Introduction to the TSSi package: Identification of Transcription Start Sites

Julian Gehring, Clemens Kreutz

April 30, 2018

Along with the advances in high-throughput sequencing, the detection of transcription start sites *(TSS)* using CAP-capture techniques has evolved recently. While many experimental applications exist, the analysis of such data is still non-trivial. Approaching this, the *TSSi* package offers a flexible statistical preprocessing for CAP-capture data and an automated identification of start sites.

1 Introduction

High throughput sequencing has become an essential experimental approach to investigate genomes and transcriptional processes. While cDNA sequencing (*RNA-seq*) using random priming and/or fragmentation of cDNA will result in a shallow distribution of reads typically biased towards the 3' end, approaches like CAP-capture enrich 5' ends and yield more clearly distinguishable peaks around the transcription start sites.

Predicting the location of transcription start sites *(TSS)* is hampered by the existence of alternative ones since their number within regions of transcription is unknown. In addition, measurements contain false positive counts. Therefore, only the counts which are significantly larger than an expected number of background reads are intended to be predicted as TSS. The number of false positive reads increases in regions of transcriptional activity and such reads obviously do not map to random positions. On the one hand, these reads seem to occur sequence dependently and therefore cluster to certain genomic positions, on the other hand, they are detected more frequently than being originated from real TSS. Because currently, there is no error model available describing such noise, the *TSSi* package implements an heuristic approach for an automated and flexible prediction of TSS.

2 Data set

To demonstrate the functionality and usage of this package, experimental CAP-capture data obtained with Solexa sequencing is used. The reads were mapped to the genome,

such that the number of sequenced 5' ends and their positions in the genome are available[1]. The data frame *physcoCounts* contains information about the chromosome, the strand, the 5' position, and the total number of mapped reads. Additionally, regions based on existing annotation are provided which are used here to divide the data into independent subsets for analysis.

```
> library(TSSi)
```

```
> data(physcoCounts)
> head(physcoCounts)
```

```
  chromosome region start strand counts
1         s1      1 82747      +      3
2         s1      1 82771      +      1
3         s1      1 82853      +      7
4         s1      1 82854      +      6
5         s1      1 82875      +      4
6         s1      1 82898      +      5
```

```
> table(physcoCounts$chromosome, physcoCounts$strand)
```

```
    -  +
  s1 47 61
  s2 43 68
```

3 Segment read data

As a first step in the analysis, the reads are passed to the *segmentizeCounts* method. Here, the data is divided into *segments*, for which the following analysis is performed independently. This is performed based on the information about the chromosomes, the strands, and the regions of the reads. The segmented data is returned as an object of the class *TssData*.

```
> attach(physcoCounts)
> x <- segmentizeCounts(counts=counts, start=start, chr=chromosome, region=region, strand
> detach(physcoCounts)
> x
```

```
* Object of class 'TssData' *
  Data imported
```

[1]The *sequencing workflow* at the bioconductor website describes basic steps and tools for the import and processing of sequencing data (`http://bioconductor.org/help/workflows/high-throughput-sequencing/`).

```
** Segments **
   Segments (5): s1_+_1, s1_+_2, s1_-_3, s2_+_4, s2_-_5
   Chromosomes (2): s1, s2
   Strands (2): +, -
   Regions (5): 1, 2, 3, 4, 5
   nCounts (5): 978, 587, 848, 466, 690
```

The segments and the associated read data are accessible through several *get* methods. Data from individual segments can be referred to by either its name or an index.

```
> segments(x)

       chr strand region nPos nCounts   start     end
s1_+_1  s1      +      1   28     978   82747   82994
s1_+_2  s1      +      2   33     587  814741  815042
s1_-_3  s1      -      3   47     848 1435037 1435157
s2_+_4  s2      +      4   68     466 1454505 1455353
s2_-_5  s2      -      5   43     690 1574882 1575467

> names(x)

[1] "s1_+_1" "s1_+_2" "s1_-_3" "s2_+_4" "s2_-_5"

> head(reads(x, 3))

     start     end counts replicate
62 1435037 1435037      4         1
63 1435039 1435039      4         1
64 1435043 1435043      3         1
65 1435045 1435045      1         1
66 1435047 1435047      1         1
67 1435049 1435049      1         1

> head(start(x, 3))

[1] 1435037 1435039 1435043 1435045 1435047 1435049

> head(start(x, names(x)[3]))

[1] 1435037 1435039 1435043 1435045 1435047 1435049
```

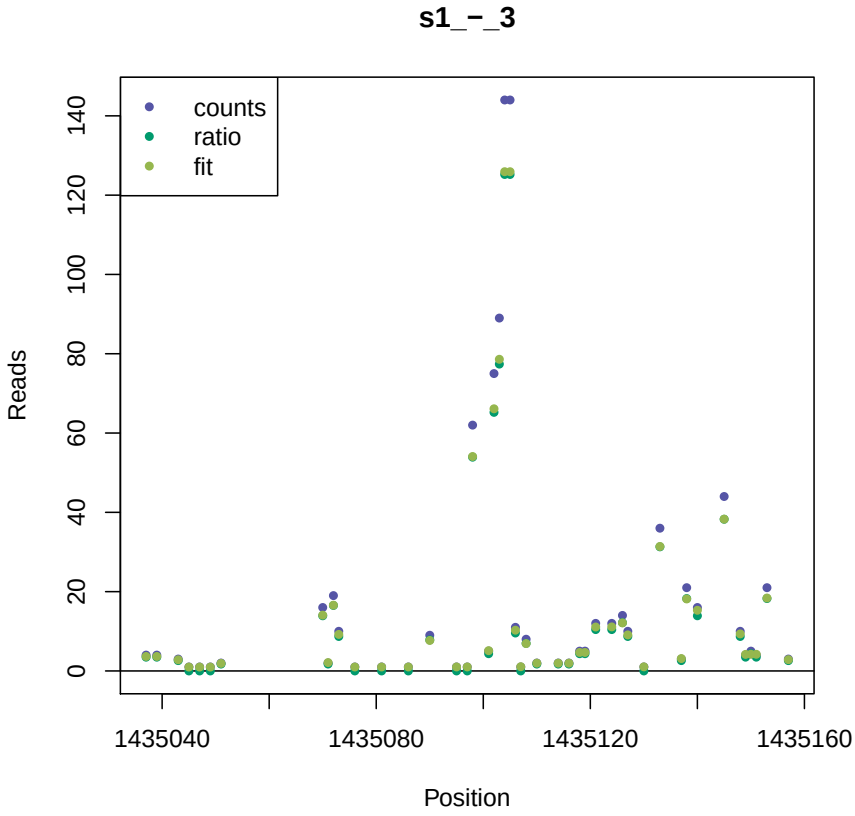
4 Normalization

The normalization reduces the noise by shrinking the counts towards zero. This step is intended to eliminate false positive counts as well as making further analyzes more robust

by reducing the impact of large counts. Such a shrinkage or regularization procedure constitutes a well-established strategy in statistics to make predictions conservative, that means to reduce the number of false positive predictions. To enhance the shrinkage of isolated counts in comparison to counts in regions of strong transcriptional activity, the information of consecutive genomic positions in the measurements is regarded by evaluating differences between adjacent count estimates.

The computation can be performed with a approximation based on the frequency of all reads or fitted explicitly for each segment. On platforms supporting the *parallel* package, the fitting can be spread over multiple processor cores in order to decrease computation time.

```
> yRatio <- normalizeCounts(x)

> yFit <- normalizeCounts(x, fit=TRUE)
> yFit

* Object of class 'TssNorm' *
  Data normalized

** Segments **
   Segments (5): s1_+_1, s1_+_2, s1_-_3, s2_+_4, s2_-_5
   Chromosomes (2): s1, s2
   Strands (2): +, -
   Regions (5): 1, 2, 3, 4, 5
   nCounts (5): 978, 587, 848, 466, 690

** Parameters **
   pattern: %1$s_%2$s_%3$s
   offset: 10
   basal: 1e-04
   lambda: c(0.1, 0.1)
   fit: TRUE
   optimizer: all

> head(reads(yFit, 3))

    start     end counts    ratio       fit
1 1435037 1435037      4 3.478260 3.6515973
2 1435039 1435039      4 3.478260 3.6515964
3 1435043 1435043      3 2.608695 2.8192247
4 1435045 1435045      1 0.000100 0.9775118
5 1435047 1435047      1 0.000100 0.9775118
6 1435049 1435049      1 0.000100 0.9775120

> plot(yFit, 3)
```



5 Identifying transcription start sites

After normalization of the count data, an iterative algorithm is applied for each segment to identify the TSS. The expected number of false positive counts is initialized with a default value given by the read frequency in the whole data set. The position with the largest counts above is identified as a TSS, if the expected transcription level is at least one read above the expected number of false positive reads. The transcription levels for all TSS are calculated by adding all counts to their nearest neighbor TSS.

Then, the expected number of false positive reads is updated by convolution with exponential kernels. The decay rates `tau` in 3' direction and towards the 5'-end can be chosen differently to account for the fact that false positive counts are preferably found in 5' direction of a TSS. This procedure is iterated as long as the set of TSS increases.

```
> z <- identifyStartSites(yFit)
> z

* Object of class 'TssResult' *
  TSS in data identified
```

```
** Segments **
   Segments (5): s1_+_1, s1_+_2, s1_-_3, s2_+_4, s2_-_5
   Chromosomes (2): s1, s2
   Strands (2): +, -
   Regions (5): 1, 2, 3, 4, 5
   nCounts (5): 978, 587, 848, 466, 690
   nTSS (5): 2, 3, 1, 9, 6

** Parameters **
   pattern: %1$s_%2$s_%3$s
   offset: 10
   basal: 1e-04
   lambda: c(0.1, 0.1)
   fit: TRUE
   optimizer: all
   tau: c(20, 20)
   threshold: 1
   fun: function (fg, bg, indTss, pos, basal, tau, extend = FALSE)
{
    idx <- pos - pos[1] + 1
    idxTss <- idx[indTss]
    n <- pos[length(pos)] - pos[1] + 1
    fak <- 1/.exppdf(1, tau)
    win1 <- fak[1] * .exppdf(n:1, tau[1])
    win2 <- fak[2] * .exppdf(1:n, tau[2])
    win <- c(win1, 0, win2)
    bgb <- rep(0, n)
    bgb[idxTss] <- bg[indTss]
    cums <- convolve(win, rev(bgb), type = "open")
    expect <- cums[(n + 1):(length(cums) - n)]
    if (!extend)
        expect <- expect[idx]
    expect[expect < basal] <- basal
    delta <- fg - expect
    delta[delta < 0] <- 0
    res <- list(delta = delta, expect = expect)
    return(res)
}
   readCol: fit
   neighbor: TRUE

> head(segments(z))
```

```
       chr strand region nPos nCounts   start     end nTss
s1_+_1  s1      +      1   28     978   82747   82994    2
s1_+_2  s1      +      2   33     587  814741  815042    3
s1_-_3  s1      -      3   47     848 1435037 1435157    1
s2_+_4  s2      +      4   68     466 1454505 1455353    9
s2_-_5  s2      -      5   43     690 1574882 1575467    6
```

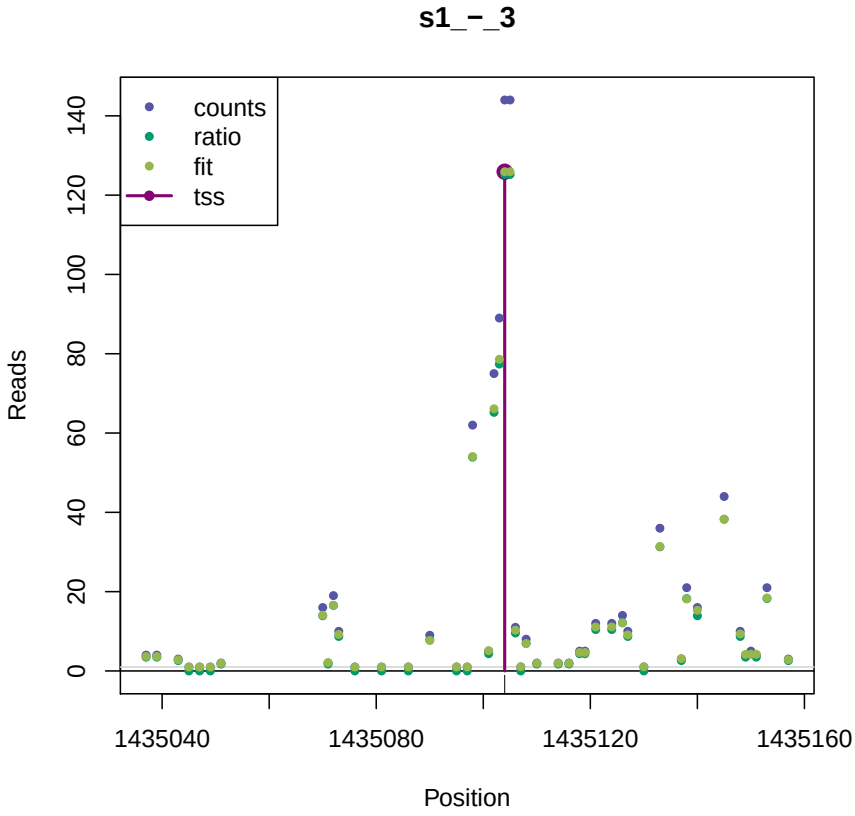
```
> head(tss(z, 3))
```

```
      pos    reads
1 1435104 125.9111
```

```
> head(reads(z, 3))
```

```
    start     end counts    ratio       fit delta   expect
1 1435037 1435037      4 3.478260 3.6515973     0 18.54737
2 1435039 1435039      4 3.478260 3.6515964     0 20.49801
3 1435043 1435043      3 2.608695 2.8192247     0 25.03633
4 1435045 1435045      1 0.000100 0.9775118     0 27.66943
5 1435047 1435047      1 0.000100 0.9775118     0 30.57944
6 1435049 1435049      1 0.000100 0.9775120     0 33.79551
```

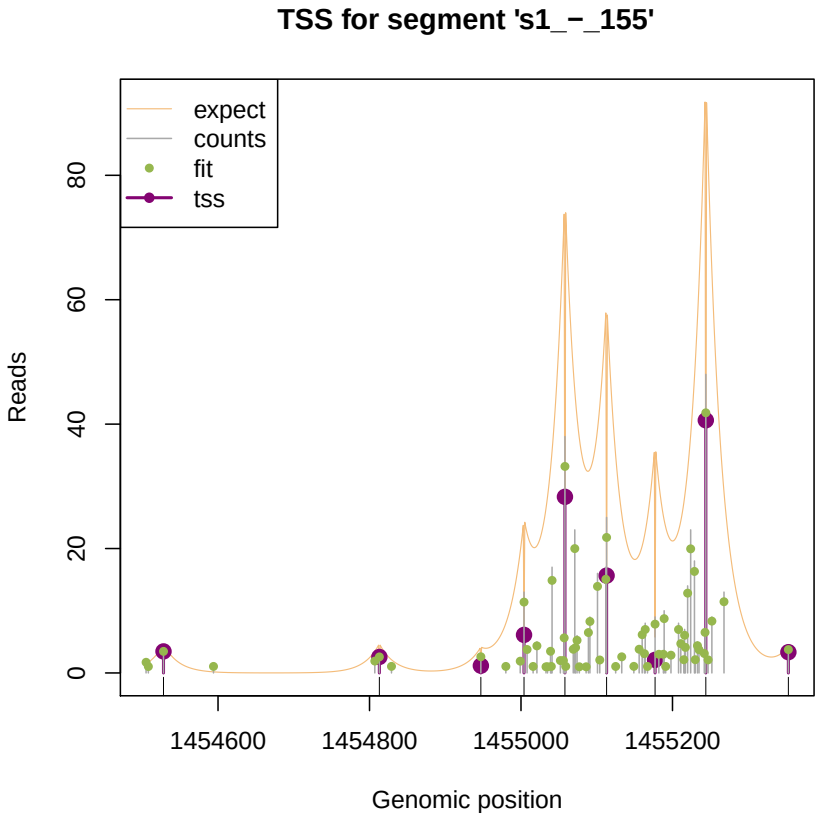
```
> plot(z, 3)
```



6 Visualizing and customizing figures

The *plot* method allows for a simple, but powerful visualization and customization of the produced figures. For each element of the figure, all graphical parameters can be set, supplying them in the form of named lists. In the following, plotting of the threshold and the ratio estimates are omitted, as well as the representation of some components is adapted. For a detailed description on the individual settings, please refer to the *plot* documentation of this package.

```
> plot(z, 4,
+ ratio=FALSE,
+ threshold=FALSE,
+ baseline=FALSE,
+ expect=TRUE, expectArgs=list(type="l"), extend=TRUE,
+ countsArgs=list(type="h", col="darkgray", pch=NA),
+ plotArgs=list(xlab="Genomic position", main="TSS for segment 's1_-_155'"))
```



7 Converting and exporting results

While the get methods *reads*, *segments*, and *tss* provide a simple access to relevant results, such data can also be represented with the framework provided by the *IRanges* package. Converting the data to an object of class *RangedData* allows for a standard representation and interface to other formats, for example using the *rtracklayer* package.

```
> readsRd <- readsAsRangedData(z)
> segmentsRd <- segmentsAsRangedData(z)
> tssRd <- tssAsRangedData(z)

> #library(rtracklayer)
> #tmpFile <- tempfile()
> #export.gff3(readsRd, paste(tmpFile, "gff", sep="."))
> #export.bed(segmentsRd, paste(tmpFile, "bed", sep="."))
> #export.bed(tssRd, paste(tmpFile, "bed", sep="."))
```

Session info

- R version 3.5.0 (2018-04-23), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 16.04.4 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: TSSi 1.26.0
- Loaded via a namespace (and not attached): Biobase 2.40.0, BiocGenerics 0.26.0, Formula 1.2-2, Hmisc 4.1-1, IRanges 2.14.0, Matrix 1.2-14, RColorBrewer 1.1-2, Rcpp 0.12.16, S4Vectors 0.18.0, acepack 1.4.1, backports 1.1.2, base64enc 0.1-3, checkmate 1.8.5, cluster 2.0.7-1, colorspace 1.3-2, compiler 3.5.0, data.table 1.10.4-3, digest 0.6.15, foreign 0.8-70, ggplot2 2.2.1, grid 3.5.0, gridExtra 2.3, gtable 0.2.0, htmlTable 1.11.2, htmltools 0.3.6, htmlwidgets 1.2, knitr 1.20, lattice 0.20-35, latticeExtra 0.6-28, lazyeval 0.2.1, magrittr 1.5, minqa 1.2.4, munsell 0.4.3, nnet 7.3-12, parallel 3.5.0, pillar 1.2.2, plyr 1.8.4, rlang 0.2.0, rpart 4.1-13, rstudioapi 0.7, scales 0.5.0, splines 3.5.0, stats4 3.5.0, stringi 1.1.7, stringr 1.3.0, survival 2.42-3, tibble 1.4.2, tools 3.5.0