CS3245

Information Retrieval

AY2022/23 Semester 2

Notes by Jonathan Tay

Last updated on March 5, 2023

Contents

Part I

Boolean Retrieval

1 Language Models

A **language model** is a grammarless, computational model created from collections of text.

They are used to assign scores (e.g. probabilities) to a sequence of words.

unigram model

Create a **frequency table** of all tokens (words) that appear in the collection.

Unigram models have insufficient context to model the order of words in a sentence.

***n*-gram model**

By remembering sequences of n tokens we can predict the n-th token given only the previous $n-1$ tokens as context (**Markov assumption**).

A unigram model is a 1-gram model, bigram model is a 2-gram model, etc.

However, n-gram models require exponentially more space as n increases.

The **count** of an input is the *sum* of the counts of all tokens in the input, while the **probability** of an input is the *product* of the probabilities of all tokens in the input.

However, if a token does not appear in the collection, its probability is 0, resulting in a probability of 0 for the entire input, which is undesirable.

1-smoothing is a technique to avoid this problem. It adds a count of 1 to every token in the collection, even if it does not appear in the input.

2 Index Compression

Index compression decreases the disk space required, increases the speed of postings lists transfer from from disk to memory, and increases the amount of data that can be stored in memory.

Heaps' law

$M = kT^b$, where

M is the vocabulary size (number of distinct terms in the dictionary), T is the number of tokens in the collection, and $30 \le k \le 100$ and $b \approx 0.5$.

Heaps' law is used to predict the size of the dictionary given the size of the collection.

Next, we define the **collection frequency (cf)** of a term to be the number of times that it appears in the collection.

This is different from (but positively correlated with) the **document frequency (df)** of a term, which is the number of documents that contain the term.

Zipf's law

$\mathrm{cf}_i = \frac{K}{i}$, where

cf_i is the collection frequency of the i-th most frequent term, and K is a normalizing constant (typically $K = \mathrm{cf}_1$).

Zipf's law is used to predict the collection frequency of a term given its rank.

In general, there are a few very common terms and very many rare terms.

2.1 Dictionary Compression

Every search begins with the dictionary, so keeping its memory footprint small is important.

Each entry in an uncompressed dictionary is typically of the form <term, df, postings pointer>.

However, this means every term has a fixed-width, which not only limits which the maximum length of a term that can be stored, but also wastes space for short terms.

dictionary-as-a-string

Every term in the dictionary is concatenated into a single string.

Each entry in the dictionary is then of the form <term pointer, df, postings pointer>, where each term pointer points to the start of each term.

blocking

Let k be the number of terms in a block.

For each term in a block, prefix it with its length (number of characters).

Then, only the first term in each block retains a term pointer, while the remaining terms are retrieved by adding the length of all the previous terms to the term pointer of the first term in the block.

e.g.:

```
...7systile9syzygetic8syzygial6syzygy...
...^ptr
```

front-coding

Sorted words typically have a long common prefix, so we can take advantage of this by storing the common prefix once and then only storing the suffix for each word.

An asterisk $*$ marks the end of the prefix and the start of the first suffix.

The remaining $k-1$ suffixes in the block are each prefixed with their length, followed by a diamond $\diamond$, then the suffix itself.

e.g.

```
8automata8automate9automatic10automation
8automat∗a1⋄e2⋄ic3⋄ion
```

2.2 Postings File Compression

The postings file contains document IDs which can grow to large integers.

By Zipf's law a small number of terms have very high cfs, which implies high dfs, and as such the gaps should be small.

gap encoding

After the first document ID, subsequent values are the difference between the current document ID and the previous document ID.

However, it's still inefficient to use the same number of bits to store a small gap versus a large gap.

Therefore, we want to minimize the number of bits used to store the gaps.

variable byte encoding

The smallest unit of data is still the byte, of which the first bit will be used as a **continuation bit**.

If the continuation bit is 0, then the next byte is part of the same integer. Otherwise, the byte is the last (least significant) byte of the integer.

Therefore, each byte can only store 7 bits of data, so we need to use multiple bytes to store large integers.

The postings list will be every single byte from the encoded gaps, concatenated together.

e.g.

```
5 => 10000101
214577 => 00001101 00001100 10110001
```

Part II

Ranked Retrieval