Journal of Statistical Software

March 2011, Volume 39, Code Snippet 2. http://www.jstatsoft.org/

Fast Computation of Trimmed Means

Gleb Beliakov
Deakin University

Abstract

We present two methods of calculating trimmed means without sorting the data in $O(n)$ time. The existing method implemented in major statistical packages relies on sorting, which takes $O(n \log n)$ time. The proposed algorithm is based on the quickselect algorithm for calculating order statistics with $O(n)$ expected running time. It is an order of magnitude faster than the existing method for large data sets.

Keywords: trimmed mean, Winsorized mean, order statistics, sorting.

1. Introduction

Trimmed means are used very frequently in statistical sciences as robust estimates of location. Trimmed means are much less sensitive to the outliers compared to the arithmetic mean. They are also often used in image processing as filters. The α%-trimmed mean of $x \in \Re^n$ is the quantity

$$TM(x) = \frac{1}{n-2K} \sum_{i=K+1}^{n-K} x_{(i)}, \tag{1}$$

where $x_{(i)}$ are order statistics and $K = [\alpha n]$.

The current method of computation of a trimmed mean requires sorting the data, and its complexity is $O(n \log n)$. This method based on sorting is implemented in major statistical packages such as SAS, SPSS and R. For small to medium sized data sets the CPU time is negligible. However for very large data sets $n \geq 10^5$ the sort operation becomes a bottleneck. Data sets of this size or larger are now common (analysis of internet traffic and microarray data processing are two noteworthy examples).

In this note we draw attention to a little known fact that the sort operation can be avoided, by using a simple alternative formula based on computation of order statistics, which can be done in $O(n)$ expected time using the *quickselect* algorithm (Sedgewick 1988; Press, Teukolsky,

Vetterling, and Flannery 2002), or in $O(n)$ worst case time using the median of medians algorithm *BFPRT* (Blum, Floyd, Watt, Rive, and Tarjan 1973)[1].

We propose two different algorithms to calculate trimmed means. The gain in numerical performance is six to ten-fold as illustrated by our numerical experiments. Existing algorithms can be modified very easily, and the source code in C++ is provided.

2. Computation of trimmed and Winsorized means

The following quantity is known as K-Winsorized mean,

$$WM(x) = \frac{1}{n}\left(\sum_{i=K+1}^{n-K} x_{(i)} + K(x_{(K+1)} + x_{(K)})\right). \tag{2}$$

By comparing to (1), we can easily see that

$$TM(x) = \frac{1}{n-2K}\left(n\,WM(x) - K(x_{(K+1)} + x_{(K)})\right). \tag{3}$$

Suppose we have computed the order statistics $x_{(K+1)}$ and $x_{(n-K)}$ by using the quickselect algorithm in $O(n)$ time. Then we can compute the Winsorized mean without sorting, by using

$$WM(x) = \frac{1}{n}\left(\sum_{i=1}^{n} \min(\max(x_i, x_{(K+1)}), x_{(n-K)})\right). \tag{4}$$

The trimmed mean is then computed from WM also without sorting by (3).

For very large $n \geq 10^8$ caution should be exercised when using (4) because of the roundoff errors. The sum contains an extra $2K$ terms compared to (2) which may result in loss of precision. In our experiments, when we used a 4-byte float type (in C language) to calculate WM using (4), it lead to unacceptable loss of precision and resulted in values as twice as much as the actual trimmed mean for $n = 10^6$. Of course, the use of double or extended numbers easily solves this problem on CPUs, this is not the case when calculations are performed on Graphic Processing Units (GPU).

General purpose GPUs, such as NVIDIA's Tesla unit (NVIDIA Corporation 2008), allow one to offload routine numerical calculations to a GPU device, which possesses several hundreds of processor cores, and is capable of executing hundreds of thousands of threads in parallel. Many modern software packages are capable to execute parts of their code on GPUs. Summation is easily parallelized for GPUs by using a generic *reduce* operation using binary trees, see NVIDIA Corporation (2007). However the use of extended precision in reduction on GPUs bears a cost, as a working array of extended precision numbers of size n has to be used. Below we present an alternative method, which does not use the extra terms in the sum (4).

Note that it would be incorrect to compute the trimmed mean similarly to (4) by replacing the terms outside the range $[x_{(K+1)}, x_{(n-K)}]$ with zeroes. In that case the resulting expression is not a trimmed mean, and is in fact a discontinuous function of the components of x.

[1] We used an implementation of *BFPRT* from `http://www.moonflare.com/code/select/index.php`, and the tests show that the simpler *quickselect* with $O(n)$ average case complexity outperforms *BFPRT* by a factor of four (for $n = 3\,200\,000$).

Instead, let us compute the expression

$$S(x) = \sum_{i=K+1}^{n-K} x_{(i)} \tag{5}$$

in the following way

$$S(x) = \sum_{i=1}^{n} w(x_i; x) x_i, \tag{6}$$

with

$$w(t; x) = \begin{cases} 1 & \text{if } x_{(K+1)} < t < x_{(n-K)}, \\ \frac{a}{b} & \text{if } t = x_{(K+1)}, \\ \frac{c}{d} & \text{if } t = x_{(n-K)}, \\ 0 & \text{otherwise,} \end{cases}$$

where the integers a, b, c, d are computed as described below. Note that if no elements of x coincide, there are no issues with computation and we let $a = b = c = d = 1$. A problem occurs when some elements coincide. The multiplicity of the k-th order statistic is the number of components of x equal to $x_{(k)}$.

Consider the numbers $b_- = count(x_i < x_{(K+1)})$ and $b = count(x_i = x_{(K+1)})$, and similarly, $d_- = count(x_i < x_{(n-K)})$ and $d = count(x_i = x_{(n-K)})$. Then $K + 1 = b_- + p$ and $n - K = d_- + q$ for some $p \leq K + 1$, $q \leq n - K$. Now if we take $a = b - p + 1 = b - K + b_-$ and $c = q = n - K - d_-$, we notice that the expressions (5) and (6) coincide.

The calculation of the numbers a, b, c, d above is done in $O(n)$ operations when $x_{(K+1)}$ and $x_{(n-K)}$ are known. Therefore the overall complexity of evaluating S in (6) is $O(n)$.

Compared to the formula (3), there are less roundoff errors at an additional marginal cost. In calculations we performed when using 4-byte floats, the loss of precision was ten times smaller than in (3)–(4).

3. Numerical comparison

Let us now illustrate the advantages of the formulas based on (3) and (6) numerically. We considered several data sets, generated as follows.

1. Uniform $x_i \sim U(0, 1)$;
2. Normal $x_i \sim N(0, 1)$;
3. Half-normal $x_i = |y_i|$ and $y_i \sim N(0, 1)$;
4. Beta $x_i \sim \beta(2, 5)$;
5. Mixture 1, 80% of elements of x_i chosen from $N(0, 1)$ and 20% from $N(100, 1)$;
6. Mixture 2, 80% of elements of x_i chosen from half-normal $N(0, 1)$ and 20% from $N(100, 1)$;
7. Mixture 3, 50% of elements of x_i chosen from $N(0, 1)$ and 50% from $N(100, 1)$.

n	With sort	Without sort by (6)	Without sort by (3)	Speedup
100 000	0.016	< 0.001	< 0.001	10
1 000 000	0.172	0.022	0.015	11.4
2 000 000	0.359	0.063	0.031	11.5
4 000 000	0.766	0.109	0.078	9.8
8 000 000	1.563	0.235	0.188	8.6
16 000 000	3.235	0.501	0.344	9.5
32 000 000	6.516	0.859	0.56	11.6

Table 1: Average cost of computation of TM (averaged over 7 data sets with different distributions and over 10 instances of each data set) in seconds.

Calculations were performed on a Pentium IV 2.1 GHz workstation with 1 GB RAM. The results were very similar for all different data sets and are averaged in Table 1.

In a separate series of experiments, we compared the CPU time when calculating multiple trimmed means of smaller data sets $n < 100$ by formulas (6) and (3), and by using the traditional sort operation. Here all methods have performed equally and there was no speedup.

4. Conclusion

We described two alternative algorithms to calculate trimmed means without using the sort operation. The speedup is an order of magnitude on very large data sets $n \geq 10^5$. The algorithm based on Winsorized mean is faster, but requires care dealing with roundoff errors. The proposed algorithms are easy to implement and incorporate into existing statistical packages.

References

Blum M, Floyd RW, Watt V, Rive RL, Tarjan RE (1973). "Time Bounds for Selection." *Journal of Computer and System Sciences*, **7**, 448–461.

NVIDIA Corporation (2007). "NVIDIA CUDA SDK – Data-Parallel Algorithms." Accessed 2010-06-01, URL http://developer.download.nvidia.com/compute/cuda/1_1/Website/Data-Parallel_Algorithms.html.

NVIDIA Corporation (2008). "NVIDIA Tesla Personal Supercomputer – Tesla Datasheet." Accessed 2010-12-01, URL http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_PSC_US_Nov08_LowRes.pdf.

Press AH, Teukolsky SA, Vetterling WT, Flannery BP (2002). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York.

Sedgewick R (1988). *Algorithms*. 2nd edition. Addison-Wesley, Reading, MA.

A. Source code

C++ source code for trimmed means based on (6). Note that the arrays in C are indexed from 0. The code below requires the `quickselect` procedure available from Press *et al.* (2002). n is the length of the array x and $K = [\alpha n]$.

```
float Weighted(float x, float t1, float t2, float w1, float w2)
{
    if(x<t2 && x>t1) return x;
    if(x<t1) return 0;
    if(x>t2) return 0;
    if(x==t1) return w1*x;
    return w2*x; // if(x==t2)
}

double TrimmedSum(float x[], int n, int K)
{
    float w1, w2, OS1, OS2;
        double t=0;
        OS1=quickselect(x, n, K);
        OS2=quickselect(x, n, n-K-1);

// compute weights
    double a, b=0, c, d=0, dm=0, bm=0, r;

    for(int i=0; i<n; i++){
        r = x[i];
        if(r < OS1) bm += 1;
        else if(r == OS1) b += 1;
        if(r < OS2) dm += 1;
        else if(r == OS2) d += 1;
    }

    a = b + bm - K;
    c = n - K - dm;
    w1 = a/b;
    w2 = c/d;

    for(int i=0; i<n; i++)
        t += Weighted(x[i], OS1, OS2, w1, w2);
    return t;
}

double TrimmedMean(float x[],  int n, int K)
{
    return TrimmedSum(x,n,K)/(n-2*K);
}
```

Affiliation:

Gleb Beliakov
School of Information Technology
Deakin University
221 Burwood Hwy, Burwood 3125, Australia
Telephone: +61/3/92517475
Fax: +61/3/92517604
E-mail: gleb@deakin.edu.au

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 39, Code Snippet 2 *Submitted:* 2010-06-07
March 2011 *Accepted:* 2010-10-28