Package ‘vclust’

October 8, 2023

Type Package

Version 1.0

Date 2023-10-08

Title Validation and Generation of Latent Labels Using Unsupervised Clusters For the Use in Super- vised Learning

Description The program implements a 3-step approach to facilitate the use of unsupervised clustering with the focus on user-defined validation. In step 1, it conducts unsupervised clustering based on multivariate outcomes using existing clustering methods such as growth mixture modeling (GMM), model-based clustering (MBC), and K-means clustering. In step 2, in each clustering, latent classes or clusters are regrouped into two coarsened clusters using all possible ways of splits, resulting in a large pool of binary labels. These labels are systematically validated using a priori sets of validators defined by the users. In step 3, the validated and selected labels are deployed in supervised learning.

Roxygen list(markdown = TRUE)

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports cluster,
cvTools,
dplyr,
glmnet,
magrittr,
mclust,
mix,
MplusAutomation,
rhdf5,
sjmisc,
stats,
stringr,
utils

RoxygenNote 7.2.3

Author Booil Jo <booil@stanford.edu> [aut, cre]
Zetan Li <zetanli@stanford.edu> [aut]

Maintainer Booil Jo <booil@stanford.edu>

Suggests rmarkdown,
knitr

VignetteBuilder knitr

Depends R (>= 2.10)

R topics documented:

---

genclust *Conducts unsupervised clustering using existing clustering methods*

Description

Conducts unsupervised clustering using existing clustering methods.

Usage

```
genclust(
    model_type,
    class_range,
    min_units = 10,
    data_path,
    variable_names,
    naString = NULL,
    y_names,
    output_path_prefix = "output/",
    useobs,
    listwise_deletion_variables,
    clustering_data_fraction = 1,
    seed_num = c(seed_num_unsupervised_model = 4561234, seed_num_impute_missing = 4561234),
    kmeans_gap_stats_B = 50,
    kmeans_iter = 25,
    MBCtype,
    Ogroups_cutpoint,
    Ogroups_cutpoint_sign,
    Ogroups_cutpoint_max_min_mean,
    GMM_time_scores,
    GMM_covariates,
    GMM_random_intercept,
    GMM_trend = "quadratic",
    GMM_initial_starts = 500,
    GMM_final_optimizations = 50,
    GMM_ID = NULL,
    GMM_AUXILIARY = NULL
  )
```

Arguments

model_type	A string indicates a clustering method. Currently available options include GMM (growth mixture modeling), MBC (model-based clustering), and Kmeans. An additional option is Ogroups, where the user generates observed subgroups without conducting clustering. For GMM, commercial software Mplus is used (Muthén and Muthén, 1998- 2017). For MBC, R package mclust is used (Scrucca, Fop, Murphy, and Raftery, 2016). For K-means, R function kmeans is used. For instance, *model_type="MBC"*, *model_type="GMM"*, *model_type="Kmeans"*, or *model_type="Ogroups"*.
class_range	An integer vector specifies the desired number of clusters. For example, *class_range = 2:4* means clustering with 2, 3, and 4 clusters.
min_units	An integer indicates the minimum number of units in each cluster. If the number is less than the minimum, unsupervised clustering will stop. For example, when the unit of analysis is a person and *min_units=10*, clustering will stop if the smallest cluster has less than 10 people.
data_path	A string indicates the path of the input data. The data should be in csv format. For example, *"/Users/username/Desktop/inputdata.csv"* for Mac user or *"D:/folder/inputdata.csv"* for Windows user.
variable_names	A text string indicates names of variables from data_path, where names are separated by white spaces, or commas. For example, when input data has 9 columns, a1, a2, a3, a4, b1, b2, b3, cov1, and cov2, *variable_names = "a1,a2,a3,a4,b1,b2,b3,cov1,cov2"* or *variable_names="a1 a2 a3 a4 b1 b2 b3 cov1 cov2"*. These variable names will overwrite the original names when the data file already has variable names (i.e., header). The user can choose to use those original names by specifying *variable_names = NULL*.
naString	A string indicates what string is interpreted as NA value in the input data.
y_names	A string vector specifies the variable names used as multivariate outcomes in unsupervised clustering. When these are repeated measures used with GMM, they should be chronologically ordered. For example, *y_names = c(a1,a2,a3,a4).* When *model_type = Ogroups*, specified cupoints are directly applied to the variables listed under y_names.
output_path_prefix	A string indicates the output folder path of model results. The path should be absolute path (full path) when using Windows operation system. Remember to use "/" instead of "\" for the path.
useobs	A text string indicates observations to use. This one is the same as USEOBS in Mplus, which is a filter to screen out observations

	(rows for most cases). For example, to exclude observations with *id=9* and *id=13*, users may set *useobs = "(id ne 9) and (id ne 13)".*
listwise_deletion_variables	The user can specify listwise deletion based on specific variables listed in variable_names. For example, *listwise_deletion_variables = c("a1","b1")*. The user is also allowed to use listwise deletion with variables that are not being used in the genclust procedure. The use of useobs and listwise_deletion_variables is particularly important when *model_type=Ogroups* because it affects interpretation of subgroups.
clustering_data_fraction	A single value indicates the fraction of the samples to be used in unsupervised clustering. The value range is (0, 1] and the default value is 1.
seed_num	An integer vector indicates seed numbers for clustering and imputing missing data, which may affect the results depending on the clustering method. The vector should follow the below format. *seed_num = c(seed_num_clustering = 4561234, seed_num_impute_missing = 4561234)*
kmeans_gap_stats_B	An integer indicates the number of bootstrap samples (B) used to calculate gap statistics.
kmeans_iter	An integer indicates the number of iterations used in Kmeans clustering.
MBCtype	A string indicates the desired type of MBC model. One of the 14 types of constraints on the covariance matrix can be specified in line with mclust (EEE, EEI, EEV, EII, EVE, EVI, EVV, VEE, VEI, VEV, VII, VVE, VVI, VVV).
Ogroups_cutpoint	A numeric value/vector specifies a threshold/thresholds to form observed subgroups without conducting clustering.
Ogroups_cutpoint_sign	A character value/vector specifies a/multiple comparison operator(s). Available options include >=, <=, >, <, ==, GE, LE, GT, LT, EQ. When Ogroups_cutpoint is a vector with multiple cutpoints, the Ogroups_cutpoint_sign will be applied to each cutpoint.
Ogroups_cutpoint_max_min_mean	A character specifies what aggregation function is used to construct subgroups. Available options are max, min, and mean. When *model_type = Ogroups* and Ogroups_cutpoint is a single value, above three arguments are used to define subgroups. For example, if *y_names = c('a','b','c'), Ogroup_cutpoint = 12, Ogroups_cutpoint_sign=">="*, and *cutpoint_max_min_mean="max"*, all cases with *max(a, b, c) >= 12* will be assigned the value of 1, and the rest the value of 0. When *model_type = Ogroups* and Ogroups_cutpoint is a vector with multiple thresholds, Ogroups_cutpoint_max_min_mean will

	be ignored. For example, if *y_names = c('a','b','c'), Ogroup_cutpoint = c(12,13,14)*, and *Ogroups_cutpoint_sign = c('>=','<','>'),* all cases with *a >= 12*, *b < 13*, and *c > 14* will be assigned the value of 1, and the rest the value of 0. Formation of observed groups using more complex manipulations should be conducted externally before using this program.
GMM_time_scores	An integer vector specifies time measures at each time point when GMM is used. This one should have the same length as y_names. For example, *y_names = c(a1,a2,a3)* and *time_scores = c(0,1,2)* might mean that a1 is measured at baseline, a2 at 1 year, and a3 at 2 years from the baseline.
GMM_covariates	A string contains covariates used in clustering. Currently, this option applies only to GMM. For example, if *covariates="cov1 cov2 cov3"*, GMM runs with using these covariates as predictors of growth parameters (intercept and slope) and the cluster membership. If *covariates = NA*, GMM runs without covariates.
GMM_random_intercept	A Boolean variable indicates whether GMM is conducted allowing for a random intercept. If *GMM_random_intercept = TRUE*, GMM is conducted with allowing for a random intercept. If *GMM_random_intercept = FALSE*, GMM is conducted without allowing for a random intercept.
GMM_trend	For modeling of longitudinal trends, we use polynomial growth. Our program supports linear, quadratic, and cubic growth. For example, *GMM_trend="linear".* The current version of the program uses quadratic growth as a default.
GMM_initial_starts	An integer indicates the number of initial stage starting values in maximum likelihood optimization of GMM.
GMM_final_optimizations	An integer indicates the number of final stage optimizations in maximum likelihood optimization of GMM.
GMM_ID	A string specifies the variable name of ID in the input file. This ID variable will be included in the final .pp file.
GMM_AUXILIARY	A string vector specifies several additional variables which are intended to included in the final .pp file for subsequent analyses.

Value

Clustering results are saved in the folder specified in output_path_prefix. The summary will be provided as a csv file (genclust_results.csv).

References

Jo, B., Hastie, T. J., Li, Z., Youngstrom, E. A., Findling, R. L., & Horwitz, S. M. (2023). Reorienting Latent Variable Modeling for Supervised Learning. Multivariate Behavioral Research, 1-15.

---

validclust *Validate Binary Coarsened Clusters By Validators*

Description

Generates binary labels by regrouping clusters into two coarsened clusters using all possible ways of splits, and systematically validates the generated labels using a priori sets of validators defined by the users.

Usage

```
validclust(
  sync_genclust,
  info_genclust,
  useobs,
  if_CV,
  K_fold,
  seed_num_kfold,
  class_range,
  kappa_filter_maxN,
  kappa_filter_value,
  kappa_filter_results,
  validators
)
```

Arguments

sync_genclust	A Boolean variable indicates whether validation is conducted directly using the results from genclust. If *sync_genclust = TRUE*, all model and estimation specifications used in genclust will be automatically imported into validclust. If *sync_genclust = FALSE*, validclust is used as a standalone procedure, which is useful when using clustering models or methods that are not currently covered in genclust. In this case, the user is required to provide the details about the data and clustering results.
info_genclust	This argument will be applied when *sync_genclust = FALSE* and ignored when *sync_genclust = TRUE.* users can use the format *info_genclust = list(subcomponents).* There are a few subcomponents described below. • data_path: When *sync_genclust = FALSE*, the user needs to specify the folder path here that stores the data that contains clustering results and intended validators. A string indicates the path of the input data. The data should be in csv format. For example, *"/Users/username/Desktop/inputdata.csv"* for Mac user or *"D:/folder/inputdata.csv"* for Windows user. Use "/" instead of "\" for the path.

- output_path_prefix: The user needs to specify the folder path that will store validation results. The path should be absolute path (full path) when using Windows operation system.
- variable_names: When *sync_genclust = FALSE*, the user needs to specify variable names. A string vector indicates names of variables in the data specified in data_path. For example, *variable_names = c("e1","e2","e3","f1","f2","z1","q1","w1","w2","w3").* These variable names will overwrite the original names when the data file already has variables names (i.e., header). The user can choose to use those original names by specifying *variable_names = NULL.*
- naString:A string indicates what string is interpreted as NA value in the input data.
- cluster_names: A string vector indicates names of clusters. When *sync_genclust = FALSE*, the user needs to specify the names of clusters. For example, when validating outcome labels based on 3-cluster clustering, cluster_names = c("e1","e2","e3") and when based on 2-cluster clustering, cluster_names = c("f1","f2"). Note that the total should add up to 1. That is,

  $$e1 + e2 + e3 = 1$$

  and

  $$f1 + f2 = 1$$

  For example, when using cluster membership in probabilities (soft cluster- ing), an individual may have

  $$e1 = 0.3, e2 = 0.1, e3 = 0.6$$

  , which add up to 1. When using observed or hard cluster membership (one unit or person belongs to only one cluster), for a person who belongs to the third cluster,

  $$e1 = 0, e2 = 0, e3 = 1$$

  Note that, when *sync_genclust = FALSE*, the current version allows only one set of cluster names. For example, *cluster_names=c("e1","e2","e3").*

useobs — The user may specify a text string that indicates observations to use. For example, if we want to exclude observations with *x=9* and *x=13*, we can set *useobs = "(x ne 9) and (x ne 13)"*. If *sync_genclust = TRUE* and useobs has been already used, this argument can be used to specify additional observations to be excluded.

if_CV — A Boolean variable indicates whether K-fold cross validation is used in the validation step.

K_fold — An integer indicates the number of folds in cross-validation. It is applicable when *if_CV = TRUE*.

seed_num_kfold — When *if_CV = TRUE*, the user may provide a seed number for randomly dividing the data into K folds.

class_range — When *sync_genclust=TRUE*, the user can specify the desired range of clusters that will be included in validation. For example, with *class_range = 2:4*, clustering results with 2, 3, and 4 clusters will

be validated. When *sync_genclust=FALSE*, this argument will be ignored. Instead, the set of clusters defined in cluster_names will be validated.

kappa_filter_maxN

An integer indicates the maximum number of candidate labels to be validated. When it is NULL, no filter is applied. In this method, candidate labels are ranked by roughly calculating Cohen's Kappa between each candidate label and the primary validator (the first one on the validator list) without cross validation. For example, if *kappa_filter_maxN = 500*, only the top 500 labels based on Kappa will enter the validation procedure. The threshold is used to choose combinations with the best Cohen's kappa.

kappa_filter_value

An alternative way of limiting the number of candidate labels to be validated is to apply a minimum Kappa value. For example, if *kappa_filter_value = 0.15*, only the labels with Kappa value of 0.15 or greater will enter the validation procedure. When it is NULL, no filter is applied.

kappa_filter_results

The user can also specify the number of labels to be included in the summary file (i.e., validclust_results.csv). When it is NULL, all candidate labels that went through validation will appear in the summary.

validators

A list specifies one or more validator objects following the format below.

validators = list(
validator(subcomponents),
validator(subcomponents),
validator(subcomponents),
. . .)

The subcomponents include the following:

- listwise_deletion_variables: A vector indicates variables to be used to conduct listwise deletion. For example, *listwise_deletion_variables = c("a1","b1").* The user is allowed to use listwise deletion with variables that are not being used in the validclust procedure. The user is also allowed to use different variables for listwise deletion for different validators. Note that the rest of subcomponent arguments will no longer apply to the deleted cases. If *sync_genclust = TRUE* and *listwise_deletion_variables* has been already used in the genclust step, this argument can be used to specify additional deletion.
- validator_source_variables: A list of variables to be used to construct a validator. For example, *validator_source_variables = c("a1","a2","a3","a4").*
- validator_source_all_missing: An integer specifies which value to take when all variables listed in validator_source_variables are missing. The three possible options are NA, 1, or 0. If *validator_source_all_missing = NA*, the validator of these

individuals or units will be treated as missing. The default is 0.

- validator_type: A string indicates the type of each set of validators. There are 3 allowed types:
  "binary", when a single validator is already binary (0/1).
  "cutpoint", when a single binary validator needs to be created based on a cutpoint applied to a single or multiple variables.
  "combination", when a single continuous variable or a set of multiple variables (continuous and/or binary) are used together as a set of predictors of cluster membership.
- validator_cutpoint:A numeric value/vector specifies a threshold or multiple thresholds to create a binary validator.For example, *validator_cutpoint = 12,* or *validator_cutpoint = c(12,13,14).*
- validator_cutpoint_sign: A character value/vector specifies comparison operator(s) to be used with thresholds. Available options include >=, <=, >, <, ==, GE, LE, GT, LT, and EQ. When using a vector of multiple thresholds, the signs will be applied to each cutpoint.
- validator_cutpoint_max_min_mean:A string specifies a function to use to summarize multiple variables into a single validator. The options include max, min, and mean. For example, *max_min_mean = "max".*
  When validator_cutpoint is a single value, all cutpoint related arguments can be used together.For example, if *validator_source_variables = c('a','b','c'), validator_cutpoint = 12, validator_cutpoint_sign =">=",* and *validator_cutpoint_max_min_mean="max",* all cases with *max(a, b, c) >= 12* will be assigned the value of 1, and the rest the value of 0.
  When validator_cutpoint has multiple values, validator_max_min_mean will be ignored. For example, when *validator_source_variables = c('a','b','c'), validator_cutpoint = c(12,13,14),* and *validator_cutpoint_sign = c('>=','<','>'),*
  all cases with a>=12 and b<13 and c>14 will be assigned the value of 1, and the rest the value of 0.

The procedure validclust generates binary labels by regrouping all provided clusters into two coarsened clusters using all possible ways of splits. When *sync_genclust = TRUE*, this could lead to a very large pool of candidate labels to be validated, which will significantly slow down the validation procedure. There are three ways to reduce the pool of candidate labels using the following three arguments, class_range, kappa_filter_maxN, and kappa_filter_value.

Value

The validation results will be provided as a csv file (validclust_results.csv) in the user-specified folder. For each validator set and each candidate label, Cohen's Kappa, accuracy, sensitivity, specificity, and AUC estimates are provided (their means and standard errors if K-fold cross validation is used).

- Model_type: When *genclust_sync=TRUE*, the clustering method used in the genclust procedure (specified in model_type) will be shown here.
- Model_spec1 to Model_spec3: When *genclust_sync=TRUE*, specific model specifications used in the genclust procedure will be shown here.
- Cluster_n: The total number of clusters or classes in each clustering method.
- Cluster_names: When *genclust_sync=TRUE*, each cluster will be named starting with "P" and then numbered following the original cluster order in each clustering result in the genclust procedure. When *genclust_sync=FALSE*, the names and the order provided in cluster_names will be used.
- label_category1: In the validclust procedure, in each clustering, all clusters are split into two categories to generate binary labels. The clusters categorized in the first category will be shown under label_category1. The rest are categorized into the second category.
- Validator: Each validator in the order specified in *validators = list( ).*
- Kappa, sensitivity, specificity, accuracy, AUC: These are the measures of association between the validators and the binary labels generated based on clustering. When *if_CV = TRUE*, the provided values are the means across K folds.
- Kappa_SE, sensitivity_SE, specificity_SE, accuracy_SE, AUC_SE: When *if_CV = TRUE*, these are the standard deviations across K folds.

References

Jo, B., Hastie, T. J., Li, Z., Youngstrom, E. A., Findling, R. L., & Horwitz, S. M. (2023). Reorienting Latent Variable Modeling for Supervised Learning. Multivariate Behavioral Research, 1-15.

predclust *Conducts supervised learning treating a validated/selected cluster la- bel as a known input or output variable*

Description

Conducts supervised learning treating a validated/selected cluster label as a known input or output variable. A label identified as a good outcome from the validation step (validclust) is recommended to be used as a prediction output (Jo et al., in press). A label identified as a good predictor of an outcome is recommended to be used as a prediction input. Note that predclust can be used as a standalone procedure or in conjunction with genclust and/or validclust.

Usage

```
predclust(
  sync_genclust,
  sync_validclust,
  output_path_prefix,
  data_path,
  variable_names,
  naString,
  predictors_names,
  cluster_names,
  label_category1,
  cluster_label_position,
  outcome_obs,
  supervised_method,
  glmnet_specs,
  seed_numbers,
  useobs,
  listwise_deletion_variables,
  train_fraction,
  if_CV,
  K_fold,
  repeated_CV,
  if_PCD,
  r_PCD,
  lr_maxiter
)
```

Arguments

sync_genclust	A Boolean variable indicates whether predclust will use the input data and clustering results from genclust.
sync_validclust	A Boolean variable indicates whether predclust will use the input data and validation results from validclust. Our program doesn't support the case when *sync_validclust = T* and *sync_genclust = T*. Here are two counterparts for this case,

	1. When used *sync_genclust = T* in validclust, *sync_validclust = T* and *sync_genclust = T* is same to *sync_genclust = T* and *sync_validclust =F* 2. When used *sync_genclust = F* in validclust, *sync_validclust = T* and *sync_genclust = T* is same to *sync_genclust = F* and *sync_validclust = T*.
output_path_prefix	The user needs to specify the folder path that will store supervised learning results. The path should be absolute path (full path) when using Windows operation system. For example, *"/Users/username/Desktop"* for Mac user or *"D:/folder"* for Windows user. Use "/" instead of "\" for the path.
data_path	If *sync_genclust = FALSE* and *sync_validclust = FALSE*, the user is expected to specify the folder path that stores the data that will be used in predclust. The data should be in the csv format. The information provided here will supersede the information from genclust and validclust.
variable_names	When data_path is used, the user needs to specify variable names. For example, *variable_names = c('x','e1','e2','e3','f1','f2','z1','q1','w1','w2','w3','u1','u2').* These variable names will overwrite the original names when the data file already has variables names (i.e., header). The user can choose to use those original names by specifying *variable_names = NULL*.
naString	A string indicates what string is interpreted as NA value in the input data.
predictors_names	A string vector indicates names of variables to be used as predictors (input variables). For example, *predictors_names = c("x","w1","w2","w3","u1","u2").*
cluster_names	When data_path is not used, *sync_genclust = TRUE*, and *sync_validclust = FALSE*, the user is expected to use the cluster names from the summary of the genclust procedure provided in genclust_results.csv. For example, *cluster_names = c("P1","P2","P3")* When data_path is not used and *sync_validclust = TRUE*, the user is expected to use the cluster names from the summary of the validclust procedure provided in validclust_results.csv. When data_path is used, the user is expected to use the cluster names from the variables listed in variable_names. Note that, when using cluster membership in probabilities (soft clustering), the total should add up to 1. For example, an individual may have *e1=0.3, e2=0.1, e3=0.6*, which add up to 1. When using observed or hard cluster membership (one unit or person belongs to one cluster), for a person who belongs to the third cluster, *e1=0, e2=0, e3=1*.
label_category1	The user needs to specify which clusters will be categorized into the first category of the label that will be used in predclust. The rest are automatically categorized into the second category. For example, based on a 5-cluster clustering solution, if *cluster_names= c("P1","P2","P3","P4","P5")* and *label_category1= c("P1","P3")* each

unit or person will have the probability of *P1+P3* of belonging to the first category and the probability of *P2+P4+P5* of belonging to the second category of the label.

cluster_label_position

A string indicates the location of the cluster label in prediction.
When *cluster_label_position="predictor",* the cluster label defined in label_category1 will be used as a predictor.
When *cluster_label_position="predicted",* the cluster label will be used as an outcome predicted by provided predictors (input variables).
If *cluster_label_position="none",* the cluster label will be omitted in supervised learning.

outcome_obs

When cluster_label_position = "predictor" or cluster_label_position = "none", the user is expected to specify the outcome variable to be predicted by the cluster label and other provided predictors. This argument comes with the following subcomponents.

- outcome_type: In the current version, only a binary variable is allowed to be used as a prediction (classification) outcome. There are 2 allowed types: outcome_type="binary", when a single outcome variable is already binary (0/1). outcome_type="cutpoint", when a single binary variable will be created based on a cutpoint (or cutpoints) applied to a single or multiple variables.
- outcome_source_variables: The user may specify a single binary outcome or set of source variables that will be used to create a binary outcome. For example, *outcome_source_variables= c("a","b","c").*
- outcome_source_all_missing: An integer specifies which value to take when all variables listed in outcome_source_variables are missing. The three possible options are NA, 1, or 0. If *outcome_source_all_missing = NA*, the outcome of these individuals or units will be treated as missing. The default is 0.
- outcome_cutpoint:A numeric value/vector specifies a threshold or multiple thresholds to create a binary outcome. For example, *outcome_cutpoint=12*, or *outcome_cutpoint=c(12,13,14).*
- outcome_cutpoint_sign: A character value/vector specifies comparison operator(s) to be used with thresholds. Available options include >=, <=, >, <, ==, GE, LE, GT, LT, and EQ. When using a vector of multiple thresholds, the signs will be applied to each cutpoint.
- outcome_cutpoint_max_min_mean: A string specifies a function to use to summarize multiple variables into a single variable. The options include max, min, and mean. For example, *outcome_cutpoint_max_min_mean="max".*

When outcome_cutpoint is a single value, all cutpoint related arguments can be used together.
For example, if *outcome_source_variables=c("a","b","c"), outcome_cutpoint = 12, outcome_cutpoint_sign =">=",*
and *outcome_cutpoint_max_min_mean="max",* all cases with

max(a, b, c) >= 12

will be assigned the value of 1, and the rest the value of 0.

When outcome_cutpoint has multiple values, outcome_max_min_mean will be ignored.
For example, when *outcome_source_variables=c("a","b","c"), outcome_cutpoint = c(12,13,14), outcome_cutpoint_sign = c(">=","<",">"),* all cases with

a >= 12 and b < 13 and c > 14

will be assigned the value of 1, and the rest the value of 0.

supervised_method

A string indicates the type of supervised learning. In the current version, we allow logistic regression and glmnet. That is, *supervised_method="logistic",* or *supervised_method="glmnet".*

glmnet_specs

When glmnet is used, the user may utilize the same arguments used in glmnet such as family, lambda, alpha, etc. That is, *glmnet_specs(family="binomial",alpha=1,nlambda=100,lambda = NULL. . .)* Note that, in the current version of predclust, we only allow *family="binomial"* and one pair of lambda/alpha.
The user can also employ an external program called superclust (beta version available), which implements various supervised learning methods with cluster labels in probabilities.

seed_numbers

An integer vector includes 4 items with respect to seed numbers of splitting train/test datasets, cross-validation, pseudoclass draws as well as the supervised model. Their names are seed_num_split, seed_num_kfold, seed_num_pcd, and seed_num_supervised_model respectively. For example,

seed_numbers = c(seed_num_split = 4561234,
seed_num_kfold = 4561234,
seed_num_pcd = 4561234,
seed_num_supervised_model = 4561234)

useobs

The user may specify a text string that indicates observations to use. For example, if we want to exclude observations with *x=9 and x=13*, we can set *useobs="(x ne 9) and (x ne 13)".* If useobs has been already used under genclust and/or validclust, this argument can be used to specify additional observations to be excluded.

listwise_deletion_variables

The user can specify listwise deletion based on specific variables. For example, *listwise_deletion_variables = c("a1","b1").* This feature is useful when the user wants to conduct listwise deletion with variables that are not being used in the predclust procedure. As a default, the program uses the standard listwise deletion method for the variables included in the predclust procedure.

train_fraction

A single value between 0 and 1 indicating the fraction of the samples for the train/test split. For example, *train_fraction = 0.7* means that 70% are used as the train data and 30% are used as the test data. The program uses 0.7 as the default.

if_CV	A Boolean variable indicates whether K-fold cross validation is used in supervised learning.
K_fold	An integer indicates the number of folds in cross-validation. The default is 10. It is applicable when *if_CV = TRUE*.
repeated_CV	An integer indicates the number of repeated K-fold CV. It is applicable when *if_CV = TRUE*.
if_PCD	A Boolean variable indicates whether pseudo class draws will be used to take into account uncertainties in cluster or latent class assignment (Jo et al., 2017). This argument is relevant when soft clustering methods are used.
r_PCD	When *if_PCD = TRUE,* the user needs to specify the number of pseudo class draws. The default is 20.
lr_maxiter	An integer indicates maximum iterations in logistic regression, which is the default supervised learning method in this program. The default is 25.

Value

The supervised learning results will be provided as a csv file (predclust_results.csv) in the user- specified folder. For each supervised model, Cohen's Kappa, accuracy, sensitivity, specificity, and AUC estimates are provided (their means and standard errors if K-fold cross validation and/or pseu- doclass draws are used).

Supervised_method: The employed supervised learning method.

- Supervised_spec1 to Supervised_spec3:Further details regarding the employed supervised learning method.
- Cluster_n: The total number of clusters or classes used in creating a cluster label.
- Cluster_names: The names of all clusters used in creating a cluster label.
- Label_category1: The clusters categorized in the first category when generating a binary cluster label.
- Label_position: Whether the cluster label defined in label_category1 is used as a predictor (predictor), or as an outcome predicted by provided predictors (predicted), or the cluster label is omitted in supervised learning (none).
- Predictors: The names of the first two variables used as predictors (input variables) in supervised learning.
- Kappa, sensitivity, specificity, accuracy, AUC: These are the measures of association between the cluster label and the predicted label. When *if_CV = TRUE* and/or *if_PCD = TRUE*, the provided values are the means across K folds and R pseudoclass draws. These measures are reported separately for the training and test data.
- Kappa_SE, sensitivity_SE, specificity_SE, accuracy_SE, AUC_SE: When *if_CV = TRUE* and/or *if_PCD = TRUE*, these are the standard deviations across K folds and R pseudoclass draws. These measures are reported separately for the training and test data.

References

Jo, B., Hastie, T. J., Li, Z., Youngstrom, E. A., Findling, R. L., & Horwitz, S. M. (2023). Reorienting Latent Variable Modeling for Supervised Learning. Multivariate Behavioral Research, 1-15.