Package 'HIBAG'

December 14, 2024

Type Package

Title HLA Genotype Imputation with Attribute Bagging

Version 1.43.1

Date 2024-11-19

Depends R (>= 3.2.0)

Imports methods, RcppParallel

Suggests parallel, ggplot2, reshape2, gdsfmt, SNPRelate, SeqArray, knitr, markdown, rmarkdown, Rsamtools

LinkingTo RcppParallel (>= 5.0.0)

Description Imputes HLA classical alleles using GWAS SNP data, and it relies on a training set of HLA and SNP genotypes. HIBAG can be used by researchers with published parameter estimates instead of requiring access to large training sample datasets. It combines the concepts of attribute bagging, an ensemble classifier method, with haplotype inference for SNPs and HLA types. Attribute bagging is a technique which improves the accuracy and stability of classifier ensembles using bootstrap aggregating and random variable selection.

License GPL-3

LazyData yes

VignetteBuilder knitr

SystemRequirements C++11, GNU make

ByteCompile TRUE

biocViews Genetics, StatisticalMethod

URL https://github.com/zhengxwen/HIBAG, https://hibag.s3.amazonaws.com/index.html

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

git_branch devel

git_last_commit 4d1767d

git_last_commit_date 2024-11-19

Repository Bioconductor 3.21

Date/Publication 2024-12-13

Author Xiuwen Zheng [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-1390-0708>),
Bruce Weir [ctb, ths] (ORCID: <https://orcid.org/0000-0002-4883-1247>)

Maintainer Xiuwen Zheng <zhengx@u.washington.edu>

Contents

`HIBAG-package` *HLA Genotype Imputation with Attribute Bagging*

Description

To impute HLA types from unphased SNP data using an attribute bagging method.

Details

Package:	HIBAG
Type:	R/Bioconductor Package
License:	GPL version 3
Kernel Version:	v1.5

HIBAG is a state of the art software package for imputing HLA types using SNP data, and it uses the R statistical programming language. HIBAG is highly accurate, computationally tractable, and can be used by researchers with published parameter estimates instead of requiring access to large training sample datasets. It combines the concepts of attribute bagging, an ensemble classifier method, with haplotype inference for SNPs and HLA types. Attribute bagging is a technique which improves the accuracy and stability of classifier ensembles using bootstrap aggregating and random variable selection.

Features:
1) HIBAG can be used by researchers with published parameter estimates (`https://hibag.s3.amazonaws.com/hlares_index.html`) instead of requiring access to large training sample datasets.
2) A typical HIBAG parameter file contains only haplotype frequencies at different SNP subsets

rather than individual training genotypes.
3) SNPs within the xMHC region (chromosome 6) are used for imputation.
4) HIBAG employs unphased genotypes of unrelated individuals as a training set.
5) HIBAG supports parallel computing with R.

Author(s)

Xiuwen Zheng [aut, cre, cph] <zhengx@u.washington.edu>, Bruce S. Weir [ctb, ths] <bsweir@u.washington.edu>

References

Zheng X, Shen J, Cox C, Wakefield J, Ehm M, Nelson M, Weir BS; HIBAG – HLA Genotype Imputation with Attribute Bagging. The Pharmacogenomics Journal. doi: 10.1038/tpj.2013.18. https://www.nature.com/articles/tpj201318

Examples

```
# HLA_Type_Table data
head(HLA_Type_Table)
dim(HLA_Type_Table)  # 60 13

# HapMap_CEU_Geno data
summary(HapMap_CEU_Geno)


#############################################################################

# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel=match(snpid, HapMap_CEU_Geno$snp.id),
```

```
    samp.sel=match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train a HIBAG model
set.seed(100)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=4,
    verbose.detail=TRUE)
summary(model)

# validation
pred <- hlaPredict(model, test.geno)
summary(pred)

# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0))
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0.5))


# save the parameter file
mobj <- hlaModelToObj(model)
save(mobj, file="HIBAG_model.RData")
save(test.geno, file="testgeno.RData")
save(hlatab, file="HLASplit.RData")

# Clear Workspace
hlaClose(model)  # release all resources of model
rm(list = ls())


#############################################################################

# NOW, load a HIBAG model from the parameter file
mobj <- get(load("HIBAG_model.RData"))
model <- hlaModelFromObj(mobj)

# validation
test.geno <- get(load("testgeno.RData"))
hlatab <- get(load("HLASplit.RData"))

pred <- hlaPredict(model, test.geno)
# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0.5))


#############################################################################
# import a PLINK BED file
```

```
#
bed.fn <- system.file("extdata", "HapMap_CEU.bed", package="HIBAG")
fam.fn <- system.file("extdata", "HapMap_CEU.fam", package="HIBAG")
bim.fn <- system.file("extdata", "HapMap_CEU.bim", package="HIBAG")
hapmap.ceu <- hlaBED2Geno(bed.fn, fam.fn, bim.fn, assembly="hg19")


##############################################################################
# predict
#
pred <- hlaPredict(model, hapmap.ceu, type="response")
head(pred$value)
#   sample.id allele1 allele2      prob
# 1   NA10859   01:01   03:01 0.9999992
# 2   NA11882   01:01   29:02 1.0000000
# ...


# delete the temporary files
unlink(c("HIBAG_model.RData", "testgeno.RData", "HLASplit.RData"), force=TRUE)
```

HapMap_CEU_Geno *SNP genotypes of a study simulated from HapMap CEU genotypic data*

Description

An object of hlaSNPGenoClass of 60 samples and 1564 SNPs.

Usage

```
HapMap_CEU_Geno
```

Value

A list

References

https://www.ncbi.nlm.nih.gov/variation/news/NCBI_retiring_HapMap/

The International HapMap Consortium. A second generation human haplotype map of over 3.1 million SNPs. Nature 449, 851-861. 2007.

hlaAASeqClass *Class of HLA Amino Acid Sequence Type*

Description

The definition of a class for HLA protein amino acid sequences.

Value

There are following components:

`locus`	HLA locus
`pos.start`	the starting position in basepair
`pos.end`	the end position in basepair
`value`	a data frame
`assembly`	the human genome reference, such like "hg19"
`start.position`	the start position
`reference`	reference sequence

The component `value` includes:

`sample.id`	sample ID
`allele1`	amino acid or nucleotide sequence
`allele2`	amino acid or nucleotide sequence
`P1, ..., Pn`	if applicable, a matrix of posterior probability, row – sample, column – position of amino acid

Author(s)

Xiuwen Zheng

See Also

`hlaConvSequence`

`hlaAllele` *A list of HLA/KIR types*

Description

Return an object of `hlaAlleleClass`, which contains HLA/KIR types.

Usage

```
hlaAllele(sample.id, H1, H2, max.resolution="", locus="any", assembly="auto",
    locus.pos.start=NA_integer_, locus.pos.end=NA_integer_, prob=NULL,
    na.rm=TRUE)
```

Arguments

`sample.id`	sample IDs
`H1`	a vector of HLA/KIR alleles
`H2`	a vector of HLA/KIR alleles
`max.resolution`	"2-digit", "1-field", "4-digit", "2-field", "6-digit", "3-field", "8-digit", "4-field", "allele", "protein", "full", "none", or "": "allele" = "2-digit"; "protein" = "4-digit"; "full", "none" or "" for no limit on resolution
`locus`	the name of HLA locus: "A", "B", "C", "DRB1", "DRB5", "DQA1", "DQB1", "DPB1", KIR locus, or "any", where "any" indicates any other multiallelic locus; see `hlaLociInfo` for possible locus names
`assembly`	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
`locus.pos.start`	the starting position in basepair
`locus.pos.end`	the end position in basepair
`prob`	the probabilities assigned to the samples
`na.rm`	if TRUE, remove the samples without valid HLA types

Details

The format of `H1` and `H2` is "allele group : different protein : synonymous mutations in exons : synonymous mutations in introns"L, where the suffix L is express level (N, null; L, low; S, secreted; A, aberrant; Q: questionable). For example, "44:02:01:02L". If `max.resolution` is specified, the HLA alleles will be trimmed with a possible maximum resolution.

Value

Return a `hlaAlleleClass` object, and it is a list:

`locus`	HLA locus
`pos.start`	the starting position in basepair

pos.end	the end position in basepair
value	a data frame
assembly	the human genome reference, such like "hg19"

The component value includes:

sample.id	sample ID
allele1	HLA allele
allele2	HLA allele
prob	the posterior probability

Author(s)

Xiuwen Zheng

See Also

hlaAlleleDigit, hlaAlleleSubset, hlaLociInfo, hlaAlleleToVCF

Examples

```
head(HLA_Type_Table)
dim(HLA_Type_Table)  # 60 13

# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")
summary(hla)


# encode other loci
hlaAllele("HD0010", "1", "2", locus="NewLocus")
```

---

hlaAlleleClass *Class of HLA/KIR Type*

Description

The definition of a class for HLA/KIR types, returned from hlaAllele.

Value

There are following components:

`locus`	HLA/KIR locus
`pos.start`	the starting position in basepair
`pos.end`	the end position in basepair
`value`	a data frame
`assembly`	the human genome reference, such like "hg19"
`postprob`	if applicable, a matrix of all posterior probabilities

~

The component `value` includes:

`sample.id`	sample ID
`allele1`	HLA allele
`allele2`	HLA allele
`prob`	if applicable, the posterior probability

Author(s)

Xiuwen Zheng

See Also

`hlaAllele`

---

hlaAlleleDigit *Trim HLA alleles*

Description

Trim HLA alleles to specified width.

Usage

```
hlaAlleleDigit(obj, max.resolution=NA_character_, rm.suffix=FALSE)
```

Arguments

`obj`	should be a `hlaAlleleClass` object or characters
`max.resolution`	"2-digit", "1-field", "4-digit", "2-field", "6-digit", "3-field", "8-digit", "4-field", "allele", "protein", "full", "none", or "": "allele" = "2-digit"; "protein" = "4-digit"; "full", "none" or "" for no limit on resolution
`rm.suffix`	whether remove the non-digit suffix in the last field, e.g., for "01:22N", "N" is a non-digit suffix

Details

If `max.resolution` is specified, the HLA alleles will be trimmed with the maximum resolution. See https://hla.alleles.org/nomenclature/naming.html for the HLA nomenclature.

Value

Return a `hlaAlleleClass` object if `obj` is `hlaAlleleClass`-type, or characters if `obj` is character-type.

Author(s)

Xiuwen Zheng

See Also

`hlaAllele`

Examples

```
head(HLA_Type_Table)
dim(HLA_Type_Table)  # 60 13

# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus = hla.id, assembly="hg19")
summary(hla)

hla2 <- hlaAlleleDigit(hla, "2-digit")
summary(hla2)
```

`hlaAlleleSubset` *Get a subset of HLA/KIR types*

Description

Get a subset of HLA/KIR types from an object of `hlaAlleleClass`.

Usage

```
hlaAlleleSubset(hla, samp.sel=NULL)
```

Arguments

`hla`	an object of `hlaAlleleClass`
`samp.sel`	a logical vector, or an integer vector of indices

Value

Return hlaAlleleClass.

Author(s)

Xiuwen Zheng

See Also

hlaAllele, hlaAlleleDigit

Examples

```
head(HLA_Type_Table)
dim(HLA_Type_Table)  # 60 13

# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")
summary(hla)

subhla <- hlaAlleleSubset(hla, 1:100)
summary(subhla)
```

---

hlaAlleleToVCF *Convert HLA alleles to VCF*

Description

To convert the HLA allele data to a VCF file.

Usage

```
hlaAlleleToVCF(hla, outfn, DS=TRUE, allele.list=FALSE, prob.cutoff=NaN,
    verbose=TRUE)
```

Arguments

hla	an object of hlaAlleleClass for HLA alleles, or a list of hlaAlleleClass objects
outfn	a VCF file name or a connection; if outfn ends with ".gz" or ".xz", gzfile or xzfile will be used to compress the output file
DS	if TRUE, output dosages in the DS field

allele.list	a logical value or a character vector for a list of alleles; when it is a logical value, if TRUE and dosage is available, use all possible alleles in the dosages; otherwise, use the alleles predicted at least once
prob.cutoff	a probability threshold for setting the output alleles and dosages to missing; the output VCF file contains all samples in hla ignoring prob.cutoff
verbose	if TRUE, show information

Value

Return outfn.

Author(s)

Xiuwen Zheng

References

Zheng X, Shen J, Cox C, Wakefield J, Ehm M, Nelson M, Weir BS; HIBAG – HLA Genotype Imputation with Attribute Bagging. Pharmacogenomics Journal. doi: 10.1038/tpj.2013.18. https://www.nature.com/articles/tpj201318

See Also

hlaAttrBagging, hlaAllele

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# train a HIBAG model
set.seed(100)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hla, HapMap_CEU_Geno, nclassifier=2)
summary(model)

# validation
pred <- hlaPredict(model, HapMap_CEU_Geno)
summary(pred)
```

```
# output to standard output with dosages
hlaAlleleToVCF(hlaAlleleSubset(pred, 1:4), stdout())
```

`hlaAssocTest` *Statistical Association Tests*

Description

Perform statistical association tests via Pearson's Chi-squared test, Fisher's exact test and logistic regressions.

Usage

```
## S3 method for class 'hlaAlleleClass'
hlaAssocTest(hla, formula, data,
    model=c("dominant", "additive", "recessive", "genotype"),
    model.fit=c("glm"), prob.threshold=NaN, use.prob=FALSE, showOR=FALSE,
    verbose=TRUE, ...)
## S3 method for class 'hlaAASeqClass'
hlaAssocTest(hla, formula, data,
    model=c("dominant", "additive", "recessive", "genotype"),
    model.fit=c("glm"), prob.threshold=NaN, use.prob=FALSE, showOR=FALSE,
    show.all=FALSE, verbose=TRUE, ...)
```

Arguments

`hla`	an object of `hlaAlleleClass`
`formula`	an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted, e.g., `y ~ 1`, `y ~ h + a`
`data`	an optional data frame, list or environment containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`
`model`	dominant, additive, recessive or genotype models: `"dominant"` is default
`model.fit`	"glm" – generalized linear regression
`prob.threshold`	the probability threshold to exclude individuals with low confidence scores
`use.prob`	if `TRUE`, use the posterior probabilities as weights in `glm` models
`showOR`	show odd ratio (OR) instead of log OR if `TRUE`
`show.all`	if `TRUE`, show both significant and non-significant results; if `FALSE`, only show significant results
`verbose`	if TRUE, show information
`...`	optional arguments to `glm` or `nlme` call

Details

model	description (given a specific HLA allele h)
dominant	[-/-] vs. [-/h,h/h] (0 vs. 1 in design matrix)
additive	[-] vs. [h] in Chi-squared and Fisher's exact test, the allele dosage in regressions (0: -/-, 1: -/h, 2: h/h)
recessive	[-/-,-/h] vs. [h/h] (0 vs. 1 in design matrix)
genotype	[-/-], [-/h], [h/h] (0 vs. 1 in design matrix)

In allelic associations, Chi-squared and Fisher exact tests are preformed on the cross tabulation, which is constructed according to the specified model (dominant, additive, recessive and gneotype).

In amino acid associations, Fisher exact test is performed on a cross tabulation with the numbers of each amino acid stratified by response variable (e.g., disease status).

In linear and logistic regressions, 95% confidence intervals are calculated based on asymptotic normality. The option `use.prob=TRUE` might be useful in the sensitivity analysis.

Value

Return a `data.frame` with

`[-]`	the number of haplotypes not carrying the specified HLA allele
`[h]`	the number of haplotype carrying the specified HLA allele
`%.[-]`, `...`	case/disease proportion in the group [-], ...
`[-/-]`	the number of individuals or haplotypes not carrying the specified HLA allele
`[-/h]`	the number of individuals or haplotypes carrying one specified HLA allele
`[-/h]`	the number of individuals or haplotypes carrying two specified HLA alleles
`[-/h, h/h]`	the number of individuals or haplotypes carrying one or two specified HLA alleles
`[-/-, -/h]`	the number of individuals or haplotypes carrying at most one specified HLA allele
`%.[-/-]`, `...`	case/disease proportion in the group [-/-], ...
`avg.[-/-]`, `...`	outcome average in the group [-/-], ...
`chisq.st`	the value the chi-squared test statistic
`chisq.p`	the p-value for the Chi-squared test
`fisher.p`	the p-value for the Fisher's exact test
`h.est`	the coefficient estimate of HLA allele
`h.25%`, `h.75%`	the 95% confidence interval for HLA allele
`h.pval`	p value for HLA allele

Author(s)

Xiuwen Zheng

See Also

`hlaConvSequence`, `summary.hlaAASeqClass`

Examples

```
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

set.seed(1000)
n <- nrow(hla$value)
dat <- data.frame(case = c(rep(0, n/2), rep(1, n/2)), y = rnorm(n),
    pc1 = rnorm(n))

hlaAssocTest(hla, case ~ 1, data=dat)
hlaAssocTest(hla, case ~ 1, data=dat, model="additive")
hlaAssocTest(hla, case ~ 1, data=dat, model="recessive")
hlaAssocTest(hla, case ~ 1, data=dat, model="genotype")

hlaAssocTest(hla, y ~ 1, data=dat)
hlaAssocTest(hla, y ~ 1, data=dat, model="genotype")

hlaAssocTest(hla, case ~ h, data=dat)
hlaAssocTest(hla, case ~ h + pc1, data=dat)
hlaAssocTest(hla, case ~ h + pc1, data=dat, showOR=TRUE)

hlaAssocTest(hla, y ~ h, data=dat)
hlaAssocTest(hla, y ~ h + pc1, data=dat)
hlaAssocTest(hla, y ~ h + pc1, data=dat, showOR=TRUE)

hlaAssocTest(hla, case ~ h, data=dat, model="additive")
hlaAssocTest(hla, case ~ h, data=dat, model="recessive")
hlaAssocTest(hla, case ~ h, data=dat, model="genotype")
```

---

hlaAttrBagClass *The class of HIBAG model*

Description

The class of a HIBAG model, and its instance is returned from `hlaAttrBagging`.

Value

Return a list of:

`n.samp`	the total number of training samples
`n.snp`	the total number of candidate SNP predictors
`sample.id`	the sample IDs
`snp.id`	the SNP IDs
`snp.position`	SNP position in basepair

snp.allele	a vector of characters with the format of "A allele/B allele"
snp.allele.freq	the allele frequencies
hla.locus	the name of HLA locus
hla.allele	the HLA alleles used in the model
hla.freq	the HLA allele frequencies
assembly	the human genome reference, such like "hg19"
model	internal use
appendix	an optional list: platform – supported platform(s); information – other information, like training sets, authors; warning – any warning message
matching	matching proportion in the training set

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, hlaParallelAttrBagging, hlaAttrBagObj

---

hlaAttrBagging *Build a HIBAG model*

Description

To build a HIBAG model for predicting HLA types with SNP markers.

Usage

```
hlaAttrBagging(hla, snp, nclassifier=100L, mtry=c("sqrt", "all", "one"),
    prune=TRUE, na.rm=TRUE, mono.rm=TRUE, maf=NaN, nthread=1L, verbose=TRUE,
    verbose.detail=FALSE)
```

Arguments

hla	the training HLA types, an object of hlaAlleleClass
snp	the training SNP genotypes, an object of hlaSNPGenoClass
nclassifier	the total number of individual classifiers
mtry	a character or a numeric value, the number of variables randomly sampled as candidates for each selection. See details
prune	if TRUE, to perform a parsimonious forward variable selection, otherwise, exhaustive forward variable selection. See details
na.rm	if TRUE, remove the samples with missing HLA alleles

mono.rm	if TRUE, remove monomorphic SNPs
maf	MAF threshold for SNP filter, excluding any SNP with MAF < maf
nthread	specify the number of threads used in the model building; if TRUE, use the number of threads returned from RcppParallel::defaultNumThreads() (by default using all threads)
verbose	if TRUE, show information
verbose.detail	if TRUE, show more information

Details

mtry (the number of variables randomly sampled as candidates for each selection, "sqrt" by default): "sqrt", using the square root of the total number of candidate SNPs; "all", using all candidate SNPs; "one", using one SNP; an integer, specifying the number of candidate SNPs; 0 < r < 1, the number of candidate SNPs is "r * the total number of SNPs".

prune: there is no significant difference on accuracy between parsimonious and exhaustive forward variable selections. If prune=TRUE, the searching algorithm performs a parsimonious forward variable selection: if a new SNP predictor reduces the current out-of-bag accuracy, then it is removed from the candidate SNP set for future searching. Parsimonious selection helps to improve the computational efficiency by reducing the searching times on non-informative SNP markers.

hlaParallelAttrBagging extends hlaAttrBagging to allow parallel computing with multiple compute nodes in a cluster. An autosave function is available in hlaParallelAttrBagging when an new individual classifier is built internally without completing the ensemble.

Value

Return an object of hlaAttrBagClass:

n.samp	the total number of training samples
n.snp	the total number of candidate SNP predictors
sample.id	the sample IDs
snp.id	the SNP IDs
snp.position	SNP position in basepair
snp.allele	a vector of characters with the format of “A allele/B allele”
snp.allele.freq	the allele frequencies
hla.locus	the name of HLA locus
hla.allele	the HLA alleles used in the model
hla.freq	the HLA allele frequencies
assembly	the human genome reference, such like "hg19"
model	internal use
matching	matching proportion in the training set

Author(s)

Xiuwen Zheng

References

Zheng X, Shen J, Cox C, Wakefield J, Ehm M, Nelson M, Weir BS; HIBAG – HLA Genotype Imputation with Attribute Bagging. Pharmacogenomics Journal. doi: 10.1038/tpj.2013.18. https://www.nature.com/articles/tpj201318

See Also

hlaClose, hlaParallelAttrBagging, summary.hlaAttrBagClass, predict.hlaAttrBagClass, hlaPredict, hlaSetKernelTarget

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel=match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel=match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train a HIBAG model
set.seed(100)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=4,
    verbose.detail=TRUE)
summary(model)

# validation
pred <- hlaPredict(model, test.geno)
summary(pred)
```

```
# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0))
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0.5))


# save the parameter file
mobj <- hlaModelToObj(model)
save(mobj, file="HIBAG_model.RData")
save(test.geno, file="testgeno.RData")
save(hlatab, file="HLASplit.RData")

# Clear Workspace
hlaClose(model)  # release all resources of model
rm(list = ls())


######################################################################

# NOW, load a HIBAG model from the parameter file
mobj <- get(load("HIBAG_model.RData"))
model <- hlaModelFromObj(mobj)

# validation
test.geno <- get(load("testgeno.RData"))
hlatab <- get(load("HLASplit.RData"))

pred <- hlaPredict(model, test.geno, type="response")
summary(pred)

# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0))
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0.5))


# delete the temporary files
unlink(c("HIBAG_model.RData", "testgeno.RData", "HLASplit.RData"), force=TRUE)
```

`hlaAttrBagObj` *The class of HIBAG object*

Description

The class of a HIBAG object, which can be saved in the `.RData` file.

Value

A list of:

`n.samp`	the total number of training samples
`n.snp`	the total number of candidate SNP predictors
`sample.id`	the sample IDs
`snp.id`	the SNP IDs
`snp.position`	SNP position in basepair
`snp.allele`	a vector of characters with the format of "A allele/B allele"
`snp.allele.freq`	the allele frequencies
`hla.locus`	the name of HLA locus
`hla.allele`	the HLA alleles used in the model
`hla.freq`	the HLA allele frequencies
`assembly`	the human genome reference, such like "hg19"
`classifiers`	a list of all classifiers (described as follows)
`matching`	matching proportion in the training set
`appendix`	`platform` – supported platform(s); `information` – other information, like training sets, authors; `warning` – any warning message

`classifiers` has the following components:

`samp.num`	the number of copies of samples in a bootstrap sample
`haplos`	a data.frame of haplotype frequencies
.	`freq` – haplotype frequency
.	`hla` – a HLA allele
.	`haplo` – a SNP haplotype, with an entry value 0 standing for B (ZERO A allele), 1 for A (ONE A allele)
`snpidx`	the SNP indices used in this classifier
`outofbag.acc`	the out-of-bag accuracy of this classifier

Author(s)

Xiuwen Zheng

See Also

`hlaAttrBagging`, `hlaParallelAttrBagging`, `hlaModelToObj`, `hlaModelFiles`, `hlaAttrBagClass`

hlaBED2Geno *Convert from PLINK BED format*

Description

To convert a PLINK BED file to an object of hlaSNPGenoClass.

Usage

```
hlaBED2Geno(bed.fn, fam.fn, bim.fn, rm.invalid.allele=FALSE,
    import.chr="xMHC", assembly="auto", verbose=TRUE)
```

Arguments

bed.fn	binary file, genotype information
fam.fn	family, individual information, etc
bim.fn	extended MAP file: two extra cols = allele names
rm.invalid.allele	if TRUE, remove SNPs with non-standard alleles (except A,G,C,T)
import.chr	the chromosome, "1" .. "22", "X", "Y", "XY", "MT", "xMHC", or "", where "xMHC" implies the extended MHC on chromosome 6, and "" for all SNPs; "6" for all SNPs on chromosome 6 for HLA; "19" for all SNPs on chromosome 19 for KIR
assembly	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
verbose	if TRUE, show information

Value

Return an object of hlaSNPGenoClass.

Author(s)

Xiuwen Zheng

See Also

hlaGeno2PED, hlaGDS2Geno

Examples

```
# Import a PLINK BED file
bed.fn <- system.file("extdata", "HapMap_CEU.bed", package="HIBAG")
fam.fn <- system.file("extdata", "HapMap_CEU.fam", package="HIBAG")
bim.fn <- system.file("extdata", "HapMap_CEU.bim", package="HIBAG")

hapmap.ceu <- hlaBED2Geno(bed.fn, fam.fn, bim.fn, assembly="hg19")
```

```
summary(hapmap.ceu)

# Or

hapmap.ceu <- hlaBED2Geno(bed.fn, fam.fn, bim.fn, assembly="hg19",
    rm.invalid.allele=TRUE, import.chr="6")
summary(hapmap.ceu)
```

`hlaCheckAllele` *Check SNP alleles*

Description

Check SNP reference and non-reference alleles.

Usage

```
hlaCheckAllele(allele1, allele2)
```

Arguments

`allele1`	two alleles for the first individual, like `c("A/G", "C/G")`
`allele2`	two alleles for the second individual, like `c("A/G", "C/G")`

Value

Return a logical vector, where `TRUE` indicates the alleles are matching at that locus.

Author(s)

Xiuwen Zheng

See Also

`hlaCheckSNPs`

Examples

```
hlaCheckAllele(c("A/G", "T/G", "0/A"), c("G/A", "C/A", "G/0"))
```

`hlaCheckSNPs` *Check the SNP predictors in a HIBAG model*

Description

Check the SNP predictors in a HIBAG model, by calculating the overlapping between the model and SNP genotypes.

Usage

```
hlaCheckSNPs(model, object,
   match.type=c("Position", "Pos+Allele", "RefSNP+Position", "RefSNP"), verbose=TRUE)
```

Arguments

`model`	an object of `hlaAttrBagClass`, or an object of `hlaAttrBagObj`
`object`	a genotype object of `hlaSNPGenoClass`, or a character vector like c("rs2523442", "rs9257863", ...)
`match.type`	`"RefSNP+Position"` (by default) – using both of RefSNP IDs and positions; `"RefSNP"` – using RefSNP IDs only; `"Position"` – using positions only
`verbose`	if TRUE, show information

Value

Return a `data.frame` for individual classifiers:

`NumOfValidSNP`	the number of non-missing SNPs in an individual classifier
`NumOfSNP`	the number of SNP predictors in an individual classifier
`fraction`	NumOfValidSNP / NumOfSNP

Author(s)

Xiuwen Zheng

See Also

`hlaAttrBagging`, `predict.hlaAttrBagClass`

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "DQB1"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")
```

```
# training genotypes
region <- 100   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

# train a HIBAG model
set.seed(1000)
model <- hlaAttrBagging(hla, train.geno, nclassifier=2)
print(model)


hlaCheckSNPs(model, train.geno)

# close the HIBAG model explicitly
hlaClose(model)
```

`hlaClose` *Dispose a model object*

Description

Release all resources stored in the `hlaAttrBagClass` object. The HIBAG package allows up to 256 `hlaAttrBagClass` objects stored in memory.

Usage

```
hlaClose(model)
```

Arguments

`model`	an object of `hlaAttrBagClass`

Value

None.

Author(s)

Xiuwen Zheng

See Also

`hlaAttrBagging`, `summary.hlaAttrBagClass`

`hlaCombineAllele` *Combine two datasets of HLA types*

Description

Combine two objects of `hlaAlleleClass`.

Usage

```
hlaCombineAllele(H1, H2)
```

Arguments

H1	the first `hlaAlleleClass` object
H2	the second `hlaAlleleClass` object

Value

Return `hlaAlleleClass`.

Author(s)

Xiuwen Zheng

See Also

`hlaAllele`, `hlaAlleleSubset`

Examples

```
head(HLA_Type_Table)
dim(HLA_Type_Table)  # 60 13

# make a "hlaAlleleClass" object
hla.id <- "C"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")
summary(hla)

subhla1 <- hlaAlleleSubset(hla,   1:100)
summary(subhla1)
subhla2 <- hlaAlleleSubset(hla, 201:300)
summary(subhla2)

H <- hlaCombineAllele(subhla1, subhla2)
summary(H)
```

hlaCombineModelObj *Combine two HIBAG models together*

Description

Merge two objects of hlaAttrBagObj together, which is useful for building an ensemble model in parallel.

Usage

```
hlaCombineModelObj(obj1, obj2)
```

Arguments

obj1	an object of hlaAttrBagObj
obj2	an object of hlaAttrBagObj

Value

Return an object of hlaAttrBagObj.

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, hlaModelFiles

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

# train a HIBAG model
```

```
set.seed(100)
m1 <- hlaAttrBagging(hla, train.geno, nclassifier=1)
m2 <- hlaAttrBagging(hla, train.geno, nclassifier=1)

m1.obj <- hlaModelToObj(m1)
m2.obj <- hlaModelToObj(m2)

m.obj <- hlaCombineModelObj(m1.obj, m2.obj)
summary(m.obj)
```

`hlaCompareAllele` *Evaluate prediction accuracies*

Description

To evaluate the overall accuracy, sensitivity, specificity, positive predictive value, negative predictive value.

Usage

```
hlaCompareAllele(TrueHLA, PredHLA, allele.limit=NULL, call.threshold=NaN,
    match.threshold=NaN, max.resolution="", output.individual=FALSE,
    verbose=TRUE)
```

Arguments

`TrueHLA`	an object of `hlaAlleleClass`, the true HLA types
`PredHLA`	an object of `hlaAlleleClass`, the predicted HLA types
`allele.limit`	a list of HLA alleles, the validation samples are limited to those having HLA alleles in `allele.limit`, or `NULL` for no limit. `allele.limit` could be character-type, `hlaAttrBagClass` or `hlaAttrBagObj`
`call.threshold`	the call threshold for posterior probability, i.e., call or no call is determined by whether `prob >= call.threshold` or not
`match.threshold`	the matching threshold for SNP haplotype similiarity, e.g., use 1% quantile of matching statistics of a training model
`max.resolution`	"2-digit", "4-digit", "6-digit", "8-digit", "allele", "protein", "2", "4", "6", "8", "full" or "": "allele" = "2-digit", "protein" = "4-digit", "full" and "" indicating no limit on resolution
`output.individual`	if TRUE, output accuracy for each individual
`verbose`	if TRUE, show information

Value

Return a `list(overall, confusion, detail)`, or `list(overall, confusion, detail, individual)` if `output.individual=TRUE`.

`overall` (data.frame):

`total.num.ind`	the total number of individuals
`crt.num.ind`	the number of individuals with correct HLA types
`crt.num.haplo`	the number of chromosomes with correct HLA alleles
`acc.ind`	the proportion of individuals with correctly predicted HLA types (i.e., both of alleles are correct, the accuracy of an individual is 0 or 1.)
`acc.haplo`	the proportion of chromosomes with correctly predicted HLA alleles (i.e., the accuracy of an individual is 0, 0.5 or 1, since an individual has two alleles.)
`call.threshold`	call threshold, if it is `NaN`, no call threshold is executed
`n.call`	the number of individuals with call
`call.rate`	overall call rate

`confusion` (matrix): a confusion matrix.

`detail` (data.frame):

`allele`	HLA alleles
`train.num`	the number of training haplotypes
`train.freq`	the training haplotype frequencies
`valid.num`	the number of validation haplotypes
`valid.freq`	the validation haplotype frequencies
`call.rate`	the call rates for HLA alleles
`accuracy`	allele accuracy
`sensitivity`	sensitivity
`specificity`	specificity
`ppv`	positive predictive value
`npv`	negative predictive value
`miscall`	the most likely miss-called alleles
`miscall.prop`	the proportions of the most likely miss-called allele in all miss-called alleles

`individual` (data.frame):

`sample.id`	sample id
`true.hla`	the true HLA type
`pred.hla`	the prediction of HLA type
`accuracy`	accuracy, 0, 0.5, or 1

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, predict.hlaAttrBagClass, hlaReport

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel=match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel=match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train a HIBAG model
set.seed(100)
model <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=4,
    verbose.detail=TRUE)
summary(model)

# validation
pred <- hlaPredict(model, test.geno)
# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0))
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0.5))
```

hlaConvSequence *Conversion From HLA Alleles to Amino Acid Sequences*

Description

Convert (P-coded or G-coded) HLA alleles to amino acid sequences.

Usage

```
hlaConvSequence(hla=character(), locus=NULL,
    method=c("protein", "protein_reference"),
    code=c("exact", "P.code", "G.code", "P.code.merge", "G.code.merge"),
    region=c("auto", "all", "P.code", "G.code"), release=c("v3.22.0"),
    replace=NULL)
```

Arguments

hla	characters, or an object of hlaAlleleClass, at least 4-digit or 2-field (P-coded) HLA alleles
locus	"A", "B", "C", "DRB1", "DQA1", "DQB1", "DPB1" or "DPA1"
method	"protein": returns protein sequence alignments, "protein_reference": returns the protein sequence alignment reference
code	"exact": requires full resolution; "P.code": allows ambiguous alleles according to P code; "G.code": allows ambiguous alleles according to G code; "P.code.merge" and "G.code.merge" merge multiple ambiguous allele sequences by masking unknown or ambiguous amino acid an asterisk
region	"all": returns all amino acid or nucleotide sequences; "P.code", "G.code": returns the exon 2 and 3 for HLA class I, and the exon 2 for HLA class II alleles; "auto": region="all" if code=="exact", region="P.code" if code=="P.code"|"P.code.merge", region="G.code" if code=="G.code"|"G.code.merge"
release	"v3.22.0" – IPD-IMGT/HLA 3.22.0 database (2015-10-07)
replace	NULL, or a character vector, e.g., c("09:02"="107:01"), any "09:02" will be replaced by "107:01". Due to the change of HLA nomenclature from 2010, HLA-DPB1*09:02 is replaced by DPB1*107:01

Details

The P or G codes for reporting of ambiguous allele typings can be found: http://hla.alleles.org/alleles/p_groups.html or http://hla.alleles.org/alleles/g_groups.html. The protein sequences for each HLA alleles could be found: http://hla.alleles.org/alleles/text_index.html.

Due to allelic ambiguity, multiple alleles are assigned to a 2-field P-coded allele or 3-field G-coded allele. For HLA Class I alleles, identity in the 'antigen binding domains' is based on identical protein sequences as encoded by exons 2 and 3. For HLA Class II alleles this is based on identical protein sequences as encoded by exon 2. P codes and G codes encode the same protein sequence

for the peptide binding domains (exon 2 and 3 for HLA class I and exon 2 only for HLA class II alleles).

1. the sequence is displayed as a hyphen "-" where it is identical to the reference.
2. an insertion or deletion is represented by a period ".".
3. an unknown or ambiguous position in the alignment is represented by an asterisk "*".
4. a capital X is used for the 'stop' codons in protein alignments.

http://hla.alleles.org/alleles/formats.html

HLA class I and II sequence alignments (Text Index): http://hla.alleles.org/alleles/text_index.html

WARNING: if you are not familiar with HLA nomenclature, you might consult with the package author or anyone who is familiar with HLA sequence alignments.

Value

Return an object of hlaAASeqClass or a list of characters. NULL or NA in the list indicates no matching.

Author(s)

Xiuwen Zheng

References

Robinson J, Halliwell JA, Hayhurst JH, Flicek P, Parham P, Marsh SGE: The IPD and IMGT/HLA database: allele variant databases. Nucleic Acids Research. 2015 43:D423-431

Robinson J, Malik A, Parham P, Bodmer JG, Marsh SGE: IMGT/HLA - a sequence database for the human major histocompatibility complex. Tissue Antigens. 2000 55:280-7

See Also

hlaAlleleSubset

Examples

```
hlaConvSequence(locus="A", method="protein_reference")

# exact match
hlaConvSequence(c("01:01", "02:02", "01:01:01G", "01:01:01:01", "07"),
    locus="A")

# allow ambiguity
hlaConvSequence(c("01:01", "02:02", "01:01:01G", "01:01:01:01", "07"),
    locus="A", code="P.code")
hlaConvSequence(c("01:01", "02:02", "01:01:01G", "01:01:01:01", "07"),
    locus="A", code="P.code.merge")
```

```
hlaConvSequence(locus="DPB1", method="protein_reference")
hlaConvSequence(c("09:01", "09:02"), locus="DPB1", replace=c("09:02"="107:01"))
hlaConvSequence(c("09:01", "09:02"), locus="DPB1", code="P.code",
    replace=c("09:02"="107:01"))
hlaConvSequence(c("09:01", "09:02"), locus="DPB1", code="P.code.merge",
    replace=c("09:02"="107:01"))


hlaConvSequence(locus="DQB1", method="protein_reference")
hlaConvSequence(c("05:01:01:01", "06:01:01"), locus="DQB1")
hlaConvSequence(c("05:01", "06:01"), locus="DQB1", code="P.code")
hlaConvSequence(c("05:01", "06:01"), locus="DQB1", code="P.code.merge")



hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

(v <- hlaConvSequence(hla, code="P.code.merge"))
summary(v)

v <- hlaConvSequence(hla, code="P.code.merge", region="all")
summary(v)



hla.id <- "DQB1"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

(v <- hlaConvSequence(hla, code="P.code.merge"))
summary(v)

v <- hlaConvSequence(hla, code="P.code.merge", region="all")
summary(v)
```

`hlaDistance` *Distance matrix of HLA alleles*

Description

To calculate the distance matrix of HLA alleles from a HIBAG model.

Usage

```
hlaDistance(model)
```

Arguments

model	a model of hlaAttrBagClass or hlaAttrBagObj

Value

Return a distance matrix with row and column names for HLA alleles.

Author(s)

Xiuwen Zheng

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# flanking genotypes
train.geno <- hlaGenoSubsetFlank(HapMap_CEU_Geno, hla.id, 500000)
summary(train.geno)

# train a HIBAG model
set.seed(100)
model <- hlaAttrBagging(hla, train.geno, nclassifier=10)
summary(model)

# distance matrix
d <- hlaDistance(model)

# draw
p <- hclust(as.dist(d))
plot(p, xlab="HLA alleles")
```

---

hlaFlankingSNP *SNP IDs or SNP genotypes in Flanking Region*

Description

To get SNPs in the flanking region of a specified HLA/KIR locus.

Usage

```
hlaFlankingSNP(snp.id, position, locus, flank.bp=500000L, assembly="auto",
    pos.mid=NA_integer_)
hlaGenoSubsetFlank(genoobj, locus="any", flank.bp=500000L, assembly="auto",
    pos.mid=NA_integer_)
```

Arguments

snp.id	a vector of SNP IDs
genoobj	a genotype object of hlaSNPGenoClass
position	a vector of positions
locus	the name of HLA locus, or "any" for other genes and using pos.mid
flank.bp	the size of flanking region on each side in basepair
assembly	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
pos.mid	the middle position of the flanking region

Details

hla.id is "A", "B", "C", "DRB1", "DRB5", "DQA1", "DQB1", "DPB1" or "any".

Value

Return selected SNP IDs from snp.id.

Author(s)

Xiuwen Zheng

See Also

hlaGenoSubset, hlaLociInfo

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel  = match(snpid, HapMap_CEU_Geno$snp.id))
summary(train.geno)
```

```
# or using hlaGenoSubsetFlank
train.geno <- hlaGenoSubsetFlank(HapMap_CEU_Geno, hla.id, region*1000)
summary(train.geno)


## customize positions
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    "any", 500*1000, pos.mid=29954010)
```

---

hlaGDS2Geno *Import genotypes from a GDS file*

Description

To convert a SNPRelate or SeqArray GDS file to an object of `hlaSNPGenoClass`.

Usage

```
hlaGDS2Geno(gds.fn, rm.invalid.allele=FALSE, import.chr="xMHC", assembly="auto",
    verbose=TRUE)
```

Arguments

`gds.fn`	a file name for the GDS file defined in the SNPRelate or SeqArray package
`rm.invalid.allele`	if TRUE, remove SNPs with non-standard alleles (except A,G,C,T)
`import.chr`	the chromosome, "1" .. "22", "X", "Y", "XY", "MT", "xMHC", or "", where "xMHC" implies the extended MHC on chromosome 6, and "" for all SNPs
`assembly`	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
`verbose`	if TRUE, show information

Value

Return an object of `hlaSNPGenoClass`.

Author(s)

Xiuwen Zheng

See Also

`hlaGeno2PED`, `hlaBED2Geno`

Examples

```
# Import a SNP GDS file
fn <- system.file("extdata", "HapMap_CEU_Chr6.gds", package="HIBAG")

geno <- hlaGDS2Geno(fn, assembly="hg18", rm.invalid.allele=TRUE)

summary(geno)
```

---

hlaGeno2PED *Convert to PLINK PED format*

Description

Convert an object of hlaSNPGenoClass to a file of PLINK PED format.

Usage

```
hlaGeno2PED(geno, out.fn)
```

Arguments

geno	a genotype object of hlaSNPGenoClass
out.fn	the file name of output ped file

Details

Two files ".map" and ".ped" are created.

Value

None.

Author(s)

Xiuwen Zheng

See Also

hlaBED2Geno

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    max.resolution="4-digit", locus=hla.id, assembly="hg19")

# training genotypes
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")

train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

hlaGeno2PED(train.geno, "test")


# delete the temporary files
unlink(c("test.map", "test.ped"), force=TRUE)
```

---

hlaGenoAFreq *Allele Frequency*

Description

To calculate the allele frequencies from genotypes or haplotypes.

Usage

```
hlaGenoAFreq(obj)
```

Arguments

obj	an object of hlaSNPGenoClass

Value

Return allele frequecies.

Author(s)

Xiuwen Zheng

See Also

hlaGenoAFreq, hlaGenoMFreq, hlaGenoMRate, hlaGenoMRate_Samp

Examples

```
summary(HapMap_CEU_Geno)

summary(hlaGenoAFreq(HapMap_CEU_Geno))
```

`hlaGenoCombine` *Combine two genotypic data sets into one*

Description

To combine two genotypic data sets into one dataset.

Usage

```
hlaGenoCombine(geno1, geno2,
    match.type=c("Position", "Pos+Allele", "RefSNP+Position", "RefSNP"),
    allele.check=TRUE, same.strand=FALSE, verbose=TRUE)
```

Arguments

`geno1`	the first genotype object of `hlaSNPGenoClass`
`geno2`	the second genotype object of `hlaSNPGenoClass`
`match.type`	`"RefSNP+Position"` (by default) – using both of RefSNP IDs and positions; `"RefSNP"` – using RefSNP IDs only; `"Position"` – using positions only
`allele.check`	if `TRUE`, call `hlaGenoSwitchStrand` to check and then switch allele pairs if needed
`same.strand`	`TRUE` assuming alleles are on the same strand (e.g., forward strand); otherwise, `FALSE` not assuming whether on the same strand or not
`verbose`	show information, if TRUE

Details

The function merges two SNP dataset `geno1` and `geno2`, and returns a SNP dataset consisting of the SNP intersect between `geno1` and `geno2`, and having the same SNP information (allele and position) as `geno1`.

Value

An object of `hlaSNPGenoClass`.

Author(s)

Xiuwen Zheng

See Also

`hlaMakeSNPGeno`, `hlaGenoSubset`

Examples

```
# import a PLINK BED file
bed.fn <- system.file("extdata", "HapMap_CEU.bed", package="HIBAG")
fam.fn <- system.file("extdata", "HapMap_CEU.fam", package="HIBAG")
bim.fn <- system.file("extdata", "HapMap_CEU.bim", package="HIBAG")
hapmap.ceu <- hlaBED2Geno(bed.fn, fam.fn, bim.fn, assembly="hg19")

# combine two datasets together
geno <- hlaGenoCombine(HapMap_CEU_Geno, hapmap.ceu)
summary(geno)
```

---

hlaGenoLD *Composite Linkage Disequilibrium*

Description

To calculate composite linkage disequilibrium (r2) between HLA locus and SNP markers.

Usage

```
hlaGenoLD(hla, geno)
```

Arguments

hla	an object of hlaAlleleClass
geno	an object of hlaSNPGenoClass, or a vector or matrix for SNP data

Value

Return a vector of linkage disequilibrium (r2) for each SNP marker.

Author(s)

Xiuwen Zheng

References

Weir BS, Cockerham CC: Complete characterization of disequilibrium at two loci; in Feldman MW (ed): Mathematical Evolutionary Theory. Princeton, NJ: Princeton University Press, 1989.

Zaykin, D. V., Pudovkin, A., and Weir, B. S. (2008). Correlation-based inference for linkage disequilibrium with multiple alleles. Genetics 180, 533-545.

Examples

```
# plot linkage disequilibrium
ymax <- 0.16
plot(NaN, NaN, xlab="SNP Position (in KB)",
    ylab="Composite Linkage Disequilibrium (r2)",
    xlim=range(HapMap_CEU_Geno$snp.position)/1000, ylim=c(0, ymax),
    main="Major Histocompatibility Complex")

hla.list <- c("A", "C", "DQA1")
col.list <- 1:3

# for-loop
for (i in 1:3)
{
    hla.id <- hla.list[i]

    # make a "hlaAlleleClass" object
    hla <- hlaAllele(HLA_Type_Table$sample.id,
        H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
        H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
        locus=hla.id, assembly="hg19")

    # linkage disequilibrium between HLA locus and SNP markers
    ld <- hlaGenoLD(hla, HapMap_CEU_Geno)

    # draw
    points(HapMap_CEU_Geno$snp.position/1000, ld, pch="*", col=i)
    x <- (hla$pos.start/1000 + hla$pos.end/1000)/2
    abline(v=x, col=col.list[i], lty=3, lwd=2.5)
    points(x, ymax, pch=25, col=7, bg=col.list[i], cex=1.5)
}
legend("topleft", col=col.list, pt.bg=col.list, text.col=col.list, pch=25,
    legend=paste("HLA -", hla.list))
```

`hlaGenoMFreq` *Minor Allele Frequency*

Description

To calculate the minor allele frequencies from genotypes or haplotypes.

Usage

```
hlaGenoMFreq(obj)
```

Arguments

`obj`	an object of `hlaSNPGenoClass`

Value

Return minor allele frequecies.

Author(s)

Xiuwen Zheng

See Also

hlaGenoAFreq, hlaGenoMFreq, hlaGenoMRate, hlaGenoMRate_Samp

Examples

```
summary(HapMap_CEU_Geno)

summary(hlaGenoMFreq(HapMap_CEU_Geno))
```

---

hlaGenoMRate *Missing Rates Per SNP*

Description

To calculate the missing rates from genotypes or haplotypes per SNP.

Usage

```
hlaGenoMRate(obj)
```

Arguments

obj	an object of hlaSNPGenoClass

Value

Return missing rates per SNP.

Author(s)

Xiuwen Zheng

See Also

hlaGenoAFreq, hlaGenoMFreq, hlaGenoMRate, hlaGenoMRate_Samp

Examples

```
summary(HapMap_CEU_Geno)

summary(hlaGenoMRate(HapMap_CEU_Geno))
```

hlaGenoMRate_Samp *Missing Rates Per Sample*

Description

To calculate the missing rates from genotypes or haplotypes per sample.

Usage

```
hlaGenoMRate_Samp(obj)
```

Arguments

obj	an object of hlaSNPGenoClass

Value

Return missing rates per sample.

Author(s)

Xiuwen Zheng

See Also

hlaGenoAFreq, hlaGenoMFreq, hlaGenoMRate, hlaGenoMRate_Samp

Examples

```
summary(HapMap_CEU_Geno)

summary(hlaGenoMRate_Samp(HapMap_CEU_Geno))
```

hlaGenoSubset *Get a subset of genotypes*

Description

To get a subset of genotypes from a hlaSNPGenoClass object.

Usage

```
hlaGenoSubset(genoobj, samp.sel=NULL, snp.sel=NULL, snp.id=NULL)
```

Arguments

genoobj	a genotype object of hlaSNPGenoClass
samp.sel	a logical vector, or an integer vector of indices
snp.sel	a logical vector, or an integer vector of indices
snp.id	SNP IDs to be selected, or NULL

Details

genoobj$genotype is a numeric matrix, with an entry value 0 standing for BB (ZERO A allele), 1 for AB (ONE A allele), 2 for AA (TWO A alleles) and others for missing values (missing genotypes are usually set to be NA).

Value

Return a hlaSNPGenoClass object, and it is a list:

genotype	a genotype matrix, "# of SNPs" - by - "# of individuals"
sample.id	a vector of sample IDs
snp.id	a vector of SNP IDs
snp.position	a vector of SNP positions in basepair
snp.allele	a vector of characters with the format of "A allele/B allele"
assembly	optional, human genome information

Author(s)

Xiuwen Zheng

See Also

hlaMakeSNPGeno, hlaGenoCombine

Examples

```
summary(HapMap_CEU_Geno)

geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = (hlaGenoMFreq(HapMap_CEU_Geno)>0.10))
summary(geno)
```

`hlaGenoSwitchStrand` *Allele flipping if needed*

Description

Determine the ordered pair of A and B alleles, using the allele information provided by `template`.

Usage

```
hlaGenoSwitchStrand(target, template,
    match.type=c("Position", "Pos+Allele", "RefSNP+Position", "RefSNP"),
    same.strand=FALSE, verbose=TRUE)
```

Arguments

`target`	an object of `hlaSNPGenoClass`
`template`	a genotypic object of `hlaSNPGenoClass`, a model object of `hlaAttrBagClass` or a model object of `hlaAttrBagObj`
`match.type`	`"RefSNP+Position"` (by default) – using both of RefSNP IDs and positions; `"RefSNP"` – using RefSNP IDs only; `"Position"` – using positions only
`same.strand`	`TRUE` assuming alleles are on the same strand (e.g., forward strand); otherwise, `FALSE` not assuming whether on the same strand or not
`verbose`	show information, if TRUE

Details

The A/B pairs of `target` are determined using the information from `template`.

Value

Return a `hlaSNPGenoClass` object consisting of the SNP intersect between `target` and `template`.

Author(s)

Xiuwen Zheng

See Also

`hlaMakeSNPGeno`, `hlaGenoSubset`

Examples

```
summary(HapMap_CEU_Geno)
# A/C A/G C/T G/T
# 136 655 632 141

# import a PLINK BED file
bed.fn <- system.file("extdata", "HapMap_CEU.bed", package="HIBAG")
fam.fn <- system.file("extdata", "HapMap_CEU.fam", package="HIBAG")
bim.fn <- system.file("extdata", "HapMap_CEU.bim", package="HIBAG")
hapmap.ceu <- hlaBED2Geno(bed.fn, fam.fn, bim.fn, assembly="hg19")
summary(hapmap.ceu)
# A/C  A/G  A/T  C/G  C/T  G/T
# 332 1567   64  111 1510  348

# combine two datasets together
geno <- hlaGenoSwitchStrand(HapMap_CEU_Geno, hapmap.ceu)
summary(geno)
# There are 1564 SNPs in common.
# The allele pairs of 763 SNPs need to be switched.
# A/C A/G C/T G/T
# 104 505 496 109
```

---

hlaLDMatrix *Composite Linkage Disequilibrium in a Region*

Description

To calculate composite linkage disequilibrium (r2) among SNPs within a region.

Usage

```
hlaLDMatrix(geno, loci=NULL, maf=0.01, assembly="auto", draw=TRUE,
    verbose=TRUE)
```

Arguments

geno	an object of hlaSNPGenoClass
maf	MAF filter >= maf
loci	NULL or a character vector, e.g., "A", "B"
assembly	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
draw	if TRUE, return a ggplot2 object
verbose	if TRUE, show information

Value

Return a ggplot2 object if draw=TRUE or a matrix correlation.

Author(s)

Xiuwen Zheng

References

Weir BS, Cockerham CC: Complete characterization of disequilibrium at two loci; in Feldman MW (ed): Mathematical Evolutionary Theory. Princeton, NJ: Princeton University Press, 1989.

Examples

```
region <- 500*1000   # basepair
geno <- hlaGenoSubsetFlank(HapMap_CEU_Geno, "A", region)
summary(geno)

hlaLDMatrix(geno, "A")
```

---

hlaLociInfo *HLA/KIR Locus Information*

Description

To get the starting and ending positions in basepair of HLA/KIR loci.

Usage

```
hlaLociInfo(assembly=c("auto", "auto-silent", "hg18", "hg19", "hg38",
    "unknown"))
```

Arguments

assembly	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning

Value

Return a data frame include the genomic locations.

Author(s)

Xiuwen Zheng

References

NCBI Resources: https://www.ncbi.nlm.nih.gov/gene, HLA Nomenclature: http://hla.alleles.org/genes/index.html

Examples

```
hlaLociInfo()
```

`hlaMakeSNPGeno` *Make a SNP genotype object*

Description

To create a `hlaSNPGenoClass` object (SNP genotypic object).

Usage

```
hlaMakeSNPGeno(genotype, sample.id, snp.id, snp.position,
    A.allele, B.allele, assembly="auto")
```

Arguments

`genotype`	a genotype matrix, "# of SNPs" - by - "# of individuals"
`sample.id`	a vector of sample IDs
`snp.id`	a vector of SNP IDs
`snp.position`	a vector of SNP positions
`A.allele`	a vector of A alleles, A is usually defined as a minor or alternative allele
`B.allele`	a vector of B alleles, B is usually defined as a major or reference allele
`assembly`	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning

Details

`genotype` is a numeric matrix, with an entry value 0 standing for BB (ZERO A allele), 1 for AB (ONE A allele), 2 for AA (TWO A alleles) and others for missing values (missing genotypes are usually set to be NA).

Value

Return a `hlaSNPGenoClass` object, and it is a list:

`genotype`	a genotype matrix, "# of SNPs" - by - "# of individuals"
`sample.id`	a vector of sample IDs
`snp.id`	a vector of SNP IDs
`snp.position`	a vector of SNP positions in basepair
`snp.allele`	a vector of characters with the format of "A allele/B allele"
`assembly`	the human genome reference

Author(s)

Xiuwen Zheng

See Also

hlaGenoSubset, hlaGenoCombine

Examples

```
summary(HapMap_CEU_Geno)

allele <- strsplit(HapMap_CEU_Geno$snp.allele, "/")
A.allele <- sapply(allele, function(x) { x[1] })
B.allele <- sapply(allele, function(x) { x[2] })

geno <- hlaMakeSNPGeno(HapMap_CEU_Geno$genotype, HapMap_CEU_Geno$sample.id,
    HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position, A.allele, B.allele,
    assembly="hg19")

summary(geno)
```

---

hlaModelFiles *Load a model object from files*

Description

To load HIBAG models from a list of files, and merge all together.

Usage

```
hlaModelFiles(fn.list, action.missingfile=c("ignore", "stop"), verbose=TRUE)
```

Arguments

fn.list	a vector of file names
action.missingfile	"ignore", ignore the missing files, by default; "stop", stop if missing
verbose	if TRUE, show information

Value

Return hlaAttrBagObj.

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, hlaModelToObj

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "C"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 100   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel = match(hla$value$sample.id, HapMap_CEU_Geno$sample.id))

#
# train HIBAG models
#
set.seed(1000)

model1 <- hlaAttrBagging(hla, train.geno, nclassifier=1)
mobj1 <- hlaModelToObj(model1)
save(mobj1, file="tm1.RData")

model2 <- hlaAttrBagging(hla, train.geno, nclassifier=1)
mobj2 <- hlaModelToObj(model2)
save(mobj2, file="tm2.RData")

model3 <- hlaAttrBagging(hla, train.geno, nclassifier=1)
mobj3 <- hlaModelToObj(model3)
save(mobj3, file="tm3.RData")

# load all of mobj1, mobj2 and mobj3
mobj <- hlaModelFiles(c("tm1.RData", "tm2.RData", "tm3.RData"))
summary(mobj)


# delete the temporary files
unlink(c("tm1.RData", "tm2.RData", "tm3.RData"), force=TRUE)
```

`hlaModelFromObj` *Conversion between the in-memory model and the object that can be saved in a file*

Description

Build a model `hlaAttrBagClass` from an object of `hlaAttrBagObj` which is stored in an R object file, or convert `hlaAttrBagClass` to `hlaAttrBagObj`.

Usage

```
hlaModelFromObj(obj)
hlaModelToObj(model)
```

Arguments

obj	an object of hlaAttrBagObj
model	an object of hlaAttrBagClass

Value

hlaModelFromObj returns hlaAttrBagClass, and hlaModelToObj returns hlaAttrBagObj.

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "DQB1"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 100   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

# train a HIBAG model
set.seed(1000)
model <- hlaAttrBagging(hla, train.geno, nclassifier=2)
print(model)

mobj <- hlaModelToObj(model)

is(model)
is(mobj)


# close the HIBAG model explicitly
hlaClose(model)
```

hlaOutOfBag *Out-of-bag estimation of overall accuracy, per-allele sensitivity, etc*

Description

Out-of-bag estimation of overall accuracy, per-allele sensitivity, specificity, positive predictive value, negative predictive value and call rate.

Usage

```
hlaOutOfBag(model, hla, snp, call.threshold=NaN, verbose=TRUE)
```

Arguments

model	an object of hlaAttrBagClass or hlaAttrBagObj
hla	the training HLA types, an object of hlaAlleleClass
snp	the training SNP genotypes, an object of hlaSNPGenoClass
call.threshold	the specified call threshold; if NaN, no threshold is used
verbose	if TRUE, show information

Value

Return hlaAlleleClass.

Author(s)

Xiuwen Zheng

See Also

hlaCompareAllele, hlaReport

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
```

```
geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel = match(hla$value$sample.id, HapMap_CEU_Geno$sample.id))

# train a HIBAG model
set.seed(100)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hla, geno, nclassifier=4)
summary(model)

# out-of-bag estimation
(comp <- hlaOutOfBag(model, hla, geno, call.threshold=NaN, verbose=TRUE))

# report
hlaReport(comp, type="txt")

hlaReport(comp, type="tex")

hlaReport(comp, type="html")
```

`hlaParallelAttrBagging` *Build a HIBAG model via parallel computation*

Description

To build a HIBAG model for predicting HLA types via parallel computation.

Usage

```
hlaParallelAttrBagging(cl, hla, snp, auto.save="",
    nclassifier=100L, mtry=c("sqrt", "all", "one"), prune=TRUE, na.rm=TRUE,
    mono.rm=TRUE, maf=NaN, stop.cluster=FALSE, verbose=TRUE,
    verbose.detail=FALSE)
```

Arguments

`cl`	`NULL`, `FALSE`, `TRUE`, an integer, or a cluster object created by the parallel-package; if `NULL` or `FALSE`, use the serial implementation; if `TRUE`, use the number of threads returned from `RcppParallel::defaultNumThreads()` (by default using all threads); if an integer, specify the number of threads; When `cl` is `TRUE` or an integer, the multithreading implementation will be used; when `cl` is a cluster, the multi-processing implementation will be used where each individual classifier is built within a child process
`hla`	training HLA types, an object of `hlaAlleleClass`
`snp`	training SNP genotypes, an object of `hlaSNPGenoClass`
`auto.save`	specify a autosaved file name for an R object (.rda, .RData or .rds); "", no file saving; see details

nclassifier	the total number of individual classifiers
mtry	a character or a numeric value, the number of variables randomly sampled as candidates for each selection. See details
prune	if TRUE, to perform a parsimonious forward variable selection, otherwise, exhaustive forward variable selection. See details
na.rm	if TRUE, remove the samples with missing HLA types
mono.rm	if TRUE, remove monomorphic SNPs
maf	MAF threshold for SNP filter, excluding any SNP with MAF < maf
stop.cluster	TRUE: stop cluster nodes after completing the calculation
verbose	if TRUE, show information
verbose.detail	if TRUE, show more information

Details

mtry (the number of variables randomly sampled as candidates for each selection): "sqrt", using the square root of the total number of candidate SNPs; "all", using all candidate SNPs; "one", using one SNP; an integer, specifying the number of candidate SNPs; 0 < r < 1, the number of candidate SNPs is "r * the total number of SNPs".

prune: there is no significant difference on accuracy between parsimonious and exhaustive forward variable selections. If prune = TRUE, the searching algorithm performs a parsimonious forward variable selection: if a new SNP predictor reduces the current out-of-bag accuracy, then it is removed from the candidate SNP set for future searching. Parsimonious selection helps to improve the computational efficiency by reducing the searching times of non-informative SNP markers.

An autosave function is available in hlaParallelAttrBagging when an new individual classifier is built internally without completing the ensemble.

Value

Return an object of hlaAttrBagClass if auto.save="", and NULL otherwise.

Author(s)

Xiuwen Zheng

References

Zheng X, Shen J, Cox C, Wakefield J, Ehm M, Nelson M, Weir BS; HIBAG – HLA Genotype Imputation with Attribute Bagging. Pharmacogenomics Journal. doi: 10.1038/tpj.2013.18. https://www.nature.com/articles/tpj201318

See Also

hlaAttrBagging, hlaClose, hlaSetKernelTarget

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel = match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))


##############################################################################
# Multithreading

set.seed(100)

# train a HIBAG model in parallel with 2 cores
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaParallelAttrBagging(2, hlatab$training, train.geno, nclassifier=4)


##############################################################################
# Multicore & autosave

library(parallel)

# choose an appropriate cluster size, e.g., 2
cl <- makeCluster(2)
set.seed(100)

# train a HIBAG model in parallel
```

```
# please use "nclassifier=100" when you use HIBAG for real data
hlaParallelAttrBagging(cl, hlatab$training, train.geno, nclassifier=4,
    auto.save="tmp_model.RData", stop.cluster=TRUE)

mobj <- get(load("tmp_model.RData"))
summary(mobj)
model <- hlaModelFromObj(mobj)

# validation
pred <- hlaPredict(model, test.geno)
summary(pred)

# compare
hlaCompareAllele(hlatab$validation, pred, allele.limit=model)$overall


# since 'stop.cluster=TRUE' used in 'hlaParallelAttrBagging'
# need a new cluster
cl <- makeCluster(2)

pred <- hlaPredict(model, test.geno, cl=cl)
summary(pred)

# stop parallel nodes
stopCluster(cl)


# delete the temporary file
unlink(c("tmp_model.RData"), force=TRUE)
```

`hlaPredict` *HIBAG model prediction (in parallel)*

Description

To predict HLA type based on a HIBAG model (in parallel).

Usage

```
hlaPredict(object, snp, cl=FALSE,
    type=c("response+dosage", "response", "prob", "response+prob"),
    vote=c("prob", "majority"), allele.check=TRUE,
    match.type=c("Position", "Pos+Allele", "RefSNP+Position", "RefSNP"),
    same.strand=FALSE, verbose=TRUE, verbose.match=TRUE)
## S3 method for class 'hlaAttrBagClass'
predict(object, snp, cl=FALSE,
    type=c("response+dosage", "response", "prob", "response+prob"),
    vote=c("prob", "majority"), allele.check=TRUE,
    match.type=c("Position", "Pos+Allele", "RefSNP+Position", "RefSNP"),
    same.strand=FALSE, verbose=TRUE, verbose.match=TRUE, ...)
```

Arguments

object	a model of hlaAttrBagClass
snp	a genotypic object of hlaSNPGenoClass
cl	FALSE, TRUE, an integer, or a cluster object created by the parallel-package; if FALSE, use the serial implementation; if TRUE, use the number of threads returned from RcppParallel::defaultNumThreads() (by default using all threads); if an integer, specify the number of threads
type	"response+dosage": return the best-guess types and dosages for each allele (by default); "response": return the best-guess types with its posterior probability; "prob": return a matrix for all posterior probabilities; "response+prob": return the best-guess, dosages and all posterior probabilities
vote	"prob" (default behavior) – make a prediction based on the averaged posterior probabilities from all individual classifiers; "majority" – majority voting from all individual classifiers, where each classifier votes for an HLA type
allele.check	if TRUE, check and then switch allele pairs if needed
match.type	"Position" – use positions only (by default); "RefSNP+Position" – use both of SNP IDs and positions; "RefSNP" – using SNP IDs only
same.strand	TRUE assuming alleles are on the same strand (e.g., forward strand); otherwise, FALSE not assuming whether on the same strand or not
verbose	if TRUE, show information
verbose.match	if TRUE, show missing SNP proportions for different match.type
...	unused

Details

If more than 50% of SNP predictors are missing, a warning will be given.

When match.type="RefSNP+Position", the matching of SNPs requires both SNP IDs and positions. A lower missing fraction maybe gained by matching SNP IDs or positions only. Call hlaPredict(..., match.type="RefSNP") or hlaPredict(..., match.type="Position") for this purpose. It could be safe to assume that the SNPs with the same positions on the same genome reference (e.g., hg19) are the same variant albeit the different SNP IDs. Any concern about SNP mismatching should be emailed to the genotyping platform provider.

Value

Return a hlaAlleleClass object with posterior probabilities of predicted HLA types, or a matrix of pairwise possible HLA types with all posterior probabilities. If type = "response+prob", return a hlaAlleleClass object with a matrix of postprob for the probabilities of all pairs of alleles. If a probability matrix is returned, colnames is sample.id and rownames is an unordered pair of HLA alleles.

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, hlaAllele, hlaCompareAllele, hlaParallelAttrBagging, hlaSetKernelTarget, hlaAlleleToVCF

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel=match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel=match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train a HIBAG model
set.seed(100)
model <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=4,
    verbose.detail=TRUE)
summary(model)

# validation
pred <- hlaPredict(model, test.geno, type="response+dosage")
pred

head(pred$value)
pred$dosage[, 1:4]  # a dosage matrix

# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0))
```

```
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0.5))
```

---

hlaPredMerge *Merge prediction results from multiple HIBAG models*

Description

Return an object of hlaAlleleClass, which contains predicted HLA types.

Usage

```
hlaPredMerge(..., weight=NULL, equivalence=NULL, use.matching=TRUE,
    ret.dosage=TRUE, ret.postprob=FALSE, max.resolution="", rm.suffix=FALSE,
    verbose=TRUE)
```

Arguments

...	The object(s) of hlaAlleleClass, having a field of 'postprob', and returned by hlaPredict(..., type="response+prob")
weight	the weight used for each prediction; if NULL, equal weights to be used; or set the weight vector to be the training sample sizes
equivalence	a data.frame with two columns, the first column for new equivalent alleles, and the second for the alleles possibly exist in the object(s) passed to this function; there is no replace if the allele is not found in the second column
use.matching	if TRUE, use actual probabilities (i.e., poster prob. * matching) for merging; otherwise, use poster prob. instead. use.matching=TRUE is recommended.
ret.dosage	if TRUE, return dosages
ret.postprob	if TRUE, return average posterior probabilities
max.resolution	"2-digit", "1-field", "4-digit", "2-field", "6-digit", "3-field", "8-digit", "4-field", "allele", "protein", "full", "none", or "": "allele" = "2-digit"; "protein" = "4-digit"; "full", "none" or "" for no limit on resolution
rm.suffix	whether remove the non-digit suffix in the last field, e.g., for "01:22N", "N" is a non-digit suffix
verbose	if TRUE, show information

Details

Calculate a new probability matrix for each pair of HLA alleles, by averaging (posterior) probabilities from all models with specified weights. If equivalence is specified, multiple alleles might be collapsed into one class.

Value

Return a hlaAlleleClass object.

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, hlaAllele, predict.hlaAttrBagClass

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel=match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel=match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train HIBAG models
set.seed(100)

# please use "nclassifier=100" when you use HIBAG for real data
m1 <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=2,
    verbose.detail=TRUE)
m2 <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=2,
    verbose.detail=TRUE)


# validation
pd1 <- hlaPredict(m1, test.geno, type="response+prob")
pd2 <- hlaPredict(m2, test.geno, type="response+prob")
```

```
hlaCompareAllele(hlatab$validation, pd1)$overall
hlaCompareAllele(hlatab$validation, pd2)$overall

# merge predictions from multiple models, by voting from all classifiers
pd <- hlaPredMerge(pd1, pd2)
pd

hlaCompareAllele(hlatab$validation, pd)$overall

# collapse to 2-digit
pd <- hlaPredMerge(pd1, pd2, max.resolution="2-digit", ret.postprob=FALSE)
pd
```

`hlaPublish` *Finalize a HIBAG model*

Description

Finalize a HIBAG model by removing unused SNP predictors and adding appendix information (platform, training set, authors, warning, etc)

Usage

```
hlaPublish(mobj, platform=NULL, information=NULL, warning=NULL,
    rm.unused.snp=TRUE, anonymize=TRUE, verbose=TRUE)
```

Arguments

`mobj`	an object of `hlaAttrBagObj` or `hlaAttrBagClass`
`platform`	the text of platform information
`information`	the other information, like authors
`warning`	any warning message
`rm.unused.snp`	if `TRUE`, remove unused SNPs from the model
`anonymize`	if `TRUE`, remove sample IDs
`verbose`	if TRUE, show information

Value

Returns a new object of `hlaAttrBagObj`.

Author(s)

Xiuwen Zheng

See Also

`hlaModelFromObj`, `hlaModelToObj`

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 250   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel = match(hla$value$sample.id, HapMap_CEU_Geno$sample.id))


#
# train a HIBAG model
#
set.seed(1000)

# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hla, train.geno, nclassifier=2, verbose.detail=TRUE)
summary(model)
length(model$snp.id)

mobj <- hlaPublish(model,
    platform = "Illumina 1M Duo",
    information = "Training set -- HapMap Phase II")
model2 <- hlaModelFromObj(mobj)
length(mobj$snp.id)
mobj$appendix
summary(mobj)

p1 <- hlaPredict(model, train.geno)
p2 <- hlaPredict(model2, train.geno)

# check
cbind(p1$value, p2$value)
```

`hlaReport` *Format a report*

Description

Create a report for evaluating prediction accuracies.

Usage

```
hlaReport(object, export.fn="", type=c("txt", "tex", "html", "markdown"),
    header=TRUE)
```

Arguments

object	an object returned by hlaCompareAllele
export.fn	a file name for output, or "" for stdout
type	"txt" – tab-delimited text format; "tex" – tex format using the 'longtable' package; "html" – html file
header	if TRUE, output the header of text file associated corresponding format

Value

None.

Author(s)

Xiuwen Zheng

See Also

hlaCompareAllele

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id),
```

```
    samp.sel = match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train a HIBAG model
set.seed(100)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=4,
    verbose.detail=TRUE)
summary(model)

# validation
pred <- hlaPredict(model, test.geno)
# compare
(comp <- hlaCompareAllele(hlatab$validation, pred, allele.limit=model,
    call.threshold=0))


# report
hlaReport(comp, type="txt")

hlaReport(comp, type="tex")

hlaReport(comp, type="html")

hlaReport(comp, type="markdown")
```

`hlaReportPlot` *Format a report with figures*

Description

Create figures for evaluating prediction accuracies.

Usage

```
hlaReportPlot(PredHLA=NULL, TrueHLA=NULL, model=NULL,
    fig=c("matching", "call.rate", "call.threshold"), match.threshold=NaN,
    log_scale=TRUE)
```

Arguments

`PredHLA`	NULL, an object of `hlaAlleleClass`, the predicted HLA types
`TrueHLA`	NULL, an object of `hlaAlleleClass`, the true HLA types
`model`	NULL, or a model of `hlaAttrBagClass`

fig	"matching": violin plot for matching measurements; "call.rate": relationship between accuracy and call rate; "call.threshold": relationship between accuracy and call threshold
match.threshold	the threshold for matching proportion
log_scale	if TRUE, use log scale for matching violin plot

Value

Return a ggplot2 object.

Author(s)

Xiuwen Zheng

See Also

hlaReport

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)

# SNP predictors within the flanking region on each side
region <- 500   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
length(snpid)  # 275

# training and validation genotypes
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id),
    samp.sel = match(hlatab$training$value$sample.id,
    HapMap_CEU_Geno$sample.id))
test.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    samp.sel=match(hlatab$validation$value$sample.id,
    HapMap_CEU_Geno$sample.id))

# train a HIBAG model
```

```
set.seed(100)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hlatab$training, train.geno, nclassifier=4,
    verbose.detail=TRUE)
summary(model)

# validation
pred <- hlaPredict(model, test.geno)


# visualize
hlaReportPlot(pred, fig="matching")

hlaReportPlot(model=model, fig="matching")

hlaReportPlot(pred, model=model, fig="matching")

hlaReportPlot(pred, hlatab$validation, fig="call.rate")

hlaReportPlot(pred, hlatab$validation, fig="call.threshold")
```

`hlaSampleAllele` *Get sample IDs from HLA types with a filter*

Description

Get sample IDs from HLA types limited to a set of HLA alleles.

Usage

```
hlaSampleAllele(TrueHLA, allele.limit=NULL, max.resolution="")
```

Arguments

`TrueHLA`	an object of `hlaAlleleClass`
`allele.limit`	a list of HLA alleles, the validation samples are limited to those having HLA alleles in `allele.limit`, or `NULL` for no limit. `allele.limit` could be character-type, `hlaAttrBagClass` or `hlaAttrBagObj`
`max.resolution`	"2-digit", "4-digit", "6-digit", "8-digit", "allele", "protein", "2", "4", "6", "8", "full" or "": "allele" = "2-digit", "protein" = "4-digit", "full" and "" mean no limit on resolution

Value

Return a list of sample IDs.

Author(s)

Xiuwen Zheng

See Also

hlaCompareAllele

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")
summary(hla)

hlaSampleAllele(hla)

hlaSampleAllele(hla, allele.limit=c(
    "01:01","02:01","02:06", "03:01", "11:01", "23:01"))
```

---

hlaSetKernelTarget *Set the CPU target*

Description

Set the CPU target that the HIBAG algorithm is built on.

Usage

```
hlaSetKernelTarget(cpu=c("max", "auto.avx2", "base",
    "sse2", "sse4", "avx", "avx2", "avx512f", "avx512bw", "avx512vpopcnt"))
```

Arguments

cpu	Specify the Intel/AMD CPU flag; "max" by default

Details

If `cpu="max"`, the kernel target will be automatically determined according to the CPU capabilities to maximize the algorithm efficiency. When `cpu="auto.avx2"`, "avx2" is used instead of "avx512f", "avx512bw", "avx512vpopcnt" even if the CPU supports the AVX512F, AVX512BW or AVX512VPOPCNT intrinsics, since the CPU may reduce the frequency of the cores dynamically to keep power usage of AVX512 within bounds; if AVX2 is not applicable, other target will be automatically determined.

The HIBAG algorithm is optimized using different SIMD instruction sets to leverage the efficiency of the target Intel/AMD platform. The higher version of the C++ compiler is needed to enable the compilation of AVX2 and AVX512F intrinsics, e.g., GCC >= v6.0. If the compiler does not support the CPU target, the implementation on that target will be disabled.

Value

Return a character vector for describing the CPU capabilities, the compiler information and the supported implementation.

Author(s)

Xiuwen Zheng

See Also

hlaAttrBagging, hlaParallelAttrBagging, predict.hlaAttrBagClass, hlaPredict

Examples

```
hlaSetKernelTarget("auto")
```

---

hlaSNPGenoClass *The class of SNP genotypes*

Description

The class of SNP genotypes, and its instance is returned from hlaMakeSNPGeno.

Value

There are five components:

genotype	a genotype matrix, "# of SNPs"-by-"# of individuals"; 0 standing for BB (ZERO A allele), 1 for AB (ONE A allele), 2 for AA (TWO A alleles) and NA for missing values (other values have no meaning)
sample.id	a vector of sample IDs
snp.id	a vector of SNP IDs
snp.position	a vector of SNP positions in basepair
snp.allele	a vector of characters with a format of "A allele/B allele"; B is usually defined as a major or reference allele, while A is defined as a minor or alternative allele
assembly	the human genome reference, such like "hg19"

Author(s)

Xiuwen Zheng

See Also

hlaMakeSNPGeno

hlaSNPID *Get SNP IDs and positions*

Description

Get the information of SNP ID with or without position.

Usage

```
hlaSNPID(obj, type=c("Position", "Pos+Allele", "RefSNP+Position", "RefSNP"))
```

Arguments

obj	a genotypic object of hlaSNPGenoClass, a model object of hlaAttrBagClass or a model object of hlaAttrBagObj
type	"RefSNP+Position" (by default), "RefSNP" or "Position"

Value

If type = "RefSNP+Position", return paste(obj$snp.id, obj$snp.position, sep="-"); if type = "RefSNP", return obj$snp.id; if type = "Position", return obj$snp.position; if type = "Pos+Allele", return paste(obj$snp.position, obj$snp.allele, sep="-").

Author(s)

Xiuwen Zheng

See Also

hlaGenoSwitchStrand, hlaGenoCombine

Examples

```
x <- hlaSNPID(HapMap_CEU_Geno)
head(x)

x <- hlaSNPID(HapMap_CEU_Geno, "RefSNP")
head(x)

x <- hlaSNPID(HapMap_CEU_Geno, "Position")
head(x)
```

`hlaSplitAllele` *Divide the samples randomly*

Description

Divide the samples to the training and validation sets randomly.

Usage

```
hlaSplitAllele(HLA, train.prop=0.5)
```

Arguments

`HLA`	an object of `hlaAlleleClass`
`train.prop`	the proporion of training set

Details

The algorithm tries to divide each HLA alleles into training and validation sets randomly with a training proportion `train.prop`.

Value

Return a list:

`training`	an object of `hlaAlleleClass`
`validation`	an object of `hlaAlleleClass`

Author(s)

Xiuwen Zheng

See Also

`hlaAllele`

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# divide HLA types randomly
set.seed(100)
hlatab <- hlaSplitAllele(hla, train.prop=0.5)
names(hlatab)
```

```
# "training"   "validation"
summary(hlatab$training)
summary(hlatab$validation)
```

`hlaSubModelObj` *Get a subset of individual classifiers*

Description

Get the first n individual classifiers.

Usage

```
hlaSubModelObj(obj, n)
```

Arguments

`obj`	an object of `hlaAttrBagObj`
`n`	an integer, get the first n individual classifiers

Value

Return an object of `hlaAttrBagObj`.

Author(s)

Xiuwen Zheng

See Also

`hlaAttrBagging`

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "C"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 50   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

# train a HIBAG model
```

```
set.seed(1000)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hla, train.geno, nclassifier=2, verbose.detail=TRUE)
mobj <- hlaModelToObj(model)
summary(mobj)

newmobj <- hlaSubModelObj(mobj, 1)
summary(newmobj)
```

`hlaUniqueAllele` *Get unique HLA alleles*

Description

Get unique HLA alleles, which are in ascending order.

Usage

```
hlaUniqueAllele(hla, all=NA)
```

Arguments

`hla`	character-type HLA alleles, a `hlaAlleleClass` object, a `link{hlaAttrBagClass}` object, or a `link{hlaAttrBagObj}` object
`all`	when `hla` is a `hlaAlleleClass` object and `all=TRUE`, return all HLA alleles if `hla$dosage` or `hla$postprob` exists; otherwise, only return the alleles in `hla$value`

Details

Each HLA allele name has a unique number corresponding to up to four sets of digits separated by colons. The name designation depends on the sequence of the allele and that of its nearest relative. The digits before the first colon describe the type, which often corresponds to the serological antigen carried by an allotype. The next set of digits are used to list the subtypes, numbers being assigned in the order in which DNA sequences have been determined. Alleles whose numbers differ in the two sets of digits must differ in one or more nucleotide substitutions that change the amino acid sequence of the encoded protein. Alleles that differ only by synonymous nucleotide substitutions (also called silent or non-coding substitutions) within the coding sequence are distinguished by the use of the third set of digits. Alleles that only differ by sequence polymorphisms in the introns or in the 5' or 3' untranslated regions that flank the exons and introns are distinguished by the use of the fourth set of digits.

In addition to the unique allele number there are additional optional suffixes that may be added to an allele to indicate its expression status. Alleles that have been shown not to be expressed, 'Null' alleles have been given the suffix 'N'. Those alleles which have been shown to be alternatively expressed may have the suffix 'L', 'S', 'C', 'A' or 'Q'.

http://hla.alleles.org/nomenclature/index.html

Value

Return a character vector of HLA alleles

Author(s)

Xiuwen Zheng

See Also

hlaAllele, hlaAlleleDigit

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "A"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")
summary(hla)
hlaUniqueAllele(hla)

hlaUniqueAllele(c("01", "01:03", "01:01", "03:05", "03:01G",
    "03:05P", "03:104:01", "104:01"))
```

HLA_Type_Table *Four-digit HLA types of a study simulated from HapMap CEU*

Description

A data.frame object including HLA-A, B, C, DRB1, DQA1 and DQB1 loci of 60 samples.

Usage

```
HLA_Type_Table
```

Value

A `data.frame`

References

A high-resolution HLA and SNP haplotype map for disease association studies in the extended human MHC. de Bakker PI, McVean G, Sabeti PC, Miretti MM, Green T, Marchini J, Ke X, Monsuur AJ, Whittaker P, Delgado M, Morrison J, Richardson A, Walsh EC, Gao X, Galver L, Hart J, Hafler DA, Pericak-Vance M, Todd JA, Daly MJ, Trowsdale J, Wijmenga C, Vyse TJ, Beck S, Murray SS, Carrington M, Gregory S, Deloukas P, Rioux JD. Nat Genet. 2006 Oct;38(10):1166-72. Epub 2006 Sep 24.

`plot.hlaAttrBagObj` *Plot a HIBAG model*

Description

To show a scatterplot of the numbers of individual classifiers and SNP positions.

Usage

```
## S3 method for class 'hlaAttrBagObj'
plot(x, snp.col="gray33", snp.pch=1, snp.sz=1,
    locus.col="blue", locus.lty=1L, locus.lty2=2L, addplot=NULL,
    assembly="auto", ...)
## S3 method for class 'hlaAttrBagClass'
plot(x, ...)
```

Arguments

`x`	an object of `hlaAttrBagObj`
`snp.col`	the color of SNP uses
`snp.pch`	the point type of SNP uses
`snp.sz`	the point size of SNP uses
`locus.col`	the color of text and line for HLA locus
`locus.lty`	the type of line for the bounds of HLA locus
`locus.lty2`	the type of line for HLA locus
`addplot`	NULL for creating a plot, or a ggplot object to be appended
`assembly`	the human genome reference: "hg18", "hg19" (default), "hg38"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
`...`	further arguments passed to or from other methods

Value

None

Author(s)

Xiuwen Zheng

See Also

`print.hlaAttrBagObj`, `summary.hlaAttrBagObj`

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "C"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 100   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

# train a HIBAG model
set.seed(1000)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hla, train.geno, nclassifier=2, verbose.detail=TRUE)
plot(model)
```

---

print.hlaAttrBagClass *Summarize a "hlaAttrBagClass" or "hlaAttrBagObj" object.*

Description

Summarize an object of hlaAttrBagClass or hlaAttrBagObj.

Usage

```
## S3 method for class 'hlaAttrBagClass'
print(x, ...)
## S3 method for class 'hlaAttrBagObj'
print(x, ...)
## S3 method for class 'hlaAttrBagClass'
summary(object, show=TRUE, ...)
## S3 method for class 'hlaAttrBagObj'
summary(object, show=TRUE, ...)
```

Arguments

x	an object of hlaAttrBagClass or hlaAttrBagObj
object	an object of hlaAttrBagClass or hlaAttrBagObj
show	if TRUE, show information
...	further arguments passed to or from other methods

Value

`print` returns `NULL`.

`summary.hlaAttrBagClass` and `summary.hlaAttrBagObj` return a `list`:

`num.classifier`	the total number of classifiers
`num.snp`	the total number of SNPs
`snp.id`	SNP IDs
`snp.position`	SNP position in basepair
`snp.hist`	the number of classifier for each SNP, and it could be used for SNP importance
`info`	a `data.frame` for the average number of SNPs (`num.snp`), haplotypes (`num.haplo`), out-of-bag accuracies (`accuracy`) among all classifiers: mean, standard deviation, min, max

Author(s)

Xiuwen Zheng

See Also

`plot.hlaAttrBagClass`, `plot.hlaAttrBagObj`

Examples

```
# make a "hlaAlleleClass" object
hla.id <- "C"
hla <- hlaAllele(HLA_Type_Table$sample.id,
    H1 = HLA_Type_Table[, paste(hla.id, ".1", sep="")],
    H2 = HLA_Type_Table[, paste(hla.id, ".2", sep="")],
    locus=hla.id, assembly="hg19")

# training genotypes
region <- 100   # kb
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
    hla.id, region*1000, assembly="hg19")
train.geno <- hlaGenoSubset(HapMap_CEU_Geno,
    snp.sel = match(snpid, HapMap_CEU_Geno$snp.id))

# train a HIBAG model
set.seed(1000)
# please use "nclassifier=100" when you use HIBAG for real data
model <- hlaAttrBagging(hla, train.geno, nclassifier=2, verbose.detail=TRUE)
print(model)
```

summary.hlaAlleleClass *Summarize a "hlaAlleleClass" or "hlaAASeqClass" object*

Description

Show the information of a hlaAlleleClass or hlaAASeqClass object.

Usage

```
## S3 method for class 'hlaAlleleClass'
summary(object, verbose=TRUE, ...)
## S3 method for class 'hlaAASeqClass'
summary(object, poly.only=TRUE, head=0L,
    verbose=TRUE, ...)
## S3 method for class 'hlaAlleleClass'
print(x, ...)
```

Arguments

object	an object of hlaAlleleClass or hlaAASeqClass
x	an object of hlaAlleleClass or hlaAASeqClass
poly.only	if TRUE, only show the amino acid positions with polymorphism; otherwise, show all sequences
head	show the first head rows of cross tabulation, or 0L for all rows
verbose	if TRUE, show information
...	further arguments passed to or from other methods

Value

Return a data.frame of count and frequency for each HLA allele, if object is hlaAlleleClass; a matrix of cross tabulation of amino acids at each position, if object is hlaAASeqClass.

Author(s)

Xiuwen Zheng

See Also

hlaAllele, hlaConvSequence

summary.hlaSNPGenoClass *Summarize a SNP dataset*

Description

Summarize the genotypic dataset.

Usage

```
## S3 method for class 'hlaSNPGenoClass'
summary(object, show=TRUE, ...)
## S3 method for class 'hlaSNPGenoClass'
print(x, ...)
```

Arguments

object	a genotype object of hlaSNPGenoClass
x	a genotype object of hlaSNPGenoClass
show	if TRUE, print information
...	further arguments passed to or from other methods

Value

None.

Author(s)

Xiuwen Zheng

See Also

hlaMakeSNPGeno, hlaGenoSubset

Examples

```
summary(HapMap_CEU_Geno)
```

Index