BIOINFORMATICS APPLICATIONS NOTE Vol. 22 no. 17 2006, pages 2158–2159
doi:10.1093/bioinformatics/btl357

Sequence analysis

Branch and bound computation of exact *p*-values

Gill Bejerano
Center for Biomolecular Science and Engineering, School of Engineering, 1156 High Street, University of California, Santa Cruz, CA 95064, USA

Received on December 27, 2006; revised on June 23, 2006; accepted on June 24, 2006
Advance Access publication August 7, 2006
Associate Editor: Alex Bateman

ABSTRACT

Summary: P-value computation is often used in bioinformatics to quantify the surprise, or significance, associated with a given observation. An implementation is provided that computes the exact *p*-value associated with any observed sample, against a null multinomial distribution, using the likelihood-ratio statistic. The efficient branch and bound code, far exceeding the full enumeration implemented by commercial packages, is especially useful with small sample, sparse data and rare events, common scenarios in bioinformatics, where approximations are often inaccurate and inappropriate. This code base can also be adapted to compute exact *p*-values of other statistics in diverse sampling scenarios.

Availability: Freely available at http://www.soe.ucsc.edu/~jill/src/

Contact: jill@soe.ucsc.edujill

INTRODUCTION

Bioinformatics is largely an exploratory science. A routine question that comes up over and over again in the context of data exploration is how surprising, or significant, an observation we make about some given data really is. One way to formalize this question is to describe the data using a probabilistic model Q. For a given sample T we ask how unexpected T is, assuming it was drawn from the probabilistic model Q. To quantify the answer, we can define a test statistic D, which measures how dissimilar any given sample T is from our expectation under the model Q. Formally, $D : T \rightarrow R$ is a function from every possible sample outcome T to a non-negative real number. The larger $D(T)$ the more different the observed sample T is from a sample we expect to obtain from Q. In this scenario, the p-value of a sample T is the chance probability of seeing samples as surprising as T under the null model Q. When the sample space is finite, this amounts to the following:

$$p\text{-value} = \sum_{\substack{T' s.t. \\ D(T') \geq D(T)}} P_Q(T'), \tag{1}$$

where $P_Q(T)$ is the probability to draw sample T from model Q. While desirable, in practice the sample space over which this sum has to be performed is often too large. Other times the actual terms we want to sum are close to machine accuracy, or much smaller than the already accumulated partial sum, resulting in loss of accuracy of the obtained result.

With the advent of ever growing computational power, much algorithmic and numerical research evolved to allow an accurate estimation of exact p-values [surveyed in Agresti (2001)]. In Bejerano *et al*. (2004) we recently introduced a generic branch and bound approach to efficiently compute Equation (1). Here I provide an implementation that successfully applies this methodology to compute a specific p-value more rapidly, more accurately, and over a broader range of cases than is possible using the explicit summation of Equation (1), currently implemented by commercial packages.

METHODS

The provided implementation computes the exact p-value of the following scenario: $Q = (q_1, q_2, \ldots, q_{n_k})$ is a multinomial distribution over a finite set of k possible outcomes. $T = (n_1, n_2, \ldots, n_k)$ is a sample of size $n = \sum_{i=1}^{k} n_i$ over the k categories. In this sample, the i-th category appears n_i times. The statistic D used to measure how dissimilar sample T is from the expected sample governed by Q is the likelihood ratio statistic:

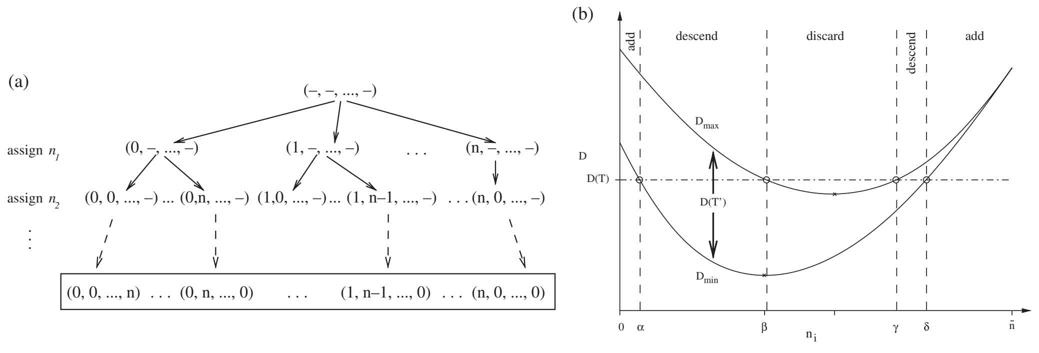
$$D(T) = 2 \sum_{i=1}^{k} n_i \log \frac{n}{nq_i}$$

This sum is zero iff $\forall i,\ n_i = nq_i$, which is the most probable result (up to fractions) when drawing a sample according to distribution Q. In all other cases this sum is positive, growing larger the more unlikely T is according to Q.

In order to span the search space of all possible samples of size n, a standard recursive procedure is used to unfold the assignment tree of Figure 1a. The set of all $\binom{n+k-1}{n}$ possible assignments makes up the leaves of this tree. Internal nodes of the tree hold partial assignments of the form $\tau = (n_1, \ldots, n_{i-1}, —, \ldots, —)$, where $n_1, \ldots, n_{i-1}$ have already been assigned, while categories i through k await assignment of the remainder $\bar{n} = n - \sum_{j=1}^{i-1} \mathrm{n}_j$ counts. Let $[\tau]$ denote all the leaves in the subtree rooted in the node labeled τ. Equivalently, this is the set of all valid assignments that can be completed from τ. The key to performing branch and bound computation on this tree lies in defining two functions, $D_{\min}, D_{\max} : \tau \rightarrow R$ such that

$$\forall \tau, T \in [\tau]\ D_{\min}(\tau) \leq D(T) \leq D_{\max}(\tau).$$

In Bejerano *et al*. (2004), Lemma 1 we provide two such closed formulas, and show that as a function of assigning the next category n_i with a count between 0 and $\bar{n}$, they obtain the shapes shown in Figure 1b. We can now take advantage of this knowledge by intersecting the two curves with the threshold $D(T)$ from Equation (1). This divides the range $\{0, 1, \ldots, \bar{n}\}$ into up to five distinct regions. We find these regions efficiently using binary searches, and then proceed to treat each region appropriately—the recursion descends only into nodes where $D(T')$ may be either above or below $D(T)$; we discard internal nodes where $D(T')$ will invariably be smaller than $D(T)$; and we sum up the probability of the appropriate subtrees where $D(T')$ will invariably be at least as large as $D(T)$. By pruning or summing internal nodes we drastically reduce the search space actually traversed. By computing the


(a)
(–, –, ..., –)
assign n_1
(0, –, ..., –)
(1, –, ..., –)
...
(n, –, ..., –)
assign n_2
(0, 0, ..., –) ... (0,n, ..., –) (1,0, ..., –)... (1, n–1, ..., –) . . . (n, 0, ..., –)
(0, 0, ..., n) . . . (0, n, ..., 0) . . . (1, n–1, ..., 0) . . . (n, 0, ..., 0)
(b)
add
descend
discard
descend
add
D
D(T)
D_{max}
D(T')
D_{min}
0
α
β
γ
δ
$\bar{n}$
n_i

(c)

column	exact p-val	χ^2 error
1,3,5,13	0.354	+33.98%
2	0.481	+21.26%
4	2.56e-4	-24.90%
6,9	0.136	+48.17%
7,8,10	1.53e-5	-3.49%
11	0.596	+5.34%
12	0.654	-9.36%
14	0.027	+18.85%

Fig. 1. Branch and bound exact p-value computation. (**a**) To compute the p-value of a sample T, using statistic D, according to Equation (1), we first decompose the sample space using partial assignments. The implementation recursively makes all allowed assignments of remaining counts to categories 1 through k. (**b**) At each internal node, corresponding to a partial assignment, the lower and upper bounds on the likelihood statistic $D_{\min}$, $D_{\max}$ are both convex as a function of assigning to the next category n_i between 0 and the maximal allowed value of all remaining counts, $\bar{n}$. Binary searches are used to find where these curves intersect the $D(T)$ threshold. Subtrees entirely above the threshold are added as a whole to the cumulative sum, subtrees that are completely below it are discarded, and ambiguous nodes are descended to facilitate further refinement. Note that different configurations of $D_{\min}$, $D_{\max}$ and $D(T)$ induce a smaller number of partitions. (**c**) To illustrate the difference between the exact p-value and its asymptotic approximation in a bioinformatic context, both are computed in the context of the p-value logo (Rahmann, 2003) of an arbitrary Transfac 8.3 matrix (M00044). [a,b modified from Bejerano *et al*., 2004]

probability of an entire subtree $P_Q(\tau)$ directly, without enumerating all assignments $T \in [\tau]$ we also increase the accuracy of the computation by avoiding small quotients and imbalanced summations. In Bejerano *et al.* (2004) we demonstrate the speed-up and accuracy gain obtained by this approach, and show that it is particularly efficient in small sample, sparse null hypotheses and rare events, all common scenarios in bioinformatics.

DESCRIPTION

The tarball contains the C++ source code, and a readme file that describes how to configure, compile and run it. When invoked the user is asked to enter the number of categories k, the null multinomial distribution Q and an observed sample T from which n and $D(T)$ are computed. Alternatively the user can input n, $D(T)$ directly. The user can then choose to compute the p-value in one of five ways:

- Asymptotic χ^2 approximation.
- Full enumeration of Equation (1).
- Recursive branch and bound computation.
- Branch and bound coupled with binary search speed up.
- Monte Carlo simulation.

When performing an exact computation the user is asked to specify a permutation π over the k categories, such that when assigning them, instead of assigning n_1 first, then n_2, etc. as in Figure 1a, one first assigns $n_{\pi(1)}$, then $n_{\pi(2)}$, etc. Changing the assignment order changes how the sample space is unfolded, which for the branch and bound alternatives means that different subtrees are examined and resolved for different permutations. When coupled with the binary searches, it appears empirically that the best assignment order is the one that orders the q_i's in ascending order. In this case, the implementation recommends this permutation but does not enforce it. While greatly reducing the actual search space, runtime complexity does remain exponential in k (Bejerano *et al*., 2004) Thus, exact computation is mostly adequate for small values of k and small to moderate values of n.

Two additional speed-ups that may have an adverse affect on accuracy are offered as configuration options (see the readme file): replacing actual log/exp operations with a look-up table of selected values, and linear interpolation between them; and allowing, instead of adding subtrees in Figure 1b, to add the probability of the father node, and subtract from it the complementary subset of subtrees, when this results in performing less arithmetic operations overall. The resulting p-value, as well as additional parameters of the computation, are outputted both in the form of a table row, and as more detailed untabulated free text (sent to cout and cerr, respectively).

As discussed in (Bejerano *et al*., 2004), and illustrated in Figure 1c, the implementation is valuable in computing the exact likelihood-ratio p-value in different bioinformatic scenarios. The code base serves as an ideal starting point to expand into other test statistics, such as Pearson's X^2, and all other statistics of the Cressie-Read family (Read and Cressie, 1988). It is also readily extensible to many other hypothesis tests where the sample space can be traversed in a similar fashion.

ACKNOWLEDGEMENTS

G.B. performed parts of this work at the Hebrew university, where he was supported by a grant from the ministry of science, Israel.

Conflict of Interest. none declared.

REFERENCES

Agresti,A. (2001) Exact inference for categorical data: recent advances and continuing controversies. *Stat. Med.*, **20**, 2709–2722.

Bejerano,G. *et al.* (2004) Efficient exact p-value computation for small sample, sparse, and surprising categorical data. *J. Comput. Biol.*, **11**, 867–886.

Rahmann,S. (2003) Dynamic programming algorithms for two statistical problems in computational biology. *Lecture Notes Comp. Sci.*, **2812**, 151–164.

Read,T.R.C. and Cressie,N.A.C. (1988) *Goodness-of-Fit Statistics for Discrete Multivariate Data*. Springer-Verlag, New York.