Package 'DMTL'

October 12, 2022

Type Package

Title Tools for Applying Distribution Mapping Based Transfer Learning

Version 0.1.2

Description Implementation of a transfer learning framework employing distribution mapping based domain transfer. Uses the renowned concept of histogram matching (see Gonzalez and Fittes (1977) <doi:10.1016/0094-114X(77)90062-3>, Gonzalez and Woods (2008) <isbn:9780131687288>) and extends it to include distribution measures like kernel density estimates (KDE; see Wand and Jones (1995) <isbn:978-0-412-55270-0>, Jones et al. (1996) <doi:10.2307/2291420). In the typical application scenario, one can use the underlying sample distributions (histogram or KDE) to generate a map between two distinct but related domains to transfer the target data to the source domain and utilize the available source data for better predictive modeling design. Suitable for the case where a one-to-one sample matching is not possible, thus one needs to transform the underlying data distribution to utilize the more available data for modeling.

Encoding UTF-8

Depends R (>= 3.6)

Imports caret (>= 6.0-86), glmnet (>= 4.1), kernlab (>= 0.9-29), ks (>= 1.11.7), randomForest (>= 4.6-14)

License GPL-3

URL https://github.com/dhruba018/DMTL

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Author Saugato Rahman Dhruba [aut, cre] (<https://orcid.org/0000-0001-5947-6757>), Souparno Ghosh [ctb], Ranadip Pal [aut]

Maintainer Saugato Rahman Dhruba <dhruba018@gmail.com>

Repository CRAN

Date/Publication 2021-02-18 10:50:02 UTC

R topics documented:

`confined` *Restrict data in a given interval*

Description

This function filters a data vector using a given interval so that only the values falling inside the interval remains and any value that is less than the leftmost end gets replaced by that end-value, and similarly, any value greater than the rightmost end gets replaced by that end-value.

Usage

```
confined(x, lims = c(0, 1))
```

Arguments

`x`	Vector containing data.
`lims`	Limit for the values. Values falling within this limit will pass without any change. Any value `x < lims[1]` will get replaced by `lims[1]`, and any value `x > lims[2]` will get replaced by `lims[2]`. Defaults to `c(0, 1)`.

Value

The filtered vector.

Examples

```
x <- rnorm(100, 0, 1)
x_filt <- confined(x, lims = c(-0.5, 0.5))
print(range(x_filt))
```

dist_match *Distribution Matching for Source and Reference Datasets*

Description

This function matches a source distribution to a given reference distribution such that the data in the source space can effectively be transferred to the reference space *i.e.* domain transfer *via* distribution matching.

Usage

```
dist_match(
  src,
  ref,
  src_cdf,
  ref_cdf,
  lims,
  density = FALSE,
  samples = 1e+06,
  seed = NULL
)
```

Arguments

src	Vector containing the source data to be matched.
ref	Vector containing the reference data to estimate the reference distribution for matching.
src_cdf	Vector containing source distribution values. If missing, these values are estimated from the source data using `estimate_cdf()`.
ref_cdf	Vector containing reference distribution values. If missing, these values are estimated from the reference data using `estimate_cdf()`.
lims	Vector providing the range of the knot values for mapping. If missing, these values are estimated from the reference data.
density	Flag for using kernel density estimates for matching instead of histogram counts. Defaults to `False`.
samples	Sample size for estimating distributions if `src_cdf` and/or `ref_cdf` are missing. Defaults to `1e6`.
seed	Seed for random number generator (for reproducible outcomes). Defaults to `NULL`.

Value

A vector containing the matched values corresponding to `src`.

Examples

```
set.seed(7531)
x1 <- rnorm(100, 0.2, 0.6)
x2 <- runif(200)
matched <- dist_match(src = x1, ref = x2, lims = c(0, 1))

## Plot histograms...
opar <- par(mfrow = c(1, 3))
hist(x1);    hist(x2);    hist(matched)
par(opar)              # Reset par
```

DMTL *Distribution Mapping based Transfer Learning*

Description

This function performs distribution mapping based transfer learning (DMTL) regression for given target (primary) and source (secondary) datasets. The data available in the source domain are used to design an appropriate predictive model. The target features with unknown response values are transferred to the source domain *via* distribution matching and then the corresponding response values in the source domain are predicted using the aforementioned predictive model. The response values are then transferred to the original target space by applying distribution matching again. Hence, this function needs an **unmatched** pair of target datasets (features and response values) and a **matched** pair of source datasets.

Usage

```
DMTL(
  target_set,
  source_set,
  use_density = FALSE,
  pred_model = "RF",
  model_optimize = FALSE,
  sample_size = 1000,
  random_seed = NULL,
  all_pred = FALSE,
  get_verbose = FALSE,
  allow_parallel = FALSE
)
```

Arguments

`target_set`	List containing the target datasets. A named list with components `X` (predictors) and `y` (response). The predictions are performed to estimate the response values corresponding to `X` while `y` is only used to estimate the response distribution parameters.

source_set	List containing the source datasets. A named list with components X (predictors) and y (response). These two sets must be matched and used in both distribution estimation and predictive modeling.
use_density	Flag for using kernel density as distribution estimate instead of histogram counts. Defaults to FALSE.
pred_model	String indicating the underlying predictive model. The currently available options are - • RF for random forest regression. If model_optimize = FALSE, builds a model with n_tree = 200 and m_try = 0.4. • SVM for support vector regression. If model_optimize = FALSE, builds a model with kernel = "poly", C = 2, and degree = 3. • EN for elastic net regression. If model_optimize = FALSE, builds a model with alpha = 0.8 and lambda generated from a 5-fold cross validation.
model_optimize	Flag for model parameter tuning. If TRUE, performs a grid search to optimize parameters and train with the resulting model. If FALSE, uses a set of predefined parameters. Defaults to FALSE.
sample_size	Sample size for estimating distributions of target and source datasets. Defaults to 1e3.
random_seed	Seed for random number generator (for reproducible outcomes). Defaults to NULL.
all_pred	Flag for returning the prediction values in the source space. If TRUE, the function returns a named list with two components- target and source (predictions in the target space and source space, respectively). Defaults to FALSE.
get_verbose	Flag for displaying the progress when optimizing the predictive model *i.e.*, model_optimize = TRUE. Defaults to FALSE.
allow_parallel	Flag for allowing parallel processing when performing grid search *i.e.*, model_optimimze = TRUE. Defaults to FALSE.

Value

If all_pred = FALSE, a vector containing the final prediction values.

If all_pred = TRUE, a named list with two components target and source *i.e.*, predictions in the original target space and in source space, respectively.

Note

- The datasets in target_set (*i.e.*, X and y) do not need to be matched (*i.e.*, have the same number of rows) since the response values are used only to estimate distribution for mapping while the feature values are used for both mapping and final prediction. In contrast, the datasets in source_set (*i.e.*, X and y) must have matched samples.
- It is recommended to normalize the two response values (y) so that they will be in the same range. If normalization is not performed, DMTL() uses the range of target y values as the prediction range.

Examples

```
set.seed(8644)

## Generate two dataset with different underlying distributions...
x1 <- matrix(rnorm(3000, 0.3, 0.6), ncol = 3)
dimnames(x1) <- list(paste0("sample", 1:1000), paste0("f", 1:3))
y1 <- 0.3*x1[, 1] + 0.1*x1[, 2] - x1[, 3] + rnorm(1000, 0, 0.05)
x2 <- matrix(rnorm(3000, 0, 0.5), ncol = 3)
dimnames(x2) <- list(paste0("sample", 1:1000), paste0("f", 1:3))
y2 <- -0.2*x2[, 1] + 0.3*x2[, 2] - x2[, 3] + rnorm(1000, 0, 0.05)

## Model datasets using DMTL & compare with a baseline model...
library(DMTL)

target <- list(X = x1, y = y1)
source <- list(X = x2, y = y2)
y1_pred <- DMTL(target_set = target, source_set = source, pred_model = "RF")
y1_pred_bl <- RF_predict(x_train = x2, y_train = y2, x_test = x1)

print(performance(y1, y1_pred, measures = c("MSE", "PCC")))
print(performance(y1, y1_pred_bl, measures = c("MSE", "PCC")))
```

EN_predict *Predictive Modeling using Elastic Net*

Description

This function trains a Elastic Net regressor using the training data provided and predict response for the test features. This implementation depends on the `glmnet` package.

Usage

```
EN_predict(
  x_train,
  y_train,
  x_test,
  lims,
  optimize = FALSE,
  alpha = 0.8,
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE
)
```

Arguments

`x_train`	Training features for designing the EN regressor.

y_train	Training response for designing the EN regressor.
x_test	Test features for which response values are to be predicted. If x_test is not given, the function will return the trained model.
lims	Vector providing the range of the response values for modeling. If missing, these values are estimated from the training response.
optimize	Flag for model tuning. If TRUE, performs a grid search for parameters. If FALSE, uses the parameters provided. Defaults to FALSE.
alpha	EN mixing parameter with $0 \leq \alpha \leq 1$. alpha = 1 is the lasso penalty, and alpha = 0 the ridge penalty. Defaults to 0.8. Valid only when optimize = FALSE.
seed	Seed for random number generator (for reproducible outcomes). Defaults to NULL.
verbose	Flag for printing the tuning progress when optimize = TRUE. Defaults to FALSE.
parallel	Flag for allowing parallel processing when performing grid search *i.e.*, optimimze = TRUE. Defaults to FALSE.

Value

If x_test is missing, the trained EN regressor.

If x_test is provided, the predicted values using the model.

Note

The response values are filtered to be bound by range in lims.

Examples

```
set.seed(86420)
x <- matrix(rnorm(3000, 0.2, 1.2), ncol = 3);    colnames(x) <- paste0("x", 1:3)
y <- 0.3*x[, 1] + 0.1*x[, 2] - x[, 3] + rnorm(1000, 0, 0.05)

## Get the model only...
model <- EN_predict(x_train = x[1:800, ], y_train = y[1:800], alpha = 0.6)

## Get predictive performance...
y_pred <- EN_predict(x_train = x[1:800, ], y_train = y[1:800], x_test = x[801:1000, ])
y_test <- y[801:1000]
print(performance(y_test, y_pred, measures = "RSQ"))
```

---

estimate_cdf *Estimate Cumulative Distribution*

Description

This function estimates the values of the cumulative distribution function (CDF) for a vector.

Usage

```
estimate_cdf(
  x,
  bootstrap = TRUE,
  samples = 1e+06,
  density = FALSE,
  binned = TRUE,
  grids = 10000,
  unit_range = FALSE,
  seed = NULL,
  ...
)
```

Arguments

x	Vector containing data.
bootstrap	Flag for performing bootstrapping on x to get a better estimate of the CDF. Defaults to TRUE.
samples	Sample size for bootstrapping. Defaults to 1e6. Ignored when bootstrap = FALSE.
density	Flag for calculating kernel density estimates (KDE) instead of histogram counts. Depends on the ks package for density estimation. Defaults to FALSE.
binned	Flag for calculating binned KDE. Defaults to TRUE. Ignored when density = FALSE.
grids	Size parameter for the estimation grid when density = TRUE. Used to calculate the grid sizes for KDE bandwidth estimation (grids*10), and grid size KDE estimation (bgridsize = grids if binned = TRUE else gridsize = grids/10). Defaults to 1e4.
unit_range	Flag for unity data range (*i.e.*, data is normalized between 0 and 1). Defaults to FALSE.
seed	Seed for random number generator (for reproducible outcomes). Defaults to NULL.
...	Other options relevant for distribution estimation.

Value

If density = FALSE, a function of class ecdf, inheriting from the stepfun class, and hence inheriting a knots() method.

If density = TRUE, an object of class kcde which has the fields eval.points and estimate necessary for calculating a map.

Examples

```
x <- runif(100)
x_hist_cdf <- estimate_cdf(x, samples = 1000, unit_range = TRUE)
x_kde_cdf <- estimate_cdf(x, density = TRUE, unit_range = TRUE)
```

match_func *Estimate Inverse Mapping*

Description

This function estimates an inverse map g for a given set of knots (input) and values (output) corresponding to a certain map f *i.e.*, given $x, y | f : x --> y$, `match_func()` estimates $g : y --> x$ using linear interpolation.

Usage

```
match_func(knots, vals, new_vals, lims, get_func = FALSE)
```

Arguments

`knots`	Vector containing knots for the distribution estimate.
`vals`	Vector containing distribution values corresponding to the knots.
`new_vals`	Vector containing distribution values for which the knots are unknown. If missing, `match_func()` simply returns the map function.
`lims`	Vector providing the range of the knot values for mapping. If missing, these values are estimated from the given knots.
`get_func`	Flag for returning the map function if `new_vals` is provided. If `TRUE`, `match_func()` returns a named list with two components- `mapped` and `func` (mapped knots for `new_vals` and the mapping function, respectively). Defaults to `FALSE`.

Value

If `new_vals` is missing, a function performing interpolation (linear or constant) of the given data points.

If `get_func = FALSE`, a vector containing the matched knots that will produce `new_vals` for the map f.

If `get_func = TRUE`, a named list with two components- `mapped` and `func` (mapped knots for `new_vals` and the mapping function, respectively).

Examples

```
set.seed(654321)
x <- rnorm(100, 1, 0.5)
F <- ecdf(x)
fval <- F(x)
map <- match_func(knots = x, vals = fval)

x2 <- rnorm(20, 0.8, 0.5)
F2 <- ecdf(x2)
fval2 <- F2(x2)
matched <- match_func(knots = x, vals = fval, new_vals = fval2)
```

```
## Plot histograms...
opar <- par(mfrow = c(1, 3))
hist(x);    hist(x2);    hist(matched)
par(opar)              # Reset par
```

`norm01` *Normalize vector in [0, 1]*

Description

This function normalizes a given vector between 0 and 1.

Usage

```
norm01(x)
```

Arguments

`x`	Vector containing data.

Value

The normalized vector.

Examples

```
x <- rnorm(100, 0.2, 0.3)
x_norm <- norm01(x)
print(range(x_norm))
```

`norm_data` *Normalize matrix per column in [0, 1]*

Description

This function normalizes each column of a dataframe or matrix (-alike) between 0 and 1.

Usage

```
norm_data(X)
```

Arguments

`X`	Dataframe or matrix (-alike) containing data.

Value

The normalized dataframe.

Examples

```
X <- matrix(rnorm(1000, 0.2, 0.3), nrow = 100)
X_norm <- norm_data(X)
print(range(X_norm))
```

---

performance *Evaluate Regression Model Performance using Various Metrics*

Description

This function produces the predictive performance for a regression model using various common performance metrics such as MSE, R-squared, or Correlation coefficients.

Usage

```
performance(y_obs, y_pred, measures = c("NRMSE", "NMAE", "PCC"))
```

Arguments

y_obs	Observed response values
y_pred	Predicted response values
measures	Performance measures. One can specify a single measure or a vector containing multiple measures in terms of common error or similarity metrics. The available options are roughly divided into 3 categories - • "MSE", "RMSE", "NRMSE" for mean squared error, root mean squared error, and normalized root mean squared error, respectively. • "MAE", "NMAE" for mean absolute error, and normalized mean absolute error, respectively. • "PCC", "SCC", "RSQ" for Pearson's correlation, Spearman's correlation, and R-squared, respectively. Defaults to `c("NRMSE", "NMAE", "PCC")`.

Value

A vector containing the performance metric values.

Examples

```
set.seed(654321)
x <- rnorm(1000, 0.2, 0.5)
y <- x^2 + rnorm(1000, 0, 0.1)
y_fit <- predict(lm(y ~ x))
print(performance(y, y_fit, measures = c("MSE", "RSQ")))
```

RF_predict *Predictive Modeling using Random Forest Regression*

Description

This function trains a Random Forest regressor using the training data provided and predict response for the test features. This implementation depends on the `randomForest` package.

Usage

```
RF_predict(
  x_train,
  y_train,
  x_test,
  lims,
  optimize = FALSE,
  n_tree = 300,
  m_try = 0.3333,
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE
)
```

Arguments

`x_train`	Training features for designing the RF regressor.
`y_train`	Training response for designing the RF regressor.
`x_test`	Test features for which response values are to be predicted. If `x_test` is not given, the function will return the trained model.
`lims`	Vector providing the range of the response values for modeling. If missing, these values are estimated from the training response.
`optimize`	Flag for model tuning. If `TRUE`, performs a grid search for parameters. If `FALSE`, uses the parameters provided. Defaults to `FALSE`.
`n_tree`	Number of decision trees to be built in the forest. Defaults to `300`. Valid only when `optimize = FALSE`.
`m_try`	Fraction of the features to be used for building each tree. Defaults to `0.3333` (or 33.33%). Valid only when `optimize = FALSE`.

seed	Seed for random number generator (for reproducible outcomes). Defaults to NULL.
verbose	Flag for printing the tuning progress when optimize = TRUE. Defaults to FALSE.
parallel	Flag for allowing parallel processing when performing grid search *i.e.*, optimimze = TRUE. Defaults to FALSE.

Value

If x_test is missing, the trained RF regressor.

If x_test is provided, the predicted values using the model.

Note

The response values are filtered to be bound by range in lims.

Examples

```
set.seed(86420)
x <- matrix(rnorm(3000, 0.2, 1.2), ncol = 3);    colnames(x) <- paste0("x", 1:3)
y <- 0.3*x[, 1] + 0.1*x[, 2] - x[, 3] + rnorm(1000, 0, 0.05)

## Get the model only...
model <- RF_predict(x_train = x[1:800, ], y_train = y[1:800], n_tree = 300)

## Get predictive performance...
y_pred <- RF_predict(x_train = x[1:800, ], y_train = y[1:800], x_test = x[801:1000, ])
y_test <- y[801:1000]
print(performance(y_test, y_pred, measures = "RSQ"))
```

SVM_predict *Predictive Modeling using Support Vector Machine*

Description

This function trains a Support Vector Machine regressor using the training data provided and predict response for the test features. This implementation depends on the kernlab package.

Usage

```
SVM_predict(
  x_train,
  y_train,
  x_test,
  lims,
  kernel = "rbf",
  optimize = FALSE,
  C = 2,
```

```
  kpar = list(sigma = 0.1),
  eps = 0.01,
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE
)
```

Arguments

x_train	Training features for designing the SVM regressor.
y_train	Training response for designing the SVM regressor.
x_test	Test features for which response values are to be predicted. If x_test is not given, the function will return the trained model.
lims	Vector providing the range of the response values for modeling. If missing, these values are estimated from the training response.
kernel	Kernel function for SVM implementation. The available options are linear, poly, rbf, and tanh. Defaults to rbf.
optimize	Flag for model tuning. If TRUE, performs a grid search for parameters. If FALSE, uses the parameters provided. Defaults to FALSE.
C	Cost of constraints violation. This is the constant "C" of the regularization term in the Lagrange formulation. Defaults to 2. Valid only when optimize = FALSE.
kpar	List of kernel parameters. This is a named list that contains the parameters to be used with the specified kernel. The valid parameters for the existing kernels are - • sigma for the radial basis (rbf) kernel. Note that this is the **inverse** kernel width. • degree, scale, offset for the polynomial kernel. • scale, offset for the hyperbolic tangent kernel. Valid only when optimize = FALSE. Defaults to list(sigma = 0.1).
eps	The insensitive-loss function used for epsilon-SVR. Defaults to 0.01.
seed	Seed for random number generator (for reproducible outcomes). Defaults to NULL.
verbose	Flag for printing the tuning progress when optimize = TRUE. Defaults to FALSE.
parallel	Flag for allowing parallel processing when performing grid search *i.e.*, optimimze = TRUE. Defaults to FALSE.

Value

If x_test is missing, the trained SVM regressor.

If x_test is provided, the predicted values using the model.

Note

The response values are filtered to be bound by range in lims.

Examples

```
set.seed(86420)
x <- matrix(rnorm(3000, 0.2, 1.2), ncol = 3);    colnames(x) <- paste0("x", 1:3)
y <- 0.3*x[, 1] + 0.1*x[, 2] - x[, 3] + rnorm(1000, 0, 0.05)

## Get the model only...
model <- SVM_predict(x_train = x[1:800, ], y_train = y[1:800], kernel = "rbf")

## Get predictive performance...
y_pred <- SVM_predict(x_train = x[1:800, ], y_train = y[1:800], x_test = x[801:1000, ])
y_test <- y[801:1000]
print(performance(y_test, y_pred, measures = "RSQ"))
```

---

zscore *Standardize matrix per column*

Description

This function standardized each column of a dataframe or matrix (-alike) to have $mean = 0$ and $sd = 1$.

Usage

```
zscore(X)
```

Arguments

X	Dataframe or matrix (-alike) containing data.

Value

The standardized dataframe.

Examples

```
X <- matrix(rnorm(100, 0.2, 0.3), nrow = 20)
X_std <- zscore(X)
print(apply(X_std, 2, mean))
print(apply(X_std, 2, sd))
```

Index