CS3245

Information Retrieval

AY2022/23 Semester 2

Notes by Jonathan Tay

Last updated on March 12, 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 $30 \leq k \leq 100$ 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

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

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

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 *cf*s, which implies high *df*s, 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

In Boolean retrieval, documents are either included or excluded from the result based on whether they contain the query terms.

While accurate, this can yield too many results without any consideration for the relevance of the documents.

3 Vector Space Model

In ranked retrieval, documents are scored from $[0, 1]$ based on how well the document matches the query, and sorted thereafter to yield only the most relevant.

scoring: Jaccard coefficient

Let A be the set of terms in the query and B the set of terms in the document.

The Jaccard coefficient is defined as:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad \text{if } A \cap B \neq \emptyset$$
$$\text{Jaccard}(A, B) = 0 \quad \text{if } A \cap B = \emptyset$$
$$\text{Jaccard}(A, A) = 1$$

However, the Jaccard coefficient consider neither the **term frequency (tf)** nor the **document frequency (df)**.

A document which contains the query terms more times should be scored higher than a document which contains the query terms fewer times, but the Jaccard coefficient assumes $tf = 1$.

Furthermore, rare terms are more informative than frequent terms, so we introduce the **inverse document frequency (idf)** weighting scheme.

tf-idf weighting

Let $tf_{t,d}$ be the number of times term t appears in document d.

Also, let df_t be the number of documents which contain t, and N the number of documents in the collection.

Then, $idf_t = \log_{10}\left(\frac{N}{df_t}\right)$.

Then, define the log-frequency weight $w_{t,d}$ as:

$$w_{t,d} = 1 + \log_{10}(tf_{t,d}) \quad \text{if } tf_{t,d} > 0$$
$$w_{t,d} = 0 \quad \text{otherwise}$$

Putting the two together, the tf-idf weight $(tf.idf_{t,d})$
$= w_{t,d} \times idf_t$
$= \left(1 + \log_{10} tf_{t,d}\right) \times \log_{10}(N \div df_t)$.

As such, the tf-idf weight increases with both the number of occurrences of a term in a document *and* its rarity in the collection.

scoring: tf-idf

Finally, for each term t in both the query q and document d, we define the term score as:

$$\text{Score}(q, d) = \sum_{t \in (q \cap d)} tf.idf_{t,d}$$

---

vector space model

Documents and queries can be represented in a **tf-idf matrix**, where each document is a column vector of tf-idf weights for each term.

These $|V|$-dimensional column vectors are sparse.

Therefore, we want to rank documents $\vec{d_i}$ in increasing order of their **cosine similarity** to the query vector $\vec{q}$.

In other words, the smaller the angle between $\vec{d_i}$ and $\vec{q}$, the more similar they are.

We do not rank by Euclidean distance as the distance between $\vec{d_i}$ and $\vec{q}$ is large even if their term distributions are similar.

scoring: cosine similarity

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

The square root terms performs **length normalization** so that weights are comparable across different vectors even if they have different lengths.

Queries and documents may have different weighting schemes, in the form *ddd.qqq*.

lnc.ltc

document: logarithmic *tf*, no *idf*, with cosine normalization.

query: logarithmic *tf*, with *idf* and cosine normalization.

We avoid using *idf* for documents as insertion of new documents would require recomputation of the *idf* for all terms in that document, and normalization for all existing documents, which is inefficient.
