Similarity Filters
An Introduction

Marcus Gregersen
mabg@itu.dk

Martin Faartoft
mlfa@itu.dk

Rick Marker
rdam@itu.dk

April 22nd 2014

1 Introduction

In the following we investigate the Similarity Search problem. Different approaches have been explored by earlier work, and in this paper we will investigate two of those; 'Distance-Sensitive Bloom Filters'[1] by Kirsch and Mitzenmacher, and 'Locality-Sensitive Bloom Filter for Approximate Membership Query'[3] by Hua et al.

1.1 The Problem

The Similarity Search problem can be stated as follows: Given a set of elements S, determine if there exists an element $s \in S$ that is 'close' to a given query element q. Where 'close' is defined as being within a given distance, d using a certain metric.

In the following we consider the Similarity Search problem for bit-vectors of length l, and Hamming distance as metric. This reduction maintains a high degree of generality, since many domains can be encoded with bit-vectors.

1.2 Definitions

Hamming distance

A distance metric for bit-vectors. The Hamming distance between two vectors v_1, v_2 is defined as the number of positions i where $v_1[i] \neq v_2[i]$.

Bloom Filter

A Bloom filter is an inexact representation of a set that allows for false positives when queried; that is, it can sometimes say that an element is in the set when it is not. In return, a Bloom filter offers very compact storage: less than

10 bits per element are required for a 1% false positive probability, independent of the size or number of elements in the set.[2]

Locality-Sensitive Hashing

Regular hashing tries to spread out the hash-values of different elements, to minimize the probability of a collision. Locality-Sensitive Hashing (LSH) tries to group similar elements, by maximizing the collision probability for similar elements.

The LSH is closely tied to the distance metric, and many distance metrics have no known LSH. The Hamming distance metric on bit-vectors has a particularly simple LSH: Sample a fixed number of bits from the input vector, uniformly at random.

It is intuitively obvious, that if two elements only differ on the non-sampled bits, then they will hash to the same value, and thus be considered 'close' by the LSH.

The more bits two vectors have in common, the higher the probability will be that a random LSH will hash them to the same value.

1.3 Naïve Approaches

Brute force The most obvious idea for solving the Similarity Search problem, is by brute force. Store the elements S in a linked list. When a query is made, simply scan the linked list, and calculate the distance from each element $s \in S$ the query element q. If s satisfies the distance requirement, a match has been found. If the end of the linked list is reached, no match exists.

This will give a correct answer, and will work well for small problem instances, But the linear requirement on time and space in the total number of bits in S, is prohibitively expensive for many real-world applications.

Bloom filters If we relax the requirements on space or time, we can change these characteristics to support either constant-time queries, or space-efficiency, but not both.

While keeping in mind that standard Bloom filters only answer exact membership, we see that one way of achieving this is by relying on extra insertions. Everytime an element is added to the Bloom filter, all elements within distance d are generated and added in addition to the original element.

Another way to go around this is by sacrificing time. This can be done by still doing normal insertions but instead querying for all elements with distance d of the element we are comparing to. This approach has the benefit of still being space efficient but will cause an exponential blowup of the running time.

2 Related work

2.1 Distance-Sensitive Bloom Filters

To be able to perform non-exact matching in Bloom filters, Kirsch and Mitzenmacher propose, in [1], a novel way of using Bloom filters. They replace the ordinary hash functions with LSH hash functions. When a query is received, they ask all the hash functions if they have seen something similar to this element, and if more than a certain threshold, t returns true, then the data structure returns true, meaning that a similar element was found in the data structure.

The main disadvantage to this approach, is that it introduces the possibility of false negatives.

In their experiments they manage to achieve false postive rates and false negative rates of 1.5% and 0.2% respectively for 1000 elements, and 0.016% and 0.003% for 1000. For 1000 elements they use 64% of the total space of the elements, and for 10000 elements they use 51% of the total space.

2.2 "Locality-Sensitive Bloom Filter for Approximate Membership Query"

The paper written by Hua et al.[3], takes a slightly different approach than [1]. They use a standard Bloom Filter data structure, with LSH functions in place of ordinary hash functions, but no thresholding. They call this a 'Locality Sensitive Bloom Filter'. This 'naive' approach, has a high probability of both false positives and false negatives. To minimize these, they augment the Bloom filter with additional data structures, that will not be discussed further here.

In the report they have chosen to use a 'proximity measure', which makes it impossible to compare the results to the ones found in [1] directly and all data presented is only available through graphs. From the graphs, however, we can see that they achieve between 85% and 100% accuracy.

3 Ideas and Preliminary Results

We have mainly been focused on the 'Distance-Sensitive Bloom Filters'[1] paper, and have achieved the following:

We have implemented the data-structure, and can roughly reproduce their results.

We suggest a small improvement to the way the LSH functions choose what bits to sample. If we force each bit to be sampled the same number of times, we make the thresholding more robust against 'unfortunate' choices

of bits to sample.

Building on the previous idea, we suggest a way to eliminate false negatives completely: If every bit is only sampled a fixed number of times n, and a query element q is 'close', meaning it only differs from s in at most ϵ bits, then setting the threshold to: $t = k - n \cdot \epsilon$ will eliminate false negatives. The downside is an increase in the number of false positives, and the fact that the data structure no longer works for some parameter settings ($t < 1$).

Finally we aim to experimentally investigate if the choice of threshold in [1] can be improved.

References

[1] A. Kirsch and M. Mitzenmacher, "Distance-Sensitive Bloom Filters", Proc. Eighth Workshop Algorithm Eng. and Experiments (ALENEX), 2006.

[2] Bonomi, Flavio and Mitzenmacher, Michael and Panigrahy, Rina and Singh, Sushil and Varghese, George, "An Improved Construction for Counting Bloom Filters", Algorithms – ESA 2006

[3] Yu Hua, Bin Xiao, Bharadwaj Veeravalli, Dan Feng, "Locality-Sensitive Bloom Filter for Approximate Membership Query", IEEE Transactions on Computers, Vol. 61, No. 6, 2012