Final Project Report: Priority-Sensitive Bloom Filters *

Chad Brubaker & Alex Clemmer

April 9, 2012

Abstract

A Bloom filter is a space-efficient representation of an arbitrary set that allows for probabilistic membership queries, providing an ϵ-small one-sided probability of false positive.

One problem with the traditional Bloom filter is that this error is always the same for all possible membership queries. However, the assumption that all elements *should* have the same probability of error is inappropriate in many cases, particularly so in situations where the accuracy of the set membership query for certain elements is vastly more important than it is for others.

In this report we will demonstrate that it is possible to drive down the error selectively for "important" elements in the set by adjusting the number of hash functions used to test set membership. We will also begin to pin down the analysis of the guarantees that can be provided by the techniques we outline.

1 Motivation

On a more practical basis, it is not at all uncommon for data to exhibit a Zipf-like trend in the real world, which generally has meant that actionable knowledge that provides realistic tools for dealing with sparse features tend to be very welcome, especially in domains where experiments tend to be very memory-intensive (*e.g.*, NLP and ML).

Our motivation specifically was to make it possible to do machine learning algorithms using massive amounts of natural language data, but using only commodity computers. Recent algorithmic advances have undoubtedly made this much easier, but our progress was nevertheless still impeded by the fact that the long tail of sparse features washed out accuracy for the very important features, and this was the inception of the techniques presented here.

2 Problem Statement

We adopt the traditional notation for Bloom filters. That is, the classical Bloom filter represents an arbitrary set of n elements $S = \{s_1, \ldots, s_n\}$, and consists of two basic components: (1) an m-bit array A, where all bits are initially set to 0, and (2) a set of k hash functions $H = \{h_1, \ldots, h_k\}$, where each hash function $h_i : \mathbb{R} \to \mathbb{N}$ maps integers into the range $[0, m)$.

*CS 6955 Data Mining; Spring 2012 Instructor: Jeff M. Phillips, University of Utah

2.1 Background on the Bloom Filter

The Bloom filter represents a particular $s_i \in S$ first by feeding it into each of the k hash functions. Each particular hash function $h_j \in H$ will produce an integer value, for a total of k integer values. For each of these k integer values, we set $A[h_j(s_i)] = 1$, *i.e.*, we treat each of the k values as an index into A, and at each of these k points in A, we set the value to 1 (note that we set it to 1 even if it has already been set to 1 beforehand!).

This leaves us the problem of checking set membership. Let's call our query point q_i. Similar to the above, we begin be feeding q_i into each of the k hash functions to get k integer values. However, unlike above, we don't set any values in A—instead, we check to see that each of the k value in A is set to 1. If all k of the values are set to 1, we conclude that q_i is in the set S; otherwise, we conclude that it is not.

2.2 Error Rates in Bloom Filters

Bloom filters have one-sided error, *i.e.*, they are only capable of producing false positives. Our job in this paper will be to *reduce the probability of false positives on a selective basis*, and as such it will be useful to characterize the false positive rate of the generic Bloom filter as a starting point.

Classically, the probability of any particular bit being set to one in an m-bit Bloom filter representing an n-element set using k hash functions is denoted by

$$\mathcal{E} = 1 - \left(1 - \frac{1}{m}\right)^{nk} \tag{1}$$

Another way to think of this is as the probability that all the bits except one particular bit are *not* flipped, over k hash functions and n different items. Another classic result gives us the false positive rate:

$$\text{error}(k) = \mathcal{E}^k = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \tag{2}$$

2.3 Establishing a Notion of "Importance"

In addition to the set S let us adopt $Q = \{q_1, \ldots, q_p\}$ to indicate our set of p query points. Generically, we'd like to attach to each of these points a measure of *importance*, which we will eventually optimize to find the allocation of hash functions that most greatly reduces the probability of false positives.

Formally, this can be represented as a function $I : \mathbb{R}^d \to \mathbb{R}$, which takes a query point and returns a real-valued importance score. At the outset, it is obvious that this formulation will cause us problems—I is assigning real-valued importance scores to each of the query points in the set Q, but what we really need is a natural number that we can optimize to tell us how many hash functions to allocate to a particular point. How, after all, do you allocate 2.73 hash functions to query point q_i? This looks dangerously like an IP problem.

Unfortunately, it is necessary to formulate I in this way for the sake of generality. Consider, for example, the very likely scenario that I is a function that operates on a probability distribution, such as the probability that a particular query point is queried.

2.4 Theoretical Setup