Package 'GenMOSS'

December 1, 2011

Type Package

Title Application of MOSS algorithm to genome-wide association study (GWAS)

Version 1.0

Date 2009-08-12

Author Olga Vesselova, Laurent Briollais, Adrian Dobra, Helene Massam.

Maintainer <laurent@lunenfeld.ca>

Description Performs genome-wide analysis of dense SNP array data using the mode oriented stochastic search (MOSS) algorithm in a case-control design. Finds combination of best predictive SNPs associated with the response. The identified regression models are then tested by peforming cross-validation and prediction in a test set. Includes function for visualization of the obtained results. Includes preprocessing of the data from Plink format to the format required by the MOSS algorithm.

License GPL (>=2)

R topics documented:

`GenMOSS-package` *Application of MOSS algorithm to dense SNP array data*

Description

Performs genome-wide analysis of dense SNP array data using the mode oriented stochastic search (MOSS) algorithm in a case-control design. Finds combination of best predictive SNPs associated with the response. The identified regression models are then tested by peforming cross-validation and prediction in a test set. Includes function for visualization of the obtained results. Includes preprocessing of the data from Plink format to the format required by the MOSS algorithm.

Details

Package:	GenMOSS
Type:	Package
Version:	1.0
Date:	2009-08-12
License:	GPL (>=2)

System Requirements:

```
* Linux
* 64 bit machine
* MaCH software (http://www.sph.umich.edu/csg/abecasis/MACH/download/)
```

This package contains the source code of the following open-source softwares:

```
* GSL C++ library (http://www.gnu.org/software/gsl/)
* CLAPACK library (http://www.netlib.org/clapack/)
```

The package consists of three groups of files: preprocessing functions, main MOSS functions, and helper functions. The name of the the first two groups of functions begins with "pre" and "run", respectively. The preprocessing ("pre") functions are necessary for converting data from Plink format to required binary MOSS format. The main MOSS functions ("run") are needed to perform the model selection, cross-validation, data prediction, as well as plotting the results. The helper functions are available for user's convenience to check things out for their datasets. We describe basic steps for "pre" and "run" functions below.

```
Preprocessing Functions
-----------------------
```

The preprocessing step converts data from Plink format (ex2plink describes the Plink format) to the format required by the MOSS algorithm. Frequently geno data has missing values, for their imputation we use MACH software (http://www.sph.umich.edu/csg/abecasis/MACH/download/). This imputation may require to run MACH algorithm on one chromosome at a time, thus all preprocessing steps deal with multiple files: one for each chromosome. There is a total of 10 preprocessing steps that should be run in their proper order (the names of these functions begin with "pre" followed by the sequence number, followed by short description of what it does). Thus the number of intermediate files generated will be very large, for which good organization of files into directories is necessary. It is recommended to use the directory structure of the format created by pre0.dir.create.

Almost every preprocessing function has two versions: normal mode and batch mode. In normal mode, users are requested to provide input and output directory names, full names of the required files, and some other additional parameters specific to the task. Whereas the batch mode is designed to run the function for ALL the files in the input directory that satisfy a naming criterion. This batch mode saves the user from having to call the same function 22-25 times for each chromosome. The naming criterion is as follows:

```
 * prefix - The beginning string of the file name up until the chromosome number.
     Here the assumption is that when a dataset is split into 22-25 files,
     one chromosome in each, then the beginning of the file name is usually the
     same, followed by the chromosome number.
     E.x. Files with names:
        ~ "geno.data_chr1.my.ped"
        ~ "geno.data_chr2.my.ped"
        ~ "geno.data_chr3.my.ped"
          ...
        ~ "geno.data_chr22.my.ped"
     They all share the same beginning string:
        "geno.data_chr" - this is the 'prefix' for the above example.
        Note that it must be immediately followed by chromosome number.
        Also the chromosome number is expected to be a 1- or 2-digit number.
        Rename all X, Y, M, etc, to some 2-digit number.

 * key - Any string that appears in the file name. In case that the input directory
     contains files that begin with the same prefix, but should not be processed
     by the function, this parameter gives additional flexibility to filter
     such files out.
```

```
    E.x. Suppose input directory contains the following files:
       ~ "geno.data_1.CASE.ped"
       ~ "geno.data_2.CASE.ped"
         ...
       ~ "geno.data_22.CASE.ped"
       ~ "geno.data_1.CONTROL.ped"
       ~ "geno.data_2.CONTROL.ped"
         ...
       ~ "geno.data_22.CONTROL.ped"
       ~ "geno.data_1.short_try.ped"
    First note that they all have the same prefix = "geno.data_".
    Now if you wish to specify that only CASE files should be processed,
    set key="CASE" - this will ignore all CONTROL files. Also it will ignore all
    those testing files like "geno.data_1.short_try.ped", which might have
    been manually created by users for testing purposes.
    Note: this key is usually optional: if the input directory contains ONLY the
    files that need to be processed, then key can be set to an empty string "".

* ending - A string that appears at the end of the file name. Normally this does
    not have to be the filename extension, unless specifically stated. The ending
    should not include chromosome number. If preprocessing functions are run in
    their proper order, then the suggested default values for endings in the
    preprocessing functions should apply.
    Ex.
       ~ "geno.data_1.CASE.ped" - ".ped" or "d" or "CASE.ped" or "E.ped", etc.
       ~ "geno.data_2.CASE" - "CASE" or ".CASE" or "" or "E" or "SE", etc.
       ~ "geno.data_15.CONTROL.dat" - ".dat" or "t" or "CONTROL.dat", etc.

* Note: it is preferable to name files such that they have a filename extension,
   Ex.
       ~ good: "geno.data_1.CASE.ped";  bad: "geno.data_1.CASE"
       ~ good: "CGEM.chr11CONTROL.dat"; bad: "CGEM.chr11CONTROL"
  Sometimes preprocessing functions name their output functions by slightly
  modifying the name of the input file. When this is done, filename extension
  is usually removed. For example, suppose function wants to add word
  "_cleaned.txt" to the end of your filename "CGEM.chr_12CONTROL.ped"
  Resultant filename would be: "CGEM.chr_12CONTROL_cleaned.txt",
  since ".ped" will be identified as filename extension and will be lost.

  Consider what happens if you are not using filename extensions:
  then filename "CGEM.chr_12CONTROL" will be renamed as "CGEM_cleaned.txt",
  since the entire ".chr_12CONTROL" will be identified as file name extension,
  but it contains valuable chromosome information that will be lost.

  Thus always use file name extensions: ".ped", ".dat", ".txt", ".map", etc.
```

It is recommended to run the preprocessing functions in the following order:

```
 * pre0.dir.create - creates a set of empty directories d0 to d10.
 * get.file.copy - copy original format files to dir d0.
 * ex2plink - modify this function, or write something similar to
                    convert your format into Plink, this may involve
                    splitting dataset into multiple files: one per
                    chromosome; place the result into dir d1.
 * pre1.plink2mach.batch - converts Plink format to MaCH's input format,
                    which splits each chromosome into CASE and CONTROL
                    files; store result into dir d2.
 * pre2.remove.genos.batch - remove all SNPs that have too many missing
                    values, store result into dir d3.
 * pre3.call.mach.batch - imputes missing values using MaCH1, store
                    results in dir d5 (current version does not use d4).
 * pre4.combine.case.control.batch - combines CASE and CONTROL files,
                    place result into dir d6.
 * pre5.genos2numeric.batch - convert data from "A/G", "C/T", "G/G", etc
                    format to 3 levels: 1, 2, 3; store into dir d7.
 * pre6.discretize.batch - convert 3 levels: 1, 2, 3 into binary: 0, 1;
                    store result into dir d8.
 * pre7.merge.genos - merges all files across all chromosomes into one,
                    result should go into dir d9.
 * pre9.split.train.test.batch - split the full dataset into train and
                    test files; save the result into d10.

MOSS Functions
---------------
```

After the preprocessing steps are complete, continue running the main steps in the order described below. Note that main functions always require the full file name, including the directory, and their output will always go into that same directory. Thus if you start from directory d10, then all the output will end up going to dir d10 as well.

```
 * run1.moss.regression - perform MOSS search for log-linear models.
 * run2.prediction.cvv - does prediction by cross-validation using the
                    regression models identified in "run1" step.
 * run3.prediction.train.test - performs prediction on the test file, using
                    the regression models identified in "run1" step.
 * run4.save.prediction - saves a plot (as .pdf file) of the predicted values
                    and the corresponding ROC curve for the resulting
                    predictions from "run2" and/or "run3". To see the plots,
                    open the .pdf files (they should be in the same input
                    directory, d10).
 * run4.show.prediction - shows the plot, without saving it - useful only if
                    you have graphical interface to see R's plots.
 * run5.brier - computes the Brier score, its mean and standard deviation.
```

To see the functionality of preprocessing and MOSS algorithm, try running:

demo("gendemo")

Author(s)

Author: Olga Vesselova, Laurent Briollais, Adrian Dobra, Helene Massam.

Maintainer: <laurent@lunenfeld.ca>

References

Dobra, A., Briollais, L., Jarjanazi, H., Ozcelic, H. and Massam, H. (2008). Applications of the mode oriented stochastic search (MOSS) agorithm for discrete multi-way data to genomewide studies. Bayesian Modelling in Bioinformatics (D. Dey, S. Ghosh and B. Mallick, eds.), Taylor & Francis. To appear.

Examples

```
write(rbinom(200,1,0.5), file="randbinary.txt", append=FALSE, sep=" ", ncolumns=50)
run1.moss.regression("randbinary.txt")
try(system("rm randbinary.txt*"))
```

---

ex2plink *Convert example dataset to Plink format*

Description

Converts the example dataset provided with the package to PLINK format. This file is for demo purposes only. You will need to modify it to go from your file format to PLINK.

Usage

```
ex2plink(dir.file, dir.out, file.name = "genotypes_10_90.txt",
annotation.name = "Identifiers_comma.csv", out.prefix.ped = "genotypes_",
out.prefix.dat = "genos_chr")
```

Arguments

dir.file	The directory where *file.name* and *annotation.name* can be found.
dir.out	The directory to which output files should go.
file.name	The name of the file that contains the example dataset. This file should be of the following format:

```
Status        1       0       1 ...
1719214 AG      GG      AG    ...
2320341 TT      TT      TT   ...
...
```

- Tab delimited

```
  - No header
  - First row is the disease status
  - First column is the list of Markers
  - rows: geno information, no separator between alleles.
  - columns: individuals/patients/samples
```

annotation.name

The file containing SNP information about columns of *file.name*. This file should be of the following format:

```
Marker,RefSNP_ID,CHROMOSOME,CHROMOSOME_LOCATION     ...
1546,,1,2103664 ...
1996,rs1338382,1,2708522 ....
2841,"rs2887274,rs4369170",1,3504300 ...
...
```

```
  - Comma delimited (due to missing values)
  - Has a header
  - Col 1: Markers, most appear in Col 1 of file.name
  - Col 2: RefSNP_ID:
     * empty if missing
     * one SNP ID
     * two or 3 corresponding SNP IDs, in double quotes,
       comma separated, no space.
  - Col 3: chromosome number
  - Col 4: physical location
  - First 4 columns are important, other columns will be ignored.
  - rows: correspond to all available SNP IDs
```

out.prefix.ped

The beginning of output file name for pedegree files. This prefix will be used to name .ped files for each chromosome. These files will be of the following format:

```
p1    p1    0       0       1       2       C/C     N/N     T/C ...
p2    p2    0       0       1       2       T/T     A/C     G/G ...
...
```

```
  - Tab separated
  - No header
  - 6 non-SNP leading columns
  - Col 1 and Col 2: patient ID: some unique ID
  - Col 3 and Col 4: parents: mother/father: set to 0
  - Col 5: gender, default to 1 (male)
  - Col 6: disease status: 1 CONTROL and 2 CASE
  - Col 7+: geno information, slash separator between alleles.
```

out.prefix.dat

The beginning of output file name for .map file. This prefix will be used to name .map file. The file will be of the following format:

```
19 rs32453434 0 5465475
19 rs6547434 0 23534543
...

  - Space separated
  - No header
  - 4 columns:
  - Col 1: Chromosome number (Col 3 from annotation file)
  - Col 2: SNP ID or Marker if SNP is not known (Col 2 from annotatic
       or Col1 if Col2="")
  - Col 3: always 0
  - Col 4: physical locations (Col 4 from annotation file)
  - Number of rows is the number of SNPs used in the given chromosome
      (= number of SNP columns of .ped)
```

Details

This program is not part of the functionality of GenMOSS package. It is merely a demo that helps to show the conversion from one existing file format, to the desired Plink format. Users will need to write something similar to this program to convert their file format to Plink in a similar way. This function will write 2 files for each chromosome: .ped, and .map.

Author(s)

Olia Vesselova

References

Wherever genotype file is obtained from.

See Also

pre0.dir.create, pre1.plink2mach.batch, pre1.plink2mach

Examples

```
## The function is currently defined as
function (dir.file, dir.out, file.name = "genotypes_10_90.txt",
    annotation.name = "Identifiers_comma.csv", out.prefix.ped = "genotypes_",
    out.prefix.dat = "genos_chr")
{
    ## Read in the data file and annotation file
    data.file <- read.table(paste(dir.file, file.name, sep = "/"),
        sep = "\t", header = FALSE, stringsAsFactors = FALSE)
    ann.file <- read.table(paste(dir.file, annotation.name, sep = "/"),
        sep = ",", header = TRUE, stringsAsFactors = FALSE)
    # Transpose the data.file, such that columns are SNPs,
    # and 1st column becomes disease status.
    # and 1st row lists all the SNP Markers.
    data.file <- t(data.file)
    # Save the disease status and SNP Marker names separately
```

```
disease.status <- data.file[2:nrow(data.file), 1]
marker.names <- data.file[1, 2:ncol(data.file)]
# Now set data.file to be pure data
data.file <- data.file[2:nrow(data.file), 2:ncol(data.file)]
ncols <- ncol(data.file)
# ********************************************************************* #
# Iterate over all the Markers of data file.
# For each marker, find its corresponding row in annotation file
# If a marker does not exist in annotation file, print error
# (since we don't know chromosome number for it)
i <- 1
# Array that keeps at which index in annotation file Marker was found.
ids.ann <- matrix(0, ncols, 1)
# Since finding the indexes takes a long time, we can save them and
# use them instead of generating them every time.
index.name <- paste(dir.file, "indices.ann.txt", sep = "/")
if (file.exists(index.name)) {
    ids.ann <- read.table(index.name, header = FALSE, sep = " ",
        stringsAsFactors = FALSE)
    ids.ann <- unlist(ids.ann)
}
else {
    # The following code shows how to generate that file with indices.
    print(paste("Processing ", ncols, " SNPs. This is slow...",
        sep = ""))
    while (i <= ncols) {
        if (i%%1000 == 0)
            print(paste("i = ", i, sep = ""))
        # Find index of current Marker in annotation file's 1st column
        id <- match(marker.names[i], ann.file[, 1])
        # If the search failed, then we do not know anything about this marker
        if (is.na(id)) {
            print(paste("Warning: Marker ", data.file[1,
              i], " was not found in annotation file", sep = ""))
        }
        else {
            ids.ann[i] <- id
        }
        i <- i + 1
    }
    # save the indexes
    write.table(ids.ann, file = index.name, sep = " ", col.names = FALSE,
        row.names = FALSE, quote = FALSE)
}
# ********************************************************************* #
# Now ids.ann contain annotation file IDs for each marker in data.file.
# Get all the SNPs that are used and throw out the rest.
# Set ann.file to contain all info from annotation file only for used SNPs,
#   ordered in the same way as SNPs are ordered in the data file.
# Get all chromosome numbers that are used (all.chroms) and sort them.
ann.file <- ann.file[ids.ann, 1:4]
all.chroms <- unique(ann.file[, 3])
# Convert all chromosomes to numeric values (luckily for this dataset,
```

```
# all chroms are numeric, but if they were not, we would need to encode
# non-numeric values as numeric: for example "X" as 23, "Y" as 24, etc).
all.chroms.sort <- sort(as.numeric(all.chroms))
# ********************************************************************* #
# For each chromosome, create 2 files: .ped and .map of the format described above.
i <- 1
while (i <= length(all.chroms.sort)) {
    curr.chrom <- all.chroms.sort[i]
    # boolean has TRUE for all rows that correspond to current chromosome
    bool.chrom <- (ann.file[, 3] == curr.chrom)
    # Data for this chromosome, its annotation, and its markers
    chrom.data <- data.file[, bool.chrom]
    chrom.ann <- ann.file[bool.chrom, ]
    chrom.markers <- marker.names[bool.chrom]
    # Data should consist of Alleles separated by a slash,
    # whereas this dataset currently has no separator between Alleles
    chrom.data <- matrix(paste(substr(chrom.data, 1, 1),
        substr(chrom.data, 2, 2), sep = "/"), nrow = nrow(chrom.data),
        byrow = F)
    # Prepare the .ped file format:
    # Col 1 and 2: invent some unique names for data rows
    # Col 3 and 4: remain 0s
    # Col 5: set to 1, as if all are males.
    # Col 6: disease status, originally we have 0-CONTROL and 1-CASE,
    #       now we re-encode it as 1-CONTROL and 2-CASE
    ped.file <- matrix(0, nrow(chrom.data), 6)
    ped.file[, 1] <- paste("p", (1:nrow(chrom.data)), sep = "")
    ped.file[, 2] <- ped.file[, 1]
    ped.file[, 5] <- rep(1, nrow(chrom.data))
    ped.file[, 6] <- as.numeric(disease.status) + 1
    ped.file <- cbind(ped.file, chrom.data)
    # Save .ped file:
    ped.name <- paste(dir.out, "/", out.prefix.ped, curr.chrom,
        ".ped", sep = "")
    write.table(ped.file, file = ped.name, col.names = FALSE,
        row.names = FALSE, quote = FALSE, sep = "\t")
    # Prepare the .map file format:
    # Col1: chrom number
    # Col2: SNP ID, or Marker if no SNP ID
    # Col3: 0
    # Col4: physical location, Col4 from annotation
    dat.file <- matrix(0, ncol(chrom.data), 4)
    dat.file[, 1] <- rep(curr.chrom, ncol(chrom.data))
    # Iterate over all SNP IDs in annotation, extract the first SNP ID from
    # each row (since for any one entry there may be multiple SNP IDs, comma separated)
    # If there is no SNP ID for given entry, then use the Marker name
    id.splits <- strsplit(chrom.ann[, 2], ",")
    j <- 1
    while (j <= ncol(chrom.data)) {
        dat.file[j, 2] <- unlist(id.splits[j])[1]
        if (is.na(dat.file[j, 2]))
            dat.file[j, 2] <- chrom.markers[j]
        j <- j + 1
```

```
            }
            dat.file[, 4] <- chrom.ann[, 4]
            # Save the .map file
            dat.name <- paste(dir.out, "/", out.prefix.dat, curr.chrom,
                ".map", sep = "")
            write.table(dat.file, file = dat.name, col.names = FALSE,
                row.names = FALSE, quote = FALSE, sep = " ")
            print(paste("Chromosome ", curr.chrom, " written.", sep = ""))
            i <- i + 1
        }
    }

print("See the demo 'gendemo'.")
```

`genos.clean` *Removes badly predicted SNPs by MaCH*

Description

Same thing as `pre5.genos2numeric`, only leaves genotypes the way they are, without categorizing them into 3 levels. Removes all SNPs that have missing or bad values. Intended to be done after imputation, to ensure consistency. Geno values should use letters A, T, C, G if *letter.encoding*=TRUE.

Usage

```
genos.clean(file.ped, ending.ped = ".txt", dir.ped, file.dat, ending.dat = ".dat",
dir.dat = dir.ped, dir.out, num.nonsnp.col = 2, num.nonsnp.last.col = 1,
letter.encoding = TRUE, save.ids.name = "")
```

Arguments

`file.ped`	The name of file with genotypes, after imputation. Entries should be either tab or space separated.
`ending.ped`	The extension of the *file.ped*, should contain the dot '.', if file has no ending, use an empty string "". This is needed to name the output file as <file.ped>_num<ending.ped>, where *file.ped* is without ending.
`dir.ped`	The name of the directory where *file.ped* can be found.
`file.dat`	The name of .dat file. This file should be tab separated, and no header.
`ending.dat`	The extension of the *file.dat*, should contain the dot '.'. This is needed to name the output file as <file.dat>_num<ending.dat>, where *file.dat* is without ending.
`dir.dat`	The name of directory where *file.dat* can be found. Defaults to *dir.ped*.
`dir.out`	The name of output directory to which resulting files should be saved.

`num.nonsnp.col`
: The number of leading columns that do not correspond to geno values. Ex. for MaCH1 input file format there are 5 non-snp columns; for MaCH1 output format .mlgeno it is 2; for Plink it is 6.

`num.nonsnp.last.col`
: The number of last columns that do not correspond to geno values. Ex. If last column is the disease status (0s and 1s), then set this variable to 1. If 2 last columns correspond to confounding variables, set the variable to 2.

`letter.encoding`
: Flag whether or not the encoding used for Alleles is letters (A, C, T, G). If True, then does additional check for Alleles corresponding to the letters, and removes SNPs that contain values other than these 4 letters. Useful to eliminate 2s that may appear after MaCH1 imputation.

`save.ids.name`
: The file name to which patient IDs should be saved. If not empty, then will save IDs of patients into another file with this name. Useful for extracting patient ID from MaCH1 output format "ID->ID". Since dataset is generally split across many files, one chromosome each, the patient IDs should be the same across these files, thus it is enough to extract the patient ID ONCE, when running this code on the smallest chromosome. For runs on all other chromosomes, leave save.ids.name="" to save time and avoid redundant work. Could name output file as "patients.fam".

Details

This function is needed since results of MaCH might contain weird symbols (like '2' can appear instead of A, T, C, G). This function removes all the SNPs that have not been properly imputed by MaCH, making sure that there are no missing/strange values. This is only effective when *letter.encoding* = True. The reason for calling this function, and not `pre5.genos2numeric` is because you might wish to call other software packages on the fully imputed data, which will not need the data categorized into 3 levels.

Outputs the following files:

```
<file.ped>_clean<ending.ped> - in dir.out directory, the resultant file:
     the SNP columns + last columns (but no user IDs will be recorded).
<file.dat>_clean.dat - in dir.out directory, the corresponding .dat file, will
     be different from original <file.dat> if any bad SNPs get removed.
<save.ids.name> - the column of patient IDs, if save.ids.name is not empty "".
```

Value

<file.ped>_clean<ending.ped> filename - the name of the output file.

Author(s)

Olia Vesselova

See Also

`pre5.genos2numeric`, `pre5.genos2numeric.batch`, `pre3.call.mach`, `pre4.combine.case.control`

Examples

```
print("later")
```

`genos.clean.batch` *Removes badly predicted SNPs by MaCH for all files*

Description

For all files in *dir.ped*, does the same thing as `pre5.genos2numeric.batch`, only leaves genotypes the way they are, without categorizing them into 3 levels. Removes all SNPs that have missing or bad values. Intended to be done after imputation, to ensure consistency. Geno values should use letters A, T, C, G if *letter.encoding*=TRUE.

Usage

```
genos.clean.batch(dir.ped, dir.dat = dir.ped, dir.out, prefix.ped, prefix.dat,
key.ped = "", key.dat = "", ending.ped = ".txt", ending.dat = ".dat",
num.nonsnp.col = 2, num.nonsnp.last.col = 1, letter.encoding = TRUE,
save.ids.name = "patients.fam")
```

Arguments

`dir.ped`	The name of directory that contains all the .ped files.
`dir.dat`	The name of directory that contains all the .dat files.
`dir.out`	The name of output directory to which resulting files should be saved.
`prefix.ped`	The beginning of the file name for the pedegree file (up until chrom number).
`prefix.dat`	The beginning of the file name for .dat file (up until chrom number).
`key.ped`	Any keyword in the name of the pedegree file that distinguishes it from other non-pedegree files.
`key.dat`	Any keyword in the name of the .dat file that distinguishes it from others.
`ending.ped`	MUST be the filename extension of the pedegree file, including the dot ".". For example, if your file is named "CGEM_2.txt", then set this variable to ".txt"; if your file is named "CGEM_2.ped", then set this variable to ".ped"; if your file is named "CGEM_2", then set this variable to "".
`ending.dat`	MUST be the extension of the .dat file, including the dot ".".
`num.nonsnp.col`	The number of leading columns that do not correspond to geno values. Ex. for MaCH1 input file format there are 5 non-snp columns; for MaCH1 output format .mlgeno it is 2; for Plink it is 6.
`num.nonsnp.last.col`	The number of last columns that do not correspond to geno values. Ex. If last column is the disease status (0s and 1s), then set this variable to 1. If 2 last columns correspond to confounding variables, set the variable to 2.

`letter.encoding`
: Flag whether or not the ecoding used for Alleles is letters (A, C, T, G). If True, then does additional check for Alleles corresponding to the letters, and removes SNPs that contain any other symbols.

`save.ids.name`
: The name of the file to which all patient IDs should be saved.

Details

This function calls `genos.clean` for all the files in the directory, so that users do not have to call that function as many times as there are chromosomes.

For all the .ped files that start with *prefix.ped*, contain *key.ped*, and end with *ending.ped* in the directory *dir.ped*; and for similarly obtained .dat files, this function removes all the SNPs that have not been properly imputed by MaCH, making sure that there are no missing/strange values. This function is needed since results of MaCH might contain weird symbols (like '2' can appear instead of A, T, C, G). This is only effective when *letter.encoding* = True. The reason for calling this function, and not `pre5.genos2numeric` is because you might wish to call other software packages on the fully imputed data, which will not need the data categorized into 3 levels.

Outputs the following files:

```
<file.ped>_clean<ending.ped> - in dir.out directory, the resultant file:
     the SNP columns + last columns (but no user IDs will be recorded).
<file.dat>_clean.dat - in dir.out directory, the corresponding .dat file, will
     be different from original <file.dat> if any bad SNPs get removed.
<save.ids.name> - the patient IDs, if save.ids.name is not empty "".
```

Author(s)

Olia Vesselova

See Also

`pre3.call.mach`, `pre5.genos2numeric`, `pre5.genos2numeric.batch`

Examples

```
print("See demo for pre5.genos2numeric()")
```

---

get.data.dims *Obtains matrix dimensions*

Description

Obtains the number of rows and columns in a matrix that is stored in a text file. The entries in the file should be either space or tab delimited. No missing values.

Usage

```
get.data.dims(genome.file)
```

Arguments

genome.file	Name of any file that contains a matrix of values in it, separated by either spaces or tabs.

Value

out$nrows	Number of rows in the matrix
out$ncols	Number of columns in the matrix

Note

Uses LINUX's wc functionality.

Author(s)

Olga Vesselova

See Also

run1.moss.regression, run2.prediction.cvv, run3.prediction.train.test

Examples

```
write(rbinom(200,1,0.5), file="randbinary.txt", append=FALSE, sep=" ", ncolumns=50)
get.data.dims("randbinary.txt")
try(system("rm randbinary.txt*"))
```

---

get.file.copy *Copies files from one directory to another*

Description

From given directory *dir.in*, copies files into *dir.out*. Either list of file names in *fname*, or all files from *dir.in* that start from given *prefix* and end with *ending* and contain keyword *key*.

Usage

```
get.file.copy(dir.in, dir.out, fname = "", prefix = "", key = "", ending = "",
verbal = TRUE)
```

Arguments

dir.in	The name of directory which contains files that need to be copied.
dir.out	The name of directory to which files should be copied.
fname	The list of file names (should be empty if you want it to find files itself given specifications of *prefix*, *key* and *ending*).
prefix	The beginning of the file names that need to be copied.
key	Any keyword that uniquely distinguishes the files from others.
ending	The ending of the file names that need to be copied.
verbal	Flag whether or not to print error messages if files with *prefix*, *key* and *ending* could not be found. This flag only matters if *fname*="".

Details

This function can be used in two ways:

1. Either user provides a list of filenames that need to be copied over to *dir.out* directory, in which case all *prefix*, *key* and *ending* will be ignored.

2. Or *fname*="" and some of the 3 parameters *prefix*, *key* and *ending* are set. In which case the program will search for files in *dir.in* that fulfill the specifications.

This function is basically file.copy, only it allows to pass in a list instead of a single file, and takes input in format that is similar to all other preprocessing functions in GenMOSS.

Author(s)

Olia Vesselova

See Also

pre0.dir.create

Examples

```
print("See the demo 'gendemo'.")
```

---

pre0.dir.create *Generate working subdirectory structure*

Description

Function to help create the recommended subdirectory structure for the pre-processing. In dir.out a directory with name *out.name* will be created. Inside of this *out.name* directory will be a set of subdirectories, whose names will begin with *prefix.dir*, followed by a number, followed by short description of what the folder is designed to contain.

Usage

```
pre0.dir.create(dir.out = ".", out.name = "newdata", prefix.dir = "d")
```

Arguments

dir.out	The name of directory to which new folder *out.name* should be saved.
out.name	The name of the new working directory.
prefix.dir	The start of the name of all subdirectories that will be located inside *out.name* folder.

Details

The subdirectory structure is designed to easily work with preprocessing functions of GenMOSS. Since GenMOSS preprocessing steps need to be performed in a fixed order, and there are several files per chromosome at each step, very good organization of these files is necessary to know what files have come from where and which .dat, .ped, and .fam files correspond. This function creates the directory and subdirectory structure, and it also returns the names of all the subdirectories, which can be easily used as out$d0 to out$d11. See the demo "gendemo" that shows how to effortlessly use this return variable when calling all the pre-processing steps.

Value

out$d0	The name of subdirectory into which original data should be placed.
out$d1	The name of subdirectory into which data converted into Plink format should go. This can be done by function similar to ex2plink.
out$d2	The name of subdirectory into which data converted into MaCH input format should go. This can be done by pre1.plink2mach.batch.
out$d3	The name of subdirectory into which data with removed empty SNPs should go. This can be done by pre2.remove.genos.batch.
out$d4	The name of subdirectory into which reference files needed for MaCH1 can be downloaded.
out$d5	The name of subdirectory into which output of MaCH1 should go. This can be done by pre3.call.mach.batch.
out$d6	The name of subdirectory into which combined CASE and CONTROL files should go. This can be done by pre4.combine.case.control.batch.
out$d7	The name of subdirectory into which data converted to numeric 3 levels should go. This can be done by pre5.genos2numeric.batch.
out$d8	The name of subdirectory into which data converted to binary should go. This can be done by pre6.discretize.batch.
out$d9	The name of subdirectory into which binary data merged across all chromosomes should go. This can be done by pre7.merge.genos.
out$d10	The name of subdirectory into which merged data split into train and test sets should go. Also all the main GenMOSS computation would go into this subdirectory. The train-test split can be done by pre9.split.train.test.batch.

`out$d11` The name of subdirectory into which desired subsets of the data should go. Also all the main GenMOSS computation for each subset would go into this subdirectory. The train-test split can be done by `pre9.split.train.test.batch`.

Author(s)

Olia Vesselova

See Also

`ex2plink`, `pre1.plink2mach.batch`, `pre2.remove.genos.batch`, `pre3.call.mach.batch`, `pre4.combine.case.control.batch`, `pre5.genos2numeric.batch`, `pre6.discretize.batch`, `pre7.merge.genos`, `pre9.split.train.test.batch`

Examples

```
print("See the demo 'gendemo'.")
```

---

pre1.plink2mach *Convert Plink to MaCH input format*

Description

Provided with Plink-format files *file.ped* and *file.map* in *dir.in*, this function re-formats it into MACH pedigree (*file.ped*) and data (*file.dat*) file formats, and saves the reformatted files in *dir.out*.

Usage

```
pre1.plink2mach(file.ped = "", file.map = "", dir.in, dir.out)
```

Arguments

`file.ped` The name of the pedegree file. This file should be in Plink format:

```
p1    p1    0       0       1       2       C/C     N/N     T/C ...
p2    p2    0       0       1       2       T/T     A/C     G/G ...
...
```

- Tab separated
- No header
- 6 non-SNP leading columns
- Col 1 and Col 2: patient ID: some unique ID
- Col 3 and Col 4: parents: mother/father: can be set to 0
- Col 5: gender, 1 - male, and 2 - female
- Col 6: disease status: 1 CONTROL and 2 CASE
- Col 7+: geno information, slash separator between alleles.

`file.map` The name of the .map input file. This file should contain names of SNPs in the following format:

```
19 rs32453434 0 5465475
19 rs6547434 0 23534543
...

  - Space separated
  - No header
  - 4 columns:
  - Col 1: Chromosome number
  - Col 2: SNP ID or any other marker for SNP
  - Col 3: genetic distance (can be set to 0)
  - Col 4: physical locations (can be set to 0)
  - Number of rows is the number of SNPs used in the given
      chromosome. (= number of SNP columns of .ped)
```

`dir.in` The directory where *file.ped* and *file.map* can be found.

`dir.out` The directory to which output .ped and .dat files should go.

Details

This function converts from Plink to MaCH input format. There is no need to specify both *file.ped* and *file.map*; so one of them can be an empty string (""), in which case, this file will not be processed. So that you can use this function to do ONLY PED files but not map, and vice versa.

Note

Note: the function does NOT change unknown Allele values from "0" to "N", as MACH program can use either. Does NOT recode gender to "M" and "F", since MaCH1 doesn't care, but further file processing interprets "F" as "FALSE".

Author(s)

Olia Vesselova

See Also

`pre1.plink2mach.batch`, `pre0.dir.create`, `pre2.remove.genos`, `pre2.remove.genos.batch`

Examples

```
print("See the demo 'gendemo'.")
```

pre1.plink2mach.batch

Convert Plink to MaCH input format for all files

Description

For all files in *dir.in* directory that end with *ending.ped* and contain keyword *key.ped*, and all files ending with *ending.map* and contain keyword *key.map*, runs the converter pre1.plink2mach. This will re-format all the files from Plink format to MaCH input format.

Usage

```
pre1.plink2mach.batch(dir.in, dir.out, ending.ped = ".ped", ending.map = ".map",
key.ped = "", key.map = "")
```

Arguments

dir.in	The name of directory where all the files with *ending.ped*, *key.ped*, *ending.map* and *key.map* specifications are located.
dir.out	The name of directory to which output files .ped and .dat should be saved to.
ending.ped	The ending of the filenames that contain pedegree data in Plink format. See format of .ped file in pre1.plink2mach.
ending.map	The ending of the filenames that contain SNP ID information. See format of .map file in pre1.plink2mach.
key.ped	Any keyword in the Plink pedegree file names to uniquely distinguish them from other files in the *dir.in* directory.
key.map	Any keyword in the .map file names to uniquely distinguish them from other files in the *dir.in* directory.

Details

The input file formats of .ped and .map files are described in pre1.plink2mach.

Author(s)

Olia Vesselova

See Also

pre1.plink2mach, pre0.dir.create, pre2.remove.genos, pre2.remove.genos.batch

Examples

```
print("See the demo 'gendemo'.")
```

pre2.remove.genos *Remove genos with many empty values*

Description

Remove columns (genos) that have too many missing values. All genos that have more than *perc.snp* values missing in both *case.ped* AND *control.ped* files will be removed.

Usage

```
pre2.remove.genos(file.dat, case.ped, control.ped, dir.dat, dir.out,
dir.warning = dir.out, perc.snp = 10, perc.patient = 20, empty = "0/0",
num.nonsnp.col = 5)
```

Arguments

`file.dat` The name of data file as required for MaCH1. The file should be of the format:

```
M SNP1
M SNP2
```

- Space separated
- No header
- Column 1: consists of "M"
- Column 2: character SNP names

`case.ped` The name of pedegree data file that contains CASEs in MaCH input format.

`control.ped` The name of pedegree data file that contains CONTROLs in MaCH input format.

`dir.dat` The directory name where *file.dat* and *file.ped* can be found.

`dir.out` The directory name to which output files should be saved.

`dir.warning` The directory name to which warnings about patients with too many missing SNPs should go. Defaults to the same place as *dir.out*.

`perc.snp` The percentage (0-100 percent) of maximum empty values allowed for each geno (column). All genos that have more empty values than this threshold will be removed.

`perc.patient` The percentage (0-100 percent) of empty values allowed for each patient (row). Names of all patients who end up having more empty values than this threshold will be recorded in the warnings file.

`empty` The representation of a missing SNP value in the file ("0 0", "0/0", "1/1", "N N", etc).

`num.nonsnp.col` The number of leading columns in the .ped files that do not contain SNP values. The first columns of the file represent non-SNP values (like patient ID, gender, etc). For MaCH1 input format, the *num.nonsnp.col=5*, for PLINK it is 6 (due to extra disease status column).

Details

Remove columns (genos) that have too many missing values. All genos that have more than *perc.snp* values missing in both *case.ped* AND *control.ped* files will be removed.

All patients that have more than *perc.patient* values missing will have their IDs written into "warning.<case.ped>.txt" files. Output will be two clean versions of *case.ped* and *control.ped* files in *dir.out* directory, and optionally the warning files in *dir.warning* directory.

The following files will be saved after the program is run:

```
- <file.dat>.removed.dat - the .dat file containing only the SNPs that were not
      removed, will be placed in dir.out directory
- <case.ped>.removed.ped - the CASE .ped file without columns that contain too
      many missing values based on the thresholds perc.snp; in dir.out directory
- <control.ped>.removed.ped - the CONTROL .ped file without columns that contain
      too many missing values based on the thresholds perc.snp;
      in dir.out directory

- warning.<case.ped>.txt - file containing warning messages about patients that
      have too many SNPs missing (based on perc.patients) in CASE.ped file,
      after the removal of bad SNPs.
- warning.<control.ped>.txt - similar to warning.<case.ped>.txt, only for
      CONTROL file.
```

Author(s)

Olia Vesselova

See Also

pre1.plink2mach, pre1.plink2mach.batch, pre2.remove.genos.batch, pre3.call.mach, pre3.call.mach.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre2.remove.genos.batch

Remove genos with many empty values for all files

Description

For all specified files, remove columns (genos) that have too many missing values. This program will automatically match CASEs and CONTROLs and their corresponding .dat files based on the specifications of prefixes, keys, and endings.

Usage

```
pre2.remove.genos.batch(dir.dat, dir.ped = dir.dat, dir.out,
dir.warning = dir.out, perc.snp = 10, perc.patient = 20, empty = "0/0",
num.nonsnp.col = 5, prefix.dat, prefix.case, prefix.control, key.dat = "",
key.case = "CASE", key.control = "CONTROL", ending.dat = ".dat",
ending.case = ".ped", ending.control = ".ped")
```

Arguments

`dir.dat`	The directory name where all .dat files can be found.
`dir.ped`	The directory name where all .ped CASE and CONTROL files can be found. Defaults to same place as *dir.dat*
`dir.out`	The directory name to which output files should be saved.
`dir.warning`	The directory name to which warnings about patients with too many missing SNPs should go. Defaults to the same place as *dir.out*.
`perc.snp`	The percentage (0-100 percent) of maximum empty values allowed for each geno (column). All genos that have more empty values than this threshold will be removed.
`perc.patient`	The percentage (0-100 percent) of empty values allowed for each patient (row). Names of all patients who end up having more empty values than this threshold will be recorded in the warnings file.
`empty`	The representation of a missing SNP value in the file ("0 0", "0/0", "1/1", "N N", etc).
`num.nonsnp.col`	The number of leading columns in the .ped files that do not contain SNP values. The first columns of the file represent non-SNP values (like patient ID, gender, etc). For MaCH1 input format, the *num.nonsnp.col=5*, for PLINK it is 6 (due to extra disease status column).
`prefix.dat`	The beginning of the file name for the .dat file (up until chrom number).
`prefix.case`	The beginning of the file name for the CASE pedegree file (up until chrom number).
`prefix.control`	The beginning of the file name for the CONTROL pedegree file (up until chrom number).
`key.dat`	Any keyword in the name of the pedegree file that distinguishes it from other files.
`key.case`	Any keyword in the name of the CASE pedegree file that distinguishes it from other non-pedegree non-CASE files.
`key.control`	Any keyword in the name of the CONTROL pedegree file that distinguishes it from other non-pedegree non-CONTROL files.
`ending.dat`	The ending of the .dat filenames.
`ending.case`	The ending of the CASE pedegree filenames.
`ending.control`	The ending of the CONTROL pedegree filenames.

Details

Removes SNPs that contain more than *perc.snp* empty geno values, from all the corresponding CASE and CONTROL .ped and .dat files in directory *dir.dat*. If a .ped file for some chromosome is split into several files, these files will be concatenated into one file for that chromosome, in alphabetical order. Those chromosomes that have files that satisfy the (prefix, key, ending) selection criterion but do NOT have complete set of 3 files (CASE, CONTROL, and .dat), will NOT be processed.

Author(s)

Olia Vesselova

See Also

pre1.plink2mach, pre1.plink2mach.batch, pre2.remove.genos, pre3.call.mach, pre3.call.mach.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre3.call.mach *Call MaCH imputation with and without Hapmap*

Description

Calls MACH1 program on *file.ped* and *file.dat*. MaCH1 can be run in 2 different ways: 1. with Hapmap, and 2. without Hapmap. NOTE: In this implementation, do NOT run "with Hapmap".

This program first runs MaCH1 on *file.ped* with Hapmap to fill in missing values for those SNPs that exist in the reference file; and then MaCH1 is run on the result without Hapmap to fill in all the remaining missing values. If no reference files *ref.phase* and *ref.legend* are provided, then the program runs MaCH1 without Hapmap only. To clean up any weird MaCH output, use genos.clean or pre5.genos2numeric.

Usage

```
pre3.call.mach(file.dat, file.ped, dir.file, ref.phase = "", ref.legend = "",
dir.ref = "", dir.out, out.prefix = "result", chrom.num = "", num.iters = 2,
num.subjects = 200, step2.subjects = 50, empty = "0/0", resample = FALSE,
mach.loc = "/software/mach1")
```

Arguments

`file.dat` The name of data file as required for MaCH1. The file should be of the format:

```
M SNP1
M SNP2

- Space separated
- No header
- Column 1: consists of "M"
- Column 2: character SNP names
```

`file.ped` The name of pedegree data file in MaCH1 input format.

```
p1    p1    0       0       1       C/C     N/N     T/C ...
p2    p2    0       0       1       T/T     A/C     G/G ...
...

  - Tab separated
  - Alleles are separated by slash '/' (IMPORTANT!)
  - No header
  - 5 non-SNP leading columns
  - Col 1: sample/patient ID: some unique ID
  - Col 2: family ID: can be same as patient ID
  - Col 3 and Col 4: parents: mother/father: can all be 0
  - Col 5: gender, 1-male, 2-female
  - Col 6+: geno information, slash separator between alleles.
```

`dir.file` The name of directory where *file.ped* can be found.

`ref.phase` The name of the reference file, must have no missing values, can be obtained from websites like: http://hapmap.ncbi.nlm.nih.gov/downloads/phasing/2007-08_rel22/phased/ or similar/updated versions. No zip. Must be a normal and readable by R file.

`ref.legend` The name of legend file for *file.phase*, obtained from same website. No zip.

`dir.ref` The name of directory where *ref.phase* and *ref.legend* can be found.

`dir.out` The name of directory where MaCH1 output should go.

`out.prefix` The prefix for naming output files that MaCH1 should use. If *num.subjects* > 0 then the *num.subjects* will be appended to the prefix name.

`chrom.num` The optional string denoting the chromosome number, for better naming of intermediate files.

`num.iters` The number of iterations MaCH1 should make in its first step to estimate its model parameters. The same number will be used for parameter estimation when using Hapmap and when NO Hapmap is used.

`num.subjects` How many individuals from the sample should be used for model building by the first step of MaCH1. The random subset of inidividuals will be extracted by this program. Recommended number of subjects is 200-500. Value <= 0 corresponds to using ALL the subjects in the dataset.

`step2.subjects`
: How many individuals should be processed at a time during the second step of MaCH computation. Value <= 0 will use ALL the subjects in the dataset. This variable is important to reduce exponential computation time required by MaCH when number of individuals is too large. However if this number is too low, the second step of MaCH might not get enough samples, thus making weird prediction of '2' instead of an Allele value. To reduce the number of '2's, try to set step2.subjects to a larger value. To remove all SNPs that have a 2 predicted for any of its entries, use `genos.clean` or `pre5.genos2numeric`.

`empty`
: The way a missing/empty entry of SNP is represented in *file.ped*.

`resample`
: Whether or not to overwrite the existing file containing the *num.subjects* entries produced by previous runs of this algorithm with same *file.dat*, *file.ped* and *num.subjects* parameters. By default, if the subjects have been sampled before, they are re-used.

`mach.loc`
: The location directory where "mach" executable can be found.

Details

This program first runs MaCH1 on *file.ped* with Hapmap to fill in missing values for those SNPs that exist in the reference file; and then MaCH1 is run on the result without Hapmap to fill in all the remaining missing values. If no reference files *ref.phase* and *ref.legend* are provided, then the program runs MaCH1 without Hapmap only.

It is recommended to avoid using Hapmap functionality in this implementation.

The MaCH1 algorithm requires 2 steps to be performed. The first step of MaCH1 will be run on num.subjects randomly chosen from the set. The file with randomly chosen individuals will be saved as *file.ped.<num.subjects>.ped* in *dir.file* directory. If the file already exists for this num.subjects, the old file will be used if *resample*=F. If *resample*=T then old files will be ignored, and new sampling will take place. The step1 of MaCH will only be run if *resample*=T, or if the files that MaCH1 produces do not exist yet. Thus if step1 runs well, but step2 crashes, re-calling this function will not waste time on re-running step1 over again.

The second step without Hapmap takes exponentially long wrt number of subjects processed. Thus the second step will be run on bunches of subjects, *step2.subjects* at a time.

A subdirectory structure for debugging will be formed in *dir.out*, the directory will be named 'working'.

Two output files will be produced in *dir.out*: the .ped file that will not have any missing values, will be named <*out.prefix*><*chrom.num*>.mlgeno, and a .dat file (same as before).

Note

Since instead of filling in missing values, MaCH1 is re-predicting ALL the values in the dataset, the Hapmap functionality is not desirable. Thus avoid using Hapmap reference files.

Also, MaCH prediction is not always valid, as it may contain Allele of value '2' (when only A, C, T, G are used). Programs `pre5.genos2numeric` and `genos.clean` help to remove those.

Author(s)

Olia Vesselova

References

MaCH website: http://www.sph.umich.edu/csg/abecasis/MACH/download/

See Also

pre2.remove.genos, pre2.remove.genos.batch, pre3.call.mach.batch, pre4.combine.case.co
pre4.combine.case.control.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre3.call.mach.batch

Call MaCH imputation with and without Hapmap

Description

This is the same program as pre3.call.mach, only it provides an easier way to set function input parameters. This is the only .batch function that does NOT run on all files. Since MaCH computation on each chromosome takes too long, it is faster to process chromosomes in parallel, rather than sequentially. This function imputes all missing values, for details, see pre3.call.mach. NOTE: In this implementation, do NOT run "with Hapmap" - so do NOT provide phases and legend files.

Usage

```
pre3.call.mach.batch(dir.file, dir.ref = "", dir.out, prefix.dat, prefix.ped,
prefix.phase = "", prefix.legend = prefix.phase, prefix.out = "result",
key.dat = "", key.ped = "", key.phase = "", key.legend = "", ending.dat = ".dat",
ending.ped = ".ped", ending.phase = ".phase", ending.legend = "legend.txt",
chrom.num, num.iters = 2, num.subjects = 200, step2.subjects = 50, empty = "0/0",
resample = FALSE, mach.loc = "/software/mach1")
```

Arguments

dir.file	The name of directory where .ped and .dat files can be found. The format of these files is described in pre3.call.mach
dir.ref	The name of directory where .phase and .legend files have been downloaded to.
dir.out	The name of directory to which output files should go.
prefix.dat	The beginning of the file name for the .dat file (up until chrom number).
prefix.ped	The beginning of the file name for the .ped pedegree file (up until chrom number).

`prefix.phase` The beginning of the file name for the phase file (up until chrom number). This file can be obtained from websites like: http://hapmap.ncbi.nlm.nih.gov/downloads/phasing/2007-08_rel22/phased/ or similar/updated versions. No zip. Must be a normal and readable by R file.

`prefix.legend` The beginning of the file name for the legend file (up until chrom number). This file can be obtained from same website as phase file. No zip.

`prefix.out` The prefix for naming output files that MaCH1 should use. If *num.subjects* > 0 then the *num.subjects* will be appended to the prefix name.

`key.dat` Any keyword in the name of the .dat file that distinguishes it from other files.

`key.ped` Any keyword in the name of the pedegree file that distinguishes it from other files.

`key.phase` Any keyword in the name of the phase file that distinguishes it from other files.

`key.legend` Any keyword in the name of the legend file that distinguishes it from other files.

`ending.dat` The ending of the .dat filename.

`ending.ped` The ending of the pedegree filename.

`ending.phase` The ending of the phase filename.

`ending.legend` The ending of the legend filename.

`chrom.num` The chromosome number for which processing should be done.

`num.iters` The how many iterations MaCH1 should make in its first step to estimate its model parameters.

`num.subjects` How many individuals from the sample should be used for model building by the first step of MaCH1. The random subset of inidividuals will be extracted by this program. Recommended number of subjects is 200-500. Value <= 0 corresponds to using ALL the subjects in the dataset.

`step2.subjects` How many individuals should be processed at a time during the second step of MaCH computation. Value <= 0 will use ALL the subjects in the dataset. This variable is important to reduce exponential computation time required by MaCH when number of individuals is too large.

`empty` The way a missing/empty entry of SNP is represented in pedegree file.

`resample` Whether or not to overwrite the existing file containing the *num.subjects* entries produced by previous runs of this algorithm with same .dat, .ped, and *num.subjects* parameters. By default, if the subjects have been sampled before, they are re-used.

`mach.loc` The location directory where "mach" executable can be found.

Details

This function imputes all missing values in the data. See `pre3.call.mach` for details. This is the same program as `pre3.call.mach`, only it provides an easier way to set function input parameters. Recall that `pre3.call.mach` function requires users to specify names of .ped, .dat, .phase, and .legend for each chromosome - these files normally would have exactly same names

across all chromosomes, and would only differ by the chromosome number. Thus after running pre3.call.mach, for chromosome 1, and in order to run next chromosome (say, chrom "2"), user would need to change this chromosome number in 4 places: from "1" to "2" in .ped, .dat, .phase, and .legend. This function allows user to just change one variable *chrom.num*, from "1" to "2", and all the other files will be obtained automatically.

This is the only .batch function that does NOT run on all files. Since MaCH computation on each chromosome takes too long, it is faster to process chromosomes in parallel, rather than sequentially. Thus if your dataset is large, then it is recommended to run this function on different computers/nodes for different chromosomes.

Note

In this current version, avoid using Hapmap. So do NOT provide reference and legend files.

Author(s)

Olia Vesselova

References

MaCH website: http://www.sph.umich.edu/csg/abecasis/MACH/download/

See Also

pre2.remove.genos, pre2.remove.genos.batch, pre3.call.mach, pre4.combine.case.control, pre4.combine.case.control.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre4.combine.case.control

Combine CASE and CONTROL files

Description

Combines CASE and CONTROL files into one file, and appends disease status as the last column. The disease status is encoded as 1 for CASE and 0 for CONTROL.

Usage

```
pre4.combine.case.control(case.file, control.file, dir.file, name.out,
dir.out = dir.file, separ = " ")
```

Arguments

case.file	The name of the CASE file.
control.file	The name of the CONTROL file.
dir.file	The name of directory where CASE and CONTROL input files can be found.
name.out	The desired name for the output file.
dir.out	The name of directory to which output file should be written.
separ	The separator used in the CASE and CONTROL input files.

Details

The function combines CASE and CONTROL together, attaching disease status as the last column: 1 for CASE and 0 for CONTROL. There will be two output files:

- <dir.out>/<name.out> - the file containing both CASE and CONTROL values, with the disease status as the last column.
- <dir.out>/<all.dat> - also will copy over ALL the files ending with ".dat" that exist in \code{*dir.file*}.

Author(s)

Olia Vesselova

See Also

pre3.call.mach, pre3.call.mach.batch, pre4.combine.case.control.batch, pre5.genos2numeric, pre5.genos2numeric.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre4.combine.case.control.batch

Combine CASE and CONTROL files for all files

Description

For each pair of CASE and CONTROL files, combine them into one file. Last column of each output file will contain the disease status. The disease status is encoded as 1 for CASE and 0 for CONTROL.

Usage

```
pre4.combine.case.control.batch(dir.file, dir.out = dir.file, prefix.case,
prefix.control, prefix.out, key.case = "", key.control = "",
ending.case = ".mlgeno", ending.control = ".mlgeno", separ = " ")
```

Arguments

dir.file	The name of directory where CASE and CONTROL files can be found.
dir.out	The name of directory to which output file should be written.
prefix.case	The beginning of the file name for the CASE file (up until chrom number).
prefix.control	The beginning of the file name for the CONTROL file (up until chrom number).
prefix.out	The beginning of the file name for the output file (up until chrom number).
key.case	Any keyword in the name of the CASE file that distinguishes it from other files.
key.control	Any keyword in the name of the CONTROL file that distinguishes it from other files.
ending.case	The ending of the CASE filename.
ending.control	The ending of the CONTROL filename.
separ	The separator used in the CASE and CONTROL input files.

Details

The function combines CASE and CONTROL together, attaching disease status as the last column: 1 for CASE and 0 for CONTROL. There will be two output files for each pair of CASE and CONTROL:

- <dir.out>/<prefix.out><chrom.num><ending.case> - the file containing both CASE and CONTROL values, with the disease status as the last column.
- <dir.out>/<all.dat> - also will copy over ALL the files ending with ".dat" that exist in \code{*dir.file*}.

Author(s)

Olia Vesselova

See Also

pre3.call.mach, pre3.call.mach.batch, pre4.combine.case.control, pre5.genos2numeric, pre5.genos2numeric.batch

Examples

```
print("See the demo 'gendemo'.")
```

`pre5.genos2numeric` *Categorize genotype data into 3 levels*

Description

Categorizes genotype data into 3 levels, 1, 2, 3. Genos with two different Alleles are encoded as "2". Other genotypes are encoded as "1" or "3", where most frequent geno is "1". No missing values allowed, must be done after imputation. Geno values should use letters A, T, C, G if letter.encoding=TRUE. Also can work as a check for weird imputed values.

Usage

```
pre5.genos2numeric(file.ped, dir.ped, file.dat, dir.dat = dir.ped, dir.out,
num.nonsnp.col = 2, num.nonsnp.last.col = 1, letter.encoding = TRUE,
ped.has.ext = TRUE, dat.has.ext = TRUE, remove.bad.genos = FALSE,
save.ids.name = "")
```

Arguments

`file.ped`	The name of file with genotypes, after imputation.
`dir.ped`	The name of directory where *file.ped* can be found.
`file.dat`	The .dat file, should be tab separated, and no header.
`dir.dat`	The name of directory where *file.dat* can be found. Defaults to *dir.ped*.
`dir.out`	The name of output directory to which resulting file should be saved. The file will be named "Num.<file.ped>".
`num.nonsnp.col`	The number of leading columns in the .ped files that do not contain SNP values. The first columns of the file represent non-SNP values (like patient ID, gender, etc). For MaCH1 input format, the *num.nonsnp.col=5*, for PLINK it is 6 (due to extra disease status column).
`num.nonsnp.last.col`	The number of last columns that do not correspond to geno values. Ex. If last column is the disease status (0s and 1s), then set this variable to 1. If 2 last columns correspond to confounding variables, set the variable to 2.
`letter.encoding`	Flag whether or not the ecoding used for Alleles is letters (A, C, T, G). If True, then does additional check for Alleles corresponding to the letters, and prints out warning messages if other symbols appear instead.
`ped.has.ext`	Flag whether or not *file.ped* name has a filename extension (ex. ".ped", ".txt"). This is necessary for naming the output file.
`dat.has.ext`	Flag whether or not file.dat name has a filename extension (ex. ".dat", ".txt").
`remove.bad.genos`	Flag whether or not you want to remove a geno if at least one of its values is not valid (ex. "2" when only letters are expected, or "NA", etc). Warning: set

this to TRUE only if the CASE and CONTROLs have been merged into the *file.ped*, (otherwise we do not want to remove some SNPs from CASE but not from CONTROL and generate two different .dat files).

`save.ids.name`

The file name to which patient IDs should be saved. If not empty, then will save IDs of patients into another file with this name. Since dataset is generally split across many files, one chromosome each, the patient IDs should be the same across these files, thus it is enough to extract the patient ID ONCE, when running this code on the smallest chromosome. For runs on all other chromosomes, leave save.ids.name="" to save time and avoid redundant work. Could name output file as "patients.fam".

Details

Categorizes genotype data into 3 levels, 1, 2, 3. Genos with two different Alleles are encoded as "2". Other genotypes are encoded as "1" or "3", where most frequent geno is "1". No missing values allowed, must be done after imputation. Geno values should use letters A, T, C, G if letter.encoding=TRUE. Also can work as a check for weird imputed values. For example, it is possible that an Allele is predicted by MaCH1 having value "2" (instead of A, T, C, or G) - it is best to remove SNPs that contain these weirdly imputed values.

The following files will be produced:

- <file.ped>_num<ending.ped> - in \code{*dir.out*} directory, the resultant binary file: the SNP columns + last columns (but no user IDs will be recorded), where <ending.ped> is the filename extension of file.ped.
- <file.dat>_num.dat - in dir.out directory, the corresponding .dat file, will be different from original <file.dat> if remove.bad.genos=TRUE.
- <save.ids.name> - the patient IDs, if save.ids.name is not empty "".

Value

<file.ped>_num<ending.ped> filename - the name of the output file.

Note

Note: in case of any bad values in the *file.ped* (ex. "NA", "0/0", "0", "1 1", etc), the output file Num_<file.ped> will still be produced, with '2' encoded by default in the place of bad input values, if *remove.bad.genos*=FALSE. Warning messages will be printed. If *remove.bad.genos*=TRUE, then these SNPs will be entirely removed, along with their names in the .dat file.

Author(s)

Olia Vesselova

See Also

`pre4.combine.case.control`, `pre4.combine.case.control.batch`, `pre5.genos2numeric.batch`, `pre6.discretize`, `pre6.discretize.batch`

Examples

```
print("See the demo 'gendemo'.")
```

`pre5.genos2numeric.batch`
Categorize genotype data into 3 levels for each file

Description

For each .ped file in *dir.ped*, categorizes genotype data into 3 levels, 1, 2, 3. Genos with two different Alleles are encoded as "2". Other genotypes are encoded as "1" or "3", where most frequent geno is "1". No missing values allowed, must be done after imputation. Geno values should use letters A, T, C, G if letter.encoding=TRUE.

Usage

```
pre5.genos2numeric.batch(dir.ped, dir.dat = dir.ped, dir.out, prefix.ped,
prefix.dat, key.ped = "", key.dat = "", ending.ped = ".txt", ending.dat = ".dat",
num.nonsnp.col = 2, num.nonsnp.last.col = 1, letter.encoding = TRUE,
ped.has.ext = TRUE, dat.has.ext = TRUE, remove.bad.genos = FALSE,
save.ids.name = "patients.fam")
```

Arguments

`dir.ped`	The name of directory where .ped files can be found.
`dir.dat`	The name of directory where .dat files can be found.
`dir.out`	The name of directory to which output files should go.
`prefix.ped`	The beginning of the file name for the .ped pedegree file (up until chrom number).
`prefix.dat`	The beginning of the file name for the .dat file (up until chrom number).
`key.ped`	Any keyword in the name of the pedegree file that distinguishes it from other files.
`key.dat`	Any keyword in the name of the .dat file that distinguishes it from other files.
`ending.ped`	The ending of the pedegree filename.
`ending.dat`	The ending of the .dat filename.
`num.nonsnp.col`	The number of leading columns in the .ped files that do not contain SNP values. The first columns of the file represent non-SNP values (like patient ID, gender, etc). For MaCH1 input format, the *num.nonsnp.col=5*, for PLINK it is 6 (due to extra disease status column).
`num.nonsnp.last.col`	The number of last columns that do not correspond to geno values. Ex. If last column is the disease status (0s and 1s), then set this variable to 1. If 2 last columns correspond to confounding variables, set the variable to 2.

`letter.encoding`
: Flag whether or not the ecoding used for Alleles is letters (A, C, T, G). If True, then does additional check for Alleles corresponding to the letters, and prints out warning messages if other symbols appear instead.

`ped.has.ext`
: Flag whether or not *file.ped* name has a filename extension (ex. ".ped", ".txt"). This is necessary for naming the output file.

`dat.has.ext`
: Flag whether or not file.dat name has a filename extension (ex. ".dat", ".txt").

`remove.bad.genos`
: Flag whether or not you want to remove a geno if at least one of its values is not valid (ex. "2" when only letters are expected, or "NA", etc). Warning: set this to TRUE only if the CASE and CONTROLs have been merged into the *file.ped*, (otherwise we do not want to remove some SNPs from CASE but not from CONTROL and generate two different .dat files).

`save.ids.name`
: The file name to which patient IDs should be saved. If not empty, then will save IDs of patients into another file with this name. Since dataset is generally split across many files, one chromosome each, the patient IDs should be the same across these files, thus it is enough to extract the patient ID ONCE, when running this code on the smallest chromosome. For runs on all other chromosomes, leave save.ids.name="" to save time and avoid redundant work. Could name output file as "patients.fam".

Details

For every pair of .dat and .ped files, categorizes genotype data into 3 levels, 1, 2, 3. Genos with two different Alleles are encoded as "2". Other genotypes are encoded as "1" or "3", where most frequent geno is "1". No missing values allowed, must be done after imputation. Geno values should use letters A, T, C, G if letter.encoding=TRUE. Also can work as a check for weird imputed values. For example, it is possible that an Allele is predicted by MaCH1 having value "2" (instead of A, T, C, or G) - it is best to remove SNPs that contain these weirdly imputed values.

The following files will be produced for each chromosome in the directory *dir.ped*:

- <file.ped>_num<ending.ped> - in \code{*dir.out*} directory, the resultant binary file: the SNP columns + last columns (but no user IDs will be recorded), where <ending.ped> is the filename extension of file.ped.
- <file.dat>_num.dat - in dir.out directory, the corresponding .dat file, will be different from original <file.dat> if remove.bad.genos=TRUE.
- <save.ids.name> - the patient IDs, if save.ids.name is not empty "".

Author(s)

Olia Vesselova

See Also

pre4.combine.case.control, pre4.combine.case.control.batch, pre5.genos2numeric, pre6.discretize, pre6.discretize.batch

Examples

```
print("See the demo 'gendemo'.")
```

`pre6.discretize` *Discretize the SNP data*

Description

This function finds the splits in diallelic SNPs that are represented as three category discrete variables. The SNPs are dichotomized based on presense of 0 vs. absence of 0.

Usage

```
pre6.discretize(file.train, file.test=file.train, dir.file, dir.out=dir.file,
train.output.append="binary", test.output.append="testbinary", splits.min=0.01,
splits.inc=0.01, splits.max=0.99)
```

Arguments

`file.train`	The name of the train input file that contains the SNPs (represented as 1, 2, and 3) together with binary outcomes. The binary outcome is assumed to occupy the last column in the file.
`file.test`	The name of the test input file that contains the SNPs (represented as 1, 2, and 3) together with binary outcomes. The binary outcome is assumed to occupy the last column in the file. Defaults to *file.train*, if the *file.test* name is not specified.
`dir.file`	The name of directory/path name which contains the *file.train* and *file.test* files.
`dir.out`	The name of directory/path name to which output files should be saved. Defaults to same location as *dir.file*
`train.output.append`	The suffix that should be appended to the end of *file.train* filename to create the filename of the output file. The resultant output filename will be of the format <dir.out>/<file.train>.<train.output.append>.txt, where <file.train> is the filename without extension.
`test.output.append`	The suffix that should be appended to the end of *file.test* filename to create the filename of the output test file. The resultant output filename will be of similar format to that created with *train.output.append*.
`splits.min`	The minimum parameters that define the quantiles that are used to identify the possible splits. Unnecessary parameter when geno values are encoded as 3 categories.
`splits.inc`	The minimum parameters that define the quantiles that are used to identify the possible splits. Unnecessary parameter when geno values are encoded as 3 categories.
`splits.max`	The minimum parameters that define the quantiles that are used to identify the possible splits. Unnecessary parameter when geno values are encoded as 3 categories.

Details

This is the function for Step1 of the MOSS algorithm. It requires its input files to contain genotype values as 1, 2, and 3, and the last column should represent the disease status (1 for CASE and 0 for CONTROL). The diallelic SNPs can be represented as three category discrete variables, since a segregating SNP site has three possible genotypes: 0/0, 0/1, and 1/1, where 0 is the wild type and 1 is the mutant allele. This algorithm dichotomizes the SNPs as presence of 0 vs. absence of 0, or as presence of 1 vs absence of 1. Two output files will be produced for train and test data, and the names of the output files will be returned. This function will also copy all files in directory *dir.file* that end with ".dat" and ".fam", into output directory *dir.out*.

Also this function will write an output file ending with .scores, which contains 4 columns of score values that are used for deciding the split. The .scores file is for debugging purposes: column 1 is the score for encoding 1 as 0 and 2&3 as 1; column 2: encode 1&2 as 0 and 3 as 1; column 3: encode 1&3 as 0 and 2 as 1; column 4 is the highest score chosen.

Value

`out$train`	The name of the train output file.
`out$test`	The name of the test output file.

Note

This step requires LINUX's `wc` functionality.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

pre5.genos2numeric, pre5.genos2numeric.batch, pre6.discretize.batch, pre7.merge.genos, run1.moss.regression

Examples

```
# Split fname into 2 parts for easy deletion of files afterwards.
fstart <- "randvals"
fname <- paste(fstart, ".txt", sep="")

# Create a random matrix of 1,2,3s, and last column of 0s and 1s:
n.rows <- 6
n.cols <- 8
m.part1 <- matrix(round(runif(n.rows*n.cols, min=1, max=3)), n.rows)
m.part2 <- round(runif(n.rows))
#write.table(cbind(m.part1, m.part2), file=fname, append=FALSE, sep="\t",
#col.names=FALSE, row.names=FALSE, quote=FALSE)
write.table(cbind(m.part1, m.part2), file=fname, append=FALSE, sep="\t", col.names=FALSE, rc

outname <- pre6.discretize(file.train=fname, dir.file=".")
```

```
try(system(paste("rm ", fstart, "*", sep="")))
```

`pre6.discretize.batch` *Discretize the SNP data for all files*

Description

For all the files, this function finds the splits in diallelic SNPs that are represented as three category discrete variables. The SNPs are dichotomized based on presense of 0 vs. absence of 0.

Usage

```
pre6.discretize.batch(dir.file, dir.out, prefix.train, prefix.test = prefix.train,
key.train = "", key.test = "", ending.train = ".txt", ending.test = ending.train,
train.output.append = "binary", test.output.append = "testbinary",
splits.min = 0.01, splits.inc = 0.01, splits.max = 0.99)
```

Arguments

`dir.file`	The name of directory that contains the train and test genotype files.
`dir.out`	The name of directory to which output files should go.
`prefix.train`	The beginning of the file name for the TRAIN file (up until chrom number).
`prefix.test`	The beginning of the file name for the TEST file (up until chrom number).
`key.train`	Any keyword in the name of the TRAIN file that distinguishes it from other files.
`key.test`	Any keyword in the name of the TEST file that distinguishes it from other files.
`ending.train`	MUST be the filename extension of the TRAIN file, including the dot ".". For example, if your file is named "CGEM_2.txt", then set this variable to ".txt"; if your file is named "CGEM_2.ped", then set this variable to ".ped".
`ending.test`	MUST be the filename extension of the TEST file, including the dot ".". For example, if your file is named "CGEM_2.txt", then set this variable to ".txt"; if your file is named "CGEM_2.ped", then set this variable to ".ped".
`train.output.append`	The suffix that should be appended to the end of TRAIN filename to create the filename of the output file. The resultant output filename will be of the format <dir.out>/<file.train>.<train.output.append>.txt, where <file.train> is the filename without extension.
`test.output.append`	The suffix that should be appended to the end of TEST filename to create the filename of the output test file. The resultant output filename will be of similar format to that created with *train.output.append*.
`splits.min`	The minimum parameters that define the quantiles that are used to identify the possible splits.
`splits.inc`	The minimum parameters that define the quantiles that are used to identify the possible splits.
`splits.max`	The minimum parameters that define the quantiles that are used to identify the possible splits.

Details

For all the TRAIN and TEST files, this is the function for Step1 of the MOSS algorithm. It requires its input files to contain genotype values as 1, 2, and 3, and the last column should represent the disease status (1 for CASE and 0 for CONTROL). The diallelic SNPs can be represented as three category discrete variables, since a segregating SNP site has three possible genotypes: 0/0, 0/1, and 1/1, where 0 is the wild type and 1 is the mutant allele. This algorithm dichotomizes the SNPs as presence of 0 vs. absence of 0, or as presence of 1 vs absence of 1. Two output files will be produced for train and test data.

This function will also copy all files in directory *dir.file* that end with ".dat" and ".fam", into output directory *dir.out*.

Since TRAIN and TEST set might not yet be split, it is all right to use the same file for both TRAIN and TEST.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

pre5.genos2numeric, pre5.genos2numeric.batch, pre6.discretize, pre7.merge.genos, run1.moss.regression

Examples

```
print("See the demo 'gendemo'.")
```

---

pre7.merge.genos *Combine geno files across all chromosomes*

Description

Puts together all the genos files and their corresponding .dat files for all chromosomes. The files should have last column as the disease status, and the number of individuals (rows) must match across all files. Also the files are expected to have no leading non-snp columns. If they exist, they will be removed. The dat files are expected to have the SNP names in their second column. If the first column of .dat file is 'M', then it will be replaced by the chomosome number of the file name (the number that follows prefix.dat). This function tries to make sure that the geno files and dat files correspond.

Usage

```
pre7.merge.genos(dir.file, dir.dat = dir.file, dir.out = dir.file,
file.out = "CGEM_Breast_complete.txt", dat.out = "CGEM_Breast_complete.dat",
prefix.file, prefix.dat, key.file = "", key.dat = "", ending.file = ".txt",
ending.dat = ".dat", num.nonsnp.col = 0, num.nonsnp.last.col = 1,
weak.check = FALSE, plan = FALSE)
```

Arguments

`dir.file` The name of directory containing files with geno information. The files in this directory must have their last column as the disease status.

`dir.dat` The name of directory containing .dat files. Should be a list of geno IDs, one ID per line, no header. Defaults to same directory as *dir.genos*.

`dir.out` The name of directory where the two output files will go. Defaults to same directory as *dir.genos*.

`file.out` The name of the output file which will contain the combined geno information and the last column will be the disease status.

`dat.out` The name of the output file which will contain all the corresponding SNP values.

`prefix.file` The string that appears at the beginning of all the geno input file names. The file names are expected to begin with *prefix.file*, and then be immediately followed by chromosome number, for example, in *dir.file* directory files named like :

```
"cgem_breast.21.pure.txt"
"cgem_breast.5.pure.txt"
"cgem_breast.24_and_25.txt"
 must have prefix="cgem_breast."
```

`prefix.dat` The string that appears at the beginning of all the .dat file names. Similarly to *prefix.file*, it must be immediately followed by the chromosome number.

`key.file` Any keyword in the name of the geno file that distinguishes it from other files.

`key.dat` Any keyword in the name of the .dat file that distinguishes it from other files.

`ending.file` The string with which all the geno filenames end.

`ending.dat` The string with which all the .dat filenames end.

`num.nonsnp.col`
The number of leading columns in the .ped files that do not contain SNP values. The first columns of the file represent non-SNP values (like patient ID, gender, etc). For MaCH1 input format, the *num.nonsnp.col=5*, for PLINK it is 6 (due to extra disease status column).

`num.nonsnp.last.col`
The number of last columns that do not correspond to geno values. Ex. If last column is the disease status (0s and 1s), then set this variable to 1. If 2 last columns correspond to confounding variables, set the variable to 2.

`weak.check` Since this function will try to check correspondence of the number of genos in the genos file to the .dat file, the function would expect there to be the same number of genos and .dat files. If you wish to by-pass these checks, set *weak.check*=TRUE, in which case only the total final number of the resultant geno and .dat files will be checked for consistency, and only a warning message will be printed if there is a problem.

`plan` Flag: if this option is TRUE, then this function will "do" nothing, but will simply print which files it plans to combine in which order, since combination step itself might take time for large files.

Details

Puts together all the genos files and their corresponding .dat files for all chromosomes. The files should be tab separated and have last column as the disease status, and the number of individuals (rows) must match across all files. Also the files are expected to have no leading non-snp columns. If they exist, they will be removed. The dat files are expected to have the SNP names in their second column. If the first column of .dat file is 'M', then it will be replaced by the chomosome number of the file name (the number that follows prefix.dat). This function tries to make sure that the geno files and dat files correspond.

The resultant combined geno file will be saved into *file.out* and .dat file will be saved in *dat.out*.

Value

The FULL name of the combined result geno file (including the directory).

Note

The function makes use of LINUX commands: 'paste', 'cat', and 'wc'.

Author(s)

Olia Vesselova

See Also

pre6.discretize, pre6.discretize.batch, pre8.add.conf.var, pre9.split.train.test, pre9.split.train.test.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre8.add.conf.var *Append confounding variables*

Description

Appends confounding variables listed in *file.conf* to the end of the *file.name*, right before the disease status (last) column. The output will contain only the patients for which confounding variables exist (other patients will be omitted), so new family file will be written.

Usage

```
pre8.add.conf.var(file.name, dir.file, file.fam, dir.fam = dir.file, file.conf,
dir.conf = dir.file, file.out, fam.out = file.fam, dir.out)
```

Arguments

file.name	The name of the binary data file. The format of this file should have last column as the disease status, tab separated, no header.
dir.file	The name of directory where *file.name* can be found.
file.fam	The name of the family file. Format: one column - one patient ID per line.
dir.fam	The name of directory where *file.fam* can be found.
file.conf	The name of the file that contains confounding variable information. The file should be in the following format:

```
patientID1 1       2 ...
patientID2 3       1 ...
patientID3 2       2 ...
...

- Column 1: patient ID, exactly the same names should appear in
    file.fam;
           * order does not matter;
           * some patients may be missing;
           * no new patients should appear in file.conf (if they
            don't exist in file.fam)
- Column 2: the confounding variable must have no more than 3
            different values.
- Other columns are optional, may be included if there are more
    confounding variables (3 categories each)
- No header
- Tab separated
- No missings or NAs
```

dir.conf	The name of directory where *file.conf* can be found.
file.out	The name of the output file, which will contain all information of *file.name*, plus confounding variables, only for the patients mentioned in *file.conf*.
fam.out	The name of the family output file.
dir.out	The name of directory to which *file.out* and *fam.out* should be saved.

Author(s)

Olia Vesselova

See Also

pre7.merge.genos, pre8.add.conf.var.unix, pre9.split.train.test, pre9.split.train.test

Examples

```
print("See the demo 'gendemo'.")
```

pre8.add.conf.var.unix

Append confounding variables using Linux

Description

Uses Linux functions to append confounding variables listed in *file.conf* to the end of the *file.name*, right before the disease status (last) column. The output will contain only the patients for which confounding variables exist (other patients will be omitted), so new family file will be written. This function is similar to pre8.add.conf.var, only it avoids having to load up into memory the *file.name* (since this file can be very large).

Usage

```
pre8.add.conf.var.unix(file.name, dir.file, file.fam, dir.fam = dir.file,
file.conf, dir.conf = dir.file, file.out, fam.out = file.fam, dir.out)
```

Arguments

file.name	The name of the binary data file. The format of this file should have last column as the disease status, tab separated, no header.
dir.file	The name of directory where *file.name* can be found.
file.fam	The name of the family file. Format: one column - one patient ID per line.
dir.fam	The name of directory where *file.fam* can be found.
file.conf	The name of the file that contains confounding variable information. The file should be in the following format:

```
patientID1 1       2 ...
patientID2 3       1 ...
patientID3 2       2 ...
...

- Column 1: patient ID, exactly the same names should appear in
    file.fam;
           * order does not matter;
           * some patients may be missing;
           * no new patients should appear in file.conf (if they
            don't exist in file.fam)
- Column 2: the confounding variable must have no more than 3
    different values.
- Other columns are optional, may be included if there are
    more confounding variables (3 categories each)
- No header
- Tab separated
- No missings or NAs
```

dir.conf	The name of directory where *file.conf* can be found.

file.out	The name of the output file, which will contain all information of *file.name*, plus confounding variables, only for the patients mentioned in *file.conf*.
fam.out	The name of the family output file.
dir.out	The name of directory to which *file.out* and *fam.out* should be saved.

Author(s)

Olia Vesselova

See Also

pre8.add.conf.var, pre7.merge.genos, pre9.split.train.test, pre9.split.train.test.batch

Examples

```
print("See the demo 'gendemo'.")
```

---

pre9.split.train.test

Split dataset into TRAIN and TEST files

Description

Splits the data file named *file.name* in *dir.file*, into TRAIN and TEST files, based on the percentage *train.percent* - how many percent of the data should go into TRAIN file.

Usage

```
pre9.split.train.test(file.name, dir.file, dir.out, train.percent = 80,
separ = "\t", index.prefix = "index", file.has.ext = TRUE, resample = FALSE)
```

Arguments

file.name	The name of the geno file. This file is expected to have the disease status as its last column (1 for CASE and 0 for CONTROL).
dir.file	The name of directory where *file.name* can be found.
dir.out	The name of directory into which the TRAIN and TEST output files should go.
train.percent	The pecentage (0 to 100) of what portion of data (rows) should go into the TRAIN file; the rest will be in TEST file. Ex: for 1000 entries, if *train.percent=80*, then 800 entries will appear in <file.name>.test, and 200 entries will go into <file.name>.train.
separ	The separator used in the *file.name* to separate entries.
index.prefix	The name of the index file to use for the separation of train from test entries. This file may already exist in *dir.out* (if it has been created by previous runs of this program).

`file.has.ext` Flag whether or not *file.name* has a filename extension (ex. ".txt", ".ped", ".mlgeno").

`resample` Additional file beginning with the name *index.prefix* will be saved in the *dir.out* directory for the given *train.percent*. This file will contain indices that correspond to entries taken into the TRAIN file. If *resample*=FALSE, then all subsequent runs of this function on other files (for example for different chromosomes on the same dataset) with the same *train.percent* will use that saved file. This is to make sure that the same individuals go into TRAIN file, across all chromosomes. If *resample*=TRUE, then new random resampling will take place and new index file will be generated and saved to the *dir.out* directory; note, in this case the entries generated by this file will no longer correspond to entries generated by previous runs for previous index files; so for consistency, re-run all chromosomes with resample flag set to FALSE.

Details

Splits the data file named *file.name* in *dir.file*, into TRAIN and TEST files, based on the percentage *train.percent* - how many percent of the data should go into TRAIN file.

The file *file.name* is expected to have last column represent CASE and CONTROL; this is necessary to make sure that *train.percent* of CASE and *train.percent* of CONTROL entries go into TRAIN file, to have even sample of both types of entries. If the data is saved in many files (for example one file per chromosome), this function is designed to first randomly sample the individuals for the TRAIN file for the first file it is run on. Then it uses this sampling for all other chromosomes on subsequent runs (if resample=FALSE), such that individuals in TRAIN file correspond to one another across all chromosome files (same holds for TEST files). The index file is also useful for processing familyl .fam file after the data has been split.

The following files will be output:

```
- <file.name>.train.<train.percent>.<ext> - the output TRAIN file containing
     train.percent percent of the original data;
     will appear in dir.out directory.
     * <file.name> here is the name without extension;
     * <ext> is the extension part of <file.name> (i.e. the section that
         follows the last "." symbol)
     * <train.percent> is specifying the percentage that was used to generate
         the file.
- <file.name>.test.<train.percent>.<ext> - the entries for TEST file, containing
     the remaining (100 - train.percent) data. Similar to the TRAIN file above.
- <index.prefix>.<train.percent>.txt - the file containing indicies of the
     entries corresponding to TRAIN file, this file will be generated if it
     does not already exist in dir.out, or if resample=TRUE.
```

Value

`out$train` The FULL name of the output TRAIN file

`out$test` The FULL name of the output TEST file

Author(s)

Olia Vesselova

See Also

pre7.merge.genos, pre8.add.conf.var, pre9.split.train.test.batch, run1.moss.regression

Examples

```
print("See the demo 'gendemo'.")
```

---

pre9.split.train.test.batch

Split dataset into TRAIN and TEST files for all files

Description

For all files, splits the data files whose names begin with *prefix.file*, contain a keyword *key.file*, and end with *ending.file*, in *dir.file* into TRAIN and TEST files, based on the percentage *train.percent* - how many percent of the data should go into TRAIN file.

Usage

```
pre9.split.train.test.batch(dir.file, dir.out, prefix.file, key.file = "",
ending.file = ".txt", train.percent = 80, separ = "\t", index.prefix = "index",
file.has.ext = TRUE, resample = FALSE)
```

Arguments

dir.file	The name of directory where input files can be found.
dir.out	The name of directory into which the TRAIN and TEST output files should go.
prefix.file	The beginning of the file name for the geno files (up until chrom number).
key.file	Any keyword in the name of the geno files that distinguishes it from other files.
ending.file	The ending of the geno filenames.
train.percent	The pecentage (0 to 100) of what portion of data (rows) should go into the TRAIN file; the rest will be in TEST file. Ex: for 1000 entries, if *train.percent=80*, then 800 entries will appear in <file.name>.test, and 200 entries will go into <file.name>.train.
separ	The separator used in the *file.name* to separate entries.
index.prefix	The name of the index file to use for the separation of train from test entries. This file may already exist in *dir.out* (if it has been created by previous runs of this program).
file.has.ext	Flag whether or not *file.name* has a filename extension (ex. ".txt", ".ped", ".mlgeno").

`resample` Additional file beginning with the name *index.prefix* will be saved in the *dir.out* directory for the given *train.percent*. This file will contain indices that correspond to entries taken into the TRAIN file. If *resample*=FALSE, then all subsequent runs of this function on other files (for example for different chromosomes on the same dataset) with the same *train.percent* will use that saved file. This is to make sure that the same individuals go into TRAIN file, across all chromosomes. If *resample*=TRUE, then new random resampling will take place and new index file will be generated and saved to the *dir.out* directory; note, in this case the entries generated by this file will no longer correspond to entries generated by previous runs for previous index files; so for consistency, re-run all chromosomes with resample flag set to FALSE.

Details

For all the files in directory *dir.file* satisfying the naming criterion of *prefix.file*, *key.file*, and *ending.file*, split each of these files into TRAIN and TEST files, based on the percentage *train.percent* - how many percent of the data should go into TRAIN file.

The input files are expected to have last column represent CASE and CONTROL; this is necessary to make sure that *train.percent* of CASE and *train.percent* of CONTROL entries go into TRAIN file, to have even sample of both types of entries. If the data is saved in many files (for example one file per chromosome), this function is designed to first randomly sample the individuals for the TRAIN file for the first file it is run on. Then it uses this sampling for all other chromosomes on subsequent runs (if resample=FALSE), such that individuals in TRAIN file correspond to one another across all chromosome files (same holds for TEST files). The index file is also useful for processing familyl .fam file after the data has been split.

The following files will be output:

```
- <file.name>.train.<train.percent>.<ext> - the output TRAIN file containing
    train.percent percent of the original data; will appear in dir.out directory.
     * <file.name> here is the file name without extension;
     * <ext> is the extension part of <file.name> (i.e. the section that follows
       the last "." symbol)
     * <train.percent> is specifying the percentage that was used to generate
       the file.
- <file.name>.test.<train.percent>.<ext> - the entries for TEST file, containing
    the remaining (100 - train.percent) data. Similar to the TRAIN file above.
- <index.prefix>.<train.percent>.txt - the file containing indicies of the
    entries corresponding to TRAIN file, this file will be generated if it
    does not already exist in dir.out, or if resample=TRUE.
```

Author(s)

Olia Vesselova

See Also

pre7.merge.genos, pre8.add.conf.var, pre9.split.train.test, run1.moss.regression

Examples

```
print("See the demo 'gendemo'.")
```

`run1.moss.regression` *Runs MOSS regression algorithm*

Description

This function performs a MOSS search for log-linear models as described in the paper Dobra and Massam (2009) published in Statistical Methodology.

Usage

```
run1.moss.regression(genome.file, max.regressors = 1, chain.iterations = 10000,
chain.replicates = 5, cutoff.max = 0.5, cutoff.min = 0.001, prob.max = 0.1,
num.confounding.vars=0)
```

Arguments

`genome.file`	The input file that contains the dichotomized SNPs together with the binary outcome. The binary outcome is assumed to occupy the last column in the file.
`max.regressors`	The maximum number of predictors allowed to enter a regression. Should be small, at most 5 for most applications.
`chain.iterations`	The number of iterations the stochastic search algorithm will run.
`chain.replicates`	The number of instances of the MOSS algorithm that will be run. Typically 5 instances should be run, with a minimum of 3.
`cutoff.max`	The maximum Bayes factor used to determine which regressions will be retained in the list of current best models.
`cutoff.min`	The minimum Bayes factor.
`prob.max`	The probability of pruning of the current list of models.
`num.confounding.vars`	The number of variables that must always be present in the model. These variables come after the SNP data and before the last column (which denotes the response variable).

Details

MOSS algorithm is run several times *chain.replicates*, to determine if the best regressions have been identified. After the regressions are identified, the best log-linear model associated with the regressions is found. The process outputs 6 files in the same directory where *genome.file* is located. These files begin with the name of the *genome.file*, and end with .countmodels.txt, .log, .reg, .reg.model1.txt, .spaceratio.txt, and .var. The file ending with .reg is necessary for the prediction step functions `run2.prediction.cvv` and `run3.prediction.train.test`.

Value

`out.name` The name of one of the output files (ending with .reg) that will be necessary for the prediction steps.

Note

This is a very computationally expensive step. Requires LINUX's `wc` functionality.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

`run2.prediction.cvv`, `run3.prediction.train.test`

Examples

```
write(rbinom(200,1,0.5), file="randbinary.txt", append=FALSE, sep=" ", ncolumns=50)
outname <- run1.moss.regression("randbinary.txt")
try(system("rm randbinary.txt*"))
```

---

run2.prediction.cvv

Perform cross-validation on regression models

Description

Uses the regression models identified in the MOSS regression step (function `run1.moss.regression`), to perform prediction by cross-validation.

Usage

```
run2.prediction.cvv(genome.file, models.file, max.regressors = 1, cvv.fold = 2,
chain.iterations = 10000)
```

Arguments

`genome.file` The input file that contains the dichtomized SNPs together with the binary outcome. The binary outcome is assumed to occupy the last column in the file. This is the same file as would be given as input to MOSS regression function `run1.moss.regression`.

`models.file` The output file from `run1.moss.regression` function when run on *genome.file* with same value for *max.regressors* parameter. This filename ends with .reg.

`max.regressors` Should be the same as the value used for `run1.moss.regression` function.

`cvv.fold` The type of cross-validation to perform. For example cvv.fold=2 splits the samples in half, fits the models with one half and predicts the second half. cvv.fold=k performs k-fold validation.

`chain.iterations` The number of samples to be drawn from the mixture of regression models.

Details

Performs cross-validation on the *genome.file*, writes a file with name ending with ".cvv.txt".

Value

`name.cvv` The name of the output file (ending with .cvv.txt) that will be necessary for the plotting.

Note

Requires LINUX's `wc` functionality.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

`run1.moss.regression`, `run3.prediction.train.test`

Examples

```
write(rbinom(200,1,0.5), file="randbinary.txt", append=FALSE, sep=" ", ncolumns=50)
name.reg <- run1.moss.regression("randbinary.txt")

name.cvv <- run2.prediction.cvv("randbinary.txt", name.reg)

try(system("rm randbinary.txt*"))
```

---

run3.prediction.train.test

Prediction of test data using regression models

Description

Uses the regression models identified in the MOSS regression step (function `run1.moss.regression`), to perform prediction on the *genome.test.file*.

Usage

```
run3.prediction.train.test(genome.train.file, genome.test.file, models.file,
max.regressors = 1, chain.iterations = 10000)
```

Arguments

genome.train.file	The input file that contains the dichtomized SNPs together with the binary outcome. The binary outcome is assumed to occupy the last column in the file. This is the same file as would be given as input to MOSS regression function run1.moss.regression.
genome.test.file	This file is used for testing. Should be of the same format as *genome.train.file* with the same number of variables (columns), and arbitrary sample size (rows).
models.file	The output file from run1.moss.regression function when run on *genome.train.file* with same value for *max.regressors* parameter. The file name would end in ".reg".
max.regressors	Should be the same as the value used for run1.moss.regression function.
chain.iterations	The number of samples to be drawn from the mixture of regression models.

Value

name.fitted	The name of the output file (ending with .fitted) that will be necessary for the plotting.

Note

Requires LINUX's wc functionality.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

run1.moss.regression, run2.prediction.cvv

Examples

```
write(rbinom(200,1,0.5), file="randbinary.txt", append=FALSE, sep=" ", ncolumns=50)
name.reg <- run1.moss.regression("randbinary.txt")

name.fitted <- run3.prediction.train.test("randbinary.txt", "randbinary.txt", name.reg)

try(system("rm randbinary.txt*"))
```

`run4.save.prediction` *Saves the plot of predicted values and ROC curve*

Description

Saves a plot of the predicted values and the corresponding ROC curve for the resulting prediction file generated by either `run3.prediction.train.test` or `run2.prediction.cvv` functions into specified pdf file.

Usage

```
run4.save.prediction(filename, outfile)
```

Arguments

`filename`	The file produced either by `run2.prediction.cvv` or `run3.prediction.train.test` functions.
`outfile`	The name of output file which will contain the pdf plot. The default output file name is <filename>.plot.pdf.

Value

`out.tpr`	The true positive rate.
`out.fpr`	The false positive rate.
`out.rocarea`	The ROC curve area.

Note

This function takes a while to run.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

`run4.show.prediction`, `run1.moss.regression`, `run2.prediction.cvv`, `run3.prediction.train`

Examples

```
fname <- "randbinary.txt"
write(rbinom(200,1,0.5), file=fname, append=FALSE, sep=" ", ncolumns=50)
name.reg <- run1.moss.regression(fname)
name.cvv <- run2.prediction.cvv(fname, name.reg)
run4.save.prediction(name.cvv)

try(system(paste("rm ", fname, "*", sep="")))
```

`run4.show.prediction` *Plot predicted values and ROC curve*

Description

Creates a plot of the predicted values and the corresponding ROC curve for the resulting prediction file generated by either `run3.prediction.train.test` or `run2.prediction.cvv` functions.

Usage

```
run4.show.prediction(filename)
```

Arguments

`filename`	The file produced either by `run2.prediction.cvv` or `run3.prediction.train.test` functions.

Value

`out.tpr`	The true positive rate.
`out.fpr`	The false positive rate.
`out.rocarea`	The ROC curve area.

Note

This function takes a while to run.

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

`run4.save.prediction`, `run1.moss.regression`, `run2.prediction.cvv`, `run3.prediction.train`

Examples

```
fname <- "randbinary.txt"
write(rbinom(200,1,0.5), file=fname, append=FALSE, sep=" ", ncolumns=50)
name.reg <- run1.moss.regression(fname)
name.cvv <- run2.prediction.cvv(fname, name.reg)
run4.show.prediction(name.cvv)

try(system(paste("rm ", fname, "*", sep="")))
```

run5.brier *Compute the Brier Score.*

Description

Computes the Brier score. Returns the mean and standard deviation.

Usage

```
run5.brier(filename)
```

Arguments

filename	The name of file that was produced either by run2.prediction.cvv or run3.prediction.train.test functions.

Value

out$mean	The mean of the results
out$std	The standard deviation

Author(s)

Laurent Briollais, Adrian Dobra, Olga Vesselova

See Also

run1.moss.regression, run2.prediction.cvv, run3.prediction.train.test, run4.show.prediction

Examples

```
write(rbinom(200,1,0.5), file="randbinary.txt", append=FALSE, sep=" ", ncolumns=50)
run1.moss.regression("randbinary.txt")
run2.prediction.cvv("randbinary.txt", "randbinary.txt.shotgun.50.1.reg")

run5.brier("randbinary.txt.49.cvv.txt")

try(system("rm randbinary.txt*"))
```

tune1.subsets *Imputes dense map of SNPs on chromosome regions with MaCH*

Description

For chromosomes and their small regions specified, run MaCH1 with hapmap to get more detailed sampling of SNPs in the region, and prepares this subset of data to be processed by MOSS algorithm.

Usage

```
tune1.subsets(dir.dat, dir.ped, dir.ann, dir.pos.snp, dir.pos.ann, dir.pos.hap, dir
```

Arguments

dir.dat	The name of directory where file listing SNPs of the dataset can be found.
dir.ped	The name of directory where file with data of the dataset can be found.
dir.ann	The name of directory where SNP position information for the dataset can be found. Note: this file must contain position information about all SNPs that are listed in .dat; all other SNPs will be ignored.
dir.pos.snp	The name of directory where hapmap SNP list can be found.
dir.pos.ann	The name of directory where hapmap annotation file containing position information can be found.
dir.pos.hap	The name of directory where the hapmap zipped data can be found.
dir.out	The name of directory to which output folder should be placed.
prefix.dat	The beginning of the file name for dataset's list of SNPs.
prefix.ped	The beginning of the file name for dataset's data.
prefix.ann	The beginning of the file name for dataset's SNP position information.
prefix.pos.snp	The beginning of the file name for hapmap's list of SNPs.
prefix.pos.ann	The beginning of the file name for hapmap's SNP position information.
prefix.pos.hap	The beginning of the file name for hapmap's data.
key.dat	Any keyword in the name of dataset's list of SNPs.
key.ann	Any keyword in the name of dataset's SNP position information.
key.pos.ann	Any keyword in the name of hapmap's SNP position information.
key.pos.hap	Any keyword in the name of hapmap's data.
ending.dat	The ending of dataset's list of SNPs filename.
ending.ped	The ending of dataset's data filename.
ending.ann	The ending of dataset's SNP position information filename.

`ending.pos.snp`
: The ending of hapmap's list of SNPs filename.

`ending.pos.ann`
: The ending of hapmap's SNP position information filename.

`ending.pos.hap`
: The ending of hapmap's data filename.

`pos.list.triple`
: A list of chromosomes and position boundaries to be expanded upon. The list should contain information in the order: (chromosome number, start position, end position, chromosome number, start position, end position, etc.). This allows users to specify multiple chromosomes with multiple regions within each chromosome. For example, specifying region of positions 6000-19000 and 111000-222000 in chrom 15, together with positions 55000-77000 in chrom 21, can be listed as: c(15, 6000, 19000, 15, 111000, 222000, 21, 55000, 77000). Note that MaCH will be run on one chromosome at a time, and for all its specified regions.

`ped.nonsnp`
: The number of non-snp leading columns in dataset's data file. For example input to MaCH format has 5 columns, Plink has 6 columns.

`ann.header`
: Whether or not hapmap's SNP position information file has a header. Ex. .annotation.txt = TRUE, .legend.txt = TRUE. Since format of the hapmap file is not hard-coded, specify the format of your prefered hapmap library; the defaults are set to the 1000 Genome data (from MaCH website).

`pos.ann.snpcol`
: The column number in hapmap's SNP position information file that contains SNP names/ids. For example in .annotation.txt it's column 5; in .legend.txt it's column 1.

`pos.ann.poscol`
: The column number in hapmap's SNP position information file that contains position information. For example in .annotation.txt it's column 2; in .legend.txt it's also column 2.

`pos.ann.header`
: Whether or not dataset's SNP position information file has a header. Ex. .map = FALSE, but other formats might have a header. Since format of this file is not hard-coded, specify the format that your dataset comes with.

`ann.snpcol`
: The column number in dataset's SNP position information file that contains SNP names/ids. For example in .map it is column 2.

`ann.poscol`
: The column number in dataset's SNP position information file that contains position information. For example in .map it is column 4.

`ann.chrcol`
: The column number in dataset's SNP position information file that contains chromosome number information. In .map format there is no such column, since there is a unique file per chromosome, thus default for this parameter is 0. In case if all position information is included in one single file for all/many chromosomes, specify which column corresponds to chromosome number.

`pos.hap.nonsnp`
: The number of non-SNP leading columns in hapmap's data file. In .hap.gz it is 2.

`out.name.subdir`
: The name of subdirectory structure to be created for output for this sequence of chromosomes and positions. Note: this folder name MUST be different for each different set of chromosome and position boundaries triplets.

`out.prefix`
: The beginning of output file names.

`rsq.thresh`
: Threshold for RSQ of MaCH's imputation. Recommended default is 0.5.

`num.iters`
: The number of iterations MaCH should make in its first step to estimate its model parameters.

`hapmapformat`
: The type of haplotype data format: 1000G haplotype dataset has .snps file with one column, so *hapmapformat* defaults to FALSE. Another dataset format listing SNPs (.legend.txt) has 4 columns - change *hapmapformat* to TRUE.

`mach.loc`
: The location directory where "mach" executable can be found.

Details

The input files for this function are inteded to be from different folders of the subdirectory structure used in preprocessing steps (see `pre0.dir.create`). The dataset's SNP (.dat) and data (.ped) information are intended to come from d3 (d03_removed); whereas the dataset's position information (.map) can be obtained from d1 (d01_plink) subdirectory. The hapmap files are huge and can be used by many datasets, thus there is no need to keep a copy of them in our subdirectory structure for each dataset. Note: if the hapmap file that specifies SNP information ALSO lists their position information, simply provide that file (and it's column format) to this function twice (as *prefix.pos.snp* and *prefix.pos.ann*). This function is meant to begin from early preprocessing steps, re-run MaCH with hapmap on desired regions, then combine CASE with CONTROL, and call all the pre-processing functions in sequence up until `pre7.merge.genos`. At the end, the output will be a single file ready to be called by MOSS `run1.moss.regression`. A new convenient subdirectory structure will be created, similar to `pre0.dir.create` within new directory *out.name.subdir*. This function requires two sets of data: user's dataset and reference haplotypes. There are many hapmap libraries for download from the web, so this function tries to be as general as possible to allow users to give column information about the format. MaCH also needs to understand the given hapmap format. The defaults are set for 1000G Phase I(a) from MaCH's website: `http://www.sph.umich.edu/csg/abecasis/MaCH/download/1000G-PhaseI-Interim.html`. Note: the data file (.hap.gz) is expected to be zipped. However please unzip the .annotation.txt file before calling this function. The first thing this function would do is extract the given position intervals from user's datafiles and from haplotype files. This would make both files smaller so that running MaCH is feasible. MaCH will be run on CASE and CONTROL data files separately. After MaCH is run with hapmap, most of the predicted SNPs would have very low RSQ score, thus out of thousands of SNPs that are within the region in hapmap file, only hundreds will be actually reliable. This function prunes out all the SNPs with RSQ score lower than *rsq.thresh*. Then CASE and CONTROL will be combined based on common remaining SNPs. Then the function will run the three preprocessing functions (`pre5.genos2numeric.batch`, `pre6.discretize.batch`, `pre7.merge.genos`) to output the final ready-to-use file.

Value

The FULL name of the combined result geno file (including the directory).

Author(s)

Olia Vesselova

References

MaCH website: http://www.sph.umich.edu/csg/abecasis/MACH/download/

See Also

pre2.remove.genos, pre2.remove.genos.batch, pre3.call.mach, pre4.combine.case.control, pre4.combine.case.control.batch, pre5.genos2numeric, pre5.genos2numeric.batch, pre6.discretize, pre6.discretize.batch, pre7.merge.genos, run1.moss.regression

Examples

```
print("See the demo 'gendemo'.")
```

Index