Data mining: lecture 4

Edo liberty

Bit Hashs vs. Bloom Filters

We want to represent a set of elements S with a data structure such that we can query if an element q is in S or not.

In other words, we must implement a data structure that supports $Add(e)$ and $Query(q)$. Our demand is that if $q \in S$ the data structure function $Query(q)$ will always return True. if $q \notin S$ then $Query(q)$ should return False, but we are willing to receive False Positives with probability δ, i.e., $Query(q)$ returning True although $q \notin S$. For notation's sake, we say that $|S| = n$.

Although we can use a traditional Hash table, we want to avoid keeping the elements of S. These might be long strings, for example, and keeping them would be wasteful. Also, because we are willing to accept False Positives, we can alter the structure of the Hash table and keep only one bit per entry in the array, we will call them bit-hashes.

Bit-Hashs

For a bit-hash structure we keep a bit array A of length N all initialized to $False$. and a hash function h : elements $\rightarrow [1, \ldots, N]$.

1. define $Add(e)$:
2. $\quad A[h[e]] \leftarrow True$
1. define Query(q):
2. $\quad$ return $A[h[q]]$

Let us see how large N (the array) needs to be in order to insure that the probability of a false negative is at most δ. Clearly, after inserting n elements into the array, the number of bits that are set to $True$ is at most n. Thus, the probability of a false negative is at most n/N. This gives us that $N \geq n/\delta$, the error probability is at most δ.

It is easy to see that this is not only sufficient but also necessary. Assume, $n < N/3$, the probability that an element in S occupies a new cell in the array when inserted is at least $2/3$. Thus, with very high probability, the number of cells set to $True$ after all the elements of S have been added is at least $n/2$.

This gives us the other side of the equation. If $N < n/2\delta$, the error probability is at least δ.

Bloom Filters

Now assume that we have the same bit array of size N but instead of keeping one hash function we keep k of them, $h_1, \ldots, h_k$.

1. define $Add(e)$:
2. for i in $[1, \ldots, k]$
3. $A[h_i[e]] \leftarrow True$

1. define Query(q):
2. return $\wedge_{i=1}^{k} A_i[h[q]]$

It is easy to see that for any $e \in S$ we will always get $Query(e) = True$. Now, what is the probability that $Query(q) = True$ for $q \notin S$. That will happen if all k bits of $h_i[q]$ are set to $True$. Since we have entered n elements for which we have randomly chosen k bits (possibly with repetition) the probability of a single bit to still be set to $False$ is $(1 - 1/N)^{nk}$. The event that all k positions of $h_i[q]$ are set to $True$ thus happens with probability $(1 - (1 - 1/N)^{nk})^k$. By choosing $k = N/n$ we get that this is roughly $(1 - e^{-1})^{N/n}$. We get that this is less than δ if we choose $N > 3n \log(1/\delta)$. Note the huge improvement, $N > 3n \log(1/\delta)$ for Bloom filters vs. $N > n/2\delta$ for bit hashes.

Also, note that since $k = N/n$ we get that we need only $O(\log(1/\delta))$ hash functions and thus $O(\log(1/\delta))$ operations per insertion of an element into the structure.