CummeRbund: Visualization and Exploration of Cufflinks High-throughput Sequencing Data

Loyal A. Goff, Cole Trapnell, David Kelley

May 7, 2014

Contents

1 Requirements 2

2 Introduction 3

3 CummeRbund Classes 4
  3.1 CuffSet Class .............................................................. 4
  3.2 CuffData Class ............................................................ 4
  3.3 CuffDist Class ............................................................ 4
  3.4 CuffFeatureSet Class .................................................... 4
  3.5 CuffFeature Class ....................................................... 5

4 Reading cuffdiff output 6
  4.1 Adding additional feature annotation ............................. 7

5 Global statistics and Quality Control 8

6 Accessing Data 14
  6.1 Writing your own SQL accessors .................................. 18

7 Creating Gene Sets 20
  7.1 Geneset level plots ...................................................... 22

8 Individual Genes 27
  8.1 Gene-level plots .......................................................... 27
    8.1.1 Gene Feature plots ................................................. 31

9 Data Exploration 34
  9.1 Overview of significant features .................................. 34
  9.2 Creating gene sets from significantly regulated genes ........ 34
  9.3 Exploring the relationships between conditions .................. 36
    9.3.1 Distance matrix ................................................... 36
    9.3.2 Dimensionality reduction ....................................... 37
  9.4 Partitioning .............................................................. 39
  9.5 Specificity ............................................................... 40
  9.6 Finding similar genes .................................................. 41

10 Miscellaneous 43

11 Known Issues 44

12 Session info 45
1 Requirements

NOTE: cummeRbund 2.0 was designed in conjunction with the release of cufflinks 2.0. While we attempted to preserve backwards-compatibility, it is highly recommended that you update your cufflinks installation to version $\geq 2.0$ to take full advantage of the improvements in modeling, reporting, and visualization that have been incorporated.

- Cufflinks $\geq v2.0.1$
- SQLite
- R $\geq v3.0$
- Packages:
  - RSQLite
  - ggplot2 $\geq v0.9.3$
  - reshape2
  - plyr
  - fastcluster
  - rtracklayer
  - Gviz
  - BiocGenerics $\geq 0.3.2$
  - Recommended:
    - Hmisc
2 Introduction

cummeRbund is a visualization package for Cufflinks high-throughput sequencing data. It is designed to help you navigate through the large amount of data produced from a Cuffdiff RNA-Seq differential expression analysis. The results of this analysis are typically a large number of inter-related files that are not terribly intuitive to navigate through. cummeRbund helps promote rapid analysis of RNA-Seq data by aggregating, indexing, and allowing you easily visualize and create publication-ready figures of your RNA-Seq data while maintaining appropriate relationships between connected data points. CummeRbund is a multifaceted suite for streamlined analysis and visualization of massively parallel RNA differential expression data sequencing data.

CummeRbund begins by re-organizing output files of a cuffdiff analysis, and storing these data in a local SQLite database. CummeRbund indexes the data to speed up access to specific feature data (genes, isoforms, TSS, CDS, etc.), and preserves the various relationships between these features. Access to data elements is managed via the RSQLite package and data are presented in appropriately structured R classes with various convenience functions designed to streamline your workflow. This persistent database storage means that inter-connected expression values are rapidly accessible and quickly searchable in future analyses.

CummeRbund defines two types of data classes, 'pointer' or reference classes describe SQL connections to the database without directly containing data, and 'data' classes that retrieve a subset of related data points such as associated features from a given gene or gene set. Each class type has methods for direct access to FPKM values, differential expression information, statistical test results, raw and normalized fragment counts, individual replicate FPKM values, and additional annotation information for features. Output formats allow for browsing and analysis of data in standard R objects (data.frame, list, etc). CummeRbund was designed to provide analysis and visualization tools analogous to microarray data. In this regard, numerous plotting methods are provided for visualization of RNA-Seq data quality and global statistics, and simple routines for plotting expression levels for one or thousands of genes, their isoforms, TSS groups, or CDS groups.

The base class, cuffSet is a 'pointer' to cuffdiff data that are stored out-of-memory in a sqlite database.
3 CummeRbund Classes

3.1 CuffSet Class

A pointer class to control access to the SQLite tables holding the Cufflinks data. The primary slot is DB which contains the RSQLite connection object. This can be accessed using the DB() accessor. The additional slots (genes, isoforms, TSS, and CDS) are each instances of the CuffData class and are pointers to sets of tables for each data subtype. They can be accessed with similar accessor wrappers. This is the default class created by readCufflinks. By default, CuffData accessor methods applied to a CuffSet class will operate on the 'genes' slot. The runInfo() method can be used to retrieve information about the actual cuffdiff run itself, including command-line arguments used to generate the results files.

3.2 CuffData Class

The CuffData class is also a pointer class to the SQL backend, but each instance is specific for a data subtype (genes, isoforms, TSS, CDS). Again, there is an DB slot (accessible using DB()) that contains the RSQLite connection object. There are several accessor, setter, and plotting methods that allow for global analysis of all features within a CuffData class. Subsetting is currently being re-written, however, it is primarily done through the 'gene_id' field. Available slots for the CuffData class are:

- DB: RSQLite connection object
- tables: A list of tables in the SQLite DB that contain the cufflinks data.
- filters: A list of filters for subsetting (not implemented yet).
- type: A character field describing the data (ie. 'genes','isoforms','TSS','CDS','other')
- idField: The name of the identifying index field for this object (eg. 'gene_id' for type='gene', or 'isoform_id' for type='isoform')

Making the best use of either the CuffSet or CuffData classes will enable you to keep the entire dataset out of memory and significantly improve performance for large cufflinks datasets.

3.3 CuffDist Class

The CuffDist class is an pointer class that contains the results of the various 'distribution tests' performed by cuffdiff. These include differential promoter usage, differential splicing, and differential CDS usage. These are independent tests from the differential analysis of gene-, isoform-, TSS-, and CDS-level features and therefore have their own container type to distinguish them as such. The 'promoters', 'relCDS', and 'splicing' slots of a CuffSet class are all CuffDist instances.

Available slots for the CuffDist class are:

- DB: RSQLite connection object
- tables: A list of tables in the SQLite DB that contain the distribution test data.
- type: A character field describing the data (ie. 'promoters','relCDS','splicing')
- idField: The name of the identifying index field for this object (eg. 'TSS_group_id' for type='promoters', or 'CDS_id' for type='relCDS', etc.)

3.4 CuffFeatureSet Class

The CuffFeatureSet class is a data-storage container that holds all available data for a pre-determined list of features. Slots for FPKM data, differential regulation data, and feature-level annotation are all available. Unlike the previous classes, this class contains no connection information to the SQL database, but rather contains several slots with data.frame objects storing multiple-features worth of information. There are available accessors, and plotting methods that are designed to present multiple-features worth of information (eg. heatmaps, scatterplots, etc. Available slots for a CuffFeatureSet object include:

- annotation: Holds all feature-level annotation information for all features in object.
- fpkm: A data frame of FPKM data across all conditions, for all features in object.
- repFpkm: A data frame of deconvolved FPKM values across individual replicates, for all features in object.
- diff: A data frame of differential expression/regulation data for all features in object.
- count: A data frame containing raw and normalized fragment counts, variance, dispersion, and uncertainty for all features in object.
- genome: A character string indicating which build of the genome the associated features are derived from. (e.g. ‘hg19’, ‘mm9’)

A specialized sub-class of CuffFeatureSet is the CuffGeneSet class. This subclass adds additional slots to contain all isoforms, TSS, and CDS information for a given set of gene_ids. The CuffGeneSet class is designed to aggregate all relevant information for a set of genes into one object for easy analysis and/or manipulation. The CuffGeneSet object adds the following slots:

- ids: A ‘character’ list of all gene_ids used in object.
- isoforms: A CuffFeatureSet object for all isoforms of genes in object.
- TSS: A CuffFeatureSet object for all TSS of genes in object.
- CDS: A CuffFeatureSet object for all CDS of genes in object.

3.5 CuffFeature Class

The CuffFeature class is designed for single-feature-level data analysis and plotting. The methods available for this object are designed to analyze or visualize information about a specific feature. This is a ‘data’ object, as opposed to a ‘pointer’ object to the database backend. There is a validity requirement that a CuffFeature object only point to data from a single feature. Available slots for a CuffFeature object include:

- annotation: Holds feature-level annotation information for a given feature.
- fpkm: A data frame of FPKM data across all samples for a given feature.
- repFpkm: A data frame of deconvolved FPKM values across all replicates for a given feature.
- diff: A data frame of differential expression/regulation data for a given feature.
- count: A data frame containing raw and normalized fragment counts, variance, dispersion, and uncertainty for a given feature.

A specialized sub-class of CuffFeature is the CuffGene class. This subclass adds additional slots to contain all isoform, TSS, and CDS information for a given gene. The CuffGene object adds the following slots:

- id: The common ‘gene_id’ for all data in object
- isoforms: A CuffFeature object for all isoforms of a given gene.
- TSS: A CuffFeature object for all TSS of a given gene.
- CDS: A CuffFeature object for all CDS of a given gene.
- features: A data.frame object containing feature information for the transcript models describing the gene.
4 Reading cuffdiff output

cummeRbund was designed to process the multi-file output format for a `cuffdiff` differential expression analysis. In this type of analysis, a user will use a reference .gtf file (either known annotation or a .gtf file created from a cufflinks assembly or merge of assemblies) and quantitate the expression values and differential regulation of the annotation(s) in the .gtf file across two or more SAM/BAM files. By design, cuffdiff produces a number of output files that contain test results for changes in expression at the level of transcripts, primary transcripts, and genes. It also tracks changes in the relative abundance of transcripts sharing a common transcription start site, and in the relative abundances of the primary transcripts of each gene. Tracking the former allows one to see changes in splicing, and the latter lets one see changes in relative promoter use within a gene.

Note: Early versions of Cuffdiff required that transcripts in the input GTF be annotated with certain attributes in order to look for changes in primary transcript expression, splicing, coding output, and promoter use. This is no longer the case with >=v1.1.1 of cummeRbund, however we still recommend the use of both the following attributes in your GTF file to enable all downstream features of cummeRbund.

These attributes are:

- **tss_id**: The ID of this transcript’s inferred start site. Determines which primary transcript this processed transcript is believed to come from. Cuffcompare appends this attribute to every transcript reported in the .combined.gtf file.
- **p_id**: The ID of the coding sequence this transcript contains. This attribute is attached by Cuffcompare to the .combined.gtf records only when it is run with a reference annotation that include CDS records. Further, differential CDS analysis is only performed when all isoforms of a gene have p_id attributes, because neither Cufflinks nor Cuffcompare attempt to assign an open reading frame to transcripts.

Cuffdiff calculates the FPKM of each transcript, primary transcript, and gene in each sample. Primary transcript and gene FPKMs are computed by summing the FPKMs of transcripts in each primary transcript group or gene group. The results are output in FPKM tracking files, the structure of which can be found in the cufflinks manual.

There are four FPKM tracking files:

- **isoforms.fpkm_tracking**: Transcript FPKMs
- **genes.fpkm_tracking**: Gene FPKMs. Tracks the summed FPKM of transcripts sharing each gene_id
- **cds.fpkm_tracking**: Coding sequence FPKMs. Tracks the summed FPKM of transcripts sharing each p_id, independent of tss_id
- **tss_groups.fpkm_tracking**: Primary transcript FPKMs. Tracks the summed FPKM of transcripts sharing each tss_id

Cuffdiff also performs differential expression tests between supplied conditions. This tab delimited file lists the results of differential expression testing between samples for spliced transcripts, primary transcripts, genes, and coding sequences. For detailed file structure see cufflinks manual.

Four .diff files are created:

- **isoform_exp.diff**: Transcript differential FPKM.
- **gene_exp.diff**: Gene differential FPKM. Tests difference in the summed FPKM of transcripts sharing each gene_id
- **tss_group_exp.diff**: Primary transcript differential FPKM. Tests differences in the summed FPKM of transcripts sharing each tss_id
- **cds_exp.diff**: Coding sequence differential FPKM. Tests differences in the summed FPKM of transcripts sharing each p_id independent of tss_id

In addition, cuffdiff also performs differential splicing, CDS usage, and promoter usage tests for each gene across conditions:
- **splicing.diff** Differential splicing tests.
- **CDS.diff** Differential coding output.
- **promoters.diff** Differential promoter use.

All of these output files are related to each other through their various tracking_ids, but parsing through individual files to query for important result information requires both a good deal of patience and a strong grasp of command-line text manipulation. Enter cummeRbund, an R solution to aggregate, organize, and help visualize this multi-layered dataset.

One of the principle benefits of using cummeRbund is that data are stored in a SQLite database. This allows for out-of-memory analysis of data, quick retrieval, and only a one-time cost to setup the tables. By default, cummeRbund assumes that all output files from cuffdiff are in the current working directory. To read these files, populate the `cuffData.db` database backend, and return the **CuffSet** pointer object, you can do the following.

```r
> library(cummeRbund)
> cuff<-readCufflinks()
> cuff
```

CuffSet instance with:
- 3 samples
- 400 genes
- 1203 isoforms
- 662 TSS
- 906 CDS
- 1062 promoters
- 1986 splicing
- 990 relCDS

Again, by default `dir` is assumed to be the current working directory and `cuff<-readCufflinks()` should work if all appropriate files are in the current working directory. We now also recommend that you use both the `genome` and `gtfFile` arguments to `readCufflinks()`. This will allow cummeRbund to archive the transcript structure information located in the .gtf file associated with your particular cuffdiff run, as well as associate these transcripts with an appropriate genome build (e.g. 'hg19', 'mm9', etc) so as to allow for transcript-level visualizations and future integration with other external resources. Should you need to rebuild the SQLite backend for any reason, you can add the option `rebuild=T` to `readCufflinks`. Once the database is created, `readCufflinks` will default to using the SQL backend and should not need to rebuild this database. Each R session should begin with a call to `readCufflinks` so as to initialize the database connection and create an object with the appropriate RSQLite connection information.

### 4.1 Adding additional feature annotation

Gene- or feature-level annotation can be permanently added to the database tables for future querying. If you have a data.frame where the first column contains the 'tracking_id' (eg. 'gene_id' for genes, 'isoform_id' for isoforms, etc). You can easily add feature level annotation using the **addFeatures()** function:

```r
> #annot<-read.table("gene_annotation.tab",sep="\t",header=T,na.string="-")
> #addFeatures(cuff,annot,level="genes")
```

By default, features added to a **CuffSet** object are assumed to be gene-level annotations, but the level can selected using the argument `level`. Features added to a **CuffData** object are assumed to be of the same type as the 'type' value for that given object (e.g. gene-level features for 'genes', isoform-level features for isoforms, etc.)
5 Global statistics and Quality Control

Several plotting methods are available that allow for quality-control or global analysis of cufflinks data. A good place to begin is to evaluate the quality of the model fitting. Overdispersion is a common problem in RNA-Seq data. As of cufflinks v2.0 mean counts, variance, and dispersion are all emitted, allowing you to visualize the estimated overdispersion for each sample as a quality control measure.

> disp<-dispersionPlot(genes(cuff))
> disp

(a) Count vs dispersion plot by condition for all genes.

Alternatively a call to dispersionPlot(cuff) directly will allow you to visualize the full model fit.

The squared coefficient of variation is a normalized measure of cross-replicate variability that can be useful for evaluating the quality your RNA-seq data. Differences in $CV^2$ can result in lower numbers of differentially expressed genes due to a higher degree of variability between replicate fpkm estimates.

> genes.scv<-fpkmSCVPlot(genes(cuff))
> isoforms.scv<-fpkmSCVPlot(isoforms(cuff))

To assess the distributions of FPKM scores across samples, you can use the csDensity plot (Figure 1).

> dens<-csDensity(genes(cuff))
> dens
> densRep<-csDensity(genes(cuff),replicates=T)
> densRep

Boxplots can be visualized using the csBoxplot method (Figure 2).

> b<-csBoxplot(genes(cuff))
> b
> brep<-csBoxplot(genes(cuff),replicates=T)
> brep

A matrix of pairwise scatterplots can be drawn using the csScatterMatrix() method.

> s<-csScatterMatrix(genes(cuff))
>

Individual Pairwise comparisons can be made by using csScatter. You must specify the sample names to use for the $x$ and $y$ axes:
(a) The squared coefficient of variation allows visualization of cross-replicate variability between conditions and can be a useful metric in determining data quality at the gene level (left) or isoform level (right). Here we demonstrate the variability of each individual ENCODE project RNA-seq conditions.

(b) Density plot with replicates=TRUE exposes individual replicate FPKM distributions.
(a) Box plot of FPKM distributions for individual conditions.

(b) Box plot with replicates=TRUE exposes individual replicate FPKM distributions.

(a) Scatterplots can be useful to identify global changes and trends in gene expression between pairs of conditions.
(a) Pairwise scatterplots can identify biases in gene expression between two particular conditions.

```r
> s <- csScatter(genes(cuff), "hESC", "Fibroblasts", smooth=T)
> s
```

```r
dend <- csDendro(genes(cuff))
dend.rep <- csDendro(genes(cuff), replicates=T)
```

(a) Dendrogram of JS distances between conditions.  
(b) Dendrogram with replicates=TRUE can identify outlier replicates.

MvsA plots can be useful to determine any systematic bias that may be present between conditions. The CuffData method `MAplot()` can be used to examine these intensity vs fold-change plots. You must specify the sample names to use for the pairwise comparison with `x` and `y`:

```r
> m <- MAplot(genes(cuff), "hESC", "Fibroblasts")
> m
> mCount <- MAplot(genes(cuff), "hESC", "Fibroblasts", useCount=T)
> mCount
```
Volcano plots are also available for the *CuffData* objects.

```r
> v<-csVolcanoMatrix(genes(cuff))
> v
```

For individual pairwise comparisons, you must specify the comparisons by sample name.

```r
> v<-csVolcano(genes(cuff),"hESC","Fibroblasts")
> v
```
(a) Volcano plots explore the relationship between fold-change and significance.
6 Accessing Data

Cuffdiff run information

Run-level information such as run parameters, and sample information can be accessed from a CuffSet object by using the runInfo and replicates methods:

```r
> runInfo(cuff)

       param
  1  cmd_line
  2  version
  3  SVN_revision
  4  boost_version
  5  genome

1 cuffdiff -L iPS,hESC,Fibroblasts -p 6 chr1_snippet.gtf -o iPS_hESC_fibro iPS_rep1.bam,iPS_rep2.bam H1_rep1.bam,H1_rep3.bam NHLF_rep1.bam,NHLF_rep2.bam

> replicates(cuff)

    file sample_name replicate rep_name total_mass
  1  iPS_rep1.bam   iPS      0   iPS_0   173431
  2  iPS_rep2.bam   iPS      1   iPS_1   173007
  3  H1_rep1.bam   hESC      0  hESC_0   754749
  4  H1_rep3.bam   hESC      1  hESC_1   762643
  5 NHLF_rep1.bam Fibroblasts 0 Fibroblasts_0   876775
  6 NHLF_rep2.bam Fibroblasts 1 Fibroblasts_1  1412130

    norm_mass internal_scale external_scale
  1  706934   0.958068   0.584877
  2  706934   1.037970   0.584877
  3  706934   0.989851   1.513060
  4  706934   1.010250   1.513060
  5  706934   0.840416   1.223240
  6  706934   1.198470   1.223240

Features/Annotation

Feature-level information can be accessed directly from a CuffData object using the fpkm, repFpkm, count, diffData, or annotation methods:

```r
> gene.features<-annotation(genes(cuff))
> head(gene.features)

     gene_id class_code nearest_ref_id gene_short_name
  1  XLOC_000001     <NA>         <NA>            <NA>
  2  XLOC_000001     <NA>         <NA>            <NA>
  3  XLOC_000001     <NA>         <NA>            <NA>
  4  XLOC_000001     <NA>         <NA>            <NA>
  5  XLOC_000001     <NA>         <NA>            <NA>
  6  XLOC_000001     <NA>         <NA>            <NA>

  locus length coverage seqnames start end width
  1 chr1:11873-29961  NA  NA      chr1   11874   12227    354
  2 chr1:11873-29961  NA  NA      chr1   12646   12697    52
  3 chr1:11873-29961  NA  NA      chr1   13221   14409   1189
  4 chr1:11873-29961  NA  NA      chr1   11874   12227    354
chr1:11873-29961  NA  NA  chr1 12596 12721  127
chr1:11873-29961  NA  NA  chr1 13403 14409  1007

<table>
<thead>
<tr>
<th>strand</th>
<th>source</th>
<th>type</th>
<th>score</th>
<th>phase</th>
<th>isoform_id</th>
<th>exon_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>nearCoding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000003</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>nearCoding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000003</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>nearCoding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000003</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>coding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000002</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>coding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000002</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>coding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000002</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>strand</th>
<th>source</th>
<th>type</th>
<th>score</th>
<th>phase</th>
<th>isoform_id</th>
<th>exon_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>nearCoding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000003</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>nearCoding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000003</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>nearCoding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000003</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>coding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000002</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>coding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000002</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>coding</td>
<td>exon</td>
<td>NA</td>
<td>NA</td>
<td>TCONS_00000002</td>
<td>3</td>
</tr>
</tbody>
</table>

> gene.fpkm<-fpkm(genes(cuff))
> head(gene.fpkm)

gene_id sample_name fpkm conf_hi conf_lo
1  XLOC_000001 Fibroblasts  16.1506  182.9240  0.00000
2  XLOC_000002 Fibroblasts  0.0000   0.0000   0.00000
3  XLOC_000003 Fibroblasts  0.0000   0.0000   0.00000
4  XLOC_000004 Fibroblasts 14237.7000 78180.9000  0.00000
5  XLOC_000005 Fibroblasts  48.0566  90.6526  5.46055
6  XLOC_000006 Fibroblasts  0.0000   0.0000   0.00000

quant_status stdev
1  OK   83.3867
2  OK   0.0000
3  OK   0.0000
4  OK  31971.6000
5  OK   21.2980
6  OK   0.0000

> gene.repFpkm<-repFpkm(genes(cuff))
> head(gene.repFpkm)

gene_id sample_name replicate rep_name raw_frags
1  XLOC_000001 Fibroblasts 0 Fibroblasts_0 12.000000
2  XLOC_000002 Fibroblasts 0 Fibroblasts_0 0.000000
3  XLOC_000003 Fibroblasts 0 Fibroblasts_0 0.000000
4  XLOC_000004 Fibroblasts 0 Fibroblasts_0 0.333333
5  XLOC_000005 Fibroblasts 0 Fibroblasts_0 137.100000
6  XLOC_000006 Fibroblasts 0 Fibroblasts_0 0.000000

internal_scaled_frags external_scaled_frags fpkm
1    14.278800   11.672900  11.1815
2    0.000000    0.000000   0.0000
3    0.000000    0.000000   0.0000
4    0.396629    0.324245  28475.5000
5    163.134000  133.362000  57.8929
```r
> gene.counts <- count(genes(cuff))
> head(gene.counts)

gene_id sample_name count variance uncertainty
group1
1 XLOC_000001 Fibroblasts 20.072500 1.05160e+04 1.77636e-15
2 XLOC_000002 Fibroblasts 0.000000 0.00000e+00 0.00000e+00
3 XLOC_000003 Fibroblasts 0.000000 0.00000e+00 0.00000e+00
4 XLOC_000004 Fibroblasts 0.198315 1.98315e-01 0.00000e+00
5 XLOC_000005 Fibroblasts 131.575000 3.35882e+03 5.68434e-14
6 XLOC_000006 Fibroblasts 0.000000 0.00000e+00 0.00000e+00

> isoform.fpkm <- fpkm(isoforms(cuff))
> head(isoform.fpkm)

isoform_id sample_name fpkm conf_hi conf_lo
group1
1 TCONS_00000001 Fibroblasts 10.996300 118.178 0
2 TCONS_00000002 Fibroblasts 0.000000 0.000 0
3 TCONS_00000003 Fibroblasts 5.154340 132.926 0
4 TCONS_00000004 Fibroblasts 0.000000 0.000 0
5 TCONS_00000005 Fibroblasts 0.000000 0.000 0
6 TCONS_00000006 Fibroblasts 14237.70000 78180.900 0

> gene.diff <- diffData(genes(cuff))
> head(gene.diff)

gene_id sample_1 sample_2 status value_1 value_2
1 XLOC_000001 iPS hESC NOTEST 20.21750 3.47386e-01
2 XLOC_000002 iPS hESC NOTEST 0.00000 0.00000e+00
3 XLOC_000003 iPS hESC NOTEST 1.51396 0.00000e+00
4 XLOC_000004 hESC iPS NOTEST 20.21750 3.47386e-01
5 XLOC_000005 hESC iPS OK 0.00000 6.97259e+05
6 XLOC_000006 hESC iPS OK 1.51396 0.00000e+00

log2.fold_change test_stat p_value q_value
1 -5.86292 7.13525e-01 0.47552100 1.00000000
2 0.00000 0.00000e+00 1.00000000 1.00000000
3 0.00000 0.00000e+00 1.00000000 1.00000000
```
**Condition and feature names**

Vectors of sample names and feature names are available by using the `samples` and `featureNames` methods:

```r
> sample.names<-samples(genes(cuff))
> head(sample.names)
[1] "iPS"    "hESC"   "Fibroblasts"

> gene.featurenames<-featureNames(genes(cuff))
> head(gene.featurenames)
[1] "XLOC_000001" "XLOC_000002" "XLOC_000003" "XLOC_000004"
[5] "XLOC_000005" "XLOC_000006"
```

**Convenience functions**

To facilitate Bioconductor-like operations, an 'FPKM-matrix' can be returned easily using the `fpkmMatrix` method:

```r
> gene.matrix<-fpkmMatrix(genes(cuff))
> head(gene.matrix)

        iPS   hESC Fibroblasts
XLOC_000001 20.2175 3.47386e-01   16.1506
XLOC_000002  0.0000     0.0000e+00   0.0000
XLOC_000003  0.0000     0.0000e+00   0.0000
XLOC_000004  0.0000     6.97259e+05   14237.7000
XLOC_000005 355.8230  6.96704e+02   48.0566
XLOC_000006  1.51396  0.000000e+00   0.0000
```

A matrix of replicate FPKM values can be retrieved by using `repFpkmMatrix`:

```r
> gene.rep.matrix<-repFpkmMatrix(genes(cuff))
> head(gene.rep.matrix)

        iPS_0  iPS_1  hESC_0  hESC_1
XLOC_000001 17.2049 22.92880  0.000  6.21918e-01
XLOC_000002  0.0000     0.0000e+00  0.000          0.000000e+00
XLOC_000003  0.0000     0.0000e+00  0.000          0.000000e+00
XLOC_000004  0.0000     0.00000000 1377990.000  8.35811e+05
XLOC_000005 319.0300  6.96704e+02  48.0566          7.21983e+02
XLOC_000006  1.51396  0.00000000        0.000           0.00000000
```

**Fibroblasts_0  Fibroblasts_1**

```r
  XLOC_000001  11.1815  20.6009
  XLOC_000002  0.0000     0.0000
  XLOC_000003  0.0000     0.0000
  XLOC_000004  28475.5000  0.0000
  XLOC_000005  57.8929     38.0084
  XLOC_000006  0.0000     0.0000
```
Similarly, a matrix of normalized counts can be generated by using `countMatrix`

```r
> gene.count.matrix<-countMatrix(genes(cuff))
> head(gene.count.matrix)
```

```
       iPS     hESC Fibroblasts
XLOC_000001 11.440900 0.494996 20.072500
XLOC_000002  0.000000 0.000000  0.000000
XLOC_000003  0.000000 0.000000  0.000000
XLOC_000004  0.000000 5.680560  0.198315
XLOC_000005 486.456000 2495.560000 131.575000
XLOC_000006  0.481711  0.000000  0.000000
```

### 6.1 Writing your own SQL accessors

Since the cuffData.db is a SQLite database backend, if you are familiar with SQL and/or RSQLite query construction, you can simply design your own SQL queries to access the data that you are after.
7 Creating Gene Sets

Gene Sets (stored in a CuffGeneSet object) can be created using the getGenes method on a CuffSet object. You must first create a vector of 'gene_id' or 'gene_short_name' values to identify the genes you wish to select:

```r
> data(sampleData)
> myGeneIds<-sampleIDs
> myGeneIds

[1] "XLOC_001363" "XLOC_001297" "XLOC_001339" "XLOC_000132"
[5] "XLOC_001265" "XLOC_000151" "XLOC_001359" "XLOC_000069"
[9] "XLOC_000170" "XLOC_000105" "XLOC_001262" "XLOC_001348"
[13] "XLOC_001411" "XLOC_001369" "XLOC_000158" "XLOC_001370"
[17] "XLOC_001263" "XLOC_000115" "XLOC_000089" "XLOC_001240"

> myGenes<-getGenes(cuff,myGeneIds)
> myGenes

CuffGeneSet instance for 20 genes

Slots:

- annotation
- fpkm
- repFpkm
- diff
- count
- isoforms
- TSS
- CDS
- promoters
- splicing
- relCDS

The same fpkm, repFpkm, count, annotation, diffData, samples, and featureNames methods are available for instances of the CuffGeneSet class, but additional accessor methods are available for the promoters, relCDS, and splicing slot data as well.

```r
> #FPKM values for genes in gene set
> head(fpkm(myGenes))

    gene_id sample_name  fpkm conf_hi conf_lo
1  XLOC_000069    Fibroblasts 2.05083e-01  8.94501e-01     0.000
2  XLOC_000069       hESC 1.77686e+02 2.15888e+02 139.484
3  XLOC_000069        iPS 1.90000e+03 3.88149e+01  868.215
4  XLOC_000089    Fibroblasts 1.39701e+04 2.19187e+04 6021.450
5  XLOC_000089       hESC 7.18339e+03 7.92296e+03  644.3810
6  XLOC_000089        iPS 1.41690e+03 1.96558e+03  686.215

quant_status     stdev
1      OK  0.344709
2      OK  19.101000
3      OK  9.907450
4      OK  3974.300000
5      OK  369.785000
6      OK  274.340000
```

> #Isoform-level FPKMs for gene set
> head(fpkm(isoforms(myGenes)))
<table>
<thead>
<tr>
<th>isoform_id</th>
<th>sample_name</th>
<th>fpkm</th>
<th>conf_hi</th>
<th>conf_lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCONS_00000179</td>
<td>Fibroblasts</td>
<td>0.205083</td>
<td>0.894501</td>
<td>0.00000</td>
</tr>
<tr>
<td>TCONS_00000179</td>
<td>hESC</td>
<td>20.360900</td>
<td>33.840500</td>
<td>6.88119</td>
</tr>
<tr>
<td>TCONS_00000179</td>
<td>iPS</td>
<td>7.694340</td>
<td>16.356900</td>
<td>0.00000</td>
</tr>
<tr>
<td>TCONS_00000180</td>
<td>Fibroblasts</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.00000</td>
</tr>
<tr>
<td>TCONS_00000180</td>
<td>hESC</td>
<td>157.325000</td>
<td>198.673000</td>
<td>115.97800</td>
</tr>
<tr>
<td>TCONS_00000180</td>
<td>iPS</td>
<td>11.305600</td>
<td>27.105700</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>quant_status</th>
<th>stdev</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>0.344709</td>
</tr>
<tr>
<td>OK</td>
<td>6.739800</td>
</tr>
<tr>
<td>OK</td>
<td>4.331280</td>
</tr>
<tr>
<td>OK</td>
<td>0.000000</td>
</tr>
<tr>
<td>OK</td>
<td>20.674000</td>
</tr>
<tr>
<td>OK</td>
<td>7.900050</td>
</tr>
</tbody>
</table>

> # Replicate FPKMs for TSS groups within gene set
> head(repFpkm(TSS(myGenes)))

<table>
<thead>
<tr>
<th>TSS_group_id</th>
<th>sample_name</th>
<th>replicate</th>
<th>rep_name</th>
<th>raw_frags</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSS93</td>
<td>iPS</td>
<td>0</td>
<td>iPS_0</td>
<td>14.16970</td>
</tr>
<tr>
<td>TSS93</td>
<td>iPS</td>
<td>1</td>
<td>iPS_1</td>
<td>7.35392</td>
</tr>
<tr>
<td>TSS93</td>
<td>hESC</td>
<td>1</td>
<td>hESC_1</td>
<td>84.28660</td>
</tr>
<tr>
<td>TSS93</td>
<td>hESC</td>
<td>0</td>
<td>hESC_0</td>
<td>61.70490</td>
</tr>
<tr>
<td>TSS93 Fibroblasts</td>
<td>1</td>
<td>Fibroblasts_1</td>
<td>0.00000</td>
<td></td>
</tr>
<tr>
<td>TSS93 Fibroblasts</td>
<td>0</td>
<td>Fibroblasts_0</td>
<td>1.00000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>internal_scaled_frags</th>
<th>external_scaled_frags</th>
<th>fpkm</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.78990</td>
<td>25.287200</td>
<td>10.637900</td>
</tr>
<tr>
<td>7.08493</td>
<td>12.113500</td>
<td>5.095950</td>
</tr>
<tr>
<td>83.43110</td>
<td>55.140800</td>
<td>23.569500</td>
</tr>
<tr>
<td>62.33760</td>
<td>41.199800</td>
<td>17.610500</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>1.18989</td>
<td>0.972736</td>
<td>0.410165</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>effective_length</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA</td>
<td>OK</td>
</tr>
<tr>
<td>NA</td>
<td>OK</td>
</tr>
<tr>
<td>NA</td>
<td>OK</td>
</tr>
<tr>
<td>NA</td>
<td>OK</td>
</tr>
<tr>
<td>NA</td>
<td>OK</td>
</tr>
<tr>
<td>NA</td>
<td>OK</td>
</tr>
</tbody>
</table>

As of cummeRbund v2.0, CuffGeneSet classes can be created from any type of identifier ('gene_id', 'isoform_id', 'TSS_group_id', or 'CDS_id'). When you pass a list of identifiers that are not gene_id to getGenes(), the function attempts to lookup the parent gene_id for each feature and returns all relevant information for the given genes and all of their sub-features (not just the sub-features passed to getGenes()). If you are interested in just retrieving information for a given set of features, please use the new getFeatures() method described later.

More recent versions of cummeRbund allow for subsetting of conditions as well, by passing a vector of condition names to getGenes using the sampleIdList argument.

> myGeneset.pluri<-getGenes(cuff,myGeneIds,sampleIdList=c("hESC","iPS"))
> myGeneset.pluri

CuffGeneSet instance for 20 genes

Slots:
- annotation
- fpkm
- repFpkm
7.1 Geneset level plots

There are several plotting functions available for gene-set-level visualization:

The `csHeatmap()` function is a plotting wrapper that takes as input either a CuffGeneSet or a CuffFeatureSet object (essentially a collection of genes and/or features) and produces a heatmap of FPKM expression values. The 'cluster' argument can be used to re-order either 'row', 'column', or 'both' dimensions of this matrix. By default, the Jensen-Shannon distance is used as the clustering metric, however, any function that produces a `dist` object can be passed to the 'cluster' argument as well.

```r
> h<-csHeatmap(myGenes,cluster='both')
> h
> h.rep<-csHeatmap(myGenes,cluster='both',replicates=T)
> h.rep
```

If you prefer barplots over heatmaps for genesets (although this is not necessarily recommended for large gene sets). You can use the `expressionBarplot()` method on a CuffFeatureSet or a CuffGeneSet object.

```r
> b<-expressionBarplot(myGenes)
> b
```

The `csScatter()` method can be used to produce scatter plot comparisons between any two conditions.

```r
> s<-csScatter(myGenes,"Fibroblasts","hESC",smooth=T)
> s
```

The volcano plot is a useful visualization to compare fold change between any two conditions and significance (-log P-values).

```r
> v<-csVolcano(myGenes,"Fibroblasts","hESC")
> v
```

Similar plots can be made for all sub-level features of a CuffGeneSet class by specifying which slot you would like to plot (eg. `isoforms(myGenes),TSS(myGenes),CDS(myGenes)`).

```r
> ih<-csHeatmap(isoforms(myGenes),cluster='both',labRow=F)
> ih
> th<-csHeatmap(TSS(myGenes),cluster='both',labRow=F)
> th
```

Dendrograms can provide insight into the relationships between conditions for various genesets (e.g. significant genes used to draw relationships between conditions). As of v1.1.3 the method `csDendro()` can be used to plot a dendrogram based on Jensen-Shannon distances between conditions for a given CuffFeatureSet or CuffGeneSet.

```r
> den<-csDendro(myGenes)
```
(a) Heatmaps provide a convenient way to visualize the expression of entire gene sets at once.

(b) Same heatmap, with replicates=T can help to visualize variance between replicates.
(a) A (somewhat crowded) barplot for all genes in a CuffGeneSet object.

(b) Scatterplot showing relationship between two conditions for genes in a CuffGeneSet.
(a) Fold-change vs significance for genes in a CuffGeneSet object.

(a) A heatmap of isoform-level FPKM values for all genes in a CuffGeneSet object.

(b) A heatmap of TSS-level FPKM values for all genes in a CuffGeneSet object.
(a) A dendrogram of the relationship between conditions based on the expression of genes in a CuffGeneSet.
8 Individual Genes

An individual CuffGene object can be created by using the `getGene()` function for a given ‘gene_id’ or ‘gene_short_name’. As of cummeRbund ≥ v2.0 you can also use isoform_id, tss_group_id, or cds_id values to retrieve the corresponding parent gene object.

```r
> myGeneId<="PINK1"
> myGene<-getGene(cuff,myGeneId)
> myGene

CuffGene instance for gene XLOC_000172
Short name: PINK1
Slots:
  annotation
  features
  fpkm
  repFpkm
  diff
  count
  isoforms CuffFeature instance of size 2
  TSS CuffFeature instance of size 2
  CDS CuffFeature instance of size 2

> head(fpkm(myGene))
gene_id sample_name fpkm conf_hi conf_lo
1 XLOC_000172 Fibroblasts 3045.930 4529.840 1562.020
2 XLOC_000172 hESC 441.939 523.916 359.961
3 XLOC_000172 iPS 640.208 914.562 365.853
quant_status
1 OK
2 OK
3 OK

> head(fpkm(isoforms(myGene)))
isoform_id sample_name fpkm conf_hi conf_lo
1 TCONS_00000480 Fibroblasts 2213.850 3284.980 1142.710
2 TCONS_00000481 Fibroblasts 832.083 1760.820 0.000
3 TCONS_00000480 hESC 326.979 388.108 265.850
4 TCONS_00000481 hESC 114.960 166.530 63.389
5 TCONS_00000480 iPS 640.208 914.562 365.853
6 TCONS_00000481 iPS 0.000 0.000 0.000
quant_status
1 OK
2 OK
3 OK
4 OK
5 OK
6 OK

8.1 Gene-level plots

> gl<-expressionPlot(myGene)
> gl
> gl.rep<-expressionPlot(myGene,replicates=TRUE)
> gl.rep
> gl.iso.rep<-expressionPlot(isoforms(myGene),replicates=T)
> gl.iso.rep
> gl.cds.rep<-expressionPlot(CDS(myGene),replicates=T)
> gl.cds.rep

(a) Expression plot of a single gene.

(b) Expression plot of a single gene with replicate FPKMs exposed.

(c) Expression plot of all isoforms of a single gene with replicate FPKMs exposed.

(d) Expression plot of all CDS for a single gene with replicate FPKMs exposed.

> gb<-expressionBarplot(myGene)
> gb
> gb.rep<-expressionBarplot(myGene,replicates=T)
> gb.rep

> igb<-expressionBarplot(isoforms(myGene),replicates=T)
> igb

> gp<-csPie(myGene,level="isoforms")
> gp

28
(a) Expression Barplot of a single gene.

(b) Expression Barplot of a single gene with replicate FPKMs exposed.

(a) Expression Barplot of all isoforms single gene with replicates exposed.
(a) Pie charts showing relative proportion of individual isoforms for a single gene across conditions.
8.1.1 Gene Feature plots

If you included both the genome build and gtfFile in your call to `readCufflinks()` then you will be able to access some of the transcript-structure level features that are now being integrated into cummeRbund. For now, these features are extended only to the single gene, `CuffGene` objects.

Feature data are loaded into the `features` table of the `cuffData.db` database. When a `CuffGene` object is created using `getGene()`, all relative features are selected from this table and a `features` slot is added to the resulting object.

```r
> head(features(myGene))
```

```
seqnames start   end  width strand source type score
chr1   20959948 20960428 481   + coding exon NA
chr1   20964335 20964622 288   + coding exon NA
chr1   20966385 20966485 101   + coding exon NA
chr1   20970983 20971165 183   + coding exon NA
chr1   20972053 20972216 164   + coding exon NA
chr1   20974998 20975125 128   + coding exon NA
```

The `Gviz` package can be used to display features in a 'track'-like format. In particular, the `GeneRegionTrack` class creates a mechanism by which we can start to visualize transcript-level structures in their genomic context. `cummeRbund` implements the `makeGeneRegionTrack()` method to quickly create a `GeneRegionTrack` from the gene features.

```r
> genetrack<-makeGeneRegionTrack(myGene)
> plotTracks(genetrack)
```
We can then use some of the additional features from the Gviz package to add additional tracks from an external data source.

Note: This feature is now deprecated owing to developmental constraints. It is still provided with cummeRbund, but is no longer supported.

```r
> trackList<-list()
> myStart<-min(features(myGene)$start)
> myEnd<-max(features(myGene)$end)
> myChr<-unique(features(myGene)$seqnames)
> genome<-'hg19'
> ideoTrack <- IdeogramTrack(genome = genome, chromosome = myChr)
> trackList<-c(trackList,ideoTrack)
> axtrack<-GenomeAxisTrack()
> trackList<-c(trackList,axtrack)
> genetrack<-makeGeneRegionTrack(myGene)
> genetrack

GeneRegionTrack 'CuffDiff'
| genome: hg19
| active chromosome: chr1
| annotation features: 12

> trackList<-c(trackList,genetrack)
> biomTrack<-BiomartGeneRegionTrack(genome=genome,chromosome=as.character(myChr),
```
```r
> trackList<-c(trackList,biomTrack)
> conservation <- UcscTrack(genome = genome, chromosome = myChr,
+   track = "Conservation", table = "phyloP100wayAll",
+   from = myStart-2000, to = myEnd+2000, trackType = "DataTrack",
+   start = "start", end = "end", data = "score",
+   type = "hist", window = "auto", col.histogram = "darkblue",
+   fill.histogram = "darkblue", ylim = c(-3.7, 4),
+   name = "Conservation")
> trackList<-c(trackList,conservation)
> plotTracks(trackList,from=myStart-2000,to=myEnd+2000)
```

![Genome browser with conservation tracks](image)

**Chromosome 1**
20.96 mb
20.965 mb
20.97 mb
20.975 mb

**CuffDiff**
TCONS_00000480
TCONS_00000481

**ENSEMBL**
PINK1
DDOST
PINK1
DDOST
PINK1-AS
DDOST
PINK1
AL391357.1

**Conservation**
-2
0
2
4

![Conservation track](image)
9 Data Exploration

The cummeRbund package is more than just a visualization tool as well. We are working to implement several different means of data exploration from gene and condition clustering, finding features with similar expression profiles, as well as incorporating Gene Ontology analysis.

9.1 Overview of significant features

The \texttt{sigMatrix()} function can provide you with a “quick-and–dirty” view of the number of significant features of a particular type, and at a given alpha (0.05 by default). For example:

\begin{verbatim}
> mySigMat<-sigMatrix(cuff,level='genes',alpha=0.05)

(a) Significant features overview matrix. This plot describes the number of significant genes at a 5\%FDR for each pairwise interaction tested.
\end{verbatim}

9.2 Creating gene sets from significantly regulated genes

One of the primary roles of a differential expression analysis is to conduct significance tests on each feature (genes, isoforms, TSS, and CDS) for appropriate pairwise comparisons of conditions. The results of these tests (after multiple testing correction of course) can be used to determine which genes are differentially regulated. \texttt{cummeRbund} makes accessing the results of these significance tests simple via \texttt{getSig()}. This function takes a CuffSet object and will scan at various feature levels (‘genes’ by default) to produce a vector of feature IDs. By default \texttt{getSig() } outputs a vector of tracking IDs corresponding to all \texttt{genes} that reject the null hypothesis in any condition tested. The default feature type can be changed by adjusting the ‘level’ argument to \texttt{getSig()}. In addition, a alpha value can be provided on which to filter the resulting list (the default is 0.05 to match the default of cuffdiff).

\begin{verbatim}
> mySigGeneIds<-getSig(cuff,alpha=0.05,level='genes')
> head(mySigGeneIds)
\end{verbatim}
By default, `getSig()` outputs a vector of tracking IDs corresponding to all genes that reject the null hypothesis in any condition tested. The default feature type can be changed by adjusting the 'level' argument to `getSig()`. In addition, an alpha value can be provided on which to filter the resulting list (the default is 0.05 to match the default of cuffdiff). Significance results for specific pairwise comparisons can be retrieved as well by specifying the two conditions as 'x' and 'y'. In this case, p-values are adjusted to reduce the impact of multiple-testing correction when only one set of tests is being conducted.

```r
> hESC_vs_iPS.sigIsoformIds <- getSig(cuff, x='hESC', y='iPS', alpha=0.05, level='isoforms')
> head(hESC_vs_iPS.sigIsoformIds)
```

The values returned for each level of this list can be used as an argument to `getGenes`, to create a `CuffGeneSet` object of significantly regulated genes (or features).

```r
> mySigGenes <- getGenes(cuff, mySigGeneIds)
> mySigGenes
```

CuffGeneSet instance for 207 genes

Slots:

- annotation
- fpkm
- repFpkm
- diff
- count
- isoforms
- TSS
- CDS
- promoters
- splicing
- relCDS

Alternatively, you can use the `getSigTable()` method to return a full test-table of 'significant features' x 'pairwise tests' for all comparisons. Only features in which the null hypothesis can be rejected in at least one test are reported.

```r
> mySigTable <- getSigTable(cuff, alpha=0.01, level='genes')
> head(mySigTable, 20)
```
9.3 Exploring the relationships between conditions

9.3.1 Distance matrix

Similarities between conditions and/or replicates can provide useful insight into the relationship between various groupings of conditions and can aid in identifying outlier replicates that do not behave as expected. cummeRbund provides the `csDistHeat()` method to visualize the pairwise similarities between conditions:

```r
> myDistHeat <- csDistHeat(genes(cuff))
> # (a) JS distance heatmap between conditions across all gene features.
```

Again with the `replicates` argument, distances between individual replicates can be presented.

```r
> myRepDistHeat <- csDistHeat(genes(cuff), replicates = T)
> #
```
This method can be used to explore similarities between conditions for all features, or just those features contained within a `CuffGeneSet` class. Additionally, the `samples.not.genes=F` argument will display distances between individual genes or features across conditions.

### 9.3.2 Dimensionality reduction

Dimensionality reduction is an informative approach for clustering and exploring the relationships between conditions. It can be useful for feature selection as well as identifying the sources of variability within your data. To this end, we have applied two different dimensionality reduction strategies in `cummeRbund`: principal component analysis (PCA) and multi-dimensional scaling (MDS). We provide the two wrapper methods, `PCAplot` and `MDSplot`.

```r
> genes.PCA<-PCAplot(genes(cuff),"PC1","PC2")
> genes.MDS<-MDSplot(genes(cuff))
> genes.PCA.rep<-PCAplot(genes(cuff),"PC1","PC2",replicates=T)
> genes.MDS.rep<-MDSplot(genes(cuff),replicates=T)
```

CummeRbund also includes a convenience wrapper around the `NMFN` function `nnmf` for non-negative matrix factorization. You can use the `csNMF()` method for either CuffData and CuffFeatureSet objects.
(a) PCA plot for gene-level features

(b) MDS plot for gene-level features

(c) Individual replicate level PCA plot for gene-level features

(d) Individual replicate level MDS plot for gene-level features
9.4 Partitioning

K-means clustering is a useful tool that can be helpful in identifying clusters of genes with similar expression profiles. In fact, these profiles are learned from the data during the clustering. `csCluster()` uses the `pam()` method from the `clustering` package to perform the partitioning around medoids. In this case however, the distance metric used by default is the Jensen-Shannon distance instead of the default Euclidean distance. Prior to performing this particular partitioning, the user must choose the number of clusters (K) into which the expression profiles should be divided.

```r
> ic<-csCluster(myGenes,k=4)
> head(ic$cluster)

XLOC_000069 XLOC_000089 XLOC_000105 XLOC_000115 XLOC_000132
   1  2  2  3  2
XLOC_000151
   1

> icp<-csClusterPlot(ic)
> icp
```

As of v1.1.1 of `cummeRbund`, the output of csCluster is a modified `pam` object. This replaces the default plotting behavior of the original csCluster plot to allow for further analysis of the clustering results. The original plotting behavior has been recapitulated in the `csClusterPlot()` method.

![PAM clustering with JS distance for a CuffGeneSet.](image)

(a) PAM clustering with JS distance for a CuffGeneSet.
9.5 Specificity

In some cases, a researcher may be interested in identifying features that are 'condition-specific'. Or, more likely, producing an ordered list of genes based on their specificity for a given condition. We define a specificity score \( S \) as the following:

\[
S_{g,i} = 1 - JSD(p_g, \hat{q}_i)
\]

(1)

Where \( JSD \) is the Jensen-Shannon distance, \( p_g \) is the expression profile of a given gene \( g \) expressed as a density (probability) of \( \log_{10} FPKM + 1 \), and \( \hat{q}_i \) is the unit vector of 'perfect expression' in a particular condition \( i \).

We have created a method, \texttt{csSpecificity()} that outputs a matrix (with identical shape to that produced by \texttt{fpkmMatrix()}) of specificity scores \( S \) across all conditions for all features in a \texttt{CuffFeatureSet} or \texttt{CuffGeneSet}.

\[
\texttt{myGenes.spec<-csSpecificity(myGenes)}
\]

\[
\texttt{head(myGenes.spec)}
\]

\[
\begin{array}{cccc}
\text{iPS\_spec} & \text{hESC\_spec} & \text{Fibroblasts\_spec} \\
\text{XLOC\_000069} & 0.3404865 & 0.5281363 & 0.03949159 \\
\text{XLOC\_000089} & 0.2843602 & 0.3315380 & 0.35041646 \\
\text{XLOC\_000105} & 0.3006743 & 0.3138327 & 0.35217793 \\
\text{XLOC\_000115} & 1.0000000 & 0.0000000 & 0.0000000 \\
\text{XLOC\_000132} & 0.2940644 & 0.3073414 & 0.36481155 \\
\text{XLOC\_000151} & 0.4977991 & 0.3247807 & 0.11444635 \\
\end{array}
\]

\( S = 1.0 \) if the feature is expressed exclusively in that condition. The \texttt{findSimilar()} method outlined below is another method that can be used to identify genes based on specificity but has the added feature that you can determine similarity to a more complex \( q \) expression profile.
9.6 Finding similar genes

Another common question in large-scale gene expression analyses is 'How can I find genes with similar expression profiles to gene x?'. We have implemented a method, `findSimilar` to allow you to identify a fixed number of the most similar genes to a given gene of interest. For example, if you wanted to find the 20 genes most similar to "PINK1", you could do the following:

```r
> mySimilar<-findSimilar(cuff,"PINK1",n=20)
> mySimilar.expression<-expressionPlot(mySimilar,logMode=T,showErrorbars=F)
```

(a) Top 20 most similar genes to 'PINK1'.

By default, `findSimilar` will return a CuffGeneSet of similar genes matching your criteria. Recently a few additional features have been added as well to enhance this type of exploration:

- If 'returnGeneSet' is set to FALSE, then `findSimilar` returns a data.frame of distance-ranked similar genes with distances. This is useful if you would like to see a rank-ordered list of similar genes.
- The 'distThresh' argument allows you to pass a value [between 0-1] to be used as a distance threshold instead of an arbitrary 'n' number of genes. Setting `distThresh=1.0` will return all genes ranked by their distance to your gene of interest.

You are also able to provide your own expression profile in lieu of a 'gene_id'. The vector provided must match the order and length of `samples()`.

```r
> myProfile<-c(500,0,400)
> mySimilar2<-findSimilar(cuff,myProfile,n=10)
> mySimilar2.expression<-expressionPlot(mySimilar2,logMode=T,showErrorbars=F)
```

`findSimilar()` also uses the Jensen-Shannon distance between the probability distributions of each gene across conditions to determine the similarity. We have found this to be a more robust way to determine distance between genes using the high dynamic range of FPKM data. Future versions may allow for other dissimilarity measures to be used instead.
(a) Top 10 genes most similar genes to a provided expression profile.
10 Miscellaneous

- In appropriate plots, using the argument replicates=T will allow you to visualize replicate-level FPKM values either in lieu of or in addition to condition-level FPKMs.

- As of v1.1.3 we attempt to provide new visual cues in most plots that will indicate the quantification status for a particular feature in each given condition. We have enabled this feature by default for most plots to suggest a measure of reliability for each feature in a particular condition. In most cases, this feature can be disabled by setting ‘showStatus=FALSE’.

- CummeRbund will now work with the hidden ‘–no-diff’ argument for cuffdiff. This will quantify features against .bam files but not do differential testing. This is useful when you want to aggregate very large numbers of conditions, and cannot afford the time or space for the differential test results. (Not recommended unless you have a SPECIFIC need for this).

- All plotting functions return ggplot objects and the resulting objects can be manipulated/faceted/altered using standard ggplot2 methods.

- There are occasional DB connectivity issues that arise. Not entirely sure why yet. If necessary, just readCufflinks again and this should solve connectivity issues with a new RSQLite connection object. If connectivity continues to be a problem, try cuff<-readCufflinks(rebuild=T)

- I am still working on fully documenting each of the methods. There are a good number of arguments that exist, but might be hard to find without looking at the reference manual.
11 Known Issues

- Large cuffdiff runs (e.g. ≥10 conditions) produce very large results files. These will take some time to parse and populate the cuffData.db sqlite database. While this is only a one time cost, the process can take a while. We are working on making the table writes and indexing significantly faster.

- Cuffdiff does not ‘require’ that gene_ids, isoform_ids, TSS_group_ids, or CDS_ids be unique in your reference gtf file. In fact, duplicate IDs will be aggregated by cummeRbund in the indexing phase and will produce undesirable effects. Please ensure that all of your IDs are unique prior to running cuffdiff (see cuffmerge for help) to avoid this issue.
> sessionInfo()

R version 3.3.1 (2016-06-21)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
[1] LC_CTYPE=en_US.UTF-8   LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8    LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8   LC_NAME=C
[9] LC_ADDRESS=C           LC_TELEPHONE=C

attached base packages:
[1] grid     stats4    parallel    stats    graphics    grDevices
[7] utils    datasets  methods    base

other attached packages:
[1] cluster_2.0.5       cummeRbund_2.16.0
[3] Gviz_1.18.0          rtracklayer_1.34.0
[5] GenomicRanges_1.26.0 GenomeInfoDb_1.10.0
[7] IRanges_2.8.0        S4Vectors_0.12.0
[9] fastcluster_1.1.21   reshape2_1.4.1
[11] ggplot2_2.1.0        RSQLite_1.0.0
[13] DBI_0.5-1            BiocGenerics_0.20.0

loaded via a namespace (and not attached):
[1] SummarizedExperiment_1.4.0  VariantAnnotation_1.20.0
[3] splines_3.3.1               lattice_0.20-34
[5] colorspace_1.2-7            htmltools_0.3.5
[7] GenomicFeatures_1.26.0      chron_2.3-47
[9] interactiveDisplayBase_1.12.0 XML_3.98-1.4
[11] survival_2.39-5            foreign_0.8-67
[13] ensembldb_1.6.0             BiocParallel_1.8.0
[15] RColorBrewer_1.1-2          matrixStats_0.51.0
[17] plyr_1.8.4                  stringr_1.1.0
[19] zlibbioc_1.20.0             Biostrings_2.42.0
[21] munsell_0.4.3               gtable_0.2.0
[23] labeling_0.3                latticeExtra_0.6-28
[25] Biobase_2.34.0              biomaRt_2.30.0
[27] BiocInstaller_1.24.0        httpuv_1.3.3
[29] AnnotationDbi_1.36.0        Rcpp_0.12.7
[31] xtable_1.8-2                acepack_1.3-3.3
[33] BSgenome_1.42.0             scales_0.4.0
[35] Hmisc_3.17-4                XVector_0.14.0
[37] mime_0.5                    Rsamtools_1.26.0
[39] gridExtra_2.2.1             AnnotationHub_2.6.0
[41] digest_0.6.10               stringi_1.1.2
[43] biovizBase_1.22.0           shiny_0.14.1
[45] tools_3.3.1                 bitops_1.0-6
[47] magrittr_1.5                RCurl_1.95-4.8
[49] dichromat_2.0-0             Formula_1.2-1
[51] Matrix_1.2-7.1              data.table_1.9.6
[53] httr_1.2.1                   R6_2.2.0
[55] rpart_4.1-10                 GenomicAlignments_1.10.0