Package ‘SEMdeep’

September 16, 2024

Title Structural Equation Modeling with Deep Neural Network and Machine Learning

Version 0.1.0

Date 2024-08-20

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 cito, corpcor, lavaan, mgcv, NeuralNetTools, nnet, ranger, shapr, torch, xgboost

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

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2024-09-16 15:30:02 UTC

Contents

benchmark *Prediction benchmark evaluation utility*

Description

This function is able to calculate a series of binary classification evaluation statistics given (i) two vectors: one with the true target variable values, and the other with the predicted target variable values or (ii) a confusion matrix with the counts for False Positives (FP), True Positives (TP), True Negatives (TN), and False Negatives (FN). The user can specify the desired set of metrics to compute: (i) precision, recall, f1 score and Matthews Correlation Coefficient (mcc) or (ii) specificity, sensitivity, accuracy and mcc.

Usage

```
benchmark(yobs, yhat, CT = NULL, thr = 0, F1 = TRUE, verbose = FALSE, ...)
```

Arguments

`yobs`	A binary vector with the true target variable values.
`yhat`	A continuous vector with the predicted target variable values.
`CT`	An optional confusion matrix of dimension 2x2 containing the counts for FP, TP, TN, and FN.
`thr`	A numerical value indicating the threshold for converting the `yhat` continuous vector to a binary vector. If `yhat` vector ranges between -1 and 1, the user can specify `thr = 0` (default); if `yhat` ranges between 0 and 1, the user can specify `thr = 0.5`.
`F1`	A logical value. If TRUE (default), precision (pre), recall (rec), f1 and mcc will be computed. Otherwise, if FALSE, specificity (sp), sensitivity (se), accuracy (acc) and mcc will be obtained.
`verbose`	A logical value. If FALSE (default), the density plots of `yhat` per group will not be plotted to screen.
`...`	Currently ignored.

Details

#' Suppose a 2x2 table with notation

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$se = A/(A + C)$$

$$sp = D/(B + D)$$

$$acc = (A + D)/(A + B + C + D)$$

$$pre = A/(A + B)$$

$$rec = A/(A + C)$$

$$F1 = (2 * pre * rec)/(pre + rec)$$

$$mcc = (A * D - B * C)/sqrt((A + B) * (A + C) * (D + B) * (D + C))$$

Value

A data.frame with classification evaluation statistics is returned.

Author(s)

Mario Grassi <mario.grassi@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

Chicco, D., Jurman, G. (2020) The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics 21, 6.

Examples

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

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

```
#...with a binary outcome (1=case, 0=control)
ig1<- mapGraph(ig, type = "outcome"); gplot(ig1)
outcome<- group; table(outcome)
data1<- cbind(outcome, data); data1[1:5,1:5]

res <- SEMml(ig1, data1, train, algo="rf")
mse <- predict(res, data1[-train, ])
yobs<- group[-train]
yhat<- mse$Yhat[ ,"outcome"]

# ... evaluate predictive benchmark (sp, se, acc, mcc)
benchmark(yobs, yhat, thr=0, F1=FALSE)

# ... evaluate predictive benchmark (pre, rec, f1, mcc)
benchmark(yobs, yhat, thr=0, F1=TRUE)

#... with confusion matrix table as input
ypred<- ifelse(yhat < 0, 0, 1)
benchmark(CT=table(yobs, ypred), F1=TRUE)

#...with density plots of yhat per group
#old.par <- par(no.readonly = TRUE)
benchmark(yobs, yhat, thr=0, F1=FALSE, verbose = TRUE)
#par(old.par)
```

getConnectionWeight *Connection Weight Approach for neural network variable importance*

Description

The function computes 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 (2004).

Usage

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

Arguments

`object`	A neural network object from `SEMdnn()` function.
`thr`	A value of the threshold to apply to connection weights. If NULL (default), the threshold is set to thr=mean(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 od two object: (i) a data.frame including the connections together with their weights, and (ii) the DAG with colored edges. 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.

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. 135-150. 10.1016/S0304-3800(02)00064-9.

Olden, Julian. (2004). An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. Ecological Modelling. 178. 10.1016/S0304-3800(04)00156-5.

Examples

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

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

dnn0 <- SEMdnn(ig, data, train=1:nrow(data), cowt = FALSE,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)
res<- getConnectionWeight(dnn0, thr=NULL, verbose=TRUE)
table(E(res$dag)$color)
}
```

getGradientWeight *Gradient Weight Approach for neural network variable importance*

Description

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

Usage

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

Arguments

`object`	A neural network object from `SEMdnn()` function.
`thr`	A threshold value to apply to gradient weights of input nodes (variables). If NULL (default), the threshold is set to thr=mean(abs(gradient weights)).
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

The partial derivatives method calculates 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 compute 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. Therefore, the function uses a progress bar to check the progress of the gradient evaluation per observation.

Value

A list od two object: (i) a data.frame including the connections together with their weights, and (ii) the DAG with colored edges. 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.

Author(s)

Mario Grassi <mario.grassi@unipv.it>

References

Amesöder, C., Hartig, F. and Pichler, M. (2024), 'cito': an R package for training neural networks using 'torch'. Ecography, 2024: e07143. https://doi.org/10.1111/ecog.07143

Examples

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

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

dnn0 <- SEMdnn(ig, data, train=1:nrow(data), cowt = FALSE,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)

res<- getGradientWeight(dnn0, thr=NULL, verbose=TRUE)
table(E(res$dag)$color)
}
```

`getInputPvalue` *Test for the significance of neural network inputs*

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 values, as proposed by Mohammadi (2018).

Usage

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

Arguments

`object`	A neural network object from `SEMdnn()` function.
`thr`	A value of the threshold to apply to input p-values. If thr=NULL (default), the threshold is set to thr=0.05.

`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

A neural network with an arbitrary architecture is trained, taking into account factors like the number of neurons, hidden layers, 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; network's predictions are saved, after doing this 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 with the standard t-student critical values, which represent the importance of the input variables.

Value

A list od two object: (i) a data.frame including the connections together with their p-values, and (ii) the DAG with colored edges. If p-values > thr and t-test < 0, the edge is inhibited and it is highlighted in blue; otherwise, if p-values > thr and t-test > 0, the edge is activated and it is highlighted in red.

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.

Examples

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

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

dnn0 <- SEMdnn(ig, data, train=1:nrow(data), cowt = FALSE,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)

res<- getInputPvalue(dnn0, thr=NULL, verbose=TRUE)
table(E(res$dag)$color)
}
```

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

Description

This function computes variable contributions for individual predictions using the Shapley values, a method from cooperative game theory where the variable values of an observation work together to achieve the prediction. In addition, to make variable contributions easily explainable, the function decomposes the entire model R-Squared (R2 or the coefficient of determination) into variable-level attributions of the variance (Redell, 2019).

Usage

```
getShapleyR2(object, newdata, thr = NULL, 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.
`thr`	A threshold value to apply to signed Shapley (R2) values. If thr=NULL (default), the threshold is set to thr=mean(Shapley(R2) values)).
`verbose`	A logical value. If FALSE (default), the processed graph will not be plotted to screen.
`...`	Currently ignored.

Details

Lundberg & Lee (2017) proposed a unified approach to both local explainability (the variable contribution of a single variable within a single sample) and global explainability (the variable contribution of the entire model) by applying the fair distribution of payoffs principles from game theory put forth by Shapley (1953). Now called SHAP (SHapley Additive exPlanations), this suggested framework explains predictions of ML models, where input variables take the place of players, and their contribution to a particular prediction is measured using Shapley values. Successively, Redell (2019) presented a metric that combines the additive property of Shapley values with the robustness of the R2 of Gelman (2018) to produce an R2 variance decomposition that accurately captures the contribution of each variable to the explanatory power of the model. Additionally, we use the signed R2, in order to denote the regulation of connections in line with a linear SEM, since the edges in the DAG indicate node regulation (activation, if positive; inhibition, if negative). This 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, as suggested by Joseph (2019). It should be noted that in order to ascertain the local significance of node regulation with respect to the DAG, the Shapley decomposition of the R-squared (R2) value can be employed for each outcome node (r=1,...,R) by averaging the R2 indices of their input nodes.

The Shapley values are computed using the **shapr** package that implements an extended version of the Kernel SHAP method for approximating Shapley values in which dependence between the features is taken into account. The operations necessary to compute kernel SHAP values are inherently time-consuming, with the computational time increasing in proportion to the number of predictor variables and the number of observations. Therefore, the function uses a progress bar to check the progress of the kernel SHAP evaluation per observation.

Value

A list od three object: (i) data.frame including the connections together with their signed Shapley R-squred values; (ii) the dag with colored edges, if abs(sign_R2) > thr will be highlighted in red (sign_R2 > 0) or blue (sign_R2 < 0); and (ii) the list of individual Shapley values of predictors variables per each response variable.

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: Methodology.

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=train, algo="rf", vimp=FALSE)

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

```
# average shapley R2 across response variables
R2<- abs(res$est[,4])
Y<- res$est[,1]
R2Y<- aggregate(R2~Y,data=data.frame(R2,Y),FUN="mean")
PE<- predict(rf0, data[-train, ])$PE
cbind(R2Y=R2Y[,2],PEY=PE[-1])
mean(R2) # total average R2
PE[1]    # total MSE
```

---

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. Regarding the former, source nodes can represent: (i) data variables; (ii) a group variable; (iii) Latent Variables (LV). For the latter, an outcome variable, representing the prediction of interest, can be added. Moreover, `mapGraph()` can also create a new graph object starting from a compact symbolic formula.

Usage

```
mapGraph(graph, type = "outcome", LV = NULL, f = NULL, ...)
```

Arguments

`graph`	An igraph object.
`type`	A character value specifying the type of mapping. Five types can be specified. If `type = "source"` is specified, an additional source node (or more) is added to the graph. If `type = "group"`, an additional group source node is added. If `type = "outcome"` (default), a prediction sink node is mapped to the graph. If `type = "LV"`, a LV source node is included (where the number of LV depends on the LV argument). If `type = "clusterLV"`, a series of clusters for the data are computed and a different LV source node is added separately for each cluster.
`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.
`...`	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 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 to ALS graph
ig3 <- mapGraph(ig, type = "outcome"); gplot(ig3, l="dot")

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

# ... map LV source nodes to the clusters 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 draws a neural network plot as a neural interpretation diagram using with the `plotnet` function of the **NeuralNetTools** R package.

Usage

```
nplot(dnn.fit, bias = FALSE, ...)
```

Arguments

`dnn.fit`	A neural network model from **cito** R package.
`bias`	A logical value, indicating whether to draw biases in the layers (default = FALSE).
`...`	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

nplot returns invisibly the graphical object representing the neural network architecture of 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

dnn0 <- SEMdnn(ig, data, train=1:nrow(data), grad = FALSE,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)

 for (j in 1:length(dnn0$model)) {
   nplot(dnn0$model[[j]])
   Sys.sleep(5)
 }
}
```

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

Description

Predict method for DNN objects.

Usage

```
## S3 method for class 'DNN'
predict(object, newdata, 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.
`verbose`	Print predicted out-of-sample MSE values (default = FALSE).
`...`	Currently ignored.

Value

A list of 2 objects:

1. "PE", vector of the prediction error equal to the Mean Squared Error (MSE) for each out-of-bag prediction. The first value of PE is the AMSE, where we average over all (sink and mediators) graph nodes.
2. "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)
data<- alsData$exprs; dim(data)
data<- transformData(data)$data
ig<- alsData$graph; gplot(ig)
group<- alsData$group

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

start<- Sys.time()
dnn0 <- SEMdnn(ig, data, train, cowt = FALSE, thr = NULL,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)
end<- Sys.time()
print(end-start)
mse0 <- predict(dnn0, data[-train, ], verbose=TRUE)

# SEMrun vs. SEMdnn MSE comparison
sem0 <- SEMrun(ig, data[train, ], SE="none", limit=1000)
mse0 <- predict(sem0, data[-train,], verbose=TRUE)
#...with a binary outcome (1=case, 0=control)

ig1<- mapGraph(ig, type="outcome"); gplot(ig1)
outcome<- ifelse(group == 0, -1, 1); table(outcome)
data1<- cbind(outcome, data); data1[1:5,1:5]

start<- Sys.time()
dnn1 <- SEMdnn(ig1, data1, train, cowt = TRUE, thr = NULL,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)
end<- Sys.time()
print(end-start)

mse1 <- predict(dnn1, data1[-train, ])
yobs <- group[-train]
yhat <- mse1$Yhat[ ,"outcome"]
benchmark(yobs, yhat, thr=0, F1=FALSE)
}
```

`predict.ML` *SEM-based out-of-sample prediction using node-wise ML*

Description

Predict method for ML objects.

Usage

```
## S3 method for class 'ML'
predict(object, newdata, 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.
`verbose`	Print predicted out-of-sample MSE values (default = FALSE).
`...`	Currently ignored.

Value

A list of 2 objects:

1. "PE", vector of the prediction error equal to the Mean Squared Error (MSE) for each out-of-bag prediction. The first value of PE is the AMSE, where we average over all (sink and mediators) graph nodes.
2. "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)
data<- alsData$exprs; dim(data)
data<- transformData(data)$data
ig<- alsData$graph; gplot(ig)

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

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

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

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

# ... gam
res4<- SEMml(ig, data, train, algo="gam", vimp=FALSE)
mse4<- predict(res4, data[-train, ], verbose=TRUE)

# ... sem
res5<- SEMml(ig, data, train, algo="sem", vimp=FALSE)
mse5<- predict(res5, data[-train, ], verbose=TRUE)
end<- Sys.time()
print(end-start)
```

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: (1) construction of the topological layer (TL) ordering of the input graph; (2) prediction of the node y values in a layer, where the nodes included in the previous layers act as predictors x.

Usage

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

Arguments

`object`	An object, as that created by the function `SEMrun()` with the argument `fit` set to `fit = 0` or `fit = 1`.
`newdata`	A matrix with new data, with rows corresponding to subjects, and columns to variables. If `object$fit` is a model with the group variable (`fit = 1`), the first column of newdata must be the new group binary vector (0=control, 1=case).
`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 organised in topological layers, j=1,...,L. In each iteration, the response variables y are the nodes in the j layer and the predictors x are the nodes belonging to the previous j-1 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 2 objects:

1. "PE", vector of the prediction error equal to the Mean Squared Error (MSE) for each out-of-bag prediction. The first value of PE is the AMSE, where we average over all (sink and mediators) graph nodes.
2. "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>

Examples

```
# 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))

# SEM fitting
sem0<- SEMrun(ig, data[train,], SE="none", limit=1000)

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

res0<- predict(sem0, newdata=data[-train,])
print(res0$PE)

# SEM fitting
sem1<- SEMrun(ig, data[train,], group[train], SE="none", limit=1000)

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

res1<- predict(sem1, newdata=cbind(group,data)[-train,])
print(res1$PE)


#...with a binary outcome (1=case, 0=control)

ig1<- mapGraph(ig, type="outcome"); gplot(ig1)
outcome<- ifelse(group == 0, -1, 1); table(outcome)
data1<- cbind(outcome, data); data1[1:5,1:5]

sem10 <- SEMrun(ig1, data1[train,], SE="none", limit=1000)
res10<- predict(sem10, newdata=data1[-train,], verbose=TRUE)

yobs<- group[-train]
```

```
yhat<- res10$Yhat[,"outcome"]
benchmark(yobs, yhat)

#...with predictors, source nodes; outcomes, sink nodes
ig2<- mapGraph(ig, type= "source"); gplot(ig2)

sem02 <- SEMrun(ig2, data[train,], SE="none", limit=1000)
res02<- predict(sem02, newdata=data[-train,], verbose=TRUE)
```

SEMdnn *Layer-wise SEM train with a Deep Neural Netwok (DNN)*

Description

The function builds the topological layer (TL) ordering of the input graph to fit a series of Deep Neural Networks (DNN) models, where the nodes in one layer act as response variables (output) y and the nodes in the sucessive layers act as predictors (input) x. Each fit uses the `dnn` function of the **cito** R package, based on the deep learning framework 'torch'.

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,
  train = NULL,
  cowt = FALSE,
  thr = NULL,
  loss = "mse",
  hidden = c(10L, 10L, 10L),
  link = "relu",
  validation = 0,
  bias = TRUE,
  lambda = 0,
  alpha = 0.5,
  dropout = 0,
  optimizer = "adam",
  lr = 0.01,
  epochs = 100,
  device = "cpu",
  early_stopping = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`graph`	An igraph object.
`data`	A matrix with rows corresponding to subjects, and columns to graph nodes (variables).
`train`	A numeric vector specifying the row indices corresponding to the train dataset.
`cowt`	A logical value. If `cowt=TRUE` connection weights of the input node (variables) are computing (default = FALSE).
`thr`	A numerical value indicating the threshold to apply on the absolute values of the connection matrix to color the graph (default = NULL).
`loss`	A character value specifying the loss after which network should be optimized. The user can specify: (a) "mse" (mean squared error), "mae" (mean absolute error), or "gaussian" (normal likelihood), for regression problems; (b) "poisson" (poisson likelihood), or "nbinom" (negative binomial likelihood) for regression with count data; (c) "binomial" (binomial likelihood) for binary classification problems; (d) "softmax" or "cross-entropy" for multi-class classification (default = "mse").
`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 (default = "relu").
`validation`	A numerical value indicating the proportion of the data set that should be used as a validation set (randomly selected, default = 0).
`bias`	A logical vector, indicating whether to employ biases in the layers (`bias = TRUE`), which can be either vectors of logicals for each layer (number of hidden layers + 1 (final layer)) or of length one (default = TRUE).
`lambda`	A numerical value indicating the strength of regularization: lambda penalty, $\lambda * (L1 + L2)$ (default = 0).
`alpha`	A numerical vector to add L1/L2 regularization into the training. Set the alpha parameter for each layer to $(1 - \alpha) * \|\|weights\|\|_1 + \alpha\|\|weights\|\|^2$. It must fall between 0 and 1 (default = 0.5).
`dropout`	A numerical value for the dropout rate, which is the probability that a node will be excluded from training (default = 0).
`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 back-

	propagation), "sgd" (stochastic gradient descent). As a default, `optimizer = “adam”`.
`lr`	A numerical value indicating the learning rate given to the optimizer (default = 0.01).
`epochs`	A numerical value indicating the epochs during which the training is conducted (default = 100).
`device`	A character value describing the CPU/GPU device ("cpu", "cuda", "mps") on which the neural network should be trained on (default = "cpu")
`early_stopping`	If set to integer, training will terminate if the loss increases over a predetermined number of consecutive epochs and apply validation loss when available. Default is FALSE, no early stopping is applied.
`verbose`	If `verbose = TRUE`, the training curves of the DNN models are displayed as output, comparing the training, validation and baseline curves in terms of loss (y) against the number of epochs (x) (default = TRUE).
`...`	Currently ignored.

Details

By mapping data onto the input graph, `SEMdnn()` creates a set of DNN models based on the topological layer (j=1,…,L) structure 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. Each DNN model 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.

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 estimated connection weights (parameterEstimates), if cowt=TRUE.
2. "Yhat", the matrix of continuous predicted values of graph nodes (excluding source nodes) based on training samples.
3. "model", a list of all j=1,...,(L-1) fitted MLP network models.
4. "graph", the induced DAG of the input graph mapped on data variables. If cowt=TRUE, the DAG is colored based on the estimated connection weights, 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.
5. "data", standardized training data subset mapping graph nodes.

Author(s)

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

References

Amesöder, C., Hartig, F. and Pichler, M. (2024), ‘cito’: an R package for training neural networks using ‘torch’. Ecography, 2024: e07143. https://doi.org/10.1111/ecog.07143

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>

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))

start<- Sys.time()
dnn0 <- SEMdnn(ig, data, train, cowt = TRUE, thr = NULL,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)
end<- Sys.time()
print(end-start)

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

#...with a binary outcome (1=case, 0=control)

ig1<- mapGraph(ig, type="outcome"); gplot(ig1)
outcome<- ifelse(group == 0, -1, 1); table(outcome)
data1<- cbind(outcome, data); data1[1:5,1:5]

start<- Sys.time()
dnn1 <- SEMdnn(ig1, data1, train, cowt = TRUE, thr = NULL,
#loss = "mse", hidden = 5*K, link = "selu",
loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)
end<- Sys.time()
print(end-start)
#str(dnn1, max.level=2)
dnn1$fit$fitIdx
dnn1$fit$parameterEstimates
gplot(dnn1$graph)
table(E(dnn1$graph)$color)

#...with input -> hidden structure -> output :
# 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

ig2<- mapGraph(ig, type="source"); gplot(ig2)

start<- Sys.time()
dnn2 <- SEMdnn(ig2, data, train, cowt = TRUE, thr = NULL,
loss = "mse", hidden = 5*K, link = "selu",
#loss = "mse", hidden = c(10, 10, 10), link = "selu",
validation = 0, bias = TRUE, lr = 0.01,
epochs = 32, device = "cpu", verbose = TRUE)
end<- Sys.time()
print(end-start)

#Visualization of the neural network structure
nplot(dnn2$model[[1]], bias=FALSE)

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

`SEMml` *Nodewise-predictive 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 repeated equation-by equation (r=1,...,R) times, where R is the number of mediators and sink nodes.

Usage

```
SEMml(
  graph,
  data,
  train = NULL,
  algo = "sem",
  vimp = FALSE,
  thr = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

`graph`	An igraph object.
`data`	A matrix with rows corresponding to subjects, and columns to graph nodes (variables).
`train`	A numeric vector specifying the row indices corresponding to the train dataset (default = NULL).
`algo`	ML method used for nodewise-network predictions. Six algorithms can be specified: • `algo="sem"` (default) for a linear SEM, see `SEMrun`. • `algo="gam"` for a generalized additive model, see `gam`. • `algo="rf"` for a random forest model, see `ranger`. • `algo="xgb"` for a XGBoost model, see `xgboost`. • `algo="nn"` for a small neural network model (1 hidden layer and 10 nodes), see `nnet`. • `algo="dnn"` for a large neural network model (1 hidden layers and 1000 nodes), see `dnn`.
`vimp`	A Logical value(default=FALSE). If TRUE compute the variable importance, considering: (i) the squared value of the t-statistic or F-statistic of the model parameters for "sem" or "gam"; (ii) the variable importance from the `importance` or `xgb.importance` functions for "rf" or "xgb"; (iii) the Olden's connection weights for "nn" or "dnn".
`thr`	A numerical value indicating the threshold to apply on the variable importance to color the graph. If thr=NULL (default), the threshold is set to thr = abs(mean(vimp)).
`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-based 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 can be characterized in detail as functions of their parents. An ML model (sem, gam, rf, xgb, nn, dnn) can then be fitted to each variable with non-zero inbound connectivity, taking into account the kind of relationship (linear or non-linear). With R representing the number of mediators and sink nodes in the network, the model fitting process is performed equation-by-equation (r=1,...,R) times.

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 signed Shapley R2 values (parameterEstimates), if shap = TRUE,
2. "Yhat", the matrix of continuous predicted values of graph nodes (excluding source nodes) based on training samples.
3. "model", a list of all the fitted nodewise-based models (sem, gam, rf, xgb or nn).
4. "graph", the induced DAG of the input graph mapped on data variables. If vimp = TRUE, the DAG is colored based on the variable importance measure, i.e., if abs(vimp) > thr will be highlighted in red (vimp > 0) or blue (vimp < 0).
5. "data", istandardized training data subset mapping graph nodes.

Author(s)

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

References

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>

Hastie, T. and Tibshirani, R. (1990) Generalized Additive Models. London: Chapman and Hall.

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

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

Ripley, B. D. (1996) Pattern Recognition and Neural Networks. Cambridge.

Redell, N. (2019). Shapley Decomposition of R-Squared in Machine Learning Models. arXiv: Methodology.

Examples

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

#...with train-test (0.5-0.5) samples
set.seed(123)
```

```
train<- sample(1:nrow(data), 0.5*nrow(data))

start<- Sys.time()
# ... rf
res1<- SEMml(ig, data, train, algo="rf", vimp=TRUE)

# ... xgb
res2<- SEMml(ig, data, train, algo="xgb", vimp=TRUE)

# ... nn
res3<- SEMml(ig, data, train, algo="nn", vimp=TRUE)

# ... gam
res4<- SEMml(ig, data, train, algo="gam", vimp=TRUE)
end<- Sys.time()
print(end-start)

# ... sem
res5<- SEMml(ig, data, train, algo="sem", vimp=TRUE)

#str(res5, max.level=2)
res5$fit$fitIdx
res5$fit$parameterEstimates
gplot(res5$graph)

#Comparison of AMSE (in train data)
rf <- res1$fit$fitIdx[2];rf
xgb<- res2$fit$fitIdx[2];xgb
nn <- res3$fit$fitIdx[2];nn
gam<- res4$fit$fitIdx[2];gam
sem<- res5$fit$fitIdx[2];sem

#Comparison of SRMR (in train data)
rf <- res1$fit$fitIdx[4];rf
xgb<- res2$fit$fitIdx[4];xgb
nn <- res3$fit$fitIdx[4];nn
gam<- res4$fit$fitIdx[4];gam
sem<- res5$fit$fitIdx[4];sem

#Comparison of VIMP (in train data)
table(E(res1$graph)$color) #rf
table(E(res2$graph)$color) #xgb
table(E(res3$graph)$color) #nn
table(E(res4$graph)$color) #gam
table(E(res5$graph)$color) #sem

#Comparison of AMSE (in test data)
print(predict(res1, data[-train, ])$PE[1]) #rf
print(predict(res2, data[-train, ])$PE[1]) #xgb
print(predict(res3, data[-train, ])$PE[1]) #nn
print(predict(res4, data[-train, ])$PE[1]) #gam
print(predict(res5, data[-train, ])$PE[1]) #sem
```

```
#...with a binary outcome (1=case, 0=control)

ig1<- mapGraph(ig, type="outcome"); gplot(ig1)
outcome<- ifelse(group == 0, -1, 1); table(outcome)
data1<- cbind(outcome, data); data1[1:5,1:5]

res6 <- SEMml(ig1, data1, train, algo="nn", vimp=TRUE)
gplot(res6$graph)
table(E(res6$graph)$color)

mse6 <- predict(res6, data1[-train, ])
yobs <- group[-train]
yhat <- mse6$Yhat[ ,"outcome"]
benchmark(yobs, yhat, thr=0, F1=TRUE)
benchmark(yobs, yhat, thr=0, F1=FALSE)
```

Index