# Package 'RANKS'

December 9, 2015

**Type** Package

**Title** Ranking of Nodes with Kernelized Score Functions

**Version** 1.0

**Date** 2015-12-07

**Author**
Giorgio Valentini -- AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano

**Maintainer** Giorgio Valentini <valentini@di.unimi.it>

**Description** Implementation of Kernelized score functions and other semi-supervised learning algorithms for node label ranking in biomolecular networks. RANKS can be easily applied to a large set of different relevant problems in computational biology, ranging from automatic protein function prediction, to gene disease prioritization and drug repositioning, and more in general to any bioinformatics problem that can be formalized as a node label ranking problem in a graph. The modular nature of the implementation allows to experiment with different score functions and kernels and to easily compare the results with baseline network-based methods such as label propagation and random walk algorithms, as well as to enlarge the algorithmic scheme by adding novel user-defined score functions and kernels.

**License** GPL (>= 2)

**LazyLoad** yes

**Imports** methods, graph, RBGL, limma, NetPreProc, PerfMeas

**Suggests** bionetdata

**NeedsCompilation** yes

## R topics documented:

---

RANKS-package *Ranking of Nodes with Kernelized Score Functions*

---

## Description

Implementation of Kernelized score functions and other semi-supervised learning algorithms for node label ranking in biomolecular networks. RANKS can be easily applied to a large set of different relevant problems in computational biology, ranging from automatic protein function prediction, to gene disease prioritization and drug repositioning, and more in general to any bioinformatics problem that can be formalized as a node label ranking problem in a graph. The modular nature of the implementation allows to experiment with different score functions and kernels and to easily compare the results with baseline network-based methods such as label propagation and random walk algorithms, as well as to enlarge the algorithmic scheme by adding novel user-defined score functions and kernels.

## Details

The DESCRIPTION file:

| | |
|---|---|
| Package: | RANKS |
| Type: | Package |
| Title: | Ranking of Nodes with Kernelized Score Functions |
| Version: | 1.0 |
| Date: | 2015-12-07 |
| Author: | Giorgio Valentini – AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano |
| Maintainer: | Giorgio Valentini <valentini@di.unimi.it> |
| Description: | Implementation of Kernelized score functions and other semi-supervised learning algorithms for node label ran |
| License: | GPL (>= 2) |
| LazyLoad: | yes |

Imports: methods, graph, RBGL, limma, NetPreProc, PerfMeas
Suggests: bionetdata

Index of help topics:

| | class |
|---|---|
| rw.kernel-methods | Random walk kernel |

## Author(s)

Giorgio Valentini – AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano

Maintainer: Giorgio Valentini <valentini@di.unimi.it>

## References

Re M, Mesiti M, Valentini G: A fast ranking algorithm for predicting gene functions in biomolecular networks. IEEE ACM Trans Comput Biol Bioinform 2012, 9(6):1812-1818.

Re M, Valentini G: Cancer module genes ranking using kernelized score functions. BMC Bioinformatics 2012, 13(S14):S3.

Re M, Valentini G: Network-based drug ranking and repositioning with respect to DrugBank therapeutic categories. IEEE/ACM Trans Comput Biol Bioinform 2013, 10(6):1359-1371.

G. Valentini, A. Paccanaro, H. Caniza, A. Romero, M. Re: An extensive analysis of disease-gene associations using network integration and fast kernel-based gene prioritization methods, Artif. Intell. in Med. 61 (2) (2014) 63-78

---

`do.GBA` *GBA cross-validation experiments with multiple classes*

---

## Description

High level function to perform experiments with GBA. It perform a k fold CV repeated 1 time on a given data set

## Usage

```
do.GBA(fun = GBAsum, k = 5, filter = TRUE, seed = 1, data, labels)
```

## Arguments

| | |
|---|---|
| `fun` | function performing GBA. it can be one of the following:<br>- GBAsum: it sums the edge weights connecting a node to its positive neighbours<br>- GBAmax: it computes the maximum between the edge weights connecting a node to its positive neighbours |
| `k` | number of folds for the cross validation (def. 5) |
| `filter` | if TRUE (def) the adjacency matrix is sparsified otherwise not |
| `seed` | seed of the random generator for the generation of the folds (def: 1): |
| `data` | name of the data set to loaded (without rda extension). It must be an .rda file containing the adjiacency matrix of the graph. It assumes that it is in the "data" directory |

`labels` name of the target labels (without rda extension). It must be an .rda file containing the label matrix of the examples. Rows correspond to examples and columns to classes It assumes that it is in the "data" directory

## Details

High level function to perform cross-validation experiments with multiple classes using GBA.

It performs a k fold CV on a given data set, and output scores, AUC and Precision at a given recall results for multiple classes.

Graph data are read from a matrix representing the adjiacency matrix of the graph stored as a .rda file. The labels are read from a matrix having examples as rows and classes as columns stored as a .rda file. If M is the label matrix, then M[i,j]=1, if example i is annotated with class j, otherwise M[i,j] = 0.

Results are included in matrices representing Scores, AUC and precision at a given recall results stored as .rda files.

## Value

3 rda files stored in the "Results" directory:

`Scores results` A matrix with examples on rows and classes on columns representing the computed scores for each example and for each considered class

`AUC results` AUC results files computed through `AUC.single.over.classes` from the package PerfMeas

`Precision at given recall results`
computed through `precision.at.multiple.recall.level.over.classes` from the package PerfMeas.

The name of the Score file starts with Score, of the AUC file with AUC, and of the Precision at given recall file with PXR. Other learning parameters are appended to the name of the file. All the results .rda files are stored in the Results directory (that must exist in advance).

## See Also

`GBAmax`, `GBAsum`

## Examples

```
## Not run:
# Yeast prediction of 177 FunCat classes by 5-fold cross validation using STRING data
# data obtained from the bionetdata package from CRAN
# See the AUC and Precision/recall results in the Results directory
library(bionetdata);
if (!dir.exists("data"))
  dir.create("data");
if (!dir.exists("Results"))
  dir.create("Results");
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
save(Yeast.STRING.data, file="data/net.rda");
```

```
save(Yeast.STRING.FunCat, file="data/labels.rda");
do.GBA(data="net", labels="labels");

## End(Not run)
```

---

`do.loo.RANKS` *RANKS leave-one-out experiments with multiple classes*

---

## Description

High level function to perform RANKS leave one out (loo) experiments with mutliple classes.

## Usage

```
do.loo.RANKS(score = eav.score, compute.kernel = TRUE, kernel = rw.kernel, a = 2,
k = 19, d = 2, p = 1, sparsify = FALSE, norm = FALSE, data, labels, output.name,
net.dir = "data/", labels.dir = "data/", output.dir = "Results/")
```

## Arguments

| | |
|---|---|
| `score` | function. It must be a kernel-based score method:<br>- eav.score (default)<br>- NN.score<br>- KNN.score<br>- WSLD.score |
| `compute.kernel` | logical. If TRUE (def.) a kernel matrix is computed from data according to the choice of the function kernel, otherwise the data matrix is used as it is. |
| `kernel` | kernel method or function (def. rw.kernel) |
| `a` | kernel parameter (def. 2) |
| `k` | number of neighbours for KNN.score. It is meaningful only for kNN (def.19) |
| `d` | integer. Coefficient of linear decay for the WSLD score. It is meaningful only for the WSLD score (def.2) |
| `p` | number of steps of the RW kernel (def. 1) |
| `sparsify` | boolean. If TRUE the input matrix is sparsified using Sparsify.matrix from the package NetpreProc (def: FALSE) |
| `norm` | logical. If TRUE for each class the score is normalized in [0,1], otherwise the raw scores are maintained (default). |
| `data` | name of the network data set to be loaded (without rda extension). It must be an .rda file containing the adjiacency matrix of the graph. By default it assumes that it is in the "data" directory |
| `labels` | name of the target labels (without rda extension). It must be an .rda file containing the label matrix of the examples. By default it assumes that it is in the "data" directory |

| | |
|---|---|
| `output.name` | name of the output file (without rda extension). Other informations including the learning parameters are added in the name of the file |
| `net.dir` | relative path to the directory where the adjiacency matrix is stored (def: data) |
| `labels.dir` | relative path to directory where the label matrix is stored (def: data) |
| `output.dir` | relative path to directory where the results are stored (def: Results). Note that data and labels must have the same number of rows and in the same order. Moreover if any label column corresponds to any GO root term, this is eliminated to avoid prediction of GO root nodes. |

## Details

High level function to perform loo experiments with multiple classes using RANKS.

It performs a loo on a given data set, and scores, AUC and Precision at a given recall results for multiple classes are generated.

Graph data are read from a matrix representing the adjiacency matrix of the graph stored as a .rda file. The labels are read from a matrix having examples as rows and classes as columns stored as a .rda file. If $M$ is the label matrix, then $M[i, j] = 1$, if example $i$ is annotated with class $j$, otherwise $M[i, j] = 0$.

Results are included in matrices representing Scores, AUC and precision at a given recall results stored as .rda files.

## Value

3 rda files stored in the output.dir directory:

| | |
|---|---|
| `Scores results` | A matrix with examples on rows and classes on columns representing the computed scores for each example and for each considered class |
| `AUC results` | AUC results files computed through `AUC.single.over.classes` from the package PerfMeas |
| `Precision at given recall results` | computed through `precision.at.multiple.recall.level.over.classes` from the package PerfMeas. |

The name of the Score file starts with Score.loo, of the AUC file with AUC.loo, and of the Precision at given recall file with PXR.loo. Other learning parameters are appended to the name of the file.

## See Also

`do.RANKS`

## Examples

```
## Not run:
# Yeast prediction of 177 FunCat classes by leave-one-out using STRING data
# data obtained from the bionetdata package from CRAN.
# See the AUC and Precision/recall results in the Results directory
library(bionetdata);
if (!dir.exists("data"))
```

```
  dir.create("data");
if (!dir.exists("Results"))
  dir.create("Results");
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
save(Yeast.STRING.data, file="data/net.rda");
save(Yeast.STRING.FunCat, file="data/labels.rda");
do.loo.RANKS(data="net", labels="labels", output.name="Yeast.loo");

# another  leave-one-out prediction using KNN score and 2 steps random walk kernel
do.loo.RANKS(score = KNN.score, k=3, p=2, data="net",
             labels="labels", output.name="Yeast.loo");

## End(Not run)
```

---

`do.RANKS` *RANKS cross-validation experiments with multiple classes*

---

## Description

High level function to perform RANKS cross-validation experiments with multiple classes.

## Usage

```
do.RANKS(score = eav.score, kernel = rw.kernel, a = 2, p = 1, sparsify = TRUE, kk = 5,
rep = 1, seed = 0, data.dir = "../data/", labels.dir = "../data/",
output.dir = "../Results/", data, labels, ...)
```

## Arguments

| | |
|---|---|
| `score` | function. It must be a kernel-based score method:<br>- eav.score (default)<br>- NN.score<br>- KNN.score<br>- WSLD.score |
| `kernel` | kernel metod or function (def. rw.kernel) |
| `a` | kernel parameter (def. 2) |
| `p` | number of steps of the RW kernel (def. 1) |
| `sparsify` | boolean. If TRUE (def) the input matrix is sparsified using Sparsify.matrix from the package NetpreProc |
| `kk` | number of folds of the cross validation (def: 5) |
| `rep` | number of repetitions of the cross validation (def: 1) |
| `seed` | initialization seed for the random generator to create folds (def:0) |
| `data.dir` | relative path to directory where the adjiacency matrix is stored (def: ../data) |
| `labels.dir` | relative path to directory where the label matrix is stored (def: ../data) |

| | |
|---|---|
| `output.dir` | relative path to directory where the results are stored (def: ../Results) |
| `data` | name of the data set to loaded (without rda extension). It must be an .rda file containing the adjiacency matrix of the graph. It assumes that it is in the data.dir directory |
| `labels` | name of the target labels (without rda extension). It must be an .rda file containing the label matrix of the examples. It assumes that it is in the labels.dir directory. Note that data and labels must have the same number of rows and in the same order |
| `...` | optional arguments to be passed to the function `multiple.ker.score.cv` that performs the CV |

## Details

High level function to perform cross-validation experiments with multiple classes using RANKS.

It performs a k fold CV repeated multiple times on a given data set, and scores, AUC and Precision at a given recall results for multiple classes are generated.

Graph data are read from a matrix representing the adjiacency matrix of the graph stored as a .rda file. The labels are read from a matrix having examples as rows and classes as columns stored as a .rda file. If $M$ is the label matrix, then $M[i, j] = 1$, if example $i$ is annotated with class $j$, otherwise $M[i, j] = 0$.

Results are included in matrices representing Scores, AUC and precision at a given recall results stored as .rda files.

## Value

3 rda files stored in the output.dir directory:

| | |
|---|---|
| `Scores results` | A matrix with examples on rows and classes on columns representing the computed scores for each example and for each considered class |
| `AUC results` | AUC results files computed through `AUC.single.over.classes` from the package PerfMeas |
| `Precision at given recall results` | computed through `precision.at.multiple.recall.level.over.classes` from the package PerfMeas. |

The name of the Score file starts with Score, of the AUC file with AUC, and of the Precision at given recall file with PXR. Other learning parameters are appended to the name of the file.

## See Also

`multiple.ker.score.cv`, `do.loo.RANKS`

## Examples

```
## Not run:
# Yeast prediction of 177 FunCat classes by 5-fold cross validation using STRING data
# data obtained from the bionetdata package from CRAN
# See the AUC and Precision/recall results in the Results directory
```

```
library(bionetdata);
if (!dir.exists("data"))
  dir.create("data");
if (!dir.exists("Results"))
  dir.create("Results");
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
save(Yeast.STRING.data, file="data/net.rda");
save(Yeast.STRING.FunCat, file="data/labels.rda");
do.RANKS(data.dir = "data/", labels.dir = "data/", output.dir = "Results/",
         data="net", labels="labels")

## End(Not run)
```

---

`do.RW` *Random walk cross-validation experiments with multiple classes*

---

## Description

High level function to perform random walk cross-validation experiments with multiple classes.

## Usage

```
do.RW(tmax = 1000, eps = 1e-10, k = 5, filter = TRUE, seed = 1, data, labels)
```

## Arguments

| | |
|---|---|
| `tmax` | maximum number of iterations (def: 1000) |
| `eps` | maximum allowed difference between the computed probabilities at the steady state (def. 1e-10) |
| `k` | number of folds for the cross validation (def. 5) |
| `filter` | if TRUE (def) the adjacency matrix is sparsified otherwise not |
| `seed` | seed of the random generator for the generation of the folds (def: 1): |
| `data` | name of the data set to loaded (without rda extension). It must be an .rda file containing the adjacency matrix of the graph. It assumes that it is in the "data" directory |
| `labels` | name of the target labels (without rda extension). It must be an .rda file containing the label matrix of the examples. It assumes that it is in the "data" directory |

## Details

High level function to perform cross-validation experiments with multiple classes using RW.

It performs a k fold CV on a given data set, and output scores, AUC and Precision at a given recall results for multiple classes.

Graph data are read from a matrix representing the adjiacency matrix of the graph stored as a .rda file. The labels are read from a matrix having examples as rows and classes as columns stored as a

.rda file. If $M$ is the label matrix, then $M[i, j] = 1$, if example $i$ is annotated with class $j$, otherwise $M[i, j] = 0$.

Results are included in matrices representing Scores, AUC and precision at a given recall results stored as .rda files.

## Value

3 rda files stored in the Results directory:

`Scores results` A matrix with examples on rows and classes on columns representing the computed scores for each example and for each considered class

`AUC results` AUC results files computed through `AUC.single.over.classes` from the package PerfMeas

`Precision at given recall results`
computed through `precision.at.multiple.recall.level.over.classes` from the package PerfMeas.

The name of the Score file starts with Score, of the AUC file with AUC, and of the Precision at given recall file with PXR. Other learning parameters are appended to the name of the file. All the results .rda files are stored in the Results directory (that must exist in advance).

## See Also

`RW`, `multiple.RW.cv`, `do.RWR`

## Examples

```
## Not run:
# Yeast prediction of 177 FunCat classes by 5-fold cross validation
# using 3 steps of Random walk and STRING data.
# data obtained from the bionetdata package from CRAN
# See the AUC and Precision/recall results in the Results directory
library(bionetdata);
if (!dir.exists("data"))
  dir.create("data");
if (!dir.exists("Results"))
  dir.create("Results");
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
save(Yeast.STRING.data, file="data/net.rda");
save(Yeast.STRING.FunCat, file="data/labels.rda");
do.RW(tmax = 3, filter = FALSE, seed = 1, data="net", labels="labels");

## End(Not run)
```

do.RWR — *Random walk with restart cross-validation experiments with multiple classes*

## Description

High level function to perform random walk with restart cross-validation experiments with multiple classes.

## Usage

```
do.RWR(gamma = 0.6, tmax = 1000, eps = 1e-10, k = 5, filter = TRUE,
       seed = 1, data, labels)
```

## Arguments

| | |
|---|---|
| gamma | restart parameter (def: 0.6) |
| tmax | maximum number of iterations (def: 1000) |
| eps | maximum allowed difference between the computed probabilities at the steady state (def. 1e-10) |
| k | number of folds for the cross validation (def. 5) |
| filter | if TRUE (def) the adjacency matrix is sparsified otherwise not |
| seed | seed of the random generator for the generation of the folds (def: 1) |
| data | name of the data set to loaded (without rda extension). It must be an .rda file containing the adjiacency matrix of the graph. It assumes that it is in the "data" directory |
| labels | name of the target labels (without rda extension). It must be an .rda file containing the label matrix of the examples. It assumes that it is in the "data" directory |

## Details

High level function to perform cross-validation experiments with multiple classes using RWR.

It performs a k fold CV on a given data set, and output scores, AUC and Precision at a given recall results for multiple classes.

Graph data are read from a matrix representing the adjiacency matrix of the graph stored as a .rda file. The labels are read from a matrix having examples as rows and classes as columns stored as a .rda file. If $M$ is the label matrix, then $M[i, j] = 1$, if example $i$ is annotated with class $j$, otherwise $M[i, j] = 0$.

Results are included in matrices representing Scores, AUC and precision at a given recall results stored as .rda files.

## Value

3 rda files stored in the output.dir directory:

`Scores results` A matrix with examples on rows and classes on columns representing the computed scores for each example and for each considered class

`AUC results` AUC results files computed through `AUC.single.over.classes` from the package PerfMeas

`Precision at given recall results` computed through `precision.at.multiple.recall.level.over.classes` from the package PerfMeas.

The name of the Score file starts with Score, of the AUC file with AUC, and of the Precision at given recall file with PXR. Other learning parameters are appended to the name of the file. All the results .rda files are stored in the Results directory (that must exist in advance).

## See Also

`RWR`, `multiple.RW.cv`, `do.RW`

## Examples

```
## Not run:
# Yeast prediction of 177 FunCat classes by 5-fold cross validation
# using 3 steps of Random walk with restart and STRING data.
# data obtained from the bionetdata package from CRAN
# See the AUC and Precision/recall results in the Results directory
library(bionetdata);
if (!dir.exists("data"))
  dir.create("data");
if (!dir.exists("Results"))
  dir.create("Results");
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
save(Yeast.STRING.data, file="data/net.rda");
save(Yeast.STRING.FunCat, file="data/labels.rda");
do.RWR(tmax = 3, k = 5, filter = FALSE, seed = 1, data="net", labels="labels");
# the same experiment, but the iterations are repeated till to convergence
# (this can require a quite long time ...)
do.RWR(tmax = 1000, k = 5, eps = 1e-5, filter = FALSE, seed = 1, data="net", labels="labels");

## End(Not run)
```

---

`find.optimal.thresh.cv`

*Function to find the optimal RANKS score thereshold*

---

## Description

Function to find the optimal quantile alpha and corresponding threshold by cross-validation with a kernel-based score method.

## Usage

```
find.optimal.thresh.cv(K, ind.pos, ind.non.pos, m = 5,
alpha = seq(from = 0.05, to = 0.6, by = 0.05), init.seed = NULL,
opt.fun = compute.F, fun = KNN.score, ...)
```

## Arguments

| | |
|---|---|
| K | matrix. Kernel matrix or any valid symmetric matrix |
| ind.pos | indices of the positive examples. They are the indices the row of RW corresponding to positive examples of the training set. |
| ind.non.pos | indices of the non positive examples. They are the indices the row of RW corresponding to non positive examples of the training set. |
| m | number of folds (default: 5) |
| alpha | vector of the quantiles to be tested |
| init.seed | initial seed for the random generator. If NULL (def) no initialization is performed |
| opt.fun | Function implementing the metric to select the optimal threshold. The F-score (compute.F) is the default. Available functions:<br>- compute.F: F-score (default)<br>- compute.acc:accuracy.<br>Any function having two arguments representing the vector of predicted and true labels can be in principle used. |
| fun | function. It must be a kernel-based score method (default KNN.score) |
| ... | optional arguments for the function fun |

## Details

Function to find the optimal quantile alpha and corresponding threshold by cross-validation with a kernel-based score method. The optimality is computed with respect to a specific metric (def: F-score). This function is used by `multiple.ker.score.thresh.cv`, `ker.score.classifier.holdout`, `ker.score.classifier.cv`.

## Value

A list with 3 elements:

| | |
|---|---|
| alpha | quantile corresponding to the best F-score |
| thresh | threshold corresponding to the best F-score |
| pos.scores | scores of the positive elements computed through CV |

## See Also

`multiple.ker.score.thresh.cv`, `Kernel functions`, `ker.score.classifier.holdout`

## Examples

```
# Finding the optimal threshold in the Tanimoto chemical structure similarity network
# between 1253 DrugBank drugs for the prediction of the DrugBank category Penicillins using
# the KNN-score with the random walk kernel
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
K <- rw.kernel(DD.chem.data);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
ind.non.pos <- which(labels==0);
res <- find.optimal.thresh.cv(K, ind.pos, ind.non.pos);
res
```

---

GBAmax *Guilt By Association (GBA) using the maximum rule*

---

## Description

GBAmax implements a Guilt By Association (GBA) method based on the maximum of incident edge weights

## Usage

```
GBAmax(W, ind.positives)
```

## Arguments

| | |
|---|---|
| W | numeric matrix representing the adjacency matrix of the graph |
| ind.positives | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples. |

## Details

GBAmax implements a Guilt By Association (GBA) method for label ranking based on the maximum between the edge weights connecting a node to its positive neighbours

## Value

a list with one element:

| | |
|---|---|
| p | score associated to each node |

## References

Oliver, S., Guilt-by-association goes global, Nature, 403, pp. 601-603, 2000.

## See Also

GBAsum

## Examples

```
# Application of GBAmax to the prediction of the DrugBank category Penicillins
# using the Tanimoto chemical structure similarity network
# between 1253 DrugBank drugs
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
GBAmax(DD.chem.data, ind.pos);
# Application of GBAmax to the prediction of the DrugBank category "Anti_HIV_Agents"
labels <- DrugBank.Cat[,"Anti_HIV_Agents"];
ind.pos <- which(labels==1);
GBAmax(DD.chem.data, ind.pos);
```

---

# GBAsum *Guilt By Association (GBA) using the sum rule*

---

## Description

GBAsum implements a Guilt By Association (GBA) method based on the sum of incident edge weights

## Usage

```
GBAsum(W, ind.positives)
```

## Arguments

| | |
|---|---|
| W | numeric matrix representing the adjacency matrix of the graph |
| ind.positives | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples. |

## Details

Function that implements a Guilt By Association (GBA) method for label ranking based on the sum of edge weights connecting a node to its positive neighbours.

## Value

a list with one element:

| | |
|---|---|
| p | score associated to each node |

## References

Oliver, S., Guilt-by-association goes global, Nature, 403, pp. 601-603, 2000.

## See Also

GBAmax

## Examples

```
# Application of GBAsum to the prediction of the DrugBank category Penicillins
# using the Tanimoto chemical structure similarity network
# between 1253 DrugBank drugs
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
GBAsum(DD.chem.data, ind.pos);
# Application of GBAsum to the prediction of the DrugBank category "Anti_HIV_Agents"
labels <- DrugBank.Cat[,"Anti_HIV_Agents"];
ind.pos <- which(labels==1);
GBAsum(DD.chem.data, ind.pos);
```

---

# ker.score.classifier.cv

*Multiple cross-validation with RANKS for classification*

---

## Description

Function to classify labels according to an external cross-validation procedure with a kernel-based score method.

## Usage

```
ker.score.classifier.cv(K, ind.pos, m = 5, p = 100,
alpha = seq(from = 0.05, to = 0.6, by = 0.05), init.seed = 0,
opt.fun = compute.F, fun = KNN.score, ...)
```

## Arguments

| | |
|---|---|
| K | matrix. Kernel matrix or any valid symmetric matrix |
| ind.pos | indices of the positive examples. They are the row indices of RW corresponding to positive examples. |
| m | number of folds for each cross-validation |
| p | number of repeated cross-validations |
| alpha | vector of the quantiles to be tested |

| | |
|---|---|
| init.seed | initial seed for the random generator (def: 0) |
| opt.fun | : function. Function implementing the metric to choice the optimal threshold. The F-score (compute.F) is the default. Available functions:<br>- compute.F: F-score (default)<br>- compute.acc: accuracy.<br>Any function having two arguments representing the vector of predicted and true labels can be in principle used. |
| fun | function. It must be a kernel-based score method (default KNN.score) |
| ... | optional arguments for the function fun |

## Details

Function to classify labels according to an external cross-validation procedure with a kernel-based score method. The optimal threshold for a given class id found by internal cross-validation. Scores are computed by averaging across (possibly) multiple external cross-validations. The optimal quantile and corresponding threshold are selected by internal cross-validation using the F-score (default) or the accuracy as metric.

## Value

A list with 4 components:

| | |
|---|---|
| labels | vector of the predicted labels (1 represents positive, 0 negative) |
| av.scores | a vector with the average scores across multiple cross-validations. Elements of the vector av.scores correspond to the rows of RW |
| opt.alpha | the optimal quantile alpha |
| opt.thresh | the optimal threshold |

## See Also

rw.kernel-methods, Kernel functions, ker.score.classifier.holdout

## Examples

```
# Nodel label classification of the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# using 5 fold cross-validation repeated 3 times
# and NN-score with 1-step random walk kernel
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
K <- rw.kernel(DD.chem.data);
res <- ker.score.classifier.cv(K, ind.pos, m = 5, p = 3, fun = NN.score);
```

`ker.score.classifier.holdout` *RANKS held-out procedure for a single class*

## Description

Function to perform an held-out procedure for a single class with a kernel-based score method

## Usage

```
ker.score.classifier.holdout(K, ind.pos, ind.test, m = 5, p = 10,
alpha = seq(from = 0.05, to = 0.6, by = 0.05), init.seed = 0,
opt.fun = compute.F, fun = KNN.score, ...)
```

## Arguments

| | |
|---|---|
| `K` | matrix. Kernel matrix or any valid symmetric matrix |
| `ind.pos` | indices of the positive examples of the training set. They are the indices the row of RW corresponding to positive examples of the training set |
| `ind.test` | indices of the examples of the test set. They are the indices the row of RW corresponding to examples of the test set |
| `m` | number of folds for the cross-validation on the training set |
| `p` | number of repeated cross-validations on the training set |
| `alpha` | vector of the quantiles to be tested |
| `init.seed` | nitial seed for the random generator (def: 0) |
| `opt.fun` | Function implementing the metric to select the optimal threshold. The F-score (compute.F) is the default. Available functions:<br>- compute.F: F-score (default)<br>- compute.acc:accuracy.<br>Any function having two arguments representing the vector of predicted and true labels can be in principle used. |
| `fun` | function. It must be a kernel-based score method (default KNN.score) |
| `...` | optional arguments for the function fun |

## Details

Function to classify labels according to an hold-out procedure with a kernel-based score method. The optimal threshold for a given class is obtained by (possibly multiple) internal cross-validation on the training set. Scores of the held-out nodes are computed. Thresholds are computed on the training set by cross-validation and then are used to classify the held-out nodes in the test set. The optimal quantile and corresponding threshold are selected by internal cross-validation using the F-score as metrics. Note the test examples are given as indices of the rows of the input matrix RW.

## Value

a list with four components: A list with 4 components:

| | |
|---|---|
| `labels` | vector of the predicted labels for the test set(1 represent positive, 0 negative) |
| `av.scores` | a vector with the scores computed on the test set. Elements of the vector av.scores correspond to ind.test rows of RW |
| `opt.alpha` | the optimal quantile alpha |
| `opt.thresh` | the optimal threshold |

## See Also

`rw.kernel-methods`, `Kernel functions`, `ker.score.classifier.cv`

## Examples

```
# Node label classification of the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# with eav-score with 1-step random walk kernel
# using held-out with 5-fold CV repeated 10 times on the training set
# to set the "optimal" threshold for classifiaction
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.test <- 1:300;
ind.train <- 301:length(labels);
ind.pos <- which(labels==1);
ind.pos <- ind.pos[ind.pos>300];
K <- rw.kernel(DD.chem.data);
res <- ker.score.classifier.holdout(K, ind.pos, ind.test, m = 5, p = 10, fun = eav.score);
```

---

# ker.score.cv *RANKS cross-validation for a single class*

---

## Description

Function to perform cross-validation for a single class with a kernel-based score method

## Usage

```
ker.score.cv(RW, ind.pos, m = 5, init.seed = NULL, fun = KNN.score, ...)
```

### Arguments

| | |
|---|---|
| RW | matrix. It can be a kernel matrix or the adjacency matrix of a graph |
| ind.pos | indices of the positive examples. They are the row indices of RW corresponding to positive examples. |
| m | number of folds (def: 5) |
| init.seed | initial seed for the random generator to generate folds. If NULL (default) no initialization is performed |
| fun | function. It must be a kernel-based score method (default KNN.score) |
| ... | optional arguments for the function fun |

### Details

It performs a cross-validation using RANKS to predict the cross-validated scores. The cross-validation is stratified: the folds are constructed separately for each class, to maintain an equal ratio between classes among folds.

### Value

a numeric vector with the scores computed for each example

### See Also

multiple.ker.score.cv, multiple.ker.score.thresh.cv, rw.kernel-methods, Kernel functions.

### Examples

```
# Nodel label ranking of the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# using 5 fold cross-validation
# and eav-score with 1-step random walk kernel
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
K <- rw.kernel(DD.chem.data);
res <- ker.score.cv(K, ind.pos, m = 5, init.seed = NULL, fun = eav.score);
```

---

## kernel.functions *Kernel functions*

---

### Description

Compute similarities between feature vectors according to a specific kernel function

## Usage

```
cauchy.kernel(W, sigma = 1)
laplacian.kernel(W, sigma = 1)
gaussian.kernel(W, sigma = 1)
inv.multiquadric.kernel(W, v = 1)
identity.kernel(W, a = 1)
linear.kernel(W, a = 1)
poly.kernel(W, degree = 2, scale = -1, v = 0)
```

## Arguments

| | |
|---|---|
| W | a numeric matrix, Rows are examples and columns are features |
| sigma | a real value representing the sigma parameter (def. 1) of the Cauchy, Gaussian and Laplacian kernel |
| v | constant factor (def. 1) of the inverse multiquadric kernel and of the polynomail kernel; for the inverse multiquadric kernel v must be larger than 0. |
| a | unused parameter, maintained for compatibility reasons . |
| degree | integer corresponding to a degree of the polynomial (def. 2) |
| scale | double: scaling factor of the polynomial kernel. If $scale = -1$ (def) scale is set to $1/ncol(W)$; |

## Details

All the kernel matrices are computed by calling C code to speed-up the computation.

`cauchy.kernel` computes the Cauchy kernel.

`laplacian.kernel` computes the Lapalacian kernel.

`gaussian.kernel` computes the Gaussian kernel.

`inv.multiquadric.kernel` computes the inverse multiquadric kernel.

`identity.kernel` computes the identity kernel. In this case the input W represents a similarity square matrix (obtained i.e. through the Pearson correlation) between examples.

`linear.kernel` computes the linear kernel.

## Value

A kernel matrix representing the similarities between the examples (rows of W), according to a specific kernel function.

## See Also

`rw.kernel-methods`

## Examples

```
# computing kernels on the Tanimoto chemical structure similarity matrix
library(bionetdata);
data(DD.chem.data);
K <- identity.kernel(DD.chem.data);
K <- linear.kernel(DD.chem.data);
## Not run:
K <- gaussian.kernel(DD.chem.data);
K <- inv.multiquadric.kernel(DD.chem.data);
K <- poly.kernel(DD.chem.data);

## End(Not run)
```

---

`label.prop` *Label propagation*

---

## Description

Function that implements the Label propagation algorithm of Zhu and Ghahramani

## Usage

```
label.prop(W, ind.positives, tmax = 1000, eps = 1e-05, norm = TRUE)
```

## Arguments

| | |
|---|---|
| `W` | a numeric matrix representing the adjacency matrix of the graph |
| `ind.positives` | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples |
| `tmax` | maximum number of iterations (def: 1000) |
| `eps` | numeric. Maximum allowed difference between the computed probabilities at the steady state (def. 1e-5) |
| `norm` | boolean. If TRUE (def) the adjacency matrix $W$ of the graph is normalized to $M = D^{-1} * W$, otherwise it is assumed that the matrix $W$ is just normalized |

## Details

label.prop implements the label propagation algorithm on a given graph by performing 1 or more steps on the graph, depending on the value of the tmax parameter. It stops also if the difference of the norm of the scores between two consecutive steps is less than eps.

## Value

A list with three elements:

| | |
|---|---|
| `p` | numeric vector. Scores of each node at the steady state or after tmax iterations |
| `ind.positives` | indices of the "core" positive examples of the graph (it is equal to the same input parameter) |
| `n.iter` | number of performed steps/iterations |

## References

Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In: Proc. of the Twentieth International Conference on Machine Learning, Washington DC (2003) 912-919

## Examples

```
# Application of label prop algorithm to the prediction of the DrugBank category Penicillins
# using the Tanimoto chemical structure similarity network
# between 1253 DrugBank drugs
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
label.prop(DD.chem.data, ind.pos, tmax = 10, eps = 1e-05, norm = TRUE);
```

---

# multiple.ker.score.cv *RANKS multiple cross-validation for a single class*

---

## Description

Function to execute multiple cross-validation with RANKS for a single class.

## Usage

```
multiple.ker.score.cv(RW, ind.pos, m = 5, p = 100, init.seed = 0, fun = KNN.score, ...)
```

## Arguments

| | |
|---|---|
| RW | matrix. Kernel matrix or any valid symmetric matrix |
| ind.pos | indices of the positive examples. They are the row indices of RW corresponding to positive examples. |
| m | number of folds for each cross-validation |
| p | number of repeated cross-validations |
| init.seed | initial seed for the random generator (def: 0) |
| fun | function. It must be a kernel-based score method (default KNN.score) |
| ... | optional arguments for the function fun |

## Details

It performs multiple cross-validation using RANKS to predict the cross-validated scores. The cross-validation is stratified: the folds are constructed separately for each class, to maintain an equal ratio between classes among folds. It computes the scores by averaging across multiple cross validations.

## Value

A list with two components:

| | |
|---|---|
| av.scores | a vector with the average scores across multiple cross-validations. Elements of the vector av.scores correspond to the rows of RW |
| pos.scores | a vector with the scores of positive elements collected at each iteration |

## See Also

ker.score.cv, multiple.ker.score.thresh.cv, rw.kernel-methods, Kernel functions.

## Examples

```
# Nodel label ranking for the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# using 5 fold cross-validation repeated 10 times
# and eav-score with 1-step random walk kernel
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
K <- rw.kernel(DD.chem.data);
res <- multiple.ker.score.cv(K, ind.pos, m = 5, p = 10, init.seed = 0, fun = eav.score);
# the same but using the NN-score
res <- multiple.ker.score.cv(K, ind.pos, m = 5, p = 10, init.seed = 0, fun = NN.score);
```

---

# multiple.ker.score.thresh.cv

*Function for RANKS multiple cross-validation and optimal threshold finding for a single class*

---

## Description

Function to execute multiple cross-validation and to find the optimal threshold with RANKS for a single class.

## Usage

```
multiple.ker.score.thresh.cv(K, ind.pos, m = 5, p = 100,
alpha = seq(from = 0.05, to = 0.6, by = 0.05),
init.seed = 0, fun = KNN.score, ...)
```

## Arguments

| | |
|---|---|
| K | matrix. Kernel matrix or any valid symmetric matrix |
| ind.pos | indices of the positive examples. They are the row indices of RW corresponding to positive examples. |
| m | number of folds for each cross-validation |
| p | number of repeated cross-validations |
| alpha | vector of the quantiles to be tested |
| init.seed | initial seed for the random generator (def: 0) |
| fun | function. It must be a kernel-based score method (default KNN.score) |
| ... | optional arguments for the function fun |

## Details

Function to execute multiple cross-validation with a kernel-based score method and to find the optimal threshold for a given class by internal cross-validation.

Scores are computed by averaging across multiple external cross-validations. The optimal quantile and corresponding threshold are selected by internal cross-validation using a specific metric (def: F-score).

## Value

A list with three components:

| | |
|---|---|
| av.scores | a vector with the average scores across multiple cross-validations. Elements of the vector av.scores correspond to the rows of RW |
| opt.alpha | the optimal quantile alpha |
| opt.thresh | the optimal threshold |

## See Also

ker.score.cv, multiple.ker.score.cv, rw.kernel-methods, Kernel functions.

## Examples

```
# Node label ranking and best threshold search for the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# using 5 fold cross-validation repeated 2 times
# and eav-score with 1-step random walk kernel
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
K <- rw.kernel(DD.chem.data);
res <- multiple.ker.score.thresh.cv (K, ind.pos, m = 5, p = 2, init.seed = 0, fun = KNN.score);
```

`multiple.RW.cv` *Random walk, GBA and labelprop multiple cross-validation for a single class*

## Description

Function to execute multiple cross-validation with random walk based, labelprop and GBA methods

## Usage

```
multiple.RW.cv(W, ind.pos, k = 5, p = 100, init.seed = 0, fun = RW, ...)
```

## Arguments

| | |
|---|---|
| `W` | a numeric matrix representing the adjacency matrix of the graph. Note that if the optional argument norm=TRUE (def.), the W matrix is normalized, otherwise it is assumed that W is just normalized |
| `ind.pos` | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples |
| `k` | number of folds (def: 5) |
| `p` | number of repeated cross-validations |
| `init.seed` | initial seed for the random generator. If 0 (default) no initialization is performed |
| `fun` | function. It must be one of the following functions:<br>- RW (default)<br>- RWR<br>- label.prop<br>- GBAsum<br>- GBAmax |
| `...` | optional arguments for the function fun:<br>- gamma : restart parameter (def: 0.6) (meaningful only for RWR)<br>- tmax : maximum number of iterations (def: 1000)<br>- eps : maximum allowed difference between the computed probabilities at the steady state (def. 1e-10) |

## Details

Function to execute multiple cross-validation with random walk based, labelprop and GBA methods for a single class. It computes the scores by averaging across multiple cross validations. It can be used with of the following methods: RW, RWR, label.prop, GBAsum, GBAmax.

## Value

a vector with the the probabilities for each example at the steady state averaged across multiple cross-validations

## See Also

RW, RWR, label.prop, GBAsum, GBAmax, RW.cv

## Examples

```
# Nodel label ranking of the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# using 5 fold cross-validation repeated 2 times
# and "vanilla" 2-step random walk
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);

res <- multiple.RW.cv(DD.chem.data, ind.pos, k = 5, p = 2, init.seed = 0, fun = GBAmax)

## Not run:
# the same but using the label.prop
res <- multiple.RW.cv(DD.chem.data, ind.pos, k = 5, p = 2, init.seed = 0, fun = label.prop, tmax=2)

# the same but using "vanilla" 2-step random walk
res <- multiple.RW.cv(DD.chem.data, ind.pos, k = 5, p = 2, init.seed = 0, fun = RW, tmax=2)

## End(Not run)
```

---

RW *Random walk on a graph*

---

## Description

The function performs a random Walk on a given graph.

## Usage

```
RW(W, ind.positives, tmax = 1000, eps = 1e-10, norm = TRUE)
```

## Arguments

| | |
|---|---|
| W | a numeric matrix representing the adjacency matrix of the graph |
| ind.positives | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples |
| tmax | maximum number of iterations (steps) (def: 1000) |
| eps | maximum allowed difference between the computed probabilities at the steady state (def. 1e-10) |
| norm | if TRUE (def) the adjacency matrix $W$ of the graph is normalized to $M = D^{-1} * W$, otherwise it is assumed that the matrix $W$ is just normalized |

## Details

RW performs a random Walk on a given graph by performing 1 or more steps on the graph, depending on the value of the tmax parameter. It stops also if the difference of the norm of the probabilities between two consecutive steps is less than eps.

## Value

A list with three elements:

| | |
|---|---|
| p | numeric vector. Probability of each node at the steady state or after tmax iterations |
| ind.positives | indices of the "core" positive examples of the graph (it is equal to the same input parameter) |
| n.iter | number of performed steps/iterations |

## References

L. Lovasz, Random Walks on Graphs: a Survey, Combinatorics, Paul Erdos is Eighty, vol. 2, pp. 146, 1993.

## See Also

RWR

## Examples

```
# Application of the random walk to the prediction of the DrugBank category Penicillins
# using the Tanimoto chemical structure similarity network
# between 1253 DrugBank drugs
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
# 2-step random walk
res <- RW(DD.chem.data, ind.pos, tmax = 2);
## Not run:
# 5 steps random walk
res <- RW(DD.chem.data, ind.pos, tmax = 5);

## End(Not run)
```

RW.cv *Random walk, GBA and labelprop cross-validation for a single class*

## Description

Function to execute cross-validation with random walk based, labelprop and GBA methods

## Usage

```
RW.cv(W, ind.pos, k = 5, init.seed = 0, fun = RW, ...)
```

## Arguments

| | |
|---|---|
| W | a numeric matrix representing the adjacency matrix of the graph. Note that if the optional argument norm=TRUE (def.), the W matrix is normalized, otherwise it is assumed that W is just normalized |
| ind.pos | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples |
| k | number of folds (def: 5) |
| init.seed | initial seed for the random generator. If 0 (default) no initialization is performed |
| fun | function. It must be one of the following functions:<br>- RW (default)<br>- RWR<br>- label.prop<br>- GBAsum<br>- GBAmax |
| ... | optional arguments for the function fun:<br>- gamma : restart parameter (def: 0.6) (meaningful only for RWR)<br>- tmax : maximum number of iterations (def: 1000)<br>- eps : maximum allowed difference between the computed probabilities at the steady state (def. 1e-10) |

## Details

It performs a single cross-validation for a single class. It can be used with of the following methods: RW, RWR, label.prop, GBAsum, GBAmax.

## Value

a vector with the the probabilities for each example at the steady state

## See Also

RW, RWR, label.prop, GBAsum, GBAmax, multiple.RW.cv

## Examples

```
# Nodel label ranking of the DrugBank category Penicillins
# on the Tanimoto chemical structure similarity network (1253 drugs)
# using 5 fold cross-validation and GBAsum
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
res <- RW.cv(DD.chem.data, ind.pos, k = 5, init.seed = 0, fun = GBAsum);

## Not run:
# the same but using label.prop
res <- RW.cv(DD.chem.data, ind.pos, k = 5, init.seed = 0, fun = label.prop, tmax=2);

# the same but using "vanilla" 2-step random walk
res <- RW.cv(DD.chem.data, ind.pos, k = 5, init.seed = 0, fun = RW, tmax=2);

## End(Not run)
```

---

# rw.kernel-methods *Random walk kernel*

---

## Description

Methods to compute the random walk kernel (Smola and Kondor, 2003)

## Usage

```
## S4 method for signature 'matrix'
rw.kernel(W, a = 2)
## S4 method for signature 'graph'
rw.kernel(W, a = 2)
## S4 method for signature 'graph'
p.step.rw.kernel(RW, p = 2)
## S4 method for signature 'matrix'
p.step.rw.kernel(RW, p = 2)
```

## Arguments

| | |
|---|---|
| W | a square symmetric matrix with positive values or an object of the class graphAM or graphNEL of the package graph |
| RW | matrix. It must be a random walk kernel matrix |
| a | numeric. It is correlated to the probability of remaining at the same vertex. Larger a, larger the probability (def. 2) |
| p | integer. Number of steps (def: p=2) |

## Details

`rw.kernel` methods computes the one step random walk kernel RW, i.e.:

$$RW = (a-1)I + D^{-\frac{1}{2}} * W * D^{-\frac{1}{2}}$$

where $I$ is the identity matrix, $W$ is the weighted adjacency matrix of an undirected graph, and $D$ is a diagonal matrix with $D_{ii} = \sum_j W_{ij}$

`p.step.rw.kernel` methods compute the p-step random walk kernel pRW, i.e.:

$$pRW = RW^p$$

## Value

`rw.kernel`: A numeric square matrix representing a one-step random walk kernel matrix

`p.step.rw.kernel`: A numeric square matrix representing a p-step random walk kernel matrix

## Methods

`signature(W = "graph") rw.kernel` computes the random walk kernel starting from a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`signature(W = "matrix") rw.kernel` computes the random walk kernel starting from a weighted adjacency matrix representing the graph

`signature(RW = "graph") p.step.rw.kernel` computes the a p-step random walk kernel starting from a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`signature(RW = "matrix") p.step.rw.kernel` computes the p-step random walk kernel starting from a one-step random walk kernel matrix

## Examples

```
# Random walk kernel computation using Functional Interaction network data
library(bionetdata);
data(FIN.data);
W <- as.matrix(FIN.data);
K <- rw.kernel(W);
# this a 2-step random walk kernel
K2 <- p.step.rw.kernel(K, p=2);
```

RWR *Random walk with Restart on a graph*

## Description

Function that performs a random Walk with restart (RWR) on a given graph

## Usage

```
RWR(W, ind.positives, gamma = 0.6, tmax = 1000, eps = 1e-10, norm = TRUE)
```

## Arguments

| | |
|---|---|
| W | a numeric matrix representing the adjacency matrix of the graph |
| ind.positives | indices of the "core" positive examples of the graph. They represent the indices of W corresponding to the positive examples |
| gamma | restart parameter (def: 0.6) |
| tmax | maximum number of iterations (steps) (def: 1000) |
| eps | maximum allowed difference between the computed probabilities at the steady state (def. 1e-10) |
| norm | if TRUE (def) the adjacency matrix $W$ of the graph is normalized to $M = D^{-1} * W$, otherwise it is assumed that the matrix $W$ is just normalized |

## Details

RWR performs a random Walk with restart on a given graph by performing 1 or more steps on the graph, depending on the value of the tmax parameter. The restart parameter expresses the probability of "restarting" from a "core" node at each step of the random walk algorithm. It stops also if the difference of the norm of the probabilities between two consecutive steps is less than eps.

## Value

A list with three elements:

| | |
|---|---|
| p | numeric vector. Probability of each node at the steady state or after tmax iterations |
| ind.positives | indices of the "core" positive examples of the graph (it is equal to the same input parameter) |
| n.iter | number of performed steps/iterations |

## References

L. Lovasz, Random Walks on Graphs: a Survey, Combinatorics, Paul Erdos is Eighty, vol. 2, pp. 146, 1993.

## See Also

RW

## Examples

```
# Application of the random walk with restart to the prediction of the
# DrugBank category Penicillins
# using the Tanimoto chemical structure similarity network
# between 1253 DrugBank drugs
library(bionetdata);
data(DD.chem.data);
data(DrugBank.Cat);
labels <- DrugBank.Cat[,"Penicillins"];
ind.pos <- which(labels==1);
# 2-step RWR
res <- RWR(DD.chem.data, ind.pos, tmax = 2);
## Not run:
# till to convergence
res <- RWR(DD.chem.data, ind.pos, tmax = 5000, eps=1e-6);
# 5 steps and higher gamma
res <- RWR(DD.chem.data, ind.pos, tmax = 5, gamma=0.8);

## End(Not run)
```

---

# score.multiple.vertex-methods

*Multiple vertex score functions*

---

## Description

Methods to compute score functions for multiple vertices of the graph

## Usage

```
## S4 method for signature 'graph'
NN.score(RW, x, x.pos, auto = FALSE, norm = TRUE)
## S4 method for signature 'matrix'
NN.score(RW, x, x.pos, auto = FALSE, norm = TRUE)
## S4 method for signature 'graph'
KNN.score(RW, x, x.pos, k = 3, auto = FALSE, norm = TRUE)
## S4 method for signature 'matrix'
KNN.score(RW, x, x.pos, k = 3, auto = FALSE, norm = TRUE)
## S4 method for signature 'graph'
eav.score(RW, x, x.pos, auto = FALSE, norm = TRUE)
## S4 method for signature 'matrix'
eav.score(RW, x, x.pos, auto = FALSE, norm = TRUE)
## S4 method for signature 'graph'
WSLD.score(RW, x, x.pos, d = 2, auto = FALSE, norm = TRUE)
```

```
## S4 method for signature 'matrix'
WSLD.score(RW, x, x.pos, d = 2, auto = FALSE, norm = TRUE)
```

## Arguments

| | |
|---|---|
| RW | matrix. It must be a kernel matrix or a symmetric matrix expressing the similarity between nodes |
| x | vector of integer. Indices corresponding to the elements of the RW matrix for which the score must be computed |
| x.pos | vector of integer. Indices of the positive elements of the RW matrix |
| k | integer. Number of the k nearest neighbours to be considered |
| d | integer. Coefficient of linear decay (def. 2) |
| auto | boolean. If TRUE the components $K(x,x)+K(x_i,x_i)$ are computed, otherwise are discarded (default) |
| norm | boolean. If TRUE (def.) the scores are normalized between 0 and 1. |

## Details

The methods compute the scores for multiple vertices according to NN, KNN, Empirical Average or WSLD score (see reference for bibliographic details). Note that the argument x indicates the set of nodes for which the score must be computed. The vector x represents the indices of the rows of the matrix RW corresponding to the vertices for which the scores must be computed. If x = 1:nrow(RW) the scores for all the vertices of the graph are computed.

## Value

`NN.score`: a numeric vector with the NN scores of the vertices. The names of the vector correspond to the indices x

`KNN.score`: a numeric vector with the KNN scores of the vertices. The names of the vector correspond to the indices x

`eav.score`: a numeric vector with the Empirical Average score of the vertices. The names of the vector correspond to the indices x

`WSLD.score`: a numeric vector with the Weighted Sum with Linear Decay score (WSLD) of the vertices. The names of the vector correspond to the indices x

## Methods

`signature(RW = "graph")` `NN.score` computes the NN score for multiple vertices using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`KNN.score` computes the KNN score for multiple vertices using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`eav.score` computes the Empirical Average score for multiple verticesusing a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`WSLD.score` computes the Weighted Sum with Linear Decay score for multiple vertices using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`signature(RW = "matrix")` `NN.score` computes the NN score for multiple vertices using a kernel matrix or a symmetric matrix expressing the similarity between nodes

`KNN.score` computes the KNN score for multiple vertices using a kernel matrix or a symmetric matrix expressing the similarity between nodes

`eav.score` computes the Empirical Average score multiple for vertices using a kernel matrix or a symmetric matrix expressing the similarity between nodes

`WSLD.score` computes the Weighted Sum with Linear Decay score for multiple vertices using a kernel matrix or a symmetric matrix expressing the similarity between nodes

## References

Re M, Mesiti M, Valentini G: A fast ranking algorithm for predicting gene functions in biomolecular networks. IEEE ACM Trans Comput Biol Bioinform 2012, 9(6):1812-1818.

Insuk Lee, Bindu Ambaru, Pranjali Thakkar, Edward M. Marcotte, and Seung Y. Rhee. Nature Biotechnology 28, 149-156, 2010

## See Also

Methods for scoring a single vertex

## Examples

```
# Computation of scores using STRING data with respect to
# the FunCat category 11.02.01 rRNA synthesis
library(bionetdata);
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
labels <- Yeast.STRING.FunCat[,"11.02.01"];
n <- length(labels);
ind.pos <- which(labels==1);
# NN-scores computed directly on the STRING matrix
s <- NN.score(Yeast.STRING.data, 1:n, ind.pos);
## Not run:
# NN-scores computed on the 1 step and 2-step random walk kernel matrix
K <- rw.kernel(Yeast.STRING.data);
sK <- NN.score(K, 1:n, ind.pos);
K2 <- p.step.rw.kernel(K, p=2);
sK2 <- NN.score(K2, 1:n, ind.pos);
# WSLD-scores computed directly on the STRING matrix
s <- WSLD.score(Yeast.STRING.data, 1:n, ind.pos);
# WSLD-scores computed on the 1 step and 2-step random walk kernel matrix
sK <- WSLD.score(K, 1:n, ind.pos);
sK2 <- WSLD.score(K2, 1:n, ind.pos);

## End(Not run)
```

score.single.vertex-methods
*Single vertex score functions*

## Description

Methods to compute score functions applied to a single vertex of the graph

## Usage

```
## S4 method for signature 'graph'
single.NN.score(RW, x, x.pos, auto = FALSE)
## S4 method for signature 'matrix'
single.NN.score(RW, x, x.pos, auto = FALSE)
## S4 method for signature 'graph'
single.KNN.score(RW, x, x.pos, k = 3, auto = FALSE)
## S4 method for signature 'matrix'
single.KNN.score(RW, x, x.pos, k = 3, auto = FALSE)
## S4 method for signature 'graph'
single.eav.score(RW, x, x.pos, auto = FALSE)
## S4 method for signature 'matrix'
single.eav.score(RW, x, x.pos, auto = FALSE)
## S4 method for signature 'graph'
single.WSLD.score(RW, x, x.pos, d = 2, auto = FALSE)
## S4 method for signature 'matrix'
single.WSLD.score(RW, x, x.pos, d = 2, auto = FALSE)
```

## Arguments

| | |
|---|---|
| RW | matrix. It must be a kernel matrix or a symmetric matrix expressing the similarity between nodes |
| x | integer. Index corresponding to the element of the RW matrix for which the score must be computed |
| x.pos | vector of integer. Indices of the positive elements of the RW matrix |
| k | integer. Number of the k nearest neighbours to be considered |
| d | integer. Coefficient of linear decay (def. 2) |
| auto | boolean. If TRUE the components $K(x,x)+K(x_i,x_i)$ are computed, otherwise are discarded (default) |

## Details

single.NN.score computes the NN score for a single vertex:

$$score(x) = -\min_{x_i \in V_C} (K(x,x) + K(x_i,x_i) - 2K(x,x_i))$$

where $V_C$ is the set of positive vertices.

`single.KNN.score` compute KNN score for a single vertex:

$$score(x) = - \sum_{k\ nearest\ x_i \in V_C} (K(x,x) + K(x_i,x_i) - 2K(x,x_i))$$

`single.eav.score` computes the Empirical Average score for a single vertex:

$$score(x) = -K(x,x) + \frac{2}{|V_C|} * \sum_{x_i \in V_C} K(x,x_i)$$

`single.WSLD.score` computes the WSLD score for a single vertex:

Let $K(x, x_{jk})$ be the kth rank order index w.r.t. $x_j \in V_C$, and $m = |V_C|$, then:

$$score(x) = \max_{x_i \in V_C} K(x,x_i) + \sum_{k=2}^{m} [(1/(d*(k-1))) * K(x,x_{jk})]$$

## Value

`single.NN.score`: the NN score of the vertex

`single.KNN.score`: the KNN score of the vertex

`single.eav.score`: the Empirical Average score of the vertex

`single.WSLD.score`: the Weighted Sum with Linear Decay score (WSLD) of the vertex

## Methods

`signature(RW = "graph") single.NN.score` computes the NN score for a single vertex using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`single.KNN.score` computes the KNN score for a single vertex using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`single.eav.score` computes the Empirical Average score for a single vertex using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`single.WSLD.score` computes the Weighted Sum with Linear Decay score for a single vertex using a graph of class graph (hence including objects of class graphAM and graphNEL from the package graph)

`signature(RW = "matrix") single.NN.score` computes the NN score for a single vertex using a kernel matrix or a symmetric matrix expressing the similarity between nodes

`single.KNN.score` computes the KNN score for a single vertex using a kernel matrix or a symmetric matrix expressing the similarity between nodes

`single.eav.score` computes the Empirical Average score using a kernel matrix or a symmetric matrix expressing the similarity between nodes

`single.WSLD.score` computes the Weighted Sum with Linear Decay score for a single vertex using a kernel matrix or a symmetric matrix expressing the similarity between nodes

## See Also

Methods for scoring multiple vertices

## Examples

```
# Computation of scores using STRING data with respect to
# the FunCat category 11.02.01 rRNA synthesis
library(bionetdata);
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
labels <- Yeast.STRING.FunCat[,"11.02.01"];
n <- length(labels);
ind.pos <- which(labels==1);
# NN-score computed directly on the STRING matrix on the first yeast gene YJR121W
s <- single.NN.score(Yeast.STRING.data, 1, ind.pos);
## Not run:
# NN-score computed on the 1 step and 2-step random walk kernel matrix
K <- rw.kernel(Yeast.STRING.data);
sK <- single.NN.score(K, 1, ind.pos);
K2 <- p.step.rw.kernel(K, p=2);
sK2 <- single.NN.score(K2, 1, ind.pos);

# WSLD-score computed directly on the STRING matrix on the first  yeast gene YJR121W
s <- single.WSLD.score(Yeast.STRING.data, 1, ind.pos);
# WSLD-scores computed on the 1 step and 2-step random walk kernel matrix
sK <- single.WSLD.score(K, 1, ind.pos);
sK2 <- single.WSLD.score(K2, 1, ind.pos);

## End(Not run)
```

---

`Utilities` *Utility functions*

---

## Description

Mixed utility functions to compute accuracy, norms, labels from scores and to perform stratified cross-validation.

## Usage

```
compute.acc(pred, labels)
compute.F(pred, labels)
norm1(x)
Unit.sphere.norm(K)
do.stratified.cv.data(examples, positives, k = 5, seed = NULL)
labelsfromscores(scores, thresh)
Multiple.labels.from.scores(S, thresh.vect)
selection.test(pos.scores, av.scores, ind.positives, alpha = 0.05, thresh.pos = 0)
```

## Arguments

| | |
|---|---|
| pred | vector of the predicted labels |
| labels | vector of the true labels. Note that 0 stands for negative and 1 for positive. In general the first level is negative and the second positive |
| x | numeric vector |
| K | a kernel matrix |
| examples | indices of the examples (a vector of integer) |
| positives | vector of integer. Indices of the positive examples. The indices refer to the indices of examples |
| k | number of folds (def = 5) |
| seed | seed of the random generator (def=NULL). If is set to NULL no initiazitation is performed |
| scores | numeric. Vector of scores: each element correspond to the score of an example |
| thresh | real value. Threshold for the classification |
| S | numeric matrix. Matrix of scores: rows represent examples, columns classes |
| thresh.vect | numeric vector. Vector of the thresholds for multiple classes (one threshold for each class) |
| pos.scores | vector with scores of positive examples. It is returned from multiple.ker.score.cv. |
| av.scores | a vector with the average scores computed by multiple.ker.score.cv. It may be a named vector. If not, the names attributes corresponding to the indices of the vector are added. |
| ind.positives | indices of the positive examples. They are the indices of av.scores corresponding to positive examples. |
| alpha | quantile level (def. 0.05) |
| thresh.pos | only values larger than thresh.pos are retained in pos.scores (def.: 0) |

## Details

`compute.acc` computes the accuracy for a single class

`compute.F` computes the F-score for a single class

`norm1` computes the L1-norm of a numeric vector

`Unit.sphere.norm` normalize a kernel according to the unit sphere

`do.stratified.cv.data` generates data for the stratified cross-validation. In particular subdividas the indices that refer to the rows of the data matrix in different folds (separated for positive and negative examples)

`labelsfromscores` computes the labels of a single class from the corresponding scores

`Multiple.labels.from.scores` computes the labels of multiple classes from the corresponding scores

`selection.test` is a non parametric test to select the most significant unlabeled examples

## Value

`compute.acc` returns the accuracy

`compute.F` returns the F-score

`norm1` returns the L1-norm value

`Unit.sphere.norm` returns the kernel normalized according to the unit sphere

`do.stratified.cv.data` returns a list with 2 two components:

`fold.non.positives`
: a list with k components. Each component is a vector with the indices of the non positive elements of the fold

`fold.positives`
: a list with k components. Each component is a vector with the indices of the positive elements of the fold

Indices refer to row numbers of the data matrix

`labelsfromscores` returns a numeric vector res with 0 or 1 values. The label res[i]=1 if scores[i]>thresh, otherwise res[i]=0

`Multiple.labels.from.scores` returns a binary matrix with the labels of the predictions. Rows represent examples, columns classes. Element L[i,j] is the label of example i w.r.t. class j. L[i,j]=1 if i belongs to j, 0 otherwise.

`selection.test` returns a list with 5 components:

`selected`
: a named vector with the components of av.scores selected by the test

`selected.labeled`
: a named vector with the labeled components of av.scores selected by the test

`selected.unlabeled`
: a named vector with the unlabeled components of av.scores selected by the test

`thresh`
: the score threshold selected by the test

`alpha`
: significance level (the same value of the input)

## Examples

```
# L1-norm of a vector
norm1(rnorm(10));
# generation of 5 stratified folds;
do.stratified.cv.data(1:100, 1:10, k = 5, seed = NULL);
# generation of labels form scores.
labelsfromscores(runif(20), thresh=0.3);
```

# Index