SOFTWARE TOOL ARTICLE

A step-by-step guide to analyzing CAGE data using R/Bioconductor [version 1; peer review: awaiting peer review]

Malte Thodberg¹,², Albin Sandelin¹,²

¹Biotech Research and Innovation Centre, University of Copenhagen, Copenhagen, Denmark
²Section for Computational and RNA Biology, University of Copenhagen, Copenhagen, Denmark

Abstract

Cap Analysis of Gene Expression (CAGE) is one of the most popular 5'–end sequencing methods. In a single experiment, CAGE can be used to locate and quantify the expression of both Transcription Start Sites (TSSs) and enhancers. This is workflow is a case study on how to use the CAGEfightR package to orchestrate analysis of CAGE data within the Bioconductor project. This workflow starts from BigWig-files and covers both basic CAGE analyses such as identifying, quantifying and annotating TSSs and enhancers, advanced analysis such as finding interacting TSS-enhancer pairs and enhancer clusters, to differential expression analysis and alternative TSS usage. R-code, discussion and references are intertwined to help provide guidelines for future CAGE studies of the same kind.

Keywords

CAGE, TSS, Enhancer, Promoter, DE, Motifs

This article is included in the Bioconductor gateway.

Corresponding author: Malte Thodberg (maltethodberg@gmail.com)

Author roles: Thodberg M: Conceptualization, Formal Analysis, Investigation, Methodology, Project Administration, Software, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; Sandelin A: Conceptualization, Formal Analysis, Funding Acquisition, Investigation, Methodology, Project Administration, Supervision, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: The authors are the developers of the CAGEfightR package used in the workflow.

Grant information: Work in the Sandelin Lab was supported by the Novo Nordisk Foundation, Lundbeck foundation, Danish Innovation Fund, Danish Cancer Society and Independent Research Fund Denmark.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2019 Thodberg M and Sandelin A. This is an open access article distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Thodberg M and Sandelin A. A step-by-step guide to analyzing CAGE data using R/Bioconductor [version 1; peer review: awaiting peer review] F1000Research 2019, 8:886 (https://doi.org/10.12688/f1000research.18456.1)

First published: 18 Jun 2019, 8:886 (https://doi.org/10.12688/f1000research.18456.1)
Background

Transcriptional regulation is one of the most important aspects of gene expression. Transcription Start Sites (TSSs) are focal points of this process: The TSS act as an integration point for a wide range of molecular cues from surrounding genomic areas to determine transcription and ultimately expression levels. These include proximal factors such as chromatin accessibility, chromatin modification, DNA methylation and transcription factor binding, and distal factors such as enhancer activity and chromatin confirmation.

Cap Analysis of Gene Expression (CAGE) has emerged as one of the dominant high-throughput assays for studying TSSs. CAGE is based on “cap trapping”: capturing capped full-length RNAs and sequencing only the first 20–30 nucleotides from the 5’-end, so-called CAGE tags. When mapped to a reference genome, the 5’-ends of CAGE tags identify the actual TSS for respective RNA with basepair-level accuracy. Basepair-accurate TSSs identified this way are referred to as CAGE Transcription Start Sites (CTSSs). RNA polymerase rarely initiates from just a single nucleotide: this is manifested in CAGE data by the fact that CTSSs are mostly found in tightly spaced groups on the same strand. The majority of CAGE studies have merged such CTSSs into genomic blocks typically referred to as Tag Clusters (TCs), using a variety of clustering methods (see below). TCs are often interpreted as TSSs in the more general sense, given that most genes have many CTSSs, but only a few TCs that represent a few major transcripts with highly similar CTSSs. Since the number of mapped CAGE tags in a given TC is indicative of the number of RNAs from that region, the number of CAGE tags falling in given TC can be seen as a measure of expression.

As CAGE tags can be mapped to a reference genome without the need for transcript annotations, it can detect TSSs of known mRNAs, but also mRNA from novel alternative TSSs (that might be condition or tissue dependent). Since CAGE captures all capped RNAs, it can also identify long non-coding RNA (lincRNA) and enhancers RNA (eRNA). It was previously shown that active enhancers are characterized by balanced bidirectional transcription, making it possible to predict enhancer regions and quantify their expression levels from CAGE data alone. Thus, CAGE data can predict the locations and activity of mRNAs, lincRNAs and enhancers in a single assay, providing a comprehensive view of transcriptional regulation across both inter- and intragenic regions.

Bioconductor contains a vast collection of tools for analyzing transcriptomics datasets, in particular the widely used RNA-Seq and microarray assays. Only a few packages are dedicated to analyzing 5’-end data in general and CAGE data in particular: TSRchitect, icetea, CAGEr and CAGEfightR, see Table 1.

CAGEr was the first package solely dedicated to the analysis of CAGE data and was recently updated to more closely adhere to Bioconductor S4-class standards. CAGEr takes as input aligned reads in the form of BAM-files and can identify, quantify, characterize and annotate TSSs. TSSs are found in individual samples using either simple clustering of CTSSs (greedy or distance-based clustering) or the more advanced density-based paraclu clustering method, and can be aggregated across samples to a set of consensus clusters. Several specialized routines for CAGE data is available, such as power law normalization of CTSS counts and fine-grained TSS shifts. Finally, CAGEr offers easy interface to the large collection of CAGE data from the FANTOM consortium.

<table>
<thead>
<tr>
<th>Table 1. Comparison of Bioconductor packages for CAGE data analysis.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analysis</strong></td>
</tr>
<tr>
<td>Simplest input</td>
</tr>
<tr>
<td>TSS calling</td>
</tr>
<tr>
<td>TSS shapes</td>
</tr>
<tr>
<td>Differential Expression</td>
</tr>
<tr>
<td>Enhancer calling</td>
</tr>
<tr>
<td>TSS-enhancer correlation</td>
</tr>
<tr>
<td>Super enhancers</td>
</tr>
</tbody>
</table>
icetea are two more recent additions to Bioconductor. While being less comprehensive, they aim to be more
general and handle more types of 5’-end methods that are conceptually similar to CAGE (RAMPAGE, PEAT,
PRO-Cap, etc.). Both packages can identify, quantify and annotate TSSs, with TSRchitect using an X-means
algorithm and icetea using a sliding window approach. icetea offers the unique feature of mapping reads
to a reference genome by interfacing with Rsubread. Both CAGEr, TSRchictet and icetea offers built-in
capabilities for differential expression (DE) analysis via the popular DESeq2 or edgeR packages.20,21.

CAGEfightR is a recent addition to Bioconductor focused on analyzing CAGE data, but applicable to most 5’-end
data. It aims to be general and flexible to allow for easy interfacing with the wealth of other Bioconductor
packages. CAGEfightR takes CTSSs stored in BigWig-files as input and uses only standard Bioconductor
S4-classes (GenomicRanges, SummarizedExperiment, InteractionSet) making it easy for users to learn and
combine CAGEfightR with functions from other Bioconductor packages (e.g. instead of providing custom wrap-
ners around other packages such as differential expression analysis). In addition to TSS analysis, CAGEfightR is
the only package on Bioconductor to also offer functions for enhancer analysis based on CAGE (and similarly
scoped) data. This includes enhancer identification and quantification, linking enhancers to TSSs via correlation
of expression and finding enhancer clusters, often referred to as stretch- or super enhancers.

In this workflow, we illustrate how the CAGEfightR package can be used to orchestrate an end-to-end analysis
of CAGE data by making it easy to interface with a wide range of different Bioconductor packages. Highlights
include TSS and enhancer candidate identification, differential expression, alternative TSS usage, enrichment of
motifs, GO/KEGG terms and calculating TSS-enhancer correlations.

Methods

Dataset
This workflow uses data from “Identification of Gene Transcription Start Sites and Enhancers Responding to
Pulmonary Carbon Nanotube Exposure in Vivo” by Bornholdt et al.24. This study uses mouse as a model system
to investigate how nanotubes affect lung tissue when inhaled. Inhaled nanotubes were previously found to produce
a similar response to asbestos, potentially triggering an inflammatory response in the lung tissue leading to
drastic changes in gene expression.

The dataset consists of CAGE data from mouse lung biopsies: 5 mice whose lungs were instilled with water (Ctrl)
and 6 mice whose lungs were instilled with nanotubes (Nano), with CTSSs for each sample stored in
BigWig-files, shown in Table 2:

<table>
<thead>
<tr>
<th>Group</th>
<th>Biological Replicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl</td>
<td>5 mice</td>
</tr>
<tr>
<td>Nano</td>
<td>6 mice</td>
</tr>
</tbody>
</table>

The data is acquired via the nanotubes data package:

```r
library(nanotubes)
```

R-packages
This workflow uses a large number of R-packages: Bioconductor packages are primarily used for data analysis
while packages from the tidyverse are used to wrangle and plot the results. All these packages are loaded prior to
beginning the workflow:
# CRAN packages for data manipulation and plotting
library(knitr)
library(kableExtra)
library(pheatmap)
library(ggseqlogo)
library(viridis)
library(magrittr)
library(ggforce)
library(ggthemes)
library(tidyverse)

# CAGEfightR and related packages
library(CAGEfightR)
library(GenomicRanges)
library(SummarizedExperiment)
library(GenomicFeatures)
library(BiocParallel)
library(InteractionSet)
library(Gviz)

# Bioconductor packages for differential expression
library(DESeq2)
library(limma)
library(edgeR)
library(sva)

# Bioconductor packages for enrichment analyses
library(TFBSTools)
library(motifmatchr)
library(pathview)

# Bioconductor data packages
library(BSgenome.Mmusculus.UCSC.mm9)
library(TxDb.Mmusculus.UCSC.mm9.knownGene)
library(org.Mm.eg.db)
library(JASPAR2016)

We also set some script-wide settings for later convenience:

```r
# Rename these for easier access
bsg <- BSgenome.Mmusculus.UCSC.mm9
txdb <- TxDb.Mmusculus.UCSC.mm9.knownGene
odb <- org.Mm.eg.db

# Script wide settings
register(MulticoreParam(3)) # Parallel execution when possible
theme_set(theme_light()) # White theme for ggplot2 figures
```

**Workflow**
The workflow is divided into 3 parts covering different parts of a typical CAGE data analysis:

1. Shows how to use CAGEfightR to import CTSSs and find and quantify TSS and enhancer candidates.
2. Shows examples of how to perform genomic analyses of CAGE clusters using core Bioconductor packages such as GenomicRanges and Biostrings. This part covers typical analyses made from CAGE data, from summarizing cluster annotation, TSS shapes and core promoter sequence analysis to more advanced spatial analyses (finding TSS-enhancer correlation links and clusters of enhancers).
3. Shows how CAGEfightR can be used to prepare data for differential expression analysis with popular R packages, including DESeq2, limma and edger. Borrowing from RNA-Seq terminology, differential expression can be assessed at multiple different levels: Tag cluster- and enhancer-level, gene-level and differential TSS usage. Once differential expression results have been obtained, they can be combined with other sources of information such as motifs from JASPAR and GO/KEGG terms.

**Part 1: Locating, quantifying and annotating TSSs and enhancers**

CAGEfightR starts analysis from CTSSs, which is the number of CAGE tag 5’-ends mapping to each basepair (bp) in the genome. CTSSs are normally stored as either BED-files or BigWig-files. CAGEfightR works on BigWig-files, since these can be efficiently imported and allow for random access.

Before starting the analysis, we recommend gathering all information (Filenames, groups, batches, preparation data, etc.) about the samples to be analyzed in a single data.frame, sometimes called the design matrix. CAGEfightR can keep track of the design matrix throughout the analysis:

```r
data(nanotubes)
kable(nanotubes, caption = "The initial design matrix for the nanotubes experiment") %>% kable_styling(latex_options = "hold_position")
```

### Table 3. The initial design matrix for the nanotubes experiment.

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
<th>BigWigPlus</th>
<th>BigWigMinus</th>
</tr>
</thead>
<tbody>
<tr>
<td>C547</td>
<td>Ctrl</td>
<td>mm9.CAGE_7J7P_NANO_KON_547.plus.bw</td>
<td>mm9.CAGE_7J7P_NANO_KON_547.minus.bw</td>
</tr>
<tr>
<td>C548</td>
<td>Ctrl</td>
<td>mm9.CAGE_ULAC_NANO_KON_548.plus.bw</td>
<td>mm9.CAGE_ULAC_NANO_KON_548.minus.bw</td>
</tr>
<tr>
<td>C549</td>
<td>Ctrl</td>
<td>mm9.CAGE_YM4F_Nano_KON_549.plus.bw</td>
<td>mm9.CAGE_YM4F_Nano_KON_549.minus.bw</td>
</tr>
<tr>
<td>C559</td>
<td>Ctrl</td>
<td>mm9.CAGE_RSAM_NANO_559.plus.bw</td>
<td>mm9.CAGE_RSAM_NANO_559.minus.bw</td>
</tr>
<tr>
<td>C560</td>
<td>Ctrl</td>
<td>mm9.CAGE_CCLF_NANO_560.plus.bw</td>
<td>mm9.CAGE_CCLF_NANO_560.minus.bw</td>
</tr>
<tr>
<td>N14</td>
<td>Nano</td>
<td>mm9.CAGE_RSAM_NANO_14.plus.bw</td>
<td>mm9.CAGE_RSAM_NANO_14.minus.bw</td>
</tr>
<tr>
<td>N15</td>
<td>Nano</td>
<td>mm9.CAGE_RFQS_Nano_15.plus.bw</td>
<td>mm9.CAGE_RFQS_Nano_15.minus.bw</td>
</tr>
<tr>
<td>N16</td>
<td>Nano</td>
<td>mm9.CAGE_CCLF_NANO_16.plus.bw</td>
<td>mm9.CAGE_CCLF_NANO_16.minus.bw</td>
</tr>
<tr>
<td>N17</td>
<td>Nano</td>
<td>mm9.CAGE_RSAM_NANO_17.plus.bw</td>
<td>mm9.CAGE_RSAM_NANO_17.minus.bw</td>
</tr>
<tr>
<td>N18</td>
<td>Nano</td>
<td>mm9.CAGE_CCLF_NANO_18.plus.bw</td>
<td>mm9.CAGE_CCLF_NANO_18.minus.bw</td>
</tr>
</tbody>
</table>

**Importing CTSSs.** We need to tell CAGEfightR where to find the BigWig-files containing CTSSs on the hard drive. Normally, one would supply the paths to each file (e.g. /CAGEdata/BigWigFiles/Sample1_plus.bw), but here we will use data from the nanotubes data package:

```r
# Setup paths to file on hard drive
bw_plus <- system.file("extdata", nanotubes$BigWigPlus, package = "nanotubes", mustWork = TRUE)
bw_minus <- system.file("extdata", nanotubes$BigWigMinus, package = "nanotubes", mustWork = TRUE)

# Save as named BigWigFileList
bw_plus <- BigWigFileList(bw_plus)
bw_minus <- BigWigFileList(bw_minus)
names(bw_plus) <- names(bw_minus) <- nanotubes$Name
```
The first step is quantifying CTSS usage across all samples. This is often one of the most time consuming step in a CAGEfightR analysis, but it can be speed up by using multiple cores (if available, see Materials and Methods). We also need to specify the genome, which we can get from the `BSgenome.Mmuspculus.UCSC.mm9` genome package:

```r
CTSSs <- quantifyCTSSs(plusStrand = bw_plus,
                         minusStrand = bw_minus,
                         genome = seqinfo(bsg),
                         design = nanotubes)
```

#> Checking supplied genome compatibility...
#> Iterating over 28 genomic tiles in 11 samples using 3 worker(s)...
#> Importing CTSSs from plus strand...
#> Registered S3 method overwritten by 'pryr':
#>   method from
#>   print.bytes Rcpp
#> Importing CTSSs from minus strand...
#> Merging strands...
#> ### CTSS summary ###
#> Number of samples: 11
#> Number of CTSSs: 9.339 millions
#> Sparsity: 81.68 %
#> Final object size: 282 MB

The circa 9 million CTSSs are stored as `RangedSummarizedExperiment`, which is the standard representation of high-throughput experiments in Bioconductor. We can inspect both the ranges and the CTSS counts:

```r
# Get a summary
CTSSs
```

#> class: RangedSummarizedExperiment
#> dim: 9338802 11
#> metadata(0):
#> assays(1): counts
#> rownames: NULL
#> rowData names(0):
#> colnames(11): C547 C548 ... N17 N18
#> colData names(4): Class Name BigWigPlus BigWigMinus

# Extract CTSS positions
rowRanges(CTSSs)
```

#> GPos object with 9338802 positions and 0 metadata columns:
#> seqlines     pos strand
#> [1] chr1 3024556 +
#> [2] chr1 3025704 +
#> [3] chr1 3025705 +
#> [4] chr1 3028283 +
#> [5] chr1 3146133 +
#> ... ... ... ...
#> [9338798] chrUn_random 5810899 -
#> [9338799] chrUn_random 5813784 -
#> [9338800] chrUn_random 5880838 -
#> [9338801] chrUn_random 5893536 -
#> [9338802] chrUn_random 5894263 -
#> -------
#> seqinfo: 35 sequences (1 circular) from mm9 genome
Unidirectional and bidirectional clustering for finding TSS and enhancer candidates. CAGEfightR finds clusters by calculating the pooled CTSS signal across all samples: We first normalize CTSSs count in each sample to Tags-Per-Million (TPM) values, and then sum TPM values across samples:

```
CTSSs <- CTSSs %>%
calcTPM() %>%
calcPooled()
```

This will add several new pieces of information to `CTSSs`: The total number of tags in each library, a new assay called TPM, and the pooled signal for each CTSS.

We can use unidirectional clustering to locate unidirectional clusters, often simply called Tag Clusters (TCs), which are candidates for TSSs. The `quickTSSs` will both locate and quantify TCs in a single function call:

```
TCs <- quickTSSs(CTSSs)
```

Note: `quickTSSs` runs CAGEfightR with default settings. If you have larger or more noisy datasets you most likely want to do a more robust analysis with different settings. See the CAGEfightR vignette for more information.

Many of the identified TCs will only be very lowly expressed. To obtain likely biologically relevant TSSs, we keep only TSSs expressed at more than 1 TPM in at least 5 samples (5 samples being the size of the smallest experimental group):
TSSs <- TCs %>%
calcTPM() %>%
subsetBySupport(inputAssay="TPM",
  unexpressed=1,
  minSamples=4)

#> Calculating library sizes...
#> Warning in calcTotalTags(object = object, inputAssay = inputAssay,
#> outputColumn = outputColumn): object already has a column named totalTags
#> in colData: It will be overwritten!
#> Calculating TPM...
#> Calculating support...
#> Subsetting...
#> Removed 3573214 out of 3602099 regions (99.2%)

This removed a large number of very lowly expressed TCs, leaving us with almost 30,000 TSSs candidates for analysis.

Then we turn to bidirectional clustering for identifying bidirectional clusters (BCs), which are candidate for enhancers. Similarly, we can use quickEnhancers to locate and quantify BCs:

BCs <- quickEnhancers(CTSSs)

#> Using existing score column!
#> - Running clusterBidirectionally:
#>  Pre-filtering bidirectional candidate regions...
#>  Retaining for analysis: 68.3%
#>  Splitting by strand...
#>  Calculating windowed coverage on plus strand...
#>  Calculating windowed coverage on minus strand...
#>  Calculating balance score...
#>  Slice-reduce to find bidirectional clusters...
#>  Calculating statistics...
#>  Preparing output...
#>  # Bidirectional clustering summary:
#>  Number of bidirectional clusters: 106779
#>  Maximum balance score: 1
#>  Minimum balance score: 0.950001090872574
#>  Maximum width: 1866
#>  Minimum width: 401
#>
#>  - Running subsetByBidirectionality:
#>  Calculating bidirectionality...
#>  Subsetting...
#>  Removed 73250 out of 106779 regions (68.6%)
#>
#>  - Running quantifyClusters:
#>  Finding overlaps...
#>  Aggregating within clusters...

Note: quickEnhancers runs CAGEfightR with default settings. If you have larger or more noisy datasets you most likely want to do a more robust analysis with different settings. See the CAGEfightR vignette for more information.

Again, we are not usually interested in very lowly expressed BCs. As they are overall lowly expressed, we will simply filter out BCs without at least 1 count in 5 samples:
Annotating clusters with transcript models. After having located unidirectional and bidirectional clusters, we can annotate them according to known transcript and gene models. These types of annotation are store via TxDb-objects in Bioconductor. Here we will simply use UCSC transcripts included in the `TxDb.Mmuseulcus.UCSC.mm9.knownGene` package, but the CAGEfightR vignette includes examples of how to obtain a TxDb object from other sources (GFF/GTF files, AnnotationHub, etc.).

Starting with the TSS candidates, we can not only annotate a TSS with overlapping transcripts, but also in what part of a transcript a TSS lies by using a hierarchical annotation scheme. As some TSSs might be very wide, we only use the TSS peak for annotation purposes:

```r
# Annotate with transcript IDs
TSSs <- assignTxID(TSSs, txModels = txdb, swap="thick")
#> Extracting transcripts...
#> Finding hierarchical overlaps...
#> ### Overlap Summary: ###
#> Features overlapping transcripts: 87.65 %
#> Number of unique transcripts: 31898

# Annotate with transcript context
TSSs <- assignTxType(TSSs, txModels = txdb, swap="thick")
#> Finding hierarchical overlaps with swapped ranges...
#> ### Overlap summary: ###
#> txType count percentage
#> 1  promoter  13395  46.4
#> 2  proximal   2246   7.8
#> 3 fiveUTR    2112   7.3
#> 4 threeUTR   1200   4.2
#> 5    CDS     3356  11.6
#> 6     exon    161   0.6
#> 7    intron   2810   9.7
#> 8  antisense  1294   4.5
#> 9  intergenic 2311   8.0
```

Almost half of TSSs were found at annotated promoters, while the other half is novel compared to the UCSC known transcripts.

Transcript annotation is particularly useful for enhancer candidates, as bidirectional clustering might also detect bidirectional promoters. Therefore, a commonly used filtering approached is to only consider BCs in intergenic or intronic regions as enhancer candidates:

```r
# Annotate with transcript context
BCs <- assignTxType(BCs, txModels = txdb, swap="thick")
#> Finding hierarchical overlaps with swapped ranges...
#> ### Overlap summary: ###
#> txType count percentage
#> 1 promoter  766  5.7
#> 2 proximal  1649 12.2
#> 3 fiveUTR   281  0.5
#> 4 threeUTR  596  4.4
```
> 5  CDS  420    3.1
> 6  exon  71     0.5
> 7  intron 6815   50.4
> 8  antisense 0     0.0
> 9  intergenic 3128  23.1

# Keep only non-exonic BCs as enhancer candidates
Enhancers <- subset(BCs, txType %in% c("intergenic", "intron"))

This leaves almost 10000 enhancer candidates for analysis.

**Merging into a single dataset.** For many downstream analyses, in particular normalization and differential expression, it is useful to combine both TSS and enhancer candidates into a single dataset. This ensures that TSSs and enhancers do not overlap, so each CAGE tag is only counted once.

We must first ensure that the enhancer and TSS candidates have the same information attached to them, since CAGEfightR will only allow merging of clusters if they have the same sample and cluster information:

```
# Clean colData
TSSs$totalTags <- NULL
Enhancers$totalTags <- NULL

# Clean rowData
rowData(TSSs)$balance <- NA
rowData(TSSs)$bidirectionality <- NA
rowData(Enhancers)$txID <- NA

# Add labels for making later retrieval easy
rowData(TSSs)$clusterType <- "TSS"
rowData(Enhancers)$clusterType <- "Enhancer"
```

Then the clusters can be merged: As enhancers are the most complicated type, we keep only enhancers if a TSS and enhancer overlaps:

```
RSE <- combineClusters(object1=TSSs, 
                       object2 = Enhancers, 
                       removeIfOverlapping="object1")

# Removing overlapping features from object1: 374
# Keeping assays: counts
# Keeping columns: score, thick, support, txID, txType, balance, bidirectionality, clusterType
# Merging metadata...
# Stacking and re-sorting...
```

We finally calculate the total number of tags and TPM-scaled counts for the final merged dataset:

```
RSE <- calcTPM(RSE)

# Calculating library sizes...
# Calculating TPM...
```

**Part 2: Genomic analysis of TSSs and enhancers**

*Genome-browser figures of TSSs and enhancers.* First we can simply plot some examples of TSSs and enhancers in a genome browser style figure using the Gviz package\cite{31}. It takes a bit of code to setup, but the resulting tracks can be reused for later examples:
A good general strategy for quickly generating genome browser plots is to first define a region of interest, and then only plotting data within that region using `subsetByOverlaps`. The following code demonstrates this using the first TSS:

```r
# Extract 100 bp around the first TSS.
plot_region <- RSE %>%
  rowRanges %>%
  subset(clusterType == "TSS") %>%
  [1] %>%
  add(100) %>%
  unstrand()

# CTSSs track
ctss_track <- CTSSs %>%
  rowRanges %>%
  subsetByOverlaps(plot_region) %>%
  trackCTSS(name = "CTSSs")

# Cluster track
cluster_track <- RSE %>%
  subsetByOverlaps(plot_region) %>%
  trackClusters(name = "Clusters",
                col = NA,
                showId=TRUE)

# Plot at tracks together
plotTracks(list(axis_track,
                ctss_track,
                cluster_track,
                tx_track),
           from = start(plot_region),
           to=end(plot_region),
           chromosome = seqnames(plot_region))
```
Figure 1. Genome browser example of TSS candidate.

The top track shows the pooled CTSS signal and the middle track shows the identified TC with the thick bar indicating the TSS peak (the overall most used CTSSs within the TC). The bottom track shows the known transcript model at this genomic location. In this case, the CAGE-defined TSS corresponds well to the annotation.

We can also plot the first enhancer:

```r
# Make plotting region
plot_region <- RSE %>%
  rowRanges %>%
  subset(clusterType == "Enhancer") %>%
  [1] %>%
  add(100) %>%
  unstrand()

# CTSSs track
ctss_track <- CTSSs %>%
  rowRanges %>%
  subsetByOverlaps(plot_region) %>%
```

```
trackCTSS(name = "CTSSs")

# Splitting pooled signal by strand...
# Preparing track...

# Cluster track
cluster_track <- RSE %>%
  rowRanges %>%
  subsetByOverlaps(plot_region) %>%
  trackClusters(name = "Clusters",
                col = NA,
                showId=TRUE)

# Setting thick and thin features...
# Merging and sorting...
# Preparing track...

# Plot at tracks together
plotTracks(list(axis_track, ctss_track, cluster_track, tx_track),
           from = start(plot_region),
           to = end(plot_region),
           chromosome = as.character(seqnames(plot_region)))

Figure 2. Genome browser example of enhancer candidate.
Here we see the bidirectional pattern characteristic of active enhancers. The bidirectional cluster is seen in the middle track, with the midpoint in thick marking the maximally balanced point within the bidirectional cluster.

**Location and expression of TSSs and enhancers.** In addition to looking at single examples of TSSs and enhancers, we also want to get an overview of the number and expression of clusters in relation to transcript annotation. First we extract all of the necessary data from the `RangedSummarizedExperiment` into an ordinary `data.frame`:

```r
cluster_info <- RSE %>%
  rowData() %>%
  as.data.frame()
```

Then we use `ggplot2` to plot the number and expression levels of clusters in each annotation category:

```r
# Number of clusters
ggplot(cluster_info, aes(x=txType, fill=clusterType)) +
g  geom_bar(alpha=0.75, position="dodge", color="black") +
  scale_fill_colorblind("Cluster type") +
  labs(x="Cluster annotation", y="Frequency") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

![Graph showing number of clusters within each annotation category.](image)

**Figure 3.** Number of clusters within each annotation category.

```r
# Expression of clusters
ggplot(cluster_info, aes(x=txType,
  y=log2(score/ncol(RSE)),
  fill=clusterType)) +
g  geom_violin(alpha=0.75, draw_quantiles = c(0.25, 0.50, 0.75)) +
  scale_fill_colorblind("Cluster type") +
  labs(x="Cluster annotation", y="log2(TPM)") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

```r
#> Warning in regularize.values(x, y, ties, missing(ties)): collapsing to #> unique 'x' values
```
We find that TSSs at annotated promoters are generally highly expressed. Most novel TSSs are expressed at lower levels, except for some TSSs in 5’-UTRs. Enhancers are expressed at much lower levels than TSSs.

**Analysing TSS shapes and sequences.** A classic analysis of CAGE data is to divide TSSs into *Sharp* and *Broad* classes, which show different core promoter regions and different expression patterns across tissues.

CAGEfightR can calculate several *shape statistics* that summarizes the shape of a TSS. The Interquartile Range (IQR) can be used to find sharp and broad TSSs. As lowly expressed TSSs cannot show much variation in shape due to their low width and number of tags, we here focused on highly expressed TSSs (average TPM >= 10):

```r
# Select highly expressed TSSs
highTSSs <- subset(RSE, clusterType == 'TSS' & score / ncol(RSE) >= 10)

# Calculate IQR as 10%-90% interval
highTSSs <- calcShape(highTSSs, pooled=CTSSs, shapeFunction=shapeIQR, lower = 0.10, upper = 0.90)

# Splitting by strand...
# Applying function to each cluster...
# Preparing output output...
```

We can then plot the bimodal distribution of IQRs. We use a zoom-in panel to highlight the distinction between the two classes:

```r
highTSSs %>%
  rowData %>%
  as.data.frame %>%
  ggplot(aes(x=IQR)) +
  geom_histogram(binwidth=1, fill="hotpink", alpha=0.75) +
  geom_vline(xintercept = 10, linetype="dashed", alpha=0.75, color="black") +
  facet_zoom(xlim = c(0,100)) +
  labs(x="10-90% IQR", y="Frequency")
```

---

**Figure 4.** Expression of clusters within each annotation category.

![Expression of clusters within each annotation category.](image-url)
We see most TSSs are either below or above 10 bp IQR (dashed line), so we use this cutoff to classify TSSs into Sharp and Broad:

```r
# Divide into groups
rowData(highTSSs)$shape <- ifelse(rowData(highTSSs)$IQR < 10, "Sharp", "Broad")

# Count group sizes
table(rowData(highTSSs)$shape)
#> Broad Sharp
#>  9555   812
```

We can now investigate the core promoters sequences of the two classes of TSSs. We first need to extract the sequences for each TSS: We define this as the TSS peak -40/+10 bp and extract them from using the `BSgenome.Musculus.UCSC.mm10` genome package:

```r
promoter_seqs <- highTSSs %>%
rowRanges() %>%
swapRanges() %>%
promoters(upstream=40, downstream=10) %>%
getSeq(bsg, .)
```

This returns a `DNAStringSet`-object which we can plot as a sequence logo via the `ggseqlogo` package:

```r
promoter_seqs %>%
as.character %>%
split(rowData(highTSSs)$shape) %>%
ggseqlogo(data=., ncol=2, nrow=1) +
theme_logo() +
theme(axis.title.x=element_blank(),
      axis.text.x=element_blank(),
      axis.ticks.x=element_blank())
```

**Figure 5.** Bimodal distribution of Interquartile Ranges (IQRs) of highly expressed TSSs.
As expected, we observe that Sharp TSSs tend to have a stronger TATA-box upstream of the TSS compared to Broad TSSs.

Finding candidates for interacting TSSs and enhancers. In addition to simply identifying enhancers, it is also interesting to try infer what genes they might be regulating. CAGE data can itself not provide direct evidence that an enhancer is physically interacting with a TSS, which would requires specialized chromatin confirmation capture assays such as HiC, 4C, 5C, etc. However, previous studies have shown that TSSs and enhancers that are close to each other and have highly correlated expression are more likely to be interacting. We can therefore use distance and correlation of expression between TSSs and enhancers to identify TSSs-enhancer links as candidates for physical interactions.

To do this with CAGEfightR, we first need to indicate the two types of clusters as a factor with two levels:

```r
rowData(RSE)$clusterType <- RSE %>% rowData() %>%
  use_series("clusterType") %>%
  as_factor() %>%
  fct_relevel("TSS")
```

We can then calculate all pairwise correlations between TSSs and enhancer within a distance of 50 bp. Here we use the non-parametric Kendall’s tau as a measure of correlation, but other functions for calculating correlation can be supplied (e.g. one could calculate Pearson’s r on log-transformed TPM values to only capture linear relationships):

```r
all_links <- RSE %>%
  swapRanges %>%
  findLinks(maxDist = 5e4L,
            directional="clusterType",
            inputAssay="TPM",
            method="kendall")
#> Finding directional links from TSS to Enhancer...
#> Calculating 41311 pairwise correlations...
#> Preparing output...
#> # Link summary:
#> Number of links: 41311
```
The output is a GInteractions-object from the InteractionSet package: For each TSS-enhancer both the distance and orientation (upstream/downstream relative to TSS) is calculated in addition to the correlation estimate and p-value. For now, we are only interested in positive correlations, so we subset and sort the links:

```r
# Subset to only positive correlation
cor_links <- subset(all_links, estimate > 0)
# Sort based on correlation
cor_links <- cor_links[order(cor_links$estimate, decreasing = TRUE)]
```

We can then visualize the correlation patterns across a genomic region, here using the most correlated TSS-enhancer link:

```r
# Make plotting region
plot_region <- cor_links[1] %>%
  anchors %>%
  GRangesList() %>%
  unlist() %>%
  reduce(ignore.strand=TRUE,
         min.gapwidth=1e5) %>%
  add(1000)
```
# Cluster track
```r
cluster_track <- RSE %>%
  subsetByOverlaps(plot_region) %>%
  trackClusters(name = "Clusters",
                col = NA,
                showId=TRUE)
```

> Setting thick and thin features...
> Merging and sorting...
> Preparing track...

# Cluster track
```r
link_track <- cor_links %>%
  subsetByOverlaps(plot_region) %>%
  trackLinks(name="Links",
             interaction.measure = "p.value",
             interaction.dimension.transform = "log",
             col.outside="grey",
             plot.anchors=FALSE,
             col.interactions="black")
```

# Plot at tracks together
```r
plotTracks(list(axis_track,
                 link_track,
                 cluster_track,
                 tx_track),
            from = start(plot_region),
            to=end(plot_region),
            chromosome = as.character(seqnames(plot_region)))
```

**Figure 7.** Genome browser example of TSS-enhancer link candidates.
The top track shows the strength of correlations between 3 TSSs around the Atp1b1 gene. The highest correlation is seen between the upstream TSS and the most distal enhancer.

**Finding stretches of enhancers.** Several studies have found that groups or stretches of closely spaced enhancers tend to show different chromatin characteristics and functions compared to singleton enhancers. Such groups are often referred to as “super enhancers” or “stretch enhancers”.

CAGEfightR can detect such *enhancer stretches* based on CAGE data. CAGEfightR groups nearby enhancers into groups and calculates the average pairwise correlation between them, shown below (again using Kendall’s tau):

```r
# Subset to only enhancers
Enhancers <- subset(RSE, clusterType == "Enhancer")
# Find stretches
stretches <- findStretches(Enhancers,
                           inputAssay = "TPM",
                           mergeDist = 12500L,
                           minSize = 5,
                           method = "kendall")
#> Finding stretches...
#> Calculating correlations...
#> # Stretch summary:
#> Number of stretches: 95
#> Total number of clusters inside stretches: 587 / 9943
#> Minimum clusters: 5
#> Maximum clusters: 15
#> Minimum width: 7147
#> Maximum width: 92531
#> Summary of average pairwise correlations:
#>     Min.  1st Qu.   Median     Mean  3rd   Qu.      Max.
#> -0.10038  0.01351  0.08107  0.09097  0.16171  0.37105
```

Similarly to TSSs and enhancers, we can also annotate stretches based on their relation with known transcripts:

```r
# Annotate
stretches <- assignTxType(stretches, txModels=txdb)
# Finding hierarchical overlaps...
# ### Overlap summary: ###
# txType count percentage
# 1 promoter 50 52.6
# 2 proximal 0 0.0
# 3 fiveUTR 6 6.3
# 4 threeUTR 5 5.3
# 5 CDS 3 3.2
# 6 exon 2 2.1
# 7 intron 15 15.8
# 8 antisense 0 0.0
# 9 intergenic 14 14.7

# Sort by correlation
stretches <- stretches[order(stretches$aveCor, decreasing=TRUE)]
# Inspect
stretches
```

---

Page 20 of 42
The returned GRanges contains the location, number of enhancers and average correlation for each stretch. Stretches are found in a variety of context, some being intergenic and other spanning various parts of genes. Let us plot one of the top intergenic stretches:

```r
# Make plotting region
plot_region <- stretches["chr17:26666593-26675486"] + 1000

# Cluster track
cluster_track <- RSE %>%
  subsetByOverlaps(plot_region) %>%
  trackClusters(name = "Clusters",
                col = NA,
                showId=TRUE)

# Setting thick and thin features...
# Merging and sorting...
# Preparing track...
```
# CTSS track
ctss_track <- CTSSs %>%
  subsetByOverlaps(plot_region) %>%
  trackCTSS(name="CTSSs")
#> Splitting pooled signal by strand...
#> Preparing track...

# SE track
stretch_track <- stretches %>%
  subsetByOverlaps(plot_region) %>%
  AnnotationTrack(name="Stretches", fill="hotpink", col=NULL)

# Plot at tracks together
plotTracks(list(axis_track, stretch_track, cluster_track, ctss_track),
from = start(plot_region),
to=end(plot_region),
chromosome = as.character(seqnames(plot_region)))

Figure 8. Genome browser example of enhancer stretch.
This stretch is composed of at least 5 enhancers, each of which shows bidirectional transcription.

**Part 3: Differential Expression analysis of TSSs, enhancers and genes**

*Normalization of expression and EDA.* Before performing statistical tests for various measures of Differential Expression (DE), it is important to first conduct a thorough Exploratory Data Analysis (EDA) to identify what factor we need to include in the final model.

Here we will use *DESeq2* for normalization and EDA since it offers easy to use functions for performing basic analyses. Other popular tools such as *edgeR* and *limma* offer similar functionality, as well as more specialized packages for EDA such as *EDASeq*.

*DESeq2* offers sophisticated normalization and transformation of count data in the form of the variance stabilized transformation: this adds a dynamic pseudo-count to normalized expression values before log transforming to dampen the inherent mean-variance relationship of count data. This is particularly useful for CAGE data, as CAGE can detect even very lowly expressed TSSs and enhancers.

First, we fit a “blind” version of the variance-stabilizing transformation, since we do not yet know what design is appropriate for this particular study:

```r
# Create DESeq2 object with blank design
dds_blind <- DESeqDataSet(RSE, design = ~ 1)

# Normalize and log transform
vst_blind <- vst(dds_blind, blind = TRUE)
```

A very useful first representation is a Principal Components Analysis (PCA) plot summarizing variance across the entire experiment:

```r
plotPCA(vst_blind, "Class")
```

![PCA-plot of variance stabilized expression.](image)

**Figure 9.** PCA-plot of variance stabilized expression.
We observe that PC2 separates the samples according to the experimental group (control vs nano). However, PC1 also separates samples into two groups. This is suggestive of an unwanted yet systematic effect on expression, often referred to as a batch effect. We do not want to mistake this unwanted variation for biological variation when we test for differential expression. To prevent this, we can include the batch information as a factor in the final model. Let first define the batch variable:

```r
# Extract pca results
pca_res <- plotPCA(vst_blind, "Class", returnData=TRUE)

# Define a new variable using PC1
batch_var <- ifelse(pca_res$PC1 > 0, "A", "B")

# Attach the batch variable as a factor to the experiment
RSE$Batch <- factor(batch_var)
```

<table>
<thead>
<tr>
<th>Class</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>C547</td>
<td>Ctrl</td>
</tr>
<tr>
<td>C548</td>
<td>Ctrl</td>
</tr>
<tr>
<td>C549</td>
<td>Ctrl</td>
</tr>
<tr>
<td>C559</td>
<td>Ctrl</td>
</tr>
<tr>
<td>C560</td>
<td>Ctrl</td>
</tr>
<tr>
<td>N13</td>
<td>Nano</td>
</tr>
<tr>
<td>N14</td>
<td>Nano</td>
</tr>
<tr>
<td>N15</td>
<td>Nano</td>
</tr>
<tr>
<td>N16</td>
<td>Nano</td>
</tr>
<tr>
<td>N17</td>
<td>Nano</td>
</tr>
<tr>
<td>N18</td>
<td>Nano</td>
</tr>
</tbody>
</table>

Table 4. Design matrix after adding new batch covariate.

An alternative to manually defining the batch variable, tools such as sva and RUVSeq can be used to estimate unknown batch effects from the data.

**Cluster-level differential expression.** Following our short EDA above, we are ready to specify the final design for the experiment: We want to take into account both the Class and Batch of samples:

```r
# Specify design
dds <- DESeqDataSet(RSE, design = ~ Batch + Class)

# Fit DESeq2 model
dds <- DESeq(dds)
```

> estimating size factors
> estimating dispersions
> gene-wise dispersion estimates
> mean-dispersion relationship
> final dispersion estimates
> fitting model and testing
We can now extract estimated effects (log fold changes) and statistical significance (p-values) for the Nanovs-Ctrl comparison, implicitly correcting for the batch effect:

```r
# Extract results
res <- results(dds,
    contrast=c("Class", "Nano", "Ctrl"),
    alpha=0.05,
    independentFiltering=TRUE,
    tidy = TRUE) %>%
    bind_cols(as.data.frame(rowData(RSE))) %>%
    as_tibble

# Show the top hits
res %>%
    top_n(-10, padj) %>%
    dplyr::select(cluster=row, clusterType, txType, baseMean, log2FoldChange, padj) %>%
    kable(caption = "Top differentially expressed TSS and enhancer candidates") %>%
    kable_styling(latex_options = "hold_position")
```

### Table 5. Top differentially expressed TSS and enhancer candidates.

<table>
<thead>
<tr>
<th>cluster</th>
<th>clusterType</th>
<th>txType</th>
<th>baseMean</th>
<th>log2FoldChange</th>
<th>padj</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1:73977049-73977548-</td>
<td>TSS</td>
<td>intron</td>
<td>1183.3740</td>
<td>2.838367</td>
<td>0</td>
</tr>
<tr>
<td>chr2:32243097-32243468-</td>
<td>TSS</td>
<td>promoter</td>
<td>30799.5953</td>
<td>3.741789</td>
<td>0</td>
</tr>
<tr>
<td>chr2:144423689-144423778-</td>
<td>TSS</td>
<td>promoter</td>
<td>191.0431</td>
<td>3.709530</td>
<td>0</td>
</tr>
<tr>
<td>chr4:125840648-125840820-</td>
<td>TSS</td>
<td>proximal</td>
<td>1063.4328</td>
<td>3.709530</td>
<td>0</td>
</tr>
<tr>
<td>chr4:137325466-137325712-</td>
<td>TSS</td>
<td>intron</td>
<td>176.7636</td>
<td>3.709530</td>
<td>0</td>
</tr>
<tr>
<td>chr7:53971039-53971170-</td>
<td>TSS</td>
<td>promoter</td>
<td>8720.5204</td>
<td>6.696838</td>
<td>0</td>
</tr>
<tr>
<td>chr9:120212846-120213294+:</td>
<td>TSS</td>
<td>promoter</td>
<td>316.0582</td>
<td>6.098838</td>
<td>0</td>
</tr>
<tr>
<td>chr11:83222553-83222887+:</td>
<td>TSS</td>
<td>proximal</td>
<td>228.5560</td>
<td>2.404706</td>
<td>0</td>
</tr>
<tr>
<td>chr12:105649334-105649472+:</td>
<td>TSS</td>
<td>CDS</td>
<td>175.1364</td>
<td>3.345412</td>
<td>0</td>
</tr>
<tr>
<td>chr19:56668148-56668332+:</td>
<td>TSS</td>
<td>CDS</td>
<td>103.8795</td>
<td>-2.254371</td>
<td>0</td>
</tr>
</tbody>
</table>

It always a good idea to inspect a few diagnostics plot to make sure the DESeq2 analysis was successful. One such example is an MA-plot (another useful plot is p-value histogram):

```r
ggplot(res, aes(x=log2(baseMean), y=log2FoldChange, color=padj < 0.05)) +
    geom_point(alpha=0.25) +
    geom_hline(yintercept = 0, linetype="dashed", alpha=0.75) +
    facet_grid(clusterType ~.)
```
We can see that we overall find more differentially expressed TSSs compared to enhancers, which is expected since they are also more highly expressed. Many enhancers are filtered away for the final DESeq2 analysis (The “Independent Filtering” Step), as their expression level is too low to detect any DE: This increases power for detecting DE at higher expression levels.

We can tabulate the total number of DE TSSs and enhancers:

```r
table(clusterType=rowRanges(RSE)$clusterType, DE=res$padj<0.05)
```

<table>
<thead>
<tr>
<th>clusterType</th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSS</td>
<td>22071</td>
<td>6385</td>
</tr>
<tr>
<td>Enhancer</td>
<td>3034</td>
<td>199</td>
</tr>
</tbody>
</table>

Correcting expression estimates for batch effects. In addition to looking at estimates and significance for each cluster, we might also want to look at individual expression values for some top hits. However, we then need to also correct the expression estimates themselves for batch effects, just like we did for log fold changes and p-values (using the same model of course).

Here we use ComBat\(^3\) from the sva package which is suitable for removing simple batch effects from small experiments. For more advanced setups, removeBatchEffect from limma can remove arbitrarily complex batch effects. The RUVSeq package and fsva from sva can be used to remove unknown batch effects.

---

**Figure 10.** Diagnostic MA plot of the differential expression analysis.
We again use the variance-stabilizing transformation to prepare the data for ComBat (this makes count data resemble expression estimates obtained from microarrays, as ComBat was originally developed for microarrays).

```r
# Guided variance stabilizing transformation
vst_guided <- varianceStabilizingTransformation(dds, blind=FALSE)
```

To run ComBat we need two additional pieces of information: i) A design matrix describing the biological or wanted effects and ii) the known but unwanted batch effect. We first specify the design matrix, and then run ComBat:

```r
# Design matrix of wanted effects
bio_effects <- model.matrix(~Class, data=colData(RSE))

# Run ComBat =
assay(RSE, "ComBat") <- ComBat(dat=assay(vst_guided),
batch=RSE$Batch, # Unwanted batch
mod=bio_effects)
```

Let us redo the PCA-plot, to see the global effect of the batch effect correction:

```r
# Overwrite assay
assay(vst_guided) <- assay(RSE, "ComBat")

# Plot as before
plotPCA(vst_guided, "Class")
```

![Figure 11. PCA-plot of batch corrected expression.](image)
Now Nano and Ctrl are separated along the first principal component (compared to the second principle component before correction).

Then we extract the top 10 DE enhancers using the following tidyverse code:

```r
# Find top 10 DE enhancers
top10 <- res %>%
  filter(clusterType == "Enhancer", padj < 0.05) %>%
  group_by(log2FoldChange >= 0) %>%
  top_n(5, wt=abs(log2FoldChange)) %>%
  pull(row)

# Extract expression values in tidy format
tidyEnhancers <- assay(RSE, "ComBat")[top10,] %>%
  t %>%
  as.data.frame %>%
  rownames_to_column("Sample") %>%
  mutate(Class=RSE$Class) %>%
  gather(key="Enhancer", value="Expression", -Sample, -Class, factor_key=TRUE)
```

Finally, we can plot the batch-corrected expression profiles of each individual enhancer:

```r
ggplot(tidyEnhancers, aes(x=Class, y=Expression, fill=Class)) +
  geom_dotplot(stackdir="center", binaxis="y", dotsize=3) +
  facet_wrap(~Enhancer, ncol=2, scales="free_y")
#> 'stat_bindot()' using 'bins = 30'. Pick better value with 'binwidth'.
```

**Figure 12.** Expression profile of top 10 differentially expressed enhancer candidates.
Enrichment of DNA-binding motifs. A typical question following identification of differentially expressed TSSs and enhancers, is what TFs might be involved in their regulation. To shed light on this question we can annotate TSSs and enhancers with DNA-binding motifs from the JASPAR database.

First we extract the sequences around TSSs and enhancers. Here we simply define it as +/- 500 bp around TSS peak or enhancer midpoint:

```r
cluster_seq <- RSE %>% rowRanges %>% swapRanges() %>% unstrand() %>% add(500) %>% getSeq(bsg, .)
```

Secondly, we use the `TFBTools` package to obtain motifs as Position Frequency Matrices (PFMs) from the JASPAR2016 database:

```r
# Extract motifs as Position
motif_pfms <- getMatrixSet(JASPAR2016, opts = list(species="10090"))
# Look at the IDs and names of the first few motifs:
head(name(motif_pfms))
#>    MA0004.1    MA0006.1   MA0029.1   MA0063.1   MA0067.1   MA0078.1
#>      "Arnt" "Ahr::Arnt"    "Mecom"   "Nkx2-5"     "Pax2"    "Sox17"
```

Thirdly, we use the `motifmatchr` package to find hits in the sequences:

```r
# Find matches
motif_hits <- matchMotifs(motif_pfms, subject=cluster_seq)
# Matches are returned as a sparse matrix:
motifMatches(motif_hits)[1:5, 1:5]
#> 5 x 5 sparse Matrix of class "lgCMatrix"
#>      MA0004.1 MA0006.1 MA0029.1 MA0063.1 MA0067.1
#> [1,] . . . . . |
#> [2,] . . . . . |
#> [3,] | . | . | . |
#> [4,] . . . . . |
#> [5,] . . | . | . |
```

Finally we can do a simple Fisher’s Exact test to see if a motif co-occurs more with DE TSSs and enhancer than we would expect be chance. Here we will look at the FOS::JUN motif (MA0099.2):

```r
# 2x2 table for fishers
table(FOSJUN = motifMatches(motif_hits)[,"MA0099.2"],
      DE = res$padj < 0.05) %>%
print() %>%
fisher.test()
# DE
# FOSJUN  FALSE TRUE
#   FALSE 22144  5596
#   TRUE  2961  988
# Fisher’s Exact Test for Count Data
A significant odds ratio above 1 indicate that FOS::JUN is a candidate transcription factor (or, more technically correct, a candidate transcription factor dimer) in regulation of the nanotube response. This is not surprising given that FOS::JUN is part of the TNF-alpha inflammatory pathway (see more below).

Of course, this is a just a very quick and simple analysis of motif enrichment. One could easily have used different regions around TSSs and enhancers and/or split the enrichment analysis between TSSs and enhancers. Other Bioconductor packages like `PWMEnrich`, `rGADEM` and `motifcounter` implements more advanced statistical methods for calculating enrichment of known motifs. `rGADEM`, `BCRANK` and `motifRG` can also be used to calculate enrichment of novel motifs, sometimes referred to as motif discovery.

**Gene-level differential expression.** While CAGE data is naturally analyzed at the level of clusters (TSSs and enhancers) it is in many cases interesting to also look at gene-level expression estimates. A prime example of this is looking at enrichment of Gene Ontology (GO) and Kyoto Encyclopedia of Genes and Genomes (KEGG) terms\(^\text{28,29,30}\) which are only defined at gene-level. CAGEfightR includes functions for annotating clusters with gene models and summarizing expression to gene-level.

We can annotate clusters with gene IDs in the same manner as Transcript IDs:

```r
RSE <- assignGeneID(RSE, geneModels=txdb)
#> Extracting genes...
#> Overlapping while taking distance to nearest TSS into account...
#> Finding hierachical overlaps...
#> ### Overlap Summary: ###
#> Features overlapping genes: 81.34 %
#> Number of unique genes: 13761
```

And then use CAGEfightR to sum counts of TSSs within genes:

```r
GSE <- RSE %>%
    subset(clusterType == "TSS") %>%
    quantifyGenes(genes="geneID", inputAssay="counts")
```

The result is `RangedSummarizedExperiment` where the ranges are a `GRangesList` holding the TSSs that were summed within each gene:

```r
rowRanges(GSE["100038347",])
#> GRangesList object of length 1:
#> $100038347
```
The gene IDs in this case is Entrez ID (which is widely used by Bioconductor packages). We can translate these systematic IDs into more human-readable symbols using the `org.Mm.eg.db` annotation package:

```r
# Translate symbols
rowData(GSE)$symbol <- mapIds(odb,
  keys=rownames(GSE),
  column="SYMBOL",
  keytype="ENTREZID")

# 'select()' returned 1:1 mapping between keys and columns
```

Having obtained a gene-level count matrix we can now perform gene-level DE analysis. Here we use limma-voom, since limma makes it easy to perform a subsequent enrichment analysis. Other tools such as DESeq2 (above) or edgeR (see below) could also have been used.

**Note:** limma is a powerful tool for DE analysis of count-based data. However, since it depends on log transforming counts, it is not always suitable for analyzing datasets where features have very low counts. This is usually not a problem for gene-level analysis, but can be a problem for enhancers, which are generally very lowly expressed.

Similarly to the DESeq2 analysis, we first build the necessary object and then normalize the expression values:

```r
# Create DGEList object
dge <- DGEList(counts=assay(GSE, "counts"),
  genes=as.data.frame(rowData(GSE)))

# Calculate normalization factors
dge <- calcNormFactors(dge)
```

Then we apply the voom-transformation to model the mean-variance trend, for which we also need to specify the design matrix (in this case the design must contain both wanted and unwanted effects!). The same design matrix is then used for fitting the gene-wise models:

```r
# Design matrix
mod <- model.matrix(~ Batch + Class, data = colData(GSE))

# Model mean-variance using voom
v <- voom(dge, design=mod)
```
# Fit and shrink DE model
fit <- lmFit(v, design=mod)
eb <- eBayes(fit, robust=TRUE)

# Summarize the results
dt <- decideTests(eb)

We can both report the overall summary of differential gene expression, and look at the first few top hits:

# Global summary
dt %>%
  summary %>%
  kable(caption="Global summary of differentially expressed genes.") %>%
  kable_styling(latex_options = "hold_position")

<table>
<thead>
<tr>
<th></th>
<th>(Intercept)</th>
<th>BatchB</th>
<th>ClassNano</th>
</tr>
</thead>
<tbody>
<tr>
<td>Down</td>
<td>51</td>
<td>2572</td>
<td>1505</td>
</tr>
<tr>
<td>NotSig</td>
<td>463</td>
<td>8278</td>
<td>10373</td>
</tr>
<tr>
<td>Up</td>
<td>13053</td>
<td>2717</td>
<td>1689</td>
</tr>
</tbody>
</table>

# Inspect top hits
topTable(eb, coef="ClassNano") %>%
dplyr::select(symbol, nClusters, AveExpr, logFC, adj.P.Val) %>%
kable(caption="Top differentially expressed genes.") %>%
kable_styling(latex_options = "hold_position")

<table>
<thead>
<tr>
<th>symbol</th>
<th>nClusters</th>
<th>AveExpr</th>
<th>logFC</th>
<th>adj.P.Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sh3d21</td>
<td>3</td>
<td>5.871004</td>
<td>3.075745</td>
<td>0.0e+00</td>
</tr>
<tr>
<td>Myrip</td>
<td>2</td>
<td>4.371325</td>
<td>2.414055</td>
<td>7.0e-07</td>
</tr>
<tr>
<td>Clca3a1</td>
<td>1</td>
<td>3.020528</td>
<td>3.692198</td>
<td>7.0e-07</td>
</tr>
<tr>
<td>Colq</td>
<td>3</td>
<td>2.770158</td>
<td>-3.426911</td>
<td>7.0e-07</td>
</tr>
<tr>
<td>Serpina3n</td>
<td>5</td>
<td>6.384175</td>
<td>1.872782</td>
<td>3.0e-06</td>
</tr>
<tr>
<td>2200002D01Rik</td>
<td>2</td>
<td>7.208031</td>
<td>1.693257</td>
<td>5.5e-06</td>
</tr>
<tr>
<td>Prmt8</td>
<td>4</td>
<td>4.553612</td>
<td>1.409006</td>
<td>5.8e-06</td>
</tr>
<tr>
<td>Tmem37</td>
<td>2</td>
<td>5.503908</td>
<td>1.679690</td>
<td>5.8e-06</td>
</tr>
<tr>
<td>Pgf</td>
<td>1</td>
<td>4.862055</td>
<td>2.337045</td>
<td>5.8e-06</td>
</tr>
<tr>
<td>Sema7a</td>
<td>1</td>
<td>7.612236</td>
<td>1.473680</td>
<td>5.9e-06</td>
</tr>
</tbody>
</table>
**Enrichment of GO- and KEGG-terms.** In addition to looking at individual top genes, we can look at how the differentially expressed genes relate to known databases of gene function to gain insight in what biological processes might be affected in the experiment.

`limma` makes it easy to perform such an enrichment analysis following a DE analysis. As we have gene indexed by Entrez IDs, we can directly use `goana` to find enriched GO-terms: `goana` uses a biased urn-model to estimate enrichment of GO-terms, while taking into account the expression levels of DE genes:

```r
# Find enriched GO-terms
GO <- goana(eb, coef = "ClassNano", species = "Mm", trend = TRUE)

# Show top hits
topGO(GO, ontology = "BP", number = 10) %>%
  kable(caption="Top enriched or depleted GO-terms.") %>%
  kable_styling(latex_options = "hold_position")
```

<table>
<thead>
<tr>
<th>Term</th>
<th>Ont</th>
<th>N</th>
<th>Up</th>
<th>Down</th>
<th>P.Up</th>
<th>P.Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO:0006954 inflammatory response BP</td>
<td>556</td>
<td>142</td>
<td>51</td>
<td>0</td>
<td>0.9562685</td>
<td></td>
</tr>
<tr>
<td>GO:0006952 defense response BP</td>
<td>1072</td>
<td>224</td>
<td>99</td>
<td>0</td>
<td>0.9878373</td>
<td></td>
</tr>
<tr>
<td>GO:0097529 myeloid leukocyte migration BP</td>
<td>170</td>
<td>61</td>
<td>14</td>
<td>0</td>
<td>0.9359984</td>
<td></td>
</tr>
<tr>
<td>GO:0010033 response to organic substance BP</td>
<td>2074</td>
<td>370</td>
<td>196</td>
<td>0</td>
<td>0.9987104</td>
<td></td>
</tr>
<tr>
<td>GO:0006950 response to stress BP</td>
<td>2755</td>
<td>464</td>
<td>246</td>
<td>0</td>
<td>0.9999946</td>
<td></td>
</tr>
<tr>
<td>GO:0006955 immune response BP</td>
<td>1034</td>
<td>210</td>
<td>96</td>
<td>0</td>
<td>0.9833226</td>
<td></td>
</tr>
<tr>
<td>GO:0042221 response to chemical BP</td>
<td>2762</td>
<td>467</td>
<td>292</td>
<td>0</td>
<td>0.9178712</td>
<td></td>
</tr>
<tr>
<td>GO:0050900 leukocyte migration BP</td>
<td>288</td>
<td>83</td>
<td>23</td>
<td>0</td>
<td>0.9792828</td>
<td></td>
</tr>
<tr>
<td>GO:0001816 cytokine production BP</td>
<td>634</td>
<td>143</td>
<td>45</td>
<td>0</td>
<td>0.9998658</td>
<td></td>
</tr>
<tr>
<td>GO:0001817 regulation of cytokine production BP</td>
<td>570</td>
<td>132</td>
<td>39</td>
<td>0</td>
<td>0.9998856</td>
<td></td>
</tr>
</tbody>
</table>

And similarly for KEGG terms:

```r
# Find enriched KEGG-terms
KEGG <- kegga(eb, coef="ClassNano", species = "Mm", trend = TRUE)

# Show top hits
topKEGG(KEGG, number = 10) %>%
  knitr::kable(caption="Top enriched or depleted KEGG-terms.") %>%
  kable_styling(latex_options = "hold_position")
```

<table>
<thead>
<tr>
<th>Pathway</th>
<th>N</th>
<th>Up</th>
<th>Down</th>
<th>P.Up</th>
<th>P.Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>path:mmu04060 Cytokine-cytokine receptor interaction</td>
<td>173</td>
<td>56</td>
<td>13</td>
<td>0.0000000</td>
<td>0.9579351</td>
</tr>
<tr>
<td>path:mmu04668 TNF signaling pathway</td>
<td>105</td>
<td>31</td>
<td>8</td>
<td>0.0000037</td>
<td>0.9186628</td>
</tr>
<tr>
<td>path:mmu00600 Sphingolipid metabolism</td>
<td>41</td>
<td>17</td>
<td>2</td>
<td>0.0000051</td>
<td>0.9583011</td>
</tr>
<tr>
<td>path:mmu00980 Metabolism of xenobiotics by cytochrome P450</td>
<td>48</td>
<td>4</td>
<td>17</td>
<td>0.8857194</td>
<td>0.0000137</td>
</tr>
<tr>
<td>path:mmu03010 Ribosome</td>
<td>122</td>
<td>32</td>
<td>2</td>
<td>0.0000226</td>
<td>0.9999900</td>
</tr>
<tr>
<td>path:mmu04064 NF-kappa B signaling pathway</td>
<td>85</td>
<td>24</td>
<td>5</td>
<td>0.0000704</td>
<td>0.9655534</td>
</tr>
<tr>
<td>path:mmu04657 IL-17 signaling pathway</td>
<td>74</td>
<td>22</td>
<td>2</td>
<td>0.0000806</td>
<td>0.9985563</td>
</tr>
<tr>
<td>path:mmu00982 Drug metabolism - cytochrome P450</td>
<td>46</td>
<td>5</td>
<td>15</td>
<td>0.7266916</td>
<td>0.0001238</td>
</tr>
<tr>
<td>path:mmu04630 JAK-STAT signaling pathway</td>
<td>112</td>
<td>29</td>
<td>7</td>
<td>0.0001453</td>
<td>0.9785951</td>
</tr>
<tr>
<td>path:mmu04512 ECM-receptor interaction</td>
<td>69</td>
<td>21</td>
<td>13</td>
<td>0.0001488</td>
<td>0.0577601</td>
</tr>
</tbody>
</table>
Both analyses indicate that genes related to the inflammatory response and defense response are upregulated following nanotube exposure. This supports the hypothesis that nanotube induces a response similar to asbestos.

KEGG-terms represents well defined pathways. We can use the `pathview` package to investigate in more detail the genes in a given enriched pathway. For example, we can look at regulation of gene in the TNF- signalling pathway:

```r
# Visualize a KEGG
DE_genes <- Filter(function(x) x != 0, dt[, "ClassNano"])

# This will save a png file to a temporary directory
pathview(DE_genes, species="mmu", pathway.id="mmu04668", kegg.dir = tempdir())

# Show the png file
grid.newpage()
grid.raster(png::readPNG("mmu04668.pathview.png"))
```

![Figure 13. Detailed view of differentially expressed gene in the KEGG TNF-signalling pathway.](image)
**Differential TSS Usage.** In the above two analyses we looked at whether an individual TSSs or an individual gene was changing expression between experimental groups. However, we might also want to look at whether a gene show differential TSS usage: whether a gene uses different TSSs under different conditions. This problem is similar to differential splicing in RNA-Seq, but looking at TSSs rather than isoforms. Here we will use the edgeR `diffSpliceDGE` method to find differential TSS usage, although many other packages could have been used, for example `diffSplice` from limma, `DEXSeq`, `DRIMSeq`, etc..

Intuitively, `diffSpliceDGE` tests whether a given TSSs show the same change as other TSSs in the same gene, indicating that TSSs are differentially regulated across the gene. This does however not take into account the relative composition of a given TSSs, e.g. whether a TSS increases from 1%–2% of gene output or 25%–50%. A useful preprocessing step is therefore to filter out TSSs making only a small contribution to total gene expression before analyses.

We use `CAGEfightR` to remove TSSs that are not expressed as more than 10% of total gene expression in more than 5 samples (We first remove TSSs not assigned to genes):

```r
# Filter away lowly expressed
RSE_filtered <- RSE %>%
  subset(clusterType == "TSS" & !is.na(geneID)) %>%
  subsetByComposition(inputAssay="counts",
    genes="geneID",
    unexpressed=0.1,
    minSamples=5)

#> Calculating composition...
#> Subsetting...
#> Removed 8001 out of 24500 regions (32.7%)
```

We can only do differential TSS usage analysis of genes with multiple TSSs. A useful first visualization is therefore to see how many genes use more than one TSS:

```r
RSE_filtered %>%
  rowData %>%
  as.data.frame %>%
  as_tibble %>%
  dplyr::count(geneID) %>%
  ggplot(aes(x = n, fill = n >= 2)) +
  geom_bar(alpha=0.75) +
  scale_fill_colorblind("Multi-TSS") +
  labs(x = "Number of TSSs per gene", y = "Frequency")
```

![Figure 14. Overview of alternative TSS usage within genes.](image)
While most genes utilize only a single TSS, many genes use two or more TSSs. Again, we build the necessary R-objects for running `edgeR`:

```r
# Annotate with symbols like before:
rowData(RSE_filtered)$symbol <- mapIds(odb,
    keys=rowData(RSE_filtered)$geneID,
    column="SYMBOL",
    keytype="ENTREZID")
#> 'select()' returned 1:1 mapping between keys and columns

# Extract gene info
TSS_info <- RSE_filtered %>%
    rowData %>%
    subset(select=c(score, txType, geneID, symbol)) %>%
    as.data.frame

# Build DGEList
dge <- DGEList(counts=assay(RSE_filtered, "counts"),
    genes=TSS_info)

Then we normalize and fit models using the Quasi-likelihood approach, including the `diffSpliceDGE` step:

```r
# Estimate normalization factors
dge <- calcNormFactors(dge)

# Estimate dispersion and fit GLMs
disp <- estimateDisp(dge, design = mod, tagwise = FALSE)
QLfit <- glmQLFit(disp, design=mod, robust = TRUE)

# Apply diffSpliceDGE
ds <- diffSpliceDGE(QLfit, coef = "ClassNano", geneid = "geneID")
#> Total number of exons: 16499
#> Total number of genes: 13563
#> Number of genes with 1 exon: 11098
#> Mean number of exons in a gene: 1
#> Max number of exons in a gene: 5
```

Now we can look at differential TSS usage at two-levels: Whether an individual TSS shows differential TSS usage (TSS-level) or whether a gene show differential TSS usage in any way (gene-level). First we can look at individual TSSs (TSS-level differential TSS usage):

```r
dtu_TSSs <- topSpliceDGE(ds, test = "exon")
dplyr::select(dtu_TSSs, txType, geneID, symbol, logFC, FDR) %>%
    kable(caption = "Top differentially used TSSs") %>%
    kable_styling(latex_options = "hold_position")
```
Table 10. Top differentially used TSSs.

<table>
<thead>
<tr>
<th>txType</th>
<th>geneID</th>
<th>symbol</th>
<th>logFC</th>
<th>FDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr17:13840650-13840851;-</td>
<td>intron</td>
<td>21646</td>
<td>Tcte2</td>
<td>1.7889344</td>
</tr>
<tr>
<td>chr10:57857044-57857314;+</td>
<td>promoter</td>
<td>110829</td>
<td>Lims1</td>
<td>-1.0651946</td>
</tr>
<tr>
<td>chr14:70215678-70215876;-</td>
<td>intron</td>
<td>246710</td>
<td>Rho1bb2</td>
<td>2.4933979</td>
</tr>
<tr>
<td>chr4:141154044-141154185;+</td>
<td>intron</td>
<td>74202</td>
<td>Fblim1</td>
<td>1.7018062</td>
</tr>
<tr>
<td>chr17:33966135-33966308;+</td>
<td>intron</td>
<td>66416</td>
<td>Ndufa7</td>
<td>2.1612127</td>
</tr>
<tr>
<td>chr15:76428030-76428201;-</td>
<td>intron</td>
<td>94230</td>
<td>Cpsf1</td>
<td>1.4598815</td>
</tr>
<tr>
<td>chr19:57271818-57272125;+</td>
<td>promoter</td>
<td>226251</td>
<td>Ablim1</td>
<td>1.1456163</td>
</tr>
<tr>
<td>chr9:77789668-77789200;+</td>
<td>intron</td>
<td>68801</td>
<td>Elovl5</td>
<td>0.9810692</td>
</tr>
<tr>
<td>chr11:116395161-116395462;+</td>
<td>proximal</td>
<td>20698</td>
<td>Sphk1</td>
<td>1.7471930</td>
</tr>
<tr>
<td>chr2:91496305-91496449;+</td>
<td>intron</td>
<td>228359</td>
<td>Arhgap1</td>
<td>0.9809491</td>
</tr>
</tbody>
</table>

The interpretation of log fold changes here is slightly different from before: These log fold changes are relative to the overall log fold change for all TSSs in that gene.

Then we can look at results for each gene (Gene-level differential TSS usage):

```r
dtu_genes <- topSpliceDGE(ds, test = "Simes")
dplyr::select(dtu_genes, geneID, symbol, NExons, FDR) %>%
kable(row.names = FALSE, caption = "Top genes showing any differential TSS usage.", latex_options = "hold_position")
```

Table 11. Top genes showing any differential TSS usage.

<table>
<thead>
<tr>
<th>geneID</th>
<th>symbol</th>
<th>NExons</th>
<th>FDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>21646</td>
<td>Tcte2</td>
<td>4</td>
<td>0e+00</td>
</tr>
<tr>
<td>110829</td>
<td>Lims1</td>
<td>3</td>
<td>0e+00</td>
</tr>
<tr>
<td>246710</td>
<td>Rho1bb2</td>
<td>3</td>
<td>0e+00</td>
</tr>
<tr>
<td>74202</td>
<td>Fblim1</td>
<td>3</td>
<td>0e+00</td>
</tr>
<tr>
<td>66416</td>
<td>Ndufa7</td>
<td>3</td>
<td>0e+00</td>
</tr>
<tr>
<td>94230</td>
<td>Cpsf1</td>
<td>2</td>
<td>0e+00</td>
</tr>
<tr>
<td>226251</td>
<td>Ablim1</td>
<td>3</td>
<td>0e+00</td>
</tr>
<tr>
<td>68801</td>
<td>Elovl5</td>
<td>2</td>
<td>1e-07</td>
</tr>
<tr>
<td>20698</td>
<td>Sphk1</td>
<td>3</td>
<td>1e-07</td>
</tr>
<tr>
<td>228359</td>
<td>Arhgap1</td>
<td>2</td>
<td>2e-07</td>
</tr>
</tbody>
</table>

We see that the two lists agree, which is not surprising given that the gene-level results are obtained by aggregating TSS-level p-values across genes.

We can look at closer at the TSS usage in one of the top hits: We can visualize the batch-corrected expression (See above) of each TSS in the Fblim1 gene via a heatmap:

```r
RSE_filtered %>%
  subset(geneID == "74202") %>%
  assay("ComBat") %>%
  t %>%
pheatmap(color = magma(100),
    cluster_cols = FALSE,
    main="Fblim1")
```
Figure 15. Heatmap showing expression of TSSs within Fblim1.

Fblim1 has 3 TSSs, with 2 of them being used in the Ctrl samples, while the Nano samples also uses the chr4:141154044-141154185- TSS, as also seen in the TSS-level table above. While a heatmap is useful for seeing expression changes, a genome browser view is better to inspect the genomic context of each TSSs:

```r
# Define plot area
plot_region <- subset(RSE_filtered, geneID == "74202") %>%
  rowRanges %>%
  reduce(min.gapwidth=1e6) %>%
  unstrand() %>%
  add(5e3L)

# Create cluster track
cluster_track <- subsetByOverlaps(RSE_filtered, plot_region) %>%
  trackClusters(name = "Clusters", col = NA, showId=TRUE)

# Setting thick and thin features...
# Merging and sorting...
# Preparing track...
```
### CTSS tracks for each group

```r
ctrl_track <- subset(CTSSs, select=Class == "Ctrl") %>%
    calcPooled() %>%
    subsetByOverlaps(plot_region) %>%
    trackCTSS(name="Ctrl")
#> Warning in calcPooled(.): object already has a column named score in rowData: It will be overwritten!
#> Splitting pooled signal by strand...
#> Preparing track...

nano_track <- subset(CTSSs, select=Class == "Nano") %>%
    calcPooled() %>%
    subsetByOverlaps(plot_region) %>%
    trackCTSS(name="Nano")
#> Warning in calcPooled(.): object already has a column named score in rowData: It will be overwritten!
#> Splitting pooled signal by strand...
#> Preparing track...

# Plot at tracks together
plotTracks(list(axis_track,
    tx_track,
    cluster_track,
    Ctrl=ctrl_track,
    nano_track),
    from = start(plot_region),
    to=end(plot_region),
    chromosome = seqnames(plot_region))
```

**Figure 16.** Genome-browser example of differential TSS usage within Fblim1.
The Fblim1 gene uses two annotated TSSs, but the Nano samples also uses a novel intronic TSS.

**Discussion**

This workflow is intended as providing an outline of the basic building blocks of CAGE data analysis, going from clustering, to spatial analyses to differential expression. More advanced analyses can be strung together from these basic elements: Finding enhancers linked to DE TSSs, enhancer stretches composed of DE enhancer, comparing DNA binding motif enrichments between DE enhancers and TSSs, etc.

One aspect not covered in this workflow is the utility of CAGE data (and 5’-end data in general) in providing accurate TSSs for studying other types of data. For example, having accurate TSSs is highly beneficial in chromatin research, since the location and nucleosome and TSSs are closely related. CAGE can be combined with chromatin confirmation assays such as HiC to find new enhancers that are both co-expressed and physically interacting with TSSs. Many genome-wide association studies are finding that disease-related genetic variants are found in intergenic regions, that are often poorly annotated. The accurate enhancer locations provided by CAGE can greatly aid interpretation of such variants. The adherence of CAGEfightR to standard Bioconductor classes facilitates these inter-assay analyses by making it easy to mix-and-match multiple packages developed for different experimental assays.

**Software and data availability**

The following software versions were used in this article:

- **R version**: R version 3.6.0 (2019-04-26)
- **Bioconductor version**: 3.9
- **CAGEfightR version**: 1.4.0

**CAGEWorkflow**: https://doi.org/10.18129/B9.bioc.CAGEWorkflow

License: GPL-3

**Mouse nanotube CAGE data**: https://doi.org/10.18129/B9.bioc.nanotubes

License: GPL-3

**Grant information**

Work in the Sandelin Lab was supported by the Novo Nordisk Foundation, Lundbeck foundation, Danish Innovation Fund, Danish Cancer Society and Independent Research Fund Denmark.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

**Acknowledgments**

We acknowledge all members of the Sandelin Lab and Andersson Lab for advice, discussion and input on all aspects related to CAGE data analysis.

**References**


PubMed Abstract | Publisher Full Text


PubMed Abstract | Publisher Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text | Free Full Text


PubMed Abstract | Publisher Full Text

25. CAGEfightR: Cap Analysis of Gene Expression (CAGE) Bioinformatics. 2015; 31:10623.

PubMed Abstract | Publisher Full Text


Publisher Full Text

27. 47
The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com