Package ‘iDeepViewLearn’

February 17, 2024

Type Package

Title Interpretable Deep Learning Method for Multi-view Learnings

Version 0.2.0

Author Hengkang Wang, Han Lu, Ju Sun, Sandra Safo

Maintainer Han Lu <lu000054@umn.edu> and Sandra E. Safo <ssafo@umn.edu>

Description Interpretable deep learning method that models nonlinear relationships with deep neural network, associates data from two or more views, achieves feature selection, and constructs low-dimensional representation. Also with knowledge-based approach using the normalized Laplacian of a graph to encourage variable selection.

Imports reticulate

License GPL (>=2.0)

Encoding UTF-8

LazyData true

R topics documented:

`iDeepViewLearn_checkpy` *Check python environment and packages.*

Description

Trains an iDeepViewLearn model with your choice of using default hyperparameter any time, using tuning dataset when both X_tune and y_tune are available, or using cross validation method y_train is available. Returns selected features, models trained, and classifier that goes into testing, and reconstructed data.

Usage

```
iDeepViewLearn_checkpy(python_path)
```

Arguments

`python_path`	A string to python enviroment. See Reticulate package.

Value

The function should not return anything. If there is any error, please check your python enviornment and the packages in the python environment.

References

Hengkang Wang, Han Lu, Ju Sun, Sandra E. Safo, *Interpretable Deep Learning Methods for Multiview Learning*, submitted.

See Also

`iDeepViewLearn_data_example`, `iDeepViewLearn_test`

Examples

```
######## import library and data example
library(iDeepViewLearn)
iDeepViewLearn_checkpy(python_path="~/.conda/envs/myenv/bin")
```

`iDeepViewLearn_data_example` *Data example for iDeepViewLearn.*

Usage

```
data("iDeepViewLearn_data_example")
```

Format

An R object with 6 items. 2 views, each with 350 observations and 500 variables. For simplicity and for illustration purpose, both view have the same number of features in this example. Note that while the number of observations in each view must be the same, the number of features in each view can be different.

X_train, X_tune, Xtest Each being a list of 2 views. Each with has 350 observations and 500 variables.

y_train, y_tune, y_test Each being an integer vector of length 350, obtaining values 1 and 2 indicating the class.

References

Hengkang Wang, Han Lu, Ju Sun, Sandra E. Safo, *Interpretable Deep Learning Methods for Multiview Learning*, submitted.

Examples

```
data("iDeepViewLearn_data_example")
# See iDeepViewLearn_train or iDeepViewLearn_test for details.
```

`iDeepViewLearn_Laplacian_data_example`
Data with prior information example for iDeepViewLearn with Laplacian.

Usage

```
data("iDeepViewLearn_Laplacian_data_example")
```

Format

An R object with 9 items. 2 views, each with 350 observations and 500 variables. For simplicity and for illustration purpose, both view have the same number of features and the same edge and weight information in this example. Note that while the number of observations in each view must be the same, the number of features in each view can be different. Also note that even if the edge and the weight information of different view can be different.

X_train, X_tune, Xtest Each being a list of 2 views. Each with has 350 observations and 500 variables.

y_train, y_tune, y_test Each being an integer vector of length 350, obtaining values 1 and 2 indicating the class.

The first 50 variables are connected with a graph. See Figure 4 Left for a visualization for this example data. The ground truth is that 21 features is truly important in data generating.

gt A vector of length 21 being the indices of the ground truth of important features in data generating. Note that for python indices start with 0.

Prior information.

edge_ls A list of 2 dataframes with dimension 49×2 being the edge information of 2 views. The columns represents the edge connection. For example, in a row, if the first column of writes 1 and the second columns writes 26, then it means that there is an edge connecting variable 1 and variable 26. The number of rows, 49 in this example, is the total number of edges.

weight_vec_ls A list of 2 vectors with length 49 being the weight information of 2 views.

References

Hengkang Wang, Han Lu, Ju Sun, Sandra E. Safo, *Interpretable Deep Learning Methods for Multiview Learning*, submitted.

Examples

```
data("iDeepViewLearn_Laplacian_data_example")
# See iDeepViewLearn_train for details.
```

`iDeepViewLearn_sim_downstream_outcome` *Randomly simulated outcome data of different kind to demonstrate possible downstream analysis using shared or reconstructed low-dimensinal representations.*

Usage

```
data("iDeepViewLearn_sim_downstream_outcome")
```

Format

An R object with 5 items, each of length 350. Note that this version of outcome columns are completed randomly generated and merged with the simulation dataset. These outcome columns are for demonstration purposes only.

binary_outcome A integer vector of length 350 obtaining values of 0 or 1. Used for example of downstream analyses with binary outcomes.

categorical_outcome A integer vector of length 350 obtaning values 0, 1, 2, or 3. Used for example of downstream analyses with categorical outcomes.

continuous_outcome A numeric vector of length 350. Used for example of downstream analyses with continuous outcomes.

survival_time, survival_event Each being a vector of length 350. Used for example of downstream analyses with survival outcomes

References

Hengkang Wang, Han Lu, Ju Sun, Sandra E. Safo, *Interpretable Deep Learning Methods for Multiview Learning*, submitted.

Examples

```
data("iDeepViewLearn_sim_downstream_outcome")
# See iDeepViewLearn_test for details.
```

`iDeepViewLearn_test` *Tests performance and provides low-dimensinal representations given the trained iDeepViewLearn model.*

Description

Takes the trained iDeepViewLearn model and applies on the test dataset. Returns model used, classfier and accuracy if applied, and low-dimensional representation in testing stage.

Usage

```
iDeepViewLearn_test(python_path,
                    X_train, X_test, best_comb, features, model, clf,
                    y_train=NULL, y_test=NULL,
                    top_rate=0.1, edged=NULL, vWeightd=NULL, epochs=1000L,
                    plot=FALSE, verbose=TRUE, gpu=FALSE, normalization=TRUE)
```

Arguments

`python_path`	A string to python enviroment. See Reticulate package.
`X_train`	A list of d elements, where d is the number of views, being the training data. Each element is a view with dimension $n^d \times p^d$, where observations are on the rows and features are on the columns. n^d must be the same across all views, denote $n^d = n$, while p^d can be different.
`X_test`	A list of d elements, where d is the number of views, being the testing data. Each element is a view with dimension $n^d_{test} \times p^d$, where observations are on the rows and features are on the columns. n^d_{test} must be the same across all views. p^d can be different in each view, but must be consistent with the p^d in traning data.
`best_comb`	A list of 5 elements being the best combination of hyperparameters obtained from the training result.
`features`	A list of d elements, each being a list of each view's indices of the selected features obtained from the training result.
`model`	A list object obtained from the training result, being the model trained.
`clf`	A list obtained from the training result, being the classifier trained; **or a number being the loss of the last iteration, if outcome is missing.**
`y_train`	**If outcome variable is categorical, an integer vector of length n^d representing the classes; if continuous, a numeric vector of length n^d; if survival, a n^d by 2 matrix with the first column being event indicator (0=censored, 1=event) and the second column being survival time. This argument can be NULL.**
`y_test`	See y_train for format, with length of the vector or number of rows of the matrix n^d_{test}. This argument can be NULL.
`outtype`	**A required argument to indicate the output type, including "categorical", "continuous", "survival", or "nothing" if you do not have outcome.**
`top_rate`	A number between 0 to 1 to indicate the top fraction of features being selected. If impt is available, this argument will not be used.
`edged`	A list of length d, the number of views, with each element being a dataframe representing the edge information of the variables to use the Laplacian version of iDeepViewLearn. The columns represents the edge connection. For example, in a row, if the first column of writes 1 and the second columns writes 26, then it means that there is an edge connecting variable 1 and variable 26. The number of rows, r_d, is the total amount of edges. This argument can be NULL. If NULL, the algorithm will carry out the standard iDeepViewLearn method.
`vWeighted`	A list of length d, the number of views, with each element being a vector representing the weight information of edges to use the Laplacian version of iDeepViewLearn. Each vector has length r_d, being the total amount of edges of view d. This argument can be NULL. If NULL, the algorithm will carry out the standard iDeepViewLearn method.
`epochs`	An integer indicating the epochs of traning. Need to append a letter L to the integer.
`plot`	TRUE or FALSE to plot line charts of unsupervised 1, unsupervised 2, and Z loss history.
`verbose`	TRUE or FALSE to output training status and best combination of hyperparameters during training.
`gpu`	TRUE or FALSE to use gpu.

normalization TRUE or FALSE to normalize the testing data. We recommend using TRUE to prevent predictors with large norms to shadow the result.

myseed An integer to set the seed. Need to append a letter L to the integer, for example, 1234L.

Value

The function will return train_result, a list of 5 elements. To see the elements, use double square brackets. See below for more detail.

The following arguments are related to model and model assessment.

pred A list of integers being the predicted class. Takes NULL if clf is NULL or if y_test is not available. Can be obtained by test_result[[1]].

performance **A number as the model is performance. If the outcome is categorical, classification accuracy is presented; if the outcome is numerical, mean squared error is presented; if the outcome is survival, C-statsitics is provided; if the outcome is missing, BIC calculated by ncol(Zprimetest)*ln(nrow(Zprimetest)) + 2 * loss is presented.**

The following arguments are low-dimensinal representations during the testing stage.

Zprimetest A matrix of $n_{test} \times K$, being the shared latent code using the selected features only during the testing stage. Can be obtained by test_result[[3]].

RZprimetest A list of d elements, where each element has dimention $n_{test} \times p^d_{impt}$, being the nonlinear approximations. Use in downstream analyses. Can be obtained by test_result[[4]].

References

Hengkang Wang, Han Lu, Ju Sun, Sandra E. Safo, *Interpretable Deep Learning Methods for Multiview Learning*, submitted.

See Also

iDeepViewLearn_data_example,iDeepViewLearn_train

Examples

```
######## import library and data example
library(iDeepViewLearn)
data("iDeepViewLearn_data_example")

######## Train standard iDeepViewLearn
# Use validation dataset and default hyperparameters to train an iDeepViewLearn model.
# By ground truth of the data example, the first 50 variables are truly important.
# Here we chose to normalize data, i.e. normalize X_train and X_tune for each view.
train_result = iDeepViewLearn_train(python_path="~/.conda/envs/myenv/bin", myseed=1234L,
                                    X_train = X_train, y_train = y_train,
                                    X_tune=X_tune, y_tune=y_tune, search_times=0L,
                                    best_comb=NULL, impt=c(50L,50L), top_rate=0.1,
                                    edged=NULL, vWeightd=NULL, epochs=1000L,
                                    fold=5L, plot=FALSE, verbose=TRUE, gpu=FALSE,
                                    normalization=TRUE)
######## Testing stage
test_result = iDeepViewLearn_test(python_path = "~/.conda/envs/myenv/bin",
```

```
                                  X_train = X_train, X_test = X_test,
                                  y_train = y_train, y_test = y_test,
                                  features = train_result[[1]],
                                  model = train_result[[2]],
                                  clf = train_result[[3]],
                                  best_comb = train_result[[4]],
                                  normalization=TRUE, myseed=1234L)

######## Obtaining testing result
# To get the testing classification accuracy
test_acc = test_result[[2]]
# To get the Z' and R(Z') during the testing stage
Zprimetest = test_result[[3]]
RZprimetest = test_result[[4]]

######## Downstream analyses with Z'
# To use the Z', the shared low-dimensional representation of selected features
# in the testing stage to conduct downstream analysis
# Example: real data analysis and figure 6 in the paper.
data("iDeepViewLearn_sim_downstream_outcome")
# Example: Simple Linear Regression
slr_data = as.data.frame(cbind(Zprimetest, continuous_outcome))
slr_mod = lm(continuous_outcome~., data = slr_data)
# Example: Logistic Regression
lr_data = as.data.frame(cbind(Zprimetest, binary_outcome))
lr_mod = glm(binary_outcome~., family = binomial, data = lr_data)
# Example: SVM with categorical outcome
library(e1071)
svm_data = as.data.frame(cbind(Zprimetest, categorical_outcome))
svm_fit = svm(categorical_outcome~., data = svm_data,
              kernel = "radial", cost = 1, sacle = FALSE)

######## Downstream analyses with R(Z')
# To use the R(Z'), the reconstructed low-dimensional representation of selected
# features of each view in the testing stage to conduct downstream analysis
# Example: K-means clustering followed by Survival Analysis
cluster_1 = kmeans(RZprimetest[[1]], 3, algorithm = "Lloyd")
cluster_2 = kmeans(RZprimetest[[2]], 3, algorithm = "Lloyd")
library(survival)
surv_data = as.data.frame(cbind(cluster_1$cluster, cluster_2$cluster,
                                survival_time, survival_event))
surv_fit = survfit(Surv(survival_time, survival_event)~V1+V2, data = surv_data)
```

`iDeepViewLearn_train` *Trains an iDeepViewLearn model for multi-view data.*

Description

Trains an iDeepViewLearn model with your choice of using default hyperparameter any time, using tuning dataset when both X_tune and y_tune are available, or using cross validation method y_train is available. Returns selected features, models trained, and classifier that goes into testing, and reconstructed data.

Usage

```
iDeepViewLearn_train(python_path, myseed=1234L,
                     X_train, y_train=NULL, X_tune=NULL, y_tune=NULL,
                     search_times=0L, best_comb=NULL, impt=NULL, outtype=NULL,
                     top_rate=0.1, edged=NULL, vWeightd=NULL,
                     epochs=1000L, fold=5L,
                     plot=FALSE, verbose=TRUE, gpu=FALSE,
                     normalization=TRUE)
```

Arguments

`python_path` A string to python enviroment. See Reticulate package.

`myseed` An integer to set the seed. Need to append a letter L to the integer, for example, 1234L.

`X_train` A list of d elements, where d is the number of views, being the training data. Each element is a view with dimension $n^d \times p^d$, where observations are on the rows and features are on the columns. n^d must be the same across all views, denote $n^d = n$, while p^d can be different.

`y_train` **If outcome variable is categorical, an integer vector of length n^d representing the classes; if continuous, a numeric vector of length n^d; if survival, a n^d by 2 matrix with the first column being event indicator (0=censored, 1=event) and the second column being survival time. This argument can be NULL.**

`X_tune` A list of d elements, where d is the number of views, being the tuning data. This argument can be NULL. Each element is a view with dimension $n^d_{tune} \times p^d$, where observations are on the rows and features are on the columns. n^d_{tune} must be the same across all views. p^d can be different in each view, but must be consistent with the p^d in traning data.

`y_tune` See y_train for format, with length of the vector or number of rows of the matrix n^d_{tune}. This argument can be NULL.

`search_times` An integer of searching times for the hyperparameters. Need to append a letter L to the integer. To use the default hyperparameters, put 0L. To search once, put 1L. To search multiple times, for example, if you want to search 10 times, you may put 10L. In that case, we randomly selected 10 combinations from all possible grid search combinations. Increase the search times for an increased number of combinations to be searched, but the runtime could be longer. The maximum possible search times is the maximum grid search combinations, which is 192.

`best_comb` A list of 4 elements being the best combination of hyperparameters, **being K, the number of columns of the traied shared low dimensional latent code; the learning rate of the neural network for loss function; the learning rate of the neural net work for Z; the classifier parameter, C for SVM classifier and regressor, alpha for Survival SVM, and 1 for BIC (which does not mean anything and is only a placeholder); the panelty**. This argument can be NULL. If NULL, the algorithm will choose parameters according to the entry of search_times.

`impt` An integer vector of length d, the number of views, being the number of features in the low-dimensional reconstruction for the d views. Need to append a letter L to the integer. For example, if we want top 50 features for the first view and top 25 features for the second view, put c(50L, 25L). This argument can be NULL. If NULL, the algorithm will use top_rate to decide.

`outttype` **A required argument to indicate the output type, including "categorical", "continuous", "survival", or "nothing" if you do not have outcome.**

`top_rate` A number between 0 to 1 to indicate the top fraction of features being selected. If impt is available, this argument will not be used.

`edged` A list of length d, the number of views, with each element being a dataframe representing the edge information of the variables to use the Laplacian version of iDeepViewLearn. The columns represents the edge connection. For example, in a row, if the first column of writes 1 and the second columns writes 26, then it means that there is an edge connecting variable 1 and variable 26. The number of rows, r_d, is the total amount of edges. This argument can be NULL. If NULL, the algorithm will carry out the standard iDeepViewLearn method.

`vWeighted` A list of length d, the number of views, with each element being a vector representing the weight information of edges to use the Laplacian version of iDeepViewLearn. Each vector has length r_d, being the total amount of edges of view d. This argument can be NULL. If NULL, the algorithm will carry out the standard iDeepViewLearn method.

`epochs` An integer indicating the epochs of traning. Need to append a letter L to the integer.

`fold` An integer k indicating k-fold cross validation, if cross validation is used. Need to append a letter L to the integer.

`plot` TRUE or FALSE to plot line charts of unsupervised 1, unsupervised 2, and Z loss history.

`verbose` TRUE or FALSE to output training status and best combination of hyperparameters during training.

`gpu` TRUE or FALSE to use gpu.

`normalization` TRUE or FALSE to normalize the training and tuning data. We recommend using TRUE to prevent predictors with large norms to shadow the result.

Details

If both X_tune and y_tune are available, the algorithm will by default using the validation dataset. If X_tune is available while y_tune is not, or if neither is available, if specified a search_time other than 0L, the algorithm will automatically do cross validation.

Value

The function will return train_result, a list of 8 elements. To see the elements, use double square brackets. See below for more detail.

The following arguments are needed if you want to proceed with testing.

`selected_features`
A list of d elements, where d is the number of views. Can be obtained by train_result[[1]]. Each element is a list of the indices of selected features that go into the low-dimensional reconstruction. The selected features of the d-th view can be obtained by train_result[[1]][[d]].

`model` A list object being the trained models. Can be obtained by train_result[[2]].

`clf` A list object being the trained classifier if y_train is available. **Returns SVM classifier for categorical data, SVM Regressor for continuous data, Survival SVM estimator for survival data, and the loss of the last iteration for missing outcome types.** Can be obtained by train_result[[3]].

`best_comb` A list of 4 element being the hyperparameters used in the model. Can be obtained by train_result[[4]].

The following arguments provide latent code and the low-dimensional representations.

`Ztrain` A matrix of $n \times K$, where n is the number of observations and K is one of the hyperparameters of the model, being the shared latent code of the original data containing all features. Can be obtained by train_result[[5]].

`GZ` A list of d elements, where each element has dimension $n \times p^d$, being the reconstructed data in feature selection. Can be obtained by train_result[[6]].

`Zprime` A matrix of $n \times K$, being the shared latent code using the selected features only. Can be obtained by train_result[[7]].

`RZprime` A list of d elements, where each element has dimention $n \times p^d_{impt}$, being the nonlinear approximations. Use in downstream analyses. Can be obtained by train_result[[8]].

References

Hengkang Wang, Han Lu, Ju Sun, Sandra E. Safo, *Interpretable Deep Learning Methods for Multiview Learning*, submitted.

See Also

iDeepViewLearn_data_example, iDeepViewLearn_test

Examples

```
######## import library and data example
library(iDeepViewLearn)
data("iDeepViewLearn_data_example")

######## Train default iDeepViewLearn
# Use default hyperparameters (no tuning data needed) to train an iDeepViewLearn model.
# By ground truth of the data example, the first 50 variables are truly important.
# Here we chose to normalize data, i.e. normalize X_train and X_tune for each view.
train_result = iDeepViewLearn_train(python_path="~/.conda/envs/myenv/bin", myseed=1234L,
                                    X_train = X_train, y_train = y_train,
                                X_tune=NULL, y_tune=NULL, search_times=0L, # use default
                                    best_comb=NULL, impt=c(50L,50L),top_rate=0.1,
                                    edged=NULL, vWeightd=NULL, epochs=1000L,
                                    fold=5L, plot=FALSE, verbose=TRUE, gpu=FALSE,
                                    normalization=TRUE)
######## Train iDeepViewLearn with tuning
# Use tuning dataset, search 5 times
train_result_2 = iDeepViewLearn_train(python_path="~/.conda/envs/myenv/bin", myseed=1234L,
                                      X_train = X_train, y_train = y_train,
                                      X_tune=X_tune, y_tune=y_tune, # use tuning dataset
                                      search_times=5L, best_comb=NULL, # search 5 times
                                      impt=c(50L, 50L), top_rate=0.1,
                                      edged=NULL, vWeightd=NULL,
                                      epochs=1000L, fold=5L,
                                      plot=FALSE, verbose=TRUE, gpu=FALSE,
                                      normalization = TRUE)
# Use cross-validation, search 5 times
train_result_3 = iDeepViewLearn_train(python_path="~/.conda/envs/myenv/bin", myseed=1234L,
```

```
                                    X_train = X_train, y_train = y_train,
                                    X_tune=NULL, y_tune=NULL, # no tuning data, use CV
                                    search_times=5L, best_comb=NULL, # search 5 times
                                    impt=c(50L, 50L), top_rate=0.1,
                                    edged=NULL, vWeightd=NULL,
                                    epochs=1000L, fold=5L,
                                    plot=FALSE, verbose=TRUE, gpu=FALSE,
                                    normalization = TRUE)


######## Obtaining training result
# To get the indices of selected features for each view
selected_1 = train_result[[1]][[1]]
selected_2 = train_result[[1]][[2]]
# To get True Positive Rate and False Positive Rate for the first view
# The ground truth (gt) is that the first 50 variables are important
# and the 51~500 variables are not important.
# Note: python indices start with 0
gt_imp_1 = 0:49
gt_not_imp_1 = 50:499
all_var_1 = 0:499
unselected_1 = all_var_1[!(all_var_1
TP_1 = sum(selected_1
TN_1 = sum(unselected_1
FP_1 = sum(selected_1
FN_1 = sum(unselected_1
TPR_1 = TP_1 / (TP_1 + FN_1) * 100
FPR_1 = FP_1 / (FP_1 + TN_1) * 100
F_1 = TP_1 / (TP_1 + 0.5 * (FP_1 + FN_1)) * 100
# To get training accuracy
# Pass X_train and y_train to iDeepViewLearn_test() and obtain the accuracy
test_result = iDeepViewLearn_test(python_path = "~/.conda/envs/myenv/bin",
                                X_train = X_train, X_test = X_train,
                                y_train = y_train, y_test = y_train,
                                features = train_result[[1]],
                                model = train_result[[2]],
                                clf = train_result[[3]],
                                best_comb = train_result[[4]],
                                normalization = TRUE)
train_acc = test_result[[2]]
# To obtain G(Z), Z', and R(Z')
GZ = train_result[[6]]
Zprime = train_result[[7]]
RZprime = train_result[[8]]

######## Import Laplacian data example
library(iDeepViewLearn)
data("iDeepViewLearn_Laplacian_data_example")

######## Train iDeepViewLearn-Laplacian
# Use validation dataset and default hyperparameters to train an iDeepViewLearn model.
# By ground truth of the data example, 21 variables are truly important. See gt.
# Here we chose to NOT normalize data.
train_result_L = iDeepViewLearn_train(python_path="~/.conda/envs/myenv/bin",
                                    myseed=1234L,
                                    X_train = X_train, y_train = y_train,
                                    X_tune=X_tune, y_tune=y_tune,
```

```
                                      search_times=0L, best_comb = NULL,
                                      impt=c(21L,21L), top_rate=0.1,
                                      edged=edge_ls, vWeightd=weight_vec_ls,
                                      epochs=1000L, fold=NULL,
                                      plot=FALSE, verbose=TRUE, gpu=FALSE,
                                      normalization = TRUE)
# The followed analysis should be similar.
```

Index