JnJCodelinkAPI

October 4, 2012

`JnJCodelinkAPI-package` *An interface to access Codelink data generated at Johnson & Johnson Research Center.*

Description

The API can be used to extract expression data and can also be used to search for similar compounds using connectivity map approach.

Note: This is a beta version and might have some bugs. Please report any issues or comments to the maintainer.

Details

Package:	JnJCodelinkAPI
Type:	Package
Version:	1.4
Date:	2012-10-02
License:	JnJ internal use

Author(s)

Vamsi Veeramachaneni <`vamsi@strandls.com`>, Komal Singh Sandhu <`skomal@strandls.com`>

References

Zhang, S. D. and Gant, T. W. (2008). A simple and robust method for connecting small-molecule drugs using gene-expression signatures. *BMC Bioinformatics* **9**, 258.

`probeLevelData` *Probe level Codelink expression data from JnJ.*

Description

The dataset contains probe level expression data for 122 paradigm compounds run on GE Codelink Rat Bioarray platform by Johnson & Johnson. Some compounds were run at multiple dose levels and durations. The compounds were run in multiple batches - each batch containing some vehicle/control samples and treatment samples. The data was background corrected, log (base 2) transformed and quantile normalized. Outlier samples were removed after manual inspection and the median of controls subtracted from treatments to produce log fold change data. The array has 35129 features mapping to 33889 unique Codelink probes.

Usage

```
data(probeLevelData)
```

See Also

`geneLevelData`

Examples

```
data(probeLevelData)    # Loads Codelink probe level data
```

`geneLevelData` *Gene level Codelink expression data from JnJ.*

Description

The dataset contains gene level expression data for 122 paradigm compounds run on GE Codelink Rat Bioarray platform by Johnson & Johnson. Some compounds were run at multiple dose levels and durations. The compounds were run in multiple batches - each batch containing some vehicle/control samples and treatment samples. The data was background corrected, log (base 2) transformed and quantile normalized. Outlier samples were removed after manual inspection and the median of controls subtracted from treatments to produce log fold change data. This probe/feature level data is available using the function probeLevelData and has 35129 features. After careful re-annotation it was found that there were only 9212 unique un-ambiguous probes. Gene level data contains expression values for only these probes. This data is useful for cross-platform comparisons (using methods like connectivity map) as the probe names are replaced by unique Entrez Ids.

Usage

```
data(geneLevelData)
```

See Also

`probeLevelData`

Examples

```
data(geneLevelData)    # Loads Codelink gene level data
```

`dataMetaData` *METADATA for Codelink gene expression data.*

Description

Meta-data for Codelink expression data containing 7 variables i.e., LotID, BatchID, AnimalID, Compound, Dose, Duration and Route.

Usage

```
data(dataMetaData)
```

Examples

```
data(dataMetaData)    # Loads meta-data
```

`listCompounds` *List compounds in the Codelink dataset released by JnJ.*

Description

This function is an interface to the Codelink dataset. The method will list all the 122 paradigm compounds present in the database along with their Dose and Durations.

Usage

```
listCompounds()
```

Value

An object of class list. The names in the list are Compound names and the values are Dose and Duration in the format: Dose_Duration. Duration may be represented as 1d (for 1 day), 24h (for 24 hours) etc.

Author(s)

Strand Life Sciences Pvt. Ltd.

See Also

`getExpressionData`, `getAnnotation`

Examples

```
dataDetails <- listCompounds()
```

`getExpressionData` *Get expression data for the specified Codelink compounds.*

Description

This function is an interface to the Codelink dataset. The method will generate an object of class ExpressionSet, containing expression data for specified Compounds at given Dose and Durations.

Usage

```
getExpressionData(Compound, Dose = "All", Duration = "1d",
                             Type = c("GeneLevel", "ProbeLevel"))
```

Arguments

`Compound`	a vector containing one or more Compound names.
`Dose`	a specific dose of the desired compound. In case of multiple compounds, this will be ignored.
`Duration`	a vector with single value specifying a duration for which data set has to be generated.
`Type`	a character string indicating which expression data has to be generated. This must be one of “GeneLevel”, “ProbeLevel”.

Details

For ‘Compound’, one can specify a single compound name or a charcter vector of many compounds.

‘Dose’ must be numeric and will be used if only one compound is specified. For a single compound, this will give data at all or specified doses.

‘Duration’ must be a string of format ‘1d’ or ‘6h’ etc.

If ‘Type’ is “ProbeLevel”, the returned expressionSet will have 35129 rows with FeatureIds as row identifiers. The mapping for FeatureId to Codelink ProbeID can be found in featureData of the ExpressionSet. If ‘Type’ is “GeneLevel”, the returned expressionSet will have 9212 rows with EntrezIds as row identifiers. The mapping for EntrezId to Codelink ProbeID can be found in featureData of the ExpressionSet.

Value

An object of class ExpressionSet, containing expression data for specified compounds.

Author(s)

Strand Life Sciences Pvt. Ltd.

See Also

`listCompounds`, `getAnnotation`, `probeLevelData`, `geneLevelData`

Examples

```
data(geneLevelData) # Loads JnJ gene level data
data(probeLevelData) # Loads JnJ probe level data
expData <- getExpressionData(Compound="Phenytoin", Type="ProbeLevel")
expData <- getExpressionData(Compound="Gabapentin", Dose=c("1000", "2160"), Duration="1d",
                                                                    Type="GeneLevel")

### For Multiple compounds ###
expData <- getExpressionData(Compound=c("Gabapentin", "Flufenamic acid", "Metoprolol", "Digoxin"),
                                                                    Type="GeneLevel")
```

`CodelinkGeneLevelAnnotations`

Re-derived Codelink Rat Bioarray annotations.

Description

This dataset contains annotations for the GE Codelink Rat Bioarray. These annotations were derived from scratch using sequence alignments against latest builds of Genome and Transcriptome.

Usage

```
data(CodelinkGeneLevelAnnotations)
```

Format

A data frame with 9212 observations on the following 4 variables.

`EntrezId` Entrez Ids associated with the probes

`Symbol` Gene symbols associated with probes

`FeatureId` A Codelink probe can be present as multiple features

`CodelinkId` Unique probe Ids for each Entrez Id

Details

These annotations are largely in concordance with those derived by Ensembl. The Ensembl annotations can be accessed through the package **biomaRt** in Bioconductor.

References

`http://www.biomart.org/`, `http://www.bioconductor.org/packages/2.2/bioc/html/biomaRt.html`

Examples

```
data(CodelinkGeneLevelAnnotations)    # Loads gene level annotations
```

`CodelinkRwgcodAnnotations` *Codelink Rat Bioarray annotations provided by RWGCOD*

Description

This dataset contains annotations for the GE Codelink Rat Bioarray provided as an R package called **rwgcod.db**, available at `www.bioconductor.org`

Usage

```
data(CodelinkRwgcodAnnotations)
```

Format

A data frame with 35129 observations on the following 5 variables.

`CodelinkId`	Codelink probe Ids
`Accession`	NCBI accession numbers
`UnigeneId`	Unigene cluster id
`Symbol`	Gene symbols
`EntrezId`	Entrez Ids

Examples

```
data(CodelinkRwgcodAnnotations)    # Loads Rwgcod annotations
```

`getAnnotation` *Get annotations associated with the Codelink datasets.*

Description

This function is an interface to annotations associated with the Codelink datasets included in this package. Two datasets are available - a “ProbeLevel” dataset consisting of measurements associated with the individual Codelink features, and a “GeneLevel” dataset consisting only of probes that could be mapped unambiguously to Entrez Gene Ids.

This method will return a table containing annotations for the specified dataset.

Usage

```
getAnnotation(dataType = c("GeneLevel", "ProbeLevel"))
```

Arguments

`dataType`	There are different sets of annotations available for “GeneLevel” data and “ProbeLevel” data.

Value

If the datatype is “GeneLevel”, the returned table has 4 columns EntrezId, Symbol, FeatureId and CodelinkId. The table is a result of our reannotation pipeline and consists of 9212 high-confidence Entrez ID to Codelink ID associations. The table contains:

`EntrezId`	Entrez Ids associated with the probes
`Symbol`	Gene symbols associated with probes
`FeatureID`	A Codelink probe can be present as multiple features
`CodelinkId`	Unique probe Ids for each Entrez Id

If the datatype is “ProbeLevel”, the returned table is identical to the **rwgcod** package annotations and provides annotations associated with the 35129 features on the Codelink platform. The returned table contains:

`CodelinkId`	Unique probe Ids for each Entrez Id
`Accession`	NCBI accession numbers
`UnigeneId`	Unigene ids associated with probes
`Symbol`	Gene symbols associated with probes
`EntrezId`	Entrez Ids associated with the probes

Note

The GeneLevel annotations are developed in-house. They are largely in agreement with the Ensembl annotations for Codelink, which can be accessed through **biomaRt**.

Author(s)

Strand Life Sciences Pvt. Ltd.

See Also

`listCompounds`, `getExpressionData`, `CodelinkGeneLevelAnnotations`, `CodelinkRwgcodAnnotations`

Examples

```
geneLevelAnnotation <- getAnnotation(dataType="GeneLevel")
probeLevelAnnotation <- getAnnotation(dataType="ProbeLevel")
```

---

connectivityCalculator

Computes connection score for a given query.

Description

Given a query entity list with p-value and log fold-changes, reports connection score with each of the reference samples. If specified, the p-value and log fold-change parameters will be used to filter the query entity list. If limitProbes is specified, the top fold-change probes will be used in similarity computation.

Usage

```
connectivityCalculator(query, fc = 1.5, p.val = 0.1, limitProbes = NULL)
```

Arguments

`query`	A two-column numeric matrix. First column should be log fold-change values and second column should be p-values. Rownames should be Entrez ids.
`fc`	Fold change cut-off for filtering genes.
`p.val`	p-value cut-off for filtering genes.
`limitProbes`	Maximum number of query probes to be used for similarity searching.

Value

`connectivityScores`	A matrix with rownames as reference samples and one column with corresponding connectivity scores.
`absentProbes`	A vector of Entrez ids for the probes that were present in the query but not in the JnJ reference database.

Author(s)

Strand Life Sciences Pvt. Ltd.

References

Zhang, S. D. and Gant, T. W. (2008). A simple and robust method for connecting small-molecule drugs using gene-expression signatures. *BMC Bioinformatics* **9**, 258.

Examples

```
data(sampleQuery)    # Loading sample query
res <- connectivityCalculator(query=sampleQuery, p.val=0.1,
                                          fc=1.5)   # Computing connection scores using wrapper-function
head(res$connectivityScores)
head(res$absentProbes)

##### to plot connectivity curve #####
plot(1:length(res$connectivityScores), sort(res$connectivityScores, decreasing=T),
           type="l", col="red", xlab="Reference Samples", ylab="Connection Score")

##### Display best 5 connected compounds #####
print(as.matrix(res$connectivityScores[1:5,]))
```

`rankedRefDB` *Ranked reference database created from JnJ Codelink Database.*

Description

A numeric matrix with log fold change values replaced by signed ranks. For each array, the absolute log fold-change values are sorted and each value is replaced by it's rank with smaller ranks given to small fold change values. The ranks are then multiplied by their actual direction of change (+ or -). The ranked reference database is created only for the "geneLevelData" as only that can be used for connectivity mapping. This database is not used directly but is used by the connectivity score calculation function.

Usage

```
data(rankedRefDB)
```

Examples

```
data(rankedRefDB)
```

`createRankedQuery` *Creates a ranked query for the given query expression data.*

Description

This method creates a ranked query, i.e. a query with signed ranks for each probe instead of log fold change values. The query probes are sorted by the absolute log fold-change values and each value is then replaced by it's rank. Smaller log fold change gets smaller ranks. The ranks are then multiplied by the direction of change (+ or -).

Usage

```
createRankedQuery(queryFC)
```

Arguments

`queryFC`	Query as a named vector (Entrez Id as names) of fold change values.

Value

A numeric vector of query probes with signed ranks instead of fold change values.

Author(s)

Strand Life Sciences Pvt. Ltd.

Examples

```
filteredGenes <- filterGenes(query=sampleQuery, fc=1.25, p.val=0.1, limitProbes=NULL)   #Filters query probes
rankedQuery <- createRankedQuery(queryFC=filteredQuery)    # Ranks the query probes
```

filterGenes *Filters the genes bases on given p Value and fold changes.*

Description

Given a two-column sample query where the first column contain log fold-change values and the second column contains p-values, this function filters the rows based on the given cut-off values for fold change and p value. If limitProbes is specified, only the top probes with highest absolute log fold-changes will be retained.

Usage

```
filterGenes(query, fc=1.5, p.val=0.1, limitProbes=NULL)
```

Arguments

`query`	A two-column numeric matrix. First column should be log fold-change values and second column should be p-values. Rownames should be Entrez ids.
`fc`	Fold change cut-off for filtering genes.
`p.val`	Uncorrected p-value cut-off for filtering genes.
`limitProbes`	Maximum number of query probes to be used for calculating connection scores.

Value

A numeric vector containing query as a named vector (Entrez Id as names) of log fold-change values.

Author(s)

Strand Life Sciences Pvt. Ltd.

Examples

```
filteredGenes <- filterGenes(query=sampleQuery, fc=1.5, p.val=0.1)
                                    #Filters query probes using p-value and fc filter

filteredGenes <- filterGenes(query=sampleQuery, fc=1.25, p.val=0.1, limitProbes=500)
                              #Filters query probes using p-value, fc and limitProbes filter
```

maxConnectionScore *Computes maximum theoretical connection score.*

Description

This function computes maximum theoretical connection score for a given sized query and reference database. For a fixed size of query and reference database, this score remains constant. It is used internally by the connectivityCalculator wrapper.

Usage

```
maxConnectionScore(rankedRefDB, rankedQuery)
```

Arguments

`rankedQuery`	named vector (probe ID as names) of signed ranks, sorted in decreasing order. Can be generated using method createRankedQuery.
`rankedRefDB`	a numeric matrix where each column consists of signed ranks for a reference sample. By default, this is ranked reference database generated from JnJ codelink gene-level data. The rankedRefDB being passed to this function is not subsetted on probes common between Query and Reference.

Value

A numeric vector with maximum possible connection score.

Author(s)

Strand Life Sciences Pvt. Ltd.

References

Zhang, S. D. and Gant, T. W. (2008). A simple and robust method for connecting small-molecule drugs using gene-expression signatures. *BMC Bioinformatics* **9**, 258.

Examples

```
maxScore <- maxConnectionScore(rankedRefDB=rankedRefDB, query=sampleQuery)
```

`computeConnectionScore` *Computes connection scores.*

Description

This function computes a connection score for a ranked query, created using createRankedQuery from log-ratio data, against all samples present in the ranked reference database, using a method described by Zhang and Gant (2008). The ranked reference database has been created from JnJ codelink gene-level data and is packaged as rankedRefDB. This function is used by the connectivityCalculator wrapper and is not intended for direct use.

Usage

```
computeConnectionScore(rankedRefDB, rankedQuery, maxTheoreticalScore)
```

Arguments

`rankedQuery`	named vector (probe ID as names) of signed ranks, sorted in decreasing order. Can be generated using method createRankedQuery.
`rankedRefDB`	a numeric matrix where each column consists of signed ranks for a reference sample. By default, this is ranked reference database generated from JnJ codelink gene-level data. The rankedRefDB is subsetted on probes common between Query and Reference.
`maxTheoreticalScore`	maximum possible connection score for a given sized query and reference. Can be generated using maxConnectionScore function.

Details

Given a ranked reference database and a ranked query, the method will generate connection score of the query against each sample in the reference database.

Value

A named numeric vector of connection scores with the names being the reference samples.

Author(s)

Strand Life Sciences Pvt. Ltd.

References

Zhang, S. D. and Gant, T. W. (2008). A simple and robust method for connecting small-molecule drugs using gene-expression signatures. *BMC Bioinformatics* **9**, 258.

See Also

`createRankedQuery`

Examples

```
data(rankedRefDB)    # Loads Codelink ranked reference data.
data(sampleQuery)    # Loads sample query
cs <- connectivityCalculator(query=sampleQuery, fc=1.5, p.val=0.1, limitProbes=NULL)
```