relevante Dokument r: Anteil von Inversionen in Ergebnisliste bis zu r. Das Ganze geteilt durch Gesamtzahl relevanter Dokumente.

$$bpref = \frac{1}{R}\sum_{r}(1 - \frac{|n \text{ ranked higher than } r|}{\min(R, N)})$$

4.5.1 Designing User-based Evaluations

Can either measure • some relevance score based on user assessments or • user satisfaction. Will mostly talk about first point.

Evaluating relevance of queries – Approaches

- Rate for each document if it is relevant to given query
- Given two documents, rate which is more relevant
  - More assessments needed but
  - Better inter-annotator agreement
- Let user judge the entire list of results as a whole
  - How to present this in experiment (two-panel view is limiting, maybe interspersed list with click monitoring)

Discussion

- Doubts about the expressivity of *Recall*, *MAP*, *Cumulative Gain*.
- Have to design experiments properly (realistic tasks)
- Other kinds of relevance factors than simply relevance to topic? Also want • diversity • quality • credibility • ease of access

5 Web Search

5.1 Characteristics

- Large amounts of available information
- Network is highly dynamic
- Not managed or curated
- Different kinds of tasks: • Information • Navigation • Action • Exploration
- People will actively try to cheat (link farms, hidden text, cloaking, article spinning, ...)

5.2 Crawling

1. Start from set of seed pages
2. Extract URLs they point to
3. Place these on a queue to continue with in the next step

A crawler should be

- Polite: Avoid hitting a site too often, respect explicit constraints (robots.txt)
- Robust: Be immune to spider traps, handle spam pages well
- Distributed
- Scalable
- Performant/Efficient
- Quality-aware
- Fresness-aware: Do hit previously seen pages again to update information (tradeoff with Politeness)

5.3 Ranking

5.3.1 Basics on Link Analysis

Assumptions

- A hyperlink to page is a signal of quality for that page
- The anchor text (text surrounding the hyperlink) describes the content of the page

→ A link may describe the page better than the page itself

Usefulness of anchor text For each page, also index anchor texts pointing to it.

- Taking these frequencies into account too may increase query accuracy (IBM example)
- Anchor text may contain useful related terms (e.g. synonyms, nicknames) associated to the page.

5.3.2 PageRank

See contents from GrDa

Clarification on handling Dead Ends In the matrix, a dead end is a column with only zeroes. We replace these with $\frac{1}{n}$ (random teleport to any other page).

Shortcomings

- Measures generic popularity of a page – Biased against topic-specific authorities.
- Susceptible to link spam – articially link topographies (spam farms)
- Uses a single model/idea of importance
- Random Surfer model might not be adequate to reality (bookmarks, back button, search, ...)

5.4 Query Log Analysis

Objectives • enhance effectivess/efficiency of system • enhance user experience

Distribution of query popularity follows a Power Law.

Queries can be of different user intent:

- Information
- Navigation
- (Trans-)Action

Search happens in *sessions*

Users often repeat searches.

Use Cases

- User experience
  - Query Expansion (e.g. *fix iphone* → *fix iphone repair* ...)
  - Query Suggestion – provide list of queries related to current query; autocomplete
  - Spelling correction
  - Use logs as training data for ranking algorithms
- Efficiency – exploit usage information to cache/position/partition data accordingly in the system

Challenges

- (Clicks can be taken as a quality measure, however) users do not always want results that correspond to their profile.

6 Search Interfaces

Usability Five components

1. **Learnability** – How easy is it to accomplish a task for the first time?
2. **Efficiency** – Cost/Utility ratio

3. **Memorability** – How long does it take to re-establish proficiency after a period of non-use
4. **Errors** – How many errors does the user make, how easy is it to recover?
5. **Satisfaction**

Design Guidelines for Search Interfaces

- **Offer informative feedback**
  - Show search results immediately (without intermediary step)
  - Show document surrogates, highlight query terms
  - Allow sorting of results by various criteria
  - Show query term suggestions (*related term suggestion*, *term expansion*)
  - Use explicit relevance indicators sparingly
  - Support high responsitivity of system
- **Balance user control with automated actions**
  - e.g. automatic incorporation of similar/related terms in query
- **Reduce short-term memory load**
  - History mechanisms
  - Display information in a structured way, e.g. hierarchies, categories, *faceted metadata* (e.g. person characteristics), clustering
- **Provide shortcuts**
  - Deep links (direct links to subpages in surrogate)
  - Widgets on result page (e.g. time of day, calculator, election results, ...)
- **Reduce errors**
  - Spelling corrections
  - Avoid completely empty result sets – perform query expansion, ...
- **Recognise the importance of details**
  - Size/Shape of query text input field
  - Positioning of spelling corrections
  - ...
- **Recognise the importance of aesthetics**

7 Music Retrieval

missing

8 Übungsfragen

Beschrieben Sie die Qualitätsmaße Precision und Recall sowie deren Tradeoff.

- **Precision**: Anteil von zurückgegebenen relevanten Dokumenten an allen zurückgegeben Dokumenten.
- **Recall**: Anteil von zurückgebenen relevanten Dokumenten an *allen relevanten* Dokumenten.
- *Tradeoff*: Optimierung von Recall (gebe mehr hoffentlich relevante Dokumente zurück) führt potentiell zu Fehlern (gebe doch irrelevantes Dokument zurück), was wiederum Precision negativ beeinflusst. Optimierung von Precision (schränke zurückgegebene Dokumente besser ein) führt potentiell zu Fehlern (gebe manche relevante Elemente nicht zurück), was wiederum Recall beeinflusst.

Beschreiben Sie unterschiedliche Methoden zur Feature Space Reduction bei Textanalyse-Aufgaben

- Case Folding
- Stemming
- Stop Word Removal
- Remove Rare Terms

Beschreiben Sie unterschiedliche Methoden bzw. Arten von Feature Spaces, die bei der Indizierung von Textdokumenten zum Einsatz kommen

- Bag-of-Words – $\rightarrow$ binary independence model
- n-grams
- word stems

Beschreiben Sie die tfxidf-Methode zur Gewichtung von Termen, wie diese Maße berechent werden und welche Annahmen dahinter stehen.

- *Term frequency*: Anzahl Vorkommen von Term in Document
- *Document frequency*: Anzahl von Dokumenten, in denen Term vorkommt. Nutze log-dampening.
- *Inverse document frequency*: Nutze dieses Maß, um seltene Terme höher zu gewichten: $idf(t) = \log(\frac{n}{df(t)})$ (**Annahme**)
- Damit ist das tfxidf-Maß definiert durch

$$tf.idf(t,d) = \log(1 + tf(t,d)) \cdot idf(t)$$

- **Annahmen**:
  - Binary Independence Model (Terme sind unabhängig voneinander, lediglich Vorkommen eines Terms in Dokument zählen, Dokumente sind unabhängig voneinander)
  - Allgemein (in Korpus) seltene Terme sind von höherer Ausschlagskraft für die Gewichtung.
  - Anteil (Frequency) von Term an Dokument ist Indikator für Wichtigkeit.

Weshalb ist term frequency allein ungeeignet für das Ranking?

- **Lange Dokumente** sind mit vielen Vorkommen desselben Terms liefern eine höhere *term frequency* obwohl nicht unbedingt aussagekräftiger/wichtiger – absolutes Maß, abhängig von Länge des Dokuments.
- **Sehr oft vorkommende Terme** (zB Artikel) erhalten hohe *term frequency*, sind aber aufgrund Ihrer hohen Häufigkeit im gesamten Korpus nicht aussagekräftig – Nehme deshalb *inverse document frequency* (niedrig, wenn Term hohe *df* hat).