Differential analysis of time series count data – the DTWscore package

Zhuo Wang[1], Shuilin Jin[1]

[1] Department of Science, Harbin Institute of Technology

August 31, 2016

Abstract

The development of single-cell RNA sequencing has enabled profound discoveries in biology, ranging from dissection of the composition of complex tissues to the identification of novel cell type and dynamic in some specialized cellular environments. However, large scale generation of single-cell RNA-seq data collected at multiple time points are still the obstacle to effectively measure gene expression levels in transcriptome analysis. We present an algorithm based on Dynamic Time Warping combining with time series data, which enables detection of gene expression changes across single-cell RNA-seq samples and recovery of single-cell gene expression kinetic and heterogeneity from a particular biological processes. DTWscore successfully identifies differentially expressed genes under a number of different scenarios and enables detection of the genes with the most highly variable expression patterns from time series scRNA-seq data. The study is confined to methods that are implemented and available within the R framework.

DTWscore version: 1.0

If you use *DTWscore* in published research, please cite:

Z. Wang, SL. Jin, CP. Zhang: **DTWscore: Differential expression and transcriptional heterogeneity analysis for time-series single-cell RNA-seq data**.

Contents

1 Standard workflow

1.1 Input data

The function DTWscore needs four parameters as inputs.

Count matrix input

Alternatively, the function DTWscore can be used if you already have a matrix of read counts prepared from another source. Another method for quickly producing count matrices from alignment files is the featureCounts function in the *Rsubread* package. To use DTWscore, the user should provide the counts matrix, the information about the samples (the columns of the count matrix) as a *DataFrame* or *data.frame*, and the design formula.

number of time points

The next two parameters are the numbers of time points from two time periods respectively.

threshold

The last parameter is the threshold for identifying the differentially expressed genes.

1.2 Pre-filtering

To demonstate the use of DWTscore, we will first load the the count matrix which we will name 'mydata'.

```
library("DTWscore")
mydata=read.csv("HSMM.csv", row.names = 1)
```

The numbers of genes and samples:

```
dim(mydata)
```

```
## [1] 27429   372
```

If you have used the featureCounts function in the *Rsubread* package, the matrix of read counts can be directly provided from the "counts" element in the list output. The count matrix and column data can also be read into R from flat files using base R functions such as read.csv or read.delim.

With the count matrix, we only need some specific column for comparison.

```
a=mydata
mydata1=data.frame(A01_0=a$T0_CT_A01,A01_24=a$T24_CT_A01,
                   A01_48=a$T48_CT_A01,A01_72=a$T72_CT_A01,
                   B01_0=a$T0_CT_B01,B01_24=a$T24_CT_B01,
                   B01_48=a$T48_CT_B01,B01_72=a$T72_CT_B01,row.names=rownames(a))
head(mydata1)
```

```
##                     A01_0    A01_24  A01_48   A01_72     B01_0   B01_24
## ENSG00000000003  21.984400  45.77240 68.1934 13.01190 42.750500  89.0997
## ENSG00000000419  40.059700 136.06200 34.6372 75.04030 10.808200  61.6789
## ENSG00000000457   0.937081   0.00000  0.0000  0.00000  0.000000   0.0000
## ENSG00000000460   0.740922   0.00000  0.0000  0.00000  0.762205   0.0000
```

```
## ENSG00000000971   3.002980   0.00000 51.0128  1.21678 19.515500  54.3210
## ENSG00000001036 128.197000   4.53641 34.8389 74.06250 25.989200 108.1740
##                   B01_48    B01_72
## ENSG00000000003 124.1000  0.695473
## ENSG00000000419  32.2522 27.555400
## ENSG00000000457   0.0000  0.000000
## ENSG00000000460   0.0000  0.000000
## ENSG00000000971 129.7820 44.606600
## ENSG00000001036  72.9391 33.171800
```

While it is necessary to pre-filter low count genes before running the *DTWscore* functions, there are two reasons which make pre-filtering useful: by removing rows in which there are no reads or nearly no reads. Here we perform a minimal pre-filtering to remove rows that have only 0 or 0.1 FPKM read.

```
filter=apply(mydata1,1,function(x) length(x[x>0.1])>0)
data_filter=mydata1[filter,]
dim(data_filter)
```

```
## [1] 14713     8
```

1.3 Differential expression analysis

The standard differential expression analysis steps are wrapped into a single function, `DTWscore`. Results tables are automately generated using the function `DTWscore`, which extracts a results table with gene names and their relative DTWscores which are ordeded by the largest DTWscores.

```
library("dtw")
n1=4
n2=4
result <- DTWscore(mydata1,n1,n2,0.99)
```

```
## Step 1: fliter unexpressed genes
## Step 2: calculate DTWscore for each gene
## Step 3: choose heterogeneous genes by users
## names of highly variable genes are written in the txt file
```

```
head(result)
```

```
##                     A01_0   A01_24    A01_48   A01_72    B01_0   B01_24
## ENSG00000125148 10794.800  125.763  195.8140  790.387 1598.520  260.395
## ENSG00000205542  1963.750 7496.160 5504.9200 2087.730 1298.920 3198.040
## ENSG00000159251   814.901 8558.980   15.1666  327.891 2851.140 3683.550
## ENSG00000198712 10674.900 9998.290 5957.3600 5139.660 6862.580 9426.360
## ENSG00000115414  6995.810 1371.550 4160.2800 2185.140 2299.130  631.335
## ENSG00000102265  1336.400  275.847 6403.9300 3497.220  921.939  655.596
##                    B01_48   B01_72      dis
## ENSG00000125148  326.97900 1144.470 1853.350
## ENSG00000205542 3497.32000 1108.730 1602.405
## ENSG00000159251    4.86433  117.053 1394.921
## ENSG00000198712 6142.83000 6551.070 1308.875
```

```
## ENSG00000115414 2470.76000 2751.980 1247.568
## ENSG00000102265 1215.43000 3547.470 1090.424
```

For advanced users, note that the names of all the hegerogeneous genes are saved as a 'txt' file which could be direct uploaded to 'David' for the GO analysis.

1.4 Exploring and exporting results

We will explore some statistic properties of all the DTWscores.

```
DTWscore_all=result$dis
head(DTWscore_all)
```

```
## [1] 1853.350 1602.405 1394.921 1308.875 1247.568 1090.424
```

```
tail(DTWscore_all)
```

```
## [1] 0 0 0 0 0 0
```

1.4.1 Boxplot

```
boxplot(DTWscore_all,col="blue",main="Boxplot for DTWscores")
```

```
boxplot(DTWscore_all,col="green",main="Boxplot for DTWscores without outliers" ,
        outline=FALSE)
```

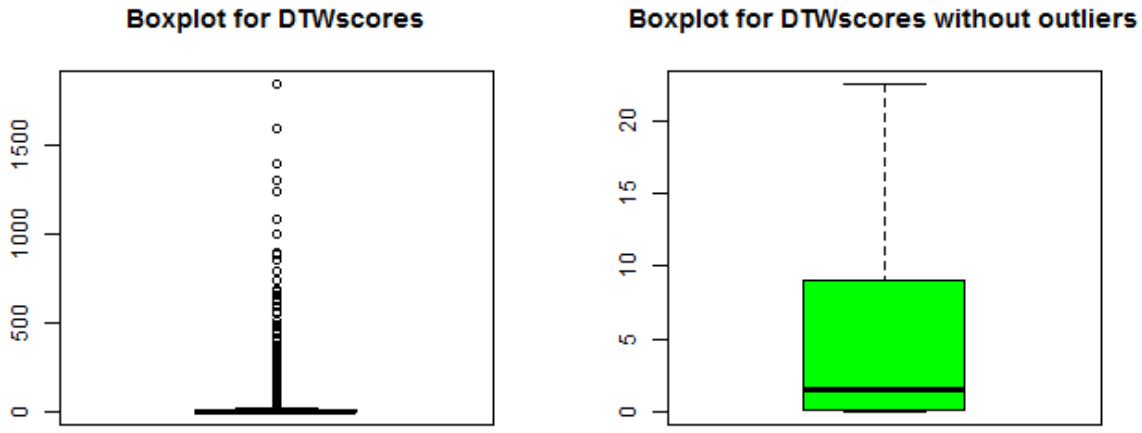

Figure 1: **BOX-plot.** These plots show the boxplot of DTWscores through their quartiles. The left plot shows the upper and lower quartiles with outliers, while the right plot, produced by the code above, shows the variation in samples of a statistical population without making any assumptions of the underlying statistical distribution. No outliers is displayed in the right plot. These plots shows where the majoratory of DTWscores gathered.

1.4.2 Trajectory plot

It can also be useful to examine the counts of reads for a single gene across the cells. A simple function for making this plot is `dtwPlot`, which plots the counts by FPKM values to allow for trajectory plotting. The counts are grouped by the cells within their time periods, where more than one time points can be specified. Here we specify the gene which had the smallest and the largest DTWscores from the results table created above. You can select the gene to plot by rowname or by numeric index.

```
dis=vector()
  for(i in 1:nrow(data_filter))
    {
    x=t(data_filter[i,(1:n1)])
    y=t(data_filter[i,(n1+1):(n1+n2)])
        dis[i]=dtw(x,y,keep=TRUE)$normalizedDistance
        }
```

```
id_max=order(dis,decreasing=TRUE)[1:10] ###
x=t(data_filter[id_max[1],(1:4)])
y=t(data_filter[id_max[1],(5:8)])
align=dtw(x,y,keep=TRUE)
plot(align,offset=0,type="two", lwd=3, match.col="grey50",xlab="Time",ylab=
     "Gene Expression Level")
legend("topright",c("cell 1","cell 2"), pch=21, col=1:6)
```

```
 ###
id_min=order(dis)[1:10]
x=t(data_filter[id_min[1],(1:4)])
y=t(data_filter[id_min[1],(5:8)])
align=dtw(x,y,keep=TRUE)
plot(align,offset=0,type="two", lwd=3, match.col="grey50",xlab="Time",ylab=
     "Gene Expression Level")
legend("topright",c("cell 1","cell 2"), pch=21, col=1:6)
```

1.4.3 Heatmap

To explore a count marix, it is often instructive to look at it as a heatmap. Below we show how to produce such a heatmap for various transformations of the time series data.

```
library(pheatmap)
library(graphics)
aa=data_filter[c(id_max,id_min),]
 pheatmap(aa,cluster_rows = F,cluster_cols = F,color=colorRampPalette(
   c("#00007F", "blue", "#007FFF", "cyan", "#7FFF7F", "yellow", "#FF7F00", "red", "#7F0000"))(50)
```

```
 m=20
sigma=2
aa = matrix(ncol=300,nrow=4*m)
for( i in 0:(m-1)){
t=seq(0+0.1*i,29.9+0.1*i,0.1)
```

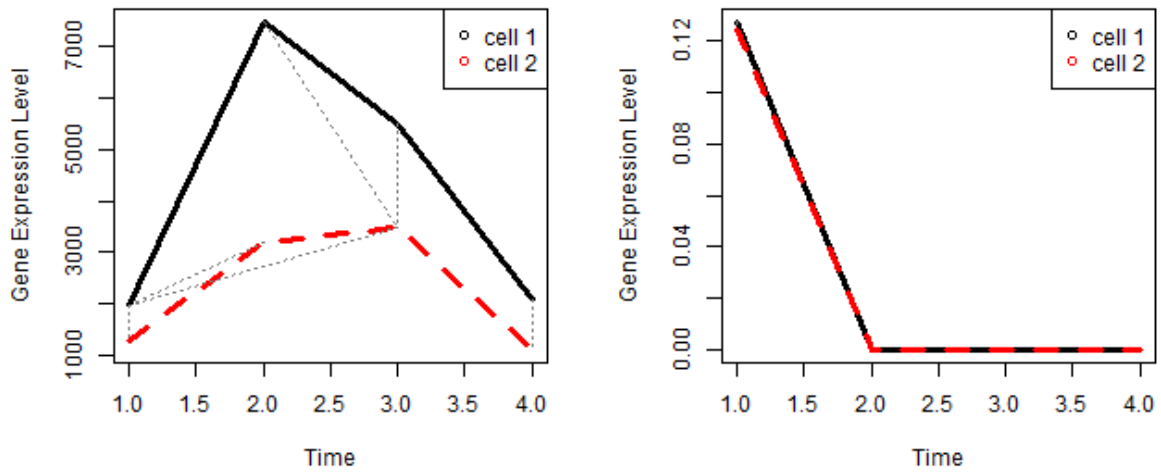

Figure 2: **trajectory-plot.** Plots display overall biological progress between genes with highest DTWscore and lowest DTWscore. The left panel of the plot are temporal process of the highly variable genes from two cells. It is obvious that they should be declared as differentially expressed. On the contrary, the left panel of the plot testify that the genes with lowest DTWscore undergo the same expression pattern.

```
f1=5*rnorm(1,mean=1,sd=0.1)*cos(t/5)+8+rnorm(1,mean=0,sd=sigma)
f2=5*rnorm(1,mean=1,sd=0.1)*sin(t/5)+8+rnorm(1,mean=0,sd=sigma)
f3=rnorm(1,mean=1,sd=0.1)*(t/10)^2+rnorm(1,mean=0,sd=sigma)+5
f4=5*log(t+1)+8
aa[m-i,] = f1
aa[2*m-i,] = f2
aa[3*m-i,] = f3
aa[4*m-i,] = f4
}
pheatmap(aa,cluster_rows = F,cluster_cols = F,color=colorRampPalette(
  c("#00007F", "blue", "#007FFF", "cyan", "#7FFF7F", "yellow", "#FF7F00", "red","#7F0000"))(50))
```

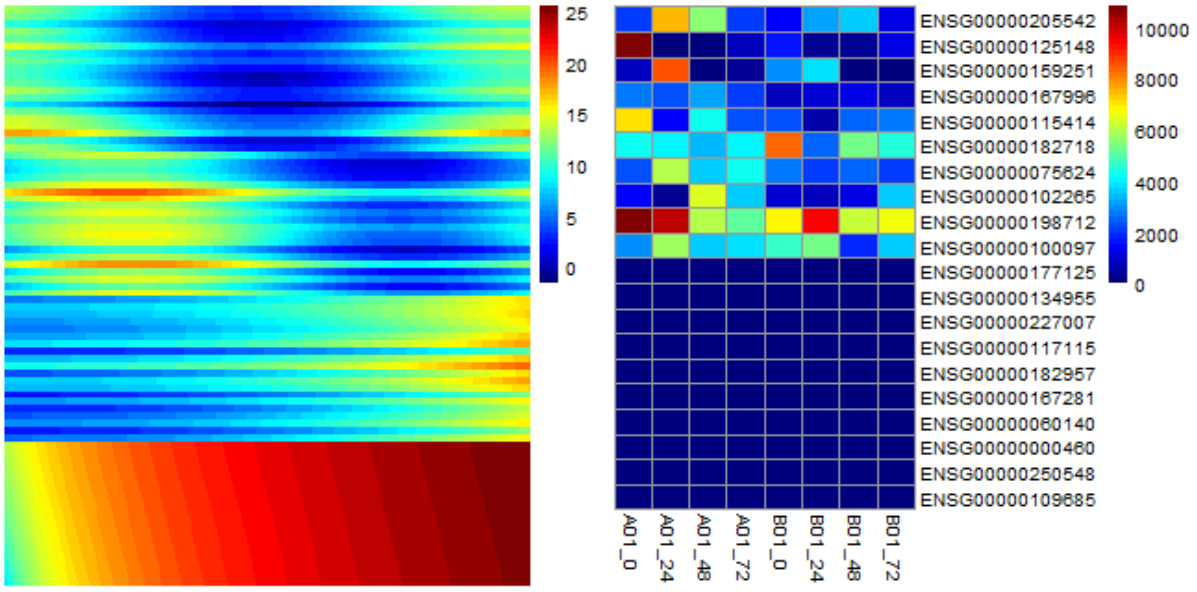

Figure 3: **heatmap-plot.** The left plot shows the gene expression levels from samples that were taken at even time intervals. Experiments shows four pattern of gene expression for each function. The right plot shows the analysis results with 10 most heterogeneous genes and 10 non-hegerogeneous genes.