Information Retrieval and Web Search

Lecture 03

Lecture by Dr. Inbal Budowski-Tal
Typeset by Steven Karas

2018-04-08
Last edited 12:16:21 2018-04-08

Disclaimer These lecture notes are based on the lecture for the course Information Retrieval and Web Search, taught by Dr. Inbal Budowski-Tal at IDC Herzliyah in the spring semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Inbal Budowski-Tal.

Agenda

- Dictionary data structures
- Wildcard queries
- Word distance
- Tolerant retrieval
- Soundex

1 Recap

1.1 Token

A **token** is an instance of a word that occurs in a document.

Tokenization is nontrivial, because sometimes a token should include punctuation, and sometimes not (ex-wife; ice cream).

1.2 Type

A **type** is an equivalence class of tokens.

Tokens are typically preprocessed to reduce them into normal forms, called **terms**. The type of a token is the equivalence class of all tokens that reduce into the same term.

Preprocessing includes many steps, from case folding/normalization, lemmatization or stemming, morphological analysis, etc. Normalization for some languages is easier than others.

1.3 Skip pointers

Skip pointers are an augmentation of linked lists to increase the average case efficiency of some set operations.

1.4 Positional indexes

A nonpositional index is an index where each posting is just the document. In a positional index, each posting includes the exact location of the token in each document.

Positional indexes allow us to answer phrase and proximity queries.

2 Dictionaries

The dictionary is the data structure that allows us to lookup terms and get back a list of postings. Typically, there are two main approaches: hash tables and trees. The choice between hashes and trees is driven by many considerations such as hardware storage layout and access times, the expected rate of changes for the dictionary, tolerance to inefficient operations, and the desire for fuzzy searches. Prefix trees can offer autocomplete, suffix trees make text search very natural, etc.

Hash tables are good if the dictionary is not expected to change, as construction can be expensive and unpredictable ($O(1^*)$). However, lookup is $O(1)$.

Self-balancing trees such as B+ trees, Red-Black trees, etc provide good solutions that provide $O(\log n)$ insertion and lookup. The exact choice of tree implementation is driven by design considerations and hardware restrictions.

3 Wildcard Queries

Prefix queries such as `mon*` are easily answered by B-trees, since it's a range query. Suffix queries such as `*mon` can be answered by maintaining a reversed index of all terms backwards. If the wildcard occurs in the middle of the search term, one option is to intersect the term sets of the prefix and suffix queries. An efficient alternative to this is a **permuterm** index, which indexes a term by all of its rotations.

3.1 Permuterm index

To construct a permuterm index, generate all rotations of a term, using a symbol outside the term alphabet to denote the end of the word:

`HELLO` → `hello$ ello$h llo$he lo$hel o$hell`

- when searching for `X`, query for `X$`
- when searching for `X*`, query for `X*$`
- when searching for `*X`, query for `X$*`
- when searching for `*X*`, query for `X*`
- when searching for `X*Y`, query for `Y$X*`

3.2 k-gram indexes

A k-gram index is more space efficient than an equivalent permuterm index, but requires some extra processing to query. To construct the index, we enumerate all the k-grams consisting of all subsequences of length k that occur in a term (including the inserted boundary markers) This index maps from the k-grams to a list of terms that contain the subsequence.

`April is the cruelest month` → `$a ap pr ri il l$ $i is s$ ...`

Wildcard queries first generate a list of terms, which need to be filtered for false positives (e.g. `mon*` will query for `$m, mo, on`, which may produce `moon`). Terms are then queried against the term-postings index.

4 Spelling correction

Spelling correction has two main applications: correcting documents being indexed, and correcting user queries. An isolated word approach considers the spelling of each word independently of the rest of the tokens. A context-sensitive approach will consider nearby tokens as well, and may correct mistakes such as `the asteroid fell form the sky`. The reason we want to have spelling correction is usually due to OCR'd documents, which may have transcription errors.

Corrections can be sourced from a canonical dictionary, which may be too general purpose, or drawn from the term vocabulary itself (appropriately weighted). The candidate correction is typically defined as the term with the shortest edit distance from the misspelled token.

4.1 Edit distance

The **edit distance** between strings s_1, s_2 is the minimum number of operations to change s_1 into s_2. The specific operations that are considered is a detail of the specific distance metric.

More advanced approaches will weigh some operations as they may have been more likely.

4.1.1 Levenshtein Distance

Introduced by Levenshtein in 1966[1]. For the Levenshtein distance metric, the operations considered are insert, delete, and replace. There is an efficient dynamic programming algorithm that produces not only the distance, but also a possible sequence of operations.

4.1.2 Damerau-Levenshtein Distance

This extends the Levenshtein distance to include transposition as an operation.

4.2 k-gram spelling correction

We can enumerate all the k-grams in the misspelled term, and threshold all resulting terms by the number of matching k-grams. The resulting list can then be sorted by edit distance and the lowest distance term returned as the correction.

4.3 Context sensitive correction

In this case, take the k-word queries, construct alternative queries based upon k-gram isolated term corrections and suggest the subqueries that have the most hits.

5 Next Session

We'll discuss context-sensitive correction and soundex, as we ran out of time. The next lecture will be in 2 weeks, due to holidays.

References

[1] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[2] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.