Algorithm for exact evaluation of bivariate two-sample Kolmogorov-Smirnov statistics in O($n$log$n$) time.

Krystian Zawistowski

January 3, 2019

Abstract

We propose an O($n$log$n$) algorithm for evaluation of bivariate Kolmogorov-Smirnov statistics for n samples which. It offers few orders of magnitude of speedup over existing implementations for $n > 10^5$ samples of input. Algorithm is based on static Binary Search Trees and sweep algorithm. We share tested C++ implementation with Python bindings.

1 Introduction

1.1 Background

Kolmogorov test introduced by [1] is notable statistical procedure used for testing for difference between univariate probability distributions.

A generalization of Kolmogorov statistics to bivariate distributions was proposed by [2] with $O(n^2)$ algorithm for evaluating it for empirical distributions with N samples. Procedure found numerous application in empirical research as many references to [2] indicate. C implementation of algorithm itself can be found in popular computing textbook [3]. Square time complexity however limits application of the procedure to small numbers of samples, around 10^4 for a immediate feedback loop on a typical PC. We present $O(n\log n)$ algorithm able to process immediately millions of samples. We hope that presented methodology allow to better leverage the procedure for research work as well as modern big data applications.

1.2 Bivariate Kolmogorov statistics

Consider random variables X_1, Y_1, X_2 and Y_2. In accordance to [2] two-dimmensional two-sample Kolmogorov-Smirnov test statistics is defined as follows:

$$D = \sup_{x,y} \|\Pr(X_1 < x, Y_1 < y) - \Pr(X_2 < x, Y_2 < y)\| \tag{1}$$

One can evaluate D for empirical distribution function of bivariate data samples $A_1 = \{(x_1, y_1)....(x_n, y_n)\}, A_2 = \{(x'_1, y'_1)....(x'_{n'}, y'_{n'})\}$:

$$F_i(u, v) = \frac{1}{\|A_i\|}\|\{(x, y) \in A_i : x < u, y < v\}\| \tag{2}$$

$$D_{12} = \sup_{x,y} \|F_1(x,y) - F_2(x,y)\| \tag{3}$$

$F_1(c,y)$ for any c is constant everywhere except at $y \in \{y_1...y_n\}$, similiar argument holds for $F_1(x,c)$. Based on this key observation [] shows an O($(n+n')^2$) algorithm for finding exact global maximum of (3). We reproduce it here as it is important to our idea:

Lemma 1.1. *To find D_{12} for two bivariate real samples $A_1 = \{(x_1,y_1)....(x_n,y_n)\}, A_2 = \{(x'_1,y'_1)....(x'_{n'},y'_{n'})\}$ it is sufficient to evaluate $\|F_1(x,y) - F_2(x,y)\|$ for all $c \in C$, $C = C_1 \times C_2, C_1 = \{x_1...x_n, x'_1...x'_{n'}\}, C_2 = \{y_1...y_n, y'_1...y'_{n'}\}$.*

Proof. Let $D(x,y) = \|F_1(x,y) - F_2(x,y)\|$ Let $(u,v) = \arg\max_{x,y} D(x,y)$, case of $(u,v) \in C$ is trivial, so we assume otherwise. let $x_{up} = \arg\max_x\{x \in C_1 : x < u\}$ and $y_{up} = \mathrm{argmax}_y\{y \in C_2 : y < v\}$. By the definition $(x_{up}, y_{up}) \in C$ and by the fact that C is a product set we have:

$$\nexists_{(x,y)\in C}(x_{up} < x < u) \vee (y_{up} < y < u).$$

Since $A_1 \subset C$ and $A_2 \subset C$ (2) implies that $D(u,v) = D(x_{up}, y_{up})$. Thus $\sup_{(x,y)\in C} D(x,y) = D_{12}$ □

2 Evaluating extrema of expanding sum with BST.

2.1 Binary search trees.

A binary search tree (BST) [4] is a data structure comprised of nodes satisfying following:

- Each node contains a single key and references to up to two other nodes. We call them parent node and child nodes respectively.
- A left (right) child is a child node with key smaller (greater) than parent's key. Each node can have at most one left child and at most one right child.
- For non empty BST there is exactly one node with no parent, we call it **root**.

A **path** is a series of nodes $n_1...n_k$ s. t. for any i n_i is a parent of n_{i+1} and n_k is a **leaf** i.e. a node without children. A **height** of a tree h is the length of longest path in a tree. If for nodes n_i, n_j exist a path that contains both of them, and n_i precedes n_j, we call n_i ancestor to n_j and n_j a descendat to n_i. Each ancestor with all it's descendants forms another BST we call **subtree**.

Lookup, insertion and deletion in BST can be done in $O(h)$ operations by straightforward top-down transversal. One can put any N elements in a tree with $h \leq 1 + \log N$, and algorithms exists to perform insertion or deletion in $O(h)$ time while making sure that we keep $h \leq 2\log N$. Those are of particular interest in practical application.

2.2 Keeping track of expanding sum extrema

Consider keys $k_i \in K \subset \mathbb{R}$, and values v_i, $i \in \{1, 2...n\}$ associated with keys. For a finite ordered set A with $m = \inf A$ are interested in following quantities:

$$U_A = \sup_{i \in A} \sum_{j=m}^{i} v_j, D_A = \inf_{i \in A} \sum_{j=m}^{i} v_j \tag{4}$$

Our goal here to be able to change values v_i while keeping track of U, D.

Consider balanced BST s.t. node n_i has key k_i and keeps value v_i. Let S_i be set of indices of descendants of node n_i and $S'_i = S_i \cup \{n_i\}$. Let $r(i)$ be right child index (if exists) and $l(i)$ left child i (if exists). Additionally each node keeps track of $s_i = v_i + \sum_{j \in S_i} v_j$ - sum of values in a node and its descendats. Additionally we define u_i, d_i as follows:

- if n_i is a leaf:
$$u_i = d_i = v_i \tag{5}$$

- otherwise if $l(i)$ and $r(i)$ exist:
$$\begin{aligned} u_i &= \max\{s_{l(i)} + v_i, u_{l(i)}, s_{l(i)} + v_i + u_{r(i)}\}, \\ d_i &= \min\{s_{l(i)} + v_i, d_{l(i)}, s_{l(i)} + v_i + d_{(i)}\}; \end{aligned} \tag{6}$$

- otherwise if only $l(i)$ exists:
$$u_i = \max\{s_{l(i)} + v_i, u_{l(i)}\}, d_i = \min\{s_{l(i)} + v_i, d_{l(i)}\}; \tag{7}$$

- otherwise if only $r(i)$ exists:
$$u_i = \max\{v_i, v_i + u_{r(i)}\}, d_i = \min\{v_i, v_i + d_{(i)}\}. \tag{8}$$

Lemma 2.1. *For each i $U_{S'(i)} = u_i$ and $D_{S'(i)} = d_i$, i.e. u_i is maximum and d_i is minimum of expanding sum of values in a subtree S'_i*

Proof. For n_i being single leaf this follows straight from the definition (5). Now consider a node n_i s.t. it has left and right child and we know: maxima and minima of expanding sum, and total sum of values in left and right child. To find $U_{S'(i)}$ we consider three cases:

- Maximum occurs at $j > i$, i.e. in right subtree. All values of n_i expanding sum in the right subtree are sum of $s_{l(i)}$, v_i and respective values of $n_{r(i)}$ expanding sum.
- Maximum occurs at $j = i$. Expanding sum value at j is $s_{l(i)} + v_i$.
- Maximum occurs at $j < i$, then $U_(S'(i)) = U_(S'(l(i))$, and we assume to know r.h.s.

By evaluating these cases and picking greatest value we reproduce formulae (6), (7) and (8). Identical argument applies to minimum $D_{S'(i)}$. Application of induction in the upwards direction (from leaves to consecutive ancestors) finishes the proof. □

Using Lemma 2.1 allows us to define a data structure we will call **expanding sum extrema tree (ESET)**, that for a given series of pairs of keys k_i and values v_i keeps track of maximum and minimum of cumulative sum of values (keys are used for labelling and ordering). It is based on static binary search tree ordered by k_i and with values v_i added to nodes. If a key is repeated then two or more pairs share a node, and its value is a sum of all key values. Moreover it is supposed to have following properties:

- It can be constructed for N pairs in $O(N\log N)$ time.
- Any key-value pair can be removed in $O(\log N)$ time.

First property can be done by construction of balanced static binary search tree and then evaluation of u_i and d_i in bottom up manner.Second is allowed by Lemma 2.1 - as we remove value we just subtract it from node value and tree structure is unchanged. Then we need only to update $O(\log N)$ ancestors of a given node. Obviously we could also insert values for any key that is present in a tree, but it is not any more practical than removal. General insertion with unchanged asymptotic complexity would require some additional mechanism such as AVL rotations and the structure described in Lemma 2.1 would need to become invariant or easily reproducible under such rotations. This seems to be not very relevant in practical applications and simultaneously a difficult problem beyond the scope of this work.

2.3 Algorithm for 2D Kolmogorov statistics

We reformulate our initial problem as follows: We have a set of triples:

$$S = \{(x_i, y_i, v_i) \text{ for } i \in 1...N\}$$

and we are interested in finding D such that:

$$D = \sup_{(a',b')} \| \sum_{i=0}^{N} \mathrm{I}_{x_i<a' \wedge y_j<b'} v_i \| \tag{9}$$

One can reproduce problem of finding 2d Kolmogorov statistics between two data samples $T = \{(X_i, Y_i) \text{for} i \in 1...N\}$, $T' = \{(X'_i, Y'_i) \text{for} i \in 1...N'\}$ by defining following set of triples:

$$S' = \{(X_i, Y_i, \frac{1}{N_1}) : i \in 1...N_1\} \cup \{(X'_i, Y'_i, -\frac{1}{N_2}) : i \in 1...N_2\}, \tag{10}$$

then $D(S')$ is exactly 2D Kolmogorov statistics for T_1, T_2. We propose an Algorithm 1 to evaluate D in $O(N\log N)$ time for $N = |S|$. The algorithm resembles a standard procedure of computational geometry known as sweep [4] - we

Data: $S' = \{(x_i, y_i, v_i) : i \in 1...N\}$
Result: D
$A \leftarrow$ ESET tree built of (x_i, v_i);
$\hat{x}_i, \hat{y}_i, \hat{v}_i \leftarrow x_i, y_i, v_i$ ordered by y_i;
$D \leftarrow 0$;
$j \leftarrow N$;
while $j \geq 1$ **do**
  $D = \max(D, A.\text{get\_max}())$;
  $A.\text{remove}(\hat{x}_j, \hat{v}_j)$;
  $j \leftarrow j - 1$;
  while $j > 1 \wedge \hat{y}_j = \hat{y}_{j-1}$ **do**
    $A.\text{remove}(\hat{x}_j, \hat{v}_j)$;
    $j \leftarrow j - 1$;
  end
end

Algorithm 1: Evaluation of (9) formula.

Lemma 2.2. *For a given main loop step with fixed* $j = j_0, j_0 \in \{1...N\}$

$$A.\text{get\_max}() = \sup_{(x,y)\in\{\hat{y}_{j0}\}\times\{\hat{x}_1...\hat{x}_N\}} D(x, y) \tag{11}$$

Proof. In each step we remove one sample or more (in case of equal y values) from ESET tree in y-descending order (by $\hat{x}_i, \hat{y}_i$ definition). Thus at step j_0 ESET-tree A contains up to date maximum of cumulative of v_i in x-ascending order for i s.t. $\hat{y}_i < \hat{y_{j0}}$ and this is this is exactly (11).

□

Theorem 2.3. *Algorithm 1 correctly evaluates* (9)*.*

Proof. Let $C = \{x_1...x_N\} \times \{y_1...y_N\}$. It follows from Lemma 1.1 that it suffices to evaluate $\| \sum_{i=0}^{N} \mathrm{I}_{x_i<a' \wedge y_j<b'} v_i \|$ for any $c \in C$ to find D, and it follows from $C = \bigcup_i \{\hat{y}_i\} \times \{\hat{x}_1...\hat{x}_N\}$ and Lemma 2.2 that algorithm does this evaluation. □

Algorithm takes $O(N\log N)$ time and $O(N)$ memory. Initially we need to build ESET-tree and store it in memory which takes $O(N\log N)$ time and $O(N)$ memory. Then we sort and store x_i, y_i and v_i by y_i which needs linear additional memory and $O(N\log N)$ time if we use e.g. mergesort. Final loop comprises update to D which is $O(1)$ and N removals that cost $O(\log N)$ each and using $O(\log N)$ memory.

3 Implementation

We share open source C++ implementation of the algorithm with bindings for Python/Numpy. Implementation was inter allia tested for aggreement with O(n^2) algorithm from [3] on few sets of random real numbers, ranging from 100 to 10000. Implementation's performance profiling confirmed significant improvement in asymptotic complexity and linear random-access-memory usage.

References

[1] Kolmogorov A, "Sulla determinazione empirica di una legge di distribuzione". G. Ist. Ital. Attuari. 4, 1933

[2] A. Justel, D. Peña, R. Zamar *A multivariate Kolmogorov–Smirnov test of goodness of fit* . Statistics and Probability Letters, 35 (3)

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery *Numerical Recipes* Cambridge University Press 1986

[4] L. Banachowski, K. Diks, W.Rytter *Algorithms and data structures (Polish)* WNT 1996