

# Package ‘HIBAG’

October 12, 2016

**Type** Package

**Title** HLA Genotype Imputation with Attribute Bagging

**Version** 1.8.3

**Date** 2016-05-30

**Depends** R (>= 3.2.0)

**Imports** methods

**Suggests** parallel, knitr, gdsfmt (>= 1.2.2), SNPRelate (>= 1.1.6)

**Description** It is a software package for imputing HLA types using SNP data, and 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

**ByteCompile** TRUE

**biocViews** Genetics, StatisticalMethod

**URL** <http://www.biostat.washington.edu/~bsweir/HIBAG/>,  
<http://github.com/zhengxwen/HIBAG>

**NeedsCompilation** yes

**Author** Xiuwen Zheng [aut, cre, cph],  
Bruce Weir [ctb, ths]

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

**R topics documented:**

HIBAG-package	3
HapMap_CEU_Geno	6
hlaAASeqClass	6
hlaAllele	7
hlaAlleleClass	9
hlaAlleleDigit	10
hlaAlleleSubset	11
hlaAssocTest	12
hlaAttrBagClass	15
hlaAttrBagging	16
hlaAttrBagObj	19
hlaBED2Geno	20
hlaCheckAllele	21
hlaCheckSNPs	22
hlaClose	23
hlaCombineAllele	24
hlaCombineModelObj	25
hlaCompareAllele	26
hlaConvSequence	28
hlaErrMsg	31
hlaFlankingSNP	32
hlaGDS2Geno	33
hlaGeno2PED	34
hlaGenoAFreq	35
hlaGenoCombine	36
hlaGenoLD	37
hlaGenoMFreq	38
hlaGenoMRate	39
hlaGenoMRate_Samp	40
hlaGenoSubset	40
hlaGenoSwitchStrand	42
hlaLociInfo	43
hlaMakeSNPGeno	44
hlaModelFiles	45
hlaModelFromObj	46
hlaOutOfBag	48
hlaParallelAttrBagging	49
hlaPredMerge	52
hlaPublish	54
hlaReport	55
hlaSampleAllele	57
hlaSNPGenoClass	58
hlaSNPID	59
hlaSplitAllele	60
hlaSubModelObj	61
hlaUniqueAllele	62

HLA_Type_Table . . . . .	63
plot.hlaAttrBagObj . . . . .	64
predict.hlaAttrBagClass . . . . .	65
print.hlaAttrBagClass . . . . .	67
summary.hlaAlleleClass . . . . .	69
summary.hlaSNPGenoClass . . . . .	70

**Index****71**

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:	v1.3

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 (<http://www.biostat.washington.edu/~bsweir/HIBAG/>) 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. <http://www.nature.com/tpj/journal/v14/n2/full/tpj201318a.html>

## 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 <- predict(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 <- predict(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 <- predict(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	<i>SNP genotypes of a study simulated from HapMap CEU genotypic data</i>
-----------------	--

---

### Description

An object of [hlaSNPGenoClass](#) of 60 samples and 1564 SNPs.

### Usage

```
HapMap_CEU_Geno
```

### Value

A list

### References

[http://hapmap.ncbi.nlm.nih.gov/downloads/genotypes/2010-08\\_phaseII+III/forward/](http://hapmap.ncbi.nlm.nih.gov/downloads/genotypes/2010-08_phaseII+III/forward/)

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

---

hlaAASeqClass	<i>Class of HLA Amino Acid Sequence Type</i>
---------------	--

---

### 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 types*

---

**Description**

Return an object of [hlaAlleleClass](#), which contains HLA types.

**Usage**

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

**Arguments**

sample.id	sample IDs
H1	a vector of HLA alleles
H2	a vector of HLA alleles
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
locus	the name of HLA locus: "A", "B", "C", "DRB1", "DRB5", "DQA1", "DQB1", "DPB1", or "any", where "any" indicates any other multiallelic locus
assembly	the human genome reference: "hg18", "hg19" (default), "hg20"; "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](#)

**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	<i>Class of HLA Type</i>
----------------	--------------------------

---

**Description**

The definition of a class for HLA types, returned from [hlaAllele](#).

**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"
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	<i>Trim HLA alleles</i>
----------------	-------------------------

---

**Description**

Trim HLA alleles to specified width.

**Usage**

```
hlaAlleleDigit(obj, max.resolution="4-digit", rm.suffix=FALSE)
```

**Arguments**

obj	should be a <a href="#">hlaAlleleClass</a> 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": "allele" = "2-digit"; "protein" = "4-digit"; "full" and "" indicating no limit on resolution
rm.suffix	whether remove the suffix, e.g., for "01:22N", "N" is a suffix

**Details**

Either HLA types or H1 H2 should be specified. The format of HLA types is "allele 1 / allele 2", e.g., "0512/0102". If max.resolution is specified, the HLA alleles will be trimmed with the maximum resolution.

**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	<i>Get a subset of HLA types</i>
-----------------	----------------------------------

---

## Description

Get a subset of HLA types from an object of [hlaAlleleClass](#).

## Usage

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

## Arguments

hla	an object of <a href="#">hlaAlleleClass</a>
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)

```

---

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 <a href="#">hlaAlleleClass</a>
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., $y \sim 1$ , $y \sim h + a$
data	an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code>
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 <code>glm</code> or <code>nlme</code> 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 performed on the cross tabulation, which is constructed according to the specified model (dominant, additive, recessive and genotype).

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.hlaASeqClass](#)

**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	<i>The class of HIBAG model</i>
-----------------	---------------------------------

---

**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

**Author(s)**

Xiuwen Zheng

**See Also**

[hlaAttrBagging](#), [hlaParallelAttrBagging](#), [hlaAttrBagObj](#)

hlaAttrBagging

*Build a HIBAG model***Description**

To build a HIBAG model for predicting HLA types.

**Usage**

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

**Arguments**

hla	the training HLA types, an object of <a href="#">hlaAlleleClass</a>
snp	the training SNP genotypes, an object of <a href="#">hlaSNPGenoClass</a>
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
rm.na	if TRUE, remove the samples with missing HLA types
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 on non-informative SNP markers.

A parallel version of hlaAttrBagging is [hlaParallelAttrBagging](#).

**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

**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. <http://www.nature.com/tpj/journal/v14/n2/full/tpj201318a.html>

**See Also**

[hlaClose](#), [hlaParallelAttrBagging](#), [summary.hlaAttrBagClass](#), [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 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 <- predict(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 <- predict(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)
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**

<code>bed.fn</code>	binary file, genotype information
<code>fam.fn</code>	family, individual information, etc
<code>bim.fn</code>	extended MAP file: two extra cols = allele names
<code>rm.invalid.allele</code>	if TRUE, remove SNPs with invalid alleles
<code>import.chr</code>	the chromosome, "1" .. "22", "X", "Y", "XY", "MT", "xMHC", or "", where "xMHC" implies the extended MHC on chromosome 6, and "" for all SNPs
<code>assembly</code>	the human genome reference: "hg18", "hg19" (default), "hg20"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
<code>verbose</code>	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	<i>Check SNP alleles</i>
----------------	--------------------------

---

**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("RefSNP+Position", "RefSNP", "Position"), verbose=TRUE)
```

**Arguments**

model	an object of <a href="#">hlaAttrBagClass</a> , or an object of <a href="#">hlaAttrBagObj</a>
object	a genotype object of <a href="#">hlaSNPGenoClass</a> , 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	<i>Dispose a model object</i>
----------	-------------------------------

---

### 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	<i>Combine two datasets of HLA types</i>
------------------	--

---

**Description**

Get a subset of HLA types from an object of [hlaAlleleClass](#).

**Usage**

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

**Arguments**

H1	the first <a href="#">hlaAlleleClass</a> object
H2	the second <a href="#">hlaAlleleClass</a> 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,
  max.resolution="", output.individual=FALSE, verbose=TRUE)

```

### Arguments

TrueHLA	an object of <a href="#">hlaAlleleClass</a> , the true HLA types
PredHLA	an object of <a href="#">hlaAlleleClass</a> , the predicted HLA types
allele.limit	a list of HLA alleles, the validation samples are limited to those having HLA alleles in <code>allele.limit</code> , or NULL for no limit. <code>allele.limit</code> could be character-type, <a href="#">hlaAttrBagClass</a> or <a href="#">hlaAttrBagObj</a>
call.threshold	the call threshold for posterior probability, i.e., call or no call is determined by whether <code>prob &gt;= call.threshold</code> or not
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

<code>crt.num.ind</code>	the number of individuals with correct HLA types
<code>crt.num.haplo</code>	the number of chromosomes with correct HLA alleles
<code>acc.ind</code>	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.)
<code>acc.haplo</code>	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.)
<code>call.threshold</code>	call threshold, if it is NaN, no call threshold is executed
<code>n.call</code>	the number of individuals with call
<code>call.rate</code>	overall call rate
confusion (matrix): a confusion matrix.	
detail (data.frame):	
<code>allele</code>	HLA alleles
<code>train.num</code>	the number of training haplotypes
<code>train.freq</code>	the training haplotype frequencies
<code>valid.num</code>	the number of validation haplotypes
<code>valid.freq</code>	the validation haplotype frequencies
<code>call.rate</code>	the call rates for HLA alleles
<code>accuracy</code>	allele accuracy
<code>sensitivity</code>	sensitivity
<code>specificity</code>	specificity
<code>ppv</code>	positive predictive value
<code>npv</code>	negative predictive value
<code>miscall</code>	the most likely miss-called alleles
<code>miscall.prop</code>	the proportions of the most likely miss-called allele in all miss-called alleles
individual (data.frame):	
<code>sample.id</code>	sample id
<code>true.hla</code>	the true HLA type
<code>pred.hla</code>	the prediction of HLA type
<code>accuracy</code>	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 <- predict(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 <a href="#">hlaAlleleClass</a> , 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, like c("09:02"="107:01"), any "09:02" will be replaced by "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](http://hla.alleles.org/alleles/p_groups.html) or [http://hla.alleles.org/alleles/g\\_groups.html](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](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](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

The licence and disclaimer of distributed HLA data: Creative Commons Attribution-NoDerivs Licence (<http://hla.alleles.org/terms.html>).

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")
hlaConvSequence(c("01:01", "02:02", "01:01:01G", "01:01:01:01", "07"),
  locus="A")
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)
```

---

hlaErrMsg

*The last error message*

---

### **Description**

Return the last error message.

### **Usage**

```
hlaErrMsg()
```

### **Value**

Character

### **Author(s)**

Xiuwen Zheng

### **Examples**

```
hlaErrMsg()
```

---

hlaFlankingSNP      *SNP IDs in Flanking Region*

---

**Description**

To select SNPs in the flanking region of a specified HLA locus.

**Usage**

```
hlaFlankingSNP(snp.id, position, hla.id, flank.bp=500*1000, assembly="auto",  
pos.start=NA_integer_, pos.end=NA_integer_)
```

**Arguments**

snp.id	a vector of SNP IDs
position	a vector of positions
hla.id	the name of HLA locus, or "any" for other genes
flank.bp	the size of flanking region on each side in basepair
assembly	the human genome reference: "hg18", "hg19" (default), "hg20"; "auto" refers to "hg19"; "auto-silent" refers to "hg19" without any warning
pos.start	the start position of the gene
pos.end	the end position of the gene

**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$snp.id),
  samp.sel = match(hla$training$value$sample.id, HapMap_CEU_Geno$sample.id))
summary(train.geno)

## customize positions
snpid <- hlaFlankingSNP(HapMap_CEU_Geno$snp.id, HapMap_CEU_Geno$snp.position,
  "any", 500*1000, pos.start=29457816, pos.end=30450205)
```

---

hlaGDS2Geno

*Convert from SNP GDS format*


---

**Description**

To convert a SNP 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	the SNP GDS file used by the SNPRelate package
rm.invalid.allele	if TRUE, remove SNPs with invalid alleles
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), "hg20"; "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, import.chr="6")

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 <a href="#">hlaSNPGenoClass</a>
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 frequencies.

**Author(s)**

Xiuwen Zheng

**See Also**

[hlaGenoAFreq](#), [hlaGenoMFreq](#), [hlaGenoMRate](#), [hlaGenoMRate\\_Samp](#)

**Examples**

```
summary(HapMap_CEU_Geno)

summary(hlaGenoAFreq(HapMap_CEU_Geno))
```

---

hlaGenoCombine	<i>Combine two genotypic data sets into one</i>
----------------	---

---

**Description**

To combine two genotypic data sets into one dataset.

**Usage**

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

**Arguments**

geno1	the first genotype object of <a href="#">hlaSNPGenoClass</a>
geno2	the second genotype object of <a href="#">hlaSNPGenoClass</a>
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 <a href="#">hlaGenoSwitchStrand</a> 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 ( $r^2$ ) 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 ( $r^2$ ) 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 frequencies.

**Author(s)**

Xiuwen Zheng

**See Also**

[hlaGenoAFreq](#), [hlaGenoMFreq](#), [hlaGenoMRate](#), [hlaGenoMRate\\_Samp](#)

**Examples**

```
summary(HapMap_CEU_Geno)
```

```
summary(hlaGenoMFreq(HapMap_CEU_Geno))
```

---

hlaGenoMRate	<i>Missing Rates Per SNP</i>
--------------	------------------------------

---

**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)
```



**Arguments**

<code>genoobj</code>	a genotype object of <a href="#">hlaSNPGenoClass</a>
<code>samp.sel</code>	a logical vector, or an integer vector of indices
<code>snp.sel</code>	a logical vector, or an integer vector of indices

**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:

<code>genotype</code>	a genotype matrix, “# of SNPs” - by - “# of individuals”
<code>sample.id</code>	a vector of sample IDs
<code>snp.id</code>	a vector of SNP IDs
<code>snp.position</code>	a vector of SNP positions in basepair
<code>snp.allele</code>	a vector of characters with the format of “A allele/B allele”

**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 switching*

---

### Description

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

### Usage

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

### Arguments

target	an object of <a href="#">hlaSNPGenoClass</a>
template	a genotypic object of <a href="#">hlaSNPGenoClass</a> , a model object of <a href="#">hlaAttrBagClass</a> or a model object of <a href="#">hlaAttrBagObj</a>
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
```

---

hlaLociInfo

*HLA Locus Information*

---

## Description

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

## Usage

```
hlaLociInfo(assembly=c("auto", "auto-silent", "hg18", "hg19", "hg20",
"unknown"))
```

## Arguments

assembly            the human genome reference: "hg18", "hg19" (default), "hg20"; "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: <http://www.ncbi.nlm.nih.gov/gene>, HLA Nomenclature: <http://hla.alleles.org/genes/index.html>

**Examples**

```
hlaLociInfo()
```

---

hlaMakeSNPGeno	<i>Make a SNP genotype object</i>
----------------	-----------------------------------

---

**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 in the SNP list
B.allele	a vector of B alleles in the SNP list
assembly	the human genome reference: "hg18", "hg19" (default), "hg20"; "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 <a href="#">hlaAttrBagObj</a>
model	an object of <a href="#">hlaAttrBagClass</a>

**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	<i>Out-of-bag estimation of overall accuracy, per-allele sensitivity, etc</i>
-------------	---

---

### 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 <a href="#">hlaAttrBagClass</a> or <a href="#">hlaAttrBagObj</a>
hla	the training HLA types, an object of <a href="#">hlaAlleleClass</a>
snp	the training SNP genotypes, an object of <a href="#">hlaSNPGenoClass</a>
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=100, mtry=c("sqrt", "all", "one"), prune=TRUE, rm.na=TRUE,
  stop.cluster=FALSE, verbose=TRUE)

```

## Arguments

cl	a cluster object, created by the package <a href="#">parallel</a> or <a href="#">snow</a> ; if NULL is given, a uniprocessor implementation will be performed
hla	training HLA types, an object of <a href="#">hlaAlleleClass</a>
snp	training SNP genotypes, an object of <a href="#">hlaSNPGenoClass</a>
auto.save	specify a autosaved file, 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
rm.na	if TRUE, remove the samples with missing HLA types
stop.cluster	TRUE: stop cluster nodes after computing
verbose	if TRUE, show 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.

If auto.save="", the function returns a HIBAG model (an object of [hlaAttrBagClass](#)); otherwise, there is no return.

### Value

Return an object of [hlaAttrBagClass](#) if auto.save is specified.

### 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. <http://www.nature.com/tpj/journal/v14/n2/full/tpj201318a.html>

### See Also

[hlaAttrBagging](#), [hlaClose](#)

### 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))

#####

library(parallel)

# use option cl.core to choose an appropriate cluster size.
cl <- makeCluster(getOption("cl.cores", 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 <- predict(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(getOption("cl.cores", 2))

```

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

# stop parallel nodes
stopCluster(cl)

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

---

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)
```

### Arguments

...	The object(s) of <a href="#">hlaAlleleClass</a> , having a field of 'postprob', and returned by <code>predict(..., type="response+prob", vote="majority")</code>
weight	the weight used for each prediction; if NULL, equal weights
equivalence	a data.frame with two columns, the first column for new equivalent alleles, and the second for the alleles possibly existed in the object(s) passed to this function

### 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 <- predict(m1, test.geno, type="response+prob", vote="majority")
pd2 <- predict(m2, test.geno, type="response+prob", vote="majority")

hlaCompareAllele(hlatab$validation, pd1)$overall

hlaCompareAllele(hlatab$validation, pd2)$overall

# merge predictions from multiple models, by voting from all classifiers
pd <- hlaPredMerge(pd1, pd2, weight=c(1,1))

```

```
hlaCompareAllele(hlatab$validation, pd)$overall
```

---

hlaPublish	<i>Finalize a HIBAG model</i>
------------	-------------------------------

---

### 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 <a href="#">hlaAttrBagObj</a> or <a href="#">hlaAttrBagClass</a>
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 <- predict(model, train.geno)
p2 <- predict(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 <a href="#">hlaCompareAllele</a>
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 <- predict(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")

```

---

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 <a href="#">hlaAlleleClass</a>
allele.limit	a list of HLA alleles, the validation samples are limited to those having HLA alleles in <code>allele.limit</code> , or NULL for no limit. <code>allele.limit</code> could be character-type, <a href="#">hlaAttrBagClass</a> or <a href="#">hlaAttrBagObj</a>
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"))
```

---

hlaSNPGenoClass	<i>The class of SNP genotypes</i>
-----------------	-----------------------------------

---

**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”
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”
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("RefSNP+Position", "RefSNP", "Position"))
```

**Arguments**

obj	a genotypic object of <a href="#">hlaSNPGenoClass</a> , a model object of <a href="#">hlaAttrBagClass</a> or a model object of <a href="#">hlaAttrBagObj</a>
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$snp.id`; otherwise, return `obj$snp.position`.

**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	<i>Divide the samples randomly</i>
----------------	------------------------------------

---

**Description**

Divide the samples to the training and validation sets randomly.

**Usage**

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

**Arguments**

HLA	an object of <a href="#">hlaAlleleClass</a>
train.prop	the proportion 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 <a href="#">hlaAlleleClass</a>
validation	an object of <a href="#">hlaAlleleClass</a>

**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	<i>Get a subset of individual classifiers</i>
----------------	---

---

## Description

Get the first n individual classifiers.

## Usage

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

## Arguments

obj	an object of <a href="#">hlaAttrBagObj</a>
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)
```

### Arguments

hla                      character-type HLA alleles, or a [hlaAlleleClass](#) object

### 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, xlab=NULL, ylab=NULL,  
      locus.color="red", locus.lty=2, locus.cex=1.25, assembly="auto", ...)  
## S3 method for class 'hlaAttrBagClass'  
plot(x, ...)
```

### Arguments

x	an object of <a href="#">hlaAttrBagObj</a>
xlab	the label of X-axis
ylab	the label of Y-axis
locus.color	the color of text and line for HLA locus
locus.lty	the type of line for HLA locus
locus.cex	the font size of HLA locus
assembly	the human genome reference: "hg18", "hg19" (default), "hg20"; "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)
```

---

```
predict.hlaAttrBagClass
```

*HIBAG model prediction (in parallel)*

---

**Description**

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

**Usage**

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

**Arguments**

object	a model of <a href="#">hlaAttrBagClass</a>
snp	a genotypic object of <a href="#">hlaSNPGenoClass</a>
cl	a cluster object, created by the package <a href="#">parallel</a> or <a href="#">snow</a> ; if NULL is given, a uniprocessor implementation will be performed

type	"response": return the best-guess type plus its posterior probability; "prob": return all posterior probabilities; "response+prob": return the best-guess 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	"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	if TRUE, show information
...	further arguments passed to or from other methods

### 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 RefSNP IDs and positions. A lower missing fraction may be gained by matching RefSNP IDs or positions only. Call `predict(..., match.type="RefSNP")` or `predict(..., match.type="Position")` for this purpose. It might 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 RefSNP 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](#)

### 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 <- predict(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))

```

---

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 <a href="#">hlaAttrBagClass</a> or <a href="#">hlaAttrBagObj</a>
object	an object of <a href="#">hlaAttrBagClass</a> or <a href="#">hlaAttrBagObj</a>
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, ...)

```

## Arguments

object	an object of <a href="#">hlaAlleleClass</a> or <a href="#">hlaAASeqClass</a>
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, ...)
```

### Arguments

object	a genotype object of <a href="#">hlaSNPGenoClass</a>
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

## \*Topic **HLA**

- HIBAG-package, 3
- HLA\_Type\_Table, 63
- hlaAASeqClass, 6
- hlaAllele, 7
- hlaAlleleClass, 9
- hlaAlleleDigit, 10
- hlaAlleleSubset, 11
- hlaAssocTest, 12
- hlaAttrBagClass, 15
- hlaAttrBagging, 16
- hlaAttrBagObj, 19
- hlaClose, 23
- hlaCombineAllele, 24
- hlaCombineModelObj, 25
- hlaCompareAllele, 26
- hlaConvSequence, 28
- hlaErrMsg, 31
- hlaLociInfo, 43
- hlaModelFiles, 45
- hlaModelFromObj, 46
- hlaOutOfBag, 48
- hlaParallelAttrBagging, 49
- hlaPredMerge, 52
- hlaSampleAllele, 57
- hlaSplitAllele, 60
- hlaSubModelObj, 61
- hlaUniqueAllele, 62
- plot.hlaAttrBagObj, 64
- predict.hlaAttrBagClass, 65
- print.hlaAttrBagClass, 67
- summary.hlaAlleleClass, 69

## \*Topic **SNP**

- HapMap\_CEU\_Geno, 6
- HIBAG-package, 3
- hlaAssocTest, 12
- hlaAttrBagging, 16
- hlaBED2Geno, 20
- hlaCheckAllele, 21

- hlaCheckSNPs, 22
- hlaConvSequence, 28
- hlaFlankingSNP, 32
- hlaGDS2Geno, 33
- hlaGeno2PED, 34
- hlaGenoAFreq, 35
- hlaGenoCombine, 36
- hlaGenoLD, 37
- hlaGenoMFreq, 38
- hlaGenoMRate, 39
- hlaGenoMRate\_Samp, 40
- hlaGenoSubset, 40
- hlaGenoSwitchStrand, 42
- hlaMakeSNPGeno, 44
- hlaSNPGenoClass, 58
- hlaSNPID, 59
- predict.hlaAttrBagClass, 65
- summary.hlaSNPGenoClass, 70

## \*Topic **datasets**

- HapMap\_CEU\_Geno, 6
- HLA\_Type\_Table, 63

## \*Topic **genetics**

- HapMap\_CEU\_Geno, 6
- HIBAG-package, 3
- HLA\_Type\_Table, 63
- hlaAASeqClass, 6
- hlaAllele, 7
- hlaAlleleClass, 9
- hlaAlleleDigit, 10
- hlaAlleleSubset, 11
- hlaAssocTest, 12
- hlaAttrBagClass, 15
- hlaAttrBagging, 16
- hlaAttrBagObj, 19
- hlaBED2Geno, 20
- hlaCheckAllele, 21
- hlaCheckSNPs, 22
- hlaClose, 23
- hlaCombineAllele, 24

- hlaCombineModelObj, 25
  - hlaCompareAllele, 26
  - hlaConvSequence, 28
  - hlaFlankingSNP, 32
  - hlaGDS2Geno, 33
  - hlaGeno2PED, 34
  - hlaGenoAFreq, 35
  - hlaGenoCombine, 36
  - hlaGenoLD, 37
  - hlaGenoMFreq, 38
  - hlaGenoMRate, 39
  - hlaGenoMRate\_Samp, 40
  - hlaGenoSubset, 40
  - hlaGenoSwitchStrand, 42
  - hlaLociInfo, 43
  - hlaMakeSNPGeno, 44
  - hlaModelFiles, 45
  - hlaModelFromObj, 46
  - hlaOutOfBag, 48
  - hlaParallelAttrBagging, 49
  - hlaPredMerge, 52
  - hlaPublish, 54
  - hlaReport, 55
  - hlaSampleAllele, 57
  - hlaSNPGenoClass, 58
  - hlaSNPID, 59
  - hlaSplitAllele, 60
  - hlaSubModelObj, 61
  - hlaUniqueAllele, 62
  - plot.hlaAttrBagObj, 64
  - predict.hlaAttrBagClass, 65
  - print.hlaAttrBagClass, 67
  - summary.hlaAlleleClass, 69
  - summary.hlaSNPGenoClass, 70
- glm, 13
- HapMap\_CEU\_Geno, 6
  - HIBAG (HIBAG-package), 3
  - HIBAG-package, 3
  - HLA\_Type\_Table, 63
  - hlaAASeqClass, 6, 30, 69
  - hlaAllele, 7, 9–11, 24, 52, 60, 63, 66, 69
  - hlaAlleleClass, 7, 8, 9, 10–12, 16, 24, 26, 29, 37, 48, 49, 52, 57, 60, 62, 66, 69
  - hlaAlleleDigit, 9, 10, 11, 63
  - hlaAlleleSubset, 9, 11, 24, 30
  - hlaAssocTest, 12
  - hlaAttrBagClass, 15, 16, 20, 22, 23, 26, 42, 47, 48, 50, 54, 57, 59, 65, 67, 68
  - hlaAttrBagging, 15, 16, 20, 22, 23, 25, 27, 46, 47, 50, 52, 61, 66
  - hlaAttrBagObj, 15, 19, 22, 25, 26, 42, 45, 47, 48, 54, 57, 59, 61, 64, 67, 68
  - hlaBED2Geno, 20, 34, 35
  - hlaCheckAllele, 21
  - hlaCheckSNPs, 21, 22
  - hlaClose, 17, 23, 50
  - hlaCombineAllele, 24
  - hlaCombineModelObj, 25
  - hlaCompareAllele, 26, 48, 56, 58, 66
  - hlaConvSequence, 7, 14, 28, 69
  - hlaErrMsg, 31
  - hlaFlankingSNP, 32
  - hlaGDS2Geno, 20, 33
  - hlaGeno2PED, 20, 34, 34
  - hlaGenoAFreq, 35, 36, 39, 40
  - hlaGenoCombine, 36, 41, 45, 59
  - hlaGenoLD, 37
  - hlaGenoMFreq, 36, 38, 39, 40
  - hlaGenoMRate, 36, 39, 39, 40
  - hlaGenoMRate\_Samp, 36, 39, 40, 40
  - hlaGenoSubset, 32, 37, 40, 42, 45, 70
  - hlaGenoSwitchStrand, 36, 42, 59
  - hlaLociInfo, 32, 43
  - hlaMakeSNPGeno, 37, 41, 42, 44, 58, 59, 70
  - hlaModelFiles, 20, 25, 45
  - hlaModelFromObj, 46, 54
  - hlaModelToObj, 20, 46, 54
  - hlaModelToObj (hlaModelFromObj), 46
  - hlaOutOfBag, 48
  - hlaParallelAttrBagging, 15–17, 20, 49, 66
  - hlaPredict (predict.hlaAttrBagClass), 65
  - hlaPredMerge, 52
  - hlaPublish, 54
  - hlaReport, 27, 48, 55
  - hlaSampleAllele, 57
  - hlaSNPGenoClass, 6, 16, 20, 22, 33–42, 44, 48, 49, 58, 59, 65, 70
  - hlaSNPID, 59
  - hlaSplitAllele, 60
  - hlaSubModelObj, 61
  - hlaUniqueAllele, 62
- parallel, 49, 65
- plot.hlaAttrBagClass, 68



plot.hlaAttrBagClass  
    (plot.hlaAttrBagObj), 64  
plot.hlaAttrBagObj, 64, 68  
predict.hlaAttrBagClass, 17, 22, 27, 52,  
    65  
print.hlaAttrBagClass, 67  
print.hlaAttrBagObj, 64  
print.hlaAttrBagObj  
    (print.hlaAttrBagClass), 67  
  
summary.hlaAASeqClass, 14  
summary.hlaAASeqClass  
    (summary.hlaAlleleClass), 69  
summary.hlaAlleleClass, 69  
summary.hlaAttrBagClass, 17, 23  
summary.hlaAttrBagClass  
    (print.hlaAttrBagClass), 67  
summary.hlaAttrBagObj, 64  
summary.hlaAttrBagObj  
    (print.hlaAttrBagClass), 67  
summary.hlaSNPGenoClass, 70