Package 'mtlgmm'

October 31, 2022

Type Package

Title Unsupervised Multi-Task and Transfer Learning on Gaussian Mixture Models

Version 0.1.0

Description Unsupervised learning has been widely used in many real-world applications. One of the simplest and most important unsupervised learning models is the Gaussian mixture model (GMM). In this work, we study the multi-task learning problem on GMMs, which aims to leverage potentially similar GMM parameter structures among tasks to obtain improved learning performance compared to single-task learning. We propose a multi-task GMM learning procedure based on the Expectation-Maximization (EM) algorithm that not only can effectively utilize unknown similarity between related tasks but is also robust against a fraction of outlier tasks from arbitrary sources. The proposed procedure is shown to achieve minimax optimal rate of convergence for both parameter estimation error and the excess mis-clustering error, in a wide range of regimes. Moreover, we generalize our approach to tackle the problem of transfer learning for GMMs, where similar theoretical results are derived. Finally, we demonstrate the effectiveness of our methods through simulations and a real data analysis. To the best of our knowledge, this is the first work studying multi-task and transfer learning on GMMs with theoretical guarantees. This package implements the algorithms proposed in Tian, Y., Weng, H., & Feng, Y. (2022) <arXiv:2209.15224>.

Imports doParallel, foreach, caret, mclust, stats

License GPL-2

Depends R (>= 3.5.0)

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Ye Tian [aut, cre],
Haolei Weng [aut],
Yang Feng [aut]

Maintainer Ye Tian <ye.t@columbia.edu>

Repository CRAN

Date/Publication 2022-10-31 14:17:37 UTC

R topics documented:

`alignment` *Align the initializations.*

Description

Align the initializations. This function implements the two alignment algorithms (Algorithms 2 and 3) in Tian, Y., Weng, H., & Feng, Y. (2022). This function is mainly for people to align the single-task initializations manually. The alignment procedure has been automatically implemented in function `mtlgmm` and `tlgmm`. So there is no need to call this function when fitting MTL-GMM or TL-GMM.

Usage

```
alignment(mu1, mu2, method = c("exhaustive", "greedy"))
```

Arguments

`mu1`	the initializations for mu1 of all tasks. Should be a matrix of which each column is a mu1 estimate of a task.
`mu2`	the initializations for mu2 of all tasks. Should be a matrix of which each column is a mu2 estimate of a task.
`method`	alignment method. Can be either "exhaustive" (Algorithm 2 in Tian, Y., Weng, H., & Feng, Y. (2022)) or "greedy" (Algorithm 3 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: "exhaustive"

Value

the index of two clusters to become well-aligned, i.e. the "r_k" in Section 2.4.2 of Tian, Y., Weng, H., & Feng, Y. (2022). The output can be passed to function `alignment_swap` to obtain the well-aligned intializations.

Note

For examples, see part "fit signle-task GMMs" of examples in function `mtlgmm`.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

See Also

`mtlgmm`, `tlgmm`, `predict_gmm`, `data_generation`, `initialize`, `alignment_swap`, `estimation_error`, `misclustering_error`.

`alignment_swap` *Complete the alignment of initializations based on the output of function* `alignment_swap`.

Description

Complete the alignment of initializations based on the output of function `alignment_swap`. This function is mainly for people to align the single-task initializations manually. The alignment procedure has been automatically implemented in function `mtlgmm` and `tlgmm`. So there is no need to call this function when fitting MTL-GMM or TL-GMM.

Usage

```
alignment_swap(L1, L2, initial_value_list)
```

Arguments

`L1`	the component "L1" of the output from function `alignment_swap`
`L2`	the component "L2" of the output from function `alignment_swap`
`initial_value_list`	the output from function `initialize`

Value

A list with the following components (well-aligned).

`w`	the estimate of mixture proportion in GMMs for each task. Will be a vector.
`mu1`	the estimate of Gaussian mean in the first cluster of GMMs for each task. Will be a matrix, where each column represents the estimate for a task.
`mu2`	the estimate of Gaussian mean in the second cluster of GMMs for each task. Will be a matrix, where each column represents the estimate for a task.
`beta`	the estimate of the discriminant coefficient for each task. Will be a matrix, where each column represents the estimate for a task.
`Sigma`	the estimate of the common covariance matrix for each task. Will be a list, where each component represents the estimate for a task.

Note

For examples, see part "fit signle-task GMMs" of examples in function `mtlgmm`.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

See Also

`mtlgmm`, `tlgmm`, `predict_gmm`, `data_generation`, `initialize`, `alignment`, `estimation_error`, `misclustering_error`.

`data_generation` *Generate data for simulations.*

Description

Generate data for simulations. All models used in Tian, Y., Weng, H., & Feng, Y. (2022)) are implemented.

Usage

```
data_generation(
  K = 10,
  outlier_K = 1,
  simulation_no = c("MTL-1", "MTL-2"),
  h_w = 0.1,
  h_mu = 1,
  n = 50
)
```

Arguments

`K`	the number of tasks (data sets). Default: 10
`outlier_K`	the number of outlier tasks. Default: 1
`simulation_no`	simulation number in Tian, Y., Weng, H., & Feng, Y. (2022)). Can be "MTL-1", "MTL-2". Default = "MTL-1".
`h_w`	the value of h_w. Default: 0.1
`h_mu`	the value of h_mu. Default: 1
`n`	the sample size of each task. Can be either an positive integer or a vector of length `K`. If it is an integer, then the sample size of all tasks will be the same and equal to `n`. If it is a vector, then the k-th number will be the sample size of the k-th task. Default: 50.

Value

a list of two sub-lists "data" and "parameter". List "data" contains a list of design matrices `x`, a list of hidden labels `y`, and a vector of outlier task indices `outlier_index`. List "parameter" contains a vector `w` of mixture proportions, a matrix `mu1` of which each column is the GMM mean of the first cluster of each task, a matrix `mu2` of which each column is the GMM mean of the second cluster of each task, a matrix `beta` of which each column is the discriminant coefficient in each task, a list `Sigma` of covariance matrices for each task.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

See Also

`mtlgmm`, `tlgmm`, `predict_gmm`, `initialize`, `alignment`, `alignment_swap`, `estimation_error`, `misclustering_error`.

Examples

```
data_list <- data_generation(K = 5, outlier_K = 1, simulation_no = "MTL-1", h_w = 0.1,
h_mu = 1, n = 50)
```

---

estimation_error — *Caluclate the estimation error of GMM parameters under the MTL setting (the worst performance among all tasks).*

Description

Caluclate the estimation error of GMM parameters under the MTL setting (the worst performance among all tasks). Euclidean norms are used.

Usage

```
estimation_error(
  estimated_value,
  true_value,
  parameter = c("w", "mu", "beta", "Sigma")
)
```

Arguments

`estimated_value`
: estimate of GMM parameters. The form of input depends on the parameter `parameter`.

`true_value`
: true values of GMM parameters. The form of input depends on the parameter `parameter`.

`parameter` which parameter to calculate the estimation error for. Can be "w", "mu", "beta", or "Sigma".

- w: the Gaussian mixture proportions. Both `estimated_value` and `true_value` require an input of a K-dimensional vector, where K is the number of tasks. Each element in the vector is an "w" (estimate or true value) for each task.
- mu: Gaussian mean parameters. Both `estimated_value` and `true_value` require an input of a list of two p-by-K matrices, where p is the dimension of Gaussian distribution and K is the number of tasks. Each column of the matrix is a "mu1" or "mu2" (estimate or true value) for each task.
- beta: discriminant coefficients. Both `estimated_value` and `true_value` require an input of a p-by-K matrix, where p is the dimension of Gaussian distribution and K is the number of tasks. Each column of the matrix is a "beta" (estimate or true value) for each task.
- Sigma: Gaussian covariance matrices. Both `estimated_value` and `true_value` require an input of a list of K p-by-p matrices, where p is the dimension of Gaussian distribution and K is the number of tasks. Each matrix in the list is a "Sigma" (estimate or true value) for each task.

Value

the largest estimation error among all tasks.

Note

For examples, see examples in function `mtlgmm`.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

See Also

`mtlgmm`, `tlgmm`, `predict_gmm`, `data_generation`, `initialize`, `alignment`, `alignment_swap`, `misclustering_error`.

---

initialize — *Initialize the estimators of GMM parameters on each task.*

Description

Initialize the estimators of GMM parameters on each task.

Usage

```
initialize(x, method = c("kmeans", "EM"))
```

Arguments

x	design matrices from multiple data sets. Should be a list, of which each component is a `matrix` or `data.frame` object, representing the design matrix from each task.
method	initialization method. This indicates the method to initialize the estimates of GMM parameters for each data set. Can be either "EM" or "kmeans". Default: "EM". • EM: the initial estimates of GMM parameters will be generated from the single-task EM algorithm. Will call `Mclust` function in `mclust` package. • kmeans: the initial estimates of GMM parameters will be generated from the single-task k-means algorithm. Will call `kmeans` function in `stats` package.

Value

A list with the following components.

w	the estimate of mixture proportion in GMMs for each task. Will be a vector.
mu1	the estimate of Gaussian mean in the first cluster of GMMs for each task. Will be a matrix, where each column represents the estimate for a task.
mu2	the estimate of Gaussian mean in the second cluster of GMMs for each task. Will be a matrix, where each column represents the estimate for a task.
beta	the estimate of the discriminant coefficient for each task. Will be a matrix, where each column represents the estimate for a task.
Sigma	the estimate of the common covariance matrix for each task. Will be a list, where each component represents the estimate for a task.

See Also

`mtlgmm`, `tlgmm`, `predict_gmm`, `data_generation`, `alignment`, `alignment_swap`, `estimation_error`, `misclustering_error`.

Examples

```
set.seed(0, kind = "L'Ecuyer-CMRG")
## Consider a 5-task multi-task learning problem in the setting "MTL-1"
data_list <- data_generation(K = 5, outlier_K = 1, simulation_no = "MTL-1", h_w = 0.1,
h_mu = 1, n = 50)  # generate the data
fit <- mtlgmm(x = data_list$data$x, C1_w = 0.05, C1_mu = 0.2, C1_beta = 0.2,
C2_w = 0.05, C2_mu = 0.2, C2_beta = 0.2, kappa = 1/3, initial_method = "EM",
trim = 0.1, lambda_choice = "fixed", step_size = "lipschitz")

## Initialize the estimators of GMM parameters on each task.
fitted_values_EM <- initialize(data_list$data$x,
"EM")  # initilize the estimates by single-task EM algorithm
fitted_values_kmeans <- initialize(data_list$data$x,
"EM")  # initilize the estimates by single-task k-means
```

misclustering_error *Calculate the misclustering error given the predicted cluster labels.*

Description

Calculate the misclustering error given the predicted cluster labels.

Usage

```
misclustering_error(y_pred, y_test, type = c("max", "all", "avg"))
```

Arguments

`y_pred`	predicted cluster labels
`y_test`	true cluster labels
`type`	which type of the misclustering error rate to return. Can be either "max", "all", or "avg". Default: "max". • max: maximum of misclustering error rates on all tasks • all: a vector of misclustering error rates on each tasks • avg: average of misclustering error rates on all tasks

Value

Depends on `type`.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

See Also

`mtlgmm`, `tlgmm`, `data_generation`, `predict_gmm`, `initialize`, `alignment`, `alignment_swap`, `estimation_error`.

Examples

```
set.seed(23, kind = "L'Ecuyer-CMRG")
## Consider a 5-task multi-task learning problem in the setting "MTL-1"
data_list <- data_generation(K = 5, outlier_K = 1, simulation_no = "MTL-1", h_w = 0.1,
h_mu = 1, n = 100)  # generate the data
x_train <- sapply(1:length(data_list$data$x), function(k){
  data_list$data$x[[k]][1:50,]
}, simplify = FALSE)
x_test <- sapply(1:length(data_list$data$x), function(k){
  data_list$data$x[[k]][-(1:50),]
}, simplify = FALSE)
```

```
y_test <- sapply(1:length(data_list$data$x), function(k){
  data_list$data$y[[k]][-(1:50)]
}, simplify = FALSE)

fit <- mtlgmm(x = x_train, C1_w = 0.05, C1_mu = 0.2, C1_beta = 0.2,
C2_w = 0.05, C2_mu = 0.2, C2_beta = 0.2, kappa = 1/3, initial_method = "EM",
trim = 0.1, lambda_choice = "fixed", step_size = "lipschitz")

y_pred <- sapply(1:length(data_list$data$x), function(i){
predict_gmm(w = fit$w[i], mu1 = fit$mu1[, i], mu2 = fit$mu2[, i],
beta = fit$beta[, i], newx = x_test[[i]])
}, simplify = FALSE)
misclustering_error(y_pred[-data_list$data$outlier_index],
y_test[-data_list$data$outlier_index], type = "max")
```

`mtlgmm` *Fit binary Gaussian mixture models (GMMs) on multiple data sets under a multi-task learning (MTL) setting.*

Description

it binary Gaussian mixture models (GMMs) on multiple data sets under a multi-task learning (MTL) setting. This function implements the modified EM algorithm (Altorithm 1) proposed in Tian, Y., Weng, H., & Feng, Y. (2022).

Usage

```
mtlgmm(
  x,
  step_size = c("lipschitz", "fixed"),
  eta_w = 0.1,
  eta_mu = 0.1,
  eta_beta = 0.1,
  lambda_choice = c("cv", "fixed"),
  cv_nfolds = 5,
  cv_upper = 5,
  cv_lower = 0.01,
  cv_length = 5,
  C1_w = 0.05,
  C1_mu = 0.2,
  C1_beta = 0.2,
  C2_w = 0.05,
  C2_mu = 0.2,
  C2_beta = 0.2,
  kappa = 1/3,
  tol = 1e-05,
  initial_method = c("EM", "kmeans"),
  alignment_method = ifelse(length(x) <= 10, "exhaustive", "greedy"),
```

```
  trim = 0.1,
  iter_max = 1000,
  iter_max_prox = 100,
  ncores = 1
)
```

Arguments

x
: design matrices from multiple data sets. Should be a list, of which each component is a `matrix` or `data.frame` object, representing the design matrix from each task.

step_size
: step size choice in proximal gradient method to solve each optimization problem in the revised EM algorithm (Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)), which can be either "lipschitz" or "fixed". Default = "lipschitz".
  - lipschitz: `eta_w`, `eta_mu` and `eta_beta` will be chosen by the Lipschitz property of the gradient of objective function (without the penalty part). See Section 4.2 of Parikh, N., & Boyd, S. (2014).
  - fixed: `eta_w`, `eta_mu` and `eta_beta` need to be specified

eta_w
: step size in the proximal gradient method to learn w (Step 3 of Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 0.1. Only used when `step_size` = "fixed".

eta_mu
: step size in the proximal gradient method to learn mu (Steps 4 and 5 of Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 0.1. Only used when `step_size` = "fixed".

eta_beta
: step size in the proximal gradient method to learn beta (Step 9 of Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 0.1. Only used when `step_size` = "fixed".

lambda_choice
: the choice of constants in the penalty parameter used in the optimization problems. See Algorithm 1 of Tian, Y., Weng, H., & Feng, Y. (2022), which can be either "fixed" or "cv". Default: "cv".
  - cv: `cv_nfolds`, `cv_upper`, and `cv_length` need to be specified. Then the C1 and C2 parameters will be chosen in all combinations in `exp(seq(log(cv_lower/10), log(cv_upper/10), length.out = cv_length))` via cross-validation. Note that this is a two-dimensional cv process, because we set `C1_w = C2_w`, `C1_mu = C1_beta = C2_mu = C2_beta` to reduce the computational cost.
  - fixed: `C1_w`, `C1_mu`, `C1_beta`, `C2_w`, `C2_mu`, and `C2_beta` need to be specified. See equations (7)-(12) in Tian, Y., Weng, H., & Feng, Y. (2022).

cv_nfolds
: the number of cross-validation folds. Default: 5

cv_upper
: the upper bound of `lambda` values used in cross-validation. Default: 5

cv_lower
: the lower bound of `lambda` values used in cross-validation. Default: 0.01

cv_length
: the number of `lambda` values considered in cross-validation. Default: 5

C1_w
: the initial value of C1_w. See equations (7) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.05

C1_mu
: the initial value of C1_mu. See equations (8) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`C1_beta` the initial value of C1_beta. See equations (9) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`C2_w` the initial value of C2_w. See equations (10) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.05

`C2_mu` the initial value of C2_mu. See equations (11) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`C2_beta` the initial value of C2_beta. See equations (12) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`kappa` the decaying rate used in equation (7)-(12) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 1/3

`tol` maximum tolerance in all optimization problems. If the difference between last update and the current update is less than this value, the iterations of optimization will stop. Default: 1e-05

`initial_method` initialization method. This indicates the method to initialize the estimates of GMM parameters for each data set. Can be either "EM" or "kmeans". Default: "EM".

- EM: the initial estimates of GMM parameters will be generated from the single-task EM algorithm. Will call `Mclust` function in `mclust` package.
- kmeans: the initial estimates of GMM parameters will be generated from the single-task k-means algorithm. Will call `kmeans` function in `stats` package.

`alignment_method`

the alignment algorithm to use. See Section 2.4 of Tian, Y., Weng, H., & Feng, Y. (2022). Can either be "exhaustive" or "greedy". Default: when `length(x)` <= 10, "exhaustive" will be used, otherwise "greedy" will be used.

- exhaustive: exhaustive search algorithm (Algorithm 2 in Tian, Y., Weng, H., & Feng, Y. (2022)) will be used.
- greedy: greey label swapping algorithm (Algorithm 3 in Tian, Y., Weng, H., & Feng, Y. (2022)) will be used.

`trim` the proportion of trimmed data sets in the cross-validation procedure of choosing tuning parameters. Setting it to a non-zero small value can help avoid the impact of outlier tasks on the choice of tuning parameters. Default: 0.1

`iter_max` the maximum iteration number of the revised EM algorithm (i.e. the parameter T in Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 1000

`iter_max_prox` the maximum iteration number of the proximal gradient method. Default: 100

`ncores` the number of cores to use. Parallel computing is strongly suggested, specially when `lambda_choice` = "cv". Default: 1

Value

A list with the following components.

`w` the estimate of mixture proportion in GMMs for each task. Will be a vector.

`mu1` the estimate of Gaussian mean in the first cluster of GMMs for each task. Will be a matrix, where each column represents the estimate for a task.

mu2	the estimate of Gaussian mean in the second cluster of GMMs for each task. Will be a matrix, where each column represents the estimate for a task.
beta	the estimate of the discriminant coefficient for each task. Will be a matrix, where each column represents the estimate for a task.
Sigma	the estimate of the common covariance matrix for each task. Will be a list, where each component represents the estimate for a task.
w_bar	the center estimate of w. Numeric. See Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022).
mu1_bar	the center estimate of mu1. Will be a vector. See Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022).
mu2_bar	the center estimate of mu2. Will be a vector. See Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022).
beta_bar	the center estimate of beta. Will be a vector. See Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022).
C1_w	the initial value of C1_w.
C1_mu	the initial value of C1_mu.
C1_beta	the initial value of C1_beta.
C2_w	the initial value of C2_w.
C2_mu	the initial value of C2_mu.
C2_beta	the initial value of C2_beta.
initial_mu1	the well-aligned initial estimate of mu1 of different tasks. Useful for the alignment problem in transfer learning. See Section 3.4 in Tian, Y., Weng, H., & Feng, Y. (2022).
initial_mu2	the well-aligned initial estimate of mu2 of different tasks. Useful for the alignment problem in transfer learning. See Section 3.4 in Tian, Y., Weng, H., & Feng, Y. (2022).

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

Parikh, N., & Boyd, S. (2014). Proximal algorithms. Foundations and trends in Optimization, 1(3), 127-239.

See Also

tlgmm, predict_gmm, data_generation, initialize, alignment, alignment_swap, estimation_error, misclustering_error.

Examples

```
set.seed(0, kind = "L'Ecuyer-CMRG")
library(mclust)
## Consider a 5-task multi-task learning problem in the setting "MTL-1"
data_list <- data_generation(K = 5, outlier_K = 1, simulation_no = "MTL-1",
```

```
h_w = 0.1, h_mu = 1, n = 50)  # generate the data
fit <- mtlgmm(x = data_list$data$x, C1_w = 0.05, C1_mu = 0.2, C1_beta = 0.2,
C2_w = 0.05, C2_mu = 0.2, C2_beta = 0.2, kappa = 1/3, initial_method = "EM",
trim = 0.1, lambda_choice = "fixed", step_size = "lipschitz")


## compare the performance with that of single-task estimators
# fit single-task GMMs
fitted_values <- initialize(data_list$data$x, "EM")  # initilize the estimates
L <- alignment(fitted_values$mu1, fitted_values$mu2,
method = "exhaustive")  # call the alignment algorithm
fitted_values <- alignment_swap(L$L1, L$L2,
initial_value_list = fitted_values)  # obtain the well-aligned initial estimates

# fit a pooled GMM
x.comb <- Reduce("rbind", data_list$data$x)
fit_pooled <- Mclust(x.comb, G = 2, modelNames = "EEE")
fitted_values_pooled <- list(w = NULL, mu1 = NULL, mu2 = NULL, beta = NULL, Sigma = NULL)
fitted_values_pooled$w <- rep(fit_pooled$parameters$pro[1], length(data_list$data$x))
fitted_values_pooled$mu1 <- matrix(rep(fit_pooled$parameters$mean[,1],
length(data_list$data$x)), ncol = length(data_list$data$x))
fitted_values_pooled$mu2 <- matrix(rep(fit_pooled$parameters$mean[,2],
length(data_list$data$x)), ncol = length(data_list$data$x))
fitted_values_pooled$Sigma <- sapply(1:length(data_list$data$x), function(k){
  fit_pooled$parameters$variance$Sigma
}, simplify = FALSE)
fitted_values_pooled$beta <- sapply(1:length(data_list$data$x), function(k){
  solve(fit_pooled$parameters$variance$Sigma) %*%
  (fit_pooled$parameters$mean[,1] - fit_pooled$parameters$mean[,2])
})
error <- matrix(nrow = 3, ncol = 4, dimnames = list(c("Single-task-GMM","Pooled-GMM","MTL-GMM"),
c("w", "mu", "beta", "Sigma")))
error["Single-task-GMM", "w"] <- estimation_error(
fitted_values$w[-data_list$data$outlier_index],
data_list$parameter$w[-data_list$data$outlier_index], "w")
error["Pooled-GMM", "w"] <- estimation_error(
fitted_values_pooled$w[-data_list$data$outlier_index],
data_list$parameter$w[-data_list$data$outlier_index], "w")
error["MTL-GMM", "w"] <- estimation_error(
fit$w[-data_list$data$outlier_index],
data_list$parameter$w[-data_list$data$outlier_index], "w")

error["Single-task-GMM", "mu"] <- estimation_error(
list(fitted_values$mu1[, -data_list$data$outlier_index],
fitted_values$mu2[, -data_list$data$outlier_index]),
list(data_list$parameter$mu1[, -data_list$data$outlier_index],
data_list$parameter$mu2[, -data_list$data$outlier_index]), "mu")
error["Pooled-GMM", "mu"] <- estimation_error(list(
fitted_values_pooled$mu1[, -data_list$data$outlier_index],
fitted_values_pooled$mu2[, -data_list$data$outlier_index]),
list(data_list$parameter$mu1[, -data_list$data$outlier_index],
data_list$parameter$mu2[, -data_list$data$outlier_index]), "mu")
error["MTL-GMM", "mu"] <- estimation_error(list(
```

```
fit$mu1[, -data_list$data$outlier_index],
fit$mu2[, -data_list$data$outlier_index]),
list(data_list$parameter$mu1[, -data_list$data$outlier_index],
data_list$parameter$mu2[, -data_list$data$outlier_index]), "mu")

error["Single-task-GMM", "beta"]  <- estimation_error(
fitted_values$beta[, -data_list$data$outlier_index],
data_list$parameter$beta[, -data_list$data$outlier_index], "beta")
error["Pooled-GMM", "beta"] <- estimation_error(
fitted_values_pooled$beta[, -data_list$data$outlier_index],
data_list$parameter$beta[, -data_list$data$outlier_index], "beta")
error["MTL-GMM", "beta"] <- estimation_error(
fit$beta[, -data_list$data$outlier_index],
data_list$parameter$beta[, -data_list$data$outlier_index], "beta")

error["Single-task-GMM", "Sigma"] <- estimation_error(
fitted_values$Sigma[-data_list$data$outlier_index],
data_list$parameter$Sigma[-data_list$data$outlier_index], "Sigma")
error["Pooled-GMM", "Sigma"] <- estimation_error(
fitted_values_pooled$Sigma[-data_list$data$outlier_index],
data_list$parameter$Sigma[-data_list$data$outlier_index], "Sigma")
error["MTL-GMM", "Sigma"] <- estimation_error(
fit$Sigma[-data_list$data$outlier_index],
data_list$parameter$Sigma[-data_list$data$outlier_index], "Sigma")

error


# use cross-validation to choose the tuning parameters
# warning: can be quite slow, large "ncores" input is suggested!!
fit <- mtlgmm(x = data_list$data$x, kappa = 1/3, initial_method = "EM", ncores = 2, cv_length = 5,
trim = 0.1, cv_upper = 2, cv_lower = 0.01, lambda = "cv", step_size = "lipschitz")
```

`predict_gmm` *Clustering new observations based on fitted GMM estimators.*

Description

Clustering new observations based on fitted GMM estimators, which is an empirical version of Bayes classifier. See equation (13) in Tian, Y., Weng, H., & Feng, Y. (2022).

Usage

```
predict_gmm(w, mu1, mu2, beta, newx)
```

Arguments

`w`	the estimate of mixture proportion in the GMM. Numeric.
`mu1`	the estimate of Gaussian mean of the first cluster in the GMM. Should be a vector.
`mu2`	the estimate of Gaussian mean of the first cluster in the GMM. Should be a vector.
`beta`	the estimate of the discriminant coefficient for the GMM. Should be a vector.
`newx`	design matrix of new observations. Should be a matrix.

Value

A vector of predicted labels of new observations.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

See Also

`mtlgmm`, `tlgmm`, `data_generation`, `initialize`, `alignment`, `alignment_swap`, `estimation_error`, `misclustering_error`.

Examples

```
set.seed(23, kind = "L'Ecuyer-CMRG")
## Consider a 5-task multi-task learning problem in the setting "MTL-1"
data_list <- data_generation(K = 5, outlier_K = 1, simulation_no = "MTL-1", h_w = 0.1,
h_mu = 1, n = 50)  # generate the data
x_train <- sapply(1:length(data_list$data$x), function(k){
  data_list$data$x[[k]][1:50,]
}, simplify = FALSE)
x_test <- sapply(1:length(data_list$data$x), function(k){
  data_list$data$x[[k]][-(1:50),]
}, simplify = FALSE)
y_test <- sapply(1:length(data_list$data$x), function(k){
  data_list$data$y[[k]][-(1:50)]
}, simplify = FALSE)

fit <- mtlgmm(x = x_train, C1_w = 0.05, C1_mu = 0.2, C1_beta = 0.2,
C2_w = 0.05, C2_mu = 0.2, C2_beta = 0.2, kappa = 1/3, initial_method = "EM",
trim = 0.1, lambda_choice = "fixed", step_size = "lipschitz")

y_pred <- sapply(1:length(data_list$data$x), function(i){
predict_gmm(w = fit$w[i], mu1 = fit$mu1[, i], mu2 = fit$mu2[, i],
beta = fit$beta[, i], newx = x_test[[i]])
}, simplify = FALSE)
misclustering_error(y_pred[-data_list$data$outlier_index],
y_test[-data_list$data$outlier_index], type = "max")
```

`tlgmm` *Fit the binary Gaussian mixture model (GMM) on target data set by leveraging multiple source data sets under a transfer learning (TL) setting.*

Description

Fit the binary Gaussian mixture model (GMM) on target data set by leveraging multiple source data sets under a transfer learning (TL) setting. This function implements the modified EM algorithm (Altorithm 4) proposed in Tian, Y., Weng, H., & Feng, Y. (2022).

Usage

```
tlgmm(
  x,
  fitted_bar,
  step_size = c("lipschitz", "fixed"),
  eta_w = 0.1,
  eta_mu = 0.1,
  eta_beta = 0.1,
  lambda_choice = c("fixed", "cv"),
  cv_nfolds = 5,
  cv_upper = 2,
  cv_lower = 0.01,
  cv_length = 5,
  C1_w = 0.05,
  C1_mu = 0.2,
  C1_beta = 0.2,
  C2_w = 0.05,
  C2_mu = 0.2,
  C2_beta = 0.2,
  kappa0 = 1/3,
  tol = 1e-05,
  initial_method = c("kmeans", "EM"),
  iter_max = 1000,
  iter_max_prox = 100,
  ncores = 1
)
```

Arguments

`x`	design matrix of the target data set. Should be a `matrix` or `data.frame` object.
`fitted_bar`	the output from `mtlgmm` function.
`step_size`	step size choice in proximal gradient method to solve each optimization problem in the revised EM algorithm (Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)), which can be either "lipschitz" or "fixed". Default = "lipschitz".

- lipschitz: `eta_w`, `eta_mu` and `eta_beta` will be chosen by the Lipschitz property of the gradient of objective function (without the penalty part). See Section 4.2 of Parikh, N., & Boyd, S. (2014).
- fixed: `eta_w`, `eta_mu` and `eta_beta` need to be specified

`eta_w`
: step size in the proximal gradient method to learn w (Step 3 of Algorithm 4 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 0.1. Only used when `step_size` = "fixed".

`eta_mu`
: step size in the proximal gradient method to learn mu (Steps 4 and 5 of Algorithm 4 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 0.1. Only used when `step_size` = "fixed".

`eta_beta`
: step size in the proximal gradient method to learn beta (Step 7 of Algorithm 4 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 0.1. Only used when `step_size` = "fixed".

`lambda_choice`
: the choice of constants in the penalty parameter used in the optimization problems. See Algorithm 4 of Tian, Y., Weng, H., & Feng, Y. (2022), which can be either "fixed" or "cv". Default = "cv".

- cv: `cv_nfolds`, `cv_upper`, and `cv_length` need to be specified. Then the C1 and C2 parameters will be chosen in all combinations in `exp(seq(log(cv_lower/10), log(cv_upper/10), length.out = cv_length))` via cross-validation. Note that this is a two-dimensional cv process, because we set `C1_w = C2_w`, `C1_mu = C1_beta = C2_mu = C2_beta` to reduce the computational cost.
- fixed: `C1_w`, `C1_mu`, `C1_beta`, `C2_w`, `C2_mu`, and `C2_beta` need to be specified. See equations (19)-(24) in Tian, Y., Weng, H., & Feng, Y. (2022).

`cv_nfolds`
: the number of cross-validation folds. Default: 5

`cv_upper`
: the upper bound of `lambda` values used in cross-validation. Default: 5

`cv_lower`
: the lower bound of `lambda` values used in cross-validation. Default: 0.01

`cv_length`
: the number of `lambda` values considered in cross-validation. Default: 5

`C1_w`
: the initial value of C1_w. See equations (19) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.05

`C1_mu`
: the initial value of C1_mu. See equations (20) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`C1_beta`
: the initial value of C1_beta. See equations (21) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`C2_w`
: the initial value of C2_w. See equations (22) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.05

`C2_mu`
: the initial value of C2_mu. See equations (23) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`C2_beta`
: the initial value of C2_beta. See equations (24) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 0.2

`kappa0`
: the decaying rate used in equation (19)-(24) in Tian, Y., Weng, H., & Feng, Y. (2022). Default: 1/3

`tol`
: maximum tolerance in all optimization problems. If the difference between last update and the current update is less than this value, the iterations of optimization will stop. Default: 1e-05

initial_method	initialization method. This indicates the method to initialize the estimates of GMM parameters for each data set. Can be either "kmeans" or "EM". • kmeans: the initial estimates of GMM parameters will be generated from the single-task k-means algorithm. Will call kmeans function in stats package. • EM: the initial estimates of GMM parameters will be generated from the single-task EM algorithm. Will call Mclust function in mclust package.
iter_max	the maximum iteration number of the revised EM algorithm (i.e. the parameter T in Algorithm 1 in Tian, Y., Weng, H., & Feng, Y. (2022)). Default: 1000
iter_max_prox	the maximum iteration number of the proximal gradient method. Default: 100
ncores	the number of cores to use. Parallel computing is strongly suggested, specially when lambda_choice = "cv". Default: 1

Value

A list with the following components.

w	the estimate of mixture proportion in GMMs for the target task. Will be a vector.
mu1	the estimate of Gaussian mean in the first cluster of GMMs for the target task. Will be a matrix, where each column represents the estimate for a task.
mu2	the estimate of Gaussian mean in the second cluster of GMMs for the target task. Will be a matrix, where each column represents the estimate for a task.
beta	the estimate of the discriminant coefficient for the target task. Will be a matrix, where each column represents the estimate for a task.
Sigma	the estimate of the common covariance matrix for the target task. Will be a list, where each component represents the estimate for a task.
C1_w	the initial value of C1_w.
C1_mu	the initial value of C1_mu.
C1_beta	the initial value of C1_beta.
C2_w	the initial value of C2_w.
C2_mu	the initial value of C2_mu.
C2_beta	the initial value of C2_beta.

References

Tian, Y., Weng, H., & Feng, Y. (2022). Unsupervised Multi-task and Transfer Learning on Gaussian Mixture Models. arXiv preprint arXiv:2209.15224.

Parikh, N., & Boyd, S. (2014). Proximal algorithms. Foundations and trends in Optimization, 1(3), 127-239.

See Also

mtlgmm, predict_gmm, data_generation, initialize, alignment, alignment_swap, estimation_error, misclustering_error.

Examples

```
set.seed(0, kind = "L'Ecuyer-CMRG")
## Consider a transfer learning problem with 3 source tasks and 1 target task in the setting "MTL-1"
data_list_source <- data_generation(K = 3, outlier_K = 0, simulation_no = "MTL-1", h_w = 0,
h_mu = 0, n = 50)  # generate the source data
data_target <- data_generation(K = 1, outlier_K = 0, simulation_no = "MTL-1", h_w = 0.1,
h_mu = 1, n = 50)  # generate the target data
fit_mtl <- mtlgmm(x = data_list_source$data$x, C1_w = 0.05, C1_mu = 0.2, C1_beta = 0.2,
C2_w = 0.05, C2_mu = 0.2, C2_beta = 0.2, kappa = 1/3, initial_method = "EM",
trim = 0.1, lambda_choice = "fixed", step_size = "lipschitz")

fit_tl <- tlgmm(x = data_target$data$x[[1]], fitted_bar = fit_mtl, C1_w = 0.05,
C1_mu = 0.2, C1_beta = 0.2, C2_w = 0.05, C2_mu = 0.2, C2_beta = 0.2, kappa0 = 1/3,
initial_method = "EM", ncores = 1, lambda_choice = "fixed", step_size = "lipschitz")


# use cross-validation to choose the tuning parameters
# warning: can be quite slow, large "ncores" input is suggested!!
fit_tl <- tlgmm(x = data_target$data$x[[1]], fitted_bar = fit_mtl, kappa0 = 1/3,
initial_method = "EM", ncores = 2, lambda_choice = "cv", step_size = "lipschitz")
```

Index