BIOINFORMATICS APPLICATIONS NOTE Vol. 26 no. 22 2010, pages 2897–2899
doi:10.1093/bioinformatics/btq540

Genome analysis

Advance Access publication September 21, 2010

FastPval: a fast and memory efficient program to calculate very low *P*-values from empirical distribution

Mulin Jun Li[1], Pak Chung Sham[2] and Junwen Wang[1,*]

[1]Department of Biochemistry and [2]Department of Psychiatry and State Key Laboratory of Cognitive and Brain Sciences, LKS Faculty of Medicine, The University of Hong Kong, 21 Sassoon Rd, Pokfulam, Hong Kong SAR, China

Associate Editor: Alfonso Valencia

ABSTRACT

Motivation: Resampling methods, such as permutation and bootstrap, have been widely used to generate an empirical distribution for assessing the statistical significance of a measurement. However, to obtain a very low *P*-value, a large size of resampling is required, where computing speed, memory and storage consumption become bottlenecks, and sometimes become impossible, even on a computer cluster.

Results: We have developed a multiple stage *P*-value calculating program called FastPval that can efficiently calculate very low (up to 10^{-9}) *P*-values from a large number of resampled measurements. With only two input files and a few parameter settings from the users, the program can compute *P*-values from empirical distribution very efficiently, even on a personal computer. When tested on the order of 10^9 resampled data, our method only uses 52.94% the time used by the conventional method, implemented by standard quicksort and binary search algorithms, and consumes only 0.11% of the memory and storage. Furthermore, our method can be applied to extra large datasets that the conventional method fails to calculate. The accuracy of the method was tested on data generated from Normal, Poison and Gumbel distributions and was found to be no different from the exact ranking approach.

Availability: The FastPval executable file, the java GUI and source code, and the java web start server with example data and introduction, are available at http://wanglab.hku.hk/pvalue

Contact: junwen@hku.hk

Supplementary information: Supplementary data are available at *Bioinformatics* online and http://wanglab.hku.hk/pvalue/.

Received on August 2, 2010; revised on September 7, 2010; accepted on September 16, 2010

1 INTRODUCTION

Permutation and bootstrap are resampling procedures to assess the statistical significance of a measurement. They are non-parametric statistical tests that can convert a measurement (score) into an empirical *P*-value, even when the distribution of the measurements is unknown. Since resampling does not assume known distribution of the data, and biological data are usually complex, it has been widely used in the bioinformatics field, such as transcription factor binding site searching, pathway analysis and genome-wide association studies.

Finding transcription factor binding sites (TFBSs) in the promoter region of a gene is important to understand gene regulation (Zhang *et al*., 2007). TFBS of a particular transcription factor are usually represented by a computational model known as the position weight matrix (PWM) (Pape *et al*., 2008). To search for a putative binding site, we use the PWM to score DNA sequences with a sliding window approach. For each window, we obtain a score. By comparing this score with the distribution of the scores from the background, we can obtain the statistical significance (empirical *P*-value) of this score. The empirical background score distribution is obtained by scoring a large set of random sequences from the intergenic regions in the genome with the same PWM. We then sort the background scores and save them for later usage. When we convert a new score into a *P*-value, we load the background into the memory and search for the score. The ranking of the score is then converted to a *P*-value (Hannenhalli, 2008).

This empirical approach of calculating *P*-values is very powerful since it does not assume any distribution of the data. However, the significance of the *P*-value is limited by the size of the background we sample. To obtain a very low *P*-value, we have to sample a very large dataset from the background. The large dataset causes three limitations: (i) sorting and searching in a large dataset are time consuming; (ii) storage of the sorted background scores requires a large amount of hard disk space; and (iii) processing of the sorted array requires a great deal of memory, which is not usually feasible on a personal computer.

Efficient methods have been developed to relieve the computational burden resulting from large-scale resampling. For example, Jensen *et al*. (2009) developed a Bayesian approach to dynamically assign resamples for multiple testing problems. For microarray expression data, they assume that each gene has a different null distribution, and allocate more resamples to the genes with *P*-values close to the classification threshold. But for the *P*-values that are far lower or far higher than the threshold and the decisions that are easy to make, they allocate much fewer resamples than the traditional method. The dynamic resampling allocation strategy has improved the computing efficiency, particularly when the number of tests is large.

While the above mentioned method deals with the efficiency of multiple tests, assuming each test has a different null distribution, *P*-value calculation from resampling based on a single test, or multiple tests assuming the same null distribution, is still hampered by computing, memory and storage limitations.

We have developed an efficient program to calculate the empirical *P*-value for a single test, or multiple tests assuming the same null

*To whom correspondence should be addressed.

Table 1. Comparison of FastPval and Exact method in memory, storage consumptions and running time

Resampling size (1 000 000)	Memory(MB)		Storage (MB)		Running time (s)			
					Model building		*P*-value searching[a]	
	Exact	FastPval	Exact	FastPval[b]	Exact	FastPval	Exact	FastPval
1	4	0.39	12.1	1.3 + 0.013	1.10	1.05	0.74 + 2.33	0.08 + 1.53
10	38	0.39	121.4	1.3 + 0.131	11.21	9.29	7.61 + 29.88	0.09 + 16.07
100	373	0.78	1200	1.3 + 1.3	116.73	91.46	77.13 + 332.13	0.14 + 249.44
500	1900	2	5900	1.3 + 6.0	677.58	455.23	380.47 + 1885.12	0.40 + 1297.44
1000	3700	4	11 900	1.3 + 11.9	1409.61	919.45	761.52 + 4019.77	0.72 + 2530.65
5000	N/A[c]	19	59 900	1.3 + 59.8	N/A[c]	5475.32	N/A[c]	3.34 + 12885.87

[a]Model loading + searching time.
[b]Sizes of first model + second model.
[c]Exact method failed to load due to large size of the dataset.

distribution. This program separates the background distribution into multiple parts, according to user specified cutoffs. The scores in the less significant part are highly condensed into one table and the *P*-values are calculated less exactly, while the scores in the more significant part are put into other tables and the *P*-values are calculated more accurately. Our experiments showed that this algorithm is more time efficient, and uses much less storage and memory. It can be used widely in resampling based *P*-value calculation, either as standalone software or as a plug-in module.

2 METHODS

For simplicity, here we illustrate our method in a two-stage approach and use the right tail of the distribution to calculate the statistics. In the first stage, we randomly sample a subset N from the original large dataset O. N is usually less than one-hundredth of the size of O, thus saving processing time. We sort N and obtain a cutoff score S_c representing the top P portion of N. Both N and P are parameters specified by the users, and are set to $N = 100\,000$ and $P = 0.001$ by the default. We then scan the original set and put scores greater than S_c into our second subset M, and we obtain the maximum score S_m in M. The two subsets N and M are sorted, saved, and serve as our two models ($M1$ and $M2$). To calculate the *P*-value for a new score S, we compare S with S_c. If $S \leq S_c$, we will find its *P*-value in $M1$. Otherwise we use $M2$. If $S > S_m$, indicating S is out of our resampling score range, we use theoretical distribution to calculate its *P*-value or simply set the *P*-value to 0, at the user's preference. The parameters of two theoretical distributions, normal and extreme value distributions, were obtained from dataset N.

To evaluate the performance of our method, we compared FastPval with the traditional approach (named Exact) on a linux machine (Intel Xeon CPU E5410 2.33 GHz; 16 G of memory, SuSE linux 10.1). In the 'Exact' approach, we used quicksort and binary search in c programming. FastPval used the same sorting and searching algorithms for $M1$ and $M2$.

To evaluate the accuracy of FastPval, we compared the calculated *P*-values with the original *P*-values in three different distributions: Normal, Poisson and Gumbel. For each distribution, we tested the *P*-values in the $-\log_{10}$(*P*-value) ranged from 0 to 9 (corresponding *P*-value range from 1 to 10^{-9}, exclusively). We took 162 *P*-values (termed theoretical *P*-values) evenly distributed in each of the nine ranges (Supplementary Table S1). We converted these *P*-values into scores with the build-in functions in the R platform, using parameters for each distribution as shown in Supplementary Table S2. Under the same parameter setting, we randomly sampled one billion data points. We used FastPval to build models $M1$ and $M2$ from these data. Finally, we used FastPval to convert these 162 scores into *P*-values (termed FastPval *P*-values) with the models. The $-\log_{10}$ (FastPval *P*-values) were plotted against the $-\log_{10}$ (theoretical *P*-values) on a Log–Log QQ plot. The Kolmogorov–Smirnov test was also used to compare the FastPval *P*-values with theoretical *P*-values.

3 RESULTS AND DISCUSSION

As shown in Table 1, FastPval shows significant improvement over the 'Exact' approach. Tested on 1 billion resampling data, FastPval only used 0.11% of the memory and storage and 52.94% of model building and searching times. When we increased the dataset size to 5 billion, the 'Exact' method failed to load due to the large memory requirement, while our method was able to calculate *P*-values successfully. FastPval has speed, memory and storage consumption approximately linear to resembling size. The accuracies of FastPval calculated *P*-values from three different distributions were compared with the theoretical *P*-values. The results are shown in the form of Log–Log QQ plots (Supplementary Fig. S1a–c). In all three tested distributions, the FastPval calculated *P*-values and the theoretical *P*-values are highly matched, forming a 45 degree line in the Log–Log QQ plots. The Kolmogorov–Smirnov tests showed the *P*-value = 1.000 for Normal and Poisson distributions, and 0.998 for Gumbel distribution, indicating that the calculated *P*-values did not deviate from the original distribution. We therefore conclude that FastPval is accurate for calculating *P*-values for data from a variety of distributions.

The Java GUI interface of FastPval is shown in Supplementary Figure S2a–c. In the 'Method' field, the user can either choose 'FastPval' or the traditional 'Exact' method to calculate *P*-values. When the resembling size is greater than 10 million, FastPval is the only suitable choice. In the 'Step' field, the user can either 'Generate model', or 'Calculate *P*-value' or 'Do both'. The 'Generate model' step allows the user to generate $M1$ and $M2$ models from the background dataset O; the 'Calculate *P*-value' step allows the user to calculate *P*-values for all the scores saved in a text file. The interface changes according to the user's selection. In the 'Generate model' step, the user has to specify the background file and directory to save two models, by clicking on 'Background file' and 'Output folder', respectively. The user also needs to select two parameters, the 'Sampling size' (N) and '*P*-value cutoff' (P). The selections of P and N are affected by the balance of accuracy and speed. Bigger Ns and Ps will give more accurate *P*-values but will be less efficient. We

recommend $N = 100\,000$ and $P = 0.001$. N^*P should be within the range of 10 to 1000, preferably 100. After the models are generated, the program will automatically change to the 'Calculate *P*-value' step, with default model files selected. The user will need to specify the file with scores for *P*-value calculation, by clicking on 'Sample file'. For scores that are out of the boundary of both models, the user can choose either 'Extreme distribution' or 'Normal distribution' in the 'Assumed distribution' field to calculate the *P*-value, or simply select 'No distribution' to assign the *P*-value to 0. The parameters of both distributions were calculated from the fitting of the dataset *N*.

The program can be run in a command line mode, which is suitable for large-scale batch processing. We provide both 32-bit and 64-bit executable GUI programs, in both linux and windows platforms. The source code of the program is provided in our companion website.

Funding: Internal funds from the CRCG and the Genomic SRT of the University of Hong Kong; GRF 778609M and AoE M-04/04 from the Research Grants Council of Hong Kong.

Conflict of Interest: none declared.

REFERENCES

Hannenhalli,S. (2008) Eukaryotic transcription factor binding sites—modeling and integrative search methods. *Bioinformatics*, **24**, 1325–1331.

Jensen,S.T. *et al*. (2009) A Bayesian approach to efficient differential allocation for resampling-based significance testing. *BMC Bioinformatics*, **10**, 198.

Pape,U.J. *et al*. (2008) Natural similarity measures between position frequency matrices with an application to clustering. *Bioinformatics*, **24**, 350–357.

Zhang,J. *et al*. (2007) Computing exact P-values for DNA motifs. *Bioinformatics*, **23**, 531–537.