BioSeqClass

April 19, 2010

`basic`	*Assistant Functions*

Description

Assistant functions including read/write files, invoke perl programs, and so on.

Usage

```
## Elements and groups of base and amino acid
elements(ele.type)
aaClass(aa.type)

pwm(seq,class=elements("aminoacid"))

sub.seq(seq, start, stop)

.pathPerl(perlName, os)
.callPerl(perlName, os)

data(dssp.ss)
data(aa.index)
data(PROPERTY)
data(DiProDB)
```

Arguments

`ele.type`	a string for the type of biological sequence. This must be one of the strings "rnaBase", "dnaBase", "aminoacid" or "aminoacid2".
`aa.type`	a string for the group of amino acids. This must be one of the strings "aaH", "aaV", "aaZ", "aaP", "aaF", "aaS" or "aaE".
`seq`	a string vector for the protein or gene sequences.
`class`	a list for the class of biological properties. It can be produced by `elements` and `aaClass`.
`start`	a positive integer vector indicating the first element of subsequence.
`stop`	a positive integer vector indicating the last element of subsequence.
`perlName`	a character string for the name of perl program.
`os`	character string, giving the Operating System (family) of the computer.

Details

pwm returns a M*N position weight matrix (PWM) of input sequences. M is the number of elements given by parameter "class". N is the length of each sequence. Each row is a kind of element, and each column is a position. The input sequences must have equal length.

.pathPerl write the path of Perl to perl program file.

.callPerl invoke Perl program via R.

sub.seq extract subsequences by given start and stop postions.

dssp.ss is a vector storing the secondary structure data from DSSP database (http://swift.cmbi.ru.nl/gv/dssp/).

aa.index is a list storing the properties of amino acids from AAIndex database (http://www.genome.jp/aaindex).

PROPERTY is a list sotring the properties of dinucleotide from B-DNA-VIDEO PROPERTY database (http://wwwmgs.bionet.nsc.ru/mgs/systems/bdnavideo/).

DiProDB is a list sotring the conformational and thermodynamic dinucleotide properties from DiProDB database (http://diprodb.fli-leibniz.de/).

Author(s)

Hong Li

Examples

```
## use readFASTA()/writeFASTA() to read/write FASTA file.
library(Biostrings)
file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.fasta")
tmp = readFASTA(file)
seq <- sapply(tmp,function(x){x[["seq"]]})
names(seq) <- sapply(tmp,function(x){x[["desc"]]})

## get the sequence fragment
sub1 = sub.seq(seq,rep(1,length(seq)),rep(4,length(seq)))
tmp = round(sapply(seq,function(x){runif(1,min=1,
  max=length(unlist(strsplit(x,split="")))-5)}))
sub2 = sub.seq(seq,tmp,tmp+5)

## cacluate the position weight matrix for sequences with equal length
m = BioSeqClass:::pwm(sub1)

## load data: dssp.ss
data(dssp.ss)
## see the data in dssp.ss
dssp.ss[1:5]
```

---

classify *Classification*

Description

Classification with cross validation and selected features

Usage

```
classify(data,classifyMethod="libsvm",cv=10,
               features, evaluator, search, n=200,
               svm.kernel="linear",svm.scale=FALSE,
               svm.path, svm.options="-t 0",
               knn.k=1,
               nnet.size=2, nnet.rang=0.7, nnet.decay=0, nnet.maxit=100)
```

Arguments

`data`	a data frame including the feature matrix and class label. The last column is a vector of class label comprising of "-1" or "+1"; Other columns are features.
`classifyMethod`	a string for the classification method. This must be one of the strings "libsvm", "svmlight", "NaiveBayes", "randomForest", "knn", "tree", "nnet", "rpart", "ctree", "ctreelibsvm", "bagging".
`cv`	an integer for the time of cross validation, or a string "leave_one_out" for the jacknife test.
`features`	an integer vector for the index of interested columns in data, which will be used as features for build classification model.
`evaluator`	a string for the feature selection method used by WEKA. This must be one of the strings "CfsSubsetEval", "ChiSquaredAttributeEval", "InfoGainAttributeEval", or "SVMAttributeEval".
`search`	a string for the search method used by WEKA. This must be one of the strings "BestFirst" or "Ranker".
`n`	an integer for the number of selected features.
`svm.kernel`	a string for kernel function of SVM.
`svm.scale`	a logical vector indicating the variables to be scaled.
`svm.path`	a character for path to SVMlight binaries (required, if path is unknown by the OS).
`svm.options`	Optional parameters to SVMlight. For further details see: "How to use" on http://svmlight.joachims.org/. (e.g.: "-t 2 -g 0.1"))
`nnet.size`	number of units in the hidden layer. Can be zero if there are skip-layer units.
`nnet.rang`	Initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(|x|) is about 1.
`nnet.decay`	parameter for weight decay.
`nnet.maxit`	maximum number of iterations.
`knn.k`	number of neighbours considered in function `classifyModelKNN`.

Details

`classify` employ feature selction method in Weka and diverse classification model in other R packages to perfrom classification. "Cross Validation" is controlled by parameter "cv"; "Feature Selection" is controlled by parameter "features", "evaluator", "search", and "n"; "Classification Model Building" is controlled by parameter "classifyMethod".

Parameter "evaluator" supportes three feature selection methods provided by WEKA: "CfsSubsetEval": Evaluate the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. "ChiSquaredAttributeEval": Evaluate the worth of an attribute by computing the value of the chi-squared statistic with respect to the class. "InfoGainAttributeEval": Evaluate attributes individually by measuring information gain with respect to the class. "SVMAttributeEval": Evaluate the worth of an attribute by using an SVM classifier. Attributes are ranked by the square of the weight assigned by the SVM. Attribute selection for multiclass problems is handled by ranking attributes for each class seperately using a one-vs-all method and then "dealing" from the top of each pile to give a final ranking.

Parameter "search" supportes three feature subset search methods provided by WEKA: "BestFirst": Searches the space of attribute subsets by greedy hillclimbing augmented with a backtracking facility. Setting the number of consecutive non-improving nodes allowed controls the level of backtracking done. Best first may start with the empty set of attributes and search forward, or start with the full set of attributes and search backward, or start at any point and search in both directions (by considering all possible single attribute additions and deletions at a given point). "Ranker": Ranks attributes by their individual evaluations.

Parameter "classifyMethod" supports multiple classification model: "libsvm": Employ `classifyModelLIBSVM` to perform Support Vecotr Machine by LibSVM. Package "e1071" is required. "svmlight": Employ `classifyModelSVMLIGHT` to Support Vecotr Machine by SVMLight. Package "klaR" is required. "NaiveBayes": Employ `classifyModelNB` to perform Naive Bayes classification. Package "klaR" is required. "randomForest": Employ `classifyModelRF` to perform random forest classification. Package "randomForest" is required. "knn": Employ `classifyModelKNN` to perform k Nearest Neighbor algorithm. Package "class" is required. "tree": Employ `classifyModelTree` to perform tree classification. Package "tree" is required. "nnet": Employ `classifyModelNNET` to perform neural net algorithm. Bundle "VR" is required. "rpart": Employ `classifyModelRPART` to perform Recursive Partitioning and Regression Trees. Package "rpart" is required. "ctree": Employ `classifyModelCTREE` to perform Conditional Inference Trees. Package "party" is required. "ctreelibsvm": Employ `classifyModelCTREELIBSVM` to combine Conditional Inference Trees and Support Vecotr Machine for classification. For each node in the tree, one SVM model will be constructed using train data in this node. Test data will be firstly classified to one node of the tree, and then use corresponding SVM to do classification. Package "party" and "e1071" is required. "bagging": Employ `classifyModelBAG` to perform bagging for classification trees. Package "ipred" is required.

Author(s)

Hong Li

Examples

```
## read positive/negative sequence from files.
tmpfile1 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep"
tmpfile2 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.neg40.pep"
posSeq = as.matrix(read.csv(tmpfile1,header=FALSE,sep="\t",row.names=1))[,1]
negSeq = as.matrix(read.csv(tmpfile2,header=FALSE,sep="\t",row.names=1))[,1]
seq=c(posSeq,negSeq)
classLable=c(rep("+1",length(posSeq)),rep("-1",length(negSeq)) )
data = data.frame(featureBinary(seq),classLable)

## Use LibSVM and 5-cross-validation to classify.
LIBSVM_CV5 = classify(data,classifyMethod="libsvm",cv=5,
               svm.kernel="linear",svm.scale=FALSE)
## Features selection is done by envoking "CfsSubsetEval" method in WEKA.
FS_LIBSVM_CV5 = classify(data,classifyMethod="libsvm",cv=5,evaluator="CfsSubsetEval",
               search="BestFirst",svm.kernel="linear",svm.scale=FALSE)
```

```
if(interactive()){

  KNN_CV5 = classify(data,classifyMethod="knn",cv=5,knn.k=1)

  RF_CV5 = classify(data,classifyMethod="randomForest",cv=5)

  TREE_CV5 = classify(data,classifyMethod="tree",cv=5)

  NNET_CV5 = classify(data,classifyMethod="nnet",cv=5)

  RPART_CV5 = classify(data,classifyMethod="rpart",cv=5,evaluator="")

  CTREE_CV5 = classify(data,classifyMethod="ctree",cv=5,evaluator="")

  BAG_CV5 = classify(data,classifyMethod="bagging",cv=5,evaluator="")

}
```

---

combine *Classify Functions*

Description

These functions use to test different feature coding schemes, and combine them to get better performance.

Usage

```
combine(seq, classLable, fileName, ele.type, featureMethod,
          cv=10, classifyMethod="libsvm",
          group=c("aaH", "aaV", "aaZ", "aaP", "aaF", "aaS", "aaE"), k, g,
          hydro.methods=c("kpm", "SARAH1"), hydro.indexs=c("hydroE", "hydroF",
          aaindex.name, n, d, w=0.05, start.pos, stop.pos, psiblast.path,
          database.path, hmmpfam.path, pfam.path, Evalue=10^-5,
          na.type="all", na.strand="all", diprodb.method="all", diprodb.type="
          svm.kernel="linear", svm.scale=FALSE, svm.path, svm.options="-t 0",
          knn.k=1, nnet.size=2, nnet.rang=0.7, nnet.decay=0, nnet.maxit=100)
```

Arguments

seq	a string vector for the protein, DNA, or RNA sequences.
classLable	a factor or vector for the class lable of sequences in seq.
fileName	a string for the output file name.
ele.type	a string for the type of biological sequence. This must be one of the strings "rnaBase", "dnaBase", "aminoacid" or "aminoacid2".
featureMethod	a string vector for the name of feature coding. The alternative names are "Binary", "CTD", "FragmentComposition", "GapPairComposition", "CKSAAP", "Hydro", "ACH", "AAindex", "ACI", "ACF", "PseudoAAComp", "PSSM", "DOMAIN", "BDNAVIDEO", and "DIPRODB".

`classifyMethod`	a string for the classification method. This must be one of the strings "libsvm", "svmlight", "NaiveBayes", "randomForest", "knn", "tree", "nnet", "rpart", "ctree", "ctreelibsvm", "bagging".
`cv`	an integer for the time of cross validation, or a string "leave_one_out" for the jacknife test.
`group`	a string vector for the group of amino acids. This alternative groups are: "aaH", "aaV", "aaZ", "aaP", "aaF", "aaS" or "aaE".
`k`	an integer indicating the length of sequence fragment (k>=1).
`g`	an integer indicating the distance between two aminoacids/bases (g>=0).
`hydro.methods`	a string vector for the methods of coding protein hydrophobic effect. This alternative groups are: "kpm" or "SARAH1".
`hydro.indexs`	a string vector for the methods of coding protein hydrophobic effect. This alternative groups are: "hydroE", "hydroF" or "hydroC".
`aaindex.name`	a string for the name of physicochemical and biochemical properties in AAindx.
`n`	an integer used as paramter of `featureACF` (1<=n<=L-2, L is the the length of sequence). featureACF takes the auto-correlation between fragment X(1)...X(L-m) and X(m+1)...X(L) (1<=m<=n) as features.
`d`	an integer used as paramter of `featurePseudoAAComp` (d>=1). Coupling between amino acids X(i) and X(i+d) are considered as features.
`w`	a numeric value for the weight factor of sequence order effect in `featurePseudoAAComp`.
`start.pos`	a integer vector denoting the start position of the fragment window. If it is missing, it is 1 by default.
`stop.pos`	a integer vector denoting the stop position of the fragment window. If it is missing, it is the length of sequence by default.
`psiblast.path`	a string for the path of PSI-BLAST program blastpgp. blastpgp will be employed to iteratively search database and generate position-specific scores for each position in the alignment.
`database.path`	a string for the path of formatted protein database. Database can be formatted by formatdb program.
`hmmpfam.path`	a string for the path of hammpfam program in HMMER. hammpfam will be employed to predict domains using models in Pfam database.
`pfam.path`	a string for the path of pfam domain database.
`Evalue`	a numeric value for the E-value cutoff of perdicted Pfam domain.
`na.type`	a string for nucleic acid type. It must be "DNA", "DNA/RNA", "RNA", or "all".
`na.strand`	a string for strand information. It must be "double", "single", or "all".
`diprodb.method`	a string for mode of property determination. It can be "experimental", "calculated", or "all".
`diprodb.type`	a string for property type. It can be "physicochemical", "conformational", "letter based", or "all".
`svm.kernel`	a string for kernel function of SVM.
`svm.scale`	a logical vector indicating the variables to be scaled.

`svm.path`	a character for path to SVMlight binaries (required, if path is unknown by the OS).
`svm.options`	Optional parameters to SVMlight. For further details see: "How to use" on http://svmlight.joachims.org/. (e.g.: "-t 2 -g 0.1"))
`nnet.size`	number of units in the hidden layer. Can be zero if there are skip-layer units.
`nnet.rang`	Initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(|x|) is about 1.
`nnet.decay`	parameter for weight decay.
`nnet.maxit`	maximum number of iterations.
`knn.k`	number of neighbours considered in function `classifyModelKNN`.

Details

`combine` can test feature coding methods for short peptide, protein, DNA or RNA. It returns a ranked list based on the accuracy of classification result. Each element in the list has three components: "data", "model", and "performance". "data" is a data.frame object, which stores feature matrix and its last column is the class label. "model" is a vector for feature coding method, which contains 6 elements: "Feature_Function", "Feature_Parameter", "Feature_Number", "Model", "Model_Parameter", and "Cross_Validataion". "performance" is a vector for the performance result of classification model, which contains 10 elements: "tp", "tn", "fp", "fn", "prcc", "sn", "sp", "acc", "mcc", "pc".

Author(s)

Hong Li

Examples

```
  ## read positive/negative sequence from files.
  tmpfile1 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep"
  tmpfile2 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.neg40.pep"
  posSeq = as.matrix(read.csv(tmpfile1,header=FALSE,sep="\t",row.names=1))[,1]
  negSeq = as.matrix(read.csv(tmpfile2,header=FALSE,sep="\t",row.names=1))[,1]
  seq=c(posSeq,negSeq)
  classLable=c(rep("+1",length(posSeq)),rep("-1",length(negSeq)) )
if(interactive()){
 ## test various feature coding methods.
 ## it may be time consuming.
  fileName = tempfile()
  testFeatureSet = combine(seq, classLable, fileName, ele.type="aminoacid",
            featureMethod=c("Binary", "CTD", "FragmentComposition", "GapPairComposition",
            "Hydro"), cv=5, classifyMethod="libsvm",
            group=c("aaH", "aaV", "aaZ", "aaP", "aaF", "aaS", "aaE"), k=3, g=7,
            hydro.methods=c("kpm", "SARAH1"), hydro.indexs=c("hydroE", "hydroF", "hydroC"
  summary = read.csv(fileName,sep="\t",header=T)
  fix(summary)

  ## combine features from different feature coding functions
  feature.index = 1:5
  tmp <- testFeatureSet[[1]]$data
  colnames(tmp) <- paste(testFeatureSet[[feature.index[1]]]$model["Feature_Function"],tes
  data <- tmp[,-ncol(tmp)]
  for(i in 2:length(feature.index) ){
    tmp <- testFeatureSet[[feature.index[i]]]$data
    colnames(tmp) <- paste(testFeatureSet[[feature.index[i]]]$model["Feature_Function"],t
    data <- data.frame(data, tmp[,-ncol(tmp)] )
  }
  name <- colnames(data)
  data <- data.frame(data, tmp[,ncol(tmp)] )
  ## feature forward selection by 'cv_FFS_classify'
  ## it is very time consuming.
  combineFeatureResult = cv_FFS_classify(data,stop.n=50,classifyMethod="knn",cv=5)
  tmp = sapply(combineFeatureResult,function(x){c(length(x$features),x$performance["acc"]
  plot(tmp[1,],tmp[2,],xlab="featureNumber",ylab="Accuracy",main="result of FFS_KNN",pch=
  lines(tmp[1,],tmp[2,])

  ## compare the prediction accuracy based on different feature coding methods and differ
  ## it is very time consuming.
  testResult = lapply(c("libsvm", "randomForest", "knn", "tree"),
    function(x){
            tmp = combine(seq, classLable, fileName = tempfile(),
            ele.type="aminoacid", featureMethod=c("Binary", "CTD", "FragmentComposition",
            "GapPairComposition", "Hydro"), cv=5, classifyMethod=x,
            group=c("aaH", "aaV", "aaZ", "aaP", "aaF", "aaS", "aaE"), k=3, g=7,
            hydro.methods=c("kpm", "SARAH1"), hydro.indexs=c("hydroE", "hydroF", "hydroC"
            sapply(tmp,function(y){c(y$model[["Feature_Function"]], y$model[["Feature_Par
  })
  tmpFeature = as.factor(c(sapply(testResult,function(x){apply(x[1:2,],2,function(y){past
  tmpModel = as.factor(c(sapply(testResult,function(x){x[3,]})))
  tmp1 = data.frame(as.integer(tmpFeature), as.integer(tmpModel), as.numeric(c(sapply(tes
  require(scatterplot3d)
  s3d=scatterplot3d(tmp1,color=c("red","blue","green","yellow")[tmp1[,2]],pch=19,
      xlab="Feature Coding", ylab="Classification Model",
      zlab="Accuracy under 5-fold cross validation",lab=c(10,6,7),
      y.ticklabs=c("",as.character(sort(unique(tmpModel))),"") )
}
```

`featureAAindex` *Feature Coding by physicochemical/biochemical properties in AAindex*

Description

Protein sequences are coded based on the physicochemical/biochemical properties of amino acids in AAindex database.

Usage

```
featureAAindex(seq,aaindex.name="all")
featureACI(seq,aaindex.name="all")
featureACF(seq,n,aaindex.name="all")
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`aaindex.name`	a string for the name of physicochemical and biochemical properties in AAindx.

`n`	an integer used as paramter of `featureACF` (1<=n<=L-2, L is the the length of sequence). featureACF takes the auto-correlation between fragment X(1)...X(L-m) and X(m+1)...X(L) (1<=m<=n) as features.

Details

`featureAAindex` returns a matrix measuring the physicochemical and biochemical properties of amino acids by AAindex (`http://www.genome.jp/aaindex`). If parameter aaindex.name="all", all properties in AAindex will be considered, and each row represented the features of one sequence coding by a 531*N dimension numeric vector. If parameter aaindex.name is a name of property in AAindex, each row represented the features of one sequence coding by a N dimension numeric vector.

`featureACI` returns a matrix with 531 columns, measuring the average cumulative value of AAindex. N is the length of input sequence, and N must be odd. Central residue of all windows are the central residue of input sequence. Each column is the average cumulative AAindex over a sliding window.

`featureACF` returns a matrix with 531*n columns, measuring the Auto-Correlation Function (ACF) of AAindex. If parameter aaindex.name is a name of property in AAindex, each row represented the features of one sequence coding by a n dimension numeric vector.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  AI_all = featureAAindex(seq)
  AI_ANDN920101 = featureAAindex(seq,"ANDN920101")

  ACI_all = featureACI(seq)
  ACI_ANDN920101 = featureACI(seq,"ANDN920101")

  ACF_all_1 = featureACF(seq,1)
  ACF_ANDN920101_3 = featureACF(seq,3,"ANDN920101")
}
```

`featureBDNAVIDEO` *Feature Coding by DNA/RNA property*

Description

DNA/RNA Sequences are coded with DNA or RNA property from B-DNA-VIDEO database.

Usage

```
featureBDNAVIDEO(seq)
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.

Details

`featureBDNAVIDEO` returns a matrix with 38 columns. Each column is the mean of DNA or RNA property from B-DNA-VIDEO database (`http://wwwmgs.bionet.nsc.ru/mgs/systems/bdnavideo/`).

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "test.rna")
  rna = as.matrix(read.csv(file,header=F,sep="\t"))[,1]

  BDNAVIDEO = featureBDNAVIDEO(rna)
}
```

`featureBinary` *Feature Coding by Binary Vectors*

Description

Sequences are coded by binary vectors.

Usage

```
featureBinary(seq,class=elements("aminoacid"))
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`class`	a list for the class of biological properties. It can be produced by `elements` and `aaClass`.

Details

`featureBinary` returns a matrix with M*N columns. Each row represented features of one sequence coding by a M*N dimension 0-1 vector. Each base/amino acid is coded as a M dimension vetor. For example: amino acid "A" is coded by "00000000000000000001"; base "T" is coded by "0010". The input sequences must have equal length.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  BIN1 = featureBinary(seq,elements("aminoacid"))
  BIN2 = featureBinary(seq,aaClass("aaE"))
}
```

`featureCKSAAP` *Feature Coding by k-spaced Aminoacids/Base Pairs*

Description

Protein sequences are coded based on the frequency of k-spaced aminoacids/base pairs.

Usage

```
featureCKSAAP(seq,g,class=elements("aminoacid"))
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`g`	an integer indicating the distance between two aminoacids/bases (g>=0).
`class`	a list for the class of biological properties. It can be produced by `elements` and `aaClass`.

Details

`featureCKSAAP` returns a matrix with (g+1)*M\^2 columns. Each row represented features of one sequence coding by a (g+1)*M\^2 dimension numeric vector. Each column is the number of k-spaced aminoacids/base pair (0<=k<=g).

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  CKSAAP0 = featureCKSAAP(seq,0,elements("aminoacid"))
  CKSAAP2 = featureCKSAAP(seq,2,elements("aminoacid"))
}
```

featureCTD *Feature Coding by composition, transition and distribution*

Description

Sequences are coded based on their composition, transition and distribution.

Usage

```
featureCTD(seq,class=elements("aminoacid"))
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`class`	a list for the class of biological properties. It can be produced by `elements` and `aaClass`.

Details

`featureCTD` returns a matrix with M+M*(M-1)/2+M*5 columns. Each row represented features of one sequence coding by a M+M*(M-1)/2+M*5 dimension numeric vector. Three kinds of coding: composition (C), transition (T) and distribution (D) are used. C is the number of amino acids of a particular property (such as hydrophobicity) divided by the total number of amino acids. T characterizes the percent frequency with which amino acids of a particular property is followed by amino acids of a different property. D measures the chain length within which the first, 25, 50, 75 and 100 acids of a particular property is located respectively.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.fasta")
  library(Biostrings)
  tmp = readFASTA(file)

  proteinSeq = sapply(tmp,function(x){x[["seq"]]})
  names(proteinSeq) = sapply(tmp,function(x){x[["desc"]]})

  CTD1 = featureCTD(proteinSeq, class=elements("aminoacid") )
  CTD2 = featureCTD(proteinSeq, class=aaClass("aaV") )
}
```

`featureDIPRODB` *Feature Coding by Dinucleotide Property*

Description

Sequences are coded by conformational or thermodynamic dinucleotide property from DiProDB database.

Usage

```
featureDIPRODB(seq, na.type="all", na.strand="all", diprodb.method="all",
  diprodb.type="all")
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`na.type`	a string for nucleic acid type. It must be "DNA", "DNA/RNA", "RNA", or "all".
`na.strand`	a string for strand information. It must be "double", "single", or "all".
`diprodb.method`	a string for mode of property determination. It can be "experimental", "calculated", or "all".
`diprodb.type`	a string for property type. It can be "physicochemical", "conformational", "letter based", or "all".

Details

`featureDIPRODB` returns a matrix with 122 columns. Each column is the mean of conformational or thermodynamic dinucleotide property from DiProDB database (`http://diprodb.fli-leibniz.de`).

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "test.rna")
  rna = as.matrix(read.csv(file,header=F,sep="\t"))[,1]

  DIPRODB1 = featureDIPRODB(rna)
  DIPRODB2 = featureDIPRODB(rna, na.type="RNA")
}
```

`featureDOMAIN` *Feature Coding by doamin organization*

Description

Protein sequences are coded based on their domains.

Usage

```
featureDOMAIN(domain)

# Protein Pfam domain prediction
predictPFAM(seq, hmmpfam.path, pfam.path, Evalue=10^-5)
```

Arguments

`domain`	a list of protein domains. It can be produced by function `predictPFAM`.
`seq`	a string vector for the protein, DNA, or RNA sequences.
`hmmpfam.path`	a string for the path of hammpfam program in HMMER. hammpfam will be employed to predict domains using models in Pfam database.
`pfam.path`	a string for the path of pfam domain database.
`Evalue`	a numeric value for the E-value cutoff of perdicted Pfam domain.

Details

`featureDOMAIN` uses Pfam domains to code 0-1 feature vector.

`predictPFAM` predict Pfam domains by hmmpfam program. It returns a list, each element is a vector which denotes the domain composition of a protein.

Author(s)

Hong Li

Examples

```
if(interactive()){

}
```

`featureFragmentComposition` *Feature Coding by the composition of k-mer fragments*

Description

Sequences are coded based on the frequency of k-mer sequence fragments.

Usage

```
featureFragmentComposition(seq,k,class=elements("aminoacid"))
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`k`	an integer indicating the length of sequence fragment (k>=1).
`class`	a list for the class of biological properties. It can be produced by `elements` and `aaClass`.

Details

`featureFragmentComposition` returns a matrix with M\^k columns. Each row represented features of one sequence coding by a M\^k dimension numeric vector. Each column is the frequency of k-mer sequence fragment.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  FC2 = featureFragmentComposition(seq,2,aaClass("aaS"))
  FC3 = featureFragmentComposition(seq,3,aaClass("aaS"))
}
```

`featureGapPairComposition` *Feature Coding by g-spaced aminoacids/bases pairs*

Description

Sequences are coded based on the frequency of g-spaced aminoacids/bases pairs.

Usage

```
featureGapPairComposition(seq,g,class=elements("aminoacid"))
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`g`	an integer indicating the distance between two aminoacids/bases (g>=0).
`class`	a list for the class of biological properties. It can be produced by `elements` and `aaClass`.

Details

`featureGapPairComposition` returns a matrix with M\^2 columns. Each row represented features of one sequence coding by a M\^2 dimension numeric vector. Each column is the frequency of g-spaced aminoacids/bases pair. featureFragmentComposition(seq,2) is same with featureGapPairComposition(seq,0).

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  GPC0 = featureGapPairComposition(seq,0,elements("aminoacid"))
  GPC2 = featureGapPairComposition(seq,2,elements("aminoacid"))
}
```

`featureHydro` *Feature Coding by hydrophobicity*

Description

Protein sequences are coded based on their hydrophobicity.

Usage

```
  featureHydro(seq,hydro.method="SARAH1")
  featureACH(seq,hydro.index="hydroE")
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`hydro.method`	a string for the method of coding protein hydrophobic effect. This must be one of the strings "kpm" or "SARAH1".
`hydro.index`	a string for the method of coding protein hydrophobic effect. This must be one of the strings "hydroE", "hydroF" or "hydroC".

Details

featureHydro returns a matrix measuring the hydrophobic effect. Parameter "hydro.method" supported following coding methods: "kpm": use a numeral to indicating the hydrophobic effect of amino acid. Each sequence is coded by a N dimension numeric vector. "SARAH1": use a 5 dimension 0-1 vector to indicating the hydrophobic effect of amino acid. Each sequence is coded by a 5*N dimension 0-1 vector.

featureACH returns a matrix with (N-1)/2 columns. N is the length of input sequence, andis N must be odd. Central residue of all windows are the central residue of input sequence. Each column is the average cumulative hydrophobicity over a sliding window.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  H1 = featureHydro(seq,"kpm")
  H2 = featureHydro(seq,"SARAH1")

  H3 = featureACH(seq,hydro.index="hydroE")
  H3 = featureACH(seq,hydro.index="hydroF")
  H3 = featureACH(seq,hydro.index="hydroC")
}
```

---

featurePseudoAAComp *Feature Coding by Pseudo Amino Acid Composiion*

Description

Protein sequences are coded by pseudo amino acid composiion.

Usage

```
featurePseudoAAComp(seq,d,w=0.05)
```

Arguments

seq	a string vector for the protein, DNA, or RNA sequences.
d	an integer used as paramter of featurePseudoAAComp (d>=1). Coupling between amino acids X(i) and X(i+d) are considered as features.
w	a numeric value for the weight factor of sequence order effect in featurePseudoAAComp.

Details

`featurePseudoAAComp` returns a matrix representing the pseudo amino acid composiion. Each row represented features of one sequence coding by a 20+d dimension numeric vector. The first 20 features indicates the composition of 20 amino acids. The last d features indicates the coupling between amino acids X(i) and X(i+d). Coupling value is cacluated by hydrophobicity, hydrophilicity and mass of amino acids.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep")
  seq = as.matrix(read.csv(file,header=F,sep="\t",row.names=1))[,1]

  PAC4 = featurePseudoAAComp(seq,4)
}
```

`featurePSSM` *Feature Coding*

Description

A set of functions for extract features from biological sequences, and coding features by numeric vector.

Usage

```
featurePSSM(seq, start.pos, stop.pos, psiblast.path, database.path)
```

Arguments

`seq`	a string vector for the protein, DNA, or RNA sequences.
`start.pos`	a integer vector denoting the start position of the fragment window.
`stop.pos`	a integer vector denoting the stop position of the fragment window.
`psiblast.path`	a string for the path of blastpgp program. blastpgp will be employed to do PSI-BLAST and get Position-Specific Scoring Matrix.
`database.path`	a string for the path of a formated reference database. Database can be formated by "formatdb" program.

Details

`featurePSSM` returns a matrix with 20*N+N columns. Each row represented features of one sequence coding by a 20*N+N dimension numeric vector generated by PSI-BLAST. It contains two kinds of fatures: normalized position-specific score of PSSM (Position-Specific Scoring Matrix), Shannon entropy for each position of WOP (weighted observed percentages). Program PSI-BLAST and formatted NCBI non-redundant protein database are needed.

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.fasta")
  tmp = readFASTA(file)
  proteinSeq = sapply(tmp,function(x){x[["seq"]]})
  names(proteinSeq) = sapply(tmp,function(x){x[["desc"]]})

  ## Need "blastpgp" program and a formated database. Database can be formated by "format
  PSSM1 = featurePSSM(proteinSeq[1:2], start.pos=rep(1,2), stop.pos=rep(10,2), psiblast.p
}
```

`featureSSC` *Feature Coding by secondary structure*

Description

It is suitable for peptides with odd residues and the central residue has important role.

Usage

```
featureSSC(secondaryStructure, confidenceScore)

# secondary structure from DSSP database
getDSSP(pdb)
# Protein secondary structure prediction
predictPROTEUS(seq,proteus2.organism="euk")
```

Arguments

`secondaryStructure`	a string vector for the protein secondary structure. It is consisted of three kinds of secondary structures: H = Helix, E = Beta Strand, C = Coil.
`confidenceScore`	a string vector for the confidence score of secondary structure prediction (0-9, 0 = low, 9 = high).
`pdb`	a string vector for the name of pdb structure. (e.g. "43ca")
`seq`	a string vector for the protein, DNA, or RNA sequences.
`proteus2.organism`	a string for the organism of proteus2 program. This must be one of the strings "gram-", "gram+", "euk".

Details

featureSSC codes for the secondary structure of the central residue of peptides. It is suitable for peptides with odd residues and the central residue has important role.

getDSSP returns a vector of secondary structure extracted from DSSP database (http://swift.cmbi.ru.nl/gv/dssp/).

predictPROTEUS predicts secondary structure based on protein sequence using following methods : "PROTEUS2", "PSIPRED", "JNET", "TRANSSEC", "JURY-OF-EXPERTS PREDICTION". Parameter "proteus2.organism" can be "gram-" for "Gram negative prokaryote", "gram+" for "Gram positive prokaryote", "euk" for "Eukaryote". It returns.....

Author(s)

Hong Li

Examples

```
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.fasta")
  tmp = readFASTA(file)
  proteinSeq = sapply(tmp,function(x){x[["seq"]]})
  names(proteinSeq) = sapply(tmp,function(x){x[["desc"]]})

  DSSP1 = getDSSP(c("108l","43ca"))
  DSSP2 = getDSSP(c("108l","43ca","aaaa"))

  ## Predict protein secordary strucutre
  PROTEUS = predictPROTEUS(proteinSeq[1:2],proteus2.organism="euk")

  ## Use general feature conding functions to codes protein secordary strucutre
  secondaryStructure = sapply(PROTEUS,function(x){paste(x[["PROTEUS2"]]$SecondaryStructur
  confidenceScore = sapply(PROTEUS,function(x){paste(x[["PROTEUS2"]]$ConfidenceScore,coll
  SSCTD = featureCTD(secondaryStructure, class=list("H"="H","E"="E","C"="C"))
  # Codes for peptides which have equal length and their central residues are important
  secondaryStructure = sapply(PROTEUS,function(x){sub.seq(paste(x[["PROTEUS2"]]$Secondary
  confidenceScore = sapply(PROTEUS,function(x){sub.seq(paste(x[["PROTEUS2"]]$ConfidenceSc

  SS1 = featureSSC(secondaryStructure, confidenceScore)
}
```

---

hr   *Homolog Reduction*

Description

Filter homolog sequences by sequence similarity.

Usage

```
hr(seq, method, identity, cdhit.path)

cdhitHR(seq, identity=0.3, cdhit.path)
aligndisHR(seq, identity=0.6)
```

```
distance(seq1,seq2)

getTrain(seqfile, posfile, aa, w, identity, balance=T)
getNegSite(posSite, seq, aa)
```

Arguments

`seq`	a list with one element for each protein/gene sequence. The elements are in two parts, one the description and the second a character string of the biological sequence.
`identity`	a numeric value ranged from 0 to 1. It is used as a maximum identity cutoff among input sequences.
`method`	a string for the method of homolog reدunction. This must be one of the strings "cdhit" or "aligndis".
`cdhit.path`	a string for the path of cdhit program directory. eg: "/people/hongli/cd-hit". It is necessary when method="cdhit".
`seq1`	a string for the protein or gene sequence.
`seq2`	a string for the protein or gene sequence. seq1 and seq2 must have same length.
`seqfile`	a string for the name of FASTA file.
`posfile`	a string for the name of file which contains the positive site dataset. It has two columns: 1st column is the protein name; 2st column is the positive site. Protein name should be consistent with the name used in seqfile.
`aa`	a character for the interested amino acid. eg: "C".
`w`	an integer for the window size of flanking peptide sequence. Window size is 2*w+1, and the central residues are the positive sites in posfile.
`balance`	a logical value indicating whether negative sites will be random selected to have the same number with positive sites.
`posSite`	a string vector for the positive sites. It is consisted of protein description and positive site, eg: "P278168:952".

Details

`hr` employs `cdhitHR` and `aligndisHR` to filter homolog sequences. It supported following methods:

"cdhit": Use cd-hit program to quickly filter sequences by given identity. It is designed to filter full-length protein or gene sequences. "formatdb" and "blastall" are required for running cd-hit program. (http://www.bioinformatics.org/download.php/cd-hit/cd-hit-2007-0131.tar.gz or http://www.bioinformatics.org/downloa
hit/cd-hit-2007-0131-win32.tar.gz)

"aligndis": Use the number of different residues to meature the identity between two sequences. It is designed to filter aligned seuqnces with equal length.

`getTrain` extract 2*w+1 flanking peptides of positive sites and filter homolog sequences. Negative sites are non-positive sites in the same proteins.

`distance` calculate the number of positions with different residues between two sequences.

Value

`hr` return a list of reduced sequences.

Author(s)

Hong Li

Examples

```
  distance("AABD","ACBD")
  distance("AABD","ECBD")
if(interactive()){
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.fasta")
  library(Biostrings)
  tmp = readFASTA(file)
  seq <- sapply(tmp,function(x){x[["seq"]]})
  names(seq) <- sapply(tmp,function(x){x[["desc"]]})
  ## Homolog reduction of whole-length sequence by cd-hit
  # need cd-hit program;
  reducSeq50 = hr(seq, method="cdhit", identity=0.5, cdhit.path="/people/hongli/cd-hit")

  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.site")
  tmp = as.matrix(read.csv(file, sep="\t",header=F))
  logical = apply(tmp,1,function(x){ l=length(unlist(strsplit(seq[x[1]],split=""))); (l>=
  fragment = sub.seq(seq[tmp[logical,1]], as.numeric(tmp[logical,2])-7, as.numeric(tmp[lo
  ## Homolog reduction of short sequence fragment
  # It may be slow.
  reducSeq = hr(fragment, method="aligndis", identity=0.4)

  ## produce train set based on given positive sites and fasta sequences.
  file = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.fasta")
  posfile = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.site")
  ## "getTrain" integrate negative set construction and homolog reduction. It is designec
  # It may be very slow.
  data = getTrain(file, posfile, aa="K", w=7, identity=0.4)
}
```

`model` *Classification Model*

Description

These functions build classification model and evaluate performance.

Usage

```
classifyModelLIBSVM(train,svm.kernel="linear",svm.scale=FALSE)
classifyModelSVMLIGHT(train,svm.path,svm.options="-t 0")
classifyModelNB(train)
classifyModelRF(train)
classifyModelKNN(train, test, knn.k=1)
classifyModelTree(train)
classifyModelNNET(train, nnet.size=2, nnet.rang=0.7, nnet.decay=0, nnet.maxit=
classifyModelRPART(train)
classifyModelCTREE(train)
classifyModelCTREELIBSVM(train, test, svm.kernel="linear",svm.scale=FALSE)
classifyModelBAG(train)
```

Arguments

`train`	a data frame including the feature matrix and class label. The last column is a vector of class label comprising of "-1" or "+1"; Other columns are features.
`svm.kernel`	a string for kernel function of SVM.
`svm.scale`	a logical vector indicating the variables to be scaled.
`svm.path`	a character for path to SVMlight binaries (required, if path is unknown by the OS).
`svm.options`	Optional parameters to SVMlight. For further details see: "How to use" on http://svmlight.joachims.org/. (e.g.: "-t 2 -g 0.1"))
`nnet.size`	number of units in the hidden layer. Can be zero if there are skip-layer units.
`nnet.rang`	Initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(|x|) is about 1.
`nnet.decay`	parameter for weight decay.
`nnet.maxit`	maximum number of iterations.
`knn.k`	number of neighbours considered in function `classifyModelKNN`.
`test`	a data frame including the feature matrix and class label. The last column is a vector of class label comprising of "-1" or "+1"; Other columns are features.

Details

`classifyModelLIBSVM` builds support vector machine model by LibSVM. R package "e1071" is needed.

`classifyModelSVMLIGHT` builds support vector machine model by SVMlight. R package "klaR" is needed.

`classifyModelNB` builds naive bayes model. R package "klaR" is needed.

`classifyModelRF` builds random forest model. R package "randomForest" is needed.

`classifyModelKNN` builds k-nearest neighbor model. R package "class" is needed.

`classifyModelTree` builds tree model. R package "class" is needed.

`classifyModelRPART` builds recursive partitioning trees model. R package "rpart" is needed.

`classifyModelCTREE` builds conditional inference trees model. R package "party" is needed.

`classifyModelCTREELIBSVM` combines conditional inference trees and support vecotr machine. R package "party" and "e1071" is needed.

`classifyModelBAG` uses bagging method. R package "ipred" is needed.

Author(s)

Hong Li

Examples

```
## read positive/negative sequence from files.
tmpfile1 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep"
tmpfile2 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.neg40.pep"
posSeq = as.matrix(read.csv(tmpfile1,header=FALSE,sep="\t",row.names=1))[,1]
negSeq = as.matrix(read.csv(tmpfile2,header=FALSE,sep="\t",row.names=1))[,1]
data = data.frame(rbind(featureBinary(posSeq,elements("aminoacid")),
     featureBinary(negSeq,elements("aminoacid")) ),
     class=c(rep("+1",length(posSeq)),
     rep("-1",length(negSeq))) )

## sample train and test data
tmp = c(sample(1:length(posSeq),length(posSeq)*0.8),
  sample(length(posSeq)+(1:length(negSeq)),length(negSeq)*0.8))
train = data[tmp,]
test = data[-tmp,]

## Build classification model using training data
model1 = classifyModelLIBSVM(train,svm.kernel="linear",svm.scale=FALSE)
## Predict test data by classification model
testClass = predict(model1, test[,-ncol(test)])
```

`performance` *Performance Evaluation*

Description

Evaluate the performance of classification model.

Usage

```
performance(predictClass,factClass)
```

Arguments

`predictClass`	a factor of predicted classifications of training set, comprising of "-1" or "+1".
`factClass`	a vector of true classifications of training set, comprising of "-1" or "+1".

Details

`performance` evaluates the performance of classification model. It cacluates: tp (true positive), tn(ture negative), fp(false positive), fn(false negative), prc(precision), sn(sensitivity), sp(specificity), acc(accuracy), mcc(Matthews Correlation Coefficient), pc(Performance Coefficient).

Author(s)

Hong Li

Examples

```
     ## read positive/negative sequence from files.
     tmpfile1 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep"
     tmpfile2 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.neg40.pep"
     posSeq = as.matrix(read.csv(tmpfile1,header=FALSE,sep="\t",row.names=1))[,1]
     negSeq = as.matrix(read.csv(tmpfile2,header=FALSE,sep="\t",row.names=1))[,1]
     data = data.frame(rbind(featureBinary(posSeq,elements("aminoacid")),
          featureBinary(negSeq,elements("aminoacid")) ),
          class=c(rep("+1",length(posSeq)),
          rep("-1",length(negSeq))) )
```

```
## sample train and test data
tmp = c(sample(1:length(posSeq),length(posSeq)*0.8),
  sample(length(posSeq)+(1:length(negSeq)),length(negSeq)*0.8))
train = data[tmp,]
test = data[-tmp,]

## Build classification model using training data
model1 = classifyModelLIBSVM(train,svm.kernel="linear",svm.scale=FALSE)
## Predict test data by classification model
testClass = predict(model1, test[,-ncol(test)])
## Evaluate the performance of classification model
performance(testClass,test[,ncol(test)])
```

`selectFFS` *Feature Selection by FFS*

Description

feature forward selection.

Usage

```
selectFFS(data, accCutoff, stop.n,
          classifyMethod="knn",cv=10)
```

Arguments

`data`	a data frame including the feature matrix and class label. The last column is a vector of class label comprising of "-1" or "+1"; Other columns are features.
`accCutoff`	a numeric indicating the minimum difference of accuracy between two models in `selectFFS`. Feature subsets will stop increasing when the difference of accuracy is samll than accCutoff.
`stop.n`	number of selected features by `selectFFS`.
`classifyMethod`	a string for the classification method. This must be one of the strings "libsvm", "svmlight", "NaiveBayes", "randomForest", "knn", "tree", "nnet", "rpart", "ctree", "ctreelibsvm", "bagging".
`cv`	an integer for the time of cross validation, or a string "leave_one_out" for the jacknife test.

Details

`selectFFS` uses FFS (Feature Forword Selection) method to increase feature, and use NNA (Neareast Neighbor Analysis) to evaluate the performance of feature subset. Two conditions are used to stop feature increasing: control the difference of accuracy between two models; control the number of selected features by Parameter "stop.n".

Author(s)

Hong Li

Examples

```
  ## read positive/negative sequence from files.
  tmpfile1 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.pos40.pep"
  tmpfile2 = file.path(.path.package("BioSeqClass"), "example", "acetylation_K.neg40.pep"
  posSeq = as.matrix(read.csv(tmpfile1,header=FALSE,sep="\t",row.names=1))[,1]
  negSeq = as.matrix(read.csv(tmpfile2,header=FALSE,sep="\t",row.names=1))[,1]
  seq=c(posSeq,negSeq)
  classLable=c(rep("+1",length(posSeq)),rep("-1",length(negSeq)) )
  data = data.frame(featureBinary(seq),classLable)

if(interactive()){
  ## Use KNN to evaluate the performance of feature subset,
  ## and use Feature Forword Selection method to increase feature.
  # If the difference of accuracy between two models is less than 0.01, feature
  # selection will stop.
  FFS_NNA_CV5 = selectFFS(data,accCutoff=0.01,classifyMethod="knn",cv=5)
  # If 20 features have been selected, feature selection will stop.
  FFS_NNA_CV5 = selectFFS(data,stop.n=3,classifyMethod="knn",cv=5)
  # If any one condiction is satisfied, feature selection will stop.
  FFS_NNA_CV5 = selectFFS(data,accCutoff=0.001,stop.n=100,classifyMethod="knn",cv=5)
}
```

---

selectWeka *Feature Selection by Weka*

Description

feature selection by Weka.

Usage

```
selectWeka(train, evaluator="CfsSubsetEval", search="BestFirst", n)
```

Arguments

train	a data frame including the feature matrix and class label of training set.
evaluator	a string for the feature selection method used by WEKA. This must be one of the strings "CfsSubsetEval", "ChiSquaredAttributeEval", "InfoGainAttributeEval", or "SVMAttributeEval".
search	a string for the search method used by WEKA. This must be one of the strings "BestFirst" or "Ranker".
n	an integer for the number of selected features.

Details

Parameter "evaluator" supportes three feature selection methods provided by WEKA: "CfsSubsetEval": Evaluate the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. "ChiSquaredAttributeEval": Evaluate the worth of an attribute by computing the value of the chi-squared statistic with respect to the class. "InfoGainAttributeEval": Evaluate attributes individually by measuring information gain with respect to the class. "SVMAttributeEval": Evaluate the worth of an attribute by using an SVM classifier. Attributes are ranked by the square of the weight assigned by the SVM. Attribute selection for multiclass problems is handled by ranking attributes for each class seperately using a one-vs-all method and then "dealing" from the top of each pile to give a final ranking.

Parameter "search" supportes three feature subset search methods provided by WEKA: "BestFirst": Searches the space of attribute subsets by greedy hillclimbing augmented with a backtracking facility. Setting the number of consecutive non-improving nodes allowed controls the level of backtracking done. Best first may start with the empty set of attributes and search forward, or start with the full set of attributes and search backward, or start at any point and search in both directions (by considering all possible single attribute additions and deletions at a given point). "Ranker": Ranks attributes by their individual evaluations.

Author(s)

Hong Li

`zzz` *Load packages and data*

Description

This functions load depended R packages and imports default data into global "options".

Usage

```
.onLoad(libname, pkgname)
```

Arguments

`libname`	a character string giving the library directory where the package defining the namespace was found.
`pkgname`	a character string giving the name of the package.

Details

After loading, loadNamespace looks for a hook function named .onLoad and runs it before sealing the namespace and processing exports.

Author(s)

Hong Li

Index