Data mining: lecture 12

Edo liberty

1 Inverted Index

Search engines are based on the idea of an Inverted index. (not taking into account ranking which is, just as, if not more important...)

By a search engine we will refer to data structure and algorithm for fetching matching document regardless of their importance or "ranking". More precisely, we will consider a corpus of N (the symbol N will refer to the set of documents as well) documents each of which is comprised of a ordered subset of words from a dictionary m of m possible words (the symbol m will refer to the number of such words as well). We can preprocess these documents into a data structure offline. Then, given a query $Q \subset m$ we should retrieve all the document which include the query, i.e. $\{P | P \in N,\ Q \subset P\}$.

The trivial solution to this problem is make one pass over the data entire corpus and filter out document which do not include the the search query. Assume documents are indexed, this solution will require at least $\Omega(\sum_{i=1}^{N} \text{polylog}(|P_i|, |Q|)) = \Omega(N)$. Unfortunately, the number of documents is usually considered to be very large and this solution is considered to be inefficient.

The inverted index structure keeps for every word in the dictionary a list all documents in which it is contained. This list is called the "posting list" of the word, which we will denote by $N(w)$. Resolving a search query is therefore identical to computing $\cap_{w \in Q} N(w)$. This can be done in a straight forward manner by examining only the documents in $\cup_{w \in Q} N(w)$. Because, most words in the dictionary appear only in a small fraction of the documents[1] usually we can expect that $\cup_{w \in Q} N(w) \ll N$.

An observation which makes this computation more efficient is that the documents in the posting lists can be kept in a sorted order by their "doc-id"s. By scanning the shortest posting list and binary searching the rest it is immediate that $O[\min_{w \in Q}(|N(w)|) \cdot \sum_{w \in Q} \log(|N(w)|)]$ should suffice. In fact, this is far from being the optimal traversal of these lists. Can you come up with a better algorithm?

Remark 1.1. *Another "trick" is to number the document (give doc-ids) according to their rank. This way, the search on the sorted lists can stop after having found any pre-specified number of wanted matches.*

[1]Excluding very common words like: "is", "the", "a", etc' which are called "stop words".

There are good news and bad news. The good news is that the nature of search queries and documents is such that these algorithms (along with stupendous feats of engineering) give us the reliable and fast search engines we know. The bad news is that they can be shown to require $O(N)$ operations per search in the worst case. For example, consider the atheistic query "Why am I? because I am and that is it". Another example would be a search engine in a fictional language in which there are only $1,000$ words (say) and every word appears in every document with equal probability.

2 The containment query problem

I the set containment problem we aim to solve the complementary problem. Our goal is to retrieve the set of documents which are contained in the query, i.e. return $\{P|P \in N,\ P \subset Q\}$. Notice that the two problems are identical. If we map each document $P \to m \setminus P$ and $Q \to m \setminus Q$, then any results to the set containment problem will hold for the search problem as well.

There are two brut force solutions for this problem. *i)* pre-compute the search results to all 2^m possible queries. This would require only $O(m)$ operations per search but $\Omega(N2^m)$ space. *ii)* perform no preprocessing at all and require only $O(Nm)$ space but also $\Omega(Nm)$ time per query search. Unfortunately, both of these solutions are unsatisfactory. Even more unfortunate is the fact that recent hardness results show that queries cannot be resolved in logarithmic time if the data structure is of polynomial size. That said, no known algorithm come close to those hardness results' bounds.

In this lecture we will present one of the algorithms presented in [1]. The idea of the algorithm is to solve the problem in a brut force manner but on a small sample of the words. We start with the preprocess stage:

1. Select each word in the dictionary into a set S w.p. p; $|S| \sim pm$.
2. For each subset $Q' \subset S$ keep a list L of all the documents which are contained in in Q' with respect to S i.e. $P \in L$ is $(P \cap S) \subseteq Q'$.

Let us consider the relation between documents in L and the query Q. First, if $P \subset Q$ then $P \in L$. Second, L does not contain too many documents which are far from being contained in Q. We say that P is x-almost contained in Q if $|P \setminus Q| \leq x$.

Fact 2.1. *A query Q is said to be "good" for a list L if at most $2N \cdot 2^{-px}$ of its members are not x-almost contained in Q. A query Q is good for its list with probability at least $1/2$.*

Proof. If $P \in L$ and P is not x-almost contained in Q ($|P \setminus Q| > x$) it means that none of the x words in $P \setminus Q$ was picked into S. That happens with probability $(1-p)^x \leq 2^{-px}$. Therefore, in expectation, not more than $N2^{-px}$ documents which are not x-almost contained in Q will be included in L. This also means that with probability at least $1/2$ no more than $2N2^{-px}$ such documents will be contained in L. □

Now, in order to search efficiently through the documents in L, we find a representative set R such that:

1. For every Q that is good for L we have $|R \setminus Q| \leq g = pm$ (Intuitively, R is not too large, it g-almost contains all the good queries for L)
2. Except for $(p/2) \cdot N \cdot 2^{-px}$ documents in L for the rest $|P \setminus R| \leq x/p$. (Intuitively, R cannot be too small because for a large portion of L it contains all their elements except for at most x/p)

Lemma 2.1. *Such a representative set R exists. See [1] for details.*

We can now restrict our search within L to documents which do not include any word in $R \setminus Q$ (otherwise they are clearly not included in Q). For that we use the first property of our representative set. Since $|R \setminus Q| \leq g$ we keep a table of size $\binom{m}{g} \leq 2^{pm\log(m)}$ in which we store in location g (a subset) every set in L for which $P \cap g = \emptyset$.

Now, we collect all documents according to their elements that lie outside of R. Since we know that all the words they contain which are contained in R are also contained in Q. Since we can have only $\binom{m}{x/p} + (2/p)N \cdot 2^{-px}$ such differences (property 2 of the R) we must check only those, which gives the running time of the algorithm. This, however, must me repeated m times since we must succeed for all 2^m possible queries and here we only succeed per query with probability 1/2. Setting $t = px$ and solving for $\binom{m}{x/p} = N \cdot 2^{-px}$ we get:

1. Space $= N \cdot 2^{O\left(m \log^2(m)\sqrt{c/\log(N)}\right)}$
2. Query time $= N/2^c$

References

[1] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 451–462, London, UK, 2002. Springer-Verlag.