Package 'SEMdeep'

November 10, 2025

Title Structural Equation Modeling with Deep Neural Network and Machine Learning Algorithms

Version 1.1.0

Date 2025-10-14

Description Training and validation of a custom (or data-driven) Structural Equation Models using layer-wise Deep Neural Networks or node-wise Machine Learning algorithms, which extend the fitting procedures of the 'SEMgraph' R package <doi:10.32614/CRAN.package.SEMgraph>.

Depends SEMgraph (>= 1.2.2), igraph (>= 2.0.0), R (>= 4.0)

Imports coro, corpcor, kernelshap, lavaan, NeuralNetTools, parabar, progress, ranger, rpart, torch, xgboost

URL `https://github.com/BarbaraTarantino/SEMdeep`

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Mario Grassi [aut], Barbara Tarantino [cre]

Maintainer Barbara Tarantino <`barbara.tarantino01@universitadipavia.it`>

Repository CRAN

Date/Publication 2025-11-10 11:40:02 UTC

Contents

`classificationReport` *Prediction evaluation report of a classification model*

Description

This function builds a report showing the main classification metrics. It provides an overview of key evaluation metrics like precision, recall, F1-score, accuracy, Matthew's correlation coefficient (mcc) and support (testing size) for each class in the dataset and averages (macro or weighted) for all classes.

Usage

```
classificationReport(yobs, yhat, CM = NULL, verbose = FALSE, ...)
```

Arguments

`yobs`	A vector with the true target variable values.
`yhat`	A matrix with the predicted target variables values.
`CM`	An optional (external) confusion matrix CxC.
`verbose`	A logical value (default = FALSE). If TRUE, the confusion matrix is printed on the screen, and if C=2, the density plots of the predicted probability for each group are also printed.
`...`	Currently ignored.

Details

Given one vector with the true target variable labels, and the a matrix with the predicted target variable values for each class, a series of classification metrics is computed. For example, suppose a 2x2 table with notation

	Predicted	
Observed	Yes Event	No Event
Yes Event	A	C
No Event	B	D

The formulas used here for the label = "Yes Event" are:

$$pre = A/(A + B)$$
$$rec = A/(A + C)$$
$$F1 = (2 * pre * rec)/(pre + rec)$$
$$acc = (A + D)/(A + B + C + D)$$
$$mcc = (A * D - B * C)/sqrt((A + B) * (C + D) * (A + C) * (B + D))$$

Metrics analogous to those described above are calculated for the label "No Event", and the weighted average (averaging the support-weighted mean per label) and macro average (averaging the unweighted mean per label) are also provided.

Value

A list of 3 objects:

1. "CM", the confusion matrix between observed and predicted counts.
2. "stats", a data.frame with the classification evaluation statistics.
3. "cls", a data.frame with the predicted probabilities, predicted labels and true labels of the categorical target variable.

Author(s)

Barbara Tarantino <barbara.tarantino@unipv.it>

References

Sammut, C. & Webb, G. I. (eds.) (2017). Encyclopedia of Machine Learning and Data Mining. New York: Springer. ISBN: 978-1-4899-7685-7

Examples

```
# Load Sachs data (pkc)
ig<- sachs$graph
data<- sachs$pkc
data<- transformData(data)$data
group<- sachs$group

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))

#...with a categorical (as.factor) variable (C=2)
outcome<- factor(ifelse(group == 0, "control", "case"))
res<- SEMml(ig, data[train, ], outcome[train], algo="rf")
pred<- predict(res, data[-train, ], outcome[-train], verbose=TRUE)

yobs<- outcome[-train]
```

```
yhat<- pred$Yhat[ ,levels(outcome)]
cls<- classificationReport(yobs, yhat)
cls$CM
cls$stats
head(cls$cls)

#...with predicted probabiliy density plots, if C=2
cls<- classificationReport(yobs, yhat, verbose=TRUE)

#...with a categorical (as.factor) variable (C=3)
group[1:400]<- 2; table(group)
outcome<- factor(ifelse(group == 0, "control",
ifelse(group == 1, "case1", "case2")))
res<- SEMml(ig, data[train, ], outcome[train], algo="rf")
pred<- predict(res, data[-train, ], outcome[-train], verbose=TRUE)

yobs<- outcome[-train]
yhat<- pred$Yhat[ ,levels(outcome)]
cls<- classificationReport(yobs, yhat)
cls$CM
cls$stats
head(cls$cls)
```

---

crossValidation *Cross-validation of linear SEM, ML or DNN training models*

Description

The function does a R-repeated K-fold cross-validation of `SEMrun()`, `SEMml()` or `SEMdnn()` models.

Usage

```
crossValidation(
  models,
  outcome = NULL,
  K = 5,
  R = 1,
  metric = NULL,
  ncores = 2,
  verbose = FALSE,
  ...
)
```

Arguments

`models`	A named list of model fitting objects from `SEMrun()`, `SEMml()` or `SEMdnn()` function, with default group=NULL (for `SEMrun()` or outcome=NULL (for `SEMml()` or `SEMdnn()`).

`outcome`	A character vector (as.factor) of labels for a categorical output (target). If NULL (default), the categorical output (target) will not be considered.
`K`	A numerical value indicating the number of k-fold to create.
`R`	A numerical value indicating the number of repetitions for the k-fold cross-validation.
`metric`	A character value indicating the metric for boxplots display, i.e.: "amse", "r2", or "srmr", for continuous outcomes, and "f1", "accuracy" or "mcc", for a categorical outcome (default = NULL).
`ncores`	Number of cpu cores (default = 2).
`verbose`	Output to console boxplots and summarized results (default = FALSE).
`...`	Currently ignored.

Details

Easy-to-use model comparison and selection of SEM, ML or DNN models, in which several models are defined and compared in a R-repeated K-fold cross-validation procedure. The winner model is selected by reporting the mean predicted performances across all runs, as outline in de Rooij & Weeda (2020).

Value

A list of 2 objects: (1) "stats", a list with performance evaluation metrics. If `outcome=FALSE`, mean and (0.025;0.0975)-quantiles of amse, r2, and srmr across folds and repetitions are reported; if `outcome=TRUE`, mean and (0.025;0.0975)-quantiles of f1, accuracy and mcc from confusion matrix averaged across all repetitions are reported; and (2) "PE", a data.frame of repeated cross-validation results.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

de Rooij M, Weeda W. Cross-Validation: A Method Every Psychologist Should Know. Advances in Methods and Practices in Psychological Science. 2020;3(2):248-263. doi:10.1177/2515245919898466

Examples

```
# Load Amyotrophic Lateral Sclerosis (ALS)
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data
group<- alsData$group

# ... with continuous outcomes

res1 <- SEMml(ig, data, algo="tree")
res2 <- SEMml(ig, data, algo="rf")
```

```
res3 <- SEMml(ig, data, algo="xgb")
res4 <- SEMml(ig, data, algo="sem")

models <- list(res1,res2,res3,res4)
names(models) <- c("tree","rf","xgb","sem")

res.cv1 <- crossValidation(models, outcome=NULL, K=5, R=10)
print(res.cv1$stats)

#... with a categorical (as.factor) outcome

outcome <- factor(ifelse(group == 0, "control", "case"))
res.cv2 <- crossValidation(models, outcome=outcome, K=5, R=10)
print(res.cv2$stats)
```

`getConnectionWeight` *Connection Weight method for neural network variable importance*

Description

The function computes the matrix multiplications of hidden weight matrices (Wx,...,Wy), i.e., the product of the raw input-hidden and hidden-output connection weights between each input and output neuron and sums the products across all hidden neurons, as proposed by Olden (2002; 2004).

Usage

```
getConnectionWeight(object, thr = NULL, verbose = FALSE, ...)
```

Arguments

`object`	A neural network object from `SEMdnn()` function.
`thr`	A numeric value [0-1] indicating the threshold to apply to the Olden's connection weights to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(connection weights)).
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

In a neural network, the connections between inputs and outputs are represented by the connection weights between the neurons. The importance values assigned to each input variable using the Olden method are in units that are based directly on the summed product of the connection weights. The amount and direction of the link weights largely determine the proportional contributions of the input variables to the neural network's prediction output. Input variables with larger connection weights indicate higher intensities of signal transfer and are therefore more important in the prediction process. Positive connection weights represent excitatory effects on neurons (raising the intensity of the incoming signal) and increase the value of the predicted response, while negative connection weights represent inhibitory effects on neurons (reducing the intensity of the incoming signal). The weights that change sign (e.g., positive to negative) between the input-hidden to hidden-output layers would have a cancelling effect, and vice versa weights with the same sign would have a synergistic effect. Note that in order to map the connection weights to the DAG edges, the element-wise product, W*A is performed between the Olden's weights entered in a matrix, W(pxp) and the binary (1,0) adjacency matrix, A(pxp) of the input DAG.

Value

A list of three object: (i) est: a data.frame including the connections together with their connection weights(W), (ii) gest: if the outcome vector is given, a data.frame of connection weights for outcome lavels, and (iii) dag: DAG with colored edges/nodes. If abs(W) > thr and W < 0, the edge W > 0, the edge is activated and it is highlighted in red. If the outcome vector is given, nodes with absolute connection weights summed over the outcome levels, i.e. sum(abs(W[outcome levels])) > thr, will be highlighted in pink.

Author(s)

Mario Grassi `<mario.grassi@unipv.it>`

References

Olden, Julian & Jackson, Donald. (2002). Illuminating the "black box": A randomization approach for understanding variable contributions in artificial neural networks. Ecological Modelling, 154(1-2): 135-150. https://doi.org/10.1016/S0304-3800(02)00064-9

Olden, Julian; Joy, Michael K; Death, Russell G (2004). An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. Ecological Modelling, 178 (3-4): 389-397. https://doi.org/10.1016/j.ecolmodel.2004.03.013

Examples

```
if (torch::torch_is_installed()){

# Load Sachs data (pkc)
ig<- sachs$graph
data<- sachs$pkc
data<- log(data)

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))
#ncores<- parallel::detectCores(logical = FALSE)

dnn0<- SEMdnn(ig, data[train, ], outcome = NULL, algo= "structured",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)

cw<- getConnectionWeight(dnn0, thr = 0.3, verbose = FALSE)
gplot(cw$dag, l="circo")
table(E(cw$dag)$color)
}
```

getGradientWeight *Gradient Weight method for neural network variable importance*

Description

The function computes the gradient matrix, i.e., the average marginal effect of the input variables w.r.t the neural network model, as discussed by Scholbeck et al (2024).

Usage

```
getGradientWeight(object, thr = NULL, verbose = FALSE, ...)
```

Arguments

`object`	A neural network object from `SEMdnn()` function.
`thr`	A numeric value [0-1] indicating the threshold to apply to the gradient weights to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(gradient weights)).
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

The gradient weights method approximate the derivative (the gradient) of each output variable (y) with respect to each input variable (x) evaluated at each observation (i=1,...,n) of the training data. The contribution of each input is evaluated in terms of both magnitude taking into account not only the connection weights and activation functions, but also the values of each observation of the input variables. Once the gradients for each variable and observation, a summary gradient is calculated by averaging over the observation units. Finally, the average weights are entered into a matrix, W(pxp) and the element-wise product with the binary (1,0) adjacency matrix, A(pxp) of the input DAG, W*A maps the weights on the DAG edges. Note that the operations required to approsimate partial derivatives are time consuming compared to other methods such as Olden's (connection weight). The computational time increases with the size of the neural network or the size of the data.

Value

A list of three object: (i) est: a data.frame including the connections together with their gradient weights, (ii) gest: if the outcome vector is given, a data.frame of gradient weights for outcome lavels, and (iii) dag: DAG with colored edges/nodes. If abs(grad) > thr and grad < 0, the edge is inhibited and it is highlighted in blue; otherwise, if abs(grad) > thr and grad > 0, the edge is activated and it is highlighted in red. If the outcome vector is given, nodes with absolute connection weights summed over the outcome levels, i.e. sum(abs(grad[outcome levels])) > thr, will be highlighted in pink.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

Scholbeck, C.A., Casalicchio, G., Molnar, C. et al. Marginal effects for non-linear prediction functions. Data Min Knowl Disc 38, 2997–3042 (2024). https://doi.org/10.1007/s10618-023-00993-x

Examples

```
if (torch::torch_is_installed()){

# Load Sachs data (pkc)
ig<- sachs$graph
data<- sachs$pkc
data<- log(data)

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))
#ncores<- parallel::detectCores(logical = FALSE)

dnn0<- SEMdnn(ig, data[train, ], outcome = NULL, algo= "neuralgraph",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)

gw<- getGradientWeight(dnn0, thr = 0.3, verbose = FALSE)
gplot(gw$dag, l="circo")
table(E(gw$dag)$color)
}
```

---

getLOCO *Compute variable importance using LOCO values*

Description

This function computes the contributions of each variable to individual predictions using LOCO (Leave Out COvariates) values.

Usage

```
getLOCO(
  object,
  newdata,
  newoutcome = NULL,
  thr = NULL,
  ncores = 2,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	A model fitting object from `SEMml()`, or `SEMrun()` functions.
`newdata`	A matrix containing new data with rows corresponding to subjects, and columns to variables.
`newoutcome`	A new character vector (as.factor) of labels for a categorical output (target)(default = NULL).
`thr`	A numeric value [0-1] indicating the threshold to apply to the LOCO values to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(LOCO values)).
`ncores`	number of cpu cores (default = 2)
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

LOCO (Verdinelli & Wasserman, 2024) is a model-agnostic method for assessing the importance of individual features (covariates) in a ML predictive model. The procedure is simple: (i) train a model on the full dataset (with all covariates) and (ii) for each covariate of interest: (a) remove (leave out) that covariate from the dataset; (b) retrain the model on the remaining features; (c) compare predictions between the full model and the reduced mode, and (d) evaluate the difference in performance (e.g., using MSE, etc.). LOCO is computationally expensive (requires retraining for each feature). The getLOCO() function uses a lowest computation cost procedure (see Delicando & Pena, 2023). The individual relevance of each variable is measured by comparing the predictions of the model in the test set with those obtained when the variable of interest is leave-out and substituted by its ghost variable in the test set. This ghost variable is defined as the linear prediction of the covariate by using the rest of the variables in the ML model. This method yields similar LOCO results but requires much less computing time.

Value

A list od three object: (i) est: a data.frame including the connections together with their LOCO values; (iii) gest: if the outcome vector is given, a data.frame of LOCO values per outcome levels; and (iii) dag: DAG with colored edges/nodes. If LOCO > thr, the edge is highlighted in red. If the outcome vector is given, nodes with absolute connection weights summed over the outcome levels, i.e. sum(LOCO[outcome levels]) > thr, will be highlighted in pink.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

Verdinelli, I; Wasserman, L. Feature Importance: A Closer Look at Shapley Values and LOCO. Statist. Sci. 39 (4) 623 - 636, November 2024. https://doi.org/10.1214/24-STS937

Delicado, P.; Peña, D. Understanding complex predictive models with ghost variables. TEST 32, 107–145 (2023). https://doi.org/10.1007/s11749-022-00826-x

Examples

```

# load ALS data
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))

rf0<- SEMml(ig, data[train, ], algo="rf")

res<- getLOCO(rf0, data[-train, ], thr=0.2, verbose=TRUE)
table(E(res$dag)$color)
```

getShapleyR2 *Compute variable importance using Shapley (R2) values*

Description

This function computes a model-agnostic variable importance based on a Shapley-value decomposition of the model R-Squared (R2, i.e., the coefficient of determination) that allocates the proportion of model- explained variability in the data to each model feature (Redell, 2019).

Usage

```
getShapleyR2(
  object,
  newdata,
  newoutcome = NULL,
  thr = NULL,
  ncores = 2,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	A model fitting object from `SEMml()`, or `SEMrun()` functions.
`newdata`	A matrix containing new data with rows corresponding to subjects, and columns to variables.
`newoutcome`	A new character vector (as.factor) of labels for a categorical output (target)(default = NULL).
`thr`	A numeric value [0-1] indicating the threshold to apply to the signed Shapley R2 to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(signed Shapley R2 values)).
`ncores`	number of cpu cores (default = 2)
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

Shapley values (Shapley, 1953; Lundberg & Lee, 2017) apply the fair distribution of payoffs principles from game theory to measure the additive contribution of individual predictors in a ML model. The function compute a signed Shapley R2 metric, that combines the additive property of Shapley values with the robustness of the R-squared (R2) of Gelman (2018) to produce a variance decomposition that accurately captures the contribution of each variable in the ML model, see Redell (2019). The signed values are used in order to denote the regulation of connections in line with a linear model, since the edges in the DAG indicate node regulation (activation, if positive; inhibition, if negative). The sign has been recovered for each edge using sign(beta), i.e., the sign of the coefficient estimates from a linear model (lm) fitting of the output node on the input nodes (see Joseph, 2019). Furthermore, to determine the local significance of node regulation in the DAG, the Shapley decomposition of the R-squared values for each outcome node (r=1,...,R) can be done by summing the Shapley R2 indices of their input nodes. It should be noted that the operations required to compute Shapley values processed with the `kernelshap` function of the **kernelshap** R package are inherently time-consuming, with the computational time increasing in proportion to the number of predictor variables and the number of observations.

Value

A list od four object: (i) shapx: the list of individual Shapley values of predictors variables per each response variable; (ii) est: a data.frame including the connections together with their signed Shapley R-squred values; (iii) gest: if the outcome vector is given, a data.frame of signed Shapley R-squred values per outcome levels; and (iv) dag: DAG with colored edges/nodes. If abs(sign_r2) > thr and sign_r2 < 0, the edge is inhibited and it is highlighted in blue; otherwise, if abs(sign_r2) > thr and sign_r2 > 0, the edge is activated and it is highlighted in red. If the outcome vector is given, nodes with absolute connection weights summed over the outcome levels, i.e. sum(abs(sign_r2[outcome levels])) > thr, will be highlighted in pink.

Author(s)

Mario Grassi `<mario.grassi@unipv.it>`

References

Shapley, L. (1953) A Value for n-Person Games. In: Kuhn, H. and Tucker, A., Eds., Contributions to the Theory of Games II, Princeton University Press, Princeton, 307-317.

Scott M. Lundberg, Su-In Lee. (2017). A unified approach to interpreting model predictions. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 4768–4777.

Redell, N. (2019). Shapley Decomposition of R-Squared in Machine Learning Models. arXiv preprint: https://doi.org/10.48550/arXiv.1908.09718

Gelman, A., Goodrich, B., Gabry, J., & Vehtari, A. (2019). R-squared for Bayesian Regression Models. The American Statistician, 73(3), 307–309.

Joseph, A. Parametric inference with universal function approximators (2019). Bank of England working papers 784, Bank of England, revised 22 Jul 2020.

Examples

```
# load ALS data
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))

rf0<- SEMml(ig, data[train, ], algo="rf")

res<- getShapleyR2(rf0, data[-train, ], thr=NULL, verbose=TRUE)
table(E(res$dag)$color)

# shapley R2 per response variables
R2<- abs(res$est[,4])
Y<- res$est[,1]
R2Y<- aggregate(R2~Y,data=data.frame(R2,Y),FUN="sum");R2Y
r2<- mean(R2Y$R2);r2
```

`getSignificanceTest` *Test for the significance of neural network input nodes*

Description

The function computes a formal test for the significance of neural network input nodes, based on a linear relationship between the observed output and the predicted values of an input variable, when all other input variables are maintained at their mean (or zero) values, as proposed by Mohammadi (2018).

Usage

```
getSignificanceTest(object, thr = NULL, verbose = FALSE, ...)
```

Arguments

`object`	A neural network object from `SEMdnn()` function.
`thr`	A numeric value [0-1] indicating the threshold to apply to the t-test values to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(t-test values)).
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

A neural network is trained, taking into account the number of hidden layers, neurons, and activation function. Then, network's output is simulated to get the predicted values of the output variable, fixing all the inputs (with the exception of one nonconstant input variable) at their mean values. Subsequently, the network's predictions are stored after this process is completed for each input variable. As last step, multiple regression analysis is applied node-wise (mapping the input DAG) on the observed output nodes with the predicted values of the input nodes as explanatory variables. The statistical significance of the coefficients is evaluated using standard t-student values, which represent the importance of the input variables.

Value

A list of three object: (i) est: a data.frame including the connections together with their t_test weights, (ii) gest: if the outcome vector is given, a data.frame of t_test weights for outcome lavels, and (iii) dag: DAG with colored edges/nodes. If abs(t_test) > thr and t_test < 0, the edge is inhibited and it is highlighted in blue; otherwise, if abs(t_test) > thr and t_test > 0, the edge is activated and it is highlighted in red. If the outcome vector is given, nodes with absolute connection weights summed over the outcome levels, i.e. sum(abs(t_test[outcome levels])) > thr, will be highlighted in pink.

Author(s)

Mario Grassi `<mario.grassi@unipv.it>`

References

S. Mohammadi. A new test for the significance of neural network inputs. Neurocomputing 2018; 273: 304-322. https://doi.org/10.1016/j.neucom.2017.08.007

Examples

```
if (torch::torch_is_installed()){

# Load Sachs data (pkc)
ig<- sachs$graph
data<- sachs$pkc
data<- log(data)

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))
#ncores<- parallel::detectCores(logical = FALSE)

dnn0<- SEMdnn(ig, data[train, ], outcome = NULL, algo= "nodewise",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)

st<- getSignificanceTest(dnn0, thr = NULL, verbose = FALSE)
gplot(st$dag, l="circo")
table(E(st$dag)$color)
}
```

getVariableImportance *Variable importance for Machine Learning models*

Description

Extraction of ML variable importance measures.

Usage

```
getVariableImportance(object, thr = NULL, verbose = FALSE, ...)
```

Arguments

object	A model fitting object from SEMml() function.
thr	A numeric value [0-1] indicating the threshold to apply to the variable importance values to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(variable importance values)).
verbose	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
...	Currently ignored.

Details

The variable (predictor) importance will be computed considering: (i) the absolute value of the z-statistic of the model parameters for "sem"; and (ii) the variable importance measures from the rpart, importance or xgb.importance functions for "tree", "rf" or "xgb" methods.

Value

A list of three object: (i) est: a data.frame including the connections together with their variable importances (VarImp)), (ii) gest: if the outcome vector is given, a data.frame of VarImp for outcome lavels, and (iii) dag: DAG with colored edges/nodes. If abs(VarImp) > thr will be highlighted in red (VarImp > 0) or blue (VarImp < 0). If the outcome vector is given, nodes with variable importances summed over the outcome levels, i.e. sum(VarImp[outcome levels])) > thr, will be highlighted in pink.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

add references

Examples

```
# load ALS data
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data

#ncores<- parallel::detectCores(logical = FALSE)
ml0<- SEMml(ig, data, outcome=NULL, algo="rf", ncores=2)

vi05<- getVariableImportance(ml0, thr=0.5, verbose=TRUE)
table(E(vi05$dag)$color)
```

---

mapGraph *Map additional variables (nodes) to a graph object*

Description

The function insert additional nodes to a graph object. Among the node types, additional source or sink nodes can be added. Source nodes can represent: (i) data variables; (ii) a group variable; (iii) latent variables (LV). Vice versa, sink nodes represent the levels of a categorical outcome variable and are linked with all graph nodes. Moreover, `mapGraph()` can also create a new graph object starting from a compact symbolic formula.

Usage

```
mapGraph(graph, type, C = NULL, LV = NULL, f = NULL, verbose = FALSE, ...)
```

Arguments

`graph`	An igraph object.
`type`	A character value specifying the type of mapping. Five types can be specified. 1. "source", source nodes are linked to sink nodes of the graph. 2. "group", an additional group source node is added to the graph. 3. "outcome", additional c=1,2,...,C sink nodes are added to the graph. 4. "LV", additional latent variable (LV) source nodes are added to the graph. 5. "clusterLV", a series of clusters for the data are computed and a different LV source node is added separately for each cluster.
`C`	the number of labels of the categorical sink node (default = NULL).
`LV`	The number of LV source nodes to add to the graph. This argument needs to be specified when `type = "LV"`. When `type = "clusterLV"` the LV number is defined internally equal to the number of clusters. (default = NULL).
`f`	A formula object (default = NULL). A new graph object is created according to the specified formula object.
`verbose`	If TRUE disply the mapped graph (default = FALSE)
`...`	Currently ignored.

Value

mapGraph returns invisibly the graphical object with the mapped node variables.

Author(s)

Mario Grassi `<mario.grassi@unipv.it>`

Examples

```
# Load Amyotrophic Lateral Sclerosis (ALS)
ig<- alsData$graph; gplot(ig)

# ... map source nodes to sink nodes of ALS graph
ig1 <- mapGraph(ig, type = "source"); gplot(ig1, l="dot")

# ... map group source node to ALS graph
ig2 <- mapGraph(ig, type = "group"); gplot(ig2, l="fdp")

# ... map outcome sink (C=2) to ALS graph
ig3 <- mapGraph(ig, type = "outcome", C=2); gplot(ig3, l="fdp")

# ... map LV source nodes to ALS graph
ig4 <- mapGraph(ig, type = "LV", LV = 3); gplot(ig4, l="fdp")

# ... map LV source nodes to the cluster nodes of ALS graph
ig5 <- mapGraph(ig, type = "clusterLV"); gplot(ig5, l="dot")

# ... create a new graph with the formula variables
formula <- as.formula("z4747 ~ z1432 + z5603 + z5630")
ig6 <- mapGraph(f=formula); gplot(ig6)
```

nplot *Create a plot for a neural network model*

Description

The function uses the `plotnet` function of the **NeuralNetTools** R package to draw a neural network plot and visualize the hidden layer structure.

Usage

```
nplot(object, hidden, bias = TRUE, sleep = 2, ...)
```

Arguments

`object`	A neural network model object
`hidden`	The hidden structure of the object
`bias`	A logical value, indicating whether to draw biases in the layers (default = FALSE).
`sleep`	Suspend plot display for a specified time (in secs, default = 2).
`...`	Currently ignored.

Details

The induced subgraph of the input graph mapped on data variables. Based on the estimated connection weights, if the connection weight W > 0, the connection is activated and it is highlighted in red; if W < 0, the connection is inhibited and it is highlighted in blue.

Value

The function invisibly returns the graphical objects representing the neural network architecture designed by NeuralNetTools.

Author(s)

Mario Grassi `<mario.grassi@unipv.it>`

References

Beck, M.W. 2018. NeuralNetTools: Visualization and Analysis Tools for Neural Networks. Journal of Statistical Software. 85(11):1-20.

Examples

```
if (torch::torch_is_installed()){

# load ALS data
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data

#ncores<- parallel::detectCores(logical = FALSE)
dnn0 <- SEMdnn(ig, data, train=1:nrow(data), algo = "layerwise",
hidden = c(10, 10, 10), link = "selu", bias =TRUE,
epochs = 32, patience = 10, verbose = TRUE)

 #Visualize the neural networks per each layer of dnn0
 nplot(dnn0, hidden = c(10, 10, 10), bias = FALSE)
}
```

`predict.DNN` *SEM-based out-of-sample prediction using DNN*

Description

Predict method for DNN objects.

Usage

```
## S3 method for class 'DNN'
predict(object, newdata, newoutcome = NULL, verbose = FALSE, ...)
```

Arguments

`object`	A model fitting object from `SEMdnn()` function.
`newdata`	A matrix containing new data with rows corresponding to subjects, and columns to variables. If newdata = NULL, the train data are used.
`newoutcome`	A new character vector (as.factor) of labels for a categorical output (target) (default = NULL).
`verbose`	Print predicted out-of-sample MSE values (default = FALSE).
`...`	Currently ignored.

Value

A list of three objects:

1. "PE", vector of the amse = average MSE over all (sink and mediators) graph nodes; r2 = 1 - amse; and srmr= Standardized Root Means Square Residual between the out-of-bag correlation matrix and the model correlation matrix.
2. "mse", vector of the Mean Squared Error (MSE) for each out-of-bag prediction of the sink and mediators graph nodes.
3. "Yhat", the matrix of continuous predicted values of graph nodes (excluding source nodes) based on out-of-bag samples.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

Examples

```
if (torch::torch_is_installed()){

# Load Amyotrophic Lateral Sclerosis (ALS)
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data
group<- alsData$group

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))
#ncores<- parallel::detectCores(logical = FALSE)

start<- Sys.time()
dnn0 <- SEMdnn(ig, data[train, ], algo ="layerwise",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)
end<- Sys.time()
print(end-start)
pred.dnn <- predict(dnn0, data[-train, ], verbose=TRUE)

# SEMrun vs. SEMdnn MSE comparison
sem0 <- SEMrun(ig, data[train, ], algo="ricf", n_rep=0)
pred.sem <- predict(sem0, data[-train,], verbose=TRUE)

#...with a categorical (as.factor) outcome
outcome <- factor(ifelse(group == 0, "control", "case")); table(outcome)

start<- Sys.time()
dnn1 <- SEMdnn(ig, data[train, ], outcome[train], algo ="layerwise",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)
end<- Sys.time()
print(end-start)

pred <- predict(dnn1, data[-train, ], outcome[-train], verbose=TRUE)
yhat <- pred$Yhat[ ,levels(outcome)]; head(yhat)
yobs <- outcome[-train]; head(yobs)
classificationReport(yobs, yhat, verbose=TRUE)$stats
}
```

predict.ML *SEM-based out-of-sample prediction using nodewise ML*

Description

Predict method for ML objects.

Usage

```
## S3 method for class 'ML'
predict(object, newdata, newoutcome = NULL, ncores = 2, verbose = FALSE, ...)
```

Arguments

`object`	A model fitting object from `SEMml()` function.
`newdata`	A matrix containing new data with rows corresponding to subjects, and columns to variables.
`newoutcome`	A new character vector (as.factor) of labels for a categorical output (target)(default = NULL).
`ncores`	number of cpu cores (default = 2)
`verbose`	Print predicted out-of-sample MSE values (default = FALSE).
`...`	Currently ignored.

Value

A list of 3 objects:

1. "PE", vector of the amse = average MSE over all (sink and mediators) graph nodes; r2 = 1 - amse; and srmr= Standardized Root Means Squared Residual between the out-of-bag correlation matrix and the model correlation matrix.
2. "mse", vector of the Mean Squared Error (MSE) for each out-of-bag prediction of the sink and mediators graph nodes.
3. "Yhat", the matrix of continuous predicted values of graph nodes (excluding source nodes) based on out-of-bag samples.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

Examples

```
# Load Amyotrophic Lateral Sclerosis (ALS)
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data
group<- alsData$group

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))

start<- Sys.time()
# ... tree
res1<- SEMml(ig, data[train, ], algo="tree")
mse1<- predict(res1, data[-train, ], verbose=TRUE)

# ... rf
res2<- SEMml(ig, data[train, ], algo="rf")
mse2<- predict(res2, data[-train, ], verbose=TRUE)

# ... xgb
res3<- SEMml(ig, data[train, ], algo="xgb")
mse3<- predict(res3, data[-train, ], verbose=TRUE)

# ... sem
res4<- SEMml(ig, data[train, ], algo="sem")
mse4<- predict(res4, data[-train, ], verbose=TRUE)
end<- Sys.time()
print(end-start)

#...with a categorical (as.factor) outcome
outcome <- factor(ifelse(group == 0, "control", "case")); table(outcome)

res5 <- SEMml(ig, data[train, ], outcome[train], algo="tree")
pred <- predict(res5, data[-train, ], outcome[-train], verbose=TRUE)
yhat <- pred$Yhat[ ,levels(outcome)]; head(yhat)
yobs <- outcome[-train]; head(yobs)
classificationReport(yobs, yhat, verbose=TRUE)$stats
```

predict.SEM *SEM-based out-of-sample prediction using layer-wise ordering*

Description

Given the values of (observed) x-variables in a SEM, this function may be used to predict the values of (observed) y-variables. The predictive procedure consists of two steps. First, the topological layer ordering of the input graph is defined. Then, the node y values in a layer are predicted, where the nodes in successive layers act as x-predictors.

Usage

```
## S3 method for class 'SEM'
predict(object, newdata, newoutcome = NULL, verbose = FALSE, ...)
```

Arguments

`object`	An object, as that created by the function `SEMrun()` with the argument `group` set to the default `group = NULL`.
`newdata`	A matrix with new data, with rows corresponding to subjects, and columns to variables.
`newoutcome`	A new character vector (as.factor) of labels for a categorical output (target)(default = NULL).
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

The function first creates a layer-based structure of the input graph. Then, a SEM-based predictive approach (Rooij et al., 2022) is used to produce predictions while accounting for the graph structure based on the topological layer (j=1,. . . ,L) of the input graph. In each iteration, the response (output) variables, y are the nodes in the j=1,...,(L-1) layer and the predictor (input) variables, x are the nodes belonging to the successive, (j+1),...,L layers. Predictions (for y given x) are based on the (joint y and x) model-implied variance-covariance (Sigma) matrix and mean vector (Mu) of the fitted SEM, and the standard expression for the conditional mean of a multivariate normal distribution. Thus, the layer structure described in the SEM is taken into consideration, which differs from ordinary least squares (OLS) regression.

Value

A list of 3 objects:

1. "PE", vector of the amse = average MSE over all (sink and mediators) graph nodes; r2 = 1 - amse; and srmr= Standardized Root Means Square Residual between the out-of-bag correlation matrix and the model correlation matrix.
2. "mse", vector of the Mean Squared Error (MSE) for each out-of-bag prediction of the sink and mediators graph nodes.
3. "Yhat", the matrix of continuous predicted values of graph nodes (excluding source nodes) based on out-of-bag samples.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

de Rooij M, Karch JD, Fokkema M, Bakk Z, Pratiwi BC, and Kelderman H (2023). SEM-Based Out-of-Sample Predictions, Structural Equation Modeling: A Multidisciplinary Journal, 30:1, 132-148. <https://doi.org/10.1080/10705511.2022.2061494>

Grassi M, Palluzzi F, Tarantino B (2022). SEMgraph: An R Package for Causal Network Analysis of High-Throughput Data with Structural Equation Models. Bioinformatics, 38 (20), 4829–4830. <https://doi.org/10.1093/bioinformatics/btac567>

Grassi, M., Tarantino, B. (2025). SEMdag: Fast learning of Directed Acyclic Graphs via node or layer ordering. PLoS ONE 20(1): e0317283. https://doi.org/10.1371/journal.pone.0317283

Examples

```
# load ALS data
data<- alsData$exprs
data<- transformData(data)$data
group<- alsData$group

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))

# predictors, source+mediator; outcomes, mediator+sink

ig <- alsData$graph; gplot(ig)
sem0 <- SEMrun(ig, data[train,], algo="ricf", n_rep=0)
pred0 <- predict(sem0, newdata=data[-train,], verbose=TRUE)

# predictors, source+mediator+group; outcomes, source+mediator+sink

ig1 <- mapGraph(ig, type = "group"); gplot(ig1)
data1 <- cbind(group, data); head(data1[,5])
sem1 <- SEMrun(ig1, data1[train,], algo="ricf", n_rep=0)
pred1 <- predict(sem1, newdata= data1[-train,], verbose=TRUE)

# predictors, source nodes; outcomes, sink nodes

ig2 <- mapGraph(ig, type = "source"); gplot(ig2)
sem2 <- SEMrun(ig2, data[train,], algo="ricf", n_rep=0)
pred2 <- predict(sem2, newdata=data[-train,], verbose=TRUE)
```

SEMdnn *SEM train with Deep Neural Netwok (DNN) models*

Description

The function builds four Deep Neural Networks (DNN) models based on the topological structure of the input graph using the 'torch' language. The **torch** package is native to R, so it's computationally efficient and the installation is very simple, as there is no need to install Python or any other API, and DNNs can be trained on CPU, GPU and MacOS GPUs.

In order to install **torch** please follow these steps:

`install.packages("torch")`

`library(torch)`

`install_torch(reinstall = TRUE)`

For setup GPU or if you have problems installing **torch** package, check out the installation help from the torch developer.

Usage

```
SEMdnn(
  graph,
  data,
  outcome = NULL,
  algo = "layerwise",
  hidden = c(10L, 10L, 10L),
  link = "selu",
  bias = TRUE,
  dropout = 0,
  loss = "mse",
  validation = 0,
  lambda = 0,
  alpha = 0.5,
  optimizer = "adam",
  lr = 0.01,
  batchsize = NULL,
  burnin = 30,
  thr = NULL,
  nboot = 0,
  epochs = 100,
  patience = 100,
  device = "cpu",
  verbose = FALSE,
  ...
)
```

Arguments

`graph`	An igraph object.
`data`	A matrix with rows corresponding to subjects, and columns to graph nodes (variables).
`outcome`	A character vector (as.factor) of labels for a categorical output (target). If NULL (default), the categorical output (target) will not be considered.
`algo`	A character value, indicating the DNN algorithm: "nodewise", "layerwise" (default), "structured", or "neuralgraph" (see details).
`hidden`	hidden units in layers; the number of layers corresponds with the length of the hidden units. As a default, hidden = c(10L, 10L, 10L).
`link`	A character value describing the activation function to use, which might be a single length or be a vector with many activation functions assigned to each layer. As a default, link = "selu".
`bias`	A logical vector, indicating whether to employ biases in the layers, which can be either vectors of logicals for each layer (number of hidden layers + 1 (final layer)) or of length one. As a default, bias = TRUE.
`dropout`	A numerical value for the dropout rate, which is the probability that a node will be excluded from training. As a default, dropout = 0.
`loss`	A character value specifying the at which the network should be optimized. For regression problem used in SEMdnn(), the user can specify: (a) "mse" (mean squared error), "mae" (mean absolute error), or "nnl" (negative log-likelihood). As a default, loss = "mse".
`validation`	A numerical value indicating the proportion of the data set that should be used as a validation set (randomly selected, default = 0).
`lambda`	A numerical value, indicating the strength of the regularization, λ(L1 + L2) for lambda penalty (default = 0).
`alpha`	A numerical value, add L1/L2 regularization into the training. Set the alpha parameter for each layer to (1-α)L1 + αL2. It must fall between 0 and 1 (default = 0.5).
`optimizer`	A character value, indicating the optimizer to use for training the network. The user can specify: "adam" (ADAM algorithm), "adagrad" (adaptive gradient algorithm), "rmsprop" (root mean squared propagation), "rprop" (resilient backpropagation), "sgd" (stochastic gradient descent). As a default, optimizer = "adam".
`lr`	A numerical value, indicating the learning rate given to the optimizer (default = 0.01).
`batchsize`	Number of samples that are used to calculate one learning rate step (default = 1/10 of the training data).
`burnin`	Training is aborted if the trainings loss is not below the baseline loss after burnin epochs (default = 30).
`thr`	A numeric value [0-1] indicating the threshold to apply to the Olden's connection weights to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(connection weights)).

`nboot`	number of bootstrap samples that will be used to compute cheap (lower, upper) CIs for all input variable weights. As a default, nboot = 0.
`epochs`	A numerical value indicating the epochs during which the training is conducted (default = 100).
`patience`	A numeric value, training will terminate if the loss increases over a predetermined number of consecutive epochs and apply validation loss when available. Default patience = 100, no early stopping is applied.
`device`	A character value describing the CPU/GPU device ("cpu", "cuda", "mps") on which the neural network should be trained on. As a default, device = "cpu".
`verbose`	The training loss values of the DNN model are displayed as output, comparing the training, validation and baseline in the last epoch (default = FALSE).
`...`	Currently ignored.

Details

Four Deep Neural Networks (DNNs) are trained with `SEMdnn()`.

If algo = "nodewise", a set of DNN models is performed equation-by-equation (r=1,...,R) times, where R is the number of response (outcome) variables (i.e., nodes in the input graph with non-zero incoming connectivity) and predictor (input) variables are nodes with a direct edge to the outcome nodes, as poposed by various authors in causal discovery methods (see Zheng et al, 2020). Note, that model learning can be time-consuming for large graphs and large R outcomes.

If algo = "layerwise" (default), a set of DNN models is defined based on the topological layer structure (j=1,...,L) from sink to source nodes of the input graph. In each iteration, the response (output) variables, y are the nodes in the j=1,...,(L-1) layer, and the predictor (input) variables, x are the nodes belonging to successive: (j+1),...,L layers, which are linked with a direct edge to the response variables (see Grassi & Tarantino, 2025).

If algo = "structured", a Structured Neural Network (StrNN) is defined with input and output units equal to D, the number of the nodes. The algorithm uses the prior knowledge of the input graph to build the neural network architecture via a per-layer masking of the neural weights (i.e., W1 * M1, W2 * M2, ..., WL *ML), with the constraint that (W1 * M1) x (W2 * M2) x ... x (WL * ML) = A, where A is the adjacency matrix of the input graph (see Chen et al, 2023).

If algo = "neuralgraph", a Neural Graphical Model (NGM) is generated. As StrNN input and output units are equal to D, the number of the nodes. The prior knowledge of the input graph is used to compute the product of the absolute value of the neural weights (i.e., W = |W1| x |W2| x ... x |WL|), under the constraint that log(W * Ac) = 0, where Ac represents the complement of the adjacency matrix A of input graph, which essentially replaces 0 by 1 and vice-versa (see Shrivastava & Chajewska, 2023).

Each DNN model (R for "nodewise", L<R for "layerwise", and 1 for "structured" and "neuralgraph") is a Multilayer Perceptron (MLP) network, where every neuron node is connected to every other neuron node in the hidden layer above and every other hidden layer below. Each neuron's value is determined by calculating a weighted summation of its outputs from the hidden layer before it, and then applying an activation function. The calculated value of every neuron is used as the input for the neurons in the layer below it, until the output layer is reached.

If boot != 0, the function will implement the cheap bootstrapping proposed by Lam (2002) to generate uncertainties (i.e., bootstrap `90%CIs`) for DNN parameters. Bootstrapping can be enabled by setting a small number (1 to 10) of bootstrap samples. Note, however, that the computation can be time-consuming for massive DNNs, even with cheap bootstrapping!

Value

An S3 object of class "DNN" is returned. It is a list of 5 objects:

1. "fit", a list of DNN model objects, including: the estimated covariance matrix (Sigma), the estimated model errors (Psi), the fitting indices (fitIdx), and the parameterEstimates, i.e., the data.frame of Olden's connection weights.
2. "gest", the data.frame of estimated connection weights (parameterEstimates) of outcome levels, if outcome != NULL.
3. "model", a list of all MLP network models fitted by torch.
4. "graph", the induced DAG of the input graph mapped on data variables. The DAG is colored based on the Olden's connection weights (W), if abs(W) > thr and W < 0, the edge is inhibited and it is highlighted in blue; otherwise, if abs(W) > thr and W > 0, the edge is activated and it is highlighted in red. If the outcome vector is given, nodes with absolute connection weights summed over the outcome levels, i.e. sum(abs(W[outcome levels])) > thr, will be highlighted in pink.
5. "data", input data subset mapping graph nodes.

Author(s)

Mario Grassi `<mario.grassi@unipv.it>`

References

Zheng, X., Dan, C., Aragam, B., Ravikumar, P., Xing E. (2020). Learning sparse nonparametric dags. International conference on artificial intelligence and statistics, PMLR, 3414-3425. https://doi.org/10.48550/arXiv.1909

Grassi, M., Tarantino, B. (2025). SEMdag: Fast learning of Directed Acyclic Graphs via node or layer ordering. PLoS ONE 20(1): e0317283. https://doi.org/10.1371/journal.pone.0317283

Chen A., Shi, R.I., Gao, X., Baptista, R., Krishnan, R.G. (2023). Structured neural networks for density estimation and causal inference. Advances in Neural Information Processing Systems, 36, 66438-66450. https://doi.org/10.48550/arXiv.2311.02221

Shrivastava, H., Chajewska, U. (2023). Neural graphical models. In European Conference on Symbolic and Quantitative Approaches with Uncertainty (pp. 284-307). Cham: Springer Nature Switzerland. https://doi.org/10.48550/arXiv.2210.00453

Lam, H. (2022). Cheap Bootstrap for Input Uncertainty Quantification. Winter Simulation Conference (WSC), Singapore, 2022, pp. 2318-2329. https://doi.org/10.1109/WSC57314.2022.10015362

Examples

```
if (torch::torch_is_installed()){

# load ALS data
ig<- alsData$graph
data<- alsData$exprs
```

```
data<- transformData(data)$data
group<- alsData$group

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))
#ncores<- parallel::detectCores(logical = FALSE)

start<- Sys.time()
dnn0<- SEMdnn(ig, data[train, ], algo = "layerwise",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)
end<- Sys.time()
print(end-start)

#str(dnn0, max.level=2)
dnn0$fit$fitIdx
parameterEstimates(dnn0$fit)
gplot(dnn0$graph)
table(E(dnn0$graph)$color)

#...with source nodes -> graph layer structure -> sink nodes

#Topological layer (TL) ordering
K<- c(12,  5,  3,  2,  1,  8)
K<- rev(K[-c(1,length(K))]);K

ig1<- mapGraph(ig, type="source"); gplot(ig1)

start<- Sys.time()
dnn1<- SEMdnn(ig1, data[train, ], algo = "layerwise",
hidden = 5*K, link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)
end<- Sys.time()
print(end-start)

#Visualization of the neural network structure
nplot(dnn1, hidden = 5*K, bias = FALSE)

#str(dnn1, max.level=2)
dnn1$fit$fitIdx
mean(dnn1$fit$Psi)
parameterEstimates(dnn1$fit)
gplot(dnn1$graph)
table(E(dnn1$graph)$color)

#...with a categorical outcome
outcome<- factor(ifelse(group == 0, "control", "case")); table(outcome)

start<- Sys.time()
dnn2<- SEMdnn(ig, data[train, ], outcome[train], algo = "layerwise",
hidden = c(10,10,10), link = "selu", bias = TRUE,
epochs = 32, patience = 10, verbose = TRUE)
```

```
end<- Sys.time()
print(end-start)

#str(dnn2, max.level=2)
dnn2$fit$fitIdx
parameterEstimates(dnn2$fit)
gplot(dnn2$graph)
table(E(dnn2$graph)$color)
table(V(dnn2$graph)$color)
}
```

SEMml *Nodewise SEM train using Machine Learning (ML)*

Description

The function converts a graph to a collection of nodewise-based models: each mediator or sink variable can be expressed as a function of its parents. Based on the assumed type of relationship, i.e. linear or non-linear, `SEMml()` fits a ML model to each node (variable) with non-zero incoming connectivity. The model fitting is performed equation-by equation (r=1,...,R) times, where R is the number of mediators and sink nodes.

Usage

```
SEMml(
  graph,
  data,
  outcome = NULL,
  algo = "sem",
  thr = NULL,
  nboot = 0,
  ncores = 2,
  verbose = FALSE,
  ...
)
```

Arguments

`graph`	An igraph object.
`data`	A matrix with rows corresponding to subjects, and columns to graph nodes (variables).
`outcome`	A character vector (as.fctor) of labels for a categorical output (target). If NULL (default), the categorical output (target) will not be considered.
`algo`	ML method used for nodewise-based predictions. Four algorithms can be specified:

- `algo="sem"` (default) for a linear SEM, see `SEMrun`.
- `algo="tree"` for a CART model, see `rpart`.
- `algo="rf"` for a random forest model, see `ranger`.
- `algo="xgb"` for a XGBoost model, see `xgboost`.

`thr`	A numeric value [0-1] indicating the threshold to apply to the variable importance values to color the graph. If thr = NULL (default), the threshold is set to thr = 0.5*max(abs(variable importance values)).
`nboot`	number of bootstrap samples that will be used to compute cheap (lower, upper) CIs for all input variable weights. As a default, nboot = 0.
`ncores`	number of cpu cores (default = 2)
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

By mapping data onto the input graph, `SEMml()` creates a set of nodewise models based on the directed links, i.e., a set of edges pointing in the same direction, between two nodes in the input graph that are causally relevant to each other. The mediator or sink variables are defined as functions of their parents. Then, an ML model (sem, tree, rf, xgb) can be fitted to each variable with non-zero inbound connectivity. The model fitting process is performed equation-by-equation (r=1,...,R) times, where R represents the number of mediators and sink nodes in the input graph.

If boot != 0, the function will implement the cheap bootstrapping proposed by Lam (2002) to generate uncertainties (i.e., bootstrap `90%CIs`) for ML parameters. Bootstrapping can be enabled by setting a small number (1 to 10) of bootstrap samples. Note, however, that the computation can be time-consuming for massive MLs, even with cheap bootstrapping!

Value

An S3 object of class "ML" is returned. It is a list of 5 objects:

1. "fit", a list of ML model objects, including: the estimated covariance matrix (Sigma), the estimated model errors (Psi), the fitting indices (fitIdx), and the parameterEstimates, i.e., the variable importance measures (VarImp).
2. "gest", the data.frame of variable importances (parameterEstimates) of outcome levels, if outcome != NULL.
3. "model", a list of all the fitted non-linear nodewise-based models (tree, rf, xgb, nn or dnn).
4. "graph", the induced DAG of the input graph mapped on data variables. The DAG with colored edge/nodes based on the variable importance measures, i.e., if abs(VarImp) > thr will be highlighted in red (VarImp > 0) or blue (VarImp < 0). If the outcome vector is given, nodes with variable importances summed over the outcome levels, i.e. sum(VarImp[outcome levels])) > thr, will be highlighted in pink.
5. "data", input data subset mapping graph nodes.

Using the default `algo="sem"`, the usual output of a linear nodewise-based SEM, see `SEMrun` (algo="cggm"), will be returned.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

Grassi M., Palluzzi F., and Tarantino B. (2022). SEMgraph: An R Package for Causal Network Analysis of High-Throughput Data with Structural Equation Models. Bioinformatics, 38 (20), 4829–4830 <https://doi.org/10.1093/bioinformatics/btac567>

Breiman L., Friedman J.H., Olshen R.A., and Stone, C.J. (1984) Classification and Regression Trees. Chapman and Hall/CRC.

Breiman L. (2001). Random Forests, Machine Learning 45(1), 5-32.

Chen T., and Guestrin C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Lam, H. (2022). Cheap bootstrap for input uncertainty quantification. WSC '22: Proceedings of the Winter Simulation Conference, 2318-2329.

Examples

```
# Load Amyotrophic Lateral Sclerosis (ALS)
ig<- alsData$graph
data<- alsData$exprs
data<- transformData(data)$data
group<- alsData$group

#...with train-test (0.5-0.5) samples
set.seed(123)
train<- sample(1:nrow(data), 0.5*nrow(data))

start<- Sys.time()
# ... tree
res1<- SEMml(ig, data[train, ], algo="tree")

# ... rf
res2<- SEMml(ig, data[train, ], algo="rf")

# ... xgb
res3<- SEMml(ig, data[train, ], algo="xgb")

# ... sem
res4<- SEMml(ig, data[train, ], algo="sem")

end<- Sys.time()
print(end-start)

#visualizaation of the colored dag for algo="sem"
gplot(res4$graph, l="dot", main="sem")

#Comparison of fitting indices (in train data)
res1$fit$fitIdx #tree
```

```
res2$fit$fitIdx #rf
res3$fit$fitIdx #xgb
res4$fit$fitIdx #sem

#Comparison of parameter estimates (in train data)
parameterEstimates(res1$fit) #tree
parameterEstimates(res2$fit) #rf
parameterEstimates(res3$fit) #xgb
parameterEstimates(res4$fit) #sem

#Comparison of VarImp (in train data)
table(E(res1$graph)$color) #tree
table(E(res2$graph)$color) #rf
table(E(res3$graph)$color) #xgb
table(E(res4$graph)$color) #sem

#Comparison of AMSE, R2, SRMR (in test data)
print(predict(res1, data[-train, ])$PE) #tree
print(predict(res2, data[-train, ])$PE) #rf
print(predict(res3, data[-train, ])$PE) #xgb
print(predict(res4, data[-train, ])$PE) #sem

#...with a categorical (as.factor) outcome
outcome <- factor(ifelse(group == 0, "control", "case")); table(outcome)

res5 <- SEMml(ig, data[train, ], outcome[train], algo="tree")
gplot(res5$graph)
table(E(res5$graph)$color)
table(V(res5$graph)$color)

pred <- predict(res5, data[-train, ], outcome[-train], verbose=TRUE)
yhat <- pred$Yhat[ ,levels(outcome)]; head(yhat)
yobs <- outcome[-train]; head(yobs)
classificationReport(yobs, yhat, verbose=TRUE)$stats
```

Index