Package 'SuRFR'

September 11, 2014

Type Package

Title SNP Ranking by Functionality

Version 0.99.0

Date 2014-06-07

Author Niamh Ryan and Stewart Morris

Maintainer Niamh Ryan <N.M.Ryan@sms.ed.ac.uk>

Description A prioritisation tool for ranking SNPs on the likelihood of functionality

License file LICENSE

Depends R (>= 2.15.0), Rsamtools (>= 1.6.0), SAILR (>= 0.99.0)

LazyData TRUE

biocViews SNP, FunctionalPrediction, FunctionalGenomics, Genetics

R topics documented:

`SuRFR-package` *This package takes in a table of SNP functional data and uses it to produce a ranked list of SNPs in order most likely to be functional to least likely*

Description

By combining data from multiple publicly available sources, this package can rank SNPs based on the likelihood of functionality. This method is particularly good at prioritising functional non-coding SNPs over a background set.We have designed this method as an aid to identifying disease causing and/or disease-associated SNPs for genomics projects such as next generation sequencing projects and genome wide association studies.

Details

Package:	SuRFR
Type:	Package
Version:	0.99.0
Date:	2014-06-13
License:	Artistic-2.0

SuRFR prioritises variants in order of most likely to be functional to least on a combination of genomic annotations including conservation, chromatin states, minor allele frequency and DNase hypersensitivity.

Author(s)

Niamh Ryan and Stewart Morris

Maintainer: Niamh Ryan <N.M.Ryan@sms.ed.ac.uk>

References

SuRFRing the genomics wave: an R package for prioritising SNPs by functionality

`ClinVar_Test` *Test data for the package. Annotation table for known pathogenic and non-pathogenic variants from the ClinVar archive of disease variants.*

Description

This dataset consists of 128 pathogenic and 150 non-pathogenic variants from the ClinVar archive of human variation. These variants were extracted from the datasets generated by Ritchie et al. (2014) to test the method GWAVA (ftp://ftp.sanger.ac.uk/pub/resources/software/gwava/v1.0/paper_data/). GWAVA and its associated data are distributed under Apache License-2.0.

Usage

```
data(ClinVar_Test)
```

Format

A data frame with 278 observations on the following 35 variables.

`Pos` a character vector

`RS` a character vector

`refbase` a character vector

`altbase` a character vector

`DAF.G1K` a character vector

`GeneUnique` a character vector

`Exon` a character vector

`Nearest.exon` a character vector

`Nearest.centro.gene` a character vector

`Nearest.telo.gene` a character vector

`Promoter` a character vector

`CpG` a character vector

`CpG_shore` a character vector

`wgEncodeRegDnaseClustered` a character vector

`DNase_footprint` a numeric vector

`wgEncodeBroadHmmGm12878HMM` a character vector

`wgEncodeBroadHmmH1hescHMM` a character vector

`wgEncodeBroadHmmHepg2HMM` a character vector

`wgEncodeBroadHmmHmecHMM` a character vector

`wgEncodeBroadHmmHsmmHMM` a character vector

`wgEncodeBroadHmmHuvecHMM` a character vector

`wgEncodeBroadHmmK562HMM` a character vector

`wgEncodeBroadHmmNhekHMM` a character vector

`wgEncodeBroadHmmNhlfHMM` a character vector

`GERP.UCSC` a numeric vector

`Splice` a character vector

`FANTOM5_promoter_peaks` a character vector

`FANTOM5_promoter_genes` a character vector

`FANTOM510kb_peaks` a character vector

`FANTOM5_10kb_genes` a character vector

`nearest_TSS` a numeric vector

`nearest_FANTOM_TSS` a numeric vector

`Downstream_dist` a character vector

`Enhancer_Atlas` a character vector

`TFBSs` a character vector

Details

Minor Allele Frequencies and RS numbers: from the 1000 Genomes phase-1 populations.

Conservation: GERP estimates position-specific evolutionary rates and identifies candidate constrained elements [1]. Constraint is measured in terms of Rejection Substitutions scores based on a comparison of the number of observed versus expected substitutions. GERP scores for the entire genome were obtained from the UCSC Genome Browser's ftp site.

DNase hypersensitivity (HS): Genome-wide DNase HS data assayed in 125 cell types generated by the University of Washington [2] and Duke University [3] ENCODE groups (wgEncodeRegDnaseClusteredV2).

DNase footprints: Deep sequencing DNase footprinting data from ENCODE project, generated by the University of Washington ENCODE group, defines regions of the genome that interact with and bind DNA binding proteins [4].

Chromatin states: Ernst et al. (2011) systematically mapped nine chromatin marks to nine cell lines and used a multivariate hidden Markov model to distinguish different chromatin states through recognition of combinatorial patterns of chromatin marks. These data have been rigorously tested and confirmed by in vitro assays [5]. Fifteen chromatin states are predicted, including active promoter, weak promoter, strong enhancer, transcriptional elongation, polycomb repressed, repetitive /copy-number-variant. The data from each of the nine cell lines are annotated in this dataset.

Position: SNP annotations across a range of positional categories: whether they are in exons; introns; splice sites; CpG islands; CpG shores; promoters (defined as being within 1kb of a transcription start site) 10kb upstream or downstream of a gene; or intragenic (beyond 10kb up or downstream of a gene). Genic data was sourced from both UCSCs gene annotation data and the FANTOM5 CAGE dataset, which defines novel transcription start sites and, therefore, novel promoters.

Transcribed Enhancers: An atlas of active, transcribed enhancers across the majority of human tissues and cell types was produced using data from the FANTOM5 project [6]. These CAGE-defined enhancers were shown to be more likely to be validated by enhancer assays than predicted enhancers identified using methods such as mapping of chromatin marks or DNase HS sites.

Transcription Factor Binding Sites (TFBSs): Putative TFBSs identified using ChiP-seq experiments for 161 transcription factors across 91 cell types and predicted transcription factor binding motifs from the ENCODE Factorbook repository (wgEncodeRegTfbsClusteredV3) [7, 8].

Source

http://www.nvbi.nlm.hin.gov/clinvar/

References

1. Cooper, G.M., et al., Distribution and intensity of constraint in mammalian genomic sequence. Genome Res, 2005. 15(7): p. 901-13.

2. Sabo, P.J., et al., Discovery of functional noncoding elements by digital analysis of chromatin structure. Proc Natl Acad Sci U S A, 2004. 101(48): p. 16837-42.

3. Song, L. and G.E. Crawford, DNase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells. Cold Spring Harb Protoc, 2010. 2010(2): p. pdb prot5384.

4. Sabo, P.J., et al., Genome-scale mapping of DNase I sensitivity in vivo using tiling DNA microarrays. Nat Methods, 2006. 3(7): p. 511-8.

5. Ernst, J., et al., Mapping and analysis of chromatin state dynamics in nine human cell types. Nature, 2011. 473(7345): p. 43-9.

6. Andersson, R., et al., An atlas of active enhancers across human cell types and tissues. Nature, 2014. 507(7493): p. 455-61.

7. Wang, J., et al., Sequence features and chromatin structure around the genomic regions bound by 119 human transcription factors. Genome Res, 2012. 22(9): p. 1798-812.

8. Wang, J., et al., Factorbook.org: a Wiki-based database for transcription factor-binding data generated by the ENCODE consortium. Nucleic Acids Res, 2013. 41(Database issue): p. D171-6.

9. Landrum, M.J., et al., ClinVar: public archive of relationships among sequence variation and human phenotype. Nucleic Acids Res, 2014. 42(Database issue): p. D980-5.

Examples

```
data(ClinVar_Test)

SuRFR_analysis_data.frame(ClinVar_Test,data_name="ClinVar_Test_out",

model=ALL, MAF=unique)
```

Denovo_anno_table *Denovo annotation table function*

Description

This function makes use of a unix shell script, rebuild, to build denovo functional annotation tables for SNPs of interest. All it requires is a tab delimited text file containing the chr, start and end coordinates for each SNP. Make sure you do NOT have a header row.

The shell script is unix dependent and requires additional tools such as bedtools and awk to function. It is therefor not suitable for use on Windows.

Usage

```
Denovo_anno_table(file, pop="EUR", threads = 1)
```

Arguments

file	A tab delimited file (no header line!) containing the columns: chr, start and end (in hg19 format)
pop	The population you wish to extract RS numbers and MAFs from. Options are EUR,AFR,AMR, ASN, or NONE (if you don't want any MAF or RS numbers).
threads	Threads can be used to speed up this function by allowing you to split up a big file into smaller chunks and run in parallel across mutiple cores on your computer. If more threads are specified than the number of cores available, the threads are queued until a core becomes available.

Examples

```
file <- system.file("extdata", "test.data.bed")


function(file, pop="EUR", threads = 1) {
        script <- system.file("scripts", "rebuild", package="SuRFR")
```

```
        run <- paste(script, file, pop, sep=" ")
        system(run)
}
```

`Ernst_states` *Weighting values for each chromatin state.*

Description

Ernst et al. (2011) systematically mapped nine chromatin marks to nine cell lines and used a multivariate hidden Markov model to distinguish different chromatin states through recognition of combinatorial patterns of chromatin marks. These data have been rigorously tested and confirmed by in vitro assays [17]. Fifteen chromatin states are predicted, including active promoter, weak promoter, strong enhancer, transcriptional elongation, polycomb repressed, repetitive /copy-number-variant.We performed multivariable regression on a training dataset of functional and background variants to correlate the chromatin states with true regulatory variants and ranked them in order of most to least informative. Chromatin states were assessed across nine cell lines, and the most highly ranked state from any cell line was chosen to represent the chromatin state of each SNP.

This function sets the ranking of the chromatin state classes.

Details

The rankings of chromatin states defined in this function were chosen based on multivariable regression analysis of a large training/validation dataset containing a combination of true functional regulatory variants and background variants. However, rankings can be changed by the user to suit personal analysis requirements, by modifying the vetor "ranks".

Author(s)

Niamh Ryan

Source

wgEncodeBroadHmm tracks from UCSC Genome Browser, available via MySQL: mysql -u genome -h genome-mysql.cse.ucsc.edu -A -D hg19 -P 3306

References

Ernst et al., Mapping and analysis of chromatin state dynamics in nine human cell types, Nature, 2011

Examples

```
function (x, ranks=c(10,9,8,7,6,4,3,2,1,5))
{

Ernst_promoter = c("1_Active_Promoter", "2_Weak_Promoter", "3_Poised_Promoter")
Ernst_Strong_En = c("4_Strong_Enhancer", "5_Strong_Enhancer")
Ernst_Weak_En = c("6_Weak_Enhancer", "7_Weak_Enhancer")
Ernst_Repressed = c("12_Repressed")
```

```
Ernst_Insulator = c("8_Insulator")
Ernst_Txn_Trans = c("9_Txn_Transition")
Ernst_Txn_Elong = c("10_Txn_Elongation")
Ernst_Weak_Txn = c("11_Weak_Txn")
Ernst_Heterochrom = c("13_Heterochrom/lo")
Ernst_unsure = c("14_Repetitive/CNV", "15_Repetitive/CNV", "NA")


    x[x %in% Ernst_promoter] = ranks[1]
    x[x %in% Ernst_Strong_En] = ranks[2]
    x[x %in% Ernst_Weak_En] = ranks[3]
    x[x %in% Ernst_Repressed] = ranks[4]
    x[x %in% Ernst_Insulator] = ranks[5]
    x[x %in% Ernst_Txn_Trans] = ranks[6]
    x[x %in% Ernst_Txn_Elong] = ranks[7]
    x[x %in% Ernst_Weak_Txn] = ranks[8]
    x[x %in% Ernst_Heterochrom] = ranks[9]
    x[x %in% Ernst_unsure] = ranks[10]
    as.numeric(x)
  }
```

`Ernst_weighted_funct` *Weighting of chromatin states*

Description

Chromatin states were correlated with true regulatory variants and ranked them in order of most to least informative. Chromatin states were assessed across nine cell lines, and the most highly ranked state from any cell line was chosen to represent the chromatin state of each SNP. This function applies the ranking of the chromatin states to each SNP across the nine cell lines. The highest ranking chromatin state from any cell line was used to rank the SNPs.

Usage

```
Ernst_weighted_funct(x, lookup, input, ranks=c(10,9,8,7,6,4,3,2,1,5))
```

Arguments

`x`

`lookup`

`input`

`ranks`

Examples

```
# The function is defined as:
function (x, lookup, input, ranks=c(10,9,8,7,6,4,3,2,1,5))
{
    numRows <- length(x[, lookup[2, 1]])
    numCols <- length(lookup[, 1])
```

```
    result_Ernst <- as.data.frame(matrix(data = 0, nrow = numRows,
        ncol = numCols), stringsAsFactors = FALSE)
    colnames(result_Ernst) <- lookup[, 1]
    for (i in 1:length(lookup[, 1])) {
        test.ernst <- data.frame(input[lookup[i, 2]], stringsAsFactors = FALSE)
        test <- sapply(test.ernst, Ernst_states, ranks)
        result_Ernst[, i] <- test
    }
    Ernst_Top <- as.data.frame(matrix(data = 0, nrow = numRows,
        ncol = 1), stringsAsFactors = FALSE)
    for (q in 1:numRows) {
        Ernst_Top$V1[q] <- max(result_Ernst[q, ])
    }
    colnames(Ernst_Top) <- c("Ernst_score")
    return(Ernst_Top)
}
```

MAF *Optimum minor allele frequency for ranking variants*

Description

High penetrance and Mendelian diseases tend to be caused by very rare variants. In contrast, it is proposed that complex disorders can be caused by a combination of common and rare variants (Manolio et al., Nature 2009). Understanding the disease model under investigation can allow scientists to hypothesise the frequency of a causal variant in the population and use this information to reduce the number of potential variants to investigate. Therefore, flexibility is required to best utilise minor allele frequency (MAF) to identify functional candidates, either by matching the MAF of an associated variant or prioritizing unique variants in a disease linked region. Variants are ranked using a Gamma distribution to allow the optimal allele frequency range to be modified to suit each analysis.

The default setting for MAF is "unique" (ie MAF = 0).

Usage

```
data(MAF)
```

Format

The format is: unique: num 1e-06

10% MAF: num 10

25% MAF: num 25

Details

Gamma distributon set as: data <- curve(dgamma(x, scale=1, shape=0.000001),from=0, to=100, main="Gamma distribution")

By keeping the scale at one and changing the shape to match the optimum MAF we can rank variants on either side of the optimum value.

If the disease variant is excpected to be unique, leave MAF as the default (unique) setting (ie 0.000001). SNPs will be ranked with the lowest MAF SNPs ranked above higher MAF SNPs.

If you expect your disease variant to be present at roughtly 10 percent MAF in the population, change MAF to 10:

> MAF <- 10

> SurFR_analysis_data.frame(data_in, model, MAF, data_name)

=> SNPs will be ranked on MAF: those closest to 10 percent MAF will be ranked highest.

Examples

```
data(ClinVar_Test)

data_in <- ClinVar_Test


# default MAF = unique
# ie default MAF = 0.000001

SuRFR_analysis_data.frame(data_in, data_name="ClinVar_Test_SuRFR", model=DFP, MAF=10)
```

`model` *Weighting models for different versions of SuRFR*

Description

SuRFR prioritises genomic variants on the likelihood of functionality. It does this by producing rank orderings of variants for a diverse range of functional genomic annotations. The individual ranks are combined into a final prioritisation rank using a weighting system. The R package provides three models: a general model for any analysis (ALL); a model designed specifically for prioritizing (rare) disease variants (DM); and a model for complex disease variants (DFP). In addition, SuRFR allows users to specify their own custom model (custom). "model" is the default model for SuRFR (same as ALL).

Usage

```
data(model)
```

Format

The format is:

model: num [1:8] 0 1 4 0 13 1 0 2

ALL: num [1:8] 0 1 4 0 13 1 0 2

DM: num [1:8] 11 3 5 0 15 1 0 4

DFP: num [1:8] 0 0 3 0 11 3 3 1

Details

Users can customize their own model by setting custom: custom <- c(a,b,c,d,e,f,g,h)

a = MAF weighting;

b = conservation weighting;

c = chromatin state weighting;

d = DNase HS cluster weighting;

e = Position wighting;

f = DNase footprint weighting;

g = transcribed Enhancer weighting;

h = TFBS weighting;

If the user wants to leave out an annotation, set its weighting to "0".

References

SuRFING the genomics wave: an R package for prioritizing SNPs by functionality

Examples

```
data(model)
```

Position_ranks *Weightings for the position categories*

Description

SNPs were ranked based on the positional categories (exon, intron, splice site, promoter- defined as being within 1kb of a transcription start site, 10kb upstream or downstream of a gene, intragenic, CpG islands, CpG shores) they fall into, the order of the categories based on enrichment data presented by Schork and Hindorff [1, 2].

Usage

```
Position_ranks(input, pos_ranks)
```

Arguments

input	The input data.frame containing all the required annotation data for each SNP
pos_ranks	Vector containing the ranking values for all the position categories.

Details

pos_ranks is a 7 element vector, providing the rank order for the position categories:

pos_ranks <- c(a,b,c,d,e,f,g)

a = introns;

b = CpG shores;

c = CpG islands;

d = 10kb upstream or downstream of genes;

e = promoters (1kb upstream of a transcription start site);

f = splice sites;

g = exons;

Default setting:

pos_ranks <- c(1,2,2,3,4,5,5)

References

1. Schork, A.J., et al., All SNPs are not created equal: genome-wide association studies reveal a consistent pattern of enrichment among functionally annotated SNPs. PLoS Genet, 2013. 9(4): p. e1003449.

2. Hindorff, L.A., et al., Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. Proc Natl Acad Sci U S A, 2009. 106(23): p. 9362-7.

Examples

```
data(ClinVar_Test)

input <- ClinVar_Test
pos_ranks<- c(1,2,2,3,4,5,5)

function (input, pos_ranks)
{

    F_Position = as.data.frame(matrix(data=0, nrow=numRows, ncol=1))

    # next test if it is in an intron:
    F_Position$V1[input$Nearest.exon != "NA"] = pos_ranks[1]

    # CpG shores:
    F_Position$V1[input$CpG_shore != "NA"] = pos_ranks[2]

    # CpG island:
    F_Position$V1[input$CpG != "NA"] = pos_ranks[3]

    # 10 kb up or down stream
    F_Position$V1[input$Nearest.telo.gene != "NA" ] = pos_ranks[4]
    F_Position$V1[input$Nearest.centro.gene != "NA" ] = pos_ranks[4]

    F_Position$up_10kb <- 0

    for (q in 1:numRows){
     if (input$nearest_FANTOM_TSS[q] < -1000 & -10000 <= input$nearest_FANTOM_TSS[q]) {
        F_Position$up_10kb[q] <- pos_ranks[4]
        }
     }

    # Promoter
    F_Position$promoter <- 0
    F_Position$V1[input$Promoter != "NA"] = pos_ranks[5]

    for (q in 1:numRows){
     if (input$nearest_FANTOM_TSS[q] < 0 & -1000 <= input$nearest_FANTOM_TSS[q]) {
        F_Position$promoter[q] <- pos_ranks[5]
```

```
        }
    }

    # Splice sites:
    F_Position$V1[input$Splice != "NA"] = pos_ranks[6]

    # Exons:
    F_Position$V1[input$Exon != "NA"] = pos_ranks[7]

    return(F_Position)

}
```

SuRFR_analysis_data.frame *SNP weighting and ranking function for annotation data in data.frame format*

Description

This function takes in the annotation data for a list of SNPs (such as the out put pf the function SuRFR_annotation) and produces an output table of SNPs ranked on this data.

Usage

```
SuRFR_analysis_data.frame(data_in,data_name, model,MAF,chrom, pos_ranks)
```

Arguments

data_in	The input data frame
model	The weighting model to be used to rank the SNPs. Four model options are available: ALL, DM, DFP and custom. See "model" for more details,
MAF	The optimum MAF to be used to rank SNPs on. Default value is 1e-06 (ie unique)
data_name	The data name to be included in the output filename
chrom	The function to rank chromatin states. Default option is Ernst_states, however a customisable version is also available, Ernst_states_custom, which the user can use to define their own ranks for the chromatin states.
pos_ranks	Vector containing the ranking values for all the position categories.

Author(s)

Niamh Ryan

Examples

```
data(ClinVar_Test)

File <- ClinVar_Test


function (data_in, data_name="ClinVar_Test_SuRFR", model=DM, MAF=unique,

chrom=c(10,9,8,7,6,4,3,2,1,5), pos_ranks=c(1,2,2,3,4,5,5))
{
    if (length(model) != 8) {
        stop("model should contain 8 weightings")
    }

    weightings <- model
    weightings <- as.data.frame(matrix(weightings, nrow = 1,
        ncol = 8), stringsAsFactors = FALSE)

    weightings$V1 <- as.numeric(weightings$V1)
    weightings$V2 <- as.numeric(weightings$V2)
    weightings$V3 <- as.numeric(weightings$V3)
    weightings$V4 <- as.numeric(weightings$V4)
    weightings$V5 <- as.numeric(weightings$V5)
    weightings$V6 <- as.numeric(weightings$V6)
    weightings$V7 <- as.numeric(weightings$V7)
    weightings$V8 <- as.numeric(weightings$V8)

    date <- paste(format(Sys.time(), "%Y_%m_%d."), sep = ".")
    data_out <- basename(paste(data_name, ".", date, "out.txt", sep = ""))

    final_result <- SuRFR_weighting_and_ranking_code(data_in,
        weightings, MAF, chrom, pos_ranks)

    final_result$Grand_Total.rank <- rank(-final_result$Grand_Total,
        ties.method = "max")

    ranked_result <- final_result[order(final_result$Grand_Total.rank),
        ]

    write.table(ranked_result, file = data_out, append = FALSE, quote = FALSE,
        sep = "\t", eol = "\n", na = "NA", dec = ".", row.names = FALSE,
        col.names = TRUE)

return(ranked_result)
  }
```

SuRFR_analysis_tab_file

SNP weighting and ranking function for annotation data in tyab delimitted text file format

---

Description

This function takes in the annotation data for SNPs from a tab-delimited text file and produces a table of SNPs ranked on this data, from most likely to be functional to least likely.

Usage

```
SuRFR_analysis_tab_file(filename, model, MAF, chrom, pos_ranks)
```

Arguments

filename	Name of the file to be analysed.
model	Weighting model to be used to rank the SNPs. Four model options are available: ALL, DM, DFP and custom. See model for more details.
MAF	Optimum MAF to be used to rank SNPs on. Default value is 1e-06 (ie unique).
chrom	Option to set the chromatin state ranking function. The default function is Ernst_states, while a second, customisable function, Ernst_states_custom, is also available.
pos_ranks	Vector containing the ranking values for all the position categories.

Author(s)

Niamh Ryan

Examples

```
data(ALL)

test_file <- "HBB_TP_noncoding.bed"
filename <- system.file("extdata", test_file)


function (filename, model=ALL, MAF=unique, chrom=c(10,9,8,7,6,4,3,2,1,5),

pos_ranks=c(1,2,2,3,4,5,5))
{
    input <- read.table(filename, header = TRUE, sep = "\t", stringsAsFactors = FALSE,
        na.strings = ".")
    input_headers <- names(input)
    correct_headers <- c("Pos", "RS", "refbase",
        "altbase", "DAF.G1K", "GeneUnique", "Exon", "Nearest.exon",
        "Nearest.centro.gene", "Nearest.telo.gene", "Promoter",
        "CpG", "CpG_shore", "wgEncodeRegDnaseClustered", "DNase_footprint",
        "wgEncodeBroadHmmGm12878HMM", "wgEncodeBroadHmmH1hescHMM",
        "wgEncodeBroadHmmHepg2HMM", "wgEncodeBroadHmmHmecHMM",
        "wgEncodeBroadHmmHsmmHMM", "wgEncodeBroadHmmHuvecHMM",
        "wgEncodeBroadHmmK562HMM", "wgEncodeBroadHmmNhekHMM",
        "wgEncodeBroadHmmNhlfHMM", "GERP.UCSC", "Splice", "FANTOM5_promoter_peaks",
        "FANTOM5_promoter_genes", "FANTOM510kb_peaks", "FANTOM5_10kb_genes",
        "nearest_TSS", "nearest_FANTOM_TSS", "Downstream_dist",
        "Enhancer_Atlas", "TFBSs")
    for (l in 1:length(correct_headers)) {
        if (!(correct_headers[l] %in% input_headers)) {
            stop("headers don't match required input. Check manual for correct input file
        }
    }
    if (length(model) != 8) {
        stop("model should contain 8 weightings")
    }
```

```
  weightings <- model
  weightings <- as.data.frame(matrix(weightings, nrow = 1,
      ncol = 8), stringsAsFactors = FALSE)
  weightings$V1 <- as.numeric(weightings$V1)
  weightings$V2 <- as.numeric(weightings$V2)
  weightings$V3 <- as.numeric(weightings$V3)
  weightings$V4 <- as.numeric(weightings$V4)
  weightings$V5 <- as.numeric(weightings$V5)
  weightings$V6 <- as.numeric(weightings$V6)
  weightings$V7 <- as.numeric(weightings$V7)
  weightings$V8 <- as.numeric(weightings$V8)
  time <- paste(format(Sys.time(), "%Y-%m-%d_%H:%M"), sep = ".")
  data_out <- basename(paste(filename, ".", time, "out.txt", sep = ""))
  final_result <- SuRFR_weighting_and_ranking_code(input, weightings,
      MAF, chrom, pos_ranks)
  final_result$Grand_Total.rank <- rank(-final_result$Grand_Total,
      ties.method = "max")
  ranked_result <- final_result[order(final_result$Grand_Total.rank),
      ]
  write.table(ranked_result, file = data_out, append = FALSE, quote = FALSE,
      sep = "\t", eol = "\n", na = "NA", dec = ".", row.names = FALSE,
      col.names = TRUE)

  return(ranked_result)
}
```

`SuRFR_annotation` *Accessing SNP annotation from SAILR*

Description

This function uses tabix to pull in precompiled SNP data from SAILR for any of the four 1000 Genomes popultations: EUR, ASN, AMR and AFR. This function requires the user to i) provide a basic tab delimited file containing chromosome number and start and end coordinates (hg19 assembly coordinates) for variants within the 1000 Genomes project and ii) select a population from which to pull the data from (default population being EUR).

It should be noted that if the SNPs of interest are not present in the chosen population a warning message will be returned and those SNPs will be removed from the data.frame.

Usage

```
SuRFR_annotation(bedfile, pop = "EUR")
```

Arguments

`bedfile`	A basic tab delimited file containing chr, start and end coordinates for know SNPs of interest (in hg19 format).
`pop`	Population to pull the SNPs from.

Author(s)

Stewart Morris

Examples

```
test_file <- "test.data.bed"

bedfile <- system.file("extdata", test_file)

function (bedfile, pop = "EUR")
{
    if (!file.exists(bedfile)) {
        stop("\nNo such file: ", bedfile)
    }
    tmpfile <- tempfile()
final <- basename(paste0(bedfile, ".", pop, ".anno"))
    features <- paste0("1000.genomes.", pop, ".AF_header.bed.gz")
    popfile <- system.file("extdata", features, package = "SAILR")
    if (!file.exists(popfile)) {
        stop("\n\nNo such population: [", pop, "]\nValid populations are: AFR AMR ASN EUR
    }
    writeLines(paste0("\nInput file:\t", bedfile, "\nPopulation:\t",
        pop, "\n"))
    tbx <- TabixFile(popfile)
    raw <- read.table(bedfile, header = FALSE, sep = "\t", stringsAsFactors = FALSE,
        na.strings = ".")
ordered <- raw[ base::order(raw[,1], raw[,2], raw[,3], na.last=NA), ]
chr <- ordered[,1]
pos <- ordered[,3]
rawrows <- length(ordered[,1])
    par <- GRanges(chr, IRanges(pos, width = 1))
    res <- scanTabix(tbx, param = par)
    ures <- unlist(res)
    nrows <- length(ures)
    dropped <- rawrows - nrows
    if (dropped != 0) {
        writeLines(paste0("Warning: dropped ", dropped, " positions from \"",
            bedfile, "\" which are not in ", pop, " file\n"))
    }
    df <- data.frame(matrix(ures, nrow = nrows, byrow = TRUE))
    write.table(df, file = tmpfile, append = FALSE, quote = FALSE, sep = "\t",
        row.names = FALSE, col.names = FALSE)
    anno <- read.table(file = tmpfile, header = FALSE, stringsAsFactors = FALSE,
        na.strings = ".", sep = "\t")
    unlink(tmpfile)
    colnames(anno) <- c("chr", "start", "end", "RS", "refbase",
        "altbase", "DAF.G1K", "GeneUnique", "Exon", "Nearest.exon",
        "Nearest.centro.gene", "Nearest.telo.gene", "Promoter",
        "CpG", "CpG_shore", "wgEncodeRegDnaseClustered", "DNase_footprint",
        "wgEncodeBroadHmmGm12878HMM", "wgEncodeBroadHmmH1hescHMM",
        "wgEncodeBroadHmmHepg2HMM", "wgEncodeBroadHmmHmecHMM",
        "wgEncodeBroadHmmHsmmHMM", "wgEncodeBroadHmmHuvecHMM",
        "wgEncodeBroadHmmK562HMM", "wgEncodeBroadHmmNhekHMM",
        "wgEncodeBroadHmmNhlfHMM", "GERP.UCSC", "Splice", "FANTOM5_promoter_peaks",
        "FANTOM5_promoter_genes", "FANTOM510kb_peaks", "FANTOM5_10kb_genes",
        "nearest_TSS", "nearest_FANTOM_TSS", "Downstream_dist",
        "Enhancer_Atlas", "TFBSs")
    Pos_change <- as.data.frame(matrix(data = 0, nrow = nrows,
```

```
        ncol = 1), stringsAsFactors = FALSE)
    Pos_change$V1 <- paste(anno[, 1], anno[, 2], sep = ":")
    Pos_change$V2 <- paste(Pos_change[, 1], anno[, 3], sep = "-")
    anno$Pos <- Pos_change$V2
    write.table(anno, file = final, col.names = TRUE, row.names = FALSE,
        append = FALSE, quote = FALSE, sep = "\t", na = ".")
    return(anno)
  }
```

`SuRFR_weighting_and_ranking_code`

The SNP weighting and ranking function

Description

This function is the central feature of SuRFR, ranking the SNPs across a variety of annotation features and returning a ranked data.frame. This function is used to rank SNPs by the two functions SuRFR_analysis_data.frame and SuRFR_analysis_tab_file.

Usage

```
SuRFR_weighting_and_ranking_code(File, weightings, freq, chrom, pos_ranks)
```

Arguments

`File`	The input data.frame containing all the required annotation data for each SNP
`weightings`	The user defined weightings to combine the individual annotation ranks into a rank of ranks.
`freq`	The user defined optimum minor allele frequency.
`chrom`	Function for ranking chromatin states. Default is Ernst_states, customisable function is Ernst_states_custom
`pos_ranks`	Vector containing the ranking values for all the position categories.

Details

SuRFR prioritises genomic variants on the likelihood of functionality. It does this by producing rank orderings of variants for a diverse range of functional genomic annotations. The individual ranks are combined into a final prioritisation rank of ranks using a weighting system.

Minor Allele Frequencies and RS numbers: from the 1000 Genomes phase-1 populations.

Conservation: GERP estimates position-specific evolutionary rates and identifies candidate constrained elements [1]. Constraint is measured in terms of Rejection Substitutions scores based on a comparison of the number of observed versus expected substitutions. GERP scores for the entire genome were obtained from the UCSC Genome Browser's ftp site.

DNase hypersensitivity (HS): Genome-wide DNase HS data assayed in 125 cell types generated by the University of Washington [2] and Duke University [3] ENCODE groups (wgEncodeRegDnaseClusteredV2).

DNase footprints: Deep sequencing DNase footprinting data from ENCODE project, generated by the University of Washington ENCODE group, defines regions of the genome that interact with and bind DNA binding proteins [4].

Chromatin states: Ernst et al. (2011) systematically mapped nine chromatin marks to nine cell lines using a multivariate hidden Markov model to distinguish different chromatin states through recognition of combinatorial patterns of chromatin marks. These data have been rigorously tested and confirmed by in vitro assays [5]. Fifteen chromatin states are predicted, including active promoter, weak promoter, strong enhancer, transcriptional elongation, polycomb repressed, repetitive /copy-number-variant. The data from each of the nine cell lines are annotated in this dataset.

Position: SNP annotations across a range of positional categories: whether they are in exons; introns; splice sites; CpG islands; CpG shores; promoters (defined as being within 1kb of a transcription start site) 10kb upstream or downstream of a gene; or intragenic (beyond 10kb up or downstream of a gene). Genic data was sourced from both UCSCs gene annotation data and the FANTOM5 CAGE dataset, which defines novel transcription start sites and, therefore, novel promoters.

Transcribed Enhancers: An atlas of active, transcribed enhancers across the majority of human tissues and cell types was produced using data from the FANTOM5 project [6]. These CAGE-defined enhancers were shown to be more likely to be validated by enhancer assays than predicted enhancers identified using methods such as mapping of chromatin marks or DNase HS sites. A binary discriminator was used to assess whether a SNP overlapped a transcribed enhancer or not.

Transcription Factor Binding Sites (TFBSs): Putative TFBSs identified using ChiP-seq experiments for 161 transcription factors across 91 cell types and predicted transcription factor binding motifs from the ENCODE Factorbook repository (wgEncodeRegTfbsClusteredV3) [7, 8].

Author(s)

Niamh Ryan

References

1. Cooper, G.M., et al., Distribution and intensity of constraint in mammalian genomic sequence. Genome Res, 2005. 15(7): p. 901-13.

2. Sabo, P.J., et al., Discovery of functional noncoding elements by digital analysis of chromatin structure. Proc Natl Acad Sci U S A, 2004. 101(48): p. 16837-42.

3. Song, L. and G.E. Crawford, DNase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells. Cold Spring Harb Protoc, 2010. 2010(2): p. pdb prot5384.

4. Sabo, P.J., et al., Genome-scale mapping of DNase I sensitivity in vivo using tiling DNA microarrays. Nat Methods, 2006. 3(7): p. 511-8.

5. Ernst, J., et al., Mapping and analysis of chromatin state dynamics in nine human cell types. Nature, 2011. 473(7345): p. 43-9.

6. Andersson, R., et al., An atlas of active enhancers across human cell types and tissues. Nature, 2014. 507(7493): p. 455-61.

7. Wang, J., et al., Sequence features and chromatin structure around the genomic regions bound by 119 human transcription factors. Genome Res, 2012. 22(9): p. 1798-812.

8. Wang, J., et al., Factorbook.org: a Wiki-based database for transcription factor-binding data generated by the ENCODE consortium. Nucleic Acids Res, 2013. 41(Database issue): p. D171-6.

9. Landrum, M.J., et al., ClinVar: public archive of relationships among sequence variation and human phenotype. Nucleic Acids Res, 2014. 42(Database issue): p. D980-5.

Examples

```
data(ClinVar_Test)
data(DM)
data(MAF)

File <- ClinVar_Test
model <- DM
freq <- MAF

 weightings <- model
    weightings <- as.data.frame(matrix(weightings, nrow = 1,
        ncol = 8), stringsAsFactors = FALSE)

    weightings$V1 <- as.numeric(weightings$V1)
    weightings$V2 <- as.numeric(weightings$V2)
    weightings$V3 <- as.numeric(weightings$V3)
    weightings$V4 <- as.numeric(weightings$V4)
    weightings$V5 <- as.numeric(weightings$V5)
    weightings$V6 <- as.numeric(weightings$V6)
    weightings$V7 <- as.numeric(weightings$V7)
    weightings$V8 <- as.numeric(weightings$V8)



function (File, weightings, chrom=c(10,9,8,7,6,4,3,2,1,5), pos_ranks=c(1,2,2,3,4,5,5), fr
{
    DAF_weight <- weightings[1]
    Conservation_weight <- weightings[2]
    Histone_weight <- weightings[3]
    DNase_weight <- weightings[4]
    DNase_foot_weight <- weightings[6]
    Position_weight <- weightings[5]
    Enhancer_weight <- weightings[7]
    TFBS_weight <- weightings[8]
    input <- File
    numRows <- length(input[, 1])
    input$DAF.G1K <- as.numeric(input$DAF.G1K)
    input$wgEncodeRegDnaseClustered <- as.numeric(input$wgEncodeRegDnaseClustered)
    input$DNase_footprint <- as.numeric(input$DNase_footprint)
    input$GERP.UCSC <- as.numeric(input$GERP.UCSC)
    Results <- as.data.frame(matrix(data = 0, nrow = numRows,
        ncol = 1))
    colnames(Results) <- "Pos"
    Results$Pos <- input$Pos
    headers <- names(input)
    NA_strings <- c(NA, "-")
    SNPs <- length(input[, 1])
    input$GERP.UCSC[which(input$GERP.UCSC < 0)] = 0
    input$GERP.UCSC[input$GERP.UCSC %in% NA_strings] = 0
    input$GERP.UCSC.rank <- rank(input$GERP.UCSC, ties.method = "max")
    input$GERP.UCSC.rank[input$GERP.UCSC == 0] = 1
    input$GERP_av_rank <- (input$GERP.UCSC.rank/SNPs)
    Results$GERP_raw <- input$GERP.UCSC
    Results$GERP.rank <- input$GERP.UCSC.rank
    Results$GERP_av_rank <- input$GERP_av_rank
    Results$Conservation.rank <- input$GERP_av_rank
```

```
input$wgEncodeRegDnaseClustered[input$wgEncodeRegDnaseClustered %in%
    NA_strings] = 0
input$E.DNase.rank <- rank(input$wgEncodeRegDnaseClustered,
    na.last = FALSE, ties.method = "max")
input$E.DNase.rank[input$wgEncodeRegDnaseClustered == 0] = 1
input$E.DNase.av.rank <- (input$E.DNase.rank/SNPs)
Results$DNaseHS_raw <- input$wgEncodeRegDnaseClustered
Results$DNaseHS.rank <- input$E.DNase.rank
Results$DNaseHS.av.rank <- input$E.DNase.av.rank
input$DNase_footprint[input$DNase_footprint %in% NA_strings] = 0
input$DNase.foot.rank <- rank(input$DNase_footprint, na.last = FALSE,
    ties.method = "max")
input$DNase.foot.rank[input$DNase_footprint == 0] = 1
input$DNase.foot.av.rank <- (input$DNase.foot.rank/SNPs)
Results$DNase_foot_raw <- input$DNase_footprint
Results$DNase.foot.rank <- input$DNase.foot.rank
Results$DNase.foot.av.rank <- input$DNase.foot.av.rank
headersOfInterest_Ernst <- grep("BroadHmm", headers, ignore.case = FALSE,
    perl = FALSE, value = TRUE, fixed = FALSE, useBytes = FALSE,
    invert = FALSE)
indexOfHeaders_Ernst <- grep("BroadHmm", headers, ignore.case = FALSE,
    perl = FALSE, value = FALSE, fixed = FALSE, useBytes = FALSE, invert = FALSE)
lookTab_Ernst <- data.frame(name = headersOfInterest_Ernst,
    index = indexOfHeaders_Ernst)
Ernst_score <- Ernst_weighted_funct(input, lookTab_Ernst,
    input, chrom)
input$Ernst_score <- Ernst_score$Ernst_score
input$Ernst_score[input$Ernst_score %in% NA_strings] = 0
input$Ernst_rank <- rank(input$Ernst_score, ties.method = "max")
input$Ernst_av_new.rank <- (input$Ernst_rank/SNPs)
Results$Ernst_score <- input$Ernst_score
Results$Ernst.Av.new.rank <- input$Ernst_av_new.rank
Position <- function(input, pos_ranks)
for (n in 1:numRows) {
    input$Position[n] <- max(Position[n, ])
}
input$Position.rank <- rank(input$Position, ties.method = "max")
input$Position.rank[input$Position == 0] = 1
input$Position.rank <- (input$Position.rank/SNPs)
Results$Position.rank <- input$Position.rank
Results$Position_score <- input$Position
x <- input$DAF.G1K
pdf(file = "~/test_gamma.pdf")
data <- curve(dgamma(x, scale = 1, shape = freq), from = 0,
    to = 100, main = "Gamma distribution")
dev.off()
G1K.EUR = as.data.frame(matrix(data = 0, nrow = numRows,
    ncol = 1))
input$DAF.G1K[input$DAF.G1K %in% NA_strings] = 0
G1K.EUR$V1 <- input$DAF.G1K * 100
G1K.EUR$rounded <- round(G1K.EUR$V1, digits = 0)
G1K.EUR$y <- data$y[match(G1K.EUR$rounded, data$x)]
G1K.EUR$y.rounded <- round(G1K.EUR$y, digits = 4)
G1K.EUR$y.rank <- rank(G1K.EUR$y, ties.method = "max")
input$DAF = G1K.EUR$rounded
input$DAF.rank = G1K.EUR$y.rank
input$DAF.rank_normalised <- (input$DAF.rank/SNPs)
```

```
Results$DAF_raw <- input$DAF.G1K
Results$DAF.rank <- input$DAF.rank
Results$DAF.rank_normalised <- input$DAF.rank_normalised
Enhancers = as.data.frame(matrix(data = 0, nrow = numRows,
    ncol = 1))
Enhancers$V1[input$Enhancer_Atlas != "NA"] = 1
input$Enhancers <- Enhancers$V1
input$Enhancers.rank <- rank(input$Enhancers, ties.method = "max")
input$Enhancers.rank[input$Enhancers == 0] = 1
input$Enhancers.rank <- (input$Enhancers.rank/SNPs)
Results$Enhancers_raw <- input$Enhancers
Results$Enhancers.rank <- input$Enhancers.rank
input$TFBSs[input$TFBSs %in% NA_strings] = 0
input$TFBSs.rank <- rank(input$TFBSs, ties.method = "max")
input$TFBSs.rank[input$TFBSs == 0] = 1
input$TFBSs.rank <- (input$TFBSs.rank/SNPs)
Results$TFBSs_raw <- input$TFBSs
Results$TFBSs.rank <- input$TFBSs.rank
feature_weights <- as.data.frame(matrix(data = 0, nrow = numRows,
    ncol = 1))
if (DAF_weight[1, 1] != 0) {
    feature_weights$DAF <- Results$DAF.rank_normalised *
        DAF_weight[1, 1]
}
else if (DAF_weight[1, 1] == 0) {
    feature_weights$DAF <- 0
}
if (Conservation_weight[1, 1] != 0) {
    feature_weights$Conservation <- Results$Conservation.rank *
        Conservation_weight[1, 1]
}
else if (Conservation_weight[1, 1] == 0) {
    feature_weights$Conservation <- 0
}
if (Histone_weight[1, 1] != 0) {
    feature_weights$Histone <- Results$Ernst.Av.new.rank *
        Histone_weight[1, 1]
}
else if (Histone_weight[1, 1] == 0) {
    feature_weights$Histone <- 0
}
if (DNase_weight[1, 1] != 0) {
    feature_weights$DNase <- Results$DNaseHS.av.rank * DNase_weight[1,1]
}
else if (DNase_weight[1, 1] == 0) {
    feature_weights$DNase <- 0
}
if (DNase_foot_weight[1, 1] != 0) {
    feature_weights$DNase_foot <- Results$DNase.foot.av.rank *
        DNase_foot_weight[1, 1]
}
else if (DNase_foot_weight[1, 1] == 0) {
    feature_weights$DNase_foot <- 0
}
if (Position_weight[1, 1] != 0) {
    feature_weights$Position <- Results$Position.rank *
        Position_weight[1, 1]
```

```
  }
  else if (Position_weight[1, 1] == 0) {
      feature_weights$Position <- 0
  }
  if (Enhancer_weight[1, 1] != 0) {
      feature_weights$Enhancer <- Results$Enhancers.rank *
          Enhancer_weight[1, 1]
  }
  else if (Enhancer_weight[1, 1] == 0) {
      feature_weights$Enhancer <- 0
  }
  if (TFBS_weight[1, 1] != 0) {
      feature_weights$TFBSs <- Results$TFBSs.rank * TFBS_weight[1,
          1]
  }
  else if (TFBS_weight[1, 1] == 0) {
      feature_weights$TFBSs <- 0
  }
  Results$Grand_Total <- feature_weights$DAF + feature_weights$Conservation +
      feature_weights$Histone + feature_weights$DNase + feature_weights$DNase_foot +
      feature_weights$Position + feature_weights$Enhancer +
      feature_weights$TFBSs
  return(Results)
  rm(headers)
  rm(numRows)
  rm(input)
  rm(Results)
  rm(SNPs)
  rm(newRanks_Ernst)
  rm(Position)
}
```

Index