Package ‘PrecisionTrialDrawer’

October 16, 2020

Type Package

Title A Tool to Analyze and Design NGS Based Custom Gene Panels

Version 1.4.0

Date 2018-10-01

Author Giorgio Melloni, Alessandro Guida, Luca Mazzarella

Maintainer Giorgio Melloni <melloni.giorgio@gmail.com>

Description A suite of methods to design umbrella and basket trials for precision oncology.

License GPL-3

Depends R (>= 3.6)

Imports graphics, grDevices, stats, utils, methods, cgdsr, parallel, stringr, reshape2, data.table, RColorBrewer, BiocParallel, magrittr, biomaRt, XML, httr, jsonlite, ggplot2, ggrepel, grid, S4Vectors, IRanges, GenomicRanges, LowMACAAnnotation, googleVis, shiny, shinyBS, DT, brglm, matrixStats

Suggests BiocStyle, knitr, rmarkdown, dplyr

VignetteBuilder knitr

biocViews SomaticMutation, WholeGenome, Sequencing, DataImport, GeneExpression

RoxygenNote 6.1.0

git_url https://git.bioconductor.org/packages/PrecisionTrialDrawer

git_branch RELEASE_3_11

git_last_commit 7ded9c6

git_last_commit_date 2020-04-27

Date/Publication 2020-10-15

R topics documented:

`PrecisionTrialDrawer-package`
PrecisionTrialDrawer: a package to analyze and design custom cancer panels

---

Description

PrecisionTrialDrawer is a package that a allows the design and study of a cancer panel to create pilot sets for a clinical trial.

Details

It allows simulations on real TCGA data to design a possible clinical trial, both as basket or umbrella design.

Furthermore, it can provide an easy function to calculate genomic space occupied by the cancer panel in a commonly used bed format ready to be submitted to a NGS company.

Author(s)

Giorgio Melloni, Alessandro Guida

References

Cbioportal web site MD Anderson TCGA Fusion Portal

Examples

```
# Retrieve a Cancer Panel
data(cpObj)
# Show the number of covered samples
coveragePlot(cpObj
            , alterationType=c("mutations" , "copynumber")
            , grouping=c("group" , "tumor_type"))
```

---

appendRepo *appendRepo: Add extra population data*

Description

Function that allows to add extra samples for running simulations. If the user wants to add custom data to an existing cBioportal dataset, this function allows to append new samples to the panel to increase its inference power. This function takes care of appending to the panel a list of alterations compatible with the @dataFull slot (the slot that stores the cBioportal downloaded info).

Usage

```
appendRepo(object, repos)
```

Arguments

`object`	An instance of class CancerPanel
`repos`	A list() in format compatible to @dataFull slot

Details

dataFull slot is a named list of 4 elements: "fusions" , "mutations" , "copynumber" , "expression". Each element is a list itself of two named elements: "data" (a data.frame), "Samples" (a list of character vectors). They must be all present, but they can be NULL if no data are provided. Check example to check correct format of colnames inside each data type.

At the end of the appending, subsetAlterations is automatically run.

Value

An updated instance of class CancerPanel

Author(s)

Giorgio Melloni , Alessandro Guida

References

data origin for mutations, copynumber and expression data

data origin for fusion data

See Also

`getAlterations` `subsetAlterations`

Examples

```
# Retrieve example data
data(cpObj)
# Check format of slot dataFull
str(cpData(cpObj))
# Create two new mutations
newmutations <- data.frame(
    entrez_gene_id=c(7157 , 7157)
    ,gene_symbol=c("TP53" , "TP53")
    ,case_id=c("new_samp1" , "new_samp2")
    ,mutation_type=c("Nonsense_Mutation" , "Nonsense_Mutation")
    ,amino_acid_change=c("R306*" , "Y126*")
    ,genetic_profile_id=c("mynewbreaststudy" , "mynewbreaststudy")
    ,tumor_type=c("brca" , "brca")
    ,amino_position=c(306 , 126)
    ,genomic_position=c("17:7577022:G,A" , "17:7578552:G,C")
    ,stringsAsFactors=FALSE
    )
newsamples <- c("new_samp1" , "new_samp2")
# A dataFull slot style list should look like this
toBeAdded <- list(fusions=list(data=NULL , Samples=NULL)
                 , mutations=list(data=newmutations
                     , Samples=list(brca=newsamples))
                 , copynumber=list(data=NULL , Samples=NULL)
                 , expression=list(data=NULL , Samples=NULL)
                 )
cpObjplus <- appendRepo(cpObj , toBeAdded)
```

CancerPanel-class *Class* CancerPanel

Description

Class CancerPanel defines an object (S4 class) that contains data and custom arguments to run panel simulations and panel designs.

Value

Class object

Objects from the Class

Objects can be created by calls of the form newCancerPanel(panel,padding_length ,utr).

Constructor

newCancerPanel(panel=data.frame, padding_length=numeric , utr=logical)

Slots

arguments Object of class list containing the all panel related information. It is filled during the CancerPanel object construction.

`dataFull` Object of class `list` containing all the alterations for all the genes in the panel and every tumor type requested. It is filled by the `getAlterations` method.

`dataSubset` Object of class `list` containing all the alteration requested. It is filled by the `subsetAlterations` method. Every alteration type is defined by a slot; all slots have the same data.frame layout.

Methods

show `show(object = "CancerPanel")`: ...

getAlterations `getAlterations(object = "CancerPanel")`: ...

subsetAlterations `subsetAlterations(object = "CancerPanel")`: ...

panelDesigner `panelDesigner(object = "CancerPanel")`: ...

cpArguments `cpArguments(object = "CancerPanel")`: ...

cpData `cpData(object = "CancerPanel")`: ...

cpDataSubset `cpDataSubset(object = "CancerPanel")`: ...

coveragePlot `coveragePlot(object = "CancerPanel")`: ...

coverageStackPlot `coverageStackPlot(object = "CancerPanel")`: ...

saturationPlot `saturationPlot(object = "CancerPanel")`: ...

coocMutexPlot `coocMutexPlot(object = "CancerPanel")`: ...

survPowerSampleSize `survPowerSampleSize(object = "CancerPanel")`: ...

survPowerSampleSize1Arm `survPowerSampleSize(object = "CancerPanel")`: ...

cpFreq `cpFreq(object = "CancerPanel")`: ...

panelOptimizer `panelOptimizer(object = "CancerPanel")`: ...

appendRepo `appendRepo(object = "CancerPanel")`: ...

dataExtractor `dataExtractor(object = "CancerPanel")`: ...

propPowerSampleSize `propPowerSampleSize(object = "CancerPanel")`: ...

filterMutations `filterMutations(object = "CancerPanel")`: ...

filterFusions `filterFusions(object = "CancerPanel")`: ...

See Also

`newCancerPanel`

Examples

```
showClass("CancerPanel")
```

coocMutexPlot *Plot cooccurence and mutual exclusivity between pairs of drugs or genes*

Description

This plot reports how two features, like drugs or genes, can be considered close or distant in terms of occurrence on the same set of samples. In case of genes, for example, it consider the number of times a gene is altered together with another one or in a mutual exclusive fashion. In case of drugs, all the drug target are pulled together and you can appreciate if they act on the same of different targets.

Usage

```
coocMutexPlot(object
 , var=c("drug","group","gene_symbol")
 , alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
 , grouping=c(NA , "drug" , "group" , "alteration_id" , "tumor_type")
 , tumor_type=NULL
 , collapseMutationByGene=FALSE
 , collapseByGene=FALSE
 , tumor.weights=NULL
 , style=c("cooc" , "dendro")
 , prob = c("hyper","firth")
 , drop=FALSE
 , noPlot=FALSE
 , pvalthr=0.05
 , plotrandom=TRUE
 , ncolPlot=FALSE
 , ...)
```

Arguments

`object` a CancerPanel object

`var` One of the following: "drug", "group", "gene_symbol". This parameter will set which variable is used in the plot

`alterationType` what kind of alteration to include. It can be one or more between "copynumber", "expression", "mutations", "fusions". Default is to include all kind of alterations.

`grouping` One of the following: "drug", "group", "alteration_id" , "tumor_type". This parameter draws a plot for every level of the chosen grouping. if set to NA, the panel is not split and the plot is one.

`tumor_type` A character vector of tumor types to include in the plot among the one included in the object

`collapseMutationByGene` A logical that collapse all mutations on the same gene for a single patient as a single alteration.

`collapseByGene` A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.

`tumor.weights` A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details

`style` If 'cooc', the default, it performs pairwise cooccurence and mutual exclusivity test and the plot is a pvalue upper triangle heatmap. If dendro, it performs hierarchical clustering using binary distance between 'var' subjects.

`prob` One of the following: "hyper" or "firth". Two ways of calculating cooccurrence mutex pvalues. The first uses the hypergeometric distribution as in the Fisher test. The second uses a penalized logistic regression and is particularly indicated when the alterations are rare.

`drop` Logical indicating if the table of cooccurence should include (FALSE) or not include (TRUE) the samples that are never altered for any element of the var of interested. Default FALSE. See details.

`noPlot` if TRUE, the plot is not shown and data to create it are reported.

`pvalthr` In the plot, every square under the threshold is depicted in gray

`plotrandom` If TRUE, all elements of var are reported, even if they have no significant pair

`ncolPlot` Number of columns required for a multiplot. With default = FALSE, the function calculates the optimal configuration based on the number of plots that need to be printed

`...` Further arguments passed to `hclust` function

Details

This plot explores if there is cooccurrence or mutual exclusivity between features selected in the panel. It is particularly useful to evaluate the opportunity to add a new gene or a new drug in an umbrella design. Two drugs that acts on mutual exclusive pathways are more suitable for an umbrella design that seek at enlarging the spectrum of covered samples, even though one of the two drugs has few affected samples. On the other hand, if a drug has been proven to be more effective or reliable and its target are alterated together with another drug, there is no point in adding the less effective cure. Another way of seeing this feature is by using clustering adding option 'dendro' to style parameter. The reported plot will lack of pvalues but it is more general including distances between hierarchically aggregated drugs or genes.

If noPlot is TRUE, the method returns a data.frame with 6 columns in case of style 'cooc':

sp1_name the first 'var' value for the mut-ex analysis

sp2_name the second 'var' value for the mut-ex analysis

pVal.MutEx pvalue associated with mutual exclusivity evaluation

pVal.Cooc pvalue associated with cooccurence evaluation

OR corrected odds ratio, of the confusion matrix between sp1 and sp2

grouping grouping variable chosen by the user

If noPlot is TRUE, the method returns a list of hclust objects in case of style 'dendro'

The option drop can completely change the results, so check exactly what is your initial question. Let's imagine a coocMutex plot by gene. If drop=FALSE, all the samples tested for mutations on those genes are included. Otherwise, only the samples with at least one mutation in the genes of interested will be included. The default is to keep all the samples but this procedure is biased towards cooccurrence. This is caused by the fact that mutations are rare, so the cooccurrence of no mutations is generally very high and it counts as no-mutations.

By default, coocMutexPlot will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your trial can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target. Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then aggregate the results
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

Value

In case of style 'cooc', an upper triangle discrete heatmap if noPlot is FALSE, a data.frame otherwise. In case of style 'dendro', a dendrogram if noPlot is FALSE, a list of patient by alteration matrices.

Author(s)

Giorgio Melloni, Alessandro Guida

References

code re-written from package cooccur implementation of penalized glm, here logistic regression

See Also

saturationPlot

Examples

```
# Load example CancerPanel object
data(cpObj)
# Plot cooccurence and mutual exclusivity between pairs of genes by tumor type
coocMutexPlot(cpObj
              , var="gene_symbol"
              , grouping="tumor_type"
              , alterationType="mutations")
```

---

coveragePlot *A series of bar charts representing the number of samples harbouring at least one or more alterations*

Description

Given a CancerPanel object, it returns one or more bar charts representing the number of samples covered by at least 1, 2 ,3 or more alterations using specified data. Each plot is controlled by the grouping parameter.

Usage

```
coveragePlot(object
, alterationType = c("copynumber", "expression", "mutations", "fusions")
, grouping = c(NA,"drug","group","gene_symbol","alteration_id","tumor_type")
, tumor_type=NULL
, alterationType.agg=TRUE
, collapseMutationByGene = TRUE
, collapseByGene = FALSE
, tumor.weights=NULL
, tumor.freqs=NULL
, maxNumAlt = 10
, colNum=NULL
, cex.main="auto"
, noPlot = FALSE)
```

Arguments

`object`	A CancerPanel object filled with genomic data.
`alterationType`	A character vector containing one or more of the following: "copynumber", "expression", "mutations", "fusions".
`grouping`	A character vector containing one or more of the following: NA, "drug", "group", "gene_symbol", "alteration_id", "tumor_type".
`tumor_type`	A character vector of tumor types to include in the plot among the one included in the object
`alterationType.agg`	logical value. If TRUE, the default, the frequencies displayed are calculated over all the samples that were tested for all the alterationType requested. If FALSE all the samples tested for the specified alteration_id stratum are used. It sorts an effect if 'alterationType' length is > 1 and 'alteration_id' is in grouping parameter. See details.
`collapseMutationByGene`	A logical that collapse all mutations on the same gene for a single patient as a single alteration.
`collapseByGene`	A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.
`tumor.weights`	A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details
`tumor.freqs`	A named vector of values between 0 and 1 which sum 1. It contains the expected proportion of patients that are planned to be recruited. See Details
`maxNumAlt`	This number represents the maximum number on X axis.
`colNum`	If set, represents the number of columns in plotting layout. If NULL, best square representation is chosen instead.
`cex.main`	a numerical value or "auto". This parameter can set the size of each plot main title. Default is "auto", for automatic resizing.
`noPlot`	If TRUE, the plot is not shown but just the data used to drawn it.

Details

According to the chosen alterationType, the package will look for all the samples with available data for all the selected alterationType. For example, if alterationType = c("mutations" , "copynumber"), only common samples with both mutation and copynumber data are used by default. If alterationType.agg is set to FALSE and "alteration_id" is in grouping, the default behaviour changes. "mutations" plot will be displayed with the frequencies relative to all the samples tested for mutations and "copynumber" with all the samples tested for CNA. If "tumor_type" is in grouping variable, each plot is evaluated on the samples relative to the tumor type. The number of plots depends on the multiplication of the levels of the grouping variable. If you put too many grouping variable, it is better to draw a coverageStackPlot or to redirect the output to a file.

By default, coveragePlot will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your trial can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then aggregate the results
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using tumor.freqs parameter. In this case the fraction of altered samples are first calculated tumor-wise and then reaggregated using the weights provided by tumor.freqs. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing. In the examples, brca and luad data are used. Breast samples are at least twice as much as luad samples and tumor.freqs can help with a more balanced simulation.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. To have a quick idea of the sample size required, it is better to use the former. For having an idea about the possible distribution of sample size giving a few samples (for example a minimum and a maximum sample size) it is better to run the function with tumor.weights several times and aggregate the results to obtain mean values, confidence intervals etc.

Value

If noPlot=FALSE, this method returns a bar chart or a series of bar charts. Y-axis represents the number of samples, X-axis the incremental number of alterations per sample. In case tumor.freqs is set, the Y-axis represents the relative frequency that is reported as text on the top of the bars. If noPlot=TRUE, it returns a named list:

plottedTable
: a matrix with absolute number of samples plotted. Every column represents how many samples retain at least 1, 2, 3 ... alterations. Every row is a different plot for one of the specified grouping levels. If tumor.freqs is used, relative frequencies are reported instead.

Samples
: a numeric vector corresponding to the rows of plottedTable representing the number of reference sample for each plot. If tumor.freqs is used, Samples is NULL.

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

saturationPlot coverageStackPlot

Examples

```
# Load example CancerPanel object
data(cpObj)
# Plot the coverage of this panel by tumor type and drug
# Using mutations and copynumber data
coveragePlot(cpObj , alterationType=c("mutations" , "copynumber")
          , grouping=c("tumor_type" , "drug")
          , maxNumAlt=5)
```

---

coverageStackPlot — *A stacked and beside bar chart representing a breakdown of covered samples*

Description

Given a CancerPanel object, it returns one bar chart representing the number of samples covered by at least 1 alteration under 'var' divided by 'grouping'

Usage

```
coverageStackPlot(object
, alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
, var=c("drug","group","gene_symbol","alteration_id","tumor_type")
, grouping=c(NA,"drug","group","gene_symbol","alteration_id","tumor_type")
, tumor_type=NULL
, collapseMutationByGene=TRUE
, collapseByGene=TRUE
, tumor.weights=NULL
, tumor.freqs=NULL
, plotFreq = FALSE
, noPlot=FALSE
, html=FALSE)
```

Arguments

`object`	A CancerPanel object filled with genomic data.
`alterationType`	A character vector containing one or more of the following: "copynumber", "expression", "mutations", "fusions".
`var`	a character vector of length 1 containing one or more of the following: "drug", "group", "gene_symbol", "alteration_id" , "tumor_type". This parameter is compulsory and decide the classes of the bars.

grouping	a character vector of length 1 containing one or more of the following: NA, "drug", "group", "gene_symbol", "alteration_id", "tumor_type". This parameter decide the breakdown of var. If not set, it is considered NA and only 'var' is plotted with no stacking.
tumor_type	a character vector containing tumor types to be plotted
collapseMutationByGene	A logical that collapse all mutations on the same gene for a single patient as a single alteration.
collapseByGene	A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copy-number, it will count for one alteration only.
tumor.weights	A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details
tumor.freqs	A named vector of values between 0 and 1 which sum 1. It contains the expected proportion of patients that are planned to be recruited. See Details
plotFreq	If TRUE, the plot return the relative frequencies instead of the absolute number of samples.
noPlot	If TRUE, the plot is not shown but just the data used to drawn it.
html	If TRUE, an html interactive version of the plot is reported using `googleVis`.

Details

This plot is a more compact (although less informative) version of the `coveragePlot`. According to the chosen `alterionType`, the package will look for all the samples with available data for all the selected `alterationType`. For example, if alteratonType = c("mutations" , "copynumber"), only common samples with both mutation and copynumber data are used. If both 'var' and 'grouping' are set, the plot will show two bars for every level of 'var'. The first one is a breakdown by 'grouping', while the second one is the total number of unique samples covered by at least one alteration. The first bar of the two is generally higher, because the breakdown does not sum up. For example, if we show a coverage stack plot of "drug" divided by "gene_symbol", the first bar will show the number of covered samples by every gene (considering a sample twice if is altered in more than one gene). The second bar is the total number of covered samples for the drug. The legend is not plotted if grouping is set to NA.

By default, `coverageStackPlot` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and trial can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then aggregate the results
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using `tumor.freqs` parameter. In this case the fraction of altered samples are first calculated tumor-wise and then reaggregated using the weights provided by `tumor.freqs`. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing. In the examples, brca and luad data are used. Breast samples are at least twice as much as luad samples and tumor.freqs can help with a more balanced simulation.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. To have a quick idea of the sample size required, it is better to use the former. For having an idea about the possible distribution of sample size giving a few samples (for example a minimum and a maximum sample size) it is better to run the function with tumor.weights several times and aggregate the results to obtain mean values, confidence intervals etc.

Value

If noPlot=FALSE, this method returns a bar chart. Y-axis represents the number of samples, X-axis the number of alterations per sample. In case tumor.freqs is set, the Y-axis represents the relative frequency that is reported as text on the top of the bars. If noPlot=TRUE, it returns a named list:

`plottedTable` a matrix with absolute number of samples plotted. Every column is a level of 'var' while every row represents one of the possible breakdown ('grouping').

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

`saturationPlot` `coveragePlot`

Examples

```
# Load example CancerPanel object
data(cpObj)
# Plot the number of covered samples
# Using mutations and copynumber data
coverageStackPlot(cpObj , alterationType=c("mutations" , "copynumber")
          , var="drug"
          , grouping="gene_symbol"
          , tumor_type="brca")
# Show an interactive version of the plot
# Save the html code first
myHtmlPlot <- coverageStackPlot(cpObj
          , alterationType=c("mutations" , "copynumber")
          , var="drug"
          , grouping="gene_symbol"
          , tumor_type="brca"
          , noPlot=FALSE
          , html=TRUE)
# Plot the code above
plot(myHtmlPlot)
```

`cpArguments` *Method that returns the* `arguments` *slot in a* `CancerPanel` *object.*

Description

This method returns a list with the panel general information after the creation of the object.

Usage

```
cpArguments(object)
```

Arguments

`object`	a CancerPanel object

Details

The length of the output list is always defined by 5 elements. For example, if there are no rs numbers present in the panel, the `data.frame` in dbSNP_rs element will be simply empty. The first 4 elements are filled at the very creation of the object with `newCancerPanel`. The `tumor_type` element is filled after data request with `getAlterations`.

Value

A list of 5 elements, one for each alteration type:

`genedata`	a data.frame with the cds and cds plus utr of the requested genes.
`dbSNP_rs`	a data.frame with all the rs number translated in genomic hg19 positions.
`panel`	a data.frame with the original panel information plus the length of each alteration.
`drugs`	a character vector containing unique drug names as reported in the panel.
`tumor_type`	a character vector with the tumor types requested in the panel.

Author(s)

Giorgio Melloni , Alessandro Guida

References

Source of gene length and exon structure.

See Also

`cpData` `cpDataSubset`

Examples

```
# Load example CancerPanel object
data(cpObj2)
# Show slot dataSubset
str(cpArguments(cpObj2) )
```

cpData — *Method that returns the `dataFull` slot in a `CancerPanel` object.*

Description

This method returns a list with an element for each alteration type contained in the class `CancerPanel` slot `dataFull`

Usage

```
cpData(object)
```

Arguments

`object`	a CancerPanel object

Details

The length of the output list is always defined by 4 elements even if no data, for a certain alteration type, were requested.In case no data were requested or if there are no data for a certain tumor type, the slot is filled with `NULL` values. Every element is a list of 2 elements:

- `data` A `data.frame` in a format specific for the alteration type.
- `Samples` The second element is a `list` of vectors containing the names of the samples for each tumor type.

Value

A list of 4 elements, one for each alteration type:

`mutations`	A list of 2 elements containing the mutation alterations for each gene requested in the panel.
`copynumber`	A list of 2 elements containing the copynumber alterations for each gene requested in the panel.
`expression`	A list of 2 elements containing the expression alterations for each gene requested in the panel.
`fusions`	A list of 2 elements containing the fusions alterations for each gene requested in the panel.

Author(s)

Giorgio Melloni , Alessandro Guida

References

data origin for mutations, copynumber and expression data data origin for fusion data

See Also

`cpDataSubset`

Examples

```
# Load example CancerPanel object
data(cpObj2)
# Show slot dataSubset
str( cpData(cpObj2) )
```

---

cpDataSubset — *Method that returns the `dataSubset` slot in a `CancerPanel` object.*

Description

This method returns a list with an element for each alteration type. It shows the content of the slot `dataSubset` in the `CancerPanel` object.

Usage

```
cpDataSubset(object)
```

Arguments

`object`	a CancerPanel object

Details

The slot showed with this getter is a list of length 4 even if no data for a certain alteration type were requested. In case no data were requested or if there are no data for a certain tumor type, the slot is filled with a `NULL` value. If data are present but there are no alteration for the specified tumor types the slot is filled with a 0 rows `data.frame`.

Value

A list of 4 elements, one for each alteration type:

`mutations`	a data.frame containing exactly the mutation alterations requested in the panel.
`copynumber`	a data.frame containing exactly the copynumber alterations requested in the panel.
`expression`	a data.frame containing exactly the expression alterations requested in the panel.
`fusions`	a data.frame containing exactly the fusions alterations requested in the panel.

Author(s)

Giorgio Melloni , Alessandro Guida

References

data origin for mutations, copynumber and expression data

data origin for fusion data

See Also

`cpData`

Examples

```
# Load example CancerPanel object
data(cpObj2)
# Show slot dataSubset
str( cpDataSubset(cpObj2) )
```

---

cpDesign *Example of a cancer panel design export*

Description

Bed style export of cpObj

Usage

```
data(cpDesign)
```

Format

A design is a list composed by four elements:

GeneIntervals: all CDS and CDS + UTR for all the genes in the panel

TargetIntervals: all requested target regions (specific single mutations) divided and collapsed by gene symbol

FullGenes: gene symbols of the genes considered for their full sequence

BedStylePanel: the entire panel in bed format, merged by chromosome, start and end.

Source

Derived from running panelDesigner on cpObj

Examples

```
#Load cpDesign and show its structure
data(cpDesign)
str(cpDesign)
```

---

cpFreq *Calculate frequencies of mutations, copynumber, fusion or expression on a CancerPanel object*

Description

Given a CancerPanel object, it returns a data.frame with absolute or relative frequencies of alteration per gene

Usage

```
cpFreq(object
, alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
, mutations.specs=c(NA ,"mutation_type","amino_acid_change"
                    ,"amino_position","genomic_position")
, fusions.specs=c("bygene" , "byfusionpair")
, tumor_type=NULL
, tumor.weights=NULL
, tumor.freqs=NULL
, collapseMutationByGene=TRUE
, freq=c("relative" , "absolute")
)
```

Arguments

`object`	A CancerPanel object filled with genomic data.
`alterationType`	A character vector containing one of the following: "copynumber", "expression", "mutations", "fusions".
`mutations.specs`	If alterationType is mutations, this parameters allows further subsetting of frequency of mutation. See details
`fusions.specs`	If "bygene", frequencies of fusion events are calculated by gene, if "byfusionpair" , the calculation are run by the pair gene1__gene2
`tumor_type`	A character vector of tumor types to include in the plot among the one included in the object
`tumor.weights`	A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details
`tumor.freqs`	A named vector of values between 0 and 1 which sum 1. It contains the expected proportion of patients that are planned to be recruited. See Details
`collapseMutationByGene`	A logical that collapse all mutations on the same gene for a single patient as a single alteration.
`freq`	return the absolute number of samples with an alteration or the relative number according to the selected cohort

Details

This simple frequency calculator allows an exploratory analysis on the frequency of alteration by gene. Using `mutations.specs`, the user can also calculate the mutation frequency by:

1. mutation_type: each gene is divided by missense , frameshift, splice site etc.
2. amino_acid_change: each gene is divided by amino acid change (V600E , R258H, etc.)
3. amino_position: each gene is stratified by aminoacidic position (600, 258, etc.)
4. genomic_position: each gene is subdivided by exact mutation at genomic level (1:10000:A,C etc.)

Both `mutations.specs` and `fusions.specs` sort no effect for copynumber and expression. The table reported for copynumber and expression is different from mutations and fusions. It reports, for every gene, the relative or absolute number of patients with "amplification" , "deletion" and "normal" CNA and "up" , "down" "normal" level of expression.

By default, `cpFreq` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your design can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then bootstrap them
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using `tumor.freqs` parameter. In this case the fraction of altered samples are first calculated tumor-wise and then reaggregated using the weights provided by `tumor.freqs`. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing. Note that the result can only be expressed as relative frequency.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. For having a quick idea of the sample size required, it is better to use the former. For having an idea about the possible distribution of sample size giving a finite number of samples it is better to run the function with tumor.weights several times and aggregate the results. By default, `survPowerSampleSize` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your design can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples , the real frequency can be distorted. to avoid this, it is better to run several small samples and then bootstrap them
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using `tumor.freqs` parameter. In this case the fraction of altered samples are first calculated tumor-wise and then reaggregated using the weights provided by `tumor.freqs`. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. For having a quick idea of the sample size required, it is better to use the former. For having an idea about the possible distribution of sample size giving a few samples (for example a minimum and a maximum sample size) it is better to run the function with tumor.weights several times and aggregate the results.

Value

the function returns a object of class data.frame

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

coveragePlot coverageStackPlot

Examples

```
# Load example CancerPanel object
data(cpObj)

# Calculate relative frequencies of mutations by gene in breast cancer
cpFreq(cpObj , alterationType="mutations", tumor_type="brca")

# Calculate relative frequencies of mutations by gene and amino_acid_change
# in both breast and lung cancer
cpFreq(cpObj , alterationType="mutations"
  , tumor_type=NULL
  , mutations.specs="amino_acid_change")

# Calculate the absolute number of samples with amplified
# deleted or normal gene
# Using lung cancer available data
cpFreq(cpObj , alterationType="copynumber"
, tumor_type="luad" , freq="absolute")

# Calculate frequencies of fusion pairs in all tumor types
cpFreq(cpObj , alterationType="fusions" , fusions.specs="byfusionpair")

# Now calculate mutation freq by gene using
# 90% of luad and 10% of brca samples
cpFreq(cpObj , alterationType="mutations", tumor.freqs=c(brca=0.9 , luad=0.1))
```

---

cpObj *Example of a cancer panel*

Description

A CancerPanel object used in PrecisionTrailDrawer

Usage

```
data(cpObj)
```

Format

A CancerPanel object with 3 slots and data from every cBioportal study on lung adenocarcinoma (LUAD) and breast carcinoma (BRCA):

`arguments`: Object of class `"list"` containing panel contents information. it id filled at the panel construction.

`dataFull`: Object of class `"list"` containing a slot for every alteration type. it is filled with getAlterations method and contains all the alterations for all the genes in the panel and every tumor type requested.

`dataSubset`: Object of class `"list"` containing a slot for every alteration type. it is filled with subsetAlterations method and reports all the alterations for every requested specific alteration. All slots have the same data.frame layout.

Source

data origin for BRCA and LUAD

Examples

```
#Load cpObj and show its structure
data(cpObj)
str(cpObj)
```

`cpObj2` *Example of a cancer panel*

Description

A CancerPanel object used in PrecisionTrailDrawer

Usage

```
data(cpObj2)
```

Format

A CancerPanel object with 3 slots and data from every cBioportal study on lung adenocarcinoma (LUAD) and breast carcinoma (BRCA):

`arguments`: Object of class `"list"` containing panel contents information. it id filled at the panel construction.

`dataFull`: Object of class `"list"` containing a slot for every alteration type. it is filled with getAlterations method and contains all the alterations for all the genes in the panel and every tumor type requested.

`dataSubset`: Object of class `"list"` containing a slot for every alteration type. it is filled with subsetAlterations method and reports all the alterations for every requested specific alteration. All slots have the same data.frame layout.

Source

data origin for BRCA and LUAD

Examples

```
#Load cpObj2 and show its structure
data(cpObj2)
str(cpObj2)
```

---

dataExtractor *dataExtractor: extract data from a CancerPanel object*

Description

Given user specified options, return specific data from a CancerPanel object, including alteration and Samples

Usage

```
dataExtractor(object
    , alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
    , tumor_type=NULL
    , collapseMutationByGene=TRUE
    , collapseByGene=FALSE
    , tumor.weights=NULL)
```

Arguments

object	An instance of class CancerPanel
alterationType	what kind of alteration to include. It can be one or more between "copynumber", "expression", "mutations", "fusions". Default is to include all kind of alterations.
tumor_type	only plot one or more tumor types among the ones available in the object.
collapseMutationByGene	A logical that collapse all mutations on the same gene for a single patient as a single alteration.
collapseByGene	A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.
tumor.weights	A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details

Details

This function is used internally by most of the methods of the package and provide a common data extractor for a CancerPanel object. It is a low level function to retrieve data for other custom usages, in particular via tumor.weights.

Value

A named list with data, samples and tumors not present in the CancerPanel object is returned.

Author(s)

Giorgio Melloni , Alessandro Guida

See Also

getAlterations subsetAlterations

Examples

```
# Retrieve example data
data(cpObj)
# Extract CNA and mutation data
mydata <- dataExtractor(cpObj
, alterationType=c("copynumber" , "mutations")
, tumor_type="brca")
# It is particularly useful for bootstrap simulations
# Here we extract 10 random samples composed by 30 brca and 40 luad
myboot <- replicate(10
, dataExtractor(cpObj
, alterationType="mutations"
, tumor.weights=c("brca"=30 , "luad"=40)
))
```

---

filterFusions — *filterFusions: remove or keep specified fusions*

Description

This method allows to keep only or to exclude certain fusions according to three different filtering formats

Usage

```
filterFusions(object , filtered , mode = c("exclude" , "keep"))
```

Arguments

object	An instance of class CancerPanel
filtered	A character vector used as filter
mode	If "exclude", fusions are removed from the object. If "keep", fusions specified in filtered are the only ones maintained.

Details

filtered vector must be in fusion format GENE1__GENE2.

At the end of the filtering procedure, subsetAlterations is automatically run.

Value

An updated instance of class CancerPanel

Author(s)

Giorgio Melloni , Alessandro Guida

References

data origin for mutations , copynumber and expression data

data origin for fusion data

See Also

getAlterations subsetAlterations filterMutations

Examples

```
# Retrieve example data
data(cpObj)
# Create a data.frame to filter
myFilter <- c("ERC1__RET", "TRIM33__RET", "EML4__ALK")
# Keep only myFilter fusions
cpObjKeep <- filterFusions(cpObj , filtered = myFilter , mode = "keep")
# Exclude myFilter fusions
cpObjExclude <- filterFusions(cpObj , filtered = myFilter , mode = "exclude")
```

---

filterMutations *filterMutations: remove or keep specified mutations*

Description

This method allows to keep only or to exclude certain mutations according to three different filtering formats

Usage

```
filterMutations(object
, filtered=NULL
, bed = NULL
, mode = c("exclude" , "keep")
, tumor_type=NULL)
```

Arguments

object	An instance of class CancerPanel
filtered	A data.frame used as filter
bed	A data.frame in bed format (chr start end)
mode	If "exclude", mutations are removed from the object. If "keep", mutations specified in filtered are the only ones maintained.
tumor_type	A vector of tumor_type names. The filter will be active only on the specified tumor types

Details

filtered data.frame can come in three different formats:

1. "gene_symbol" , "amino_position" Ex. BRAF 600
2. "gene_symbol" , "amino_acid_change" Ex. BRAF V600E
3. "genomic_poistion" Ex. 3:1234567:A,C

bed file must be composed by 3 columns: chrN, start 0-base , end 1-base

At the end of the filtering procedure, subsetAlterations is automatically run.

Value

An updated instance of class CancerPanel

Author(s)

Giorgio Melloni , Alessandro Guida

References

data origin for mutations , copynumber and expression data

data origin for fusion data

See Also

`getAlterations` `subsetAlterations` `filterFusions`

Examples

```
# Retrieve example data
data(cpObj)
# Create a data.frame to filter
myFilter <- data.frame(gene_symbol = c("BRAF" , "PIK3CA")
    , amino_position = c(600 , 118))
# Keep only myFilter mutations
cpObjKeep <- filterMutations(cpObj , filtered = myFilter , mode = "keep")
# Exclude myFilter mutations
cpObjExclude <- filterMutations(cpObj , filtered = myFilter , mode = "exclude")

# Create a bed file
myBed <- data.frame(chr = paste0("chr" , c(7 , 17))
    , start = c(140534632 , 41244326)
    , end = c(140534732 , 41244426) , stringsAsFactors=FALSE)
# Keep only myFilter mutations
cpObjKeep <- filterMutations(cpObj , bed = myBed , mode = "keep")
# Exclude myBed mutations
cpObjExclude <- filterMutations(cpObj , bed = myBed , mode = "exclude")
```

`getAlterations` *Retrieve genomic data for each gene in the panel*

Description

This method updates the CancerPanel object with data from cBioportal and MD Anderson fusion database

Usage

```
getAlterations(object, tumor_type = NULL, repos = NULL
, mutation_type = c("all_nonsynonymous" , "all_mutations"
    , "missense" , "truncating")
, expr_z_score = 2
, BPPARAM = bpparam("SerialParam")
, gene_block=50)
```

Arguments

`object`	A CancerPanel object
`tumor_type`	a vector of tumor types from the ones available using `showTumorType` or in the first column of `showCancerStudy`. See details
`repos`	a list containing custom data to be used in the object
`mutation_type`	decide the kind of mutations to retrieve. Only non synonymous, only missense, only truncating or all the mutations. The default is to retrieve only non synonymous
`expr_z_score`	a number that expresses the threshold at which a gene is considered upregulated or downregulated
`BPPARAM`	parameter for `bplapply` to parallelize part of the code
`gene_block`	Set how many genes at a time are requested to cBioPortal. Default 50

Details

This method fills the slot dataFull in a cancer panel object. It retrieves data by gene from cBioportal and MD Anderson fusion database according to the specifications of the panel. This slot is composed by a list of 4 elements, one for each alterationType: 'fusions' , 'mutations' , 'copynumber' , 'expression'. Every element is a list of 2 elements: data , Samples. The first element is a data.frame in a format specific for the alteration type. The second element is a list of vectors containing the names of all the samples analyzed for each tumor type (both altered and "wild-type").

`tumor_type` parameter can be either a list of tumor types ('brca' , 'luad') or specific cancer studies ('brca_tcga_pub2015' , 'luad_tcga_pub') but not the two things together. Check availability with the functions mentioned above. In case of cancer studies selection, fusions are retrieved from cancer type definition ('brca_tcga_pub2015' becomes simply 'brca') since they come from a different source.

The expression value are expressed as up or down according to the threshold in 'expr_z_score'. A gene is considered upregulated if its z-score is over 2 or downregulated if is lower than -2.

The copynumber values are expressed as 'amp' or 'del' according to GISTIC definition. A gene is reported as amplified or deleted if its value after GISTIC evaluation is 2 or -2.

A message about the current tumor in download is prompted for every study. If the message appears more than once, it probably means that the genes you requested were more than 100 and so the query was actually split in two or more chunks to avoid an overload on cbioportal database.

Value

The method returns the original CancerPanel object with the slot dataFull updated.

Author(s)

Giorgio Melloni , Alessandro Guida

References

data origin for mutations, copynumber and expression data

data origin for fusion data

See Also

`cgdsr-getCancerStudies` `subsetAlterations` `showTumorType` `showCancerStudy`

Examples

```
#Load panelexample
data(cpObj2)
# Retrieve data from AML
cpObj <- getAlterations(cpObj2 , tumor_type=c("laml"))
```

`newCancerPanel` *CancerPanel object constructor*

Description

Given a `data.frame` with your panel specifications, it creates a `CancerPanel` object to be used for both simulations and panel design.

Usage

```
newCancerPanel(panel
              , rules=NULL
              , padding_length = 100
              , utr = FALSE
              , canonicalTranscript=TRUE
              , myhost="www.ensembl.org"
              )
```

Arguments

- `panel` — A data.frame describing your panel with alterations, associated drugs and relative genes involved.
- `padding_length` — An integer that defines how much to extend the targeted regions. `paddind_lenght` value is subtracted from the start coordinate of the targeted region in the panel and, at the same time, it is also added to the end coordinate of the panel. The result is an extension of the target region by the `padding_length` value in both 5' and 3' direction.
- `utr` — If `TRUE`, the genomic coordinates will also include the UTR regions.
- `canonicalTranscript` — if FALSE, every exon of every transcript of the gene is taken into consideration in calculating gene length. Default to TRUE is to select the canonical transcript (see references)
- `myhost` — In case of a biomart breakdown, choose a different host than the default ensembl.org. check availability on biomart mirrors
- `rules` — a data.frame similar to the panel that implement a set of rules so that specific associations between genes/mutations/actionability are overwritten. See Details

Details

This constructor accepts a `data.frame`, `tibble` or `data.table` with the following columns:

`drug` **- character vector** A character vector of drug names or drug compounds. It is required, but it can be also filled with `NA` if no compound is associated with the alterations or the user is not interested in this feature.

`gene_symbol` **- character vector** A character vector of HGNC gene symbols. In case of specific fusion gene, the format is 'gene1__gene2'.

`alteration` **- character vector** A character vector with one of the following values: `SNV` (Single Nucleotide Variation) or `CNA` (Somatic Copy Number Alteration), `expression` (up or down gene expression), `fusion` (hybrid gene formed from two previously separated genes). This represents the class of alteration.

`exact_alteration` **- character vector** According to the alteration column, it can be one of the cases described in the table below.

`mutation_specification` **- character vector** This column refines the location of the alteration type defined in the previous column. In case the record in `alteration` is set to 'SNV', the location of the mutation must be specified according to the available options shown in the table below. In case of an alteration type different from 'SNV' the value must be left blank ("") or NA.

`group` **- character vector (Optional)** A character vector describing a specific group for the alteration. In the `panelexample`, we use it to divide the alterations between druggable (Actionable) and non druggable (Driver). Another possible use is to perform comparisons between different panels.

Possible values for a cancer panel to specify the alterations are:

alteration	**exact_alteration**	**mutation_specification**
CNA	amplification	""
CNA	deletion	""
expression	up	""
expression	down	""

SNV	""	""
SNV	mutation_type	missense
SNV	mutation_type	truncating
SNV	amino_acid_position	300-300
SNV	amino_acid_position	300-350
SNV	amino_acid_variant	V600E
SNV	genomic_position	13:20000-40000
SNV	genomic_position	13:20000-20000
SNV	genomic_variant	13:20000:A,C
SNV	dbSNP_rs	rs1234567
fusion	""	""

`drug` values NA or empty are transformed to "no_drug", that is a reserved value. `group` values NA or empty are transformed to "no_group", that is a reserved value. `gene_symbol` is mandatory for any alteration type because alterations are retrieved from cBioPortal using this key. In case of rs IDs, the closest gene symbol is the ideal annotation, even if the position is intergenic and generally not associated with any gene.

`rules` parameter implements a set of negation rules.

A data.frame like the following can be used in the rules parameter. The data.frame is the same as above but it adds tumor_type and in_out columns:

drug	**gene_symbol**	**alteration**	**exact_alteration**	**mutation_specification**	**group**	**tumor_type**	**in_ou**
Erlotinib	EGFR	SNV	amino_acid_variant	T790M	Driver		exclu
Erlotinib	KRAS	SNV			Driver		exclu
Erlotinib					Driver	luad	inclu
Olaparib					Driver	brca	inclu

The new column `tumor_type` can contain a single tumor_type code or an empty string (which means that the rule is valid for any tumor type). The other new column is `in_out` which can only contain 'include' or 'exclude' values.

The first two rows implement a resistance rule. Any sample with a T790M mutaion on EGFR or a KRAS mutation cannot be associated with Erlotinib because it generates resistance to EGFR inhibitors. The effect is that every sample with either a EGFR T790M mutation or any KRAS mutation will no longer be associated with Erlotinib (it will be considered 'no_drug') but the mutation will not be filtered out. In the group slot, the user can put what group the patient with that drug association will be changed into (in the example from Actionable to Driver). We set the rule as exclusion (in_out=exclude) for any tumor type.

The other two rows are more stringent and contain a drug inclusion/exclusion rule. Erlotinib can only be associated with luad and Olaparib only to brca (in_out=include). No matter what tumor type will be used in the future, these rules will always be applied so that any tumor type that is not luad or brca will be excluded.

Value

This method returns a `CancerPanel` object with the slot arguments updated.

Author(s)

Giorgio Melloni , Alessandro Guida

References

source of gene length and exon structure

source of official gene symbols and mapping with ensembl

canonical transcript definition according to ENSEMBL

biomart mirrors

See Also

`panelDesigner`

Examples

```
# Load the panel example
data(panelexample)
# Create a CancerPanel object for the first 3 lines
mypanel <- newCancerPanel(panelexample[1:3 , ] , canonicalTranscript=FALSE)
```

---

panelDesigner *Estimate panel design according to user specifications*

Description

This function takes a CancerPanel object and returns the regions ready to be submitted for sequencing

Usage

```
panelDesigner(object
    , alterationType = c("copynumber", "expression", "mutations", "fusions")
    , padding_length = 100
    , merge_window = 50
    , utr = FALSE
    , canonicalTranscript=TRUE
    , BPPARAM=bpparam("SerialParam")
    , myhost="www.ensembl.org")
```

Arguments

`object`	a CancerPanel Object
`alterationType`	by default, the design of the panel is created by mixing all the different types of alterations. With this parameter you can separate the design by alteration type.
`padding_length`	elongation on both side in case of single spot genomic request
`merge_window`	if two ranges are very close to each other what is the minimum length accepted for them to be separated and not merged?
`utr`	if TRUE, the genes ranges in the panel design are taken as CDS plus utr. Default is to take just the coding sequence
`canonicalTranscript`	if FALSE, every exon of every transcript of the gene is taken into consideration in calculating gene length. Default to TRUE is to select the canonical transcript

`BPPARAM`	an object of class BiocParallelParam to distribute REST API queries from Ensembl and HGNC. Serialization is the default.
`myhost`	In case of a biomart breakdown, choose a different host than the default ensembl.org. check availability on biomart mirrors

Details

In the majority of cases, copynumber and mutations data are retrieved using different technologies and the design should be separated. Use 'alterationType' parameter to create multiple libraries. In case of fusions, the design will take into account all the genes that form the fusion. The technology used to find fusion genes can rely on RNA rather than DNA, so in this case it is better to avoid this function. A similar idea can be applied for expression data. 'merge_window' parameter is generally calculated by the sequencing company, so set it to 0 if you don't want to decide it upfront. The ability of the machine to capture a region and the cost associated with a change in this measure depends on the technology itself. It can be very difficult to find the proper trade-off between library size and number of ranges. The larger is the intronic region accepted, the larger is the library size because you will accept a lot of off-targets. On the other end, the more regions in your library, the higher will be the number of amplicons used.

Value

A list of 4 elements:

`GeneIntervals`	a data.frame containing all gene wise intervals on cds and cds plus utr
`TargetIntervals`	if the panel contains specific regions, it is a data.frame of non-full gene sequences as requested in the panel
`FullGenes`	a character vector of genes that will be sequenced for their entire length
`BedStylePanel`	a bed style data.frame with chromosome start and end of the collapse of GeneIntervals and TargetIntervals

Author(s)

Giorgio Melloni, Alessandro Guida

References

bed file format according to ensembl

canonical transcript definition according to ENSEMBL

See Also

`newCancerPanel`

Examples

```
#Load a Cancer Panel Object
data(cpObj2)
# Design your panel for expression data only
# Parallelize part of the code using BiocParallel backend for Unix systems
if(tolower(Sys.info()["sysname"])=="windows"){
  mydesign <- panelDesigner(cpObj2
                          , alterationType="mutations")
```

```
} else {
  mydesign <- panelDesigner(cpObj2
                         , alterationType="mutations"
                         , BPPARAM=BiocParallel::MulticoreParam(workers=2)
                         )
}
# Retrieve bed style sequences
head( mydesign[['BedStylePanel']] )
```

---

panelexample   *Example of a cancer panel*

Description

A data frame containing a cancer panel data.frame to create a CancerPanel object

Usage

```
data(panelexample)
```

Format

A data.frame of 6 columns:

1. drug: character vector of drug names or any chemical identifiers
2. gene_symbol: character vector HGNC official gene symbol
3. alteration: a character vector of accepted alteration type, 'SNV', 'CNA', 'expression', 'fusion'
4. exact_alteration: a character vector that identifies the exact alteration depending on alteration value. For example, alteration 'CNA' could correspond to exact_alteration 'amplification' or 'deletion'
5. mutation_specification: a character vector that must be empty if alteration is not 'SNV' and identifies the exact alteration in various format
6. group: a character vector that is useful for custom annotation of each single row in the panel. In the example is used to identify actionable from driver variants

Examples

```
#Load panelexample and show its structure
data(panelexample)
str(panelexample)
```

`panelOptimizer` *A function that open a shiny app to optimize panel size while retaining the majority of mutations*

Description

Given a CancerPanel object, it automatically reads the genes and panel information and allows a custom subsetting of the panel to retain the largest amount of mutations while saving genomic space

Usage

```
panelOptimizer(object)
```

Arguments

`object`	A CancerPanel object filled with genomic data.

Details

This function reads the panel objects and retrieves all the genes requested for SNV and full sequence. It also collects all the mutation data and the tumor types available and performs an analysis on the position of the mutations on the canonical protein sequence for every gene.

At function call, a shiny app with four tabs is opened. Select a gene and the tumor types to use and click Run.

On the first and second tabs, the user is guided in the choice of most appropriate regions using an in-house Bioconductor package called `LowMACA`. `LowMACA` creates a null model where all the mutations on each gene are randomly permutated along the sequence. Every position that exceeds the threshold of 95% confidence interval is considered not random and represents an hotspot. The user can also decide to use a bandwidth in this calculation and apply a Gaussian density to the distribution of mutations along the sequence. Alterations that are closed to each other in the sequence will be aggregated to form significant regions. If the LowMACA analysis succeded, a table will appears under the plot in the first tab and on the second tab a list of significant positions is also shown. Click on Store LowMACA yellow button and all the regions identified by the algorithm will be stored in the fourth tab (Optimize Panel).

If you want to select your own custom regions, go to the third tab (Manual Selection). You can click on the red dots and retrieve information on specific mutations. If you drag a region with the mouse, a table will appear below, with the same information as the LowMACA analysis. Click on Store yellow button to keep the region selected and move it to the fourth tab.

For a new analysis on a different gene, just select the gene and click Run again.

When satisfied, just click on Close and save or simply close the browser page. All the regions selected will be merged and returned in standard output.

Value

A list of three elements after closing the shiny session.

1. regions a data.frame with the regions selected, the percentage of space occupied and percentage of mutations captured
2. mergedRegions merged protein regions by gene from user section
3. panel if no regions were selected, it reports the original panel, otherwise it substitutes the gene requested in full sequence with the regions in mergedRegions

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

entropy lmPlot

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  # Load example CancerPanel object
  data(cpObj)
  # Optimize the space on the shiny app.
  # All changes mad on the app will be saved
  newpanel <- panelOptimizer(cpObj)
  # If some changes have been made, recreate a new CancerPanel object
  if(!is.null(newpanel$regions)){
    cpObjOptimized <- newCancerPanel(newpanel$panel)
    # Fill the object with the same data of the non optimized panel
    cpObjOptimized <- getAlterations(cpObjOptimized , repos=cpData(cpObj))
    # Subset alterations on the new panel directives
    cpObjOptimized <- subsetAlterations(cpObjOptimized)
  }
}
```

---

propPowerSampleSize — *Calculate sample size or power required in a 2-sample or 1-sample proportion equality study*

Description

This plot method returns a scatter plot of required sample size at screening by statistical power divided by couples of case-control probabilities.

Usage

```
propPowerSampleSize(object
, var=c(NA , "drug" , "group" , "gene_symbol" , "alteration_id" , "tumor_type")
, alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
, tumor_type=NULL
, stratum=NULL
, tumor.weights=NULL
, tumor.freqs=NULL
, pCase=NULL
, pControl=NULL
, side = c(2,1)
, type=c("chisquare" , "arcsin" , "exact")
, alpha=0.05
, power=NULL
, sample.size=NULL
, case.fraction=0.5
```

```
, collapseMutationByGene=TRUE
, collapseByGene=FALSE
, round.result=TRUE
, priority.trial=NULL
, priority.trial.order=c("optimal" , "as.is")
, priority.trial.verbose=TRUE
, noPlot=FALSE)
```

Arguments

`object` a CancerPanel object

`var` one among NA , "drug" , "group" , "gene_symbol" , "alteration_id" or "tumor_type". It defines the arms of the studies to be projected. With var=NA, the projection of the entire panel is displayed.

`alterationType` what kind of alteration to include. It can be one or more between "copynumber", "expression", "mutations", "fusions". Default is to include all kind of alterations.

`tumor_type` only plot one or more tumor types among the ones available in the object.

`stratum` a character vector containing one or more specific elements of var to be plotted instead of all the arms of the study. If it is not present, a warning is raised and the full design is returned.

`tumor.weights` A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details

`tumor.freqs` A named vector of values between 0 and 1 which sum 1. It contains the expected proportion of patients that are planned to be screened. See Details

`pCase` a numerical vector of one or more postulated proportions for cases (between 0 and 1)

`pControl` a numerical vector of one or more postulated proportions for controls of the same length of pCase (between 0 and 1)

`side` perform a 2-tail or 1-tail calculation. Default 2

`type` calculate sample size using chisquare, arcsin or exact method. chisquare is used for a 2-sample equality while the other two are used in case in case pControl is fixed (1-sample case). case.fraction has no effect using arcsin or exact type.

`alpha` a numerical value between 0 and 1 that reports the type I error threshold. Default 0.05 (5%)

`power` a numerical vector of values between 0 and 1 that expresses the level of type II error. It is used to estimate sample size

`sample.size` a positive integer numerical vector that reports the postulated sample size at screening. It is used to estimate the power of the study.

`case.fraction` a numerical value between 0 and 1 representing the fraction of total sample size allocated to group 'Case'. Group control have an allocation of 1 - case.fraction

`collapseMutationByGene`
A logical that collapse all mutations on the same gene for a single patient as a single alteration.

`collapseByGene` A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.

`round.result` logical indicating if the sample size should be rounded with ceiling or not.

`priority.trial` A character vector of drugs or group levels to start the design of a priority trial. See Details.

`priority.trial.order`
Either "optimal" or "as.is". If "optimal" is used, the screening starts from the rarest drug or group level up to the most common to guarantee minimal sample size at screening. In case of "as.is", the order of priority.trial remains unchanged.

`priority.trial.verbose`
If TRUE, the result of a priority.trial will be a complete report in a 5-element list.

`noPlot` if TRUE, the plot is not shown and data are reported instead.

Details

This method estimates sample size or power on the basis of one of the two information. Using multiple sample sizes or power, power curves are reported simulating different scenarios. Power or sample size are required but not both at the same time. HR must be also set but if a vector is provided, the plot will show multiple curves according to the various hazard ratios. 'p.event', 'alpha' and 'case.fraction' are instead fixed for all the arms of the study (represented by the 'var' parameter).

If noPlot=TRUE, a data.frame with 6 column is reported instead:

Var levels of chosen variable

ScreeningSampleSize total sample size estimation at screening on the basis of frequency of alteration

EligibleSampleSize sample size estimated as sum of cases and controls after screening

Beta tested beta values

Power tested 1 - beta values

Proportion.In.Case.Control couples of pCase - pControl tested

The algorithm estimates sample size on the basis of no a priori probability of finding a case or control subject ("EligibleSampleSize" column). In a basket or umbrella design, this number must be multiplied by the frequency of alteration that we expect to find based on the simulation run on the panel. If our panel can cover the 50% of the samples with a target therapy and 100 samples are required to reach 80% power, we have to screen at least 200 patients in order to reach the desired number of cases in the sample size ("ScreeningSampleSize" column).

Similarly, if you want to estimate the power of the panel given an estimated sample size, we first multiply 'sample.size' by the frequency of expected alterations and then perform power estimation. 'sample.size' is therefore intended at screening.

When 'var' variable is set, the algorithm provides the estimated sample size for each stratum of the variable. For example, if we set it to 'drug', a power curve for each drug type is displayed, without taking into account possible overlaps. If a sample shows multiple targettable alterations, it will be reused for every drug type that targets those alterations.

By default, `propPowerSampleSize` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your design can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then bootstrap them
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using `tumor.freqs` parameter. In this case the fraction of altered samples are first calculated tumor-wise and then reaggregated using the weights provided by `tumor.freqs`. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. To have a quick idea of the sample size required, it is better to use the former. To get an idea about the possible distribution of sample size giving a few samples (for example a minimum and a maximum sample size) it is better to run the function with tumor.weights several times and aggregate the results.

If priority.trial is set, a cascade design is build up. Given a set of parameter (power, pCase, pControl , alpha, etc.) an Eligible Sample Size (ESS) is calculated that is the same across drugs/groups. The total Screening Sample Size (SSS) is calculated following this scheme:

1. Start screening with the first drug/group, reaching the sample size necessary to reach ESS
2. From the samples not eligible for the first drug/group, test the second drug/group and collects as many samples as possible up to ESS
3. Continue using the samples not eligible to the end of all drugs/levels. Stop if there are no leftovers.
4. If all the drugs/groups have reached ESS, stop. Otherwise start a new screening with the first drug/group that has not reached ESS
5. Repeat from point 2 up to completion

If priority.trial.order is set, the user can decide if the drugs/group levels must follow a precise order (as.is) or if the screening can start from the rarest drug/group level up to the most common (optimal). Following the optimal priority trial guarantees the best possible allocation with the minimum screening.

Value

If noPlot = FALSE (default) a scatter plot is returned. If noPlot = TRUE, a data.frame is returned. In case priority.trial is set, a list of length-5-lists is reported. See vignette for details.

Author(s)

Giorgio Melloni, Alessandro Guida

References

Chow S, Shao J, Wang H. 2008. Sample Size Calculations in Clinical Research. 2nd Ed. Chapman & Hall/CRC Biostatistics Series. page 85/89

See Also

`coveragePlot` `survPowerSampleSize`

Examples

```
# Load example CancerPanel object
data(cpObj)
# Show the study design by tumor type:
# 3 pCase - pControl couples and 4 power levels
# The full design is weighted using tumor.freqs
# The final sample size is composed by 90% luad and 10% brca
propPowerSampleSize(cpObj
  , var = "tumor_type"
  , pCase = c(0.7, 0.8 , 0.9)
  , pControl = rep(0.5 , 3)
  , power=c(0.5 , 0.6 , 0.7 , 0.8 , 0.9)
  , tumor.freqs = c(brca=0.1 , luad=0.9))
# Return power levels giving sample sizes at screening
propPowerSampleSize(cpObj
  , var = NA
  , pCase = c(0.7, 0.8 , 0.9)
  , pControl = rep(0.5 , 3)
  , sample.size=c(100 , 300 , 500 , 1000)
  , noPlot=FALSE)
```

`saturationPlot` *Plot your panel along with incremental genomic space occupied adding one piece at a time*

Description

This plot method returns a scatter plot with genomic space on X axis and average/absolute number of alterations on Y axis. The way the plot is built is incremental. We add one feature at a time starting from the most altered and we see how many samples we include at each step and how much space is occupied.

Usage

```
saturationPlot(object
    , alterationType = c( "copynumber", "expression", "mutations", "fusions")
    , grouping = c(NA, "drug", "group", "alteration_id", "tumor_type")
    , adding = c( "alteration_id", "gene_symbol", "drug", "group")
    , tumor_type = NULL
    , y_measure = c( "mean", "absolute")
    , adding.order=c( "absolute", "rate")
    , sum.all.feature=FALSE
    , collapseMutationByGene=TRUE
    , collapseByGene=FALSE
    , labelling=TRUE
    , tumor.weights=NULL
    , main=""
    , legend=c("in" , "out")
    , noPlot = FALSE)
```

Arguments

`object`	a CancerPanel object
`alterationType`	what kind of alteration to include. It can be one or more between "copynumber", "expression", "mutations", "fusions". Default is to include all kind of alterations.
`grouping`	One of the following: "drug", "group", "alteration_id", "tumor_type". This parameter draws a curve for every level of the chosen grouping. if set to NA, the panel is not split and the plot is a single curve.
`adding`	One of the following: "alteration_id", "gene_symbol", "drug", "group". This parameter will set which variable is added at every point of the plot. see details
`tumor_type`	only plot one or more tumor types among the ones available in the object.
`y_measure`	if 'mean', the measure on Y axis is the mean number of alterations per sample. Confidence interval of the measure is also reported. If 'absolute', the relative frequency of samples covered by at least one alteration is reported, similarly to `coveragePlot`.
`adding.order`	This parameter modifies the order of entrance of the adding variable. If 'absolute', the adding variable starts from the most altered up to the less frequently altered. If 'rate', the order of entrance, from left to right is based on the number of alterations divided by the length in kb.
`sum.all.feature`	logical. if TRUE every gene length of the panel is summed up by the adding variable. The effect is that if a gene is considered both for SNV and CNA, it is counted twice.
`collapseMutationByGene`	A logical that collapse all mutations on the same gene for a single patient as a single alteration.
`collapseByGene`	A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.
`labelling`	if FALSE, the dots are not labelled. It is useful for very large comparative plots. Default TRUE.
`tumor.weights`	A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details
`main`	Set a name for the plot
`legend`	if 'in' the legend is plotted in the top left corner, if 'out', outside of the plotting area
`noPlot`	if TRUE, the plot is not shown and data to create it are reported instead.

Details

This plot is particularly useful to evaluate the panel piece by piece. At the last point, we can observe the maximum coverage or maximum mean value of the panel, as reported in the coveragePlot. If we go back one point at a time, we can appreciate how many samples we gained by adding a new drug or a new gene to the panel. It is often the case that our panel is redundant for certain drugs or genes and there is no point in wasting sequencing space for a gene that is poorly altered and doesn't allow further improvement to our clinical trial.

By default, `saturationPlot` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your design can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then bootstrap them
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

If noPlot is TRUE, the method returns a data.frame with 8 or 9 columns, depending on how the adding.order parameter was set:

gene_symbol , drug , alteration_id or group the adding variable chosen by the user

grouping the grouping variable chosen by the user

Mean the value plotted on Y axis if 'mean' is chosen as y_measure parameter

Coverage the value plotted on Y axis if 'absolute' is chosen as y_measure parameter

SD standard deviation of Mean

SE standard error of Mean

CI confidence interval of Mean

Space genomic space in kBases ordered by grouping variable

num_of_variants_per_KB if adding.order='rate', this additional column is added. It represents the number of alteration divided by the feature length

Value

An incremental scatter plot if noPlot is FALSE, a data.frame otherwise.

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

coveragePlot

Examples

```
# Load example CancerPanel object
data(cpObj)
# Plot the saturation of this panel by tumor type adding one drug at a time
# Using mutations and copynumber data
saturationPlot(cpObj
    , alterationType=c( "mutations" , "copynumber")
    , adding="drug"
    , grouping="tumor_type"
    , y_measure="absolute")
# Plot with no grouping giving more weight to lung cancer samples
# Note that we ask for more samples than the availables in luad dataset
# the code will recycle the samples to account for this forced disequilibrium
```

```
saturationPlot(cpObj
    , alterationType=c( "mutations" , "copynumber")
    , adding="gene_symbol"
    , y_measure="mean"
    , tumor.weights=c(brca=500 , luad=2000))
```

---

showCancerStudy   *List of cancer studies included in the cBioportal*

Description

Show all the possible cancer studies accepted by PrecisionTrialDrawer

Usage

```
showCancerStudy(tumor_type=NULL)
```

Arguments

`tumor_type`	a valid tumor type barcode in TCGA standard coding

Details

This method is a wrapper around `cgdsr-getCancerStudies` and show all the cancer studies included in the cBioPortal. If you specify the tumor_type, follow the list of `showTumorType`. Even though the cancer studies are present, it doesn't mean that there are data for every alteration. A tumor type could have mutation data but not copynumber or viceversa. Furthermore, fusion data have a different source and could not be included in the list.

Value

A named vector of all the cancer studies available in cgdsr package that can be passed to the method getAlterations.

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

`getAlterations` `showTumorType` `cgdsr-getCancerStudies`

Examples

```
myCanStudies <- showCancerStudy()
head(myCanStudies)
```

showTumorType *List of tumor type barcode*

Description

Show all the possible tumor types accepted by PrecisionTrialDrawer

Usage

```
showTumorType()
```

Details

This method is a wrapper around `cgdsr-getCancerStudies` and show all the barcodes for the tumor types included in the cBioPortal. Even though the tumor types are present, it doesn't mean that there are data for every alteration. A tumor type could have mutation data but not copynumber or viceversa. Furthermore, fusion data have a different source and could not be included in the list.

Value

A data.frame of all the tumor types available in cgdsr package that can be passed to the method getAlterations. Every element is the aggregation of all the available sequenced data from all the studies involved in a particular tumor type.

Author(s)

Giorgio Melloni, Alessandro Guida

See Also

`getAlterations` `showCancerStudy` `cgdsr-getCancerStudies`

Examples

```
myTumTypes <- showTumorType()
head(myTumTypes)
```

subsetAlterations *Retrieve exactly the the alteration requested in the panel after a getAlterations method*

Description

As soon as all the data are retrieved from local or remote repositories, this method simply subsets the alterations as exactly requested by the panel and put them in a format that is common to all kinds of alterations

Usage

```
subsetAlterations(object , rules)
```

Arguments

`object`	a CancerPanel object
`rules`	a data.frame with the same columns as the panel plus tumor_type and in_out. Check newCancerPanel function

Details

This method will raise an error if no data are available in the slot dataFull. The method is compulsory to draw all plots since it attaches to each alteration all the panel characteristics, like for example druggability and group.

Value

This method returns the object itself with the slot dataSubset updated. This slot is formed by a list of 4 elements:

`mutations`	a data.frame containing exactly the mutation alterations requested in the panel
`copynumber`	a data.frame containing exactly the copynumber alterations requested in the panel
`expression`	a data.frame containing exactly the expression alterations requested in the panel
`fusions`	a data.frame containing exactly the fusions alterations requested in the panel
`excluded`	a data.frame containing the samples that are resistant to a drug, if rules is set

Each of the first 4 element is a data.frame with the following columns:

`drug`	drug associated with the alteration, if any
`group`	group associated with the alteration, if present
`gene_symbol`	gene associated with the alteration. In case of fusions, a fusion gene in the format gene1__gene2
`tumor_type`	tumor_type in which the alterations were found
`case_id`	sample IDs in which the alterations were found
`alteration_id`	type of alteration with autonumeric identifier. e.g. a specific fusion is fus_1, fus_2 etc.

Please note that the number of alterations per alteration type can have more rows than the original dataFull slot. This is because the same alteration can be a target of multiple drugs or belong to more than one group and is therefore repeated.

Author(s)

Giorgio Melloni, Alessandro Guida

References

data origin for mutations, copynumber and expression data

data origin for fusion data

See Also

`getAlterations` `newCancerPanel`

Examples

```
# Load example CancerPanel object
data(cpObj)
# Launch subsetAlterations excluding certain mutations
# from being considered actionable
rules <- data.frame(
    drug=c("Erlotinib" , "Erlotinib", "Erlotinib","Erlotinib","Olaparib")
    , gene_symbol=c("EGFR" , "KRAS", "" , "", "")
    , alteration=c("SNV" , "SNV", "" , "", "")
    , exact_alteration=c("amino_acid_variant" , "", "" , "", "")
    , mutation_specification=c("T790M" , "" , "" , "", "")
    , group=c("Driver" , "Driver", "Driver" , "Driver", "Driver")
    , tumor_type=c("luad" , "luad" , "luad" , "coadread","brca")
    , in_out=c("exclude" , "exclude" , "include" , "include" , "include")
    , stringsAsFactors = FALSE)
cpObj <- subsetAlterations(cpObj , rules = rules)
# See the updated slot
str(cpDataSubset(cpObj))
```

---

survPowerSampleSize *Calculate sample size or power required in a 2-sample time-to-event (survival) study*

Description

This plot method returns a scatter plot of required sample size at screening by statistical power divided by hazard ratio levels.

Usage

```
survPowerSampleSize(object
, var=c(NA , "drug" , "group" , "gene_symbol" , "alteration_id" , "tumor_type")
, alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
, tumor_type=NULL
, stratum=NULL
, tumor.weights=NULL
, tumor.freqs=NULL
, HR=NULL
, HR0=1
, ber=0.85
, med=NULL
, fu=2
, acc=NULL
, alpha=0.05
, power=NULL
, sample.size=NULL
, side=c(2,1)
, case.fraction=0.5
, collapseMutationByGene=TRUE
, collapseByGene=FALSE
, round.result=TRUE
, priority.trial=NULL
```

```
, priority.trial.order=c("optimal" , "as.is")
, priority.trial.verbose=TRUE
, noPlot=FALSE)
```

Arguments

- `object` a CancerPanel object
- `var` one among NA , "drug" , "group" , "gene_symbol" , "alteration_id" or "tumor_type". It defines the arms of the studies to be projected. With var=NA, the projection of the entire panel is displayed.
- `alterationType` what kind of alteration to include. It can be one or more between "copynumber", "expression", "mutations", "fusions". Default is to include all kind of alterations.
- `tumor_type` only plot one or more tumor types among the ones available in the object.
- `stratum` a character vector containing one or more specific elements of var to be plotted instead of all the arms of the study. If it is not present, a warning is raised and the full design is returned.
- `tumor.weights` A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details
- `tumor.freqs` A named vector of values between 0 and 1 which sum 1. It contains the expected proportion of patients that are planned to be screened. See Details
- `HR` a numerical vector of one or more postulated Hazard Ratio (case group/control group).
- `HR0` a numerical vector of the same length of HR that postulate the Hazard Ratio of the null hypothesis. Default is 1, no difference in risk between the two groups.
- `ber` a numerical value > 0. baseline event rate is the expected number of events per unit of time in control group. Default 0.85
- `med` a numerical value between > 0. median survival time in control group. If set, it takes precedence over ber as one can derive ber from med and viceversa. Default NULL.
- `fu` average follow-up time for the study. Default is 2
- `acc` accrual time for the study. Defualt is NULL so that only follow-up time is considered
- `alpha` a numerical value between 0 and 1 that reports the type I error threshold. Default 0.05 (5%)
- `power` a numerical vector of values between 0 and 1 that expresses the level of 1 - type II error. It is used to estimate sample size
- `sample.size` a positive integer numerical vector that reports the postulated sample size at screening. It is used to estimate the power of the study.
- `side` perform a 2-tail or 1-tail calculation. Default 2
- `case.fraction` a numerical value between 0 and 1 representing the fraction of total sample size allocated to group 'A'
- `collapseMutationByGene` A logical that collapse all mutations on the same gene for a single patient as a single alteration.
- `collapseByGene` A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.

`round.result`	logical indicating if the sample size should be rounded with ceiling or not.
`priority.trial`	A character vector of drugs or group levels to start the design of a priority trial. See Details.
`priority.trial.order`	Either "optimal" or "as.is". If "optimal" is used, the screening starts from the rarest drug or group level up to the most common to guarantee minimal sample size at screening. In case of "as.is", the order of priority.trial remains unchanged.
`priority.trial.verbose`	If TRUE, the result of a priority.trial will be a complete report in a 5-element list.
`noPlot`	if TRUE, the plot is not shown and data are reported instead.

Details

This method estimates sample size or power on the basis of one of the two information. Using multiple sample sizes or power, power curves are reported simulating different scenarios. Power or sample size are required but not both at the same time. HR must be also set but if a vector is provided, the plot will show multiple curves according to the various hazard ratios. 'p.event', 'alpha' and 'case.fraction' are instead fixed for all the arms of the study (represented by the 'var' parameter).

If noPlot=TRUE, a data.frame with 6 column is reported instead:

Var levels of chosen variable

ScreeningSampleSize total sample size estimation at screening on the basis of frequency of alteration

EligibleSampleSize sample size estimated as sum of cases and controls after screening

Beta tested beta values

Power tested 1 - beta values

HazardRatio levels of hazard ratio tested

The algorithm estimates sample size on the basis of no a priori probability of finding a case or control subject ("EligibleSampleSize" column). In a basket or umbrella design, this number must be multiplied by the frequency of alteration that we expect to find based on the simulation run on the panel. If our panel can cover the 50% of the samples with a target therapy and 100 samples are required to reach 80% power, we have to screen at least 200 patients in order to reach the desired number of cases in the sample size ("ScreeningSampleSize" column).

Similarly, if you want to estimate the power of the panel given an estimated sample size, we first multiply 'sample.size' by the frequency of expected alterations and then perform power estimation. 'sample.size' is therefore intended at screening.

When 'var' variable is set, the algorithm provides the estimated sample size for each stratum of the variable. For example, if we set it to 'drug', a power curve for each drug type is displayed, without taking into account possible overlaps. If a sample shows multiple targettable alterations, it will be reused for every drug type that targets those alterations.

By default, `survPowerSampleSize` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your design can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then bootstrap them
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using `tumor.freqs` parameter. In this case the fraction of altered samples are first calculated tumor-wise and then re aggregated using the weights provided by `tumor.freqs`. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. For having a quick idea of the sample size required, it is better to use the former. To get an idea about the possible distribution of sample size giving a few samples (for example a minimum and a maximum sample size) it is better to run the function with tumor.weights several times and aggregate the results.

If priority.trial is set, a cascade design is build up. Given a set of parameter (power, HR, alpha, etc.) an Eligible Sample Size (ESS) is calculated that is the same across drugs/groups. The total Screening Sample Size (SSS) is calculated following this scheme:

1. Start screening with the first drug/group, reaching the sample size necessary to reach ESS
2. From the samples not eligible for the first drug/group, test the second drug/group and collects as many samples as possible up to ESS
3. Continue using the samples not eligible to the end of all drugs/levels. Stop if there are no leftovers.
4. If all the drugs/groups have reached ESS, stop. Otherwise start a new screening with the first drug/group that has not reached ESS
5. Repeat from point 2 up to completion

If priority.trial.order is set, the user can decide if the drugs/group levels must follow a precise order (as.is) or if the screening can start from the rarest drug/group level up to the most common (optimal). Following the optimal priority trial guarantees the best possible allocation with the minimum screening.

Value

If noPlot = FALSE (default) a scatter plot is returned. If noPlot = TRUE, a data.frame is returned. In case priority.trial is set, a 5-element list is reported. See vignette for details.

Author(s)

Giorgio Melloni, Alessandro Guida

References

Schoenfeld DA. Sample-size formula for the proportional-hazards regression model. Biometrics 1983;39:499-503.

See Also

`coveragePlot` `propPowerSampleSize`

Examples

```
# Load example CancerPanel object
data(cpObj)
# Show the full design:
# 3 hazard ratios and 4 power levels
survPowerSampleSize(cpObj
  , var = NA
  , HR = c(1.5 , 1.7,  2, 3)
  , power = c(0.6 , 0.7 , 0.8 , 0.9))
# Design a priority trial to reach sufficient statistical power across drugs
# Use 4 drugs and find the optimal sample size at screening
survPowerSampleSize(cpObj , var = "drug"
    , HR = c(0.5 , 0.8 , 0.9)
    , power = c(0.7 , 0.8)
    , priority.trial = c("Idelalisib" , "Olaparib"
    , "Trastuzumab" , "Vandetanib")
    , priority.trial.order = "optimal")
```

`survPowerSampleSize1Arm` *Calculate sample size or power required in a 1-sample time-to-event (survival) study*

Description

This plot method returns a scatter plot of required sample size at screening by statistical power divided by median case survival time levels.

Usage

```
survPowerSampleSize1Arm(object
, var=c(NA , "drug" , "group" , "gene_symbol" , "alteration_id" , "tumor_type")
, alterationType=c("copynumber" , "expression" , "mutations" , "fusions")
, tumor_type=NULL
, stratum=NULL
, tumor.weights=NULL
, tumor.freqs=NULL
, MED1=NULL
, MED0=NULL
, fu=2
, acc=NULL
, alpha=0.05
, power=NULL
, sample.size=NULL
, side=c(2,1)
, collapseMutationByGene=TRUE
, collapseByGene=FALSE
, round.result=TRUE
, priority.trial=NULL
, priority.trial.order=c("optimal" , "as.is")
, priority.trial.verbose=TRUE
, noPlot=FALSE)
```

Arguments

`object` a CancerPanel object

`var` one among NA , "drug" , "group" , "gene_symbol" , "alteration_id" or "tumor_type". It defines the arms of the studies to be projected. With var=NA, the projection of the entire panel is displayed.

`alterationType` what kind of alteration to include. It can be one or more between "copynumber", "expression", "mutations", "fusions". Default is to include all kind of alterations.

`tumor_type` only plot one or more tumor types among the ones available in the object.

`stratum` a character vector containing one or more specific elements of var to be plotted instead of all the arms of the study. If it is not present, a warning is raised and the full design is returned.

`tumor.weights` A named vector of integer values containing an amount of samples to be randomly sampled from the data. Each element should correspond to a different tumor type and is named after its tumor code. See details

`tumor.freqs` A named vector of values between 0 and 1 which sum 1. It contains the expected proportion of patients that are planned to be screened. See Details

`MED1` numeric value or vector. median survival time for case group.

`MED0` numeric value or vector. historical control survival time

`fu` average follow-up time for the study. Default is 2

`acc` accrual time for the study. Defualt is NULL so that only follow-up time is considered

`alpha` a numerical value between 0 and 1 that reports the type I error threshold. Default 0.05 (5%)

`power` a numerical vector of values between 0 and 1 that expresses the level of 1 - type II error. It is used to estimate sample size

`sample.size` a positive integer numerical vector that reports the postulated sample size at screening. It is used to estimate the power of the study.

`side` perform a 2-tail or 1-tail calculation. Default 2

`collapseMutationByGene` A logical that collapse all mutations on the same gene for a single patient as a single alteration.

`collapseByGene` A logical that collapse all alterations on the same gene for a single patient as a single alteration. e.g. if a sample has TP53 both mutated and deleted as copynumber, it will count for one alteration only.

`round.result` logical indicating if the sample size should be rounded with ceiling or not.

`priority.trial` A character vector of drugs or group levels to start the design of a priority trial. See Details.

`priority.trial.order` Either "optimal" or "as.is". If "optimal" is used, the screening starts from the rarest drug or group level up to the most common to guarantee minimal sample size at screening. In case of "as.is", the order of priority.trial remains unchanged.

`priority.trial.verbose` If TRUE, the result of a priority.trial will be a complete report in a 5-element list.

`noPlot` if TRUE, the plot is not shown and data are reported instead.

Details

This method estimates sample size or power on the basis of one of the two information. Using multiple sample sizes or power, power curves are reported simulating different scenarios. Power or sample size are required but not both at the same time. HR must be also set but if a vector is provided, the plot will show multiple curves according to the various hazard ratios. 'p.event', 'alpha' and 'case.fraction' are instead fixed for all the arms of the study (represented by the 'var' parameter).

If noPlot=TRUE, a data.frame with 6 column is reported instead:

Var levels of chosen variable

ScreeningSampleSize total sample size estimation at screening on the basis of frequency of alteration

EligibleSampleSize sample size estimated as sum of cases and controls after screening

Beta tested beta values

Power tested 1 - beta values

MedianSurvivalCase levels of postulated median survival time tested

The algorithm estimates sample size on the basis of no a priori probability of finding a case or control subject ("EligibleSampleSize" column). In a basket or umbrella design, this number must be multiplied by the frequency of alteration that we expect to find based on the simulation run on the panel. If our panel can cover the 50% of the samples with a target therapy and 100 samples are required to reach 80% power, we have to screen at least 200 patients in order to reach the desired number of cases in the sample size ("ScreeningSampleSize" column).

Similarly, if you want to estimate the power of the panel given an estimated sample size, we first multiply 'sample.size' by the frequency of expected alterations and then perform power estimation. 'sample.size' is therefore intended at screening.

When 'var' variable is set, the algorithm provides the estimated sample size for each stratum of the variable. For example, if we set it to 'drug', a power curve for each drug type is displayed, without taking into account possible overlaps. If a sample shows multiple targettable alterations, it will be reused for every drug type that targets those alterations.

By default, `survPowerSampleSize1Arm` will use all the available data from the object, using all the samples for the requested alterationTypes. Nevertheless, one could be interested in creating a compound design that is composed by a certain number of samples per tumor type. This is the typical situation of basket trials, where you seek for specific alteration, rather than specific tumor types and your design can be stopped when the desired sample size for a given tumor type is reached. By adding tumor.weights, we can achieve such target (see examples). Unfortunately, there are two main drawbacks in doing so:

1. small sample size: by selecting small random samples, the real frequency can be distorted. to avoid this, it is better to run several small samples and then bootstrap them
2. recycling: if the sample size for a tumor type requested by the user is above the available number of cBioportal samples, the samples are recycled. This has the effect of stabilizing the frequencies but y_measure = "absolute" will have no real meaning when the heterogeneity of the samples is lost.

A user balanced design can be also obtained using `tumor.freqs` parameter. In this case the fraction of altered samples are first calculated tumor-wise and then re aggregated using the weights provided by `tumor.freqs`. If the fraction of altered samples are 0.3 and 0.4 for breast cancer and lung cancer respectively, if you set tumor.freqs = c(brca=0.9 , luad=0.1), the full design will have a frequency equal to 0.3*0.9 + 0.4*0.1 = 0.31, that is basically equal to the one of breast samples. If this parameter is not set, the total amount of samples available is used with unpredictable balancing.

Both tumor.freqs and tumor.weights can achieve a balanced design according to user specification. For having a quick idea of the sample size required, it is better to use the former. To get an idea about the possible distribution of sample size giving a few samples (for example a minimum and a maximum sample size) it is better to run the function with tumor.weights several times and aggregate the results.

If priority.trial is set, a cascade design is build up. Given a set of parameter (power, HR, alpha, etc.) an Eligible Sample Size (ESS) is calculated that is the same across drugs/groups. The total Screening Sample Size (SSS) is calculated following this scheme:

1. Start screening with the first drug/group, reaching the sample size necessary to reach ESS
2. From the samples not eligible for the first drug/group, test the second drug/group and collects as many samples as possible up to ESS
3. Continue using the samples not eligible to the end of all drugs/levels. Stop if there are no leftovers.
4. If all the drugs/groups have reached ESS, stop. Otherwise start a new screening with the first drug/group that has not reached ESS
5. Repeat from point 2 up to completion

If priority.trial.order is set, the user can decide if the drugs/group levels must follow a precise order (as.is) or if the screening can start from the rarest drug/group level up to the most common (optimal). Following the optimal priority trial guarantees the best possible allocation with the minimum screening.

Value

If noPlot = FALSE (default) a scatter plot is returned. If noPlot = TRUE, a data.frame is returned. In case priority.trial is set, a 5-element list is reported. See vignette for details.

Author(s)

Giorgio Melloni, Alessandro Guida

References

Lawless, Jerald F. Statistical Models and Methods for Lifetime Data. 2nd ed. John Wiley Sons, 2003.

See Also

coveragePlot propPowerSampleSize

Examples

```
# Load example CancerPanel object
data(cpObj)
# Show the full design:
# 3 median survival times (MED1) against 1 historical value (MED0)
# follow-up time at 24 months
survPowerSampleSize1Arm(cpObj
    , var = NA
    , MED1 = c(12 , 6 , 4)
    , MED0 = 3
    , fu = 18
    , power = c(0.6 , 0.7 , 0.8 , 0.9)
```

```
    )
# Show the study design by tumor type:
# 3 hazard ratios and 4 power levels
# The full design is weighted using tumor.freqs
# The final sample size is composed by 90% luad and 10% brca
survPowerSampleSize1Arm(cpObj
    , var = "tumor_type"
    , MED1 = c(12 , 6 , 4)
    , MED0 = 3
    , fu = 18
    , power=c(0.5 , 0.6 , 0.7 , 0.8 , 0.9)
    , tumor.freqs = c(brca=0.1 , luad=0.9))
```

Index