Supplements to “The Weighted Kendall and High-order Kernels for Permutations” (ICML 2018)

Yunlong Jiao [1] **Jean-Philippe Vert** [2]

Abstract

This is the supplements to the paper “The Weighted Kendall and High-order Kernels for Permutations” (ICML 2018).

1. Proofs of theorems

Proof of Theorem 2. The proof is constructive and the algorithm is summarized in Algorithm 1. C++/R implementation available in the package `kernrank` at https://github.com/YunlongJiao/kernrank.

The algorithm can be decomposed into three parts. First, we compute $\pi := \sigma'\sigma^{-1}$ by carrying out the inverse and composition of permutations, which can be done in linear time (Line 1). Due to the right-invariance of any concerning kernel, we have $K(\sigma, \sigma') = \kappa(\pi)$ where κ is the corresponding p.d. function:

$$\kappa_U^{@k}(\pi) = \sum_{1\le i<j\le n} \mathbb{1}_{i\le k}\mathbb{1}_{j\le k}\mathbb{1}_{\pi(i)\le k}\mathbb{1}_{\pi(j)\le k}\mathbb{1}_{\pi(i)<\pi(j)}\,,$$

$$\kappa_U^{\text{add}}(\pi) = \sum_{1\le i<j\le n} (u_i + u_j)\left(u_{\pi(i)} + u_{\pi(j)}\right)\mathbb{1}_{\pi(i)<\pi(j)}\,,$$

$$\kappa_U^{\text{mult}}(\pi) = \sum_{1\le i<j\le n} u_i u_j u_{\pi(i)} u_{\pi(j)}\mathbb{1}_{\pi(i)<\pi(j)}\,,$$

$$\kappa_W^{\text{avg}}(\pi) = \sum_{1\le i<j\le n} \frac{1}{n}\min\{i, \pi(i)\}\,\mathbb{1}_{\pi(i)<\pi(j)}\,.$$

Second, we register a global variable s to record $\kappa(\pi)$ (Line 2) and implement $\kappa(\pi)$ in the function QuickKappa (Lines 3–36). Finally, s is updated by calling the function QuickKappa (Line 37) and then outputted by the algorithm.

Algorithm 1 Top-k, average and weighted Kendall kernel with additive or multiplicative weight

input permutations σ, σ', size n, u for weighted Kendall kernel (optional), k for top-k Kendall kernel (optional)
1: $\pi := \sigma'\sigma^{-1}$
2: Initialize a global variable $s := 0$ then define
3: **function** QuickKappa($indices$)
4:   **if** length of $indices > 1$ **then**
5:     $pivot :=$ pick any element from $indices$
6:     $indhigh, indlow :=$ two empty arrays
7:     $cnum, ctop, cmin, cwa, cwb, cww := 0$
8:     **for each** i **in** $indices$ **do**
9:       **if** $\pi(i) < \pi(pivot)$ **then**
10:         Add i to $indlow$
11:         $cnum \mathrel{+}= 1$
12:         $cmin \mathrel{+}= \min\{i, \pi(i)\}/n$
13:         $ctop \mathrel{+}=$ **if** $i \le k$ **and** $\pi(i) \le k$ **then** 1 **else** 0
14:         $cwa \mathrel{+}= u_i$
15:         $cwb \mathrel{+}= u_{\pi(i)}$
16:         $cww \mathrel{+}= u_i * u_{\pi(i)}$
17:       **else**
18:         Add i to $indhigh$
19:         **switch** type of weighted Kendall kernel **do**
20:         **case** STANDARD**:**
21:           $s \mathrel{+}= cnum$
22:         **case** TOP-k**:**
23:           $s \mathrel{+}=$ **if** $i \le k$ **and** $\pi(i) \le k$ **then** $ctop$ **else** 0
24:         **case** AVERAGE**:**
25:           $s \mathrel{+}= cmin$
26:         **case** ADDITIVE WEIGHT**:**
27:           $s \mathrel{+}= cww + cwa * u_{\pi(i)} + cwb * u_i + cnum * u_i * u_{\pi(i)}$
28:         **case** MULTIPLICATIVE WEIGHT**:**
29:           $s \mathrel{+}= cww * u_i * u_{\pi(i)}$
30:         **end switch**
31:       **end if**
32:     **end for**
33:     QuickKappa($indhigh$)
34:     QuickKappa($indlow$)
35:   **end if**
36: **end function**
37: Call QuickKappa($[1, n]$) to update s
output $K(\sigma, \sigma') = s$

[1]University of Oxford, Oxford, UK [2]MINES ParisTech & Institut Curie & Ecole Normale Supérieure, PSL Research University, Paris, France. Correspondence to: Jean-Philippe Vert <jean-philippe.vert@mines-paristech.fr>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Central to the algorithm is the computation of $\kappa(\pi)$. It is based on an idea similar to a quicksort algorithm, where we recursively partition an array into two sub-arrays consisting of greater or smaller values according to a $pivot$, and cumulatively count the contributions between pairs of items with one in each sub-array. Specifically, suppose now π is divided into two sub-arrays $\pi_{indhigh}$ and π_{indlow} where ranks in $\pi_{indhigh}$ are all higher and those in π_{indlow}, now $\kappa(\pi)$ can be decomposed into

$$\kappa(\pi) = \kappa(\pi_{indhigh}) + \kappa(\pi_{indlow}) + c(\pi_{indhigh}, \pi_{indlow}) ,$$

where c characterizes the weighted non-inversion number of π restricted on pairs of items with one in each sub-array. The computation of $c(\pi_{indhigh}, \pi_{indlow})$ depends on specific choice of weight and is depicted in the pseudo-code (Lines 19–30). Notably a single linear-time pass over π is sufficient to compute $c(\pi_{indhigh}, \pi_{indlow})$. By the analysis of deduction typically for a quicksort algorithm, the overall time complexity of our algorithm is on average $O(n \ln(n))$.

In particular, recall that the standard Kendall kernel is merely a special case of the weighted Kendall kernel with constant weight, and hence our algorithm provides an alternative to the efficient algorithm based on merge sort proposed by Knight (1966). $\square$

References

Knight, W. R. A computer method for calculating Kendall's tau with ungrouped data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.