Distances from Sketches

Martin Strauss

November 27, 2002

Abstract

We describe a general-purpose reusable data structure that stores small "sketches" of vectors and approximates the L^2 distance between the vectors from the sketches alone.

1 Definitions

We want a data structure that supports three operations:

- `sketch_handle_t initialize( int` N`, float` ϵ`, int` k`);`
- `sketch_t sketch( vector_t` v_1`, sketch_handle_t` h`);`
- `float distance( sketch_t` s_1`, sketch_t` s_2`);`

The semantics are as follows. The length of vectors involved is N. We will seek distances correct to within the factor $(1 \pm \epsilon)$ with probability $1 - 2^{-k}$.

- `sketch_handle_t initialize( int` N`, float` ϵ`, int` k`);` Generate and return $48k/\epsilon^2$ truly random seeds for later pseudorandom number generation. Also keep the parameters N, ϵ, and k. Each seed can be $O(\log(N))$ random bits.
- `sketch_t sketch( vector_t` v_1`, sketch_handle_t` h`);` For each seed s in the handle, expand s into a string r of N 4-wise independent random variables, for example, by multiplying s by the parity-check matrix of a linear error-correcting code of distance 5 (See [AS92] for the connection between error-correcting codes and random variables of limited independence. See [6] for many examples of linear error-correcting codes.) Compute and store the real dot product $\langle v_1, r \rangle$. Thus each sketch can be regarded as a vector of $48k/\epsilon^2$ reals.
- `float distance( sketch_t` s_1`, sketch_t` s_2`);` Let $d = s_1 - s_2$. Return $\text{median}_{3k} \text{average}_{16/\epsilon^2} d[i]^2$. That is, square each element of d. Then form $3k$ groups of $16/\epsilon^2$ elements of d^2. Take the average of each group, getting $3k$ numbers. Finally, return the median of these $3k$ numbers.

2 Properties

The algorithm has several notable properties.

Intended use. The vectors may be feature vectors associated with documents. Many learning algorithms are based on Euclidean distance between feature vectors or dot products between feature vectors; this data structure is intended as a general-purpose tool, reusable by many learning algorithms.

Time and space savings. The algorithm saves space by storing just a sketch of each vector, rather than the vector itself. More typically, the original vector data *is* stored, but not readily accessible; this algorithm saves space in more accessible (and more expensive) memory, such as RAM as opposed to disk, where the original data might not be stored.

If the vectors have smaller representations, more of them can be stored in main memory. This saves time by reducing the number of times data is fetched from disk.

The sketch is a linear map. Given two vectors v_1 and v_2, one can compute the sketch of any linear combination of v_1 and v_2 by computing the corresponding linear combination of their sketches. This may be useful when combining feature vectors. For example, we may want to combine yesterday's feature vector with today's, using relative weights of 1:2.

Dot Products. A direct extension of this algorithm returns $\langle v_1, v_2 \rangle \pm \epsilon \|v_1\| \|v_2\|$ ¿from the same sketches for v_1 and v_2 as are used for distance approximation.

Use for non-sparse vectors. This algorithm is appropriate for large N, in which the vectors are not typically sparse. If the vectors are sparse, listing the non-zero items in each vector is a better strategy from perspectives of time and space. Henceforth we assume that the vectors are not sparse. (The vectors need not be dense. For example, $\sqrt{N}$ non-zero entries, for large N, might be indicate using this algorithm.)

Exploit opportunistic sparsity. Suppose some of the vectors are sparse and some are not. The sketch of vector with n non-zero components, represented sparsely, can be computed in time proportional to n instead of N.

Exploit substring and hierarchical structure. If subintervals of indexes to v_1 have useful semantics, then some variants of this algorithm can take advantage, as follows. The sketch of a vector $\chi_{[j,k]}$ with 1's in positions j to k and 0's elsewhere can be computed quickly, in time $\log^{O(1)}(k-j)$ instead of time $O(k-j)$. One can then approximate the dot product $\langle \chi_{[j,k]}, v_1 \rangle$ quickly from the (existing) sketch of v_1 and (quickly computed) sketch of $\chi_{[j,k]}$. This dot product sums the features in v_1 belonging to the semantically coherent group $[j, k]$.

3 Source

This algorithm is mostly as in [2, 1]. Other ideas relevant for variations are in the works listed in the bibliography.

References

[1] N. Alon, P. Gibbons, Y. Matias and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 1999.

[2] N. Alon, Y. Matias and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. In *ACM Symposium on Theory of Computing (STOC)*, pages 20–29, 1996.

[AS92] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley and Sons, New York, 1992.

[3] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, Small-Space Algorithms for Approximate Histogram Maintenance. In *STOC* (to appear), 2002.

[4] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. How to Summarize the Universe: Dynamic Maintenance of Quantiles In *Proceedings of VLDB Conference*, to appear, 2002.

[5] P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation. *FOCS*, 2000.

[6] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes.* North Holland Mathematical Library, Vol. 16, North Holland, New York, 1977.