libTLDA Documentation

Release 0.1.5

Wouter M. Kouw

Jan 21, 2019

Contents

libTLDA is a library of transfer learners and domain-adaptive classifiers. It is designed to give researchers and engineers an oppportunity to quickly test a number of classifiers.

More information will be added.

Contents:

CHAPTER 1

Installation

libTLDA is registered on PyPI and can be installed through:

```
pip install libtlda
```

1.1 Virtual environment

Pip takes care of all dependencies, but the addition of these dependencies can mess up your current python environment. To ensure a clean install, it is recommended to set up a virtual environment using conda or virtualenv. To ease this set up, an environment file is provided, which can be run through:

```
conda env create -f environment.yml
source activate libtlda
```

For more information on getting started, see the Examples section.

CHAPTER 2

Classifiers

This page contains the list of classes of classifiers including all member functions.

2.1 Importance-Weighted Classifier

class libtlda.iw.**ImportanceWeightedClassifier**(*loss_function='logistic', l2_regularization=None, weight_estimator='lr', smoothing=True, clip_max_value=-1, kernel_type='rbf', bandwidth=1*)

Class of importance-weighted classifiers.

Methods contain different importance-weight estimators and different loss functions.

Examples

>>>> X = np.random.randn(10, 2)
>>>> y = np.vstack((-np.ones((5,)), np.ones((5,))))
>>>> Z = np.random.randn(10, 2)
>>>> clf = ImportanceWeightedClassifier()
>>>> clf.fit(X, y, Z)
>>>> u_pred = clf.predict(Z)

Methods

fit(X, y, Z)	Fit/train an importance-weighted classifier.
get_params()	Get classifier parameters.

Continued on next page

Table 1 – continued from previous page

get_weights()	Get estimated importance weights.
is_trained()	Check whether classifier is trained.
iwe_kernel_densities(X, Z)	Estimate importance weights based on kernel density estimation.
iwe_kernel_mean_matching(X, Z)	Estimate importance weights based on kernel mean matching.
iwe_logistic_discrimination(X, Z)	Estimate importance weights based on logistic regression.
iwe_nearest_neighbours(X, Z)	Estimate importance weights based on nearest-neighbours.
iwe_ratio_gaussians(X, Z)	Estimate importance weights based on a ratio of Gaussian distributions.
predict(Z)	Make predictions on new dataset.
predict_proba(Z)	Compute posterior probabilities on new dataset.

fit (*X*, *y*, *Z*)

Fit/train an importance-weighted classifier.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

None

get_params ()

Get classifier parameters.

get_weights ()

Get estimated importance weights.

is_trained ()

Check whether classifier is trained.

iwe_kernel_densities (*X*, *Z*)

Estimate importance weights based on kernel density estimation.

Parameters

X [array] source data (N samples by D features)

Z [array] target data (M samples by D features)

Returns

array importance weights (N samples by 1)

iwe_kernel_mean_matching (*X*, *Z*)

Estimate importance weights based on kernel mean matching.

Parameters

X [array] source data (N samples by D features)

Z [array] target data (M samples by D features)

Returns

iw [array] importance weights (N samples by 1)

iwe_logistic_discrimination(X, Z)
Estimate importance weights based on logistic regression.

Parameters

X [array] source data (N samples by D features)

Z [array] target data (M samples by D features)

Returns

array importance weights (N samples by 1)

iwe_nearest_neighbours(X, Z)
Estimate importance weights based on nearest-neighbours.

Parameters

X [array] source data (N samples by D features)

Z [array] target data (M samples by D features)

Returns

iw [array] importance weights (N samples by 1)

iwe_ratio_gaussians(X, Z)
Estimate importance weights based on a ratio of Gaussian distributions.

Parameters

X [array] source data (N samples by D features)

Z [array] target data (M samples by D features)

Returns

iw [array] importance weights (N samples by 1)

predict(Z)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

predict_proba(Z)
Compute posterior probabilities on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

probs [array] label predictions (M samples by K)

2.2 Transfer Component Classifier

class libtlda.tca.**TransferComponentClassifier**(*loss_function='logistic'*, *l2_regularization=1.0*, *mu=1.0*, *num_components=1*, *kernel_type='rbf'*, *bandwidth=1.0*, *order=2.0*)

Class of classifiers based on Transfer Component Analysis.

Methods contain component analysis and general utilities.

Methods

fit(X, y, Z)	Fit/train a classifier on data mapped onto transfer components.
get_params()	Get classifier parameters.
is_trained()	Check whether classifier is trained.
kernel(X, Z[, type, order, bandwidth])	Compute kernel for given data set.
predict(Z)	Make predictions on new dataset.
transfer_component_analysis(X, Z)	Transfer Component Analysis.

fit(*X*, *y*, *Z*)

Fit/train a classifier on data mapped onto transfer components.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

None

get_params()

Get classifier parameters.

is_trained()

Check whether classifier is trained.

kernel(*X*, *Z*, *type='rbf'*, *order=2*, *bandwidth=1.0*)

Compute kernel for given data set.

Parameters

X [array] data set (N samples by D features)

Z [array] data set (M samples by D features)

type [str] type of kernel, options: 'linear', 'polynomial', 'rbf', 'sigmoid' (def: 'linear')

order [float] degree for the polynomial kernel (def: 2.0)

bandwidth [float] kernel bandwidth (def: 1.0)

Returns

array kernel matrix (N+M by N+M)

predict (*Z*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

transfer_component_analysis (*X*, *Z*)
Transfer Component Analysis.

Parameters

X [array] source data set (N samples by D features)

Z [array] target data set (M samples by D features)

Returns

C [array] transfer components (D features by num_components)

K [array] source and target data kernel distances

2.3 Subspace Aligned Classifier

class libtlda.suba.**SemiSubspaceAlignedClassifier** (*loss_function='logistic'*, *l2_regularization=None*, *subspace_dim=1*)

Class of classifiers based on semi-supervised Subspace Alignment.

Methods contain the alignment itself, classifiers and general utilities.

Examples

```
>>>> X = np.random.randn(10, 2)
>>>> y = np.vstack((-np.ones((5,)), np.ones((5,))))
>>>> Z = np.random.randn(10, 2)
>>>> clf = SubspaceAlignedClassifier()
>>>> clf.fit(X, y, Z)
>>>> preds = clf.predict(Z)
```

Methods

align_classes(X, Y, Z, u, CX, CZ, V)	Project each class separately.
find_medioid(X, Y)	Find point with minimal distance to all other points.
fit(X, Y, Z[, u])	Fit/train a classifier on data mapped onto transfer components.
get_params()	Get classifier parameters.
is_pos_def(A)	Check for positive definiteness.
predict(Z[, zscore])	Make predictions on new dataset.

Continued on next page

Table 3 – continued from previous page

predict_proba(Z[, zscore, signed_classes])	Make predictions on new dataset.
reg_cov(X)	Regularize covariance matrix until non-singular.
score(Z, U[, zscore])	Compute classification error on test set.
semi_subspace_alignment(X, Y, Z, u[, ...])	Compute subspace and alignment matrix, for each class.

align_classes (*X, Y, Z, u, CX, CZ, V*)
Project each class separately.

Parameters

X [array] source data set (N samples x D features)

Y [array] source labels (N samples x 1)

Z [array] target data set (M samples x D features)

u [array] target labels (m samples x 2)

CX [array] source principal components (K classes x D features x d subspaces)

CZ [array] target principal components (K classes x D features x d subspaces)

V [array] transformation matrix (K classes x d subspaces x d subspaces)

Returns

X [array] transformed X (N samples x d features)

Z [array] transformed Z (M samples x d features)

find_medioid (*X, Y*)
Find point with minimal distance to all other points.

Parameters

X [array] data set, with N samples x D features.

Y [array] labels to select for which samples to compute distances.

Returns

x [array] medioid

ix [int] index of medioid

fit (*X, Y, Z, u=None*)
Fit/train a classifier on data mapped onto transfer components.

Parameters

X [array] source data (N samples x D features).

Y [array] source labels (N samples x 1).

Z [array] target data (M samples x D features).

u [array] target labels, first column corresponds to index of Z and second column corresponds to actual label (number of labels x 2).

Returns

None

get_params ()
Get classifier parameters.

is_pos_def(*A*)
Check for positive definiteness.

A [array] square symmetric matrix.

Returns

bool whether matrix is positive-definite. Warning! Returns false for arrays containing inf or NaN.

predict(*Z*, *zscore=False*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples x D features)

zscore [boolean] whether to transform the data using z-scoring (def: false)

Returns

preds [array] label predictions (M samples x 1)

predict_proba(*Z*, *zscore=False*, *signed_classes=False*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples x D features)

zscore [boolean] whether to transform the data using z-scoring (def: false)

Returns

preds [array] label predictions (M samples x 1)

reg_cov(*X*)
Regularize covariance matrix until non-singular.

Parameters

C [array] square symmetric covariance matrix.

Returns

C [array] regularized covariance matrix.

score(*Z*, *U*, *zscore=False*)
Compute classification error on test set.

Parameters

Z [array] new data set (M samples x D features)

zscore [boolean] whether to transform the data using z-scoring (def: false)

Returns

preds [array] label predictions (M samples x 1)

semi_subspace_alignment(*X*, *Y*, *Z*, *u*, *subspace_dim=1*)
Compute subspace and alignment matrix, for each class.

Parameters

X [array] source data set (N samples x D features)

Y [array] source labels (N samples x 1)

Z [array] target data set (M samples x D features)

u [array] target labels, first column is index in Z, second column is label (m samples x 2)

subspace_dim [int] Dimensionality of subspace to retain (def: 1)

Returns

V [array] transformation matrix (K, D features x D features)

CX [array] source principal component coefficients

CZ [array] target principal component coefficients

class `libtlda.suba.`**`SubspaceAlignedClassifier`**(*loss_function='logistic'*, *l2_regularization=None*, *subspace_dim=1*)

Class of classifiers based on Subspace Alignment.

Methods contain the alignment itself, classifiers and general utilities.

Examples

>>>> X = np.random.randn(10, 2)
>>>> y = np.vstack((-np.ones((5,)), np.ones((5,))))
>>>> Z = np.random.randn(10, 2)
>>>> clf = SubspaceAlignedClassifier()
>>>> clf.fit(X, y, Z)
>>>> preds = clf.predict(Z)

Methods

`align_data`(X, Z, CX, CZ, V)	Align data to components and transform source.
`fit`(X, Y, Z)	Fit/train a classifier on data mapped onto transfer components.
`get_params`()	Get classifier parameters.
`is_pos_def`(A)	Check for positive definiteness.
`predict`(Z[, zscore])	Make predictions on new dataset.
`predict_proba`(Z[, zscore, signed_classes])	Make predictions on new dataset.
`reg_cov`(X)	Regularize covariance matrix until non-singular.
`score`(Z, U[, zscore])	Compute classification error on test set.
`subspace_alignment`(X, Z[, subspace_dim])	Compute subspace and alignment matrix.
`zca_whiten`(X)	Perform ZCA whitening (aka Mahalanobis whitening).

`align_data`(*X*, *Z*, *CX*, *CZ*, *V*)

Align data to components and transform source.

Parameters

X [array] source data set (N samples x D features)

Z [array] target data set (M samples x D features)

CX [array] source principal components (D features x d subspaces)

CZ [array] target principal component (D features x d subspaces)

V [array] transformation matrix (d subspaces x d subspaces)

Returns

X [array] transformed source data (N samples x d subspaces)

Z [array] projected target data (M samples x d subspaces)

fit (*X*, *Y*, *Z*)
Fit/train a classifier on data mapped onto transfer components.

Parameters

X [array] source data (N samples x D features).

Y [array] source labels (N samples x 1).

Z [array] target data (M samples x D features).

Returns

None

get_params ()
Get classifier parameters.

is_pos_def (*A*)
Check for positive definiteness.

predict (*Z*, *zscore=False*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples x D features)

zscore [boolean] whether to transform the data using z-scoring (def: false)

Returns

preds [array] label predictions (M samples x 1)

predict_proba (*Z*, *zscore=False*, *signed_classes=False*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples x D features)

zscore [boolean] whether to transform the data using z-scoring (def: false)

Returns

preds [array] label predictions (M samples x 1)

reg_cov (*X*)
Regularize covariance matrix until non-singular.

Parameters

C [array] square symmetric covariance matrix.

Returns

C [array] regularized covariance matrix.

score (*Z*, *U*, *zscore=False*)
Compute classification error on test set.

Parameters

Z [array] new data set (M samples x D features)

zscore [boolean] whether to transform the data using z-scoring (def: false)

Returns

preds [array] label predictions (M samples x 1)

subspace_alignment (*X*, *Z*, *subspace_dim=1*)
Compute subspace and alignment matrix.

Parameters

X [array] source data set (N samples x D features)

Z [array] target data set (M samples x D features)

subspace_dim [int] Dimensionality of subspace to retain (def: 1)

Returns

V [array] transformation matrix (D features x D features)

CX [array] source principal component coefficients

CZ [array] target principal component coefficients

zca_whiten (*X*)
Perform ZCA whitening (aka Mahalanobis whitening).

Parameters

X [array (M samples x D features)] data matrix.

Returns

X [array (M samples x D features)] whitened data.

2.4 Robust Bias-Aware Classifier

class libtlda.rba.**RobustBiasAwareClassifier** (*l2=0.0*, *order='first'*, *gamma=1.0*, *tau=1e-05*, *learning_rate=1.0*, *rate_decay='linear'*, *max_iter=100*, *clip=1000*, *verbose=True*)

Class of robust bias-aware classifiers.

Reference: Liu & Ziebart (20140. Robust Classification under Sample Selection Bias. NIPS.

Methods contain training and prediction functions.

Methods

feature_stats(X, y[, order])	Compute first-order moment feature statistics.
fit(X, y, Z)	Fit/train a robust bias-aware classifier.
get_params()	Get classifier parameters.
is_trained()	Check whether classifier is trained.

Continued on next page

Table 5 – continued from previous page

`iwe_kernel_densities`(X, Z[, clip])	Estimate importance weights based on kernel density estimation.
`learning_rate_t`(t)	Compute current learning rate after decay.
`posterior`(psi)	Class-posterior estimation.
`predict`(Z)	Make predictions on new dataset.
`predict_proba`(Z)	Compute posteriors on new dataset.
`psi`(X, theta, w[, K])	Compute psi function.

feature_stats (*X*, *y*, *order='first'*)
Compute first-order moment feature statistics.

Parameters

X [array] dataset (N samples by D features)

y [array] label vector (N samples by 1)

Returns

array array containing label vector, feature moments and 1-augmentation.

fit (*X*, *y*, *Z*)
Fit/train a robust bias-aware classifier.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

None

get_params ()
Get classifier parameters.

is_trained ()
Check whether classifier is trained.

iwe_kernel_densities (*X*, *Z*, *clip=1000*)
Estimate importance weights based on kernel density estimation.

Parameters

X [array] source data (N samples by D features)

Z [array] target data (M samples by D features)

clip [float] maximum allowed value for individual weights (def: 1000)

Returns

array importance weights (N samples by 1)

learning_rate_t (*t*)
Compute current learning rate after decay.

Parameters

t [int] current iteration

Returns

alpha [float] current learning rate

posterior (*psi*)
Class-posterior estimation.

Parameters

psi [array] weighted data-classifier output (N samples by K classes)

Returns

pyx [array] class-posterior estimation (N samples by K classes)

predict (*Z*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

predict_proba (*Z*)
Compute posteriors on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

psi (*X*, *theta*, *w*, *K=2*)
Compute psi function.

Parameters

X [array] data set (N samples by D features)

theta [array] classifier parameters (D features by 1)

w [array] importance-weights (N samples by 1)

K [int] number of classes (def: 2)

Returns

psi [array] array with psi function values (N samples by K classes)

2.5 Structural Correspondence Learner

class libtlda.scl.**StructuralCorrespondenceClassifier** (*loss='logistic'*, *l2=1.0*, *num_pivots=1*, *num_components=1*)

Class of classifiers based on structural correspondence learning.

Methods consist of a way to augment features, and a Huber loss function plus gradient.

Methods

`Huber_grad`(theta, X, y[, l2])	Huber gradient computation.
`Huber_loss`(theta, X, y[, l2])	Huber loss function.
`augment_features`(X, Z[, l2])	Find a set of pivot features, train predictors and extract bases.
`fit`(X, y, Z)	Fit/train an structural correpondence classifier.
`get_params`()	Get classifier parameters.
`is_trained`()	Check whether classifier is trained.
`predict`(Z)	Make predictions on new dataset.

`Huber_grad` (*theta*, *X*, *y*, *l2=0.0*)

Huber gradient computation.

Reference: Ando & Zhang (2005a). A framework for learning predictive structures from multiple tasks and unlabeled data. JMLR.

Parameters

theta [array] classifier parameters (D features by 1)

X [array] data (N samples by D features)

y [array] label vector (N samples by 1)

l2 [float] l2-regularization parameter (def= 0.0)

Returns

array Gradient with respect to classifier parameters

`Huber_loss` (*theta*, *X*, *y*, *l2=0.0*)

Huber loss function.

Reference: Ando & Zhang (2005a). A framework for learning predictive structures from multiple tasks and unlabeled data. JMLR.

Parameters

theta [array] classifier parameters (D features by 1)

X [array] data (N samples by D features)

y [array] label vector (N samples by 1)

l2 [float] l2-regularization parameter (def= 0.0)

Returns

array Objective function value.

`augment_features` (*X*, *Z*, *l2=0.0*)

Find a set of pivot features, train predictors and extract bases.

Parameters X : array

source data array (N samples by D features)

Z [array] target data array (M samples by D features)

l2 [float] regularization parameter value (def: 0.0)

Returns

None

fit (*X*, *y*, *Z*)
Fit/train an structural correpondence classifier.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

None

get_params ()
Get classifier parameters.

is_trained ()
Check whether classifier is trained.

predict (*Z*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

2.6 Feature-Level Domain-Adaptive Classifier

class libtlda.flda.**FeatureLevelDomainAdaptiveClassifier** (*l2=0.0*, *loss='logistic'*, *transfer_model='blankout'*, *max_iter=100*, *tolerance=1e-05*, *verbose=True*)

Class of feature-level domain-adaptive classifiers.

Reference: Kouw, Krijthe, Loog & Van der Maaten (2016). Feature-level domain adaptation. JMLR.

Methods contain training and prediction functions.

Methods

fit(X, y, Z)	Fit/train a robust bias-aware classifier.
flda_log_grad(theta, X, y, E, V[, l2])	Compute gradient with respect to theta for flda-log.
flda_log_loss(theta, X, y, E, V[, l2])	Compute average loss for flda-log.
get_params()	Get classifier parameters.
is_trained()	Check whether classifier is trained.
mle_transfer_dist(X, Z[, dist])	Maximum likelihood estimation of transfer model parameters.
moments_transfer_model(X, iota[, dist])	Moments of the transfer model.
predict(Z_)	Make predictions on new dataset.

fit (*X*, *y*, *Z*)
Fit/train a robust bias-aware classifier.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

None

flda_log_grad (*theta*, *X*, *y*, *E*, *V*, *l2=0.0*)
Compute gradient with respect to theta for flda-log.

Parameters

theta [array] classifier parameters (D features by 1)

X [array] source data set (N samples by D features)

y [array] label vector (N samples by 1)

E [array] expected value with respect to transfer model (N samples by D features)

V [array] variance with respect to transfer model (D features by D features by N samples)

l2 [float] regularization parameter (def: 0.0)

Returns

dR [array] Value of gradient.

flda_log_loss (*theta*, *X*, *y*, *E*, *V*, *l2=0.0*)
Compute average loss for flda-log.

Parameters

theta [array] classifier parameters (D features by 1)

X [array] source data set (N samples by D features)

y [array] label vector (N samples by 1)

E [array] expected value with respect to transfer model (N samples by D features)

V [array] variance with respect to transfer model (D features by D features by N samples)

l2 [float] regularization parameter (def: 0.0)

Returns

dL [array] Value of loss function.

get_params ()
Get classifier parameters.

is_trained ()
Check whether classifier is trained.

mle_transfer_dist (*X*, *Z*, *dist='blankout'*)
Maximum likelihood estimation of transfer model parameters.

Parameters

X [array] source data set (N samples by D features)

Z [array] target data set (M samples by D features)

dist [str] distribution of transfer model, options are 'blankout' or 'dropout' (def: 'blankout')

Returns

iota [array] estimated transfer model parameters (D features by 1)

moments_transfer_model (*X*, *iota*, *dist='blankout'*)
Moments of the transfer model.

Parameters

X [array] data set (N samples by D features)

iota [array] transfer model parameters (D samples by 1)

dist [str] transfer model, options are 'dropout' and 'blankout' (def: 'blankout')

Returns

E [array] expected value of transfer model (N samples by D feautures)

V [array] variance of transfer model (D features by D features by N samples)

predict (*Z_*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

2.7 Target Contrastive Pessimistic Classifier

class libtlda.tcpr.**TargetContrastivePessimisticClassifier** (*loss='lda'*, *l2=1.0*, *max_iter=500*, *tolerance=1e-12*, *learning_rate=1.0*, *rate_decay='linear'*, *verbosity=0*)

Classifiers based on Target Contrastive Pessimistic Risk minimization.

Methods contain models, risk functions, parameter estimation, etc.

Methods

add_intercept(X)	Add 1's to data as last features.
combine_class_covariances(Si, pi)	Linear combination of class covariance matrices.
discriminant_parameters(X, Y)	Estimate parameters of Gaussian distribution for discriminant analysis.
error_rate(preds, u_)	Compute classification error rate.
fit(X, y, Z)	Fit/train an importance-weighted classifier.
get_params()	Return classifier parameters.
learning_rate_t(t)	Compute current learning rate after decay.

Continued on next page

Table 8 – continued from previous page

neg_log_likelihood(X, theta)	Compute negative log-likelihood under Gaussian distributions.
predict(Z_)	Make predictions on new dataset.
predict_proba(Z)	Compute posteriors on new dataset.
project_simplex(v[, z])	Project vector onto simplex using sorting.
remove_intercept(X)	Remove 1's from data as last features.
risk(Z, theta, q)	Compute target contrastive pessimistic risk.
tcpr_da(X, y, Z)	Target Contrastive Pessimistic Risk - discriminant analysis.

add_intercept (*X*)
Add 1's to data as last features.

combine_class_covariances (*Si*, *pi*)
Linear combination of class covariance matrices.

Parameters

Si [array] Covariance matrix (D features by D features by K classes)

pi [array] class proportions (1 by K classes)

Returns

Si [array] Combined covariance matrix (D by D)

discriminant_parameters (*X*, *Y*)
Estimate parameters of Gaussian distribution for discriminant analysis.

Parameters

X [array] data array (N samples by D features)

Y [array] label array (N samples by K classes)

Returns

pi [array] class proportions (1 by K classes)

mu [array] class means (K classes by D features)

Si [array] class covariances (D features D features by K classes)

error_rate (*preds*, *u_*)
Compute classification error rate.

fit (*X*, *y*, *Z*)
Fit/train an importance-weighted classifier.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

None

get_params ()
Return classifier parameters.

learning_rate_t(*t*)
Compute current learning rate after decay.

Parameters

t [int] current iteration

Returns

alpha [float] current learning rate

neg_log_likelihood(*X*, *theta*)
Compute negative log-likelihood under Gaussian distributions.

Parameters

X [array] data (N samples by D features)

theta [tuple(array, array, array)] tuple containing class proportions 'pi', class means 'mu', and class-covariances 'Si'

Returns

L [array] loss (N samples by K classes)

predict(*Z_*)
Make predictions on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

predict_proba(*Z*)
Compute posteriors on new dataset.

Parameters

Z [array] new data set (M samples by D features)

Returns

preds [array] label predictions (M samples by 1)

project_simplex(*v*, *z=1.0*)
Project vector onto simplex using sorting.

Reference: "Efficient Projections onto the L1-Ball for Learning in High Dimensions (Duchi, Shalev-Shwartz, Singer, Chandra, 2006)."

Parameters

v [array] vector to be projected (n dimensions by 0)

z [float] constant (def: 1.0)

Returns

w [array] projected vector (n dimensions by 0)

remove_intercept(*X*)
Remove 1's from data as last features.

risk(*Z*, *theta*, *q*)
Compute target contrastive pessimistic risk.

Parameters

Z [array] target samples (M samples by D features)

theta [array] classifier parameters (D features by K classes)

q [array] soft labels (M samples by K classes)

Returns

float Value of risk function.

`tcpr_da` (*X*, *y*, *Z*)

Target Contrastive Pessimistic Risk - discriminant analysis.

Parameters

X [array] source data (N samples by D features)

y [array] source labels (N samples by 1)

Z [array] target data (M samples by D features)

Returns

theta [array] classifier parameters (D features by K classes)

CHAPTER 3

Examples

In the /demos folder, there are a number of example scripts. These show a potential use case on synthetic data.

Here we walk through a simple version.

First, we import a number of modules and generate a synthetic data set:

```
import numpy as np
import numpy.random as rnd

from sklearn.linear_model import LogisticRegression
from libtlda.iw import ImportanceWeightedClassifier

"""Generate synthetic data set"""

# Sample sizes
N = 100
M = 50

# Class properties
labels = [0, 1]
nK = 2

# Dimensionality
D = 2

# Source domain
pi_S = [1./2, 1./2]
si_S = 1.0
N0 = int(np.round(N*pi_S[0]))
N1 = N - N0
X0 = rnd.randn(N0, D)*si_S + (-2, 0)
X1 = rnd.randn(N1, D)*si_S + (+2, 0)
X = np.concatenate((X0, X1), axis=0)
y = np.concatenate((labels[0]*np.ones((N0,), dtype='int'),
                    labels[1]*np.ones((N1,), dtype='int')), axis=0)
```

(continues on next page)

(continued from previous page)

```
# Target domain
pi_T = [1./2, 1./2]
si_T = 3.0
M0 = int(np.round(M*pi_T[0]))
M1 = M - M0
Z0 = rnd.randn(M0, D)*si_T + (-2, -2)
Z1 = rnd.randn(M1, D)*si_T + (+2, +2)
Z = np.concatenate((Z0, Z1), axis=0)
u = np.concatenate((labels[0]*np.ones((M0,), dtype='int'),
                    labels[1]*np.ones((M1,), dtype='int')), axis=0)
```

Next, we create an adaptive classifier:

```
# Call an importance-weighted classifier
clf = ImportanceWeightedClassifier(iwe='lr', loss='logistic')

# Train classifier
clf.fit(X, y, Z)

# Make predictions
pred_adapt = clf.predict(Z)
```

We can compare this with a non-adaptive classifier:

```
# Train a naive logistic regressor
lr = LogisticRegression().fit(X, y)

# Make predictions
pred_naive = lr.predict(Z)
```

And compute error rates:

```
# Compute error rates
print('Error naive: ' + str(np.mean(pred_naive != u, axis=0)))
print('Error adapt: ' + str(np.mean(pred_adapt != u, axis=0)))
```

CHAPTER 4

Contact

Any comments, questions, or general feedback can be submitted to the repository's issues tracker.

If you would like to see a particular classifier / model / algorithm / technique / method for transfer learning or domain adaptation, please submit this to the issues tracker as well.

Python Module Index

l

Index

K

L

M

N

P

R

S

T

Z