Discussion

Pre-workshop

• Revise the cosine similarity metric and the basic approach toward using it to apply a TF-IDF model for ranking documents in order of relevance to a query.

• Revise the kinds of information stored in an inverted index.

Workshop

1. Summarise the various advantages and disadvantages of storing per-document frequencies versus per-document weights in the postings list of an inverted index.

2. Revise the rationale behind “pivoted document length normalisation”. Identify the components of the formula. What advantages and disadvantages does this method have against un-normalised cosine similarity?

3. Revise the notion of “front coding.” How does it help us compress the dictionary for an inverted index? Construct a front-coded representation of a dictionary containing the following entries:

   abacus, abaft, abate, beetle, chin, chit, deal, deed, dent, dental

4. Revise the notion of “variable byte encoding”. Why does using such a scheme allow us to compress a postings list more effectively?

Post-workshop

• (Extension) Revise the notion of “entropy.” Shannon\(^1\) famously observed that we should use \(\log_2 \frac{1}{p}\) bits to encode a symbol with probability \(p\) (in a distribution). Consider the distribution of values that we wish to store in an inverted index: what is the most common document ID value? document-gap value? term frequency value? document position value? What impact does this have on our choice of compression mechanism?

Programming

Pre-workshop

• Complete previous weeks’ exercises, if necessary.

Workshop

1. Take some time to think about — and begin work on — the project.

Post-workshop

• (Extension) Try implementing one or more index compression schemes. Note that Python isn’t necessarily the right environment for complicated bit-wise data management!