fastJT
Efficient Jonckheere-Terpstra Test Statistics for Robust Machine Learning and Genome-Wide Association Studies

2017-08-14
Outline

Introduction

Examples

Session Information
Introduction

- This document provides an example for using the \texttt{fastJT} package to calculate the Jonckheere-Terpstra test statistics for large data sets (multiple dependent and independent variables) commonly encountered in machine learning or GWAS. The functionality is also included to perform k-fold cross validation or feature sets.

- The calculation of the standardized test statistic employs the null variance equation as defined by Hollander and Wolfe (1999, eq. 6.19) to account for ties in the data.

- The major algorithm in this package is written in C++, which is ported to R by Rcpp, to facilitate fast computation.

- Features of this package include:
  1. $O(N \times \log(N))$ computational complexity (where $N$ is the number of the samples)
  2. OpenMP supported parallelization
  3. Customized output of top $m$ significant independent variables for each dependent variable
```r
res <- fastJT(Y, X, outTopN=15, numThreads=1, standardized=TRUE)
```

Function arguments:

- **Y**: Matrix of continuous values, representing dependent variables, with sample IDs as row names and variable names as column names.

- **X**: A matrix of integer values, representing independent variables, with sample IDs as row names and feature IDs as column names.

- **outTopN**: Number of top statistics to return (i.e., the largest standardized statistics). The default value is 15. If `outTopN` is set to `NA`, all results will be returned.

- **numThreads**: Number of threads to use for parallel computation. The default value is 1 (sequential computation).

- **standardized**: A boolean to specify whether to return standardized statistics or non-standardized statistics. The default value is TRUE, returning standardized statistics.

Users may wish to consider the `dplyr::recode()` function for converting non-numeric group indices into ordinal values for argument `X`. 
Returned Values

Function returns:

**J**: A matrix of standardized/non-standardized Jonckheere-Terpstra test statistics, depending on option `standardized`, with column names from input `Y`. If `outTopN` is not `NA`, results are sorted within each column.

**XIDs**: If `outTopN` is not `NA`, this is a matrix of column names from `X` associated with top standardized Jonckheere-Terpstra test statistics from `J`. Otherwise this is an unsorted vector of column names from input `X`. 
```r
res <- fastJT.select(Y, X, cvMesh=NULL, kFold=10L, selCrit=NULL, outTopN=15L, numThreads=1L)
```

**Function arguments:**

- **Y**: Matrix of continuous values, representing dependent variables, with sample IDs as row names and variable names as column names.
- **X**: Matrix of integer values, representing independent variables, with sample IDs as row names and feature IDs as column names.
- **cvMesh**: A user-defined function to specify how to separate the data into training and testing parts. The inputs of this function are a vector of the sample IDs and `kFold`, an integer representing the number of folds for cross validation. The output of this function is a list of `kFold` vectors of sample IDs forming the testing subset for each fold. The default value is `NULL`, and if no function is specified, the data are partitioned sequentially into `kFold` equal sized subsets. Optional.
fastJT.select

Function arguments continued:

**kFold**: An integer to indicate the number of folds. Optional. The default value is 10.

**selCrit**: A user-defined function to specify the criteria for selecting the top features. The inputs of this function include \( J \), the matrix of statistics resulting from fastJT, and \( P \), the matrix of p-values from \( \text{pvalues}(J) \). The output is a data frame containing the selected feature IDs for each trait of interest. Optional. The default value is **NULL**, and if no function is specified, the features the largest standardized Jonckheere-Terpstra test statistics are selected.

**outTopN**: An integer to indicate the number of top hits to be returned when \( \text{selCrit} = \text{NULL} \). Unused if \( \text{selCrit} \neq \text{NULL} \). Optional. The default value is 15.

**numThreads**: A integer to indicate the number of threads to be used in the computation. Optional. The default value is 1 (sequential computation).

The function withholds one subset while the remaining \( k\text{Fold}-1 \) subsets are used to test the features. The process is repeated \( k\text{Fold} \) times, with each of the subsets withheld exactly once as the validation data.
Returned Values

Function returns: Three lists of length kFold:

**J**: A list of matrices of standardized Jonckheere-Terpstra test statistics, one for each cross validation.

**Pval**: A list of matrices of p-values, one for each cross validation.

**XIDs**: A list of matrices of the selected feature IDs, one for each cross validation.
Simulate Data

1 Define the number of markers, samples, and features:

```r
num_sample <- 100
num_marker <- 4
num_feature <- 50
```

2 Create two matrices containing marker levels and feature information.

   a. Data contains the samples’ marker levels.
   b. Feature contains the samples’ feature values.

```r
set.seed(12345);
Data <- matrix(rnorm(num_sample*num_marker),
               num_sample, num_marker)
Feature <- matrix(rbinom(num_sample*num_feature,2,0.5),
                  num_sample, num_feature)
colnames(Data) <- paste0("Mrk:",1:num_marker)
colnames(Feature) <- paste0("Ftr:",1:num_feature)
```
Load Package

Load `fastJT` (after installing its dependent packages):

```r
library(fastJT)
```
Example Execution

```r
JTStat <- fastJT(Y=Data, X=Feature, outTopN=10)
summary(JTStat, Y2Print=1:4, X2Print=1:5)
```

<table>
<thead>
<tr>
<th>Mark:1</th>
<th>Mark:2</th>
<th>Mark:3</th>
<th>Mark:4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNPID</td>
<td>P-value</td>
<td>SNPID</td>
<td>P-value</td>
</tr>
<tr>
<td>Ftr:35</td>
<td>1.7e-02</td>
<td>Ftr:35</td>
<td>2.0e-02</td>
</tr>
<tr>
<td>Ftr:17</td>
<td>1.7e-02</td>
<td>Ftr:7</td>
<td>5.7e-02</td>
</tr>
<tr>
<td>Ftr:27</td>
<td>7.0e-02</td>
<td>Ftr:46</td>
<td>9.1e-02</td>
</tr>
<tr>
<td>Ftr:28</td>
<td>7.0e-02</td>
<td>Ftr:40</td>
<td>9.4e-02</td>
</tr>
<tr>
<td>Ftr:14</td>
<td>8.8e-02</td>
<td>Ftr:29</td>
<td>1.5e-01</td>
</tr>
</tbody>
</table>
**Example Execution: Statistics in the Summary**

```r
summary(JTStat, Y2Print=1:4, X2Print=1:5, printP=FALSE)
```

### Johckheere-Terpstra Test for Large Matrices
### Top Standardized Statistics

<table>
<thead>
<tr>
<th>SNPID</th>
<th>J*</th>
<th>SNPID</th>
<th>J*</th>
<th>SNPID</th>
<th>J*</th>
<th>SNPID</th>
<th>J*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ftr:17</td>
<td>-2.388</td>
<td>Ftr:7</td>
<td>1.905</td>
<td>Ftr:49</td>
<td>-2.106</td>
<td>Ftr:34</td>
<td>2.396</td>
</tr>
<tr>
<td>Ftr:27</td>
<td>-1.813</td>
<td>Ftr:46</td>
<td>1.693</td>
<td>Ftr:26</td>
<td>-2.072</td>
<td>Ftr:47</td>
<td>-2.089</td>
</tr>
<tr>
<td>Ftr:28</td>
<td>1.813</td>
<td>Ftr:40</td>
<td>1.676</td>
<td>Ftr:47</td>
<td>1.914</td>
<td>Ftr:23</td>
<td>2.000</td>
</tr>
<tr>
<td>Ftr:14</td>
<td>1.706</td>
<td>Ftr:29</td>
<td>1.432</td>
<td>Ftr:30</td>
<td>-1.662</td>
<td>Ftr:16</td>
<td>1.955</td>
</tr>
</tbody>
</table>
Example Execution: Sorting in the Summary

```
JTAll <- fastJT(Y=Data, X=Feature, outTopN=NA)
summary(JTAll, Y2Print=1:4, outTopN=3)
```

```
##
## Johckheere-Terpstra Test for Large Matrices
## P-values for Top Standardized Statistics
## ================================================================================
## Mrk:1| Mrk:2| Mrk:3| Mrk:4|
## --------------------------------------------------------------------------------
## SNPID P-value| SNPID P-value| SNPID P-value| SNPID P-value|
## --------------------------------------------------------------------------------
## Ftr:35 1.7e-02| Ftr:35 2.0e-02| Ftr:20 1.9e-02| Ftr:46 1.2e-02|
## Ftr:17 1.7e-02| Ftr:7 5.7e-02| Ftr:49 3.5e-02| Ftr:34 1.7e-02|
## Ftr:27 7.0e-02| Ftr:46 9.1e-02| Ftr:26 3.8e-02| Ftr:47 3.7e-02|
```
Example Execution: fastJT.select

```r
fastJT.select(Y=Data, X=Feature, cvMesh=NULL, kFold=5, selCrit=NULL, outTopN=5, numThreads=1)
```

```r
## $J
## $J[[1]]
## Mrk:1 Mrk:2 Mrk:3 Mrk:4
## [1,] -2.449202 -2.401972 2.402878 2.491444
## [2,] -2.356094 1.863353 2.159547 2.333146
## [3,] -1.798716 1.662280 -2.066804 -2.185883
## [4,]  1.796900 1.662069 -1.902853  2.004838
## [5,]  1.754959 1.399926 -1.893419  1.955443
##
## $J[[2]]
## Mrk:1 Mrk:2 Mrk:3 Mrk:4
## [1,] -2.449202 -2.401972 2.402878 2.491444
## [2,] -2.356094 1.863353 2.159547 2.333146
## [3,] -1.798716 1.662280 -2.066804 -2.185883
## [4,]  1.796900 1.662069 -1.902853  2.004838
## [5,]  1.754959 1.399926 -1.893419  1.955443
##
## $J[[3]]
## Mrk:1 Mrk:2 Mrk:3 Mrk:4
## [1,] -2.449202 -2.401972 2.402878 2.491444
## [2,] -2.356094 1.863353 2.159547 2.333146
## [3,] -1.798716 1.662280 -2.066804 -2.185883
## [4,]  1.796900 1.662069 -1.902853  2.004838
## [5,]  1.754959 1.399926 -1.893419  1.955443
##
## $J[[4]]
## Mrk:1 Mrk:2 Mrk:3 Mrk:4
## [1,] -2.449202 -2.401972 2.402878 2.491444
## [2,] -2.356094 1.863353 2.159547 2.333146
## [3,] -1.798716 1.662280 -2.066804 -2.185883
## [4,]  1.796900 1.662069 -1.902853  2.004838
## [5,]  1.754959 1.399926 -1.893419  1.955443
##
## $J[[5]]
```
Example User-Defined cvMesh Function

Mesh <- function(rownamesData, kFold){
  numSamples <- length(rownamesData)
  res <- NULL
  subSampleSize <- floor(numSamples/kFold)
  for (i in 1:kFold){
    start <- (i-1)*subSampleSize + 1
    if(i < kFold)
      end <- i*subSampleSize
    else
      end <- numSamples
    if(i == 1)
      res <- list(c(start:end))
    else
      res[[i]] <- c(start:end)
  }
  res
}
Example User-Defined selCrit Function

```r
whichpart <- function(x, n=30) {
  nx <- length(x)
  p <- nx-n
  xp <- sort(x, partial=p)[p]
  which(x > xp)
}

selectCrit <- function(J, P){
  pcut <- 0.95
  hit <- NULL
  for(i in 1:ncol(P)){
    if(i == 1)
      hit <- list(rownames(P)[whichpart(P[,i], 4)])
    else
      hit[[i]] = rownames(P)[whichpart(P[,i], 4)]
  }
  res <- do.call(cbind, hit)
  colnames(res) <- colnames(P)
  res
}
```
**Example Execution with User-Defined Functions**

```r
fastJT.select(Data, Feature, cvMesh=Mesh, kFold=5, selCrit=selectCrit, outTopN=5, numThreads=1)
```

### 

```
## $J
## $J[[1]]
```

<table>
<thead>
<tr>
<th>Mrk:1</th>
<th>Mrk:2</th>
<th>Mrk:3</th>
<th>Mrk:4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.044871948</td>
<td>0.30810327</td>
<td>-0.52243597</td>
<td>-1.41325629</td>
</tr>
<tr>
<td>0.064765493</td>
<td>-0.57607202</td>
<td>0.48744566</td>
<td>0.91694304</td>
</tr>
<tr>
<td>0.708932518</td>
<td>-0.93211498</td>
<td>0.38072302</td>
<td>-0.99775688</td>
</tr>
<tr>
<td>0.177698249</td>
<td>-0.10530267</td>
<td>1.24388774</td>
<td>0.13820975</td>
</tr>
<tr>
<td>-0.135648047</td>
<td>0.29164330</td>
<td>-1.58029974</td>
<td>1.23439722</td>
</tr>
<tr>
<td>0.626045363</td>
<td>0.20007635</td>
<td>-1.20691219</td>
<td>-0.61313721</td>
</tr>
<tr>
<td>-1.007934807</td>
<td>1.86336316</td>
<td>0.53712315</td>
<td>1.32623001</td>
</tr>
<tr>
<td>0.382152784</td>
<td>-1.08715878</td>
<td>-1.30459054</td>
<td>0.17789871</td>
</tr>
<tr>
<td>-0.627298931</td>
<td>-1.04772268</td>
<td>0.28695589</td>
<td>1.08776304</td>
</tr>
<tr>
<td>-0.552417489</td>
<td>-1.03827865</td>
<td>0.62562945</td>
<td>-0.54576186</td>
</tr>
<tr>
<td>0.718808299</td>
<td>0.69218577</td>
<td>0.79867589</td>
<td>-0.06656632</td>
</tr>
<tr>
<td>0.214931358</td>
<td>-0.73882654</td>
<td>-1.25600512</td>
<td>-1.24928852</td>
</tr>
<tr>
<td>-0.637403711</td>
<td>-1.34635274</td>
<td>-0.27317302</td>
<td>-0.61138723</td>
</tr>
<tr>
<td>1.754958762</td>
<td>-0.17582763</td>
<td>-0.27535270</td>
<td>0.53411788</td>
</tr>
<tr>
<td>0.987651750</td>
<td>0.20792668</td>
<td>0.01299542</td>
<td>-1.13709905</td>
</tr>
<tr>
<td>1.487980337</td>
<td>0.09875976</td>
<td>0.72423822</td>
<td>1.95544319</td>
</tr>
<tr>
<td>-2.356093867</td>
<td>-0.36605978</td>
<td>0.00000000</td>
<td>-1.92347776</td>
</tr>
<tr>
<td>0.846622444</td>
<td>1.31915590</td>
<td>0.22970376</td>
<td>0.01312593</td>
</tr>
<tr>
<td>0.006991013</td>
<td>0.34955065</td>
<td>0.78998447</td>
<td>-1.37023855</td>
</tr>
<tr>
<td>-0.055126760</td>
<td>-0.45074233</td>
<td>2.40287819</td>
<td>0.49614084</td>
</tr>
<tr>
<td>-0.201986904</td>
<td>0.05212565</td>
<td>-0.19547120</td>
<td>-0.81446332</td>
</tr>
<tr>
<td>-0.242183804</td>
<td>-0.22254728</td>
<td>0.88364361</td>
<td>-0.96873521</td>
</tr>
<tr>
<td>-0.359172943</td>
<td>0.58773754</td>
<td>0.34612111</td>
<td>2.00483806</td>
</tr>
<tr>
<td>0.661565280</td>
<td>-0.17050652</td>
<td>-1.35041160</td>
<td>1.05032014</td>
</tr>
<tr>
<td>0.892047636</td>
<td>0.20079301</td>
<td>1.25413339</td>
<td>-0.41146109</td>
</tr>
<tr>
<td>0.275352698</td>
<td>-0.93221817</td>
<td>-2.06680399</td>
<td>0.82605809</td>
</tr>
<tr>
<td>-1.798716484</td>
<td>-0.26774787</td>
<td>-0.46684245</td>
<td>0.51489976</td>
</tr>
<tr>
<td>1.796899533</td>
<td>1.09120808</td>
<td>0.31364065</td>
<td>1.17615242</td>
</tr>
<tr>
<td>0.364631969</td>
<td>1.39992631</td>
<td>0.20184984</td>
<td>-0.71624137</td>
</tr>
<tr>
<td>0.451127576</td>
<td>0.81993260</td>
<td>-1.89341866</td>
<td>1.28752469</td>
</tr>
<tr>
<td>-0.868212309</td>
<td>-0.42431429</td>
<td>0.41125846</td>
<td>0.15014198</td>
</tr>
</tbody>
</table>
References

Session Information

- R version 3.3.3 (2017-03-06), x86_64-pc-linux-gnu
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: fastJT 1.0.4, knitr 1.13
- Loaded via a namespace (and not attached): Rcpp 0.12.4, evaluate 0.8, formatR 1.2.1, highr 0.5.1, magrittr 1.5, stringi 1.0-1, stringr 1.0.0, tools 3.3.3

```r
```