Package 'kebabs'

April 10, 2015

Type Package

Title Kernel-Based Analysis Of Biological Sequences

Version 1.0.5

Date 2015-02-09

Author Johannes Palme

Maintainer Ulrich Bodenhofer <bodenhofer@bioinf.jku.at>

Description The package provides functionality for kernel-based analysis of DNA, RNA, and amino acid sequences via SVM-based methods. As core functionality, kebabs implements following sequence kernels: spectrum kernel, mismatch kernel, gappy pair kernel, and motif kernel. Apart from an efficient implementation of standard position-independent functionality, the kernels are extended in a novel way to take the position of patterns into account for the similarity measure. Because of the flexibility of the kernel formulation, other kernels like the weighted degree kernel or the shifted weighted degree kernel with constant weighting of positions are included as special cases. An annotation-specific variant of the kernels uses annotation information placed along the sequence together with the patterns in the sequence.
The package allows for the generation of a kernel matrix or an explicit feature representation in dense or sparse format for all available kernels which can be used with methods implemented in other R packages. With focus on SVM-based methods, kebabs provides a framework which simplifies the usage of existing SVM implementations in kernlab, e1071, and LiblineaR. Binary and multi-class classification as well as regression tasks can be used in a unified way without having to deal with the different functions, parameters, and formats of the selected SVM. As support for choosing hyperparameters, the package provides cross validation - including grouped cross validation, grid search and model selection functions. For easier biological interpretation of the results, the package computes feature weights for all SVMs and prediction profiles which show the contribution of individual sequence positions to the prediction result and indicate the

relevance of sequence sections for the learning result and the underlying biological functions.

URL http://www.bioinf.jku.at/software/kebabs/

Roxygen list(wrap=TRUE)

License GPL (>= 2.1)

Collate AllClasses.R AllGenerics.R access-methods.R svmModel.R kebabs.R kebabsData.R runtimeMessage.R parameters.R sequenceKernel.R annotationSpecificKernel.R positionDependentKernel.R spectrum.R mismatch.R gappyPair.R motif.R explicitRepresentation.R coerce-methods.R featureWeights.R kbsvm-methods.R performCrossValidation-methods.R gridSearch.R modelSelection.R trainsvm-methods.R predictsvm-methods.R predict-methods.R predictionProfile.R plot-methods.R kebabsDemo.R show-methods.R symmetricPair.R utils.R zzz.R

Depends R (>= 3.1.0), Biostrings (>= 2.33.14), kernlab

Imports methods, Rcpp (>= 0.11.2), Matrix, XVector (>= 0.5.8), S4Vectors (>= 0.2.4), e1071, LiblineaR

LinkingTo IRanges, XVector, Biostrings, Rcpp, S4Vectors

Suggests SparseM, apcluster, Biobase, BiocGenerics

biocViews SupportVectorMachine, Classification, Clustering, Regression

R topics documented:

`BioVector` *DNAVector, RNAVector, AAVector Objects and BioVector Class*

Description

Create an object containing a set of DNA-, RNA- or amino acid sequences

Usage

```
## Constructors:

RNAVector(x = character())

AAVector(x = character())

## Accessor-like methods: see below

## S4 method for signature BioVector,index,missing,ANY
x[i]

## S4 method for signature BioVector
as.character(x, use.names = TRUE)
```

Arguments

x	character vector containing a set of sequences as uppercase characters or in mixed uppercase/lowercase form.
i	numeric vector with indicies or character with element names
use.names	when set to TRUE the names are preserved

Details

The class DNAVector is used for storing DNA sequences, RNAVector for RNA sequences and AAVector for amino acid sequences. The class BioVector is derived from the R base type character representing a vector of character strings. It is an abstract class which can not be instantiated. BioVector is the parent class for DNAVector, RNAVector and AAVector. For the three derived classes identically named functions exist which are constructors. It should be noted that the constructors only wrap the sequence data into a class without copying or recoding the data.

The functions provided for DNAVector, RNAVector and AAVector classes are only a very small subset compared to those of XStringSet but are designed along their counterparts from the Biostrings package. Assignment of metadata and element metadata via mcols is supported for the DNAVector, RNAVector and AAVector objects similar to objects of XStringSet derived classes (for details on metadata assignment see annotationMetadata and positionMetadata).

In contrast to XStringSet the BioVector derived classes also support the storage of lowercase characters. This can be relevant for repeat regions which are often coded in lowercase characters. During the creation of XStringSet derived classes the lowercase characters are converted to uppercase automatically and the information about repeat regions is lost. For BioVector derived classes the user can specify during creation of a sequence kernel object whether lowercase characters should be included as uppercase characters or whether repeat regions should be ignored during sequence analysis. In this way it is possible to perform both types of analysis on the same set of sequences through defining one kernel object which accepts lowercase characters and another one which ignores them.

Value

constructors DNAVector, RNAVector, AAVector return a sequence set of identical class name

Accessor-like methods

In the code snippets below, x is a BioVector.

length(x): the number of sequences in x.

width(x): vector of integer values with the number of bases/amino acids for each sequence in the set.

names(x): character vector of sample names.

Subsetting and concatination

In the code snippets below, x is a BioVector.

x[i]: return a BioVector object that only contains the samples selected with the subsetting parameter i. This parameter can be a numeric vector with indices or a character vector which is matched against the names of x. Element related metadata is subsetted accordingly if available.

c(x, ...): return a sequence set that is a concatination of the given sequence sets.

Coercion methods

In the code snippets below, x is a BioVector.

as.character(x, use.names=TRUE): return the sequence set as named or unnamed character vector dependent on the use.names parameter.

Note

Sequence data can be processed by KeBABS in XStringSet and BioVector based format. Within KeBABS except for treatment of lowercase characters both formats are equivalent. It is recommended to use XStringSet based formats whenever the support of lowercase characters is not of interest because these classes provide in general much richer functionality than the BioVector classes. String kernels provided in the kernlab package (see stringdot) do not support XStringSet derived objects. The usage of these kernels is possible in KeBABS with sequence data in BioVector based format.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

See Also

metadata, elementMetadata, XStringSet, DNAStringSet, RNAStringSet, AAStringSet

Examples

```
## in general DNAStringSet should be prefered as described above
## create DNAStringSet object for a set of sequences
x <- DNAStringSet(c("AACCGCGATTATCGatatatatatatatatTGGAAGCTAGGACTA",
                    "GACTTACCCgagagagagagagaCATGAGAGGGAAGCTAGTA"))
## assign names to the sequences
names(x) <- c("Sample1", "Sample2")

## to show the different handling of lowercase characters
## create DNAVector object for the same set of sequences and assign names
xv <- DNAVector(c("AACCGCGATTATCGatatatatatatatatTGGAAGCTAGGACTA",
                  "GACTTACCCgagagagagagagaCATGAGAGGGAAGCTAGTA"))
names(xv) <- c("Sample1", "Sample2")

## show DNAStringSet object - lowercase characters were translated
```

```
x
## in the DNAVector object lowercase characters are unmodified
## their handling can be defined at the level of the sequence kernel
xv

## show number of the sequences in the set and their number of characters
length(xv)
width(xv)
nchar(xv)
```

BioVector-class *BioVector, DNAVector, RNAVector and AAVector Classes*

Description

BioVector, DNAVector, RNAVector and AAVector Classes

Details

This class is the parent class for representing sets of biological sequences with support of lowercase characters. The derived classes DNAVector, RNAVector and AAVector hold DNA-, RNA- or AA-sequences which can contain also lowercase characters. In many cases repeat regions are coded as lowercase characters and with the BioVector based classes sequence analysis with and without repeat regions can be performed from the same sequence set. Whenever lowercase is not needed please use the XStringSet based classes as they provide much richer functionality. The class BioVector is derived from "character" and holds the sequence information as character vector. Interfaces for the small set of functions needed in KeBABS are designed consistent with XStringSet.

Instances of the DNAVector class are used for representing sets of DNA sequences.

Instances of the RNAVector class are used for representing sets of RNA sequences.

Instances of the AAVector class are used for representing sets of amino acid sequences.

Slots

NAMES sequence names

elementMetadata element metadata, which is applicable per element and holds a DataTable with one entry per sequence in each column. KeBABS uses the column names "annotation" and "offset".

metadata metadata applicable for the entire sequence set as list. KeBABS stores the annotation character set as list element named "annotationCharset".

`ControlInformation-class` *KeBABS Control Information Class*

Description

KeBABS Control Information Class

Details

Instances of this class store control information for the KeBABS meta-SVM.

Slots

`classification` indicator for classification task

`multiclassType` type of multiclass SVM

`featureWeights` feature weights control information

`selMethod` selected processing method

`onlyDense` indicator that only dense processing can be performed

`sparse` indicator for sparse processing

`runtimeWarning` indicator for runtime warning

`CrossValidationResult-class` *Cross Validation Result Class*

Description

Cross Validation Result Class

Details

Instances of this class store the result of cross validation.

Slots

`cross` number of folds for cross validation

`noCross` number of CV runs

`groupBy` group assignment of samples

`perfParameters` collected performance parameters

`outerCV` flag indicating outer CV

`folds` folds used in CV

`cvError` cross validation error

`foldErrors` fold errors

`noSV` number of support vectors

`ACC` cross validation accuracy

`BACC` cross validation balanced accuracy

`MCC` cross validation Matthews correlation coefficient

`foldACC` fold accuracy

`foldBACC` fold balanced accuracy

`foldMCC` fold Matthews correlation coefficient

`sumAlphas` sum of alphas

`evaluatePrediction` *Evaluate Prediction*

Description

Evaluate performance results of prediction on a testset based on given labels for binary classification

Usage

```
evaluatePrediction(prediction, label, allLabels = NULL, print = TRUE,
  confmatrix = TRUE, numPrecision = 3, numPosNegTrainSamples = numeric(0))
```

Arguments

`prediction`	prediction results as returned by `predict` for predictionType="response".
`label`	label vector of same length as parameter 'prediction'.
`allLabels`	vector containing all occuring labels once. This parameter is required only if the label vector is numeric. Default=NULL
`print`	This parameter indicates whether performance values should be printed or returned as data frame without printing (for details see below). Default=TRUE
`confmatrix`	When set to TRUE a confusion matrix is printed. The rows correspond to predictions, the columns to the true labels. Default=TRUE
`numPrecision`	minimum number of digits to the right of the decimal point. Values between 0 and 20 are allowed. Default=3
`numPosNegTrainSamples`	optional integer vector with two values giving the number of positive and negative training samples. When this parameter is set the balancedness of the training set is reported. Default=numeric(0)

Details

For binary classfication this function computes the performance measures accuracy, balanced accuracy, sensitivity, specificity, precision and the Matthews Correlation Coefficient(MCC). When the number of positive and negative training samples is passed to the function it also shows the balancedness of the training set. The performance results are either printed by the routine directly or returned in a data frame. The columns of the data frame are:

column name	performance measure
TP	true positive
FP	false positive
FN	false negative
TN	true negative
ACC	accuracy
BAL_ACC	balanced accuracy
SENS	sensitivity
SPEC	specificity
PREC	precision
MAT_CC	Matthews correlation coefficient
PBAL	prediction balancedness (fraction of positive samples)
TBAL	training balancedness (fraction of positive samples)

Value

When the parameter 'print' is set to FALSE the function returns a data frame containing the prediction performance values (for details see above).

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

predict, kbsvm

Examples

```
## set seed for random generator, included here only to make results
## reproducable for this example
set.seed(456)
## load transcription factor binding site data
data(TFBS)
enhancerFB
## select 70% of the samples for training and the rest for test
train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
test <- c(1:length(enhancerFB))[-train]
## create the kernel object for gappy pair kernel with normalization
gappy <- gappyPairKernel(k=1, m=3)
## show details of kernel object
gappy

## run training with explicit representation
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
```

```
                pkg="LiblineaR", svm="C-svc", cost=80, explicit="yes",
                featureWeights="no")

## predict the test sequences
pred <- predict(model, enhancerFB[test])

## print prediction performance
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))

## Not run:
## print prediction performance including training set balance
trainPosNeg <- c(length(which(yFB[train] == 1)),
                 length(which(yFB[train] == -1)))
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB),
                   numPosNegTrainSamples=trainPosNeg)

## or get prediction performance as data frame
perf <- evaluatePrediction(pred, yFB[test], allLabels=unique(yFB),
                           print=FALSE)

## show performance values in data frame
perf

## End(Not run)
```

`ExplicitRepresentation` *Explicit Representation Dense and Sparse Classes*

Description

Explicit Representation Dense and Sparse Classes

Details

In KeBABS this class is the virtual parent class for explicit representations generated from a set of biological sequences for a given kernel. The derived classes `ExplicitRepresentationDense` and `ExplicitRepresentationSparse` are meant to hold explicit representations in dense or sparse format. The kernel used to generate the explicit representation is stored together with the data.

Instances of this class are used for storing explicit representations in dense matrix format. This class is derived from `ExplicitRepresentation`.

Instances of this class are used for storing explicit representations in sparse `dgRMatrix` format. This class is derived from `ExplicitRepresentation`.

Slots

`usedKernel` kernel used for generating the explicit representation

`quadratic` boolean indicating a quadratic explicit representation

ExplicitRepresentationAccessors
ExplicitRepresentation Accessors

Description

ExplicitRepresentation Accessors

Usage

```
## S4 methods for signature ExplicitRepresentation
## x[i,j]

## further methods see below

## S4 method for signature matrix,dgRMatrix
x %*% y

## S4 method for signature dgRMatrix,numeric
x %*% y
```

Arguments

x	an explicit representation in dense or sparse format
i	integer vector or character vector with a subset of the sample indices or names
y	in the first case and explicit representation and x is a matrix, for the second case a numeric matrix and x is an explicit representation
j	integer vector or character vector with a subset of the feature indices or names

Value

see details above

Accessor-like methods

x[i,]: return a KernelMatrix object that only contains the rows selected with the subsetting parameter i. This parameter can be a numeric vector with indices or a character vector which is matched against the names of x.

x[,j]: return a KernelMatrix object that only contains the columns selected with the subsetting parameter j. This parameter can be a numeric vector with indices or a character vector which is matched against the names of x.

x[i,j]: return a KernelMatrix object that only contains the rows selected with the subsetting parameter i and columns selected by j. Both parameters can be a numeric vector with indices or a character vector which is matched against the names of x.

Accessor-like methods

`%*%`: this function provides the multiplication of a `dgRMatrix` or a sparse explicit representation (which is derived from `dgRMatrix`) with a matrix or a vector. This functionality is not available in package **Matrix** for a `dgRMatrix`.

`gappyPairKernel` *Gappy Pair Kernel*

Description

Create a gappy pair kernel object and the kernel matrix

Usage

```
gappyPairKernel(k = 1, m = 1, r = 1, annSpec = FALSE,
  distWeight = numeric(0), normalized = TRUE, exact = TRUE,
  ignoreLower = TRUE, presence = FALSE, revComplement = FALSE,
  mixCoef = numeric(0))

## S4 method for signature GappyPairKernel
getFeatureSpaceDimension(kernel, x)
```

Arguments

`k`	length of the substrings (also called kmers) which are considered in pairs by this kernel. This parameter together with parameter m (see below) defines the size of the feature space, i.e. the total number of features considered in this kernel is (|A|^(2*k))*(m+1), with |A| as the size of the alphabet (4 for DNA and RNA sequences and 21 for amino acid sequences). Sequences with a total number of characters shorter than 2 * k + m will be accepted but not all possible patterns of the feature space can be taken into account. When multiple kernels with different k and/or m values should be generated, e.g. for model selection an integer vector can be specified instead of a single numeric values. In this case a list of kernel objects with the individual values from the integer vector of parameter k is generated as result. The processing effort for this kernel is highly dependent on the value of k because of the additional factor 2 in the exponent for the feature space size) and only small values of k will allow efficient processing. Default=1
`m`	maximal number of irrelevant positions between a pair of kmers. The value of m must be an integer value larger than 0. For example a value of m=2 means that zero, one or two irrelevant positions between kmer pairs are considered as valid features. (A value of 0 corresponds to the spectrum kernel with a kmer length of 2*k and is not allowed for the gappy pair kernel). When an integer vector is specified a list of kernels is generated as described above for parameter k. If multiple values are specified both for parameter k and parameter m one kernel object is created for each of the combinations of k and m. Default=1

r	exponent which must be > 0 (see details section in spectrumKernel). Default=1
annSpec	boolean that indicates whether sequence annotation should be taken into account (details see on help page for annotationMetadata). Annotation information is only evaluated for the kmer positions of the kmer pair but not for the irrelevant positions in between. For the annotation specific gappy pair kernel the total number of features increases to (|A|^(2*k))*(|a|^(2*k)*(m+1) with |A| as the size of the sequence alphabet and |a| as the size of the annotation alphabet. Default=FALSE
distWeight	a numeric distance weight vector or a distance weighting function (details see on help page for gaussWeight). Default=NULL
normalized	generated data from this kernel will be normalized (details see below). Default=TRUE
exact	use exact character set for the evaluation (details see below). Default=TRUE
ignoreLower	ignore lower case characters in the sequence. If the parameter is not set lower case characters are treated like uppercase. Default=TRUE
presence	if this parameter is set only the presence of a kmers will be considered, otherwise the number of occurances of the kmer is used. Default=FALSE
revComplement	if this parameter is set a kmer pair and its reverse complement are treated as the same feature. Default=FALSE
mixCoef	mixing coefficients for the mixture variant of the gappy pair kernel. A numeric vector of length k is expected for this parameter with the unused components in the mixture set to 0. Default=numeric(0)
kernel	a sequence kernel object
x	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)

Details

Creation of kernel object

The function 'gappyPairKernel' creates a kernel object for the gappy pair kernel. This kernel object can then be used with a set of DNA-, RNA- or AA-sequences to generate a kernel matrix or an explicit representation for this kernel. The gappy pair kernel uses pairs of neighboring subsequences of length k (kmers) with up to m irrelevant positions between the kmers. For sequences shorter than 2*k the self similarity (i.e. the value on the main diagonal in the square kernel matrix) is 0. The explicit representation contains only zeros for such a sample. Dependent on the learning task it might make sense to remove such sequences from the data set as they do not contribute to the model but still influence performance values.

For values different from 1 (=default value) parameter r leads to a transfomation of similarities by taking each element of the similarity matrix to the power of r. If normalized=TRUE, the feature vectors are scaled to the unit sphere before computing the similarity value for the kernel matrix. For two samples with the feature vectors x and y the similarity is computed as:

$$s = \frac{\vec{x}^T \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

For an explicit representation generated with the feature map of a normalized kernel the rows are normalized by dividing them through their Euclidean norm. For parameter `exact=TRUE` the sequence characters are interpreted according to an exact character set. If the flag is not set ambigous characters from the IUPAC characterset are also evaluated.

The annotation specific variant (for details see annotationMetadata) and the position dependent variants (for details see positionMetadata) either in the form of a position specific or a distance weighted kernel are supported for the gappy pair kernel. The generation of an explicit representation is not possible for the position dependent variants of this kernel.

Creation of kernel matrix

The kernel matrix is created with the function `getKernelMatrix` or via a direct call with the kernel object as shown in the examples below.

Value

gappyPairKernel: upon successful completion, the function returns a kernel object of class `GappyPairKernel`.

of getDimFeatureSpace: dimension of the feature space as numeric value

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Mahrenholz, 2011) – C. Mahrenholz, I. Abfalter, U. Bodenhofer, R. Volkmer and S. Hochreiter. Complex networks govern coiled-coil oligomerizations - predicting and profiling by means of a machine learning approach.
(Bodenhofer, 2009) – U. Bodenhofer, K. Schwarzbauer, M. Ionescu and S. Hochreiter. Modelling position specificity in sequence kernels by fuzzy equivalence relations.

(Kuksa, 2008) – P. Kuksa, P. Huang and V. Pavlovic. Fast Protein Homology and Fold Detection with Sparse Spatial Sample Kernels

http://www.bioinf.jku.at/software/kebabs

See Also

`getKernelMatrix`, `getExRep`, `kernelParameters-method`, `spectrumKernel`, `mismatchKernel`, `motifKernel`, `GappyPairKernel`

Examples

```
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the gappy pair kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                          "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
```

```
                          "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create the kernel object for dimer pairs with up to ten irrelevant
## position between the kmers of the pair without normalization
gappy <- gappyPairKernel(k=2, m=10, normalized=FALSE)
## show details of kernel object
gappy

## generate the kernel matrix with the kernel object
km <- gappy(dnaseqs)
dim(km)
km[1:5,1:5]

## alternative way to generate the kernel matrix
km <- getKernelMatrix(gappy, dnaseqs)
km[1:5,1:5]

## Not run:
## plot heatmap of the kernel matrix
heatmap(km, symm=TRUE)

## End(Not run)
```

`GappyPairKernel-class` *Gappy Pair Kernel Class*

Description

Gappy Pair Kernel Class

Details

Instances of this class represent a kernel object for the gappy pair kernel. The kernel considers adjacent pairs of kmers with up to m irrelevant characters between the pair. The class is derived from SequenceKernel.

Slots

`k` length of the substrings considered by the kernel

`m` maximum number of irrelevant character between two kmers

`r` exponent (for details see gappyPairKernel)

`annSpec` when set the kernel evaluates annotation information

`distWeight` distance weighting function or vector

`normalized` data generated with this kernel object is normalized

`exact` use exact character set for evaluation

`ignoreLower` ignore lower case characters in the sequence

`presence` consider only the presence of kmers not their counts

`revComplement` consider a kmer and its reverse complement as the same feature

`mixCoef` mixing coefficients for mixture kernel

`genRandBioSeqs` *Generate Random Biological Sequences*

Description

Generate biological sequences with uniform random distribution of alphabet characters.

Usage

```
genRandBioSeqs(seqType = c("DNA", "RNA", "AA"), numSequences, seqLength,
  biostring = TRUE, seed)
```

Arguments

`seqType`	defines the type of sequence as DNA, RNA or AA and the underlying alphabet. Default="DNA"
`numSequences`	single numeric value which specifies the number of sequences that should be generated.
`seqLength`	either a single numeric value or a numeric vector of length 'numSequences' which gives the length of the sequences to be generated.
`biostring`	if `TRUE` the sequences will be generated in XStringSet format otherwise as BioVector derived class. Default=TRUE
`seed`	when present the random generator will be seeded with the value passed in this parameter

Details

The function generates a set of sequences with uniform distribution of alphabet characters and returns it as XStringSet or BioVector dependent on the parameter `biostring`.

Value

When the parameter 'biostring' is set to FALSE the function returns a XStringSet derived class otherwise a BioVector derived class.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

Examples

```
## generate a set of AA sequences of fixed length as AAStringSet
aaseqs <- genRandBioSeqs("AA", 100, 1000, biostring=TRUE)

## show AA sequence set
aaseqs

## Not run:
## generate a set of "DNA" sequences as DNAStringSet with uniformly
## distributed lengths between 1500 and 3000 bases
seqLength <- runif(300, min=1500, max=3500)
dnaseqs <- genRandBioSeqs("DNA", 100, seqLength, biostring=TRUE)

## show DNA sequence set
dnaseqs

## End(Not run)
```

---

getExRep *Explict Representation*

Description

Create an explicit representation

Usage

```
getExRep(x, kernel = spectrumKernel(), sparse = TRUE,
  zeroFeatures = FALSE, features = NULL, useRowNames = TRUE,
  useColNames = TRUE, selx = NULL)

getExRepQuadratic(exRepLin, useRowNames = TRUE, useColNames = TRUE,
  zeroFeatures = FALSE)
```

Arguments

x	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)
kernel	a sequence kernel object. The feature map of this kernel object is used to generate the explicit representation.
sparse	boolean that indicates whether a sparse or dense explicit representation should be generated. Default=TRUE
zeroFeatures	indicates whether columns with zero feature counts across all samples should be included in the explicit representation. (see below) Default=FALSE

features	feature subset of the specified kernel in the form of a character vector. When a feature subset is passed to the function all other features in the feature space are not considered for the explicit representation. (see below)
useRowNames	if this parameter is set the sample names will be set as row names if available in the provided sequence set. Default=TRUE
useColNames	if this parameter is set the features will be set as column names in the explicit representation. Default=TRUE
selx	subset of indices into x. When this parameter is present the explicit representation is generated for the specified subset of samples only. default=NULL
exRepLin	a linear explicit representation

Details

Creation of an explicit representation

The function 'getExRep' creates an explicit representation of the given sequence set using the feature map of the specified kernel. It contains the feature counts in a matrix format. The rows of the matrix represent the samples, the columns the features. For a dense explicit representation of class ExplicitRepresentationDense the count data is stored in a dense matrix. To allow efficient storage all features that do not occur in the sequence set are removed from the explicit representation by default. When the parameter zeroFeatures is set to TRUE these features are also included resulting an explicit representation which contains the full feature space. For feature spaces larger than one million features the inclusion of zero features is not possible.

In case of large feature spaces a sparse explicit representation of class ExplicitRepresentationSparse is much more efficient by storing the count data as dgRMatrix from package **Matrix**). The class ExplicitRepresentationSparse is derived from dgRMatrix. As zero features are not stored in a sparse matrix the flag zeroFeatures only controls whether the column names of features not occuring in the sequences are included or not.

Both the dense and the sparse explicit representation also contain the kernel object which was used for it's creation. For an explicit representation without zero features column names are mandatory. An explicit representation can be created for position independent and annotation specific kernel variants (for details see annotationMetadata). In annotation specific kernels the annotation characters are included as postfix in the features. For kernels with normalization the explicit representation is normalized resulting in row vectors normalized to the unit sphere. For feature subsets used with normalized kernels all features of the feature space are used in the normalization.

Usage of explicit representations

Learning with linear SVMs (e.g. ksvmin package **kernlab** or svm in package **e1071**) can be performed either through passing a kernel matrix of similarity values or an explicit representation and a linear kernel to the SVM. The SVMs in package **kernlab** support dense explicit representation or kernel matrix as data representations. The SVMs in packages **e1071**) and **LiblineaR** only support dense or sparse explicit representation. In many cases there can be considerable performance differences between the two variants of passing data to the SVM. And especially for larger feature spaces the sparse explicit representation not only brings higher memory efficiency but also leads to drastically improved runtimes during training and prediction.

In general all of the complexity of converting the sequences with a specific kernel to an explicit representation or a kernel matrix and adapting the formats and parameters to the specific SVM is hidden within the KeBABS training and predict methods (see kbsvm, predict) and the user can concentrate on the actual data analysis task. During training via kbsvm the parameter explicit controls the training via kernel matrix or explicit representation and the parameter explicitType determines whether a dense or sparse explicit representation is used. Manual generation of explicit representations is only necessary for usage with other learners or analysis methods not supported by KeBABS.

Quadratic explicit representation

The package **LiblineaR** only provides linear SVMs which are tuned for efficient processing of larger feature spaces and sample numbers. To allow the use of a quadratic kernel on these SVMs a quadratic explicit representation can be generated from the linear explicit representation. It contains counts for feature pairs and the features combined to one pair are separated by '_' in the column names of the quadratic explicit representation. Please be aware that the dimensionality for a quadratic explicit representation increases considerably compared to the linear one. In the other SVMs a linear explicit representation together with a quadratic kernel is used instead. In training via kbsvm the use of a linear representation with a quadratic kernel or a quadratic explicit representation instead is indicated through setting the parameter featureType to the value "quadratic".

Value

getExRep: upon successful completion, dependent on the flag sparse the function returns either a dense explicit representation of class ExplicitRepresentationDense or a sparse explicit representation of class ExplicitRepresentationSparse.

getExRepQuadratic: upon successful completion, the function returns a quadratic explicit representation

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

ExplicitRepresentationDense, ExplicitRepresentationSparse, getKernelMatrix, kernelParameters-method, SpectrumKernel, mismatchKernel, gappyPairKernel, motifKernel

Examples

```
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the spectrum kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGACACCACACTCAGCTAGGGGGACTGGGAGC",
                          "ATAAAGGGAGCAGACATCATGACCTTTTTGACCCTAATTATTTCAGC",
```

```
                          "CAGGAATCAGCACAGGCAGGGGCACTGCATCCCAAGACATCTGGGCC",
                          "GGACATATACCCACCCTTACCTGCCATACAGGATAGGGCCACTGCCC",
                          "ATAAAGGATGCAGACATCATGGCCTTTTTGACCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create the kernel object for dimers with normalization
speck <- spectrumKernel(k=2)
## show details of kernel object
speck

## generate the dense explicit representation for the kernel
erd <- getExRep(dnaseqs, speck, sparse=FALSE)
dim(erd)
erd[1:5,]

## generate the dense explicit representation with zero features
erd <- getExRep(dnaseqs, speck, sparse=FALSE, zeroFeatures=TRUE)
dim(erd)
erd[1:5,]

## generate the sparse explicit representation for the kernel
ers <- getExRep(dnaseqs, speck)
dim(ers)
ers[1:5,]

## generate the sparse explicit representation with zero features
ers <- getExRep(dnaseqs, speck, zeroFeatures=TRUE)
dim(ers)
ers[1:5,]

## generate the quadratic explicit representation
erdq <- getExRepQuadratic(erd)
dim(erdq)
erdq[1:5,1:15]

## Not run:
## run taining and prediction with dense linear explicit representation
data(TFBS)
enhancerFB
train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
test <- c(1:length(enhancerFB))[-train]
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=speck,
               pkg="LiblineaR", svm="C-svc", cost=10, explicit="yes",
               explicitType="dense")
pred <- predict(model, x=enhancerFB[test])
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))

## run taining and prediction with sparse linear explicit representation
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=speck,
               pkg="LiblineaR", svm="C-svc", cost=10, explicit="yes",
               explicitType="sparse")
pred <- predict(model, x=enhancerFB[test])
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))
```

```
## End(Not run)
```

`getFeatureWeights` *Feature Weights*

Description

Compute Feature Weights for KeBABS Model

Usage

```
getFeatureWeights(model, exrep = NULL, features = NULL,
  weightLimit = .Machine$double.eps)
```

Arguments

`model`	model object of class `KBModel` created by `kbsvm`.
`exrep`	optional explicit representation of the support vectors from which the feature weights should be computed. If no explicit representation is passed to the function the explicit representation is generated internally from the support vectors stored in the model. default=`NULL`
`features`	feature subset of the specified kernel in the form of a character vector. When a feature subset is passed to the function all other features in the feature space are not considered for the explicit representation. (see below) default=`NULL`
`weightLimit`	the feature weight limit is a single numeric value and allows pruning of feature weights. All feature weights with an absolute value below this limit are set to 0 and are not considered in the feature weights. Default=.Machine$double.eps

Details

Overview

Feature weights represent the contribution to the decision value for a single occurance of the feature in the sequence. In this way they give a hint concerning the importance of the individual features for a given classification or regression task. Please consider that for a pattern length larger than 1 patterns at neighboring sequence positions overlap and are no longer independent from each other. Apart from the obvious overlapping possibility of patterns for e.g. gappy pair kernel, motif kernel or mixture kernels multiple patterns can be relevant for a single position. Therefore feature weights do not describe the relevance for individual features exactly.

Computation of feature weights

Feature weights can be computed automatically as part of the training (see parameter `featureWeights` in method `kbsvm`. In this case the function getFeatureWeights is called during training automatically. When this parameter is not set during training computation of feature weights after training is

possible with the function getFeatureWeights. The function also supports pruning of feature weights (see parameter `weightLimit` allowing to test different prunings without retraining.

Usage of feature weights

Feature weights are used during prediction to speed up the prediction process. Prediction via feature weights is performed in KeBABS when feature weights are available in the model (see `featureWeights`). When feature weights are not available or for multiclass prediction KeBABS defaults to the native prediction in the SVM used during training.

Feature weights are also used during generation of prediction profiles (see `getPredictionProfile`). In the feature weights the general relevance of features is reflected. When generating prediction profiles for a given set of sequences from the feature weights the relevance of single sequence positions is shown for the individual sequences according to the given learning task.

Feature weights for position dependent kernels

For position dependent kernels the generation of feature weights is not possible during training. In this case the featureWeights slot in the model contains a data representation that allows simple computation of feature weights during prediction or during generation of prediction profiles.

Value

Upon successful completion, the function returns the feature weights as numeric vector. For quadratic kernels a matrix of feature weights is returned giving the feature weights for pairs of features. In case of multiclass the function returns the feature weights for the pairwise SVMs as list of numeric vectors (or matrices for quadratic kernels).

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

`kbsvm`, `predict`, `getPredictionProfile` `featureWeights`, `KBModel`

Examples

```
## standard method to create feature weights automatically during training
## model <- kbsvm( .... , featureWeights="yes", .....)
## this example describes the case where feature weights were not created
## during training but should be added later to the model

## load example sequences and select a small set of sequences
## to speed up training for demonstration purpose
data(TFBS)
```

```
## create sample indices of training and test subset
train <- sample(1:length(yFB), 200)
test <- c(1:length(yFB))[-train]
## determin all labels
allLables <- unique(yFB)

## create a kernel object
gappyK1M4 <- gappyPairKernel(k=1, m=4)

## model is trainded with creation of feature weights
model <- kbsvm(enhancerFB[train], yFB[train], gappyK1M4,
               pkg="LiblineaR", svm="C-svc", cost=20)

## feature weights included in model
featureWeights(model)

## Not run:
## model is originally trainded without creation of feature weights
model <- kbsvm(enhancerFB[train], yFB[train], gappyK1M4,
               pkg="LiblineaR", svm="C-svc", cost=20, featureWeights="no")

## no feature weights included in model
featureWeights(model)

## later after training add feature weights and model offset of model to
## KeBABS model
featureWeights(model) <- getFeatureWeights(model)
modelOffset(model) <- getSVMSlotValue("b", model)

## show a part of the feature weights and the model offset
featureWeights(model)[1:7]
modelOffset(model)

## another scenario for getFeatureWeights is to test the performance
## behavior of different prunings of the feature weights

## show histogram of full feature weights
hist(featureWeights(model), breaks=30)

## show number of features
length(featureWeights(model))

## first predict with full feature weights to see how performance
## when feature weights are included in the model prediction is always
## performed with the feature weights
## changes through pruning
pred <- predict(model, enhancerFB[test])
evaluatePrediction(pred, yFB[test], allLabels=allLables)

## add feature weights with pruning to absolute values larger than 0.6
## model offset was assigned above and is not impacted by pruning
featureWeights(model) <- getFeatureWeights(model, weightLimit=0.6)
```

```
## show histogram of full feature weights
hist(featureWeights(model), breaks=30)

## show reduced number of features
length(featureWeights(model))

## now predict with pruned feature weights
pred <- predict(model, enhancerFB, sel=test)
evaluatePrediction(pred, yFB[test], allLabels=allLables)

## End(Not run)
```

`getPredictionProfile,BioVector-method`
Calculation Of Predicition Profiles

Description

compute prediction profiles for a given set of biological sequences from a model trained with /codekbsvm

Usage

```
## S4 method for signature BioVector
getPredictionProfile(object, kernel, featureWeights, b,
  svmIndex = 1, sel = NULL, weightLimit = .Machine$double.eps)

## S4 method for signature XStringSet
getPredictionProfile(object, kernel, featureWeights, b,
  svmIndex = 1, sel = NULL, weightLimit = .Machine$double.eps)

## S4 method for signature XString
getPredictionProfile(object, kernel, featureWeights, b,
  svmIndex = 1, sel = NULL, weightLimit = .Machine$double.eps)
```

Arguments

`object`	a single biological sequence in the form of an `DNAString`, `RNAString` or `AAString` or multiple biological sequences as `DNAStringSet`, `RNAStringSet`, `AAStringSet` (or as `BioVector`).
`kernel`	a sequence kernel object of class `SequenceKernel`.
`featureWeights`	a feature weights matrix retrieved from a KeBABS model with the accessor `featureWeights`.
`b`	model intercept from a KeBABS model.
`svmIndex`	integer value selecting one of the pairwise SVMs in case of pairwise multiclass classification. Default=1

sel	subset of indices into x as integer vector. When this parameter is present the prediction profiles are computed for the specified subset of samples only. Default=integer(0)
weightLimit	the feature weight limit is a single numeric value and allows pruning of feature weights. All feature weights with an absolute value below this limit are set to 0 and are not considered for the prediction profile computation. This parameter is only relevant when featureWeights are calculated in KeBABS during training. Default=.Machine$double.eps

Details

With this method prediction profiles can be generated explicitely for a given set of sequences with a given model represented through its feature weights and the model intercept b. A single prediction profile shows for each position of the sequence the contribution of the patterns at this position to the decision value. The prediciion profile also includes the kernel object used for the generation of the profile and the seqence data.

A single profile or a pair can be plotted with method plot showing the relevance of sequence positions for the prediction. Please consider that patterns occuring at neighboring sequence positions are not statistically independent which means that the relevance of a specific position is not only determined by the patterns at this position but is also influenced by the neighborhood around this position. Prediction profiles can also be generated implicitely during predction for the predicted samples (see parameter predProfiles in predict).

Value

getPredictionProfile: upon successful completion, the function returns a set of prediction profiles for the sequences as class PredictionProfile.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Mahrenholz, 2011) – Mahrenholz, C.C., Abfalter, I.G., Bodenhofer, U., Volkmer, R., and Hochreiter, S.; Complex networks govern coiled coil oligomerization - predicting and profiling by means of a machine learning approach.

(Bodenhofer, 2009) – Bodenhofer, U., Schwarzbauer, K., Ionescu, S. and Hochreiter,S., Modeling Position Specificity in Sequence Kernels by Fuzzy Equivalence Relations.

http://www.bioinf.jku.at/software/kebabs

See Also

PredictionProfile, predict, plot, featureWeights{KBModel}

Examples

```
## set random generator seed to make the results of this example
## reproducable
set.seed(123)

## load coiled coil data
data(CCoil)
gappya <- gappyPairKernel(k=1,m=11, annSpec=TRUE)
model <- kbsvm(x=ccseq, y=as.numeric(yCC), kernel=gappya,
               pkg="e1071", svm="C-svc", cost=15)

## show feature weights
featureWeights(model)[,1:5]

## define two new sequences to be predicted
GCN4 <- AAStringSet(c("MKQLEDKVEELLSKNYHLENEVARLKKLV",
                      "MKQLEDKVEELLSKYYHTENEVARLKKLV"))
names(GCN4) <- c("GCN4wt", "GCN_N16Y,L19T")
## assign annotation metadata
annCharset <- annotationCharset(ccseq)
annot <- c("abcdefgabcdefgabcdefgabcdefga",
           "abcdefgabcdefgabcdefgabcdefga")
annotationMetadata(GCN4, annCharset=annCharset) <- annot

## compute prediction profiles
predProf <- getPredictionProfile(GCN4, gappya,
           featureWeights(model), modelOffset(model))

## show prediction profiles
predProf

## plot prediction profile of first aa sequence
plot(predProf, sel=1, ylim=c(-0.4, 0.2), heptads=TRUE, annotate=TRUE)

## plot prediction profile of both aa sequences
plot(predProf, sel=c(1,2), ylim=c(-0.4, 0.2), heptads=TRUE, annotate=TRUE)

## prediction profiles can also be generated during prediction
## when setting the parameter predProf to TRUE
## plotting longer sequences to pdf is shown in the examples for the
## plot function
```

KBModel-class *KeBABS Model Class*

Description

KeBABS Model Class

Details

Instances of this class represent a model object for the KeBABS meta-SVM.

Slots

`call` invocation string of KeBABS meta-SVM

`numSequences` number of sequences used for training

`sel` index subset of samples used for training

`y` vector of target values

`levels` levels of target

`numClasses` number of classes

`classNames` class labels

`classWeights` class weights

`SV` support vectors

`svIndex` support vector indices

`alphaIndex` list of SVM indices per SVM

`trainingFeatures` feature names used in training

`featureWeights` feature Weights

`b` model offset

`probA` fitted logistic function parameter A

`probB` fitted logistic function parameter A

`sigma` scale of Laplacian fitted to regression residuals

`cvResult` cross validation result of class `CrossValidationResult`

`modelSelResult` model selection / grid search result of class `ModelSelectionResult`

`ctlInfo` KeBABS control info of class `ControlInformation`

`svmInfo` info about requested / used SVM of class `SVMInformation`

`svmModel` original model returned from SVM

`KBModelAccessors` *KBModel Accessors*

Description

KBModel Accessors

Usage

```
## S4 method for signature KBModel
modelOffset(object)

getSVMSlotValue(paramName, model, raw = FALSE)
```

Arguments

object	a KeBABS model
paramName	unified name of an SVM model data element
model	a KeBABS model
raw	when set to TRUE the parameter value is delivered in exactly the way as it is stored in the SVM specific model, when set to FALSE it is delivered in unified format

Value

getSVMSlotValue: value of requested parameter in unified or native format dependent on parameter raw.

Accessor-like methods

modelOffset: returns the model offset.

featureWeights: returns the feature weights.

SVindex: returns the support vector indices for the training samples.

cvResult: returns result of cross validation as object of class CrossValidationResult.

modelSelResult: returns result of model selection as object of class ModelSelectionResult.

svmModel: returns the native svm model stored within KeBABS model.

probabilityModel: returns the probability model stored within KeBABS model.

Examples

```
## create kernel object for normalized spectrum kernel
specK5 <- spectrumKernel(k=5)
## Not run:
## load data
data(TFBS)

## perform training - feature weights are computed by default
model <- kbsvm(enhancerFB, yFB, specK5, pkg="LiblineaR",
               svm="C-svc", cost=15, cross=10, showProgress=TRUE)
               showProgress=TRUE)

## show result of validation
cvResult(model)
## show feature weights
featureWeights(model)[1:5]
## show model offset
modelOffset(model)

## End(Not run)
```

kbsvm,BioVector-method

KeBABS Training Methods

Description

Train an SVM-model with a sequence kernel on biological sequences

Usage

```
## S4 method for signature BioVector
kbsvm(x, y, kernel = NULL, pkg = "auto",
  svm = "C-svc", explicit = "auto", explicitType = "auto",
  featureType = "linear", featureWeights = "auto",
  weightLimit = .Machine$double.eps, classWeights = numeric(0), cross = 0,
  noCross = 1, groupBy = NULL, nestedCross = 0, noNestedCross = 1,
  perfParameters = character(0), perfObjective = "ACC", probModel = FALSE,
  sel = integer(0), features = NULL, showProgress = FALSE,
  showCVTimes = FALSE, runtimeWarning = TRUE,
  verbose = getOption("verbose"), ...)

## S4 method for signature XStringSet
kbsvm(x, y, kernel = NULL, pkg = "auto",
  svm = "C-svc", explicit = "auto", explicitType = "auto",
  featureType = "linear", featureWeights = "auto",
  weightLimit = .Machine$double.eps, classWeights = numeric(0), cross = 0,
  noCross = 1, groupBy = NULL, nestedCross = 0, noNestedCross = 1,
  perfParameters = character(0), perfObjective = "ACC", probModel = FALSE,
  sel = integer(0), features = NULL, showProgress = FALSE,
  showCVTimes = FALSE, runtimeWarning = TRUE,
  verbose = getOption("verbose"), ...)

## S4 method for signature ExplicitRepresentation
kbsvm(x, y, kernel = NULL, pkg = "auto",
  svm = "C-svc", explicit = "auto", explicitType = "auto",
  featureType = "linear", featureWeights = "auto",
  weightLimit = .Machine$double.eps, classWeights = numeric(0), cross = 0,
  noCross = 1, groupBy = NULL, nestedCross = 0, noNestedCross = 1,
  perfParameters = character(0), perfObjective = "ACC", probModel = FALSE,
  sel = integer(0), showProgress = FALSE, showCVTimes = FALSE,
  runtimeWarning = TRUE, verbose = getOption("verbose"), ...)

## S4 method for signature KernelMatrix
kbsvm(x, y, kernel = NULL, pkg = "auto",
  svm = "C-svc", explicit = "no", explicitType = "auto",
  featureType = "linear", featureWeights = "no",
  classWeights = numeric(0), cross = 0, noCross = 1, groupBy = NULL,
```

```
nestedCross = 0, noNestedCross = 1, perfParameters = character(0),
perfObjective = "ACC", probModel = FALSE, sel = integer(0),
showProgress = FALSE, showCVTimes = FALSE, runtimeWarning = TRUE,
verbose = getOption("verbose"), ...)
```

Arguments

x	multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector). Also a precomputed kernel matrix (see getKernelMatrix or a precomputed explicit representation (see getExRep can be used instead. If they were precomputed with a sequence kernel this kernel should be specified in the parameter kernel in this case.
y	response vector which contains one value for each sample in 'x'. For classification tasks this can be either a character vector, a factor or a numeric vector, for regression tasks it must be a numeric vector. For numeric labels in binary classification the positive class must have the larger value, for factor or character based labels the positive label must be at the first position when sorting the labels in descendent order according to the C locale. If the parameter sel is used to perform training with a sample subset the response vector must have the same length as 'sel'.
kernel	a sequence kernel object or a string kernel from package kernlab. In case of grid search or model selection a list of sequence kernel objects can be passed to training.
pkg	name of package which contains the SVM implementation to be used for training, e.g. kernlab, e1071 or LiblineaR. For gridSearch or model selection multiple packages can be passed as character vector. (see also parameter svm below). Default="auto"
svm	name of the SVM used for the classification or regression task, e.g. "C-svc". For gridSearch or model selection multiple SVMs can be passed as character vector. For each entry in this character vector a corresponding entry in the character vector for parameter pkg is required, if multiple SVMs are used in one cross validation or model selection run.
explicit	this parameter controls whether training should be performed with the kernel matrix (see getKernelMatrix) or explicit representation (see getExRep). When the parameter is set to "no" the kernel matrix is used, for "yes" the model is trained from the explicit representation. When set to "auto" KeBABS automatically selects a variant based on runtime heuristics. Default="auto"
explicitType	this parameter is only relevant when parameter 'explicit' is different from "no". The values "sparse" and "dense" indicate whether a sparse or dense explicit representation should be used. When the parameter is set to "auto" KeBABS selects a variant. Default="auto"
featureType	when the parameter is set to "linear" single features areused in the analysis (with a linear kernel matrix or a linear kernel applied to the linear explicit representation). When set to "quadratic" the analysis is based on feature pairs. For an SVM from LiblineaR (which does not support kernels) KeBABS generates a quadratic explicit representation. For the other SVMs a polynomial kernel of degree 2 is used for learning via explicit representation. In the case of learning via kernel

matrix a quadratic kernel matrix (quadratic here in the sense of linear kernel matrix with each element taken to power 2) is generated. Default="linear"

`featureWeights` with the values "no" and "yes" the user can control whether feature weights are calulated as part of the training. When the parameter is set to "auto" KeBABS selects a variant (see below). Default="auto"

`weightLimit` the feature weight limit is a single numeric value and allows pruning of feature weights. All feature weights with an absolute value below this limit are set to 0 and are not considered in the model and for further predictions. This parameter is only relevant when featureWeights are calculated in KeBABS during training. Default=.Machine$double.eps

`classWeights` a numeric named vector of weights for the different classes, used for asymmetric class sizes. Each element of the vector must have one of the class names but not all class names must be present. Default=1

`cross` an integer value K > 0 indicates that k-fold cross validation should be performed. A value -1 is used for Leave-One-Out (LOO) cross validation. (see above) Default=0

`noCross` an integer value larger than 0 is used to specify the number of repetitions for cross validation. This parameter is only relevant if 'cross' is different from 0. Default=1

`groupBy` allows a grouping of samples during cross validation. The parameter is only relevant when 'cross' is larger than 1. It is an integer vector or factor with the same length as the number of samples used for training and specifies for each sample to which group it belongs. Samples from the same group are never spread over more than one fold. (see `crossValidation`). Grouped cross validation can also be used in grid search for each grid point. Default=NULL

`nestedCross` in integer value K > 0 indicates that a model selection with nested cross validation should be performed with a k-fold outer cross validation. The inner cross validation is defined with the 'cross' parameter (see below), Default=0

`noNestedCross` an integer value larger than 0 is used to specify the number of repetitions for the nested cross validation. This parameter is only relevant if 'nestedCross' is larger than 0. Default=1

`perfParameters` a character vector with one or several values from the set "ACC" , "BACC", "MCC" and "ALL". "ACC" stands for accuracy, "BACC" for balanced accuracy, "MCC" for Matthews Correlation Coefficient and "ALL" for all three. This parameter defines which performance parameters are collected in cross validation, grid search and model selection for display purpose. Default=NULL

`perfObjective` a singe character string from the set "ACC", "BACC" and "MCC" (see previous parameter). The parameter is only relevant in grid search and model selection and defines which performance measure is used to determine the best performing parameter set. Default="ACC"

`probModel` when setting this boolean parameter to TRUE a probability model is determined as part of the training (see below). Default=FALSE

`sel` subset of indices into `x`. When this parameter is present the training is performed for the specified subset of samples only. Default=`integer(0)`

features	feature subset of the specified kernel in the form of a character vector. When a feature subset is passed to the function all other features in the feature space are not considered for training (see below). A feature subset can only be used when a single kernel object is specified in the 'kernel' parameter. Default=NULL
showProgress	when setting this boolean parameter to TRUE the progress of a cross validation is displayed. The parameter is only relevant for cross validation. Default=FALSE
showCVTimes	when setting this boolean parameter to TRUE the runtimes of the cross validation runs are shown after the cross validation is finished. The parameter is only relevant for cross validation. Default=FALSE
runtimeWarning	when setting this boolean parameter to FALSE a warning for long runtimes will not be shown in case of large feature space dimension or large number of samples. Default=TRUE
verbose	boolean value that indicates whether KeBABS should print additional messages showing the internal processing logic in a verbose manner. The default value depends on the R session verbosity option. Default=getOption("verbose")
...	additional parameters which are passed to SVM training transparently.

Details

Overview

The kernel-related functionality provided in this package is specifically centered around biological sequences, i.e. DNA-, RNA- or AA-sequences (see also DNAStringSet, RNAStringSet and AAStringSet) and Support Vector Machine (SVM) based methods. Apart from the implementation of the most relevant kernels for sequence analysis (see spectrumKernel, mismatchKernel, gappyPairKernel and motifKernel) KeBABS also provides a framework which allows easy interworking with existing SVM implementations in other R packages. In the current implementation the SVMs provided in the packages kernlab, e1071 and LiblineaR are in focus.

This framework can be considered like a "meta-SVM", which provides a simple and unified user interface to these SVMs for classification (binary and multiclass) and regression tasks. The user calls the "meta-SVM" in a classical SVM-like manner by passing sequence data, a sequence kernel with kernel parameters and the SVM which should be used for the learning task togehter with SVM parameters. KeBABS internally generates the relevant representations (see getKernelMatrix or getExRep) from the sequence data using the specified kernel, adapts parameters and formats to the selected SVM and internally calls the actual SVM implementation in the requested package. KeBABS unifies the result returned from the invoked SVM and returns a unified data structure, the KeBABS model, which also contains the SVM-specific model (see svmModel.

The KeBABS model is used in prediction (see predict) to predict the response for new sequence data. On user request the feature weights are computed and stored in the Kebabs model during training (see below). The feature weights are used for the generation of prediction profiles (see getPredictionProfile) which show the importance of sequence positions for a specfic learning task.

Training of biological sequences with a sequence kernel

Training is performed via the method kbsvm for classification and regression tasks. The user passes sequence data, the response vector, a sequence kernel object and the requested SVM along with SVM parameters to kbsvm and receives the training results in the form of a KeBABS model object of class KBModel. The accessor svmModel allows to retrieve the SVM specific model from the KeBABS model object. However, for regular operation a detailed look into the SVM specific model is usually not necessary.

The standard data format for sequences in KeBABS are the XStringSet-derived classes DNAStringSet, RNAStringSet and AAStringSet. (When repeat regions are coded as lowercase characters and should be excluded from the analysis the sequence data can be passed as BioVector which also supports lowercase characters instead of XStringSet format. Please note that the classes derived from XStringSet are much more powerful than the BioVector derived classes and should be used in all cases where lowercase characters are not needed).

Instead of sequences also a precomputed explicit representation or a precomputed kernel matrix can be used for training. Examples for training with kernel matrix and explicit representation can be found on the help page for the prediction method predict.

Apart from SVM training kbsvm can be also used for cross validation (see crossValidation and parameters cross and noCross), grid search for SVM- and kernel-parameter values (see gridSearch) and model selection (see modelSelection and parameters nestedCross and noNestedCross).

Package and SVM selection

The user specifies the SVM implementation to be used for a learning task by selecting the package with the pkg parameter and the SVM method in the package with the SVM parameter. Currently the packages codekernlab, e1071 and LiblineaR are supported. The names for SVM methods vary from package to package and KeBABS provide following unified names which can be selected across packages. The following table shows the available SVM methods:

SVM name	description
C-svc:	C classification (with L2 regularization and L1 loss)
l2rl2l-svc:	classif. with L2 regularization and L2 loss (dual)
l2rl2lp-svc:	classif. with L2 regularization and L2 loss (primal)
l1rl2l-svc:	classification with L1 regularization and L2 loss
nu-svc:	nu classification
C-bsvc:	bound-constraint SVM classification
mc-natC:	Crammer, Singer native multiclass
mc-natW:	Weston, Watkins native multiclass
one-svc:	one class classification
eps-svr:	epsilon regression
nu-svr:	nu regression
eps-bsvr:	bound-constraint svm regression

Pairwise multiclass can be selected for `C-svc` and `nu-svc` if the label vector contains more than two classes. For `LiblineaR` the multiclass implementation is always based on "one against the rest" for all SVMs except for `mc-natC` which implements native multiclass according to Crammer and Singer. The following table shows which SVM method is available in which package:

SVM name	kernlab	e1071	LiblineaR
C-svc:	x	x	x
l2rl2l-svc:	-	-	x
l2rl2lp-svc:	-	-	x
l1rl2l-svc:	-	-	x
nu-svc:	x	x	-
C-bsvc:	x	-	-
mc-natC:	x	-	x
mc-natW:	x	-	-
one-svc:	x	x	-
eps-svr:	x	x	-
nu-svr:	x	x	-
eps-bsvr:	x	-	-

SVM parameters

To avoid unnecessary changes of parameters names when switching between SVM implementation in different packages unified names for identical parameters are available. They are translated by KeBABS to the SVM specific name. The obvious example is the cost parameter for the C-svm. It is named `C` in `kernlab` and `cost` in `e1071` and `LiblineaR`. The unified name in KeBABS is cost. If the parameter is passed to `kbsvm` in a package specific version it is translated back to the KeBABS name internally. This applies to following parameters - here shown with their unified names:

parameter name	description
cost:	cost parameter of C-SVM
nu:	nu parameter of nu-SVM
eps:	epsilon parameter of eps-SVR and nu-SVR
classWeights:	class weights for asymmetrical class size
tolerance:	tolerance as termination crit. for optimization
cross:	number of folds in k-fold cross validation

The following table shows the relevance of the SVM parameters cost, nu and eps for the different SVMs:

SVM name	cost	nu	eps

C-svc:	x	-	-
l1rl2l-svc:	x	-	-
l1rl2lp-svc:	x	-	-
l1rl2l-svc:	x	-	-
nu-svc:	-	x	-
C-bsvc:	x	-	-
mc-natC:	x	-	-
mc-natW:	x	-	-
one-svc:	x	-	-
eps-svr:	-	-	x
nu-svr:	-	x	-
eps-bsvr:	-	-	x

Hint: Please be aware that identical parameter names between different SVMs do not necessarily mean, that their values are also identical between packages but they depend on the actual SVM formulation which could be different. For example the `cost` parameter is identical between C-SVMs in packages `kernlab`, `e1071` and `LiblineaR` but is for example different from the `cost` parameter in l2rl2l-svc in `LiblineaR` because the C-SVM uses a linear loss but the l2rl2l-svc uses a quadratic loss.

Feature weights

On user request (see parameter `featureWeights`) feature weights are computed amd stored in the model (for a detailed description see `getFeatureWeights`). Pruning of feature weights can be achieved with the parameter `weightLimit` which defines the cutoff for small feature weights not stored in the model.

Hint: For training with a precomputed kernel matrix feature weights are not available. For multiclass prediction is currently not performed via feature weights but native in the SVM.

Cross validation, grid search and model selection

Cross validation can be controlled with the parameters `cross` and `noCross`. For details on cross validation see crossValidation. Grid search can be performed by passing multiple SVM parameter values as vector instead of a single value to `kbsvm`. Also multiple sequence kernel objects and multiple SVMs can be used for grid search. For details see gridSearch. For model selection nested cross validation is used with the parameters `nestedCross` and `noNestedCross` for the outer and `cross` and `noCross` for the inner cross validation. For details see modelSelection.

Training with feature subset

After performing feature selection repeating the learning task with a feature subset can easily be achieved by specifying a feature subset with the parameter `features` as character vector. The fea-

ture subset must be a subset from the feature space of the sequence kernel passed in the parameter kernel. Grid search and model selection with a feature subset can only be used for a single sequence kernel object in the parameter kernel.

Hint: For normalized kernels all features of the feature space are used for normalization not just the feature subset. For a normalized motif kernel (see motifKernel) only the features listed in the motif list are part of the feature space. Therefore the motif kernel defined with the same feature subset leads to a different result in the normalized case.

Probability model

SVMs from the packages kernlab and e1071 support the generation of a probability model using Platt scaling (for details see kernlab, predict.ksvm, svm and predict.svm) allowing the computation of class probabilities during prediction. The parameter probabilityModel controls the generation of a probability model during training (see also parameter predictionType in predict).

Value

kbsvm: upon successful completion, the function returns a model of class KBModel. Results for cross validation can be retrieved from this model with the accessor cvResult, results for grid search or model selection with modelSelResult. In case of model selection the results of the outer cross validation loop can be retrieved with with the accessor cvResult.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

predict, getKernelMatrix, getExRep, kernelParameters-method, spectrumKernel, mismatchKernel, gappyPairKernel, motifKernel, getFeatureWeights

Examples

```
## load transcription factor binding site data
data(TFBS)
enhancerFB
## we use 70 of the samples for training and the rest for test
train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
test <- c(1:length(enhancerFB))[-train]
## create the kernel object for dimers without normalization
specK2 <- spectrumKernel(k=2)
## show details of kernel object
specK2
```

```
## run training with kernel matrix
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
               pkg="kernlab", svm="C-svc", C=10, explicit="no")

## show KeBABS model
model
## show class of KeBABS model
class(model)
## show native SVM model contained in KeBABS model
svmModel(model)
## show class of native SVM model
class(svmModel(model))

## Not run:
## examples for package and SVM selection
## now run the same samples with the same kernel on e1071 which
## currently only supports an explicit representation
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
               pkg="e1071", svm="C-svc", C=10, explicit="yes")

## show KeBABS model
model
## show native SVM model contained in KeBABS model
svmModel(model)
## show class of native SVM model
class(svmModel(model))

## run the same samples with the same kernel on e1071 with nu-SVM
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
               pkg="e1071", svm="nu-svc",nu=0.7, explicit="yes")

## show KeBABS model
model

## training with feature weights
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
               pkg="e1071", svm="C-svc", C=10, explicit="yes",
               featureWeights="yes")

## show feature weights
dim(featureWeights(model))
featureWeights(model)[,1:5]

## training without feature weights
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
               pkg="e1071", svm="C-svc", C=10, explicit="yes",
               featureWeights="no")

## show feature weights
featureWeights(model)

## pruning of feature weights
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
```

```
                pkg="e1071", svm="C-svc", C=10, explicit="yes",
                featureWeights="yes", weightLimit=0.5)

dim(featureWeights(model))

## training with precomputed kernel matrix
km <- getKernelMatrix(specK2, x=enhancerFB, selx=train)
model <- kbsvm(x=km, y=yFB[train], kernel=specK2,
                pkg="kernlab", svm="C-svc", C=10, explicit="no")

## training with precomputed explicit representation
exrep <- getExRep(enhancerFB, sel=train, kernel=specK2)
model <- kbsvm(x=exrep, y=yFB[train], kernel=specK2,
                pkg="e1071", svm="C-svc", C=10, explicit="yes")

## computing of probability model via Platt scaling during training
## in prediction class membership probabilities can be computed
## from this probability model
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK2,
                pkg="e1071", svm="C-svc", C=10, explicit="yes",
                probModel=TRUE)

## show parameters of the fitted probability model which are the parameters
## probA and probB for the fitted sigmoid function in case of classification
## and the value sigma of the fitted Laplacian in case of a regression
probabilityModel(model)

## cross validation, grid search and model selection are also performed
## via the kbsvm method. Examples can be found on the respective help pages
## (see Details section)

## End(Not run)
```

---

kebabsData *KeBABS Sequence Data*

Description

The package contains two small sequence datasets for demonstration of the package functionality.

TFBS is a subset of EP300/CREBBP binding data provided with the publication Lee et al., 2011. The data is based on binding sites identified with ChIP-seq by Visel et al., 2009. Please note that due to package size restrictions only a small subset of the data used in Lee et al., 2011 is included in the package. Following variables are defined:

- enhancerFB contains 259 DNA sequences of tissue specific enhancers from embryonic day 11.5 mouse embryos and 241 negative sequences sampled from mm9 genome.
- yFB contains the associated labels

CCoil is a set of heptad-annotated amino acid sequences of coiled coil proteins forming dimers or trimers from the web site of the package **PrOCoil** by Mahrenholz et. al., 2011. The data contains the sequences with heptad annotation, the oligomerization state and group assignment for each sequence. The grouping was performed through single linkage clustering of sequence similarities based on pairwise ungapped alignment. Following variables are defined:

- `ccseq` contains 477 AA sequences of heptad-annotated amino acid sequences with a minimum length of 8 and a maximun length of 123 AAs.
- `yCC` contains the associated oligomerization state "DIMER" or "TRIMER".
- `ccannot` is a charcter vector with the heptad annotations for the sequences. Characters 'a' to 'f' represent specific positions within the coiled coil structure. The AA string set already contains the annotation as metadata. But for demonstration purpose it is available as separate data item.
- `ccgroups` is a numeric vector containing the group numbers of of the sequences.

Format

TFBS contains the 259 positive and 241 negative sequences as DNAStringSet and the corresponding labels as numeric vector containing a value of 1 for positive and -1 for negative samples.

CCoil contains the 477 AA sequences as AAStringSet and the corresponding labels as factor. The heptad anntoation is stored as character vector and group assignment as numeric vector.

Source

TFBS: `http://www.beerlab.org/p300enhancer`

CCoil: `http://www.bioinf.jku.at/software/procoil/data.html`

References

(Lee, 2011) – D. Lee, R. Karchin and M. A. Beer. Discriminative prediction of mammalian enhancers from DNA sequence. Genome Research, 21(12):2167-2180, 2011.

(Visel, 2009) – A. Visel, M. J. Blow, Z. Li, T. Zhang, J. A. Akiyama, A. Holt, I. Plajzer-Frick, M. Shoukry, C. Wright, F.Chen, V. Afzal, B. Ren, E. M. Rubin and L. A. Pennacchio. ChIP-seq accurately predicts tissue-specific activity of enhancers. Nature, 457(7231):854-858, 2009.

(Mahrenholz, 2011) – C. Mahrenholz, I. Abfalter, U. Bodenhofer, R. Volkmer and S. Hochreiter. Complex networks govern coiled-coil oligomerizations - predicting and profiling by means of a machine learning approach.

kebabsDemo *kebabs*

Description

KeBABS - An R package for kernel based analysis
of biological sequences

Usage

```
kebabsDemo()
```

Details

Package Overview

The package provides functionality for kernel based analysis of DNA-, RNA- and amino acid sequences via SVM based methods. As core functionality kebabs contains following sequence kernels: spectrum kernel, mismatch kernel, gappy pair kernel and motif kernel. Apart from an efficient implementation of position independent functionality the kernels are extended in a novel way to take the position of patterns into account for the similarity measure. Because of the flexibility of the kernel formulation other kernels like the weighted degree kernel or the shifted weighted degree kernel are included as special cases. An annotation specific variant of the kernels uses annotation information placed along the sequence together with the patterns in the sequence. The package allows generation of a kernel matrix or an explicit representation for all available kernels which can be used with methods implemented in other R packages. With focus on SVM based methods kebabs provides a framework which simplifies the usage of existing SVM implementations in kernlab, e1071 and LiblineaR. Binary and multiclass classification as well as regression tasks can be used in a unified way without having to deal with the different functions, parameters and formats of the selected SVM. As support for choosing hyperparameters the package provides cross validation, grid search and model selection functions.For easier biological interpretation of the results the package computes feature weights for all SVMs and prediction profiles, which show the contribution of individual sequence positions to the prediction result and give an indication about the relevance of sequence sections for the learning result and the underlying biological functions.

Value

see above

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

Examples

```
## load package provided sequence dataset
data(TFBS)

## display sequences
enhancerFB

## display part of label vector
head(yFB, 20)

## display no of samples of positive and negative class
table(yFB)

## split dataset into training and test samples
train <- sample(1:length(enhancerFB), 0.7*length(enhancerFB))
test <- c(1:length(enhancerFB))[-train]

## create the kernel object for the normalized spectrum kernel
spec <- spectrumKernel(k=5)

## train model
## pass sequence subset, label subset, kernel object, the package and
## svm which should be used for training together with the SVM parameters
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=spec,
               pkg="LiblineaR", svm="C-svc", cost=10)

## predict the test samples
pred <- predict(model, enhancerFB, sel=test)

## evaluate the prediction result
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))
```

KernelMatrix-class *Kernel Matrix Class*

Description

Kernel Matrix Class

Details

Instances of this class are used in KeBABS for storing a kernel matrix. The hidden data part ".Data" contains the matrix.

KernelMatrixAccessors *KernelMatrix Accessors*

Description

KernelMatrix Accessors

Usage

```
## S4 method for signature KernelMatrix,index,missing,ANY
x[i]

## S4 method for signature matrix
as.KernelMatrix(x, center = FALSE)
```

Arguments

x	kernel matrix of class KernelMatrix
i	numeric vector with indicies or character with element names
center	when set to TRUE the matrix is centered. Default=FALSE

Value

see above

Accessor-like methods

x[i,]: return a KernelMatrix object that only contains the rows selected with the subsetting parameter i. This parameter can be a numeric vector with indices or a character vector which is matched against the names of x.

x[,j]: return a KernelMatrix object that only contains the columns selected with the subsetting parameter j. This parameter can be a numeric vector with indices or a character vector which is matched against the names of x.

x[i,j]: return a KernelMatrix object that only contains the rows selected with the subsetting parameter i and columns selected by j. Both parameters can be a numeric vector with indices or a character vector which is matched against the names of x.

Coercion methods

In the code snippets below, x is a kernel matrix.

as.KernelMatrix(x, center=FALSE): center the kernel matrix dependent on the center parameter and coerce it to class KernelMatrix .

Examples

```
## create kernel object for normalized spectrum kernel
specK5 <- spectrumKernel(k=5)
## Not run:
## load data
data(TFBS)

km <- specK5(enhancerFB)
km1to5 <- km[1:5,1:5]
km1to5

## End(Not run)
```

`linearKernel` *Linear Kernel*

Description

Create a dense or sparse kernel matrix from an explicit representation

Usage

```
linearKernel(x, y = NULL, selx = integer(0), sely = integer(0),
  sparse = FALSE, triangular = TRUE, diag = TRUE, lowerLimit = 0)
```

Arguments

`x`	a dense or sparse explicit representation. `x` must be a sparse explicit representation when a sparse kernel matrix should be returned by the function (see parameter `sparse`).
`y`	a dense or sparse explicit representation. If `x` is dense, `y` must be dense. If `x` is sparse, `y` must be sparse.
`selx`	a numeric or character vector for defining a subset of `x`. Default=integer(0)
`sely`	a numeric or character vector for defining a subset of `y`. Default=integer(0)
`sparse`	boolean indicating whether returned kernel matrix should be sparse or dense. For value `FALSE` a dense kernel matrix of class `KernelMatrix` is returned. If set to `TRUE` the kernel matrix is returned as sparse matrix of class `dgCMatrix`. In case of a symmetric matrix either the lower triangular part or the full matrix can be returned. Please note that a sparse kernel matrix currently can not be used for SVM based learning in `kebabs`. Default=FALSE
`triangular`	boolean indicating whether just the lower triangular or the full sparse matrix should be returned. This parameter is only relevant for a sparse symmetric kernel matrix. Default=TRUE
`diag`	boolean indicating whether the diagonal should be included in a sparse triangular matrix. This parameter is only relevant when parameter `sparse` and `triangular` are set to `TRUE`. Default=TRUE

lowerLimit — a numeric value for a similarity threshold. The parameter is relevant for sparse kernel matrices only. If set to a value larger than 0 only similarity values larger than this threshold will be included in the sparse kernel matrix. Default=0

Value

linearKernel: kernel matrix as class KernelMatrix or sparse kernel matrix of class dgCMatrix dependent on parameter sparse

Examples

```
## load sequence data and change sample names
data(TFBS)
names(enhancerFB) <- paste("S", 1:length(enhancerFB), sep="_")

## create the kernel object for dimers with normalization
speck <- spectrumKernel(k=5)

## generate sparse explicit representation
ers <- getExRep(enhancerFB, speck)

## compute dense kernel matrix (as currently used in SVM based learning)
km <- linearKernel(ers)
km[1:5, 1:5]

## compute sparse kernel matrix
## because it is symmetric just the lower diagonal
## is computed to save storage
km <- linearKernel(ers, sparse=TRUE)
km[1:5, 1:5]

## compute full sparse kernel matrix
km <- linearKernel(ers, sparse=TRUE, triangular=FALSE)
km[1:5, 1:5]

## compute triangular sparse kernel matrix without diagonal
km <- linearKernel(ers, sparse=TRUE, triangular=TRUE, diag=FALSE)
km[1:5, 1:5]

## plot histogram of similarity values
hist(km@x, breaks=30)

## compute sparse kernel matrix with similarities above 0.5 only
km <- linearKernel(ers, sparse=TRUE, lowerLimit=0.5)
km[1:5, 1:5]
```

---

linWeight *Position Dependent Kernel*

Description

Assign position related metadata and reate a kernel object with position dependency

Usage

```
linWeight(d, sigma = 1)

expWeight(d, sigma = 1)

gaussWeight(d, sigma = 1)

swdWeight(d)

## S4 method for signature XStringSet
## positionMetadata(x) <- value

## S4 method for signature BioVector
## positionMetadata(x) <- value

## S4 method for signature XStringSet
positionMetadata(x)

## S4 method for signature BioVector
positionMetadata(x)
```

Arguments

d	a numeric vector of distance values
sigma	a positive numeric value defining the peak width or in case of gaussWeight the width of the bell function (details see below)
x	biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)
value	for assignment of position metadata the value is an integer vector with gives the offset to the start position 1 for each sequence. Positive and negative offset values are possible. Without position metadata all sequences must be aligned and start at position 1. For deletion of position metadata set value to NULL.

Details

Position Dependent Kernel

For the standard spectrum kernel kmers are considered independent of their position in the calculation of the similarity value between two sequences. For position dependent kernels the position of a kmer/pattern is also of importance. Position information for a pair of sequences can be used in a sequenceKernel in three different ways representing the full range of position dependency:

- Position independent kernel: ignores the position of patterns and just takes the number of their occurances or their presence (see parameter presence in functions spectrumKernel, gappyPairKernel, motifKerne
in the sequences into account for similarity determination.

- Distance weighted kernel: uses the position related distance between the occurance of the same pattern in the two sequences in weighted form as contribution to the similarity value (see below under Distance Weighted kernel)
- Position specific kernel: considers patterns only if they occur at the same position in the two sequences (see below under Position Specific Kernel)

Position dependency is available in all kernels except the mismatch kernel.

Distance Weighted Kernel

These kernels weight the contribution to the similarity value based on the distance of their start positions in the two sequences. The user can define the distance weights either through passing a distance weighting function or a weight vector to the kernel. Through this weighting the degree of locality in the similarity consideration between two sequences can be adjusted flexibly. Such a position dependent kernel can be used in the same way as the normal position independent kernel variant. Distance weighting can be used for all kernels in this package except the mismatch kernel. The package defines four predefined weighting functions (see also examples):

- linWeigth: a weighting function with linear decrease
- expWeight: a weighting function with exponential decrease
- gaussWeigth: a bell-shaped weighting function with a decrease similar to a gaussian distribution
- swdWeight: the distance weighting function used in the Shifted Weighted Degree (SWD) kernel which is similar to an exponential decrease but it has a smaller peak and larger tails

Also user-defined functions can be used for distance weighting. (see below)

Position Specific Kernel

One variant of position dependent kernels is the position specific kernel. This kernel takes patterns into account only if they are located at identical positions in the two sequences. This kernel can be selected through passing a distance weight value of 1 to the kernel indicating that the neighborhood of a pattern in the other sequence is irrelevant for the similarity consideration. This kernel is in fact one end of the spectrum (sic!) where locality is reduced to the exact location and the normal position independent kernel is at the other end - not caring about position at all. Through adjustment of sigma in the predefined functions a continous blending between these two extremes is possible for the degree of locality. Evaluation of position information is controlled through setting the parameter `distWeight` to 1 in the functions `spectrumKernel`,`gappyPairKernel`, `motifKernel`. This parameter value is in fact interpreted as a numeric vector with 1 for zero distance and 0 for all other distances.

Positive Definiteness

The standard SVMs only support positive definite kernels / kernel matrices. This means that the distance weighting function must must be chosen such that the resulting kernel is positive definite. For positive definiteness also symmetry of the distance weighting function is important. Unlike usual distances the relative distance value here can have positive and negative values dependent on whether the pattern in the second sequence is located at higher or lower positions than the pattern

in the first sequence. The predefined distance weighting functions except for swdWeight deliver a positive definite kernel for all parameter settings. According to Sonnenburg et al. 2005 the SWD kernel has empirically shown positive definiteness but it is not proved for this kernel. If a weight vector with predefined weights per distance is passed to the kernel instead of a distance weighting function positive definiteness of the kernel must also be ensured by adequate selection of the weight values.

User-Defined Distance Function

For user defined distance functions symmetry and positive definitness of the resulting kernel are important. Such a function gets a numeric distance vector 'x' as input (and possibly other parameters controlling the weighting behavior) and returns a weight vector of identical length. When called with a missing parameter x all other parameters must be supplied or have appropriate default values. In this case the function must return a new function with just the single parameter x which calls the original user defined function with x and all the other parameters set to the values passed in the call.

This behavior is needed for assignment of the function with missing parameter x to the distWeight parameter in the kernel. At the time of kernel definition the actual distance values are not available. Later when sequence data is passed to this kernel for generation of a kernel matrix or an explicit representation this single argument function is called to get the distance dependent weights. The code for the predefined `expWeight` function in the example section below shows how a user-specific function can be set up.

Offset

To allow flexible alignment of sequence positions without redefining the XStringSet or BioVector an additional metadata element named offset can be assigned to the sequence set via `positionMetadata<-` (see example below). Position metadata is a numeric vector with the same number of elements as the sequence set and gives for each sequence an offset to position 1. When positions metadata is not assigned to a sequence set the position 1 is associated with the first character in each sequence of the sequence set., i.e. in this case the sequences should be aligned such that all have the same starting positions with respect to the learning task (e.g. all sequences start at a transcription start site). Offset information is only evaluated in position dependent kernel variants.

Value

The distance weighting functions return a numerical vector with distance weights.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Bodenhofer, 2009) – U. Bodenhofer, K. Schwarzbauer, M. Ionescu and S. Hochreiter. Modelling position specificity in sequence kernels by fuzzy equivalence relations.

(Sonnenburg, 2005) – S. Sonnenburg, G. Raetsch and B. Schoelkopf. Large Scale Genomic Sequence SVM Classifiers http://www.bioinf.jku.at/software/kebabs

See Also

spectrumKernel, gappyPairKernel, motifKernel, annotationMetadata, metadata, mcols

Examples

```
## plot predefined weighting functions for sigma=10
curve(linWeight(x, sigma=10), from=-20, to=20, xlab="pattern distance",
ylab="weight", main="Predefined Distance Weighting Functions", col="green")
curve(expWeight(x, sigma=10), from=-20, to=20, col="blue", add=TRUE)
curve(gaussWeight(x, sigma=10), from=-20, to=20, col="red", add=TRUE)
curve(swdWeight(x), from=-20, to=20, col="orange", add=TRUE)
legend(topright, inset=0.03, title="Weighting Functions", c("linWeight",
       "expWeight", "gaussWeight", "swdWeight"),
       fill=c("green", "blue", "red", "orange"))
text(14, 0.70, "sigma = 10")

## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the motif kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                          "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
                          "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create a distance weighted spectrum kernel with linear decrease of
## weights in a range of 20 bases
spec20 <- spectrumKernel(k=3, distWeight=linWeight(sigma=20))

## show details of kernel object
kernelParameters(spec20)

## this kernel can be now be used in a classification or regression task
## in the usual way or a kernel matrix can be generated for use with
## another learning method
km <- spec20(x=dnaseqs, selx=1:5)
km[1:5,1:5]

## Not run:
## instead of a distance weighting function also a weight vector can be
## passed in the distWeight parameter but the values must be chosen such
## that they lead to a positive definite kernel
##
## in this example only patterns within a 5 base range are considered with
## slightly decreasing weights
specv <- spectrumKernel(k=3, distWeight=c(1,0.95,0.9,0.85,0.8))
km <- specv(dnaseqs)
km[1:5,1:5]
```

```
## position specific spectrum kernel
specps <- spectrumKernel(k=3, distWeight=1)
km <- specps(dnaseqs)
km[1:5,1:5]

## get position specific kernel matrix
km <- specps(dnaseqs)
km[1:5,1:5]

## example with offset to align sequence positions (e.g. the
## transcription start site), the value gives the offset to position 1
positionOne <- c(9,6,3,1,6)
positionMetadata(dnaseqs) <- positionOne
## show position metadata
positionMetadata(dnaseqs)
## generate kernel matrix with position-specific spectrum kernel
km1 <- specps(dnaseqs)
km1[1:5,1:5]

## example for a user defined weighting function
## please stick to the order as described in the comments below and
## make sure that the resulting kernel is positive definite

expWeightUserDefined <- function(x, sigma=1)
{
    ## check presence and validity of all parameters except for x
    if (!isSingleNumber(sigma))
        stop("sigma must be a number")

    ## if x is missing the function returns a closure where all parameters
    ## except for x have a defined value
    if (missing(x))
        return(function(x) expWeightUserDefined(x, sigma=sigma))

    ## pattern distance vector x must be numeric
    if (!is.numeric(x))
        stop("x must be a numeric vector")

    ## create vector of distance weights from the
    ## input vector of pattern distances x
    exp(-abs(x)/sigma)
}

## define kernel object with user defined weighting function
specud <- spectrumKernel(k=3, distWeight=expWeightUserDefined(sigma=5),
                  normalized=FALSE)

## End(Not run)
```

`mismatchKernel` *Mismatch Kernel*

Description

Create a mismatch kernel object and the kernel matrix

Usage

```
mismatchKernel(k = 3, m = 1, r = 1, normalized = TRUE, exact = TRUE,
  ignoreLower = TRUE, presence = FALSE)

## S4 method for signature MismatchKernel
getFeatureSpaceDimension(kernel, x)
```

Arguments

k	length of the substrings also called kmers; this parameter defines the size of the feature space, i.e. the total number of features considered in this kernel is |A|^k, with |A| as the size of the alphabet (4 for DNA and RNA sequences and 21 for amino acid sequences). Default=3
m	number of maximal mismatch per kmer. The allowed value range is between 1 and k-1. The processing effort for this kernel is highly dependent on the value of m and only small values will allow efficient processing. Default=1
r	exponent which must be > 0 (see details section in spectrumKernel). Default=1
normalized	a kernel matrix or explicit representation generated with this kernel will be normalized(details see below). Default=TRUE
exact	use exact character set for the evaluation (details see below). Default=TRUE
ignoreLower	ignore lower case characters in the sequence. If the parameter is not set lower case characters are treated like uppercase. Default=TRUE
presence	if this parameter is set only the presence of a kmers will be considered, otherwise the number of occurances of the kmer is used. Default=FALSE
kernel	a sequence kernel object
x	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)

Details

Creation of kernel object

The function 'mismatchKernel' creates a kernel object for the mismatch kernel. This kernel object can then be used with a set of DNA-, RNA- or AA-sequences to generate a kernel matrix or an explicit representation for this kernel. For values different from 1 (=default value) parameter r leads to a transfomation of similarities by taking each element of the similarity matrix to the power of r. If normalized=TRUE, the feature vectors are scaled to the unit sphere before computing the similarity value for the kernel matrix. For two samples with the feature vectors x and y the similarity is computed as:

$$s = \frac{\vec{x}^T \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

For an explicit representation generated with the feature map of a normalized kernel the rows are normalized by dividing them through their Euclidean norm. For parameter `exact=TRUE` the sequence characters are interpreted according to an exact character set. If the flag is not set ambigous characters from the IUPAC characterset are also evaluated. The annotation specific variant (for details see positionMetadata) and the position dependent variant (for details see annotationMetadata) are not available for this kernel.

Creation of kernel matrix

The kernel matrix is created with the function `getKernelMatrix` or via a direct call with the kernel object as shown in the examples below.

Value

mismatchKernel: upon successful completion, the function returns a kernel object of class `MismatchKernel`.

of getDimFeatureSpace: dimension of the feature space as numeric value

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Leslie, 2002) – C. Leslie, et al. Mismatch String Kernels for SVM Protein Classification.

http://www.bioinf.jku.at/software/kebabs

See Also

`kernelParameters`, `getKernelMatrix`, `getExRep`, `spectrumKernel`, `gappyPairKernel`, `motifKernel`, `MismatchKernel`

Examples

```
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the mismatch kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                          "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
                          "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create the kernel object with one mismatch per kmer
mm <- mismatchKernel(k=2, m=1, normalized=FALSE)
## show details of kernel object
mm

## generate the kernel matrix with the kernel object
km <- mm(dnaseqs)
```

```
dim(km)
km[1:5, 1:5]

## alternative way to generate the kernel matrix
km <- getKernelMatrix(mm, dnaseqs)
km[1:5,1:5]

## Not run:
## plot heatmap of the kernel matrix
heatmap(km, symm=TRUE)

## End(Not run)
```

`MismatchKernel-class` *Mismatch Kernel Class*

Description

Mismatch Kernel Class

Details

Instances of this class represent a kernel object for the mismatch kernel. The class is derived from SequenceKernel.

Slots

`k` length of the substrings considered by the kernel

`m` maximum number of mismatches

`r` exponent (for details see mismatchKernel)

`annSpec` not used for mismatch kernel

`distWeight` not used for mismatch kernel

`normalized` data generated with this kernel object is normalized

`exact` use exact character set for evaluation

`ignoreLower` ignore lower case characters in the sequence

`presence` consider only the presence of kmers not their counts

`revComplement` not used for mismatch kernel

`mixCoef` not used for mismatch kernel

`ModelSelectionResult-class`
Model Selection Result Class

Description

Model Selection Result Class

Details

Instances of this class store the result of grid search or model selection.

Slots

`cross` number of folds for cross validation

`noCross` number of CV runs

`groupBy` group assignment of samples

`nestedCross` number of folds for outer CV

`noNestedCross` number of runs of outer CV

`perfParameters` collected performance parameters

`perfObjective` performance criterion for grid search / model selection

`gridRows` rows in grid search (i.e. kernels)

`gridCols` columns in grid search

`gridErrors` grid errors

`gridACC` grid accuracy

`gridBACC` grid balanced accuracy

`gridMCC` grid Matthews correlation coefficient

`gridNoSV` grid number of support vectors

`gridSumAlphas` grid sum of alphas

`smallestCVError` smallest CV error

`selGridRow` grid row of best result

`selGridCol` grid col of best result

`fullModel` full model for best result

ModelSelectionResultAccessors
ModelSelectionResult Accessors

Description

ModelSelectionResult Accessors

Usage

```
## S4 method for signature ModelSelectionResult
gridRows(object)
```

Arguments

object	a model selection result object (can be extracted from KeBABS model with accessor modelSelResult)

Value

gridRows: returns a list of kernel objects
gridColumns: returns a DataFrame object with grid column parameters
gridErrors: returns a matrix with grid errors
performance: returns a list of matrices with performance values selGridRow: returns the selected kernel selGridCol: returns the selected SVM and/or hyperparameter(s) fullModel: returns a kebabs model of class KBModel

Accessor-like methods

gridRows: return the grid rows containing the kernels.

gridColumns: return the grid columns.

gridErrors: return the grid CV errors.

performance: return the collected performance parameters.

selGridRow: return the selected grid row.

selGridCol: return the selected grid column.

fullModel: return the full model.

Examples

```
## create kernel object for normalized spectrum kernel
specK5 <- spectrumKernel(k=5)
## Not run:
## load data
data(TFBS)

## perform training - feature weights are computed by default
```

```
model <- kbsvm(enhancerFB, yFB, specK5, pkg="LiblineaR",
               svm="C-svc", cost=c(1,15,50,100), cross=10,
               perfParameters="ALL", showProgress=TRUE)

## show model selection result
mres <- modelSelResult(model)
mres

## extract grid errors
gridErrors(mres)

## extract other performance parameters
performance(mres)

## End(Not run)
```

`motifKernel` *Motif Kernel*

Description

Create a motif kernel object and the kernel matrix

Usage

```
motifKernel(motifs, r = 1, annSpec = FALSE, distWeight = numeric(0),
  normalized = TRUE, exact = TRUE, ignoreLower = TRUE, presence = FALSE)

## S4 method for signature MotifKernel
getFeatureSpaceDimension(kernel, x)
```

Arguments

`motifs`	a set of motif patterns specified as character vector. The order in which the patterns are passed for creation of the kernel object also determines the order of the features in the explicit representation. Lowercase characters in motifs are always converted to uppercase. For details concerning the definition of motif patterns see below and in the examples section.
`r`	exponent which must be > 0 (see details section in spectrumKernel). Default=1
`annSpec`	boolean that indicates whether sequence annotation should be taken into account (details see on help page for `annotationMetadata`). Default=FALSE
`distWeight`	a numeric distance weight vector or a distance weighting function (details see on help page for `gaussWeight`). Default=NULL
`normalized`	generated data from this kernel will be normalized (details see below). Default=TRUE
`exact`	use exact character set for the evaluation (details see below). Default=TRUE

ignoreLower	ignore lower case characters in the sequence. If the parameter is not set lower case characters are treated like uppercase. default=TRUE
presence	if this parameter is set only the presence of a motif will be considered, otherwise the number of occurances of the motif is used; Default=FALSE
kernel	a sequence kernel object
x	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)

Details

Creation of kernel object

The function 'motif' creates a kernel object for the motif kernel for a set of given DNA-, RNA- or AA-motifs. This kernel object can then be used to generate a kernel matrix or an explicit representation for this kernel. The individual patterns in the set of motifs are built similar to regular expressions through concatination of following elements in arbitrary order:

- a specific character from the used character set - e.g. 'A' or 'G' in DNA patterns for matching a specific character
- the wildcard character '.' which matches any valid character of the character set except '-'
- a substitution group specified by a collection of characters from the character set enclosed in square brackets - e.g. [AG] - which matches any of the listed characters; with a leading '^' the character list is inverted and matching occurs for all characters of the character set which are not listed except '-'

For values different from 1 (=default value) parameter r leads to a transfomation of similarities by taking each element of the similarity matrix to the power of r. For the annotation specific variant of this kernel see annotationMetadata, for the distance weighted variants see positionMetadata. If normalized=TRUE, the feature vectors are scaled to the unit sphere before computing the similarity value for the kernel matrix. For two samples with the feature vectors x and y the similarity is computed as:

$$s = \frac{\vec{x}^T \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

For an explicit representation generated with the feature map of a normalized kernel the rows are normalized by dividing them through their Euclidean norm. For parameter exact=TRUE the sequence characters are interpreted according to an exact character set. If the flag is not set ambigous characters from the IUPAC characterset are also evaluated.

The annotation specific variant (for details see annotationMetadata) and the position dependent variants (for details see positionMetadata) either in the form of a position specific or a distance weighted kernel are supported for the motif kernel. The generation of an explicit representation is not possible for the position dependent variants of this kernel.

Hint: For a normalized motif kernel with a feature subset of a normalized spectrum kernel the explicit representation will not be identical to the subset of an explicit representation for the spectrum kernel because the motif kernel is not aware of the other kmers which are used in the spectrum kernel additionally for normalization.

Creation of kernel matrix

The kernel matrix is created with the function getKernelMatrix or via a direct call with the kernel object as shown in the examples below.

Value

motif: upon successful completion, the function returns a kernel object of class MotifKernel.

of getDimFeatureSpace: dimension of the feature space as numeric value

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Ben-Hur, 2003) – A. Ben-Hur, and D. Brutlag. Remote homology detection: a motif based approach.

(Bodenhofer, 2009) – U. Bodenhofer, K. Schwarzbauer, M. Ionescu and S. Hochreiter. Modelling position specificity in sequence kernels by fuzzy equivalence relations.
(Mahrenholz, 2011) – C. Mahrenholz, I. Abfalter, U. Bodenhofer, R. Volkmer and S. Hochreiter. Complex networks govern coiled-coil oligomerizations - predicting and profiling by means of a machine learning approach.

http://www.bioinf.jku.at/software/kebabs

See Also

kernelParameters-method, getKernelMatrix, getExRep, spectrumKernel, mismatchKernel, gappyPairKernel

Examples

```
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the motif kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                          "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
                          "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create the kernel object with the motif patterns
mot <- motifKernel(c("A[CG]T","C.G","G[^A][AT]"), normalized=FALSE)
## show details of kernel object
mot

## generate the kernel matrix with the kernel object
km <- mot(dnaseqs)
dim(km)
```

```
km

## alternative way to generate the kernel matrix
km <- getKernelMatrix(mot, dnaseqs)

## Not run:
## plot heatmap of the kernel matrix
heatmap(km, symm=TRUE)

## generate rectangular kernel matrix
km <- mot(x=dnaseqs, selx=1:3, y=dnaseqs, sely=4:5)
dim(km)
km

## End(Not run)
```

`MotifKernel-class` *Motif Kernel Class*

Description

Motif Kernel Class

Details

Instances of this class represent a kernel object for the motif kernel. The class is derived from SequenceKernel. The motif character vector is not stored in the kernel object.

Slots

`r` exponent (for details see motifKernel)

`annSpec` when set the kernel evaluates annotation information

`distWeight` distance weighting function or vector

`normalized` data generated with this kernel object is normalized

`exact` use exact character set for evaluation

`ignoreLower` ignore lower case characters in the sequence

`presence` consider only the presence of motifs not their counts

`revComplement` consider a kmer and its reverse complement as the same feature

`performCrossValidation,KernelMatrix-method`
KeBABS Cross Validation

Description

Perform cross validation as k-fold cross validation, Leave-One-Out cross validation(LOOCV) or grouped cross validation (GCV).

Usage

```
## kbsvm(......, cross=0, noCross=1, .....)

## please use kbsvm for cross validation and do not call the
## performCrossValidation method directly

## S4 method for signature ExplicitRepresentation
performCrossValidation(object, x, y, sel,
  model, cross, noCross, groupBy, perfParameters, verbose)
```

Arguments

`object`	a kernel matrix or an explicit representation
`x`	an optional set of sequences
`y`	a response vector
`sel`	sample subset for which cross validation should be performed
`model`	KeBABS model
`cross`	an integer value K > 0 indicates that k-fold cross validation should be performed. A value -1 is used for Leave-One-Out (LOO) cross validation. (see above) Default=0
`noCross`	an integer value larger than 0 is used to specify the number of repetitions for cross validation. This parameter is only relevant if 'cross' is different from 0. Default=1
`groupBy`	allows a grouping of samples during cross validation. The parameter is only relevant when 'cross' is larger than 1. It is an integer vector or factor with the same length as the number of samples used for training and specifies for each sample to which group it belongs. Samples from the same group are never spread over more than one fold. Grouped cross validation can also be used in grid search for each grid point. Default=NULL
`perfParameters`	a character vector with one or several values from the set "ACC" , "BACC", "MCC" and "ALL". "ACC" stands for accuracy, "BACC" for balanced accuracy, "MCC" for Matthews Correlation Coefficient and "ALL" for all three. This parameter defines which performance parameters are collected in cross validation, grid search and model selection for display purpose. Default=NULL

`verbose` boolean value that indicates whether KeBABS should print additional messages showing the internal processing logic in a verbose manner. The default value depends on the R session verbosity option. Default=getOption("verbose")

this parameter is not relevant for cross validation because the method `performCrossValidation` should not be called directly. Cross validation is performed with the method `kbsvm` and the parameters `cross` and `numCross` are described there

Details

Overview

Cross validation (CV) provides an estimate for the generalization performance of a model based on repeated training on different subsets of the data and evaluating the prediction performance on the remaining data not used for training. Dependent on the strategy of splitting the data different variants of cross validation exist. KeBABS implements k-fold cross validation, Leave-One-Out cross validation and Leave-Group-Out cross validation which is a specific variant of k-fold cross validation. Cross validation is invoked with `kbsvm` through setting the parameters `cross` and `noCross`. It can either be used for a given kernel and specific values of the SVM hyperparameters to compute the cross validation error of a single model or in conjuction with grid search (see gridSearch) and model selection (see modelSelection) to determine the performance of multiple models.

k-fold Cross Validation and Leave-One-Out Cross Validation(LOOCV)

For k-fold cross validation the data is split into k roughly equal sized subsets called folds. Samples are assigned to the folds randomly. In k successive training runs one of the folds is kept in round-robin manner for predicting the performance while using the other k-1 folds together as training data. Typical values for the number of folds k are 5 or 10 dependent on the number of samples used for CV. For LOOCV the fold size decreases to 1 and only a single sample is kept as hold out fold for performance prediction requiring the same number of training runs in one cross validation run as the number of sequences used for CV.

Grouped Cross Validation (GCV)

For grouped cross validation samples are assigned to groups by the user before running cross validation, e.g. via clustering the sequences. The predefined group assignment is passed to CV with the parameter `groupBy` in `kbsvm`. GCV is a special version of k-fold cross validation which respects group boundaries by avoiding to distribute samples of one group over multiple folds. In this way the group(s) in the test fold do not occur during training and learning is forced to concentrate on more complex features instead of the simple features splitting the groups. For GCV the parameter cross must be smaller than or equal to the number of groups.

Cross Validation Result

The cross validation error, which is the average of the predicition errors in all held out folds, is used as an estimate for the generalization error of the model assiciated with the cross validation run. For classification the fraction of incorrectly classified samples and for regression the mean squared error (MSE) is used as prediction error. Multiple cross validation runs can be performed through setting the parameter `noCross`. The cross validation result can be extracted from the model object returned by cross validation with the `cvResult` accessor. It contains the mean CV error over all runs, the CV errors of the single runs and the CV error for each fold. The CV result object can be plotted with the method `plot` showing the variation of the CV error for the different runs as barplot. With the parameter `perfParameters` in `kbsvm` the accuracy, the balanced accuracy and the Matthews correlation coefficient can be requested as additional performance parameters to be recorded in the CV result object which might be of interest especially for unbalanced datasets.

Value

cross validation stores the cross validation results in the KeBABS model object returned by . They can be retrieved with the accessor `cvResult` returned by `kbsvm`.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

`kbsvm`, `cvResult`, `plot`

Examples

```
## load transcription factor binding site data
data(TFBS)
enhancerFB
## select a few samples for training - here for demonstration purpose
## normally you would use 70 or 80% of the samples for training and
## the rest for test
## train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
## test <- c(1:length(enhancerFB))[-train]
train <- sample(1:length(enhancerFB), 50)
## create a kernel object for the gappy pair kernel with normalization
gappy <- gappyPairKernel(k=1, m=4)
## show details of kernel object
gappy

## run cross validation with the kernel on C-svc in LiblineaR for cost=10
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
               pkg="LiblineaR", svm="C-svc", cost=10, cross=3)
```

```
## show cross validation result
cvResult(model)

## Not run:
## perform tive cross validation runs
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
               pkg="LiblineaR", svm="C-svc", cost=10, cross=10, noCross=5)

## show cross validation result
cvResult(model)

## plot cross validation result
plot(cvResult(model))


## run Leave-One-Out cross validation
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
               pkg="LiblineaR", svm="C-svc", cost=10, cross=-1)

## show cross validation result
cvResult(model)

## run gouped cross validation with full data
## on coiled coil dataset
##
## In this example the groups were determined through single linkage
## clustering of sequence similarities derived from ungapped heptad-specific
## pairwise alignment of the sequences. The variable {\tt ccgroup} contains
## the pre-calculated group assignments for the individual sequences.
data(CCoil)
ccseq
head(yCC)
head(ccgroups)
gappyK1M6 <- gappyPairKernel(k=1, m=4)

## run k-fold CV without groups
model <- kbsvm(x=ccseq, y=as.numeric(yCC), kernel=gappyK1M6,
pkg="LiblineaR", svm="C-svc", cost=10, cross=3, noCross=2,
perfObjective="BACC",perfParameters=c("ACC", "BACC"))

## show result without groups
cvResult(model)

## run grouped CV
model <- kbsvm(x=ccseq, y=as.numeric(yCC), kernel=gappyK1M6,
pkg="LiblineaR", svm="C-svc", cost=10, cross=3,
noCross=2, groupBy=ccgroups, perfObjective="BACC",
perfParameters=c("ACC", "BACC"))

## show result with groups
cvResult(model)

## For grouped CV the samples in the held out fold are from a group which
```

```
## is not present in training on the other folds. The simimar CV error
## with and without groups shows that learning is not just assigning
## labels based on similarity within the groups but is focusing on features
## that are indicative for the class also in the CV without groups. For the
## GCV no information about group membership for the samples in the held
## out fold is present in the model. This example should show how GCV
## is performed. Because of package size limitations no specific dataset is
## available in this package where GCV is necessary.

## End(Not run)
```

---

performGridSearch *KeBABS Grid Search*

Description

Perform grid search with one or multiple sequence kernels on one or multiple SVMs with one or multiple SVM parameter sets.

Usage

```
## kbsvm(...., kernel=list(kernel1, kernel2), pkg=pkg1, svm=svm1,
##       cost=cost1, ...., cross=0, noCross=1, ....)

## kbsvm(...., kernel=kernel1, pkg=pkg1, svm=svm1,
##       cost=c(cost1, cost2), ...., cross=0, noCross=1, ....)

## kbsvm(...., kernel=kernel1, pkg=c(pkg1, pkg1, pkg1),
##       svm=c(svm1, svm2, svm3), cost=c(cost1, cost2, cost3), ....,
##       cross=0, noCross=1, ....)

## kbsvm(...., kernel=kernel1, pkg=c(pkg1, pkg2, pkg3),
##       svm=c(svm1, svm2, svm3), cost=c(cost1, cost2, cost3), ....,
##       cross=0, noCross=1, ....)

## kbsvm(...., kernel=list(kernel1, kernel2, kernel3), pkg=c(pkg1, pkg2),
##       svm=c(svm1, svm2), cost=c(cost1, cost2), ...., cross=0,
##       noCross=1, ....)

## for details see below
```

Arguments

kernel	and other parameters see kbsvm

Details

Overview

To simplify the selection of an appropriate sequence kernel (including setting of the kernel parameters), SVM implementation and setting of SVM hyperparameters KeBABS provides grid search functionality. In addition to the possibility of running the same learning tasks for different settings of the SVM hyperparameters the concept of grid search is seen here in the broader context of finding good values for all major variable parts of the learning task which includes:

- selection of the sequence kernel and standard kernel parameters: spectrum, mismatch, gappy pair or motif kernel
- selection of the kernel variant: regular, annotation-specific, position-specific or distance weighted kernel variants
- selection of the SVM implementation via package and SVM
- selection of the SVM hyperparameters for the SVM implementation

KeBABS supports the joint variation of any combination of these learning aspects together with cross validation (CV) to find the best selection based on cross validation performance. After the grid search the performance values of the different settings and the best setting of the grid search run can be retrieved from the KeBABS model with the accessor `modelSelResult`.

Grid search is started with the method `kbsvm` by passing multiple values to parameters for which in regular training only a single parameter value is used. Multiple values can be passed for the parameter `kernel` as list of kernel objects and for the parameters `pkg`, `svm` and the hyperparameters of the used SVMs as vectors (numeric or integer vector dependent on the hyperparameter). The parameter cost in the usage section above is just one representative of SVM hyperparameters that can be varied in grid search. Following types of grid search are supported (for examples see below):

- variation of one or multiple hyperparameter(s) for a given SVM implementation and one specific kernel by passing hyperparameter values as vectors
- variation of the kernel parameters of a single kernel:
  for the sequence kernels in addition to the standard kernel parameters like k for spectrum or m for gappy pair analysis can be performed in a position-independent or position-dependent manner with multiple distance weighting functions and different parameter settings for the distance weighting functions (see `positionMetadata`) or with or without annotation specific functionality (see `annotationMetadata` using one specific or multiple annotations resulting in considerable variation possibilities on the kernel side. The kernel objects for the different parameter settings of the kernel must be precreated and are passed as list to `kbsvm`. Usually each kernel has the best performance at differernt hyperparameter values. Therefore in general just varying the kernel parameters without varying the hyperparameter values does not make sense but both must be varied together as described below.
- variation of multiple SVMs from the same or different R packages with identical or different SVM hyperparameters (dependent on the formulation of the SVM objective) for one specific kernel

- combination of the previous three variants as far as runtime allows (see also runtime hints below)

For collecting performance values grid search is organized in a matrix like manner with different kernel objects representing the rows and different hyperparameter settings or SVM and hyperparameter settings as columns of the matrix. If multiple hyperparameters are used on a single SVM the same entry in all hyperparameter vectors is used as one parameter set corresponding to a single column in the grid matrix. The same applies to multiple SVMs, i.e. when multiple SVMs are used from the same package the `pkg` parameter still must have one entry for each entry in the `svm` parameter (see examples below). The best performing setting is reported dependent on the performance objective.

Instead of a single training and test cycle for each grid point cross validation should be used to get more representative results. In this case CV is executed for each parameter setting. For larger datasets or kernels with higher complexity the runtime for the full grid search should be limited through adequate selection of the parameter `cross`.

Performance measures and performance objective

The usual performance measure for grid search is the cross validation error which is stored by default for each grid point. For e.g. non-symmetrical class distribution of the dataset other performance measures can be more expressive. For such sitations also the accuracy, the balanced accuracy and the Matthews correlation coefficient can be stored for a grid point (see parameter `perfParameters` in `kbsvm`. (The accuracy corresponds fully to the CV error because it is just the inverted measure. It is included for easier comparability with the balanced accuracy). The performance values can be retrieved from the model selection result object with the accessor `performance`. The objective for selecting the best performing paramters settings is by default the CV error. With the parameter `perfObjective` in `kbsvm` one of the other above mentioned performance parameters can be chosen as objective for the best settings instead of the cross validation error.

Runtime Hints

When parameter `showCVTimes` in `kbsvm` is set to TRUE the runtime for the individual cross validation runs is shown for each grid point. In this way quick runtime estimates can be gathered through running the grid search for a reduced grid and extrapolating the runtimes to the full grid. Display of a progress indication in grid search is available with the parameter `showProgress` in `kbsvm`.

Dependent on the number of sequences, the complexity of the kernel processing, the type of chosen cross validation and the degree of variation of parameters in grid search the runtime can grow drastically. One possible strategy for reducing the runtime could be a stepwise approach searching for areas with good performance in a first coarse grid search run and then refining the areas of good performance with additional more fine grained grid searches.

The implementation of the sequence kernels was done with a strong focus on runtime performance which brings a considerable improvement compared to other implementations. In KeBABS also an interface to the very fast SVM implementations in package LiblineaR is available. Beyond these

performance improvements KeBABS also supports the generation of sparse explicit representations for every sequence kernel which can be used instead of the kernel matrix for learning. In many cases especially with a large number of samples where the kernel matrix would become too large this alternative provides additional dynamical benefits. The current implementation of grid search does not make use of multi-core infrastructures, the entire processing is done on a single core.

Value

grid search stores the results in the KeBABS model. They can be retrieved with the accessor modelSelResult{KBModel}.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

kbsvm, spectrumKernel, mismatchKernel, gappyPairKernel, motifKernel, positionMetadata, annotationMetadata, performModelSelection

Examples

```
## load transcription factor binding site data

data(TFBS)
enhancerFB
## The C-svc implementation from LiblineaR is chosen for most of the
## examples because it is the fastest SVM implementation. With SVMs from
## other packages slightly better results could be achievable.
## To get a realistic image of possible performance values, kernel behavior
## and speed of grid search together with 10-fold cross validation a
## resonable number of sequences is needed which would exceed the runtime
## restrictions for automatically executed examples. Therefore the grid
## search examples must be run manually. In these examples we use the full
## dataset for grid search.
train <- sample(1:length(enhancerFB), length(enhancerFB))

## grid search with single kernel object and multiple hyperparameter values
## create gappy pair kernel with normalization
gappyK1M3 <- gappyPairKernel(k=1, m=3)
## show details of single gappy pair kernel object
gappyK1M3

## grid search for a single kernel object and multiple values for cost
pkg <- "LiblineaR"
svm <- "C-svc"
cost <- c(0.01,0.1,1,10,100,1000)
```

```
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappyK1M3,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=3)

## show grid search results
modelSelResult(model)

## Not run:
## create the list of spectrum kernel objects with normalization and
## kernel parameters values for k from 1 to 5
specK15 <- spectrumKernel(k=1:5)
## show details of the four spectrum kernel objects
specK15

## run grid search with several kernel parameter settings for the
## spectrum kernel with a single SVM parameter setting
## ATTENTION: DO NOT USE THIS VARIANT!
## This variant does not bring comparable performance for the different
## kernel parameter settings because usually the best performing
## hyperparameter values could be quite different for different kernel
## parameter settings or between different kernels, grid search for
## multiple kernel objects should be done as shown in the next example
pkg <- "LiblineaR"
svm <- "C-svc"
cost <- 2
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=specK15,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=10)

## show grid search results
modelSelResult(model)

## grid search with multiple kernel objects and multiple values for
## hyperparameter cost
pkg <- "LiblineaR"
svm <- "C-svc"
cost <- c(0.01,0.1,1,10,50,100,150,200,500,1000)
model <- kbsvm(x=enhancerFB, sel=train, y=yFB[train], kernel=specK15,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=10,
               showProgress=TRUE)

## show grid search results
modelSelResult(model)

## grid search for a single kernel object with multiple SVMs
## from different packages
## here with display of cross validation runtimes for each grid point
## pkg, svm and cost vectors must have same length and the corresponding
## entry in each of these vectors are one SVM + SVM hyperparameter setting
pkg <- rep(c("kernlab", "e1071", "LiblineaR"),3)
svm <- rep("C-svc", 9)
cost <- rep(c(0.01,0.1,1),each=3)
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappyK1M3,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=3,
               showCVTimes=TRUE)
```

```
## show grid search results
modelSelResult(model)

## run grid search for a single kernel with multiple SVMs from same package
## here all from LiblineaR: C-SVM, L2 regularized SVM with L2 loss and
## SVM with L1 regularization and L2 loss
## attention: for different formulation of the SMV objective use different
## values for the hyperparameters even if they have the same name
pkg <- rep("LiblineaR", 9)
svm <- rep(c("C-svc","l2rl2l-svc","l1rl2l-svc"), each=3)
cost <- c(1,150,1000,1,40,100,1,40,100)
model <- kbsvm(x=enhancerFB, sel=train, y=yFB[train], kernel=gappyK1M3,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=3)

## show grid search results
modelSelResult(model)

## create the list of kernel objects for gappy pair kernel
gappyK1M15 <- gappyPairKernel(k=1, m=1:5)
## show details of kernel objects
gappyK1M15

## run grid search with progress indication with ten kernels and ten
## hyperparameter values for cost and 10 fold cross validation on full
## dataset (500 samples)
pkg <- rep("LiblineaR", 10)
svm <- rep("C-svc", 10)
cost <- c(0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000)
model <- kbsvm(x=enhancerFB, y=yFB, kernel=c(specK15, gappyK1M15),
               pkg=pkg, svm=svm, cost=cost, cross=10, explicit="yes",
               showCVTimes=TRUE, showProgress=TRUE)

## show grid search results
modelSelResult(model)

## End(Not run)
```

`performModelSelection` *KeBABS Model Selection*

Description

Perform model selection with one or multiple sequence kernels on one or multiple SVMs with one or multiple SVM parameter sets.

Usage

```
## kbsvm(...., kernel=..., pkg=..., svm=..., cost=..., ....,
##       cross=0, noCross=1, ...., nestedCross=0, noNestedCross=1, ....)
```

```
## For details see below. With parameter nestedCross > 1 model selection is
## performed, the other parameters are handled identical to grid search.
```

Arguments

nestedCross	for this and other parameters see kbsvm

Details

Overview

Model selection in KeBABS is based on nested k-fold cross validation (CV) (for details see performCrossValidation). The inner cross validation is used to determine the best parameters settings (kernel parameters and SVM parameters) and the outer cross validation to verify the performance on data that was not included in the selection of the best model. The training folds of the outer CV are used to run a grid search with the inner cross validation running for each point of the grid (see performGridSearch to find the best performing model. Once this model is selected the performance of this model on the held out fold of the outer CV is determined. Different model parameters settings could occur for different held out folds of the outer CV. This means that model selection does not deliver a performance estimate for a single best model but for the complete model selection process.

For each run of the outer CV KeBABS stores the selected parameter setting for the best performing model. The default performance objective for selecting the best parameters setting is based on minimizing the CV error on the inner CV. With the parameter perfObjective in kbsvm the balanced accuracy or the Matthews correlation coefficient can be used instead for which the parameter setting with the maximal value is selected. The parameter setting of the best performing model for each fold in the outer CV can be retrieved from the KeBABS model with the accessor modelSelResult. The performance values on the outer CV are retrieved from the model with the accessor cvResult.

Model selection is invoked through the method kbsvm through setting parameter nestedCross > 1. For the parameters kernel,pkg, svm and SVM hyperparameters the handling is identical to grid search (see performGridSearch). The parameter cost in the usage section above is just one representative of SVM hyperparameters to indicate their relevance for model selection. The complete model selection process can be repeated multiple times through setting noNestedCross to the number of desired repetitions. Nested cross validation used in model selection is dynamically more demanding than grid search. Concerning runtime please see the runtime hints for performGridSearch.

Value

model selection stores the results in the KeBABS model. They can be retrieved with the accessor modelSelResult{KBModel}. Results from the outer cross validation are extracted from the model with the accessorcvResult.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

kbsvm, performGridSearch, modelSelResult, cvResult

Examples

```
## load transcription factor binding site data
data(TFBS)
enhancerFB
## The C-svc implementation from LiblineaR is chosen for most of the
## examples because it is the fastest SVM. With SVMs from other packages
## slightly better results could be achievable. Because of the higher
## runtime needed for nested cross validation please run the examples
## below manually. All samples of the data set are used in the examples.
train <- sample(1:length(enhancerFB), length(enhancerFB))

## model selection with single kernel object and multiple
## hyperparameter values, 5 fold inner CV and 3 fold outer CV
## create gappy pair kernel with normalization
gappyK1M3 <- gappyPairKernel(k=1, m=3)
## show details of single gappy pair kernel object
gappyK1M3

pkg <- "LiblineaR"
svm <- "C-svc"
cost <- c(50,100,150,200,250,300)
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappyK1M3,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=3,
               nestedCross=2, showProgress=TRUE)

## show best parameter settings
modelSelResult(model)

## show model selection result which is the result of the outer CV
cvResult(model)
## Not run:
## repeated model selection
pkg <- "LiblineaR"
svm <- "C-svc"
cost <- c(50,100,150,200,250,300)
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappyK1M3,
               pkg=pkg, svm=svm, cost=cost, explicit="yes", cross=10,
               nestedCross=3, noNestedCross=3, showProgress=TRUE)

## show best parameter settings
```

```
modelSelResult(model)

## show model selection result which is the result of the outer CV
cvResult(model)

## plot CV result
plot(cvResult(model))

## End(Not run)
```

---

plot,PredictionProfile,missing-method
Plot Prediction Profiles, Cross Validation Result and Grid Search Performance Parameters

---

Description

Functions for visualizing prediction profiles, cross validation result and grid search performance parameters

Usage

```
## S4 method for signature PredictionProfile,missing
plot(x, sel = NULL, col = c("red",
  "blue"), standardize = TRUE, shades = NULL, legend = "default",
  legendPos = "topright", ylim = NULL, xlab = "", ylab = "weight",
  heptads = FALSE, annotate = FALSE, markOffset = TRUE, windowSize = 1,
  ...)

## S4 method for signature CrossValidationResult,missing
plot(x, col = "springgreen")

## S4 method for signature ModelSelectionResult,missing
plot(x, sel = c("ACC", "BACC",
  "MCC"))
```

Arguments

x	for the first method above a prediction profile object of class PredictionProfile containing the profiles to be plotted, for the second method a cross validation result object usually taken from the trained kebabs model object
sel	an integer vector with one or two entries to select samples of the prediction profile matrix for plotting, if this parameter is not supplied by the user the frist one or two samples are selected.
col	a character vector with one or two color names used for plotting the samples. Default=c("red", "blue").

standardize	logical. If FALSE, the profile values s_i are displayed as they are with the value $y = -b/L$ superimposed as a light gray line. If TRUE (default), the whole profile is shifted by $-b/L$ and the light gray line is displayed at y=0.
shades	vector of at least two color specifications; If not NULL, the background area above and below the base line y=-b/L are shaded in colors shades[1] and shades[2], respectively. Default=NULL
legend	a character vector with one or two character strings containing the legend/description of the profile. If empty, no legend is displayed.
legendPos	position specification for the legend(if legend is specified). Can either be a vector with coordinates or a single keyword like "topright" (see legend).
ylim	argument that allows the user to preset the y-range of the profile plot.
xlab	label of horizontal axis, empty by default.
ylab	label of vertical axis, defaults to "weight".
heptads	logical indicating whether for proteins with heptad annotation (i.e. characters a to g, usually in periodic repetition) the heptad structure should be indicated through vertical lightgray lines each heptad. Default=FALSE
annotate	logical indicating whether annotation information should be shown in the center of the plot; Default=FALSE
markOffset	logical indicating whether the start positions in the sequences according to the assigned offset elmement metadata values should be shown near the sequence characters; for the upper sequence the first position is marked by "^" below the respective character, for the lower sequence it is marked by "v" above the sequence. If no offset element metadata is assigned to the sequences the marks are suppressed. Default=TRUE
windowSize	length of sliding window. When the parameter is set to the default value 1 the contributions of each position are plotted as step function. For kernels with multiple patterns at one position (mismatch, gappy pair and motif kernel) the weight contributions of all patterns at the position are summed up. Values larger than 1 define the length of a sliding window. All contributions within the window are averaged and the resulting value is displayed at the center position of the window. For positions within half of the window size from the start and end of the sequence the averaging cannot be performed over the full window but just the remaining positions. This means that the variation of the averaged weight contributions is higher in these border regions. If an even value is specified for this parameter one is added to the parameter value. When the parameter is set to Inf (infinite) instead of averages cumulative values along the sequence are used, i.e. at each position the sum of all contributions up to this position is displayed. In this case the plot shows how the standardized or unstandardized value (see parameter standardize) of the discrimination function builds up along the sequence. Default=1
...	all other arguments are passed to the standard plot command that is called internally to display the graphics window.

Details

Plotting of Prediction Profiles

The first variant of the `plot` method mentioned in the usage section displays one or two prediction profiles as a step function with the steps connected by vertical lines. The parameter `sel` allows to select the sample(s) if the prediction profile object contains the profiles of more than two samples. The alignment of the step functions is impacted by offset metadata assigned to the sequences. When offset values are assigned one sequence if shifted horizontally to align the start position 1 pointed to by the offset value for each sequence. (see also parameter `markOffset`). If no offset metadata is available for the sequences both step functions start at their first position on the left side of the plot. The vertical plot range can be determined by the `rng` argument. If the plot is generated for one profile, the sequence is is visualized above the plot, for two sequences the first sequence is shown above, the second sequence below the plot. Matching characters at a position are shown in the same color (by default in `"black"`, the non-matching characters in the sample-specific colors (see parameter `col`). Annotation information can also be visualized along with the step function. A call with two prediction profiles should facilitate the comparison of profiles (e.g. wild type versus mutated sequence).

The baseline for the step function of a single sample represents the offset b of the model distributed equally to all sequence positions according to the following reformulation of the discriminant function

$$f(\vec{x}) = b + \sum_{i=1}^{L}(s_i(\vec{x})) = \sum_{i=1}^{L}(s_i(\vec{x}) - \frac{-b}{L})$$

For standardized plots (see parameter `standardize` this baseline value is subtracted from the weight contribution at each position. When sequences of different length are plotted together only a standardized plot gives compareable y ranges for both step functions. For sequences of equal length the visualization can be done in non-standardized or standardized form showing the lightgray horizontal baseline at positon $y = -b/L$ or at $y = 0$. If the area between the step function and the baseline lying above the baseline is larger than the area below the baseline the sample is predicted as belonging to the class assciated with positive values of the discrimination function, otherwise to the opposite class. (For multiclass problems prediction profiles can only be generated from the feature weights related to one of the classifiers in the pairwise or one-against-rest approaches leaving only two classes for the profile plot.)

When plotting to a pdf it is recommended to use a height to width ratio of around 1:(max sequence length/25), e.g. for a maximum sequence length of 500 bases or amino acids select height=10 and width=200 when opening the pdf document for plotting.

Plotting of CrossValidation Result

The second variant of `plot` method shown in the usage section displays the cross validation result as boxplot.

Plotting of Grid Performance Values

The third variant of `plot` method shown in the usage section plots grid performance data as grid with the color of each rectange corresponding to the preformance value of the grid point.

Value

see details above

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

getPredictionProfile, positionDependentKernel, mcols, spectrumKernel, mismatchKernel, gappyPairKernel, motifKernel

Examples

```
## set seed for random generator, included here only to make results
## reproducable for this example
set.seed(456)
## load transcription factor binding site data
data(TFBS)
enhancerFB
## select 70% of the samples for training and the rest for test
train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
test <- c(1:length(enhancerFB))[-train]
## create the kernel object for gappy pair kernel with normalization
gappy <- gappyPairKernel(k=1, m=3)
## show details of kernel object
gappy

## run training with explicit representation
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
               pkg="LiblineaR", svm="C-svc", cost=80, explicit="yes",
               featureWeights="yes")

## generate prediction profile for the first three test sequences
predProf <- getPredictionProfile(enhancerFB, gappy, featureWeights(model),
                                 model@b, sel=test[1:3])

## show prediction profiles
predProf

## plot prediction profile to pdf
## As sequences are usually very long select a ratio of height to width
## for the pdf which takes care of the maximum sequence length which is
## plotted. Only single or pairs of prediction profiles can be plotted.
## Plot profile for window size 1 (default) and 50. Load package Biobase
## for openPDF
## Not run:
library(Biobase)
pdf(file="PredictionProfile1_w1.pdf", height=10, width=200)
plot(predProf, sel=c(1,3))
dev.off()
openPDF("PredictionProfile1_w1.pdf")
```

```
pdf(file="PredictionProfile1_w50.pdf", height=10, width=200)
plot(predProf, sel=c(1,3), windowSize=50)
dev.off()
openPDF("PredictionProfile1_w50.pdf")
pdf(file="PredictionProfile2_w1.pdf", height=10, width=200)
plot(predProf, sel=c(2,3))
dev.off()
openPDF("PredictionProfile2_w1.pdf")
pdf(file="PredictionProfile2_w50.pdf", height=10, width=200)
plot(predProf, sel=c(2,3), windowSize=50)
dev.off()
openPDF("PredictionProfile2_w50.pdf")

## End(Not run)
```

`predict,KBModel-method`

KeBABS Prediction Methods

Description

predict response values for new biological sequences from a model trained with `kbsvm`

Usage

```
## S4 method for signature KBModel
predict(object, x, predictionType = "response",
  sel = NULL, raw = FALSE, native = FALSE, predProfiles = FALSE,
  verbose = getOption("verbose"), ...)
```

Arguments

`object`	model object of class `KBModel` created by `kbsvm`.
`x`	multiple biological sequences in the form of a `DNAStringSet`, `RNAStringSet`, `AAStringSet` (or as `BioVector`). Also a precomputed kernel matrix (see `getKernelMatrix` or a precomputed explicit representation (see `getExRep` can be used instead. The same type of input that was used for training the model should also be used for prediction. If the parameter `x` is missing the response is computed for the sequences used for SVM training.
`predictionType`	one character string of either "response", "probabilities" or "decision" which indicates the type of data returned by prediction: predicted response, class probabilities or decision values. Class probabilities can only be computed if a probability model was generated during the training (for details see parameter `probModel` in `kbsvm`). Default=`"response"`
`sel`	subset of indices into `x`. When this parameter is present the training is performed for the specified subset of samples only. Default=`integer(0)`
`raw`	when setting this boolean parameter to TRUE the prediction result is returned in raw form, i.e. in the SVM specific format. Default=FALSE

native	when setting this boolean parameter to TRUE the prediction is not preformed via feature weights in the KeBABS model but native in the SVM. Default=FALSE
predProfiles	when this boolean parameter is set to TRUE the prediction profiles are computed for the samples passed to predict. Default=FALSE
verbose	boolean value that indicates whether KeBABS should print additional messages showing the internal processing logic in a verbose manner. The default value depends on the R session verbosity option. Default=getOption("verbose")
...	additional parameters which are passed to SVM prediction transparently.

Details

Prediction for KeBABS models

For the samples passed to the predict method the response (which corresponds to the predicted label in case of classification or the predicted target value in case of regression), the decision value (which is the value of decision function separating the classes in classification) or the class probability (probability for class membership in classification) is computed for the given model of class KBModel. (see also parameter predictionType). For sequence data this includes the generation of an explicit representation or kernel matrix dependent on the processing variant that was chosen for the training of the model. When feature weights were computed during training (see parameter featureWeights in kbsvm) the response is computed entirely in KeBABS via the feature weights in the model object. The prediction performance can be evaluated with the function evaluatePrediction.

If feature weights are not available in the model then native prediction is performed via the SVM which was used for training. The parameter native enforces native prediction even when feature weights are available. Instead of sequence data also a precomputed kernel matrix or a precomputed explicit representation can be passed to predict. Prediction via feature weights is not supported for kernel variants which do not support the generation of an explicit representation, e.g. the position dependent kernel variants.

Prediction with precomputed kernel matrix

When training was performed with a precomputed kernel matrix also in prediction a precomputed kernel matrix must be passed to the predict method. In contrast to the quadratic and symmetric kernel matrix used in training the kernel matrix for prediction is rectangular and contains the similarities of test samples (rows) against support vectors (columns). support vector indices can be read from the model with the accessor SVindex. Please not that these indices refer to the sample subset used in training. An example for training and prediction via precomputed kernel matrix is shown below.

Generation of prediction profiles

The parameter predProfiles controls whether prediction profiles (for details see getPredictionProfile) are generated during the prediction process for all predicted samples. They show the contribution of the individual sequence positions to the response value. For a subset of sequences prediction profiles can also be computed independent from predicition via the function getPredictionProfile.

Value

predict.kbsvm: upon successful completion, dependent on the parameter `predictionType` the function returns either response values, decision values or probability values for class membership. When prediction profiles are also generated a list containing predictions and prediction profiles is passed back to the user.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

KBModel, evaluatePrediction, kbsvm, getPredictionProfile, PredictionProfile

Examples

```
## load transcription factor binding site data
data(TFBS)
enhancerFB
## select 70% of the samples for training and the rest for test
train <- sample(1:length(enhancerFB), length(enhancerFB) * 0.7)
test <- c(1:length(enhancerFB))[-train]
## create the kernel object for gappy pair kernel with normalization
gappy <- gappyPairKernel(k=1, m=1)
## show details of kernel object
gappy

## run training with explicit representation
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
               pkg="LiblineaR", svm="C-svc", cost=10)

## show feature weights in KeBABS model
featureWeights(model)[1:8]

## predict the test sequences
pred <- predict(model, enhancerFB[test])
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))
pred[1:10]

## output decision values instead
pred <- predict(model, enhancerFB[test], predictionType="decision")
pred[1:10]

## Not run:
## example for training and prediction via precomputed kernel matrix

## compute quadratic kernel matrix of training samples
```

```
kmtrain <- getKernelMatrix(gappy, x=enhancerFB, selx=train)

## train model with kernel matrix
model <- kbsvm(x=kmtrain, y=yFB[train], kernel=gappy,
               pkg="kernlab", svm="C-svc", cost=10)

## compute rectangular kernel matrix of test samples versus
## support vectors
kmtest <- getKernelMatrix(gappy, x=enhancerFB, y=enhancerFB,
                          selx=test, sely=train[SVindex(model)])

## predict with kernel matrix
pred <- predict(model, kmtest)
evaluatePrediction(pred, yFB[test], allLabels=unique(yFB))

## example for probability model generation during training

## compute probability model via Platt scaling during training
## and predict class membership probabilities
model <- kbsvm(x=enhancerFB[train], y=yFB[train], kernel=gappy,
               pkg="e1071", svm="C-svc", cost=10, probModel=TRUE)

## show parameters of the fitted probability model which are the parameters
## probA and probB for the fitted sigmoid function in case of classification
## and the value sigma of the fitted Laplacian in case of a regression
probabilityModel(model)

## predict class probabilities
prob <- predict(model, enhancerFB[test], predictionType="probabilities")
prob[1:10]

## End(Not run)
```

`PredictionProfile-class` *Prediction Profile Class*

Description

Prediction Profile Class

Details

This class stores prediction profiles generated for a set of biological sequences from a trained model. Prediction profiles show the relevance of individual sequence positions for the prediction result.

Slots

`sequences` sequence information for the samples with profiles

`baselines` baselines generated from the offset in the model spread to all sequence positions

`profiles` prediction profile information stored as dense matrix with the rows as samples and the columns as positions in the sample

`kernel` kernel used for training the model on which these prediction profiles are based

---

`PredictionProfileAccessors`
PredictionProfile Accessors

Description

PredictionProfile Accessors

Usage

```
## S4 method for signature PredictionProfile
sequences(object)
```

Arguments

`object`	a prediction profile object

Value

`sequences`: sequences for which profiles were generated
`profiles`: prediction profiles
`baselines`: baselines for the plot, this is the model offset distributed to all sequence positions

Accessor-like methods

`sequences`: return the sequences.

`profiles`: return the prediction profiles.

`baselines`: return the baselines.

`x[i]`: return a `PredictionProfile` object that only contains the prediction profiles selected with the subsetting parameter `i`. This parameter can be a numeric vector with indices or a character vector with sample names.

Examples

```
## create kernel object for gappy pair kernel
gappy <- gappyPairKernel(k=1,m=11, annSpec=TRUE)
## Not run:
## load data
data(CCoil)

## perform training - feature weights are computed by default
model <- kbsvm(ccseq, yCC, gappya, pkg="LiblineaR", svm="C-svc", cost=15)

## compute prediction profiles
predProf <- getPredictionProfile(ccseq, gappya,
                                 featureWeights(model),
                                 modelOffset(model))
predProf15 <- predProf[c(1,5),]
sequences(predProf15)
profiles(predProf15)
baselines(predProf15)

## End(Not run)
```

---

predictSVM *SVM Access for Training and Prediction*

Description

Functions for SVM access (used only for testing purpose)

Usage

```
predictSVM.KernelMatrix(x, model, predictionType, verbose, ...)

## S4 method for signature missing
predictSVM(x, model, predictionType, verbose, ...)

## S4 method for signature ExplicitRepresentation
predictSVM(x, model, predictionType, verbose,
  ...)

## S4 method for signature KernelMatrix
trainSVM(x, y = NULL, svmInfo, verbose, ...)

## S4 method for signature ExplicitRepresentation
trainSVM(x, y = NULL, svmInfo, verbose,
  ...)
```

Arguments

x	kernel matrix or explicit representation
model	KeBABS model
predictionType	type of prediction
verbose	controlling verbosity
...	additional arguments to be passed to the selected SVM
y	label vector
svmInfo	SVM related info

Details

These methods are exported only for test purpose and are not meant to be generally used.

Value

`trainSVM`: returns the SVM specific model

`predictSVM`: returns the prediction in native format

Examples

```
## this function is exported only for testing purpose
## use function kbsvm instead for examples see help page of kbsvm
data(TFBS)
```

---

seqKernelAsChar *Sequence Kernel*

Description

Create the kernel matrix for a kernel object

Retrieve kernel parameters from the kernel object

Usage

```
seqKernelAsChar(from)

getKernelMatrix(kernel, x, y, selx, sely)

## S4 method for signature SpectrumKernel
kernelParameters(object)

## S4 method for signature MismatchKernel
kernelParameters(object)
```

```
## S4 method for signature GappyPairKernel
kernelParameters(object)

## S4 method for signature MotifKernel
kernelParameters(object)

## S4 method for signature SymmetricPairKernel
kernelParameters(object)
```

Arguments

from	a sequence kernel object
kernel	one kernel object of class SequenceKernel or one kernlab string kernel (see stringdot
x	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)
y	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector); if this parameter is specified a rectangular kernel matrix with the samples in x as rows and the samples in y as columns is generated otherwise a square kernel matrix with samples in x as rows and columns is computed; default=NULL
selx	subset of indices into x; when this parameter is present the kernel matrix is generated for the specified subset of x only; default=NULL
sely	subset of indices into y; when this parameter is present the kernel matrix is generated for the specified subset of y only; default=NULL
object	a sequence kernel object

Details

Sequence Kernel

A sequence kernel is used for determination of similarity values between biological sequences based on patterns occuring in the sequences. The kernels in this package were specifically written for the biological domain. The corresponding term in the kernlab package is string kernel which is a domain independent implementation of the same functionality which often used in other domains, for example in text classification. For the sequence kernels in this package DNA-, RNA- or AA-acid sequences are used as input with a reduced character set compared to regular text.

In string kernels the actual position of a pattern in the sequence/text is irrelevant just the number of occurances of the pattern is important for the similarity consideration. The kernels provided in this package can be created in a position-independent or position-dependent way. Position dependent kernels are using the postion of patterns on the pair of sequences to determine the contribution of a pattern match to the similarity value. For details see help page for positionMetadata. As second method of specializing similarity consideration in a kernel is to use annotation information which is placed along the sequences. For details see annotationMetadata. Following kernels are available:

- spectrum kernel

- mismatch kernel
- gappy pair kernel
- motif kernel

These kernels are provided in a position-independent variant. For all kernels except the mismatch also the position-dependent and the annotation-specific variants of the kernel are supported. In addition the spectrum and gappy pair kernel can be created as mixture kernels with the weighted degree kernel and shifted weighted degree kernel being two specific examples of such mixture kernels. The functions described below apply for any kind of kernel in this package. Retrieving kernel paramters from the kernel object

The function 'kernelParameters' retrieves the kernel parameters and returns them as list. The function 'seqKernelAsChar' converts a sequnce kernel object into a character string.

Generation of kernel matrix

The function `getKernelMatrix` creates a kernel matrix for the specified kernel and one or two given sets of sequences. It contains similarity values between pairs of samples. If one set of sequences is used the square kernel matrix contains pairwise similarity values for this set. For two sets of sequences the similarities are calculated between these sets resulting in a rectangular kernel matrix. The kernel matrix is always created as dense matrix of the class `KernelMatrix`. Alternatively the kernel matrix can also be generated via a direct function call with the kernel object. (see examples below)

Generation of explicit representation

With the function `getExRep` an explicit representation for a specified kernel and a given set of sequences can be generated in sparse or dense form. Applying the linear kernel to the explicit representation with the function `linearKernel` also generates a dense kernel matrix.

Value

getKernelMatrix: upon successful completion, the function returns a kernel matrix of class `KernelMatrix` which contains similarity values between pairs of the biological sequences.

kernelParameters: the kernel parameters as list

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

http://www.bioinf.jku.at/software/kebabs

See Also

`as.KernelMatrix`, `KernelMatrix`, `spectrumKernel`, `mismatchKernel`, `gappyPairKernel`, `motifKernel`

Examples

```
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the motif kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                          "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
                          "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create the kernel object with the spectrum kernel
spec <- spectrumKernel(k=3, normalized=FALSE)

## generate the kernel matrix
km <- getKernelMatrix(spec, dnaseqs)
dim(km)
km[1:5,1:5]

## alternative way to generate the kernel matrix
km <- spec(dnaseqs)
km[1:5,1:5]

## generate rectangular kernel matrix
km <- getKernelMatrix(spec, x=dnaseqs, selx=1:3, y=dnaseqs, sely=4:5)
dim(km)
km[1:3,1:2]

## generate a sparse explicit representation
er <- getExRep(dnaseqs, spec)
er[1:5, 1:8]

## generate kernel matrix from explicit representation
km <- linearKernel(er)
km[1:5,1:5]
```

SequenceKernel-class *Sequence Kernel Class*

Description

Sequence Kernel Class

Details

This class represents the parent class for all sequence kernels. It is an abstract class and must not be instantiated.

Slots

.Data the kernel function is stored in this slot. It is executed when the kernel matrix is created through invoking the kernel object.

---

show.BioVector *Display Various KeBABS Objects*

Description

Display methods for BioVector, SpectrumKernel, MismatchKernel, GappyPairKernel, MotifKernel, SymmetricPairKernel, ExplicitRepresentationDense, ExplicitRepresentationSparse, PredictionProfile, CrossValidationResult, ModelSelectionResult, SVMInformation and KBModel objects

Usage

```
show.BioVector(object)

## S4 method for signature PredictionProfile
show(object)

## S4 method for signature SpectrumKernel
show(object)

## S4 method for signature MismatchKernel
show(object)

## S4 method for signature MotifKernel
show(object)

## S4 method for signature GappyPairKernel
show(object)

## S4 method for signature ExplicitRepresentationDense
show(object)

## S4 method for signature ExplicitRepresentationSparse
show(object)

## S4 method for signature CrossValidationResult
show(object)

## S4 method for signature ModelSelectionResult
show(object)

## S4 method for signature SVMInformation
show(object)
```

```
## S4 method for signature KBModel
show(object)
```

Arguments

object	object of class BioVector, PredictionProfile, SpectrumKernel, MismatchKernel, GappyPairKernel, MotifKernel, SymmetricPairKernel, ExplicitRepresentation, ExplicitRepresentationSparse, PredictionProfile, CrossValidationResult, ModelSelectionResult, SVMInformation or KBModel to be displayed

Details

show displays on overview of the selected object.

Value

show: show returns an invisible NULL

Examples

```
## load coiled coil data
data(CCoil)

## show amino acid sequences
ccseq

## define spectrum kernel object
specK1 <- spectrumKernel(k=1, normalized=FALSE)

## show kernel object
show(specK1)

## compute explicit representation for the first 5 sequences
## in dense format
er <- getExRep(ccseq, specK1, sel=1:5, sparse=FALSE)

## show dense explicit representation
show(er)
```

---

showAnnotatedSeq *Annotation Specific Kernel*

Description

Assign annotation metadata to sequences and create a kernel object which evaluates annotation information

Show biological sequence together with annotation

Usage

```
showAnnotatedSeq(x, sel = 1, ann = TRUE, pos = TRUE, start = 1,
  end = width(x)[sel], width = NA)

## S4 method for signature XStringSet
## annotationMetadata(x, annCharset= ...) <- value

## S4 method for signature BioVector
## annotationMetadata(x, annCharset= ...) <- value

## S4 replacement method for signature BioVector
annotationMetadata(x, ...) <- value

## S4 method for signature XStringSet
annotationMetadata(x)

## S4 method for signature BioVector
annotationMetadata(x)

## S4 method for signature XStringSet
annotationCharset(x)

## S4 method for signature BioVector
annotationCharset(x)
```

Arguments

x	biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)
sel	single index into x for displaying a specific sequence. Default=1
ann	show annotation information along with the sequence
pos	show position information
start	first postion to be displayed, by default the full sequence is shown
end	last position to be displayed or use parameter 'width'
width	number of positions to be displayed or use parameter 'end'
...	additional parameters which are passed transparently.
value	character vector with annotation strings with same length as the number of sequences. Each anntation string must have the same number of characters as the corresponding sequence. In addition to the characters defined in the annotation character set the character "-" can be used in the annotation strings for masking sequence parts.
annCharset	character string listing all characters used in annotation sorted ascending according to the C locale, up to 32 characters are possible

Details

Annotation information for sequences

For the annotation specific kernel additional annotation information is added to the sequence data. The annotation for one sequence consist of a character string with a single annotation character per position, i.e. the annotation sequence has the same length as the sequence. The character set used for annotation is defined user specific on XStringSet level with up to 32 different characters. Each biological sequence needs an associated annotation sequence assigned consisting of characters from this character set. The evaluation of annotation information as part of the kernel processing during generation of a kernel matrix or an explict representation can be activated per kernel object.

Assignment of annotation information

The annotation characterset consists of a character string listing all allowed annotation characters in alphabetical order. Any single byte ASCII character from the decimal range between 32 and 126, except 45, is allowed. The character '-' (ASCII dec. 45) is used for masking sequence parts which should not be evaluated. As it has assigned this special masking function it must not be used in annotation charactersets.

The annotation characterset is assigned to the sequence set with the `annotationMetadata` function (see below). It is stored in the metadata list as named element `annotationCharset` and can be stored along with other metadata assigned to the sequence set. The annotation strings for the individual sequences are represented as a character vector and can be assigned to the XStringSet together with the assignment of the annotation characterset as element related metadata. Element related metadata is stored in a DataFrame and the columns of this data frame represent the different types of metadata that can be assigned in parallel. The column name for the sequence related annotation information is "annotation". (see Example section for an example of annotation metadata assignment) Annotation metadata can be assigned together with position metadata (see `positionMetadata` to a sequence set.

Annotation Specific Kernel Processing

The annotation specific kernel variant of a kernel, e.g. the spectrum kernel appends the annotation characters corresponding to a specific kmer to this kmer and treats the resulting pattern as one feature - the basic unit for similarity determination. The full feature space of an annotation specific spectrum kernel is the cartesian product of the set of all possible sequence patterns with the set of all possible anntotions patterns. Dependent on the number of characters in the annotation character set the feature space increases drastically compared to the normal spectrum kernel. But through annotation the similarity consideration between two sequences can be split into independent parts considered separately, e.g. coding/non-coding, exon/intron, etc... . For amino acid sequences e.g. a heptad annotation (consisting of a usually periodic pattern of 7 characters (a to g) can be used as annotation like in prediction of coiled coil structures. (see reference Mahrenholz, 2011)

The flag `annSpec` passed during creation of a kernel object controls whether annotation information is evaluated by the kernel. (see functions `spectrumKernel`,`gappyPairKernel`, `motifKernel`) In this way sequences with annotation can be evaluated annotation specific and without annotation through using two different kernel objects. (see examples below) The annotation specific kernel variant is available for all kernels in this package except for the mismatch kernel.

annotationMetadata function

With this function annotation metadata can be assigned to sequences defined as XStringSet (or BioVector). The sequence annotation strings are stored as element related information and can be retrieved with the method `mcols`. The characters used for anntation are stored as annotation characterset for the sequence set and can be retrieved with the method `metadata`. For the assignment of annotation metadata to biological sequences this function should be used instead of the lower level functions metadata and mcols. The function `annotationMetadata` performs several checks and also takes care that other metadata or element metadata assigned to the object is kept. Annotation metadata are deleted if the parameters `annCharset` and `annotation` are set to NULL.

showAnnotatedSeq function

This function displays individual sequences aligned with the annotation string with 50 positions per line. The two header lines show the start postion for each bock of 10 characters.

Accessor-like methods

The method annotationMetadata<- assigns annotation metadata to a sequence set. In the assignment also the annotation characterset must be specified. Annotation characters which are not listed in the characterset are treated like invalid sequence characters. They interrupt open patterns and lead to a restart of the pattern search at this position.

Value

`annotationMetadata`: a character vector with the annotation strings

`annotationCharset`: a character vector with the annotation

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Mahrenholz, 2011) – C. Mahrenholz, I. Abfalter, U. Bodenhofer, R. Volkmer and S. Hochreiter. Complex networks govern coiled-coil oligomerization - prediction and profiling by means of a machine learning approach.

`http://www.bioinf.jku.at/software/kebabs`

See Also

`spectrumKernel`, `gappyPairKernel`, `motifKernel`, `positionMetadata`, `metadata`, `mcols`

Examples

```
## create a set of annotated DNA sequences
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
x <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                    "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                    "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
                    "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                    "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(x) <- paste("S", 1:length(x), sep="")
## define the character set used in annotation
## the masking character - is is not part of the character set
anncs <- "ei"
## annotation strings for each sequence as character vector
## in the third and fourth sample a part of the sequence is masked
annotStrings <- c("eeeeeeeeeeeeiiiiiiiiiieeeeeeeeeeeeeeeiiiiiiiii",
                  "eeeeeeeeeiiiiiiiiiiiiiiiiiieeeeeeeeeeeeeeeee",
                  "---------eeeeeeeeeeeeeeeeiiiiiiiiiiiiiiiiiii",
                  "eeeeeeeeeeeeeeeeeeeeeeeiiiiiiiiiiiiiiiiiii----",
                  "eeeeeeeeeeeeiiiiiiiiiiiiiiiiiiiiiieeeeeeeeeee")
## assign metadata to DNAString object
annotationMetadata(x, annCharset=anncs) <- annotStrings
## show annotation
annotationMetadata(x)
annotationCharset(x)

## show sequence 3 aligned with annotation string
showAnnotatedSeq(x, sel=3)

## create annotation specific spectrum kernel
speca <- spectrumKernel(k=3, annSpec=TRUE, normalized=FALSE)

## show details of kernel object
kernelParameters(speca)

## this kernel object can be now be used in a classification or regression
## task in the usual way or you can use the kernel for example to generate
## the kernel matrix for use with another learning method in another R
## package.
kma <- speca(x)
kma[1:5,1:5]
## generate a dense explicit representation for annotation-specific kernel
era <- getExRep(x, speca, sparse=FALSE)
era[1:5,1:8]

## when a standard spectrum kernel is used with annotated
## sequences the anntotation information is not evaluated
spec <- spectrumKernel(k=3, normalized=FALSE)
km <- spec(x)
km[1:5,1:5]

## finally delete annotation metadata if no longer needed
```

```
annotationMetadata(x) <- NULL
## show empty metadata
annotationMetadata(x)
annotationCharset(x)
```

`spectrumKernel` *Spectrum Kernel*

Description

Create a spectrum kernel object

Usage

```
spectrumKernel(k = 3, r = 1, annSpec = FALSE, distWeight = numeric(0),
  normalized = TRUE, exact = TRUE, ignoreLower = TRUE, presence = FALSE,
  revComplement = FALSE, mixCoef = numeric(0))

## S4 method for signature SpectrumKernel
getFeatureSpaceDimension(kernel, x)
```

Arguments

k	length of the substrings (also called kmers). This parameter defines the size of the feature space, i.e. the total number of features considered in this kernel is |A|^k, with |A| as the size of the alphabet (4 for DNA and RNA sequences and 21 for amino acid sequences). When multiple kernels with different k values should be generated e.g. for model selection a range e.g. k=3:5 can be specified. In this case a list of kernel objects with the individual k values from the range is generated as result. Default=3
r	exponent which must be > 0 (details see below). Default=1
annSpec	boolean that indicates whether sequence annotation should be taken into account (details see on help page for annotationMetadata). For the annotation specific spectrum kernel the total number of features increases to |A|^k * |a|^k with |A| as the size of the sequence alphabet and |a| as the size of the annotation alphabet. Default=FALSE
distWeight	a numeric distance weight vector or a distance weighting function (details see on help page for gaussWeight). Default=NULL
normalized	a kernel matrix or explicit representation generated with this kernel will be normalized(details see below). Default=TRUE
exact	use exact character set for the evaluation (details see below). Default=TRUE
ignoreLower	ignore lower case characters in the sequence. If the parameter is not set lower case characters are treated like uppercase. Default=TRUE
presence	if this parameter is set only the presence of a kmers will be considered, otherwise the number of occurances of the kmer is used. Default=FALSE

revComplement	if this parameter is set a kmer and its reverse complement are treated as the same feature. Default=FALSE
mixCoef	mixing coefficients for the mixture variant of the spectrum kernel. A numeric vector of length k is expected for this parameter with the unused components in the mixture set to 0. Default=numeric(0)
kernel	a sequence kernel object
x	one or multiple biological sequences in the form of a DNAStringSet, RNAStringSet, AAStringSet (or as BioVector)

Details

Creation of kernel object

The function 'spectrumKernel' creates a kernel object for the spectrum kernel. This kernel object can then be used with a set of DNA-, RNA- or AA-sequences to generate a kernel matrix or an explicit representation for this kernel. The spectrum kernel uses all subsequences for length k (also called kmers). For sequences shorter than k the self similarity (i.e. the value on the main diagonal in the square kernel matrix) is 0. The explicit representation contains only zeros for such a sample. Dependent on the learning task it might make sense to remove such sequences from the data set as they do not contribute to the model but still influence performance values.

For values different from 1 (=default value) parameter r leads to a transfomation of similarities by taking each element of the similarity matrix to the power of r. Only integer values larger than 1 should be used for r in context with SVMs requiring positive definite kernels. If normalized=TRUE, the feature vectors are scaled to the unit sphere before computing the similarity value for the kernel matrix. For two samples with the feature vectors x and y the similarity is computed as:

$$s = \frac{\vec{x}^T \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

For an explicit representation generated with the feature map of a normalized kernel the rows are normalized by dividing them through their Euclidean norm. For parameter exact=TRUE the sequence characters are interpreted according to an exact character set. If the flag is not set ambigous characters from the IUPAC characterset are also evaluated. For sequences shorter than k the self similarity (i.e. the value on the main diagonal in the square kernel matrix) is 0.

The annotation specific variant (for details see annotationMetadata) and the position dependent variants (for details see positionMetadata) either in the form of a position specific or a distance weighted kernel are supported for the spectrum kernel. The generation of an explicit representation is not possible for the position dependent variants of this kernel.

Creation of kernel matrix

The kernel matrix is created with the function getKernelMatrix or via a direct call with the kernel object as shown in the examples below.

Value

spectrumKernel: upon successful completion, the function returns a kernel object of class SpectrumKernel.

of getDimFeatureSpace: dimension of the feature space as numeric value

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Leslie, 2002) – C. Leslie, E. Eskin and W. S. Noble. The Spectrum Kernel: A String Kernel for SVM Protein Classification.

(Bodenhofer, 2009) – U. Bodenhofer, K. Schwarzbauer, M. Ionescu and S. Hochreiter. Modelling position specificity in sequence kernels by fuzzy equivalence relations.
(Mahrenholz, 2011) – C. Mahrenholz, I. Abfalter, U. Bodenhofer, R. Volkmer and S. Hochreiter. Complex networks govern coiled-coil oligomerizations - predicting and profiling by means of a machine learning approach.

http://www.bioinf.jku.at/software/kebabs

See Also

kernelParameters-method, getKernelMatrix, getExRep, mismatchKernel, motifKernel, gappyPairKernel, SpectrumKernel

Examples

```
## instead of user provided sequences in XStringSet format
## for this example a set of DNA sequences is created
## RNA- or AA-sequences can be used as well with the spectrum kernel
dnaseqs <- DNAStringSet(c("AGACTTAAGGGACCTGGTCACCACGCTCGGTGAGGGGGACGGGGTGT",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC",
                          "CAGGAATCAGCACAGGCAGGGGCACGGCATCCCAAGACATCTGGGCC",
                          "GGACATATACCCACCGTTACGTGTCATACAGGATAGTTCCACTGCCC",
                          "ATAAAGGTTGCAGACATCATGTCCTTTTTGTCCCTAATTATTTCAGC"))
names(dnaseqs) <- paste("S", 1:length(dnaseqs), sep="")

## create the kernel object for dimers without normalization
speck <- spectrumKernel(k=2, normalized=FALSE)
## show details of kernel object
speck

## generate the kernel matrix with the kernel object
km <- speck(dnaseqs)
dim(km)
km[1:5,1:5]

## alternative way to generate the kernel matrix
km <- getKernelMatrix(speck, dnaseqs)
km[1:5,1:5]

## Not run:
## plot heatmap of the kernel matrix
heatmap(km, symm=TRUE)
```

```
## End(Not run)
```

`SpectrumKernel-class` *Spectrum Kernel Class*

Description

Spectrum Kernel Class

Details

Instances of this class represent a kernel object for the spectrum kernel. The class is derived from SequenceKernel.

Slots

`k` length of the substrings considered by the kernel

`r` exponent (for details see spectrumKernel)

`annSpec` when set the kernel evaluates annotation information

`distWeight` distance weighting function or vector

`normalized` data generated with this kernel object is normalized

`exact` use exact character set for evaluation

`ignoreLower` ignore lower case characters in the sequence

`presence` consider only the presence of kmers not their counts

`revComplement` consider a kmer and its reverse complement as the same feature

`mixCoef` mixing coefficients for mixture kernel

`SVMInformation-class` *SVM Information Class*

Description

SVM Information Class

Details

Instances of this class store SVM related information.

Slots

availPackages installed SVM packages

reqSVM user requested SVM implementation

reqPackage user requested package

reqSVMPar user requested SVM parameters

reqKernel user requested kernel

reqExplicit user requested indictor of expl. rep. processing

reqExplicitType user requested expl. rep. type

reqFeatureType user requested feature type

selSVM selected SVM implementation

selPackage selected package

selSVMPar selected SVM parameters

selKernel selected kernel

selExplicit selected indictor of expl. rep. processing

explicitKernel kernel for explicit representation

featureWeights indicator for feature weights

weightLimit cutoff value for feature weights

probModel indicator for probability model

---

symmetricPairKernel *Symmetric Pair Kernel*

Description

Create a symmetric pair kernel object

Usage

```
symmetricPairKernel(siKernel, kernelType = c("mean", "TPPK"), r = 1)
```

Arguments

siKernel	kernel for single instances
kernelType	defines the type of pair kernel. It specifies in which way the similarity between two pairs of sequences are computed. Allowed values are "mean", and "TPPK" (see also details section). Default="mean"
r	exponent which must be > 0 (details see below). Default=1

Details

Creation of kernel object

The function 'symmetricPairKernel' creates a kernel object for the symmetric pair kernel. This kernel is an example for multiple instance learning and can be used for learning based on pairs of sequences. The single instance kernel passed to the symmetric pair kernel computes a similarity between two individual sequences giving a similarity for one pair of sequences. The symmetric pair kernel function gets as input two pairs of sequences and computes a similarity value between the two pairs. This similarity is computed dependent on the value of the argument `kernelType` from the similarities delivered by the single instance kernel in the following way:

`mean` (arithmetic mean):

k(<a,b>, <c,d>) = 1/4 * (k(a,c) + k(a,d) + k(b,c) + k(b,d))

`TPKK` (tensor pairwise product kernel):

k(<a,b>, <c,d>) = (k(a,c) * k(b,d) + k(a,d) * k(b,c))

Every sequence kernel available in KeBABS can be used as single instance kernel for the symmetric pair kernel allowing to create similarity measures between two pairs of sequences based on different similarity measures between individual sequences.

The row names and column names of a kernel matrix generated from a symmetric pair kernel object describe the sequence pair with the names of the individual sequences in the pair separated by the underscore character.

For values different from 1 (=default value) parameter `r` leads to a transfomation of similarities by taking each element of the similarity matrix to the power of r. Only integer values larger than 1 should be used for r in context with SVMs requiring positive definite kernels.

The symmetricPairKernel can be used in sequence based learning like any single instance kernel. Label values are defined against pairs of sequences in this case. Explicit representation, feature weights and prediction profiles are not available for the symmetric pair kernel. As kernels computed through sums and products of postive definite kernels all variants of this kernel are positive definite.

Value

symmetricPairKernel: upon successful completion, the function returns a kernel object of class `SymmetricPairKernel`.

Author(s)

Johannes Palme <kebabs@bioinf.jku.at>

References

(Hue, 2002) – M.Hue and J.-P.Vert. On learning with kernels for unordered pairs.

(Ben-Hur, 2005) – A. Ben-Hur and W.S. Noble. Kernel methods for predicting protein-protein interactions (Gaertner, 2002) – T. Gaertner, P.. Flach, A. Kowalczyk, A.J. Smola. Multi-Instance

Kernels.

http://www.bioinf.jku.at/software/kebabs

See Also

kernelParameters-method, getKernelMatrix, spectrumKernel, mismatchKernel, motifKernel, gappyPairKernel, SymmetricPairKernel

Examples

```
## load sample sequences from transcription factor binding dataset
data(TFBS)
## in this example we just use the first 30 sequences and rename samples
x <- enhancerFB[1:30]
names(x) <- paste("S", 1:length(x), sep="")

## create the single instance kernel object
specK5 <- spectrumKernel(k=5)
## show details of single instance kernel object
specK5

## create the symmetric pair kernel object for the single instance kernel
tppk <- symmetricPairKernel(siKernel=specK5, kernelType="TPPK")

## generate the kernel matrix with the symmetric pair kernel object which
## contains similarity values between two pairs of sequences.
## Hint: The kernel matrix for the single instance kernel is computed
## internally.
km <- tppk(x)
dim(km)
km[1:5,1:5]

## Not run:
## plot heatmap of the kernel matrix
heatmap(km, symm=TRUE)

## End(Not run)
```

SymmetricPairKernel-class

Symmetric Pair Kernel Class

Description

Symmetric Pair Kernel Class

Details

Instances of this class represent a kernel object for the symmetric pair kernel. The kernel does not compute similarity between single samples but between two pairs of samples based on a regular sequence kernel for single samples. The class is derived from SequenceKernel.

Slots

`siKernel` single instance kernel

`kernelType` type of pair kernel

`r` exponent (for details see gappyPairKernel)

Index