Package 'COSNet'

October 14, 2021

Type Package

Title Cost Sensitive Network for node label prediction on graphs with highly unbalanced labelings

Version 1.26.0

Date 2015-11-06

Author Marco Frasca and Giorgio Valentini -- Universita' degli Studi di Milano

Maintainer Marco Frasca<frasca@di.unimi.it>

Description Package that implements the COSNet classification algorithm. The algorithm predicts node labels in partially labeled graphs where few positives are available for the class being predicted.

License GPL (>= 2)

URL https://github.com/m1frasca/COSNet_GitHub

LazyLoad yes

Suggests bionetdata, PerfMeas, RUnit, BiocGenerics

biocViews GraphAndNetwork, Classification,Network, NeuralNetwork

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

git_branch RELEASE_3_13

git_last_commit 7f28084

git_last_commit_date 2021-05-19

Date/Publication 2021-10-14

R topics documented:

COSNet-package 2
COSNet 3
cosnet.cross.validation 5
find.division.not.strat 6
find.division.strat 7
generate_labels 8

generate_points 9
optimizep 10
optimize_pos_above 11
reg_data 13
runSubnet 14

Index **17**

`COSNet-package` *R package for binary classification and node ranking on partially labeled graphs with unbalanced labels.*

Description

Algorithm based on cost-sensitive neural network for predicting node labels in a semi-supervised setting.

Details

Package:	COSNet
Type:	Package
Version:	1.5.1
Date:	2015-11-06
License:	GPL (>= 2)
LazyLoad:	yes

This package implements the COSNet algorithm (Frasca et al. 2013). COSNet is a semi-supervised cost-sensitive neural network for predicting node labels in partially labeled graphs. The algorithm is able in inferring a labeling for unlabeled nodes in the graph starting from the graph topology and the known labels.

Author(s)

Marco Frasca, and *Giorgio Valentini*

DI, Dipartimento di Informatica

Universita' degli Studi di Milano

`<{frasca,valentini}@di.unimi.it>`

Maintainer: *Marco Frasca*

References

Frasca M., Bertoni A., Re M., Valentini G.: A neural network algorithm for semi-supervised node label learning from unbalanced data. Neural Networks, Volume 43, July, 2013 Pages 84-98.

Bertoni A., Frasca M., Valentini G.: COSNet: a Cost Sensitive Neural Network for Semi-supervised Learning in Graphs. ECML PKDD'11 Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part I. Pages 219-234

`COSNet` *Cost Sensitive Network for node label prediction on graphs with highly unbalanced labelings*

Description

This function realizes the COSNet algorithm (Frasca et al. 2013). COSNet is a semi-supervised algorithm based on parametric Hopfield networks for predicting labels for unlabeled nodes in graphs which are only partially labeled.

Usage

```
COSNet(W, labeling, cost = 0)
```

Arguments

`W`	square symmetric named matrix, whose components are in the interval [0,1]. The i,j-th component is a similarity index between node i and node j. The components of the diagonal of W are zero. Rows and columns should be named identically.
`labeling`	vector of node labels: 1 for positive examples, -1 for negative examples, 0 for unlabeled nodes.
`cost`	real value corresponding to beta parameter in the equation eta = beta*|tan((alpha - pi/4)*2)|, where eta is the coefficient of the regularization term in the energy function (Frasca et al. 2013). If cost = 0 (default) the unregularized version is executed. "cost" is a real value that reduces or increases the influence of regularization on the network dynamics. The higher the value of cost, the more the influence of the regularization term. It is suggested using small values for this parameter (e.g. cost = 0.0001)

Details

`COSNet` contructs a Hopfield network whose connection matrix is W, and it applies a cost-sensitive strategy to determine the network parameters starting from W and labels "labeling". COSNet distinguishes between labeled (1, -1 components in "labeling") and unlabeled (zero components in "labeling") nodes, and it is made up by three steps: 1) Generating a random labeling (1, -1) for unlabeled nodes. 2) Learning the parameters "alpha and "c" such that the line "cos(alpha)*y - sin(alpha)*x - q*cos(alpha) = 0 " linearly separates a suitable set of labeled points (in which each point corresponds to a labeled node) and optimizes (in terms of alpha and q) the F-score criterion. The output of this phase is the intercept "q" of the optimum line and the corresponding angle "alpha". Then each neuron has threshold c = - q*cos(alpha). 3) Extending "c" and "alpha" to the subnetwork composed of only unlabeled nodes, and simulating it until an equilibrium state is reached. The dynamics of this network is regularized by adding a term to the energy function that is minimized

when the proportion of positive in the network state is roughly the proportion of positives in the labeled part of the network. The parameter "cost" corresponds to the parameter beta in the equation eta = beta*|tan((alpha - pi/4)*2)| (see Frasca et al. 2013). When the equilibrium state is reached, positive nodes will be predicted as positive for the current task.

Value

`COSNet` returns a list with six elements:

`alpha`	the optimum angle
`c`	the optimum threshold
`Fscore`	the optimum F-score computed in Step 2
`pred`	the vector of predictions for unlabeled nodes
`scores`	the vector of scores for unlabeled nodes
`iter`	number of iterations until convergence in Step 3

References

Frasca M., Bertoni A., Re M., Valentini G.: A neural network algorithm for semi-supervised node label learning from unbalanced data. Neural Networks, Volume 43, July, 2013 Pages 84-98.

Examples

```
library(bionetdata);
## loading Binary protein-protein interactions from the STRING
## data base (von Mering et al. 2002)
data(Yeast.STRING.data)# "Yeast.STRING.data"
## FunCat classes annotations (0/1) for the genes included
## in Yeast.STRING.data. Annotations refer the funcat-2.1
## scheme, and funcat-2.1 data 20070316 data, available from the MIPS web site.
data(Yeast.STRING.FunCat) # "Yeast.STRING.FunCat"
labels <- Yeast.STRING.FunCat;
labels[labels == 0] <- -1;
## excluding the dummy "00" root
labels <- labels[, -which(colnames(labels) == "00")];
n <- nrow(labels);
k <- floor(n/10);
cat("k = ", k, "\n");
## choosing the first class
labeling <- labels[, 1];
## randomly choosing a subset of genes whose labels are hidden
hidden <- sort(sample(1:n, k));
hidden.labels <- labeling[hidden];
labeling[hidden] <- 0;
out <- COSNet(Yeast.STRING.data, labeling, 0);
prediction <- out$pred;
TP <- sum(hidden.labels == 1 & prediction == 1);
FN <- sum(hidden.labels == 1 & prediction == -1);
FP <- sum(hidden.labels == -1 & prediction == 1);
out2 <- COSNet(Yeast.STRING.data, labeling, 0.0001);
```

```
prediction <- out2$pred;
TP2 <- sum(hidden.labels == 1 & prediction == 1);
FN2 <- sum(hidden.labels == 1 & prediction == -1);
FP2 <- sum(hidden.labels == -1 & prediction == 1);
```

`cosnet.cross.validation`

Cross validation procedure for the COSNet algorithm

Description

This function applies the function `COSNet` to the input data with a cross validation procedure.

Usage

```
cosnet.cross.validation(labels, W, nfolds, cost)
```

Arguments

`labels`	named matrix of node labels. The (i-j)-th component contains the label (1 for positive examples, -1 for negative examples) of node i for j-th functional class to be predicted
`W`	square symmetric named matrix. The (i,j)-th component is a similarity index between node i and node j. The components of the diagonal of W are zero.
`nfolds`	integer corresponding to the number of desired folds
`cost`	real value that corresponds to the cost parameter of `COSNet`

Details

`cosnet.cross.validation` runs the function `COSNet` on the input data through a cross validation procedure. For each class to be predicted (column of "labels"), both "W" and "labels" are partitioned into "nfolds" subsets and at each iteration the labels of a fold are hidden and predicted with function `COSNet`. When possible, input data are partitioned by ensuring the same proportion of positive and negative instances in each fold.

Value

List with three elements:

`labels`	1/-1 named input label matrix, in which rows correspond to nodes and columns to classes
`predictions`	named 1/-1 prediction matrix, in which rows correspond to nodes and columns to classes. The position i-j-th is 1 if the node i has been predicted as positive for the class j, -1 otherwise
`scores`	named real score matrix, in which rows correspond to nodes and columns to classes. The position i-j-th is a real number corresponding to the internal energy at equlibrium for node i when predicting class j. This score is a "degree" of membership of node i to the class j

See Also

COSNet

Examples

```
library(bionetdata);
data(Yeast.STRING.data)
data(Yeast.STRING.FunCat) # "Yeast.STRING.FunCat"
## excluding the dummy "00" root
Yeast.STRING.FunCat <-
        Yeast.STRING.FunCat[, -which(colnames(Yeast.STRING.FunCat) == "00")];
nfolds <- 5;
res <- cosnet.cross.validation(Yeast.STRING.FunCat[, 1:50], Yeast.STRING.data,
        nfolds, 0.0001);
## computing performances
library(PerfMeas);
perf <- F.measure.single.over.classes(res$labels, res$predictions);
cat(perf$average);
```

---

find.division.not.strat

Random partitioning of input data

Description

Function to determine a random partition of the input vector into a specifiend number of folds.

Usage

```
find.division.not.strat(vett, n_fold)
```

Arguments

vett	vector to be partitioned
n_fold	number of folds in which the argument vett must be partitioned

Value

List with n_fold elements, the i-th element is a vector corresponding to i-th fold.

See Also

find.division.strat

Examples

```
n <- 100;
vett <- runif(n, 0, 1);
n_fold <- 5;
fold_list <- find.division.not.strat(vett, n_fold);
length(fold_list);
## indices of the first fold
fold_list[[1]];
```

`find.division.strat` *Compute a stratified random partition of the input data*

Description

Function to determine a random partition of the labeled input vector into a fixed number of folds, such that each fold has around the same proportion of two-class labels.

Usage

```
find.division.strat(labels, vett, n_fold)
```

Arguments

`labels`	binary -1/1 label vector. labels[i] is the label for the element vett[i].
`vett`	vector to be partitioned
`n_fold`	number of folds in which the argument vett must be partitioned

Details

The input vector "vett" is randomly partitioned into "n_fold" folds ensuring each fold contains roughly the same proportions of positives and negative labels, according to the labeling "labels"

Value

List with n_fold elements, the i-th element is a vector corresponding to i-th fold.

See Also

`find.division.not.strat`

Examples

```
n <- 100;
vett <- runif(n, 0, 1)
labels <- c(rep(1, floor(n/3)), rep(-1, ceiling(2*n/3)));
n_fold <- 5;
fold_list <- find.division.strat(labels, vett, n_fold);
length(fold_list);
## number of positives in the first fold
sum(labels(fold_list[[1]]) > 0);
```

---

generate_labels *Generate random labels*

Description

Function to generate a vector of random labels

Usage

```
generate_labels(n, pos_rate)
```

Arguments

n	number of labels to be generated
pos_rate	rate of expected positive labels

Details

This function generates "n" random labels in -1, 1 drawn from the binomial distribution B(n, pos_rate)

Value

-1/1 vector of length "n" containing the generated labels

See Also

generate_points

Examples

```
pos_rate <- 0.3;
n <- 100;
## generating n random labels with 0.3 expected rate of positive labels
random_labels <- generate_labels(n, pos_rate);
sum(random_labels > 0);
```

`generate_points` *Compute the points to be separated in Step 2 of COSNet algorithm (Frasca et al. 2013)*

Description

This function associates each labeled node with a point in the plane, whose coordinates are respectively the weighted sum of its positive and negative neighborhoods

Usage

```
generate_points(W, unlabeled, labeling)
```

Arguments

`W`	square symmetric named matrix, whose components are in the [0,1] interval. The i,j-th component is the weight between node i and node j. The components of the diagonal of W are zero.
`unlabeled`	vector of the indices of the unlabeled nodes
`labeling`	vector of node labels : 1 for positive nodes, -1 for negative nodes, 0 for unlabeled nodes

Details

For each labeld node k, a point (pos_vect[k], neg_vect[k]) is computed, where pos_vect[k] is the whighted sum of the positive neighbors of node k and neg_vect[k] is the weighted sum of negative neighbors of node k.

Value

List of two element:

`pos_vect`	is the vector of the abscissae; pos_vect[k] contains the whighted sum of the positive neighbors of node k
`neg_vect`	is the vector of the ordinates; neg_vect[k] contains the whighted sum of the negative neighbors of node k

References

Frasca M., Bertoni A., Re M., Valentini G.: A neural network algorithm for semi-supervised node label learning from unbalanced data. Neural Networks, Volume 43, July, 2013 Pages 84-98.

Examples

```
## randomly generating labels
labels <- generate_labels(100, 0.3);
unlabeled <- sample(1:100, 10);
labels[unlabeled] <- 0;
## randomly generating connection matrix
W <- matrix(sample(1:10000, 100*100)/1000, nrow = 100);
diag(W) <- 0;
points <- generate_points(W, unlabeled, labels);
points$pos_vect[1:5];
points$neg_vect[1:5];
```

`optimizep` *Optimizing algorithm parameters*

Description

Function to learn the parameters "alpha", determining neurons activation values, and the neuron threshold "c".

Usage

```
optimizep(pos_vect, neg_vect, training_labels)
```

Arguments

`pos_vect`	vector of abscissae of the points to be separated
`neg_vect`	vector of ordinates of the points to be separated
`training_labels`	1/-1 vector of point labels

Details

This function computes the optimal angle "alpha" and the optimal threshold "c". For each labeled neuron k, a point (pos_vect[k], neg_vect[k]) is considered. Points are labeled according to the labels contained in the vector "training_labels". Then the straight line, among those with positive slope, which separates these points by maximizing the F-score is learned. The line is represented by the angle alpha formed with the "x" axis, and its intercept "q" with "y" axis. When separating points by a straight line, there are two possibility: 1) Considering as positive the half-plane above the line; 2) Considering as positive the half-plane below the line. The procedure investigates both these possibilities, and also returns which choice between 1) and 2) corresponds to the best F-score.

Value

list "res" with 4 components:

res$alpha	value of the optimum angle alpha
res$c	value of the optimum threshold c
res$Fscore	value of the optimum F-score
res$pos_half	the position of the positive half-plane : > 0 in case 1), < 0 in case 2)

See Also

optimize_pos_above

Examples

```
library(bionetdata);
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
n <- nrow(Yeast.STRING.data);
## removing dummy node 00
Yeast.STRING.FunCat <- Yeast.STRING.FunCat[,
                        -which(colnames(Yeast.STRING.FunCat)=="00")];
## selecting the class with index 1
class <- 1;
labels <- as.vector(Yeast.STRING.data[, class]);
names(labels) <- rownames(Yeast.STRING.FunCat);
labels <- as.vector(Yeast.STRING.FunCat[, class]);
names(labels) <- rownames(Yeast.STRING.FunCat);
## partitioning the data
folds <- find.division.strat(labels, 1:n, 3);
labels[labels <= 0] <- -1;
test.set <- folds[[1]];
training.set <- setdiff(1:n, test.set);
labels[test.set] <- 0;
## generating the points to be separated
points <- generate_points(Yeast.STRING.data, test.set, labels);
opt_parameters <- optimizep(points$pos_vect[training.set],
                    points$neg_vect[training.set], labels[training.set]);
str(opt_parameters);
```

---

optimize_pos_above *Alternative algortihm for optimizing parameters*

Description

Alternative algorithm to compute the two parameters: 'alpha', determining neurons activation values, and the neuron thresholds 'c'.

Usage

```
optimize_pos_above(pos_vect, neg_vect, training_labels, res)
```

Arguments

pos_vect	vector of abscissae of the points to be separated
neg_vect	vector of ordinates of the points to be separated
training_labels	1/-1 vector of the point labels
res	list containing the optimum angle (field alpha), neuron threshold (field c), maximum F-score (field Fscore) and positive halfplane (field pos_half) computed by the procedure corresponding to the choice 1) of function optimizep.

Details

Function to optimize the the parameters 'alpha' and 'c' when the function optimizep determines that the maximum F-score corresponds to the half-plane above the separation line. The algorithm works in three steps: 1) Selecting all the points which lie on the Y-axis. The aim is to choose a positive point which will be the center of the line bundle we consider in the next step. We sort the selected points by ordinate and for each positive point we compute the F-score of the almost vertical line (but with negative slope) crossing this point considering solely the selected points. Then we choose the point k which corresponds to the highest F-score. 2) The algorithm computes the slopes of the lines crossing the point k and each point not lying on the Y-axis. Then it searches the line, among those with negative slope, which maximizes the F-score criterion by sorting the computed lines according to their slopes in an increasing order. Consequently, the angle alpha relative to the optimum line is in the interval]pi/2, pi[. 3) Compute the intercepts of the straight lines whose slope is tan(alpha) and crossing each available point. The optimum line is identified by scanning the computed lines according to their intercept in an increasing order. Let q be the intercept of the optimum line y = tan(alpha)z + q, then we set c = -cos(alpha)q. If there are no positive point with abscissa 0, the function returns the optimal parameters contained in input argument 'res' computed by the procedure corresponding to the choice 1) of function optimizep.

Value

list res with 3 components

res$alpha	value of the optimum angle alpha
res$c	value of the optimum threshold c
res$Fscore	value of the optimum F-score
res$pos_half	position of the positive half-plane (-1 below, 1 above the optimum straith line)

See Also

optimizep

Examples

```
library(bionetdata);
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
n <- nrow(Yeast.STRING.data);
## removing dummy node 00
Yeast.STRING.FunCat <- Yeast.STRING.FunCat[,
                        -which(colnames(Yeast.STRING.FunCat) == "00")];
## selecting the class with index 1
class <- 1;
labels <- as.vector(Yeast.STRING.data[, class]);
names(labels) <- rownames(Yeast.STRING.FunCat);
labels <- as.vector(Yeast.STRING.FunCat[, class]);
names(labels) <- rownames(Yeast.STRING.FunCat);
## partitioning the data
folds <- find.division.strat(labels, 1:n, 3);
labels[labels <= 0] <- -1;
test.set <- folds[[1]];
training.set <- setdiff(1:n, test.set);
labels[test.set] <- 0;
points <- generate_points(Yeast.STRING.data, test.set, labels);
## setting values for the parameter
res <- list(alpha=pi/2, c=0, Fscore=0, pos_half=-1);
opt_parameters <- optimize_pos_above(points$pos_vect[training.set],
                    points$neg_vect[training.set], labels[training.set], res);
str(opt_parameters);
```

---

reg_data *Function to compute the regularized version of COSNet (Frasca et al. 2013)*

Description

This function modifies the weights and the thresholds of the network to realized the COSNet regularization.

Usage

```
reg_data(W, theta, eta, M, m, pos_num)
```

Arguments

W	square symmetric named matrix of the network weights. The components of W are in the [0,1] interval. The i,j-th component is the weight between neuron i and neuron j. The components of the diagonal of W are 0
theta	vector of the neuron activation thresholds
eta	real value corresponding to the eta regularization coefficient in the energy function (Frasca et al. 2013). If eta = 0 no regularization is applied. The higher the value of eta, the more the influence of the regularization term

M	positive neuron activation value
m	negative neuron activation value
pos_num	number of expected positive neurons in the equilibrium state of the network

Value

list of two element:

W	the regularized connection matrix
theta	regularized threshold vector

References

Frasca M., Bertoni A., Re M., Valentini G.: A neural network algorithm for semi-supervised node label learning from unbalanced data. Neural Networks, Volume 43, July, 2013 Pages 84-98.

Examples

```
library(bionetdata);
data(Yeast.STRING.data);
n <- nrow(Yeast.STRING.data);
dim(Yeast.STRING.data);
range(Yeast.STRING.data);
## setting values for parameter alpha, for the rate of positive examples,
## for neuron thresholds and for eta parameter
alpha <- 1;
pos.rate <- 0.01;
thresholds <- runif(n);
range(thresholds);
eta <- 0.001;
a <- reg_data(Yeast.STRING.data, thresholds, eta, sin(alpha),
    -cos(alpha), ceiling(pos.rate*n));
## new connection matrix
dim(a$W);
range(a$W);
## new thresholds
range(a$theta);
```

---

runSubnet *Realizing the network dynamics.*

Description

Function to simulate the dynamics of the network composed of unlabeled nodes.

Usage

```
runSubnet(W, labeling, alpha_value, c_value, cost)
```

Arguments

W	square symmetric named matrix, whose componemts are in the interval [0,1]. The i,j-th component is a similarity index between node i and node j. The components of the diagonal of W are zero. Rows and columnns should be named identically.
labeling	vector of node labels: 1 for positive examples, -1 for negative examples, 0 for unlabeled nodes
alpha_value	real value in [0, pi/2[, determining the neuron activation values: sin(alpha) and -cos(alpha).
c_value	real value used as activation threshold for each neuron
cost	real value corresponding to beta parameter in the equation eta = beta*|tan((alpha - pi/4)*2)|, where eta is the coefficient of the regularization term in the energy function (Frasca et al. 2013). If cost = 0 (default) the unregularized version is executed. The higher the value of cost, the more the influence of the regularization term. It is suggested using small values for this parameter (e.g. cost = 0.00001)

Details

Function to simulate the subnetwork composed of the unlabeled genes, in which each neuron has sin(alpha_value), -cos(alpha_value) as activation value, and in which each neuron has a threshold "c_value" minus the contribution from the labeled neighbors.

Value

list with three components:

state	Named vector of prediction (at equilibrium) for unlabeled nodes
scores	Named vector of scores (at equilibrium) for unlabeled nodes
iter	Number of iterations of the network until convergence

References

Frasca M., Bertoni A., Re M., Valentini G.: A neural network algorithm for semi-supervised node label learning from unbalanced data. Neural Networks, Volume 43, July, 2013 Pages 84-98.

Examples

```
library(bionetdata);
data(Yeast.STRING.data);
data(Yeast.STRING.FunCat);
n<-nrow(Yeast.STRING.data);
## removing dummy node 00
Yeast.STRING.FunCat <- Yeast.STRING.FunCat[,
                        -which(colnames(Yeast.STRING.FunCat) == "00")];
class <- 1;
labels <- as.vector(Yeast.STRING.data[, class]);
names(labels) <- rownames(Yeast.STRING.FunCat);
```

```
labels <- as.vector(Yeast.STRING.FunCat[, class]);
names(labels) <- rownames(Yeast.STRING.FunCat);
folds <- find.division.strat(labels, 1:n, 3);
labels[labels <= 0] <- -1;
test.set <- folds[[1]];
training.set <- setdiff(1:n, test.set);
labels[test.set] <- 0;
res <- runSubnet(Yeast.STRING.data, labels, alpha=1, c=0, cost=0.0001);
str(res);
```

Index

∗ **package**
COSNet-package, 2

COSNet, 3, *6*
COSNet-package, 2
cosnet.cross.validation, 5

find.division.not.strat, 6, *7*
find.division.strat, *6*, 7

generate_labels, 8
generate_points, 9

optimize_pos_above, *11*, 11
optimizep, 10, *12*

reg_data, 13
runSubnet, 14