ComS 535x:
Research Report

Due on May 8, 2015

Instructor: Professor Aduri, Pavankumar

Yuanyuan Tang, Chenguang He

Contents

Hashing and Bloom Filters

Bloom Filters is the one of most advanced data structure invent in recent years. It use the hash functions to reduce the space complexity. It is a probabilistic data structure, so the false positive exists. However it is relatively small, and there is no false negatives.Although Bloom filters allow false positive, for many applications the space efficiency take over this drawback, when the probability of an error is sufficiently low. In the following section, we discuses the hash function, structure of bloom filter and the usage of bloom filter.

A hash function is a function that take an item of data and process it to produce a value or key. It is a mapping that map two objects in a relation. For example, we can easily define the hash function of a string with the sum of all character and mod a large prime. When we want to check the whether two string are same, we only need to compute the hash function and check the result whether is same. This approach can save a lot of memory to store strings. However, a hash function is hard to be a one to one mapping function, since when two different data values to produce a same hash value, we call it hashing collision. A good hash function can reduce the hashing collision by spreading the data into the array. In the class, we discuss two normal hash functions: Random Hash Function and Deterministic Hash Functions.

Random hash function is defined by choosing the large prime p and define a and b from the range of [1,p-1], the function is $h(x) = (ax + b)\%p$. The advantage of using the random function is that, the function is defined by random large prime p which the selection of p can be variant, in other word, if we want two different random hash function, we just select different large prime p, it is very efficiency to build the function. Also, when we use the random hash function, we only need to store the [a,b,p] in a 3-tuple. It save a lot of space to store the functions themselves.

Deterministic hash function is defined by a constant large prime p, we try to build a formalized hash function h, such that, h is defined by take a large prime 109951168211, we call it FNV64PRIME and a offset of 14695981039346656037, we call it FNV-64INIT, the hash function is that: h = h xor s[i] and h = (h*FNV64PRIME)$\%2^{64}$. We call it deterministic hash function, because the parameters of hash function are constant, therefore, if we want to use different hash function, we need to have a algorithm to compute the offset, it is given in http://www.isthe.com/chongo/tech/comp/fnv/. The advantage of deterministic hash function is that, when we want to use a different hash function, we only need to change the offset of hash function to get the new function, it save the time to find the prime.

The Bloom Filter is build by constructing k hash function, and it has the interface of add(), contains(). However, the original approach of Bloom Filter can not move the element, the reason is that you can not unset a bit that appears to belong to a data item because it might also be set by another data item. When we add an item into Bloom Filter, it compute k hash function of the item and set the bitset of k values to true. When we search an item in Bloom Filter, we compute the k hash functions and check whether one of them is true in bitset. The Bloom Filter trade the accuracy for space, we can calculate the false positive rate: $(1 - e^{\frac{-kN}{M}})^k$, where k is the number of hash function, N is the size of item size S, M is the size of bit table.

Document Similarity

Document Similarity is a large used technique to find out how similar two document are. There are many powerful tools in Document Similarity, such as, Min Hash, Jaccard Similarity and Cosin Similarity. In the following section, we discuses the Min Hash and Locality Sensitive Hashing, which is a technique to have the signature of a document, when we want to compare two document, we only compare their signatures, it saves a lot of time to look at the entire document. Finally, we discuses the usage of those tools in industry.

In order to compute Min Hash, we firstly need to process the document as vector. The reason is that, the document are arbitrarily large, we do not have enough memory to hold the entire set of document and compare all of them with each other. The idea is that, we need to do some pre-process to compute the entire document into vector. The first thing we need to do is that, we need to filter the STOP word, such that, are, the. and. Those words are too frequency in documents, they are not helpful for making the signature. Secondly, we define the length of shingle to split the words in document, a shingle is a substring of a word, the reason that we need to do shingling is that, words have different tense, shingling can reduce the affect of tense. Finally, we compute $T_f IDF$.

$T_f IDF$ is one of most significant way to find out the weight of a word in a document. $T_f IDF$ consists of two parts, first part is T_f and second part is IDF. T_f represents term frequency, it means that the frequency of a term in one document. It normally represent in a vector, we call it term-frequency vector. Finally, we need to normalizing the term-frequency vector, because we do not want the number in vector is too different with each other.IDF represent the Inverse Document Frequency. It means that the number of a document a term appears. The reason that we need to compute IDF is that, some words are too frequency showing in the set of document, even they are not STOP words, they are may not very important. In order to define a word is important, we have to compute IDF to reduce the affect of those words. The document vector define as the product of T_f and IDF.

Once we have the set of document vector of documents, we can compute the Min Hash matrix. The Min Hash matrix represent the signature of set of documents. Firstly, it computes k different permutation and for each of them, compute the minimum value in each document vector. It result in a set of k minimum values for each document as the column, and number of document as the row. The reason that we use Min Hash to represent the document vector is that, the original document vector are too spares, there are many null pointers in the vector, by computing the Min Hash, it largely reduce the size of document vector.

Finally, we compute the Locality Sensitive Hashing. It split the Min Hash matrix into k-tuples and pick a random hash function to compute the k-tuples into a set. It define two document are similar, if two document that are mapped into the same bucket. Because the step of computing Locality Sensitive Hashing can reduce the sensitive of words, so we can compute the possibly of two document are mapped into same bucket is that $1-(1-s^r)^b$. It is relatvily small if we choose b and r so that s approximately equal to $(1/b)^{1/r}$.

Crawling and Page Rank

To build a search engine of web, we need to build a web crawler and rank the page properly.

1 Crawling the web

Basically, the world wide web could be considered as a directed graph. Each web page is a vertex in the graph. If page p has an outgoing link to page q, then there is a directed edge from p to q.We could use Breadth First Search(BFS) to traverse this directed graph. A web crawler usually start with a URL, called the seed, which is the root in BFS search.

In crawling process, there are several policy to follow. The first one is politeness policy. If a single crawler sends a large number of requests per second or downloading large files, a server would be overloaded and unable to keep up with requests from other crawlers. Therefore, if too many requests arriving at a server from a single source, the server may choose to deny these request. To avoiding this situation, crawler should be polite, that is, after sending a batch of request to a server, the crawler waits for sometime before sending another bathc of requests to the same server. A typical interval would be 10 seconds to 15 seconds[1, 2]. Another polite action for a crawler is to follew the robots exclusion protocol(robots.txt), which tells the crawler which parts of their web servers should not be accessed by crawlers.

Besides the politeness policy, there are other policies to follow such as parallelization policy and selection policy. If a crawler apply multiple processes in parallel to maximize the download rate, it must be careful to avoid repeated downloads of the same page. The crawlers use parallelization policy for assigning the new URLs discovered during the crawling process. In this way, if the same URL is found by two different crawling processes, it is considered as new URL once. Generally, different crawlers may apply different policies which focu on different topic and might be perform better for some specific purpose. The following are some famous web crawler architectures: Bingbot, FAST Crawler[3], Googlebot[4],GM Crawl[5], PolyBot[6].

2 Page Rank

Intuitively, web search engines use web crawlers to to catch the content of each web page and listing the terms found in each page. With these information, an inverted index could be build upon all the web pages. With this index, we could perform a query and rank each web by the similarity between the page and the query. The one with the highest similarity of the query is the highest ranked page. However, this method bears a major drawback: it cannot identify spam page. Unethical people have a large opportunity to fool this kind of search engines into leading users to spam page. They could just make their own pages contain a lot of terms and repeat them a lot on the page. When users query those terms, they would be easily lead to those spam pages.

To combat term spam, a totally different criterion came into being. Beside ranking page by the terms they contain, people also rank page by the terms used in or near the links to that page. The whole precoss simulate random surfer on web: starting at a random page, we follow randomly chosen outlinks from the page at which we currently located, and repeat this option many times. Pages which have a large number of surfers were considered more important that pages that rarely be visited. Users are more likely to visit useful pages than useless pages. Therefore, considering the chance they could reach this page, page could be properly ranked.

Then we consider the mathematical definition of page rank. Page rank vector $R_0 = < r_1, r_2, ..., r_n >$, G = (V, E) is the web graph. M is the matrix presentation. Let $\tilde{M_{ij}} = M_{ij}/d_j$, where d_j is the out degree of vertex j. $R_1 = \tilde{M}R_0$ simulates one step of random surfer on the web. would be a new vector. In this way, we could computer R_1, R_2, ... If the graph is strongly connected and non-bipartite, then any random walk (starting from any initial distribution) converges to an unique stationary distribution. Therefore, R_1, R_2, ... would finally converge.

However, there are some anomalies such as sinks and spider traps. Page rank has to modify to deal with sinks and spider traps: Instead of randomly pick an outgoing link, toss a biased coin with probability of head
. If the outcome is tail, then uniformly at random pick a page (from all all possible pages), otherwise uniformly at random pick a link from the current page (if current page does not have any links, then uniformly at random pick a page). Note that this not only avoids spider traps, but also (implicitly) makes the graph strongly connected. In this way, R_1, R_2, ... would finally converge and reach an unique stationary distribution.

Information Retrieval

1 Inverted Index

An inverted index has two parts, Dictionary and Postings. The terms are stored in a dictionary. For each term t in the dictionary we have a list that records which documents in which t appears. Each item in this list (typically a document), is called a posting and the list is called postings list. The collection of all postings is list is called postings.

Given a term t, we can retrieve all documents that contain t, by simply searching the dictionary for the term t and retrieve postings list associated with t. Suppose that the query q is $t_{i1} \bigwedge t_{i2}$. Then retrieve the postings list P_1 associated with the term ti1 and the postings list P_2 associated with the term t_{i2} and retrieve all documents that are in the intersection of P_1 and P_2. If the query is $t_{i1} \bigvee t_{i2}$, then retrieve all documents in the union of P_1 and P_2. Clearly, this method can be extended to handle queries with more complicated boolean expressions consisting of multiple terms.

2 Vector Space Model

The boolean retrieval cannot rank the documents according to their relevance to the query. Vector Space Model could solve this problem by assigning a weight to each term-document pair. Let T = $(t_1, ..., t_M)$ be the collection of all terms in the document collection and D = $(d_1, d_2, ..., d_n)$ be the collection of all the documents. Given a document d, we can view it as the following vector in M-dimensional space: v(d) = $(w_{t1d}, w_{t2d}, ...w_{t_m d})$. Here, we use tf-idf as the weight of term in each document. w_{tid} is the weight of ti in document d. For each query, we could also computer the weight of each term in the query. v(q) = $(w_{t1q}, w_{t2q}, ...w_{t_m q})$.Using the same ideas in document similarities, relevance rankings of documents in a query can be calculated by comparing the deviation of angles between each document vector and the original query vector where the query is represented as the same kind of vector as the documents. Practically, we just calculate the cosine of the angle instead of angle itself: $cos\theta = (v(d) \cdot v(q))/(\|v(d)\| \cdot \|v(q)\|)$. Document with the highest cosine similarity with the query is considered as of the highest relevance with query.

References

[1] Baeza-Yates, R. and C. Castillo, Balancing volume, quality and freshness in Web crawling. In Soft Computing Systems Design, Management and Applications, 562 (2002).

[2] J. Cho et al., ACM Transactions on Database Systems, 28(4)(2003).

[3] K.M. Risvik et al.,Computer Networks**39**, 289 (2002).

[4] S. Brin et al., Computer Networks and ISDN Systems, 30(1-7):107(1998).

[5] http://www.aleph-networks.com/en/bigdata.php?solutionstabs=0, GM Crawl : Identifies and collects data from the internet (2014)

[6] V. Shkapenyuk et al., 357. In Proceedings of the 18th International Conference on Data Engineering (ICDE)(2002).