Package ‘RandMVLearn’

August 27, 2025

Type Package

Title Randomized Nonlinear Association and Prediction Methods for Multi-view Learning with Feature Selection

Version 0.1.0

Author Sandra E. Safo (ssafo@umn.edu) and Leif Verace (verac008@umn.edu)

Maintainer Sandra Safo <ssafo@umn.edu>

Url https://github.com/lasandrall/RandMVLearn

Description We present scalable randomized kernel methods for jointly associating data from multiple sources and
simultaneously predicting an outcome or classifying a unit into one of two or more classes.
The proposed methods model nonlinear relationships in multiview data together with predicting a clinical outcome and are capable of identifying variables or groups of variables that best contribute to the relationships among the views. We use the idea that random Fourier bases can approximate shift-invariant kernel functions to construct nonlinear mappings of each view and we use these mappings and the outcome variable to learn view-independent low-dimensional representations.

Imports reticulate, doParallel, parallel

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Depends R (>= 3.5.0)

Contents

convert_to_df *Convert a Tensor to a Data Frame*

Description

This function converts a PyTorch tensor (or a list of tensors) into an R data frame. If the tensor has at least two columns, the last column is assumed to be a class label.

Usage

```
convert_to_df(item)
```

Arguments

item	A PyTorch tensor, a list of tensors, or another R object.

Value

A data frame or a list of data frames.

Examples

```
## Not run:
  torch <- reticulate::import("torch")
  tensor <- torch$randn(c(10, 3))  # Random 10x3 tensor
  df <- convert_to_df(tensor)
  print(df)

## End(Not run)
```

convert_to_tensor *Convert a List or Data Frame to a PyTorch Tensor*

Description

This function converts an R list or data frame into a PyTorch tensor using ‘torch$from_numpy()‘.

Usage

```
convert_to_tensor(lst)
```

Arguments

lst	A data frame, list of data frames, or numeric matrix to convert into a tensor.

Value

A PyTorch tensor or a list of tensors.

Examples

```
## Not run:
  torch <- reticulate::import("torch")
  df <- data.frame(A = rnorm(10), B = rnorm(10))
  tensor <- convert_to_tensor(df)
  print(tensor)

## End(Not run)
```

`createVirtualenv` *Create a Python virtual environment with necessary packages.*

Description

This function is used to create a Python virtual environment (called "RandMVLearn_env") and installs necessary packages such as torch, numpy, and scikit-learn to the environment. Once created, the environment will automatically be used upon loading the package.

Usage

```
createVirtualenv()
```

Details

If there is an error installing the Python packages, try restarting your computer and running R/RStudio as administrator.

If the virtual environment is not created, the user's default system Python installation will be used. In this case, the user will need to have the following packages in their main local Python installation:

- torch
- matplotlib
- joblib
- scikit-learn
- numpy

Alternatively, the user can use their own virtual environment with reticulate by activating it with reticulate::use_virtualenv() or a similar function prior to loading RandMVLearn.

Value

The function will return a list with 2 entries containing training and testing data. The following arguments are needed if you want to proceed with testing or prediction.

`TrainData`	A list containing training Views X and outcome Y.
`TestData`	A list containing testing Views X and outcome Y.

Author(s)

Leif Verace

See Also

RandMVLearnR RandMVPredict

Examples

```
###### create Python virtual environment "RandMVLearn_env"
createVirtualenv()
```

---

cvRandMVLearnGroup *Cross-validation for randomized multiview learning with group information*

Description

Performs nfolds cross-validation to select optimal tuning parameter for randomized multiview learning based on training data. Trains a randomized nonlinear model for simultaneous association and prediction of multiview data on each cross-validated fold and predicts outcome for the test fold. Optimal tuning parameter chosen based on minimum avarage cross-validated error. If you want to apply optimal tuning parameters to testing or training data, you may use RandMVLearnGroup. Use this function if there is prior information (group information) for at least one view. Currently works for categorical or continuous outcome. Returns selected features, groups, model trained, view-independent low-dimensional representation(s), which could be used in subsequent analyses.

Usage

```
cvRandMVLearnGroup(
  myseed = 1234L,
  Xdata = Xdata,
  Y = Y,
  hasGroupInfo = GroupInfo,
  GroupIndices = groupsd,
  rhoLower = NULL,
  rhoUpper = NULL,
  myeta = NULL,
  ncomponents = NULL,
  num_features = NULL,
  outcometype = NULL,
  kernel_param = NULL,
  mylambda = NULL,
  numbercores = NULL,
  gridMethod = NULL,
  nfolds = 3L,
  ngrid = 8L,
  max_iter_nsir = NULL,
  max_iter_PG = NULL,
  update_thresh_nsir = NULL,
  update_thresh_PG = NULL,
  standardize_Y = FALSE,
  standardize_X = FALSE,
  omegaweight = 0.5
)
```

Arguments

`myseed`	An integer to set a seed. Need to append a letter L to the integer, for example 1234L. This argument can be NULL.
`Xdata`	A list of d elements, where d is the number of views for the training data. Each element is a view with dimension $n \times p^d$, where observations are on the rows and features are on the columns. The number of samples are the same across all views but p^d can be different.
`Y`	An $n \times q$ matrix of responses. Currently allows for categorical and continuous outcomes. Allows for multiple continuous outcomes, so that $q > 1$.
`hasGroupInfo`	A list of d elements indicating whether or not the dth view has prior information. If view d has prior information, denote as 1, otherwise 0.
`GroupIndices`	A list of d elements containing group information. If there is no group information for view d, enter NULL. Group information for view d is a matrix with two columns. The first column is the group number $1, 2, ...$ and the second column is the variables in that group. Method works for non-overlapping groups. .
`rhoLower`	A list of d lower bound values for $\rho > 0$. $\rho > 0$ controls the amount of sparsity, for a fixed η. Default is 10^-7.
`rhoUpper`	list of d upper bound values. $\rho > 0$ controls the amount of sparsity, for a fixed η. Default is 10^-5. Users are encouraged to try different uppder bounds as this value may be too large or too small.
`myeta`	A list of d entries. $0 \leq \eta \leq 1$ allows to select groups and variables within groups, for views with group information. This parameter is not tuned. For a fixed ρ, smaller values encourage grouping (i.e. i.e. more nonzero groups are selected) and individual variable selection within groups (i.e more variables tend to have nonzero coefficients within groups); larger variables discourage group selection and encourage sparsity within group. If view d has no group information, set as 0. Default is 0.5 when there's group information and 0 when there's no group information.
`ncomponents`	An integer for number of low-dimensional components. Need to append a letter L to the integer. Set to 0L or NULL to allow algorithm to adaptively choose the number of components.
`num_features`	An integer for number of random mappings, typically less than the number of samples. Need to append a letter L to the integer. This argument can be NULL. If NULL, the algorithm will set it to 300 if $n \geq 1000$ or $n/2$ if $n < 1000$.
`outcometype`	A string for the type of outcome. Required. Either "categorical" or "continuous". If not specified, will default to continuous, which might not be ideal for the type of outcome.
`kernel_param`	A list of d integers specifying the kernel parameters for the Gaussian kernel. If NULL, algorithm will choose kernel parameters for each view using median heuristic.
`mylambda`	A list of d integers specifying the regularization parameters controlling the trade-off between model fit and complexity. Default is 1 for each view.
`numbercores`	Number of cores to be used for parallel computing. Defaults to half the size of the system cores.
`gridMethod`	GridSearch or RandomSearch. Optimize tuning parameters over full grid or random grid. Default is Random Search.

`nfolds`	A list of d integers specifying the kernel parameters for the Gaussian kernel. If NULL, algorithm will choose kernel parameters for each view using median heuristic.
`ngrid`	A list of d integers specifying the regularization parameters controlling the trade-off between model fit and complexity. Default is 1 for each view.
`max_iter_nsir`	An integer indicating the number of iterations for the alternating minimization algorithm. Need to append a letter L to the integer. If NULL, defaults to 500.
`max_iter_PG`	An integer indicating the number of iterations for the accelerated projected gradient descent algorithm for sparse learning. Need to append a letter L to the integer. If NULL, defaults to 500.
`update_thresh_nsir`	Threshold for convergence of alternating minimization algorithm. Defaults to 10^{-6}.
`update_thresh_PG`	Threshold for convergence of accelerated projected gradient descent algorithm. Defaults to 10^{-6}.
`standardize_Y`	TRUE or FALSE. If TRUE, Y will be standardized to have mean zero and variance one. Applicable to continuous outcome. Defaults to FALSE, at which point Y is centered.
`standardize_X`	TRUE or FALSE. If TRUE, each variable in each view will be standardized to have mean zero and variance one. Defaults to FALSE.
`omegaweight`	A parameter between 0 and 1, exclusive, balancing the association and prediction terms. Defaults to 0.5.

Details

Please refer to main paper for more details. Paper can be found here: https://arxiv.org/abs/2304.04692

Value

The function will return a list of elements. To see the elements, use double square brackets. See below for more detail of the main output. Some of the arguments are needed to proceed with testing or prediction.

`Z`	A list of $d, d = 1, \ldots, D$ randomized nonlinear feature data for each view.
`Ghat`	A matrix of $n \times r$ joint nonlinear low-dimensional representations learned from the training data. Here, r is the number of latent components. This matrix could be used for further downstream analyses such as clustering.
`myb`	A list of $d, d = 1, \ldots, D$ uniform random variables used in generating random features.
`gamma`	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry is a length-p^d vector of probability estimate for each variable. A value of 0 indicates the variable is not selected.
`Var_selection`	A list with $d, d = 1, \ldots, D$ entries of variable selection for each view . Each entry is a length-p^d indicator vector. A value of 0 indicates the variable is not selected and a value of 1 indicates the variable is selected.
`GroupSelection`	A list with $d, d = 1, \ldots, D$ entries of group selection for each view . Each entry contains a $G \times 2$ matrix, G is the number of groups, the first column is the group indices and the second column is the number of variables selected in that group.

	If no variable is selected, we assign a zero value for that group. If there's no group information for view d, the dth entry is assigned a zero value.
myepsilon	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry contains the inverse Fourier transform for the Guassian Kernel.
Ahat	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry is a matrix of coeffients.
thetahat	A $M \times q$ tensor of estimated regression coefficients, where M is the number of random features used in training.
num_features	An integer for number of random mappings used in training, typically less than the number of samples.
standardize_Y	TRUE or FALSE. If TRUE, Y was standardized to have mean zero and variance one during training of the model. Applicable to continuous outcome. If FALSE, Y was centered to have mean zero. Defualts to FALSE if NULL.
standardize_X	TRUE or FALSE. If TRUE, each variable in each view was standardized to have mean zero and variance one when training the model. Defualts to FALSE if NULL.
ncomponents	An integer for number of low-dimensional components used in training.
myrho	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry is the optimum sparsity penalty ρ.

#'

Author(s)

Sandra E. Safo

References

Sandra E. Safo and Han Lu (2024) Scalable Randomized Kernel Methods for Multiview Data Integration and Prediction Accepted in Biostatistics. https://arxiv.org/abs/2304.04692

See Also

generateData, RandMVPredict, RandMVLearnGroup, RandMVLearnR

Examples

```
####generate train and test data with binary outcome- refer to manuscript for data generation
createVirtualenv()

outcometype='categorical'
mydata=generateData(n1=500L,n2=200L,p1=1000L,p2=1000L,sigmax11=0.1,
                    sigmax12=0.1,sigmax2=0.2,outcometype=outcometype)

#create a list of two views
X1=mydata[["TrainData"]][[1]][["X"]][[1]]
X2=mydata[["TrainData"]][[1]][["X"]][[2]]
Xdata=list(X1,X2)

Y=mydata[["TrainData"]][[1]][["Y"]]

################# train with adaptively chosen number of components and number of features
####If hasGroupInfo is NULL, will call RandMVLearn (i.e. no group information)
```

```
GroupInfo=NULL
RandMVTrain.Adapt=cvRandMVLearnGroup(myseed=1234L,Xdata=Xdata, Y=Y, hasGroupInfo=GroupInfo,
                              GroupIndices=NULL, rhoLower=NULL,rhoUpper=NULL,myeta=NULL,
                             ncomponents=NULL,num_features=NULL,outcometype=outcometype,
                                    gridMethod='RandomSearch',nfolds=NULL,ngrid=NULL )

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       myEstimates=RandMVTrain.Adapt)
#obtain test error
test.error=predictY["Test Error"]


######assume all views have group information
GroupInfo=list(1,1)
g1=cbind(matrix(1,nrow=20,ncol=1),1:20)
g2=cbind(matrix(2,nrow=1000-20,ncol=1),21:1000)
groupsd=list(rbind(g1,g2),rbind(g1,g2))
myrhomin=list(0.0000001,0.0000001)
myrhomax=list(0.000009,0.000009)
start_time =Sys.time()
cv.RandMVTrainAdapt=cvRandMVLearnGroup(myseed=1234L,Xdata=Xdata, Y=Y, hasGroupInfo=GroupInfo,
                         GroupIndices=groupsd, rhoLower=myrhomin,rhoUpper=myrhomax,myeta=NULL,
                             ncomponents=NULL,num_features=NULL,outcometype=outcometype,
                                      gridMethod='RandomSearch',nfolds=5L,ngrid=8L)
end_time=Sys.time()

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                      myEstimates=cv.RandMVTrainAdapt)
#obtain test error
test.error=predictY["Test Error"]

#View GroupsSelected for Views 1 and 2
GroupsSelected1=cv.RandMVTrainAdapt[["GroupSelection"]][[1]]
GroupsSelected2=cv.RandMVTrainAdapt[["GroupSelection"]][[2]]

#View Variables Selected
VarSelected1=cv.RandMVTrainAdapt[["Var_selection"]][[1]]
VarSelected2=cv.RandMVTrainAdapt[["Var_selection"]][[2]]
```

---

generateData *Generate binary or continuous nonlinear multiview data*

Description

This function is used to generate binary or continuous nonlinear data for two views. Please refer to the manuscript for data generation process. Function can generate data with multiple continuous outcomes.

Usage

```
generateData(
  myseed = 1234L,
  n1 = 500L,
  n2 = 200L,
  p1 = 1000L,
  p2 = 1000L,
  nContVar = 1L,
  sigmax1 = 0.1,
  sigmax2 = 0.1,
  sigmay = 0.1,
  sigmax11 = 0.1,
  sigmax12 = 0.1,
  ncomponents = 3L,
  nReplicate = 1L,
  outcometype = "continuous"
)
```

Arguments

`myseed`	An integer to set a seed. Need to append a letter L to the integer, for example 1234L. This argument can be NULL.
`n1`	An even integer for number of samples. If outcometype is continuous, this is the number of samples for each view. If outcometype is categorical, this is the number of samples for class 1. Need to append a letter L to the integer. Can be set to NULL.
`n2`	An even integer for number of samples in class 2 if outcome type is categorical. If outcometype is continuous, this is not used. Need to append a letter L to the integer. Can be set to NULL.
`p1`	An integer for number of variables in view 1. Need to append a letter L to the integer. Can be set to NULL.
`p2`	An integer for number of variables in view 2. Need to append a letter L to the integer. Can be set to NULL. For this data generation example, $p1 = p2$ but the method allows for different variable dimensions.
`nContVar`	An integer for number of continuous outcome variables. If outcometype is categorical, not used. Need to append a letter L to the integer. Can be set to NULL. Defaults to 1.
`sigmax1`	Variance for View 1. Refer to manuscript for more details.
`sigmax2`	Variance for View 2. Refer to manuscript for more details.
`sigmay`	Variance for continuous outcome. Refer to manuscript for more details.
`sigmax11`	Variance for Class 1 for binary data generation. Refer to manuscript for more details.
`sigmax12`	Variances for Class 2 for binary data generation. Refer to manuscript for more details.

`ncomponents` An integer for number of low-dimensional components. Need to append a letter L to the integer. Can be set to NULL. Defaults to 3.

`nReplicate` An integer for number of replicates. Need to append a letter L to the integer. Can be set to NULL. Defaults to 1.

`outcometype` A string for the type of outcome. Required. Either "categorical" or "continuous". If not specified, will default to continuous.

Details

Please refer to main paper for more details. Paper can be found here: https://arxiv.org/abs/2304.04692

Value

The function will return a list with 2 entries containing training and testing data. The following arguments are needed if you want to proceed with testing or prediction.

`TrainData` A list containing training Views X and outcome Y.

`TestData` A list containing testing Views X and outcome Y.

Author(s)

Sandra E. Safo

References

Sandra E. Safo and Han Lu (2024) Scalable Randomized Kernel Methods for Multiview Data Integration and Prediction Accepted in Biostatistics. https://arxiv.org/abs/2304.04692

See Also

RandMVLearnR RandMVPredict

Examples

```
 ####### generate train and test data with binary outcome- refer to manuscript for data generation
createVirtualenv()

outcometype='categorical'
mydata=generateData(n1=500L,n2=200L,p1=1000L,p2=1000L,sigmax11=0.1,
                    sigmax12=0.1,sigmax2=0.2,outcometype=outcometype)

#create a list of two views for training data
X1=mydata[["TrainData"]][[1]][["X"]][[1]]
X2=mydata[["TrainData"]][[1]][["X"]][[2]]
Xdata=list(X1,X2)

#training outcome
Y=mydata[["TrainData"]][[1]][["Y"]]

#testing data and outcome
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)
```

```
Ytestdata=mydata[["TestData"]][[1]][["Y"]]

####### generate train and test data with two continuous outcomes- refer to manuscript for data generation

outcometype='continuous'
mydata=generateData(n1=500L,n2=200L,p1=1000L,p2=1000L,sigmax11=0.1,nContVar=2L,
                    sigmax12=0.1,sigmax2=0.2,outcometype=outcometype)
```

RandMVLearnGroup *Trains a randomized nonlinear model for simultaneous association and prediction of multiview data when there is group information for one or more views.*

Description

Trains a randomized nonlinear model for simultaneous association and prediction of multiview data. Use this function if there is prior information (group information) for at least one view. Currently works for categorical or continuous outcome. Returns selected features, groups, model trained, view-independent low-dimensional representation(s), which could be used in subsequent analyses.

Usage

```
RandMVLearnGroup(
  myseed = 1234L,
  Xdata = Xdata,
  Y = Y,
  hasGroupInfo = GroupInfo,
  GroupIndices = groupsd,
  myrho = NULL,
  myeta = NULL,
  ncomponents = NULL,
  num_features = NULL,
  outcometype = NULL,
  kernel_param = NULL,
  mylambda = NULL,
  max_iter_nsir = NULL,
  max_iter_PG = NULL,
  update_thresh_nsir = NULL,
  update_thresh_PG = NULL,
  standardize_Y = FALSE,
  standardize_X = FALSE,
  omegaweight = 0.5
)
```

Arguments

myseed	An integer to set a seed. Need to append a letter L to the integer, for example 1234L. This argument can be NULL.
Xdata	A list of d elements, where d is the number of views for the training data. Each element is a view with dimension $n \times p^d$, where observations are on the rows and features are on the columns. The number of samples are the same across all views but p^d can be different.

Y	An $n \times q$ matrix of responses. Currently allows for categorical and continuous outcomes. Allows for multiple continuous outcomes, so that $q > 1$.
hasGroupInfo	A list of d elements indicating whether or not the dth view has prior information. If view d has prior information, denote as 1, otherwise 0.
GroupIndices	A list of d elements containing group information. If there is no group information for view d, enter NULL. Group information for view d is a matrix with two columns. The first column is the group number $1, 2, ...$ and the second column is the variables in that group. Method works for non-overlapping groups. .
myrho	A list of d entries. $\rho > 0$ controls the amount of sparsity, for a fixed η.
myeta	A list of d entries. $0 \leq \eta \leq 1$ allows to select groups and variables within groups, for views with group information. This parameter is not tuned. For a fixed ρ, smaller values encourage grouping (i.e. i.e. more nonzero groups are selected) and individual variable selection within groups (i.e more variables tend to have nonzero coefficients within groups); larger variables discourage group selection and encourage sparsity within group. If view d has no group information, set as 0. Default is 0.5 when there's group information and 0 when there's no group information.
ncomponents	An integer for number of low-dimensional components. Need to append a letter L to the integer. Set to 0L or NULL to allow algorithm to adaptively choose the number of components.
num_features	An integer for number of random mappings, typically less than the number of samples. Need to append a letter L to the integer. This argument can be NULL. If NULL, the algorithm will set it to 300 if $n \geq 1000$ or $n/2$ if $n < 1000$.
outcometype	A string for the type of outcome. Required. Either "categorical" or "continuous". If not specified, will default to continuous, which might not be ideal for the type of outcome.
kernel_param	A list of d integers specifying the kernel parameters for the Gaussian kernel. If NULL, algorithm will choose kernel parameters for each view using median heuristic.
mylambda	A list of d integers specifying the regularization parameters controlling the trade-off between model fit and complexity. Default is 1 for each view.
max_iter_nsir	An integer indicating the number of iterations for the alternating minimization algorithm. Need to append a letter L to the integer. If NULL, defaults to 500.
max_iter_PG	An integer indicating the number of iterations for the accelerated projected gradient descent algorithm for sparse learning. Need to append a letter L to the integer. If NULL, defaults to 500.
update_thresh_nsir	Threshold for convergence of alternating minimization algorithm. Defaults to 10^{-6}.
update_thresh_PG	Threshold for convergence of accelerated projected gradient descent algorithm. Defaults to 10^{-6}.
standardize_Y	TRUE or FALSE. If TRUE, Y will be standardized to have mean zero and variance one. Applicable to continuous outcome. Defaults to FALSE, at which point Y is centered.
standardize_X	TRUE or FALSE. If TRUE, each variable in each view will be standardized to have mean zero and variance one. Defaults to FALSE.
omegaweight	A parameter between 0 and 1, exclusive, balancing the association and prediction terms. Defaults to 0.5.

Details

Please refer to main paper for more details. Paper can be found here: https://arxiv.org/abs/2304.04692

Value

The function will return a list of 17 elements. To see the elements, use double square brackets. See below for more detail of the main output. Some of the arguments are needed to proceed with testing or prediction.

Z	A list of $d, d = 1, \ldots, D$ randomized nonlinear feature data for each view.
Ghat	A matrix of $n \times r$ joint nonlinear low-dimensional representations learned from the training data. Here, r is the number of latent components. This matrix could be used for further downstream analyses such as clustering.
myb	A list of $d, d = 1, \ldots, D$ uniform random variables used in generating random features.
gamma	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry is a length-p^d vector of probability estimate for each variable. A value of 0 indicates the variable is not selected.
Var_selection	A list with $d, d = 1, \ldots, D$ entries of variable selection for each view . Each entry is a length-p^d indicator vector. A value of 0 indicates the variable is not selected and a value of 1 indicates the variable is selected.
GroupSelection	A list with $d, d = 1, \ldots, D$ entries of group selection for each view . Each entry contains a $G \times 2$ matrix, G is the number of groups, the first column is the group indices and the second column is the number of variables selected in that group. If no variable is selected, we assign a zero value for that group. If there's no group information for view d, the dth entry is assigned a zero value.
myepsilon	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry contains the inverse Fourier transform for the Guassian Kernel.
Ahat	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry is a matrix of coeffients.
thetahat	A $M \times q$ tensor of estimated regression coefficients, where M is the number of random features used in training.
num_features	An integer for number of random mappings used in training, typically less than the number of samples.
standardize_Y	TRUE or FALSE. If TRUE, Y was standardized to have mean zero and variance one during training of the model. Applicable to continuous outcome. If FALSE, Y was centered to have mean zero. Defualts to FALSE if NULL.
standardize_X	TRUE or FALSE. If TRUE, each variable in each view was standardized to have mean zero and variance one when training the model. Defualts to FALSE if NULL.
ncomponents	An integer for number of low-dimensional components used in training.

Author(s)

Sandra E. Safo

References

Sandra E. Safo and Han Lu (2024) Scalable Randomized Kernel Methods for Multiview Data Integration and Prediction Accepted in Biostatistics. https://arxiv.org/abs/2304.04692

See Also

generateData RandMVPredict

Examples

```
#### generate train and test data with binary outcome- refer to manuscript for data generation
createVirtualenv()
outcometype='categorical'
mydata=generateData(n1=500L,n2=200L,p1=1000L,p2=1000L,sigmax11=0.1,
                   sigmax12=0.1,sigmax2=0.2,outcometype=outcometype)

#create a list of two views
X1=mydata[["TrainData"]][[1]][["X"]][[1]]
X2=mydata[["TrainData"]][[1]][["X"]][[2]]
Xdata=list(X1,X2)

Y=mydata[["TrainData"]][[1]][["Y"]]

################# train with fixed number of components and number of features
######assume all views have group information
GroupInfo=list(1,1)
g1=cbind(matrix(1,nrow=20,ncol=1),1:20) #signal variables form group 1
g2=cbind(matrix(2,nrow=1000-20,ncol=1),21:1000) #noise variables form group 2
groupsd=list(rbind(g1,g2),rbind(g1,g2))

myrho.parameter=list(0.000008, 0.000008)
RandMVGroupTrain=RandMVLearnGroup(myseed=1234L,Xdata=Xdata, Y=Y,hasGroupInfo=GroupInfo,
                      GroupIndices=groupsd,myrho=myrho.parameter,myeta=NULL,ncomponents=5L,
                      num_features =300L, outcometype=outcometype, standardize_X = FALSE)

#View GroupsSelected for Views 1 and 2 and number of variables selected in groups
GroupsSelected1=RandMVGroupTrain[["GroupSelection"]][[1]]
GroupsSelected2=RandMVGroupTrain[["GroupSelection"]][[2]]

#View Variables Selected
VarSelected1=RandMVGroupTrain[["Var_selection"]][[1]]
VarSelected2=RandMVGroupTrain[["Var_selection"]][[2]]

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                      myEstimates=RandMVGroupTrain)
#obtain test error
test.error=predictY["Test Error"]

######assume only view 1 has group information
GroupInfo=list(1,0)
g1=cbind(matrix(1,nrow=20,ncol=1),1:20) #signal variables form group 1
g2=cbind(matrix(2,nrow=1000-20,ncol=1),21:1000) #noise variables form group 2
groupsd=list(rbind(g1,g2), NULL)
```

```
myrho.parameter=list(0.000008, 0)
RandMVGroupTrain=RandMVLearnGroup(myseed=1234L,Xdata=Xdata, Y=Y,hasGroupInfo=GroupInfo,
                     GroupIndices=groupsd,myrho=myrho.parameter,myeta=NULL,ncomponents=5L,
                      num_features =300L, outcometype=outcometype, standardize_X = FALSE)


#View groups selected for View 1 and number of variables selected in groups
GroupsSelected=RandMVGroupTrain[["GroupSelection"]][[1]]

#count number of nonzero variables in each view
VarSel1=sum(RandMVGroupTrain$Var_selection[[1]]==1)
VarSel2=sum(RandMVGroupTrain$Var_selection[[2]]==1)

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       myEstimates=RandMVGroupTrain)
#obtain test error
test.error=predictY["Test Error"]

################# train with adaptively choosen number of components and number of features
######assume all views have group information
GroupInfo=list(1,1)
g1=cbind(matrix(1,nrow=20,ncol=1),1:20) #signal variables form group 1
g2=cbind(matrix(2,nrow=1000-20,ncol=1),21:1000) #noise variables form group 2
groupsd=list(rbind(g1,g2),rbind(g1,g2))

myrho.parameter=list(0.000008, 0.000008)
RandMVGroupTrain.Adapt=RandMVLearnGroup(myseed=1234L,Xdata=Xdata, Y=Y,hasGroupInfo=GroupInfo,
                         GroupIndices=groupsd,myrho=myrho.parameter,myeta=NULL,ncomponents=NULL,
                         num_features =NULL, outcometype=outcometype, standardize_X = FALSE)

#View GroupsSelected for Views 1 and 2 and number of variables selected in groups
GroupsSelected1=RandMVGroupTrain.Adapt[["GroupSelection"]][[1]]
GroupsSelected2=RandMVGroupTrain.Adapt[["GroupSelection"]][[2]]

#View Variables Selected
VarSelected1=RandMVGroupTrain.Adapt[["Var_selection"]][[1]]
VarSelected2=RandMVGroupTrain.Adapt[["Var_selection"]][[2]]

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                      myEstimates=RandMVGroupTrain.Adapt)
#obtain test error
test.error=predictY["Test Error"]
```

RandMVLearnR *Trains a randomized nonlinear model for simultaneous association and prediction of multiview data*

Description

Trains a randomized nonlinear model for simultaneous association and prediction of multiview data. Use this function if there is no prior information (group information) for any of the views. Currently works for categorical or continuous outcome. Returns selected features, model trained, view-independent low-dimensional representation(s), which could be used in subsequent analyses.

Usage

```
RandMVLearnR(
  myseed = 1234L,
  Xdata = Xdata,
  Y = Y,
  ncomponents = NULL,
  num_features = NULL,
  outcometype = NULL,
  kernel_param = NULL,
  mylambda = NULL,
  max_iter_nsir = NULL,
  max_iter_PG = NULL,
  update_thresh_nsir = NULL,
  update_thresh_PG = NULL,
  standardize_Y = FALSE,
  standardize_X = FALSE,
  omegaweight = 0.5
)
```

Arguments

myseed	An integer to set a seed. Need to append a letter L to the integer, for example 1234L. This argument can be NULL.
Xdata	A list of d elements, where d is the number of views for the training data. Each element is a view with dimension $n \times p^d$, where observations are on the rows and features are on the columns. The number of samples are the same across all views but p^d can be different.
Y	An $n \times q$ matrix of responses. Currently allows for categorical and continuous outcomes. Allows for multiple continuous outcomes, so that $q > 1$.
ncomponents	An integer for number of low-dimensional components. Need to append a letter L to the integer. Set to 0L or NULL to allow algorithm to adaptively choose the number of components.
num_features	An integer for number of random mappings, typically less than the number of samples. Need to append a letter L to the integer. This argument can be NULL. If NULL, the algorithm will set it to 300 if $n \geq 1000$ or $n/2$ if $n < 1000$.
outcometype	A string for the type of outcome. Required. Either "categorical" or "continuous". If not specified, will default to continuous, which might not be ideal for the type of outcome.

`kernel_param`	A list of d integers specifying the kernel parameters for the Gaussian kernel. If NULL, algorithm will choose kernel parameters for each view using median heuristic.
`mylambda`	A list of d integers specifying the regularization parameters controlling the trade-off between model fit and complexity. Default is 1 for each view.
`max_iter_nsir`	An integer indicating the number of iterations for the alternating minimization algorithm. Need to append a letter L to the integer. If NULL, defaults to 500.
`max_iter_PG`	An integer indicating the number of iterations for the accelerated projected gradient descent algorithm for sparse learning. Need to append a letter L to the integer. If NULL, defaults to 500.
`update_thresh_nsir`	Threshold for convergence of alternating minimization algorithm. Defaults to 10^{-6}
`update_thresh_PG`	Threshold for convergence of accelerated projected gradient descent algorithm. Defaults to 10^{-6}
`standardize_Y`	TRUE or FALSE. If TRUE, Y will be standardized to have mean zero and variance one. Applicable to continuous outcome. Defaults to FALSE, at which point Y is centered.
`standardize_X`	TRUE or FALSE. If TRUE, each variable in each view will be standardized to have mean zero and variance one. Defaults to FALSE.
`omegaweight`	A parameter between 0 and 1, exclusive, balancing the association and prediction terms. Defaults to 0.5.

Details

Please refer to main paper for more details. Paper can be found here: https://arxiv.org/abs/2304.04692

Value

The function will return a list of elements. To see the elements, use double square brackets. See below for more detail of the main output. Some of the arguments are needed to proceed with testing or prediction.

`Z`	A list of $d, d = 1, \dots, D$ randomized nonlinear feature data for each view.
`Ghat`	A matrix of $n \times r$ joint nonlinear low-dimensional representations learned from the training data. Here, r is the number of latent components. This matrix could be used for further downstream analyses such as clustering.
`myb`	A list of $d, d = 1, \dots, D$ uniform random variables used in generating random features.
`gamma`	A list with $d, d = 1, \dots, D$ entries for each view. Each entry is a length-p^d vector of probability estimate for each variable. A value of 0 indicates the variable is not selected.
`Var_selection`	A list with $d, d = 1, \dots, D$ entries of variable selection for each view . Each entry is a length-p^d indicator vector. A value of 0 indicates the variable is not selected and a value of 1 indicates the variable is selected.
`myepsilon`	A list with $d, d = 1, \dots, D$ entries for each view. Each entry contains the inverse Fourier transform for the Guassian Kernel.

Ahat	A list with $d, d = 1, \ldots, D$ entries for each view. Each entry is a matrix of coeffients.
thetahat	A $M \times q$ tensor of estimated regression coefficients, where M is the number of random features used in training.
num_features	An integer for number of random mappings used in training, typically less than the number of samples.
standardize_Y	TRUE or FALSE. If TRUE, Y was standardized to have mean zero and variance one during training of the model. Applicable to continuous outcome. If FALSE, Y was centered to have mean zero. Defualts to FALSE if NULL.
standardize_X	TRUE or FALSE. If TRUE, each variable in each view was standardized to have mean zero and variance one when training the model. Defualts to FALSE if NULL.
ncomponents	An integer for number of low-dimensional components used in training.

Author(s)

Sandra E. Safo

References

Sandra E. Safo and Han Lu (2024) Scalable Randomized Kernel Methods for Multiview Data Integration and Prediction Accepted in Biostatistics. https://arxiv.org/abs/2304.04692

See Also

generateData RandMVPredict

Examples

```
 ##### generate train and test data with binary outcome- refer to manuscript for data generation
createVirtualenv()
outcometype='categorical'
mydata=generateData(n1=500L,n2=200L,p1=1000L,p2=1000L,sigmax11=0.1,
                    sigmax12=0.1,sigmax2=0.2,outcometype=outcometype)

#create a list of two views
X1=mydata[["TrainData"]][[1]][["X"]][[1]]
X2=mydata[["TrainData"]][[1]][["X"]][[2]]
Xdata=list(X1,X2)

Y=mydata[["TrainData"]][[1]][["Y"]]

##### train with fixed number of components and number of features
RandMVTrain=RandMVLearnR(myseed=1234L,Xdata=Xdata, Y=Y, ncomponents=5L,num_features=300L,
                         outcometype=outcometype, standardize_X = FALSE)

#count number of nonzero variables in each view
VarSel1=sum(convert_to_df(RandMVTrain$Var_selection[[1]])==1)
VarSel2=sum(convert_to_df(RandMVTrain$Var_selection[[2]])==1)

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)
```

```
Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                      myEstimates=RandMVTrain)
#obtain test error
test.error=predictY["Test Error"]
##### train with adaptively chosen number of components and features
RandMVTrain.adapt=RandMVLearnR(myseed=1234L,Xdata=Xdata, Y=Y, ncomponents=NULL,
                              num_features=NULL,outcometype=outcometype,
                              standardize_X = FALSE)

#count number of nonzero variables in each view
VarSel1=sum(convert_to_df(RandMVTrain.adapt$Var_selection[[1]])==1)
VarSel2=sum(convert_to_df(RandMVTrain.adapt$Var_selection[[2]])==1)

#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       myEstimates=RandMVTrain.adapt)

#obtain test error. Note that for continuous outcomes, because Y is standardized
#or centered, the predicted Y is not in the scale of the original outcome so
#MSEs are not in the scale of original outcome.
#Can use Y_mean and Y_std to transform predicted Y to original scale, which
#can be used to calculate MSE in the scale of the original data.

test.error=predictY["Test Error"]
#####Predict an outcome using testing data assuming model has not been learned.
###Algorithm will first train the model.
predictY2=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       outcometype='categorical',myEstimates=NULL)
#obtain test error
test.error=predictY2["Test Error"]


########## When using your own data, remember to convert to tensor. Here's an example:
rm(list = ls())
outcometype='categorical'
data(COVIDData)
X1_ALL <- COVIDData[["Proteomic"]]
X2_ALL <- COVIDData[["RNAseq"]]
Y_ALL <- as.data.frame(COVIDData[["Clinical"]][,18])
train_index <- sample(seq_len(120), size = floor(0.7 * 120))
X1 <- convert_to_tensor(X1_ALL[train_index, ])
X2 <- convert_to_tensor(X2_ALL[train_index, ])
Xdata=list(X1,X2)
Y  <- convert_to_tensor(Y_ALL[train_index, ])
test_index <- setdiff(seq_len(120), train_index)
Xtest1 <- convert_to_tensor(X1_ALL[test_index, ])
Xtest2 <- convert_to_tensor(X2_ALL[test_index, ])
Xtestdata=list(Xtest1,Xtest2)
```

```
Ytestdata  <- convert_to_tensor(Y_ALL[test_index, ])

# Then, continue with RandMVLearnR...
# train with fixed number of components and number of features
RandMVTrain=RandMVLearnR(myseed=1234L,Xdata=Xdata, Y=Y, ncomponents=5L,num_features=100L,
                         outcometype=outcometype, standardize_X = FALSE)

#count number of nonzero variables in each view
VarSel1=sum(convert_to_df(RandMVTrain$Var_selection[[1]])==1)
VarSel2=sum(convert_to_df(RandMVTrain$Var_selection[[2]])==1)
predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       myEstimates=RandMVTrain)
test.error=predictY["Test Error"]
##### train with adaptively chosen number of components and features
RandMVTrain.adapt=RandMVLearnR(myseed=1234L,Xdata=Xdata, Y=Y, ncomponents=NULL,
                               num_features=NULL,outcometype=outcometype,
                               standardize_X = FALSE)

#count number of nonzero variables in each view
VarSel1=sum(convert_to_df(RandMVTrain.adapt$Var_selection[[1]])==1)
VarSel2=sum(convert_to_df(RandMVTrain.adapt$Var_selection[[2]])==1)
#####Predict an outcome using testing data assuming model has not been learned.
###Algorithm will first train the model.
predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       myEstimates=RandMVTrain.adapt)
test.error=predictY["Test Error"]
predictY2=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                        outcometype='categorical',myEstimates=NULL)
#obtain test error
test.error=predictY2["Test Error"]
```

`RandMVPredict` *Predicts a test joint nonlinear low-dimensional embedding and a test outcome.*

Description

This function predicts a joint low-dimensional nonlinear embedding from test data using a learned model. It also predicts a test outcome. Use this function if there is a learned model. If no learned model, the algorithm will first train a RandMVLearn model. Note that for continuous outcomes, because Y is standardized or centered, the predicted Y is not in the scale of the original outcome so MSEs are not in the scale of original outcome. Can use Y_mean and Y_std to transform predicted Y to original scale, which can be used to calculate MSE in the scale of the original data.

Usage

```
RandMVPredict(
  Ytest = Ytestdata,
  Ytrain = Y,
  Xtest = Xtestdata,
  Xtrain = Xdata,
  outcometype = NULL,
  myEstimates = NULL,
```

```
    standardize_Y = NULL,
    standardize_X = NULL
)
```

Arguments

Ytest — An $ntest \times q$ matrix of responses. Currently allows for categorical and continuous outcomes. Allows for multiple continuous outcomes, so that $q > 1$. This will be compared with the predicted Ytest.

Ytrain — An $n \times q$ matrix of responses for training. Currently allows for categorical and continuous outcomes. Allows for multiple continuous outcomes, so that $q > 1$.

Xtest — A list of d elements, where d is the number of views for the testing data. Each element is a view with dimension $ntest \times p^d$, where observations are on the rows and features are on the columns. The number of samples are the same across all views but p^d can be different..

Xtrain — A list of d elements, where d is the number of views for the training data. Each element is a view with dimension $n \times p^d$, where observations are on the rows and features are on the columns. The number of samples are the same across all views but p^d can be different.

outcometype — A string for the type of outcome. Required if myEstimates is NULL. Either "categorical" or "continuous". If not specified, will default to continuous, which might not be ideal for the type of outcome.

myEstimates — A trained RandMVLearn model. Can be NULL. If NULL, algorithm will first train a RandMVLearn model.

standardize_Y — TRUE or FALSE. If TRUE, Y will be standardized to have mean zero and variance one during training of the model. Applicable to continuous outcome. If FALSE, Y will be centered to have mean zero. Defualts to FALSE if NULL.

standardize_X — TRUE or FALSE. If TRUE, each variable in each training view will be standardized to have mean zero and variance one when training the model. Testing data for each view will be standardized with the mean and variance from training data. Defaults to FALSE if NULL.

Details

Please refer to main paper for more details. Paper can be found here: https://arxiv.org/abs/2304.04692

Value

The function will return a list of 4 elements. To see the elements, use double square brackets. See below for more detail of the main output. The following arguments are needed if you want to proceed with testing or prediction.

predictedEstimates — A list with 4 entries of prediction estimates. "PredictedEstimates$predictedYtest" is the predicted Ytest. "PredictedEstimates$predictedYtest" is the predicted Y. "PredictedEstimates$EstErrorTrain" is the estimated train error. "PredictedEstimates$EstErrorTest" is the estimated test error

TrainError — Estimated training error

TestError — Estimated testing error

`Gtrain` A matrix of $n \times r$ joint nonlinear low-dimensional representations predicted from the training data. Here, r is the number of latent components. This matrix could be used for further downstream analyses such as clustering. This matrix is used together with Gtest to predict a test outcome.

`Gtest` A matrix of $ntest \times r$ predicted test joint nonlinear low-dimensional representations. Here, r is the number of latent components. This matrix is used together with Gtrain to predict a test outcome.

`Xtest_standardized`
A list of d elements, where d is the number of views for the testing data used for prediction. Each element is a view with dimension $ntest \times p^d$, where observations are on the rows and features are on the columns. This matrix coincides with Xtest if standardize_X is FALSE or NULL.

Author(s)

Sandra E. Safo

References

Sandra E. Safo and Han Lu (2024) Scalable Randomized Kernel Methods for Multiview Data Integration and Prediction Accepted in Biostatistics. https://arxiv.org/abs/2304.04692

See Also

generateData, RandMVLearnR, RandMVLearnGroup, cvRandMVLearnGroup

Examples

```
# generate train and test data with binary outcome- refer to manuscript for data generation
createVirtualenv()
outcometype='categorical'
mydata=generateData(n1=500L,n2=200L,p1=1000L,p2=1000L,sigmax11=0.1,
                   sigmax12=0.1,sigmax2=0.2,outcometype=outcometype)
#create a list of two views
X1=mydata[["TrainData"]][[1]][["X"]][[1]]
X2=mydata[["TrainData"]][[1]][["X"]][[2]]
Xdata=list(X1,X2)

Y=mydata[["TrainData"]][[1]][["Y"]]

##### train with fixed number of components and number of features
RandMVTrain=RandMVLearnR(myseed=1234L,Xdata=Xdata, Y=Y, ncomponents=5L,num_features=300L,
                         outcometype=outcometype, standardize_X = FALSE)
#####Predict an outcome using testing data assuming model has been learned
Xtest1=mydata[["TestData"]][[1]][["X"]][[1]]
Xtest2=mydata[["TestData"]][[1]][["X"]][[2]]
Xtestdata=list(Xtest1,Xtest2)

Ytestdata=mydata[["TestData"]][[1]][["Y"]]

predictY=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                       myEstimates=RandMVTrain)
#obtain test error
test.error=predictY[["Test Error"]]
```

```
#####Predict an outcome using testing data assuming model has not been learned.
###Algorithm will first train the model.
predictY2=RandMVPredict(Ytest=Ytestdata,Ytrain=Y,Xtest=Xtestdata,Xtrain=Xdata,
                        outcometype='categorical',myEstimates=NULL)

#obtain test error
test.error=predictY2[["Test Error"]]
```

Index