Package ‘RSSL’

December 7, 2023

Version 0.9.7

Title Implementations of Semi-Supervised Learning Approaches for Classification

Depends R(>= 2.10.0)

Imports methods, Rcpp, MASS, kernlab, quadprog, Matrix, dplyr, tidyr, ggplot2, reshape2, scales, cluster

LinkingTo Rcpp, RcppArmadillo

Suggests testthat, rmarkdown, SparseM, numDeriv, LiblineaR, covr

Description A collection of implementations of semi-supervised classifiers and methods to evaluate their performance. The package includes implementations of, among others, Implicitly Constrained Learning, Moment Constrained Learning, the Transductive SVM, Manifold regularization, Maximum Contrastive Pessimistic Likelihood estimation, S4VM and WellSVM.

License GPL (>= 2)

URL https://github.com/jkrijthe/RSSL

BugReports https://github.com/jkrijthe/RSSL

Collate 'Generics.R' 'Classifier.R' 'CrossValidation.R' 'LeastSquaresClassifier.R' 'EMLeastSquaresClassifier.R' 'NormalBasedClassifier.R' 'LinearDiscriminantClassifier.R' 'EMLinearDiscriminantClassifier.R' 'NearestMeanClassifier.R' 'EMNearestMeanClassifier.R' 'LogisticRegression.R' 'EntropyRegularizedLogisticRegression.R' 'Evaluate.R' 'GRFClassifier.R' 'GenerateSSLData.R' 'HelperFunctions.R' 'ICLeastSquaresClassifier.R' 'ICLinearDiscriminantClassifier.R' 'KernelLeastSquaresClassifier.R' 'KernelICLeastSquaresClassifier.R' 'LaplacianKernelLeastSquaresClassifier.R' 'LaplacianSVM.R' 'LearningCurve.R' 'LinearSVM.R' 'LogisticLossClassifier.R' 'MCLinearDiscriminantClassifier.R' 'MCNearestMeanClassifier.R' 'MCPLDA.R' 'MajorityClassClassifier.R' 'Measures.R' 'Plotting.R' 'QuadraticDiscriminantClassifier.R' 'RSSL-package.R' 'RcppExports.R' 'S4VM.R' 'SVM.R'

'SelfLearning.R' 'TSVM.R' 'USMLeastSquaresClassifier.R' 'WellSVM.R' 'scaleMatrix.R' 'svmd.R' 'svmlin.R' 'testdata-data.R'

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Jesse Krijthe [aut, cre]

Maintainer Jesse Krijthe <jkrijthe@gmail.com>

Repository CRAN

Date/Publication 2023-12-07 06:20:03 UTC

R topics documented:

`add_missinglabels_mar` *Throw out labels at random*

Description

Original labels are saved in attribute `y_true`

Usage

```
add_missinglabels_mar(df, formula = NULL, prob = 0.1)
```

Arguments

`df`	data.frame; Data frame of interest
`formula`	formula; Formula to indicate the outputs
`prob`	numeric; Probability of removing the label

See Also

Other RSSL utilities: `LearningCurveSSL()`, `SSLDataFrameToMatrices()`, `df_to_matrices()`, `measure_accuracy()`, `missing_labels()`, `split_dataset_ssl()`, `split_random()`, `true_labels()`

adjacency_knn   *Calculate knn adjacency matrix*

Description

Calculates symmetric adjacency: objects are neighbours is either one of them is in the set of nearest neighbours of the other.

Usage

```
adjacency_knn(X, distance = "euclidean", k = 6)
```

Arguments

`X`	matrix; input matrix
`distance`	character; distance metric used in the `dist` function
`k`	integer; Number of neighbours

Value

Symmetric binary adjacency matrix

BaseClassifier   *Classifier used for enabling shared documenting of parameters*

Description

Classifier used for enabling shared documenting of parameters

Usage

```
BaseClassifier(X, y, X_u, verbose, scale, eps, x_center, intercept, lambda,
  y_scale, kernel, use_Xu_for_scaling, ...)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`verbose`	logical; Controls the verbosity of the output
`scale`	logical; Should the features be normalized? (default: FALSE)
`eps`	numeric; Stopping criterion for the maximinimization
`x_center`	logical; Should the features be centered?

`intercept`	logical; Whether an intercept should be included
`lambda`	numeric; L2 regularization parameter
`y_scale`	logical; whether the target vector should be centered
`kernel`	kernlab::kernel to use
`use_Xu_for_scaling`	logical; whether the unlabeled objects should be used to determine the mean and scaling for the normalization
`...`	Not used

`c.CrossValidation` *Merge result of cross-validation runs on single datasets into a the same object*

Description

Merge result of cross-validation runs on single datasets into a the same object

Usage

```
## S3 method for class 'CrossValidation'
c(...)
```

Arguments

`...`	Named arguments for the different objects, where the name reflects the dataset name

---

`clapply` *Use mclapply conditional on not being in RStudio*

Description

Use mclapply conditional on not being in RStudio

Usage

```
clapply(X, FUN, ..., mc.cores = getOption("mc.cores", 2L))
```

Arguments

`X`	vector
`FUN`	function to be applied to the elements of X
`...`	optional arguments passed to FUN
`mc.cores`	number of cores to use

cov_ml — *Biased (maximum likelihood) estimate of the covariance matrix*

Description

Biased (maximum likelihood) estimate of the covariance matrix

Usage

```
cov_ml(X)
```

Arguments

X	matrix with observations

CrossValidationSSL — *Cross-validation in semi-supervised setting*

Description

Cross-validation for semi-supervised learning, in which the dataset is split in three parts: labeled training object, unlabeled training object and validation objects. This can be used to evaluate different approaches to semi-supervised classification under the assumption the labels are missing at random. Different cross-validation schemes are implemented. See below for details.

Usage

```
CrossValidationSSL(X, y, ...)

## S3 method for class 'list'
CrossValidationSSL(X, y, ..., verbose = FALSE, mc.cores = 1)

## S3 method for class 'matrix'
CrossValidationSSL(X, y, classifiers, measures = list(Error
  = measure_error), k = 10, repeats = 1, verbose = FALSE,
  leaveout = "test", n_labeled = 10, prop_unlabeled = 0.5, time = TRUE,
  pre_scale = FALSE, pre_pca = FALSE, n_min = 1, low_level_cores = 1,
  ...)
```

Arguments

X	design matrix of the labeled objects
y	vector with labels
...	arguments passed to underlying functions
verbose	logical; Controls the verbosity of the output

mc.cores	integer; Number of cores to be used
classifiers	list; Classifiers to crossvalidate
measures	named list of functions giving the measures to be used
k	integer; Number of folds in the cross-validation
repeats	integer; Number of repeated assignments to folds
leaveout	either "labeled" or "test", see details
n_labeled	Number of labeled examples, used in both leaveout modes
prop_unlabeled	numeric; proportion of unlabeled objects
time	logical; Whether execution time should be saved.
pre_scale	logical; Whether the features should be scaled before the dataset is used
pre_pca	logical; Whether the features should be preprocessed using a PCA step
n_min	integer; Minimum number of labeled objects per class
low_level_cores	integer; Number of cores to use compute repeats of the learning curve

Details

The input to this function can be either: a dataset in the form of a feature matrix and factor containing the labels, a dataset in the form of a formula and data.frame or a named list of these two options. There are two main modes in which the cross-validation can be carried out, controlled by the `leaveout` parameter. When leaveout is "labeled", the folds are formed by non-overlapping labeled training sets of a user specified size. Each of these folds is used as a labeled set, while the rest of the objects are split into the an unlabeled and the test set, controlled by `prop_unlabeled` parameter. Note that objects can be used multiple times for testing, when training on a different fold, while other objects may never used for testing.

The "test" option of `leaveout`, on the other hand, uses the folds as the test sets. This means every object will be used as a test object exactly once. The remaining objects in each training iteration are split randomly into a labeled and an unlabeled part, where the number of the labeled objects is controlled by the user through the n_labeled parameter.

Examples

```
X <- model.matrix(Species~.-1,data=iris)
y <- iris$Species

classifiers <- list("LS"=function(X,y,X_u,y_u) {
  LeastSquaresClassifier(X,y,lambda=0)},
  "EM"=function(X,y,X_u,y_u) {
    SelfLearning(X,y,X_u,
                 method=LeastSquaresClassifier)}
)

measures <- list("Accuracy" =  measure_accuracy,
                 "Loss" = measure_losstest,
                 "Loss labeled" = measure_losslab,
                 "Loss Lab+Unlab" = measure_losstrain
```

```
)

# Cross-validation making sure test folds are non-overlapping
cvresults1 <- CrossValidationSSL(X,y,
                                 classifiers=classifiers,
                                 measures=measures,
                                 leaveout="test", k=10,
                                 repeats = 2,n_labeled = 10)
print(cvresults1)
plot(cvresults1)

# Cross-validation making sure labeled sets are non-overlapping
cvresults2 <- CrossValidationSSL(X,y,
                                 classifiers=classifiers,
                                 measures=measures,
                                 leaveout="labeled", k=10,
                                 repeats = 2,n_labeled = 10,
                                 prop_unlabeled=0.5)
print(cvresults2)
plot(cvresults2)
```

---

decisionvalues *Decision values returned by a classifier for a set of objects*

Description

Returns decision values of a classifier

Usage

```
decisionvalues(object, newdata)

## S4 method for signature 'LeastSquaresClassifier'
decisionvalues(object, newdata)

## S4 method for signature 'KernelLeastSquaresClassifier'
decisionvalues(object, newdata)

## S4 method for signature 'LinearSVM'
decisionvalues(object, newdata)

## S4 method for signature 'SVM'
decisionvalues(object, newdata)

## S4 method for signature 'TSVM'
decisionvalues(object, newdata)

## S4 method for signature 'svmlinClassifier'
decisionvalues(object, newdata)
```

Arguments

`object`	Classifier object
`newdata`	new data to classify

`df_to_matrices` *Convert data.frame with missing labels to matrices*

Description

Convert data.frame with missing labels to matrices

Usage

```
df_to_matrices(df, formula = NULL)
```

Arguments

`df`	data.frame; Data
`formula`	formula; Description of problem

See Also

Other RSSL utilities: `LearningCurveSSL()`, `SSLDataFrameToMatrices()`, `add_missinglabels_mar()`, `measure_accuracy()`, `missing_labels()`, `split_dataset_ssl()`, `split_random()`, `true_labels()`

`diabetes` *diabetes data for unit testing*

Description

Useful for testing the WellSVM implementation

EMLeastSquaresClassifier

An Expectation Maximization like approach to Semi-Supervised Least Squares Classification

Description

As studied in Krijthe & Loog (2016), minimizes the total loss of the labeled and unlabeled objects by finding the weight vector and labels that minimize the total loss. The algorithm proceeds similar to EM, by subsequently applying a weight update and a soft labeling of the unlabeled objects. This is repeated until convergence.

Usage

```
EMLeastSquaresClassifier(X, y, X_u, x_center = FALSE, scale = FALSE,
  verbose = FALSE, intercept = TRUE, lambda = 0, eps = 1e-09,
  y_scale = FALSE, alpha = 1, beta = 1, init = "supervised",
  method = "block", objective = "label", save_all = FALSE,
  max_iter = 1000)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
X_u	matrix; Design matrix for unlabeled data
x_center	logical; Should the features be centered?
scale	Should the features be normalized? (default: FALSE)
verbose	logical; Controls the verbosity of the output
intercept	logical; Whether an intercept should be included
lambda	numeric; L2 regularization parameter
eps	Stopping criterion for the minimization
y_scale	logical; whether the target vector should be centered
alpha	numeric; the mixture of the new responsibilities and the old in each iteration of the algorithm (default: 1)
beta	numeric; value between 0 and 1 that determines how much to move to the new solution from the old solution at each step of the block gradient descent
init	objective character; "random" for random initialization of labels, "supervised" to use supervised solution as initialization or a numeric vector with a coefficient vector to use to calculate the initialization
method	character; one of "block", for block gradient descent or "simple" for LBFGS optimization (default="block")
objective	character; "responsibility" for hard label self-learning or "label" for soft-label self-learning
save_all	logical; saves all classifiers trained during block gradient descent
max_iter	integer; maximum number of iterations

Details

By default (method="block") the weights of the classifier are updated, after which the unknown labels are updated. method="simple" uses LBFGS to do this update simultaneously. Objective="responsibility" corresponds to the responsibility based, instead of the label based, objective function in Krijthe & Loog (2016), which is equivalent to hard-label self-learning.

References

Krijthe, J.H. & Loog, M., 2016. Optimistic Semi-supervised Least Squares Classification. In International Conference on Pattern Recognition (To Appear).

See Also

Other RSSL classifiers: `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClassifier`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

Examples

```
library(dplyr)
library(ggplot2)

set.seed(1)

df <- generate2ClassGaussian(200,d=2,var=0.2) %>%
 add_missinglabels_mar(Class~.,prob = 0.96)

# Soft-label vs. hard-label self-learning
classifiers <- list(
 "Supervised"=LeastSquaresClassifier(Class~.,df),
 "EM-Soft"=EMLeastSquaresClassifier(Class~.,df,objective="label"),
 "EM-Hard"=EMLeastSquaresClassifier(Class~.,df,objective="responsibility")
)

df %>%
 ggplot(aes(x=X1,y=X2,color=Class)) +
 geom_point() +
 coord_equal() +
 scale_y_continuous(limits=c(-2,2)) +
 stat_classifier(aes(linetype=..classifier..),
                 classifiers=classifiers)
```

`EMLinearDiscriminantClassifier`

Semi-Supervised Linear Discriminant Analysis using Expectation Maximization

Description

Expectation Maximization applied to the linear discriminant classifier assuming Gaussian classes with a shared covariance matrix.

Usage

```
EMLinearDiscriminantClassifier(X, y, X_u, method = "EM", scale = FALSE,
  eps = 1e-08, verbose = FALSE, max_iter = 100)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`method`	character; Currently only "EM"
`scale`	logical; Should the features be normalized? (default: FALSE)
`eps`	Stopping criterion for the maximinimization
`verbose`	logical; Controls the verbosity of the output
`max_iter`	integer; Maximum number of iterations

Details

Starting from the supervised solution, uses the Expectation Maximization algorithm (see Dempster et al. (1977)) to iteratively update the means and shared covariance of the classes (Maximization step) and updates the responsibilities for the unlabeled objects (Expectation step).

References

Dempster, A., Laird, N. & Rubin, D., 1977. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society. Series B, 39(1), pp.1-38.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClassifier`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

EMNearestMeanClassifier

Semi-Supervised Nearest Mean Classifier using Expectation Maximization

Description

Expectation Maximization applied to the nearest mean classifier assuming Gaussian classes with a spherical covariance matrix.

Usage

```
EMNearestMeanClassifier(X, y, X_u, method = "EM", scale = FALSE,
  eps = 1e-04)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`method`	character; Currently only "EM"
`scale`	Should the features be normalized? (default: FALSE)
`eps`	Stopping criterion for the maximinimization

Details

Starting from the supervised solution, uses the Expectation Maximization algorithm (see Dempster et al. (1977)) to iteratively update the means and shared covariance of the classes (Maximization step) and updates the responsibilities for the unlabeled objects (Expectation step).

References

Dempster, A., Laird, N. & Rubin, D., 1977. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society. Series B, 39(1), pp.1-38.

`EntropyRegularizedLogisticRegression`
Entropy Regularized Logistic Regression

Description

R Implementation of entropy regularized logistic regression implementation as proposed by Grandvalet & Bengio (2005). An extra term is added to the objective function of logistic regression that penalizes the entropy of the posterior measured on the unlabeled examples.

Usage

```
EntropyRegularizedLogisticRegression(X, y, X_u = NULL, lambda = 0,
  lambda_entropy = 1, intercept = TRUE, init = NA, scale = FALSE,
  x_center = FALSE)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`lambda`	l2 Regularization
`lambda_entropy`	Weight of the labeled observations compared to the unlabeled observations
`intercept`	logical; Whether an intercept should be included
`init`	Initial parameters for the gradient descent
`scale`	logical; Should the features be normalized? (default: FALSE)
`x_center`	logical; Should the features be centered?

Value

S4 object of class EntropyRegularizedLogisticRegression with the following slots:

`w`	weight vector
`classnames`	the names of the classes

References

Grandvalet, Y. & Bengio, Y., 2005. Semi-supervised learning by entropy minimization. In L. K. Saul, Y. Weiss, & L. Bottou, eds. Advances in Neural Information Processing Systems 17. Cambridge, MA: MIT Press, pp. 529-536.

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)


# An example where ERLR finds a low-density separator, which is not
# the correct solution.
set.seed(1)
df <- generateSlicedCookie(1000,expected=FALSE) %>%
  add_missinglabels_mar(Class~.,0.98)

class_lr <- LogisticRegression(Class~.,df,lambda = 0.01)
class_erlr <- EntropyRegularizedLogisticRegression(Class~.,df,
                                lambda=0.01,lambda_entropy = 100)


ggplot(df,aes(x=X1,y=X2,color=Class)) +
  geom_point() +
  stat_classifier(aes(linetype=..classifier..),
                  classifiers = list("LR"=class_lr,"ERLR"=class_erlr)) +
  scale_y_continuous(limits=c(-2,2)) +
  scale_x_continuous(limits=c(-2,2))

df_test <- generateSlicedCookie(1000,expected=FALSE)
mean(predict(class_lr,df_test)==df_test$Class)
mean(predict(class_erlr,df_test)==df_test$Class)
```

`find_a_violated_label` *Find a violated label*

Description

Find a violated label

Usage

```
find_a_violated_label(alpha, K, y, ind_y, lr, y_init)
```

Arguments

alpha	classifier weights
K	kernel matrix
y	label vector

`ind_y`	Labeled/Unlabeled indicator
`lr`	positive ratio
`y_init`	label initialization

`gaussian_kernel` *calculated the gaussian kernel matrix*

Description

calculated the gaussian kernel matrix

Usage

```
gaussian_kernel(x, gamma, x_test = NULL)
```

Arguments

`x`	A d x n training data matrix
`gamma`	kernel parameter
`x_test`	A d x m testing data matrix

Value

k - A n x m kernel matrix and dis_mat - A n x m distance matrix

---

`generate2ClassGaussian` *Generate data from 2 Gaussian distributed classes*

Description

Generate data from 2 Gaussian distributed classes

Usage

```
generate2ClassGaussian(n = 10000, d = 100, var = 1, expected = TRUE)
```

Arguments

`n`	integer; Number of examples to generate
`d`	integer; dimensionality of the problem
`var`	numeric; size of the variance parameter
`expected`	logical; whether the decision boundary should be the expected or perpendicular

See Also

Other RSSL datasets: generateABA(), generateCrescentMoon(), generateFourClusters(), generateParallelPlanes(), generateSlicedCookie(), generateSpirals(), generateTwoCircles()

Examples

```
data <- generate2ClassGaussian(n=1000,d=2,expected=FALSE)
plot(data[,1],data[,2],col=data$Class,asp=1)
```

---

generateABA *Generate data from 2 alternating classes*

Description

Two clusters belonging to three classes: the cluster in the middle belongs to one class and the two on the outside to the others.

Usage

```
generateABA(n = 100, d = 2, var = 1)
```

Arguments

n	integer; Number of examples to generate
d	integer; dimensionality of the problem
var	numeric; size of the variance parameter

See Also

Other RSSL datasets: generate2ClassGaussian(), generateCrescentMoon(), generateFourClusters(), generateParallelPlanes(), generateSlicedCookie(), generateSpirals(), generateTwoCircles()

Examples

```
data <- generateABA(n=1000,d=2,var=1)
plot(data[,1],data[,2],col=data$Class,asp=1)
```

generateCrescentMoon *Generate Crescent Moon dataset*

Description

Generate a "crescent moon"/"banana" dataset

Usage

```
generateCrescentMoon(n = 100, d = 2, sigma = 1)
```

Arguments

`n`	integer; Number of objects to generate
`d`	integer; Dimensionality of the dataset
`sigma`	numeric; Noise added

See Also

Other RSSL datasets: `generate2ClassGaussian()`, `generateABA()`, `generateFourClusters()`, `generateParallelPlanes()`, `generateSlicedCookie()`, `generateSpirals()`, `generateTwoCircles()`

Examples

```
data<-generateCrescentMoon(150,2,1)
plot(data$X1,data$X2,col=data$Class,asp=1)
```

generateFourClusters *Generate Four Clusters dataset*

Description

Generate a four clusters dataset

Usage

```
generateFourClusters(n = 100, distance = 6, expected = FALSE)
```

Arguments

`n`	integer; Number of observations to generate
`distance`	numeric; Distance between clusters (default: 6)
`expected`	logical; TRUE if the large margin equals the class boundary, FALSE if the class boundary is perpendicular to the large margin

See Also

Other RSSL datasets: generate2ClassGaussian(), generateABA(), generateCrescentMoon(), generateParallelPlanes(), generateSlicedCookie(), generateSpirals(), generateTwoCircles()

Examples

```
data <- generateFourClusters(1000,distance=6,expected=TRUE)
plot(data[,1],data[,2],col=data$Class,asp=1)
```

---

generateParallelPlanes

Generate Parallel planes

Description

Generate Parallel planes

Usage

```
generateParallelPlanes(n = 100, classes = 3, sigma = 0.1)
```

Arguments

n	integer; Number of objects to generate
classes	integer; Number of classes
sigma	double; Noise added

See Also

Other RSSL datasets: generate2ClassGaussian(), generateABA(), generateCrescentMoon(), generateFourClusters(), generateSlicedCookie(), generateSpirals(), generateTwoCircles()

Examples

```
library(ggplot2)
df <- generateParallelPlanes(100,3)
ggplot(df, aes(x=x,y=y,color=Class,shape=Class)) +
 geom_point()
```

generateSlicedCookie *Generate Sliced Cookie dataset*

Description

Generate a sliced cookie dataset: a circle with a large margin in the middle.

Usage

```
generateSlicedCookie(n = 100, expected = FALSE, gap = 1)
```

Arguments

`n`	integer; number of observations to generate
`expected`	logical; TRUE if the large margin equals the class boundary, FALSE if the class boundary is perpendicular to the large margin
`gap`	numeric; Size of the gap

Value

A data.frame with n objects from the sliced cookie example

See Also

Other RSSL datasets: `generate2ClassGaussian()`, `generateABA()`, `generateCrescentMoon()`, `generateFourClusters()`, `generateParallelPlanes()`, `generateSpirals()`, `generateTwoCircles()`

Examples

```
data <- generateSlicedCookie(1000,expected=FALSE)
plot(data[,1],data[,2],col=data$Class,asp=1)
```

generateSpirals *Generate Intersecting Spirals*

Description

Generate Intersecting Spirals

Usage

```
generateSpirals(n = 100, sigma = 0.1)
```

Arguments

`n`	integer; Number of objects to generate per class
`sigma`	numeric; Noise added

See Also

Other RSSL datasets: generate2ClassGaussian(), generateABA(), generateCrescentMoon(), generateFourClusters(), generateParallelPlanes(), generateSlicedCookie(), generateTwoCircles()

Examples

```
data <- generateSpirals(100,sigma=0.1)
#plot3D::scatter3D(data$x,data$y,data$z,col="black")
```

---

generateTwoCircles *Generate data from 2 circles*

Description

One circle circumscribes the other

Usage

```
generateTwoCircles(n = 100, noise_var = 0.2)
```

Arguments

n	integer; Number of examples to generate
noise_var	numeric; size of the variance parameter

See Also

Other RSSL datasets: generate2ClassGaussian(), generateABA(), generateCrescentMoon(), generateFourClusters(), generateParallelPlanes(), generateSlicedCookie(), generateSpirals()

---

geom_classifier *Plot RSSL classifier boundary (deprecated)*

Description

Deprecated: Use geom_linearclassifier or stat_classifier to plot classification boundaries

Usage

```
geom_classifier(..., show_guide = TRUE)
```

Arguments

...	List of trained classifiers
show_guide	logical (default: TRUE); Show legend

geom_linearclassifier *Plot linear RSSL classifier boundary*

Description

Plot linear RSSL classifier boundary

Usage

```
geom_linearclassifier(..., show_guide = TRUE)
```

Arguments

`...`	List of trained classifiers
`show_guide`	logical (default: TRUE); Show legend

Examples

```
library(ggplot2)
library(dplyr)

df <- generate2ClassGaussian(100,d=2,var=0.2) %>%
 add_missinglabels_mar(Class~., 0.8)

df %>%
 ggplot(aes(x=X1,y=X2,color=Class)) +
 geom_point() +
 geom_linearclassifier("Supervised"=LinearDiscriminantClassifier(Class~.,df),
                       "EM"=EMLinearDiscriminantClassifier(Class~.,df))
```

GRFClassifier *Label propagation using Gaussian Random Fields and Harmonic functions*

Description

Implements the approach proposed in Zhu et al. (2003) to label propagation over an affinity graph. Note, as in the original paper, we consider the transductive scenario, so the implementation does not generalize to out of sample predictions. The approach minimizes the squared difference in labels assigned to different objects, where the contribution of each difference to the loss is weighted by the affinity between the objects. The default in this implementation is to use a knn adjacency matrix based on euclidean distance to determine this weight. Setting `adjacency="heat"` will use an RBF kernel over euclidean distances between objects to determine the weights.

Usage

```
GRFClassifier(X, y, X_u, adjacency = "nn",
  adjacency_distance = "euclidean", adjacency_k = 6,
  adjacency_sigma = 0.1, class_mass_normalization = FALSE, scale = FALSE,
  x_center = FALSE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
X_u	matrix; Design matrix for unlabeled data
adjacency	character; "nn" for nearest neighbour graph or "heat" for radial basis adjacency matrix
adjacency_distance	character; distance metric for nearest neighbour adjacency matrix
adjacency_k	integer; number of neighbours for the nearest neighbour adjacency matrix
adjacency_sigma	double; width of the rbf adjacency matrix
class_mass_normalization	logical; Should the Class Mass Normalization heuristic be applied? (default: FALSE)
scale	logical; Should the features be normalized? (default: FALSE)
x_center	logical; Should the features be centered?

References

Zhu, X., Ghahramani, Z. & Lafferty, J., 2003. Semi-supervised learning using gaussian fields and harmonic functions. In Proceedings of the 20th International Conference on Machine Learning. pp. 912-919.

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClassifier, LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantClassifier, S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)

set.seed(1)
df_circles <- generateTwoCircles(400,noise=0.1) %>%
```

```
  add_missinglabels_mar(Class~.,0.99)

# Visualize the problem
df_circles %>%
  ggplot(aes(x=X1,y=X2,color=Class)) +
  geom_point() +
  coord_equal()

# Visualize the solution
class_grf <- GRFClassifier(Class~.,df_circles,
                           adjacency="heat",
                           adjacency_sigma = 0.1)
df_circles %>%
  filter(is.na(Class)) %>%
  mutate(Responsibility=responsibilities(class_grf)[,1]) %>%
  ggplot(aes(x=X1,y=X2,color=Responsibility)) +
  geom_point() +
  coord_equal()

# Generate problem
df_para <- generateParallelPlanes()
df_para$Class <- NA
df_para$Class[1] <- "a"
df_para$Class[101] <- "b"
df_para$Class[201] <- "c"
df_para$Class <- factor(df_para$Class)

# Visualize problem
df_para %>%
  ggplot(aes(x=x,y=y,color=Class)) +
  geom_point() +
  coord_equal()

# Estimate GRF classifier with knn adjacency matrix (default)
class_grf <- GRFClassifier(Class~.,df_para)

df_para %>%
  filter(is.na(Class)) %>%
  mutate(Assignment=factor(apply(responsibilities(class_grf),1,which.max))) %>%
  ggplot(aes(x=x,y=y,color=Assignment)) +
  geom_point()
```

`harmonic_function` *Direct R Translation of Xiaojin Zhu's Matlab code to determine harmonic solution*

Description

Direct R Translation of Xiaojin Zhu's Matlab code to determine harmonic solution

Usage

```
harmonic_function(W, Y)
```

Arguments

W	matrix; weight matrix where the fist L rows/column correspond to the labeled examples.
Y	matrix; l by c 0,1 matrix encoding class assignments for the labeled objects

Value

The harmonic solution, i.e. eq (5) in the ICML paper, with or without class mass normalization

---

ICLeastSquaresClassifier

Implicitly Constrained Least Squares Classifier

Description

Implementation of the Implicitly Constrained Least Squares Classifier (ICLS) of Krijthe & Loog (2015) and the projected estimator of Krijthe & Loog (2016).

Usage

```
ICLeastSquaresClassifier(X, y, X_u = NULL, lambda1 = 0, lambda2 = 0,
  intercept = TRUE, x_center = FALSE, scale = FALSE, method = "LBFGS",
  projection = "supervised", lambda_prior = 0, trueprob = NULL,
  eps = 1e-09, y_scale = FALSE, use_Xu_for_scaling = TRUE)
```

Arguments

X	Design matrix, intercept term is added within the function
y	Vector or factor with class assignments
X_u	Design matrix of the unlabeled data, intercept term is added within the function
lambda1	Regularization parameter in the unlabeled+labeled data regularized least squares
lambda2	Regularization parameter in the labeled data only regularized least squares
intercept	TRUE if an intercept should be added to the model
x_center	logical; Whether the feature vectors should be centered
scale	logical; If TRUE, apply a z-transform to all observations in X and X_u before running the regression
method	Either "LBFGS" for solving using L-BFGS-B gradient descent or "QP" for a quadratic programming based solution
projection	One of "supervised", "semisupervised" or "euclidean"
lambda_prior	numeric; prior on the deviation from the supervised mean y

`trueprob`	numeric; true mean y for all data
`eps`	numeric; Stopping criterion for the maximinimization
`y_scale`	logical; whether the target vector should be centered
`use_Xu_for_scaling`	logical; whether the unlabeled objects should be used to determine the mean and scaling for the normalization

Details

In Implicitly Constrained semi-supervised Least Squares (ICLS) of Krijthe & Loog (2015), we minimize the quadratic loss on the labeled objects, while enforcing that the solution has to be a solution that minimizes the quadratic loss for all objects for some (fractional) labeling of the data (the implicit constraints). The goal of this classifier is to use the unlabeled data to update the classifier, while making sure it still works well on the labeled data.

The Projected estimator of Krijthe & Loog (2016) builds on this by finding a classifier within the space of classifiers that minimize the quadratic loss on all objects for some labeling (the implicit constrained), that minimizes the distance to the supervised solution for some appropriately chosen distance measure. Using the projection="semisupervised", we get certain guarantees that this solution is always better than the supervised solution (see Krijthe & Loog (2016)), while setting projection="supervised" is equivalent to ICLS.

Both methods (ICLS and the projection) can be formulated as a quadratic programming problem and solved using either a quadratic programming solver (method="QP") or using a gradient descent approach that takes into account certain bounds on the labelings (method="LBFGS"). The latter is the preferred method.

Value

S4 object of class ICLeastSquaresClassifier with the following slots:

`theta`	weight vector
`classnames`	the names of the classes
`modelform`	formula object of the model used in regression
`scaling`	a scaling object containing the parameters of the z-transforms applied to the data
`optimization`	the object returned by the optim function
`unlabels`	the labels assigned to the unlabeled objects

References

Krijthe, J.H. & Loog, M., 2015. Implicitly Constrained Semi-Supervised Least Squares Classification. In E. Fromont, T. De Bie, & M. van Leeuwen, eds. 14th International Symposium on Advances in Intelligent Data Analysis XIV (Lecture Notes in Computer Science Volume 9385). Saint Etienne. France, pp. 158-169.

Krijthe, J.H. & Loog, M., 2016. Projected Estimators for Robust Semi-supervised Classification. arXiv preprint arXiv:1602.07865.

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClassifier, LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantClassifier, S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

Examples

```
data(testdata)
w1 <- LeastSquaresClassifier(testdata$X, testdata$y,
                              intercept = TRUE,x_center = FALSE, scale=FALSE)
w2 <- ICLeastSquaresClassifier(testdata$X, testdata$y,
                              testdata$X_u, intercept = TRUE, x_center = FALSE, scale=FALSE)
plot(testdata$X[,1],testdata$X[,2],col=factor(testdata$y),asp=1)
points(testdata$X_u[,1],testdata$X_u[,2],col="darkgrey",pch=16,cex=0.5)

abline(line_coefficients(w1)$intercept,
       line_coefficients(w1)$slope,lty=2)
abline(line_coefficients(w2)$intercept,
       line_coefficients(w2)$slope,lty=1)
```

ICLinearDiscriminantClassifier

Implicitly Constrained Semi-supervised Linear Discriminant Classifier

---

Description

Semi-supervised version of Linear Discriminant Analysis using implicit constraints as described in (Krijthe & Loog 2014). This method finds the soft labeling of the unlabeled objects, whose resulting LDA solution gives the highest log-likelihood when evaluated on the labeled objects only. See also ICLeastSquaresClassifier.

Usage

```
ICLinearDiscriminantClassifier(X, y, X_u, prior = NULL, scale = FALSE,
  init = NULL, sup_prior = FALSE, x_center = FALSE, ...)
```

Arguments

X	design matrix of the labeled objects
y	vector with labels
X_u	design matrix of the labeled objects
prior	set a fixed class prior

`scale`	logical; Should the features be normalized? (default: FALSE)
`init`	not currently used
`sup_prior`	logical; use the prior estimates based only on the labeled data, not the imputed labels (default: FALSE)
`x_center`	logical; Whether the data should be centered
`...`	Additional Parameters, Not used

References

Krijthe, J.H. & Loog, M., 2014. Implicitly Constrained Semi-Supervised Linear Discriminant Analysis. In International Conference on Pattern Recognition. Stockholm, pp. 3762-3767.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClassifier`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`KernelICLeastSquaresClassifier`

Kernelized Implicitly Constrained Least Squares Classification

---

Description

A kernel version of the implicitly constrained least squares classifier, see `ICLeastSquaresClassifier`.

Usage

```
KernelICLeastSquaresClassifier(X, y, X_u, lambda = 0,
  kernel = vanilladot(), x_center = TRUE, scale = TRUE, y_scale = TRUE,
  lambda_prior = 0, classprior = 0, method = "LBFGS",
  projection = "semisupervised")
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`lambda`	numeric; L2 regularization parameter
`kernel`	kernlab::kernel to use
`x_center`	logical; Should the features be centered?

`scale`	logical; Should the features be normalized? (default: FALSE)
`y_scale`	logical; whether the target vector should be centered
`lambda_prior`	numeric; regularization parameter for the posterior deviation from the prior
`classprior`	The classprior used to compare the estimated responsibilities to
`method`	character; Estimation method. One of c("LBFGS")
`projection`	character; The projection used. One of c("supervised","semisupervised")

`KernelLeastSquaresClassifier`
Kernelized Least Squares Classifier

Description

Use least squares regression as a classification technique using a numeric encoding of classes as targets. Note this method minimizes quadratic loss, not the truncated quadratic loss.

Usage

```
KernelLeastSquaresClassifier(X, y, lambda = 0, kernel = vanilladot(),
  x_center = TRUE, scale = TRUE, y_scale = TRUE)
```

Arguments

`X`	Design matrix, intercept term is added within the function
`y`	Vector or factor with class assignments
`lambda`	Regularization parameter of the l2 penalty in regularized least squares
`kernel`	kernlab kernel function
`x_center`	TRUE, whether the dependent variables (features) should be centered
`scale`	If TRUE, apply a z-transform to the design matrix X before running the regression
`y_scale`	TRUE center the target vector

Value

S4 object of class LeastSquaresClassifier with the following slots:

`theta`	weight vector
`classnames`	the names of the classes
`modelform`	formula object of the model used in regression
`scaling`	a scaling object containing the parameters of the z-transforms applied to the data

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClassifier, LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantClassifier, S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)

# Two class problem
df <- generateCrescentMoon(200)

class_lin <- KernelLeastSquaresClassifier(Class~.,df,
                                          kernel=kernlab::vanilladot(), lambda=1)
class_rbf1 <- KernelLeastSquaresClassifier(Class~.,df,
                                          kernel=kernlab::rbfdot(), lambda=1)
class_rbf5 <- KernelLeastSquaresClassifier(Class~.,df,
                                          kernel=kernlab::rbfdot(5), lambda=1)
class_rbf10 <- KernelLeastSquaresClassifier(Class~.,df,
                                           kernel=kernlab::rbfdot(10), lambda=1)

df %>%
  ggplot(aes(x=X1,y=X2,color=Class,shape=Class)) +
  geom_point() +
  coord_equal() +
  stat_classifier(aes(linetype=..classifier..),
                  classifiers = list("Linear"=class_lin,
                                     "RBF sigma=1"=class_rbf1,
                                     "RBF sigma=5"=class_rbf5,
                                     "RBF sigma=10"=class_rbf10),
                  color="black")

# Second Example
dmat<-model.matrix(Species~.-1,iris[51:150,])
tvec<-droplevels(iris$Species[51:150])
testdata <- data.frame(tvec,dmat[,1:2])
colnames(testdata)<-c("Class","X1","X2")

precision<-100
xgrid<-seq(min(dmat[,1]),max(dmat[,1]),length.out=precision)
ygrid<-seq(min(dmat[,2]),max(dmat[,2]),length.out=precision)
gridmat <- expand.grid(xgrid,ygrid)

g_kernel<-KernelLeastSquaresClassifier(dmat[,1:2],tvec,
                                       kernel=kernlab::rbfdot(0.01),
                                       lambda=0.000001,scale = TRUE)
plotframe <- cbind(gridmat, decisionvalues(g_kernel,gridmat))
```

```
colnames(plotframe)<- c("x","y","Output")
ggplot(plotframe, aes(x=x,y=y)) +
  geom_tile(aes(fill = Output)) +
  scale_fill_gradient(low="yellow", high="red",limits=c(0,1)) +
  geom_point(aes(x=X1,y=X2,shape=Class),data=testdata,size=3) +
  stat_classifier(classifiers=list(g_kernel))

# Multiclass problem
dmat<-model.matrix(Species~.-1,iris)
tvec<-iris$Species
testdata <- data.frame(tvec,dmat[,1:2])
colnames(testdata)<-c("Class","X1","X2")

precision<-100
xgrid<-seq(min(dmat[,1]),max(dmat[,1]),length.out=precision)
ygrid<-seq(min(dmat[,2]),max(dmat[,2]),length.out=precision)
gridmat <- expand.grid(xgrid,ygrid)

g_kernel<-KernelLeastSquaresClassifier(dmat[,1:2],tvec,
                      kernel=kernlab::rbfdot(0.1),lambda=0.00001,
                      scale = TRUE,x_center=TRUE)

plotframe <- cbind(gridmat,
                   maxind=apply(decisionvalues(g_kernel,gridmat),1,which.max))
ggplot(plotframe, aes(x=Var1,y=Var2)) +
  geom_tile(aes(fill = factor(maxind,labels=levels(tvec)))) +
  geom_point(aes(x=X1,y=X2,shape=Class),data=testdata,size=4,alpha=0.5)
```

LaplacianKernelLeastSquaresClassifier

Laplacian Regularized Least Squares Classifier

Description

Implements manifold regularization through the graph Laplacian as proposed by Belkin et al. 2006. As an adjacency matrix, we use the k nearest neighbour graph based on a chosen distance (default: euclidean).

Usage

```
LaplacianKernelLeastSquaresClassifier(X, y, X_u, lambda = 0, gamma = 0,
  kernel = kernlab::vanilladot(), adjacency_distance = "euclidean",
  adjacency_k = 6, x_center = TRUE, scale = TRUE, y_scale = TRUE,
  normalized_laplacian = FALSE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector

`X_u`	matrix; Design matrix for unlabeled data
`lambda`	numeric; L2 regularization parameter
`gamma`	numeric; Weight of the unlabeled data
`kernel`	kernlab::kernel to use
`adjacency_distance`	character; distance metric used to construct adjacency graph from the dist function. Default: "euclidean"
`adjacency_k`	integer; Number of of neighbours used to construct adjacency graph.
`x_center`	logical; Should the features be centered?
`scale`	logical; Should the features be normalized? (default: FALSE)
`y_scale`	logical; whether the target vector should be centered
`normalized_laplacian`	logical; If TRUE use the normalized Laplacian, otherwise, the Laplacian is used

References

Belkin, M., Niyogi, P. & Sindhwani, V., 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. Journal of Machine Learning Research, 7, pp.2399-2434.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClassifier`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)

## Example 1: Half moons

# Generate a dataset
set.seed(2)
df_orig <- generateCrescentMoon(100,sigma = 0.3)
df <- df_orig %>%
  add_missinglabels_mar(Class~.,0.98)

lambda <- 0.01
gamma <- 10000
rbf_param <- 0.125

# Train classifiers
```

```
## Not run:
class_sup <- KernelLeastSquaresClassifier(
                Class~.,df,
                kernel=kernlab::rbfdot(rbf_param),
                lambda=lambda,scale=FALSE)

class_lap <- LaplacianKernelLeastSquaresClassifier(
                    Class~.,df,
                    kernel=kernlab::rbfdot(rbf_param),
                    lambda=lambda,gamma=gamma,
                    normalized_laplacian = TRUE,
                    scale=FALSE)

classifiers <- list("Lap"=class_lap,"Sup"=class_sup)

# Plot classifiers (can take a couple of seconds)

df %>%
  ggplot(aes(x=X1,y=X2,color=Class)) +
  geom_point() +
  coord_equal() +
  stat_classifier(aes(linetype=..classifier..),
                  classifiers = classifiers ,
                  color="black")


# Calculate the loss
lapply(classifiers,function(c) mean(loss(c,df_orig)))

## End(Not run)

## Example 2: Two circles
set.seed(1)
df_orig <- generateTwoCircles(1000,noise=0.05)
df <- df_orig %>%
  add_missinglabels_mar(Class~.,0.994)

lambda <- 10e-12
gamma <- 100
rbf_param <- 0.1

# Train classifiers
## Not run:
class_sup <- KernelLeastSquaresClassifier(
  Class~.,df,
  kernel=kernlab::rbfdot(rbf_param),
  lambda=lambda,scale=TRUE)

class_lap <- LaplacianKernelLeastSquaresClassifier(
  Class~.,df,
  kernel=kernlab::rbfdot(rbf_param),
  adjacency_k = 30,
  lambda=lambda,gamma=gamma,
```

```
  normalized_laplacian = TRUE,
  scale=TRUE)

classifiers <- list("Lap"=class_lap,"Sup"=class_sup)

# Plot classifiers (Can take a couple of seconds)
df %>%
  ggplot(aes(x=X1,y=X2,color=Class,size=Class)) +
  scale_size_manual(values=c("1"=3,"2"=3),na.value=1) +
  geom_point() +
  coord_equal() +
  stat_classifier(aes(linetype=..classifier..),
                  classifiers = classifiers ,
                  color="black",size=1)

## End(Not run)
```

`LaplacianSVM` *Laplacian SVM classifier*

Description

Manifold regularization applied to the support vector machine as proposed in Belkin et al. (2006). As an adjacency matrix, we use the k nearest neighbour graph based on a chosen distance (default: euclidean).

Usage

```
LaplacianSVM(X, y, X_u = NULL, lambda = 1, gamma = 1, scale = TRUE,
  kernel = vanilladot(), adjacency_distance = "euclidean",
  adjacency_k = 6, normalized_laplacian = FALSE, eps = 1e-09)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`lambda`	numeric; L2 regularization parameter
`gamma`	numeric; Weight of the unlabeled data
`scale`	logical; Should the features be normalized? (default: FALSE)
`kernel`	kernlab::kernel to use
`adjacency_distance`	character; distance metric used to construct adjacency graph from the dist function. Default: "euclidean"
`adjacency_k`	integer; Number of of neighbours used to construct adjacency graph.

normalized_laplacian
: logical; If TRUE use the normalized Laplacian, otherwise, the Laplacian is used

eps
: numeric; Small value to ensure positive definiteness of the matrix in the QP formulation

Value

S4 object of type LaplacianSVM

References

Belkin, M., Niyogi, P. & Sindhwani, V., 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. Journal of Machine Learning Research, 7, pp.2399-2434.

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LeastSquaresClassifier, LinearDiscriminantClassifier, LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantCl S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)

## Example 1: Half moons

# Generate a dataset
set.seed(2)
df_orig <- generateCrescentMoon(100,sigma = 0.3)
df <- df_orig %>%
  add_missinglabels_mar(Class~.,0.98)

lambda <- 0.001
C <- 1/(lambda*2*sum(!is.na(df$Class)))
gamma <- 10000
rbf_param <- 0.125

# Train classifiers
class_sup <- SVM(
  Class~.,df,
  kernel=kernlab::rbfdot(rbf_param),
  C=C,scale=FALSE)

class_lap <- LaplacianSVM(
  Class~.,df,
```

```
  kernel=kernlab::rbfdot(rbf_param),
  lambda=lambda,gamma=gamma,
  normalized_laplacian = TRUE,
  scale=FALSE)

classifiers <- list("Lap"=class_lap,"Sup"=class_sup)

# This takes a little longer to run:
# class_tsvm <- TSVM(
#   Class~.,df,
#   kernel=kernlab::rbfdot(rbf_param),
#   C=C,Cstar=10,s=-0.8,
#   scale=FALSE,balancing_constraint=TRUE)
# classifiers <- list("Lap"=class_lap,"Sup"=class_sup,"TSVM"=class_tsvm)

# Plot classifiers (Can take a couple of seconds)
## Not run:
df %>%
  ggplot(aes(x=X1,y=X2,color=Class)) +
  geom_point() +
  coord_equal() +
  stat_classifier(aes(linetype=..classifier..),
                  classifiers = classifiers ,
                  color="black")

## End(Not run)

# Calculate the loss
lapply(classifiers,function(c) mean(loss(c,df_orig)))

## Example 2: Two circles
set.seed(3)
df_orig <- generateTwoCircles(1000,noise=0.05)
df <- df_orig %>%
  add_missinglabels_mar(Class~.,0.994)

lambda <- 0.000001
C <- 1/(lambda*2*sum(!is.na(df$Class)))
gamma <- 100
rbf_param <- 0.1

# Train classifiers (Takes a couple of seconds)
## Not run:
class_sup <- SVM(
  Class~.,df,
  kernel=kernlab::rbfdot(rbf_param),
  C=C,scale=FALSE)

class_lap <- LaplacianSVM(
  Class~.,df,
  kernel=kernlab::rbfdot(rbf_param),
  adjacency_k=50, lambda=lambda,gamma=gamma,
  normalized_laplacian = TRUE,
```

```
  scale=FALSE)


classifiers <- list("Lap"=class_lap,"Sup"=class_sup)

## End(Not run)

# Plot classifiers (Can take a couple of seconds)
## Not run:
df %>%
  ggplot(aes(x=X1,y=X2,color=Class,size=Class)) +
  scale_size_manual(values=c("1"=3,"2"=3),na.value=1) +
  geom_point() +
  coord_equal() +
  stat_classifier(aes(linetype=..classifier..),
                  classifiers = classifiers ,
                  color="black",size=1)

## End(Not run)
```

`LearningCurveSSL` *Compute Semi-Supervised Learning Curve*

Description

Evaluate semi-supervised classifiers for different amounts of unlabeled training examples or different fractions of unlabeled vs. labeled examples.

Usage

```
LearningCurveSSL(X, y, ...)

## S3 method for class 'matrix'
LearningCurveSSL(X, y, classifiers, measures = list(Accuracy
  = measure_accuracy), type = "unlabeled", n_l = NULL,
  with_replacement = FALSE, sizes = 2^(1:8), n_test = 1000,
  repeats = 100, verbose = FALSE, n_min = 1, dataset_name = NULL,
  test_fraction = NULL, fracs = seq(0.1, 0.9, 0.1), time = TRUE,
  pre_scale = FALSE, pre_pca = FALSE, low_level_cores = 1, ...)
```

Arguments

`X`	design matrix
`y`	vector of labels
`...`	arguments passed to underlying function
`classifiers`	list; Classifiers to crossvalidate
`measures`	named list of functions giving the measures to be used

`type`	Type of learning curve, either "unlabeled" or "fraction"
`n_l`	Number of labeled objects to be used in the experiments (see details)
`with_replacement`	Indicated whether the subsampling is done with replacement or not (default: FALSE)
`sizes`	vector with number of unlabeled objects for which to evaluate performance
`n_test`	Number of test points if with_replacement is TRUE
`repeats`	Number of learning curves to draw
`verbose`	Print progressbar during execution (default: FALSE)
`n_min`	Minimum number of labeled objects per class in
`dataset_name`	character; Name of the dataset
`test_fraction`	numeric; If not NULL a fraction of the object will be left out to serve as the test set
`fracs`	list; fractions of labeled data to use
`time`	logical; Whether execution time should be saved.
`pre_scale`	logical; Whether the features should be scaled before the dataset is used
`pre_pca`	logical; Whether the features should be preprocessed using a PCA step
`low_level_cores`	integer; Number of cores to use compute repeats of the learning curve

Details

`classifiers` is a named list of classifiers, where each classifier should be a function that accepts 4 arguments: a numeric design matrix of the labeled objects, a factor of labels, a numeric design matrix of unlabeled objects and a factor of labels for the unlabeled objects.

`measures` is a named list of performance measures. These are functions that accept seven arguments: a trained classifier, a numeric design matrix of the labeled objects, a factor of labels, a numeric design matrix of unlabeled objects and a factor of labels for the unlabeled objects, a numeric design matrix of the test objects and a factor of labels of the test objects. See `measure_accuracy` for an example.

This function allows for two different types of learning curves to be generated. If `type="unlabeled"`, the number of labeled objects remains fixed at the value of `n_l`, where `sizes` controls the number of unlabeled objects. `n_test` controls the number of objects used for the test set, while all remaining objects are used if `with_replacement=FALSE` in which case objects are drawn without replacement from the input dataset. We make sure each class is represented by at least `n_min` labeled objects of each class. For `n_l`, additional options include: "enough" which takes the max of the number of features and 20, max(ncol(X)+5,20), "d" which takes the number of features or "2d" which takes 2 times the number of features.

If `type="fraction"` the total number of objects remains fixed, while the fraction of labeled objects is changed. `frac` sets the fractions of labeled objects that should be considered, while `test_fraction` determines the fraction of the total number of objects left out to serve as the test set.

Value

LearningCurve object

See Also

Other RSSL utilities: `SSLDataFrameToMatrices()`, `add_missinglabels_mar()`, `df_to_matrices()`, `measure_accuracy()`, `missing_labels()`, `split_dataset_ssl()`, `split_random()`, `true_labels()`

Examples

```
set.seed(1)
df <- generate2ClassGaussian(2000,d=2,var=0.6)

classifiers <- list("LS"=function(X,y,X_u,y_u) {
 LeastSquaresClassifier(X,y,lambda=0)},
  "Self"=function(X,y,X_u,y_u) {
    SelfLearning(X,y,X_u,LeastSquaresClassifier)}
)

measures <- list("Accuracy" =  measure_accuracy,
                 "Loss Test" = measure_losstest,
                 "Loss labeled" = measure_losslab,
                 "Loss Lab+Unlab" = measure_losstrain
)

# These take a couple of seconds to run
## Not run:
# Increase the number of unlabeled objects
lc1 <- LearningCurveSSL(as.matrix(df[,1:2]),df$Class,
                        classifiers=classifiers,
                        measures=measures, n_test=1800,
                        n_l=10,repeats=3)

plot(lc1)

# Increase the fraction of labeled objects, example with 2 datasets
lc2 <- LearningCurveSSL(X=list("Dataset 1"=as.matrix(df[,1:2]),
                               "Dataset 2"=as.matrix(df[,1:2])),
                        y=list("Dataset 1"=df$Class,
                               "Dataset 2"=df$Class),
                        classifiers=classifiers,
                        measures=measures,
                        type = "fraction",repeats=3,
                        test_fraction=0.9)

plot(lc2)

## End(Not run)
```

`LeastSquaresClassifier` *Least Squares Classifier*

Description

Classifier that minimizes the quadratic loss or, equivalently, least squares regression applied to a numeric encoding of the class labels as target. Note this method minimizes quadratic loss, not the truncated quadratic loss. Optionally, L2 regularization can be applied by setting the `lambda` parameter.

Usage

```
LeastSquaresClassifier(X, y, lambda = 0, intercept = TRUE,
  x_center = FALSE, scale = FALSE, method = "inverse", y_scale = FALSE)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`lambda`	Regularization parameter of the l2 penalty
`intercept`	TRUE if an intercept should be added to the model
`x_center`	TRUE, whether the dependent variables (features) should be centered
`scale`	If TRUE, apply a z-transform to the design matrix X before running the regression
`method`	Method to use for fitting. One of c("inverse","Normal","QR","BFGS")
`y_scale`	If True scale the target vector

Value

S4 object of class LeastSquaresClassifier with the following slots:

`theta`	weight vector
`classnames`	the names of the classes
`modelform`	formula object of the model used in regression
`scaling`	a scaling object containing the parameters of the z-transforms applied to the data

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LinearDiscriminantClassifier`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl` `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

LinearDiscriminantClassifier
Linear Discriminant Classifier

Description

Implementation of the linear discriminant classifier. Classes are modeled as Gaussians with different means but equal covariance matrices. The optimal covariance matrix and means for the classes are found using maximum likelihood, which, in this case, has a closed form solution.

Usage

```
LinearDiscriminantClassifier(X, y, method = "closedform", prior = NULL,
  scale = FALSE, x_center = FALSE)
```

Arguments

`X`	Design matrix, intercept term is added within the function
`y`	Vector or factor with class assignments
`method`	the method to use. Either "closedform" for the fast closed form solution or "ml" for explicit maximum likelihood maximization
`prior`	A matrix with class prior probabilities. If NULL, this will be estimated from the data
`scale`	logical; If TRUE, apply a z-transform to the design matrix X before running the regression
`x_center`	logical; Whether the feature vectors should be centered

Value

S4 object of class LeastSquaresClassifier with the following slots:

`modelform`	weight vector
`prior`	the prior probabilities of the classes
`mean`	the estimates means of the classes
`sigma`	The estimated covariance matrix
`classnames`	a vector with the classnames for each of the classes
`scaling`	scaling object used to transform new observations

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl`
`S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`LinearSVM` *Linear SVM Classifier*

Description

Implementation of the Linear Support Vector Classifier. Can be solved in the Dual formulation, which is equivalent to `SVM` or the Primal formulation.

Usage

```
LinearSVM(X, y, C = 1, method = "Dual", scale = TRUE, eps = 1e-09,
  reltol = 1e-13, maxit = 100)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`C`	Cost variable
`method`	Estimation procedure c("Dual","Primal","BGD")
`scale`	Whether a z-transform should be applied (default: TRUE)
`eps`	Small value to ensure positive definiteness of the matrix in QP formulation
`reltol`	relative tolerance using during BFGS optimization
`maxit`	Maximum number of iterations for BFGS optimization

Value

S4 object of type LinearSVM

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass` `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl` `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`LinearSVM-class` *LinearSVM Class*

Description

LinearSVM Class

LinearTSVM *Linear CCCP Transductive SVM classifier*

Description

Implementation for the Linear TSVM. This method is mostly for debugging purposes and does not allow for the balancing constraint or kernels, like the TSVM function.

Usage

```
LinearTSVM(X, y, X_u, C, Cstar, s = 0, x_center = FALSE, scale = FALSE,
  eps = 1e-06, verbose = FALSE, init = NULL)
```

Arguments

`X`	matrix; Design matrix, intercept term is added within the function
`y`	vector; Vector or factor with class assignments
`X_u`	matrix; Design matrix of the unlabeled data, intercept term is added within the function
`C`	numeric; Cost parameter of the SVM
`Cstar`	numeric; Cost parameter of the unlabeled objects
`s`	numeric; parameter controlling the loss function of the unlabeled objects
`x_center`	logical; Should the features be centered?
`scale`	logical; If TRUE, apply a z-transform to all observations in X and X_u before running the regression
`eps`	numeric; Convergence criterion
`verbose`	logical; print debugging messages (default: FALSE)
`init`	numeric; Initial classifier parameters to start the convex concave procedure

References

Collobert, R. et al., 2006. Large scale transductive SVMs. Journal of Machine Learning Research, 7, pp.1687-1712.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass` `LinearSVM`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl` `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`line_coefficients` *Loss of a classifier or regression function*

Description

Loss of a classifier or regression function

Usage

```
line_coefficients(object, ...)

## S4 method for signature 'LeastSquaresClassifier'
line_coefficients(object)

## S4 method for signature 'NormalBasedClassifier'
line_coefficients(object)

## S4 method for signature 'LogisticRegression'
line_coefficients(object)

## S4 method for signature 'LinearSVM'
line_coefficients(object)

## S4 method for signature 'LogisticLossClassifier'
line_coefficients(object)

## S4 method for signature 'QuadraticDiscriminantClassifier'
line_coefficients(object)

## S4 method for signature 'SelfLearning'
line_coefficients(object)
```

Arguments

`object`	Classifier; Trained Classifier object
`...`	Not used

Value

numeric of the total loss on the test data `localDescent` *Local descent*

Description

Local descent used in S4VM

Usage

```
localDescent(instance, label, labelNum, unlabelNum, gamma, C, beta, alpha)
```

Arguments

`instance`	Design matrix
`label`	label vector
`labelNum`	Number of labeled objects
`unlabelNum`	Number of unlabeled objects
`gamma`	Parameter for RBF kernel
`C`	cost parameter for SVM
`beta`	Controls fraction of objects assigned to positive class
`alpha`	Controls fraction of objects assigned to positive class

Value

list(predictLabel=predictLabel,acc=acc,values=values,model=model)

`LogisticLossClassifier` *Logistic Loss Classifier*

Description

Find the linear classifier which minimizing the logistic loss on the training set, optionally using L2 regularization.

Usage

```
LogisticLossClassifier(X, y, lambda = 0, intercept = TRUE, scale = FALSE,
  init = NA, x_center = FALSE, ...)
```

Arguments

`X`	Design matrix, intercept term is added within the function
`y`	Vector with class assignments
`lambda`	Regularization parameter used for l2 regularization
`intercept`	TRUE if an intercept should be added to the model
`scale`	If TRUE, apply a z-transform to all observations in X and X_u before running the regression
`init`	Starting parameter vector for gradient descent
`x_center`	logical; Whether the feature vectors should be centered
`...`	additional arguments

Value

S4 object with the following slots

`w`	the weight vector of the linear classifier
`classnames`	vector with names of the classes

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass:` `LinearSVM`, `LinearTSVM()`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`LogisticLossClassifier-class`
LogisticLossClassifier

Description

LogisticLossClassifier

LogisticRegression *(Regularized) Logistic Regression implementation*

Description

Implementation of Logistic Regression that is useful for comparisons with semi-supervised logistic regression implementations, such as EntropyRegularizedLogisticRegression.

Usage

```
LogisticRegression(X, y, lambda = 0, intercept = TRUE, scale = FALSE,
  init = NA, x_center = FALSE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
lambda	numeric; L2 regularization parameter
intercept	logical; Whether an intercept should be included
scale	logical; Should the features be normalized? (default: FALSE)
init	numeric; Initialization of parameters for the optimization
x_center	logical; Should the features be centered?

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClassi LinearSVM, LinearTSVM(), LogisticLossClassifier, MCLinearDiscriminantClassifier, MCNearestMeanClassifier MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantClassifier, S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

LogisticRegressionFast *Logistic Regression implementation that uses R's glm*

Description

Logistic Regression implementation that uses R's glm

Usage

```
LogisticRegressionFast(X, y, lambda = 0, intercept = TRUE, scale = FALSE,
  init = NA, x_center = FALSE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
lambda	numeric; not used
intercept	logical; Whether an intercept should be included
scale	logical; Should the features be normalized? (default: FALSE)
init	numeric; not used
x_center	logical; Should the features be centered?

---

logsumexp — *Numerically more stable way to calculate log sum exp*

Description

Numerically more stable way to calculate log sum exp

Usage

```
logsumexp(M)
```

Arguments

M	matrix; m by n input matrix, sum with be over the rows

Value

matrix; m by 1 matrix

---

loss — *Loss of a classifier or regression function*

Description

Hinge loss on new objects of a trained LinearSVM

Hinge loss on new objects of a trained SVM

Usage

```
loss(object, ...)

## S4 method for signature 'LeastSquaresClassifier'
loss(object, newdata, y = NULL, ...)

## S4 method for signature 'NormalBasedClassifier'
loss(object, newdata, y = NULL)

## S4 method for signature 'LogisticRegression'
loss(object, newdata, y = NULL)

## S4 method for signature 'KernelLeastSquaresClassifier'
loss(object, newdata, y = NULL, ...)

## S4 method for signature 'LinearSVM'
loss(object, newdata, y = NULL)

## S4 method for signature 'LogisticLossClassifier'
loss(object, newdata, y = NULL, ...)

## S4 method for signature 'MajorityClassClassifier'
loss(object, newdata, y = NULL)

## S4 method for signature 'SVM'
loss(object, newdata, y = NULL)

## S4 method for signature 'SelfLearning'
loss(object, newdata, y = NULL, ...)

## S4 method for signature 'USMLeastSquaresClassifier'
loss(object, newdata, y = NULL, ...)

## S4 method for signature 'svmlinClassifier'
loss(object, newdata, y = NULL)
```

Arguments

`object`	Classifier; Trained Classifier
`...`	additional parameters
`newdata`	data.frame; object with test data
`y`	factor; True classes of the test data

Value

numeric; the total loss on the test data losslogsum *LogsumLoss of a classifier or regression function*

Description

LogsumLoss of a classifier or regression function

Usage

```
losslogsum(object, ...)

## S4 method for signature 'NormalBasedClassifier'
losslogsum(object, newdata, Y, X_u, Y_u)
```

Arguments

object	Classifier or Regression object
...	Additional parameters
newdata	Design matrix of labeled objects
Y	label matrix of labeled objects
X_u	Design matrix of unlabeled objects
Y_u	label matrix of unlabeled objects

losspart *Loss of a classifier or regression function evaluated on partial labels*

Description

Loss of a classifier or regression function evaluated on partial labels

Usage

```
losspart(object, ...)

## S4 method for signature 'NormalBasedClassifier'
losspart(object, newdata, Y)
```

Arguments

object	Classifier; Trained Classifier
...	additional parameters
newdata	design matrix
Y	class responsibility matrix

MajorityClassClassifier

Majority Class Classifier

Description

Classifier that returns the majority class in the training set as the prediction for new objects.

Usage

```
MajorityClassClassifier(X, y, ...)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
...	Not used

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClass: LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, NearestMeanClassifier, QuadraticDiscriminantClassifier, S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

MCLinearDiscriminantClassifier

Moment Constrained Semi-supervised Linear Discriminant Analysis.

Description

A linear discriminant classifier that updates the estimates of the means and covariance matrix based on unlabeled examples.

Usage

```
MCLinearDiscriminantClassifier(X, y, X_u, method = "invariant",
  prior = NULL, x_center = TRUE, scale = FALSE)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`method`	character; One of c("invariant","closedform")
`prior`	Matrix (k by 1); Class prior probabilities. If NULL, estimated from data
`x_center`	logical; Should the features be centered?
`scale`	logical; Should the features be normalized? (default: FALSE)

Details

This method uses the parameter updates of the estimated means and covariance proposed in (Loog 2014). Using the method="invariant" option, uses the scale invariant parameter update proposed in (Loog 2014), while method="closedform" using the non-scale invariant version from (Loog 2012).

References

Loog, M., 2012. Semi-supervised linear discriminant analysis using moment constraints. Partially Supervised Learning, LNCS, 7081, pp.32-41.

Loog, M., 2014. Semi-supervised linear discriminant analysis through moment-constraint parameter estimation. Pattern Recognition Letters, 37, pp.24-31.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass`, `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`MCNearestMeanClassifier` *Moment Constrained Semi-supervised Nearest Mean Classifier*

Description

Update the means based on the moment constraints as defined in Loog (2010). The means estimated using the labeled data are updated by making sure their weighted mean corresponds to the overall mean on all (labeled and unlabeled) data. Optionally, the estimated variance of the classes can be re-estimated after this update is applied by setting update_sigma to `TRUE`. To get the true nearest mean classifier, rather than estimate the class priors, set them to equal priors using, for instance `prior=matrix(0.5,2)`.

Usage

```
MCNearestMeanClassifier(X, y, X_u, update_sigma = FALSE, prior = NULL,
  x_center = FALSE, scale = FALSE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
X_u	matrix; Design matrix for unlabeled data
update_sigma	logical; Whether the estimate of the variance should be updated after the means have been updated using the unlabeled data
prior	matrix; Class priors for the classes
x_center	logical; Should the features be centered?
scale	logical; Should the features be normalized? (default: FALSE)

References

Loog, M., 2010. Constrained Parameter Estimation for Semi-Supervised Learning: The Case of the Nearest Mean Classifier. In Proceedings of the 2010 European Conference on Machine learning and Knowledge Discovery in Databases. pp. 291-304.

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClass LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantClassifier, S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

MCPLDA — *Maximum Contrastive Pessimistic Likelihood Estimation for Linear Discriminant Analysis*

Description

Maximum Contrastive Pessimistic Likelihood (MCPL) estimation (Loog 2016) attempts to find a semi-supervised solution that has a higher likelihood compared to the supervised solution on the labeled and unlabeled data even for the worst possible labeling of the data. This is done by attempting to find a saddle point of the maximin problem, where the max is over the parameters of the semi-supervised solution and the min is over the labeling, while the objective is the difference in likelihood between the semi-supervised and the supervised solution measured on the labeled and unlabeled data. The implementation is a translation of the Matlab code of Loog (2016).

Usage

```
MCPLDA(X, y, X_u, x_center = FALSE, scale = FALSE, max_iter = 1000)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`x_center`	logical; Should the features be centered?
`scale`	logical; Should the features be normalized? (default: FALSE)
`max_iter`	integer; Maximum number of iterations

References

Loog, M., 2016. Contrastive Pessimistic Likelihood Estimation for Semi-Supervised Classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(3), pp.462-475.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass` `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantClassifie` `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`measure_accuracy` *Performance measures used in classifier evaluation*

Description

Classification accuracy on test set and other performance measure that can be used in `CrossValidationSSL` and `LearningCurveSSL`

Usage

```
measure_accuracy(trained_classifier, X_l = NULL, y_l = NULL, X_u = NULL,
  y_u = NULL, X_test = NULL, y_test = NULL)

measure_error(trained_classifier, X_l = NULL, y_l = NULL, X_u = NULL,
  y_u = NULL, X_test = NULL, y_test = NULL)

measure_losstest(trained_classifier, X_l = NULL, y_l = NULL, X_u = NULL,
  y_u = NULL, X_test = NULL, y_test = NULL)
```

```
measure_losslab(trained_classifier, X_l = NULL, y_l = NULL, X_u = NULL,
  y_u = NULL, X_test = NULL, y_test = NULL)

measure_losstrain(trained_classifier, X_l = NULL, y_l = NULL, X_u = NULL,
  y_u = NULL, X_test = NULL, y_test = NULL)
```

Arguments

`trained_classifier`	the trained classifier object
`X_l`	design matrix with labeled object
`y_l`	labels of labeled objects
`X_u`	design matrix with unlabeled object
`y_u`	labels of unlabeled objects
`X_test`	design matrix with test object
`y_test`	labels of test objects

Functions

- `measure_error()`: Classification error on test set
- `measure_losstest()`: Average Loss on test objects
- `measure_losslab()`: Average loss on labeled objects
- `measure_losstrain()`: Average loss on labeled and unlabeled objects

See Also

Other RSSL utilities: `LearningCurveSSL()`, `SSLDataFrameToMatrices()`, `add_missinglabels_mar()`, `df_to_matrices()`, `missing_labels()`, `split_dataset_ssl()`, `split_random()`, `true_labels()`

`minimaxlda` *Implements weighted likelihood estimation for LDA*

Description

Implements weighted likelihood estimation for LDA

Usage

```
minimaxlda(a, w, u, iter)
```

Arguments

`a`	is the data set
`w`	is an indicator matrix for the K classes of a or, potentially, a weight matrix in which the fraction with which a sample belongs to a particular class is indicated
`u`	is a bunch of unlabeled data
`iter`	decides on the amount of time we spend on minimaxing the stuff

Value

m contains the means, p contains the class priors, iW contains the INVERTED within covariance matrix, uw returns the weights for the unlabeled data, i returns the number of iterations used

---

`missing_labels` *Access the true labels for the objects with missing labels when they are stored as an attribute in a data frame*

Description

Access the true labels for the objects with missing labels when they are stored as an attribute in a data frame

Usage

```
missing_labels(df)
```

Arguments

`df`	data.frame; data.frame with y_true attribute

See Also

Other RSSL utilities: `LearningCurveSSL()`, `SSLDataFrameToMatrices()`, `add_missinglabels_mar()`, `df_to_matrices()`, `measure_accuracy()`, `split_dataset_ssl()`, `split_random()`, `true_labels()`

`NearestMeanClassifier` *Nearest Mean Classifier*

Description

Implementation of the nearest mean classifier modeled. Classes are modeled as gaussians with equal, spherical covariance matrices. The optimal covariance matrix and means for the classes are found using maximum likelihood, which, in this case, has a closed form solution. To get true nearest mean classification, set prior as a matrix with equal probability for all classes, i.e. `matrix(0.5,2)`.

Usage

```
NearestMeanClassifier(X, y, prior = NULL, x_center = FALSE,
  scale = FALSE)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`prior`	matrix; Class prior probabilities. If NULL, this will be estimated from the data
`x_center`	logical; Should the features be centered?
`scale`	logical; Should the features be normalized? (default: FALSE)

Value

S4 object of class LeastSquaresClassifier with the following slots:

`modelform`	weight vector
`prior`	the prior probabilities of the classes
`mean`	the estimates means of the classes
`sigma`	The estimated covariance matrix
`classnames`	a vector with the classnames for each of the classes
`scaling`	scaling object used to transform new observations

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClassi` `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `QuadraticDiscriminantClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`plot.CrossValidation` *Plot CrossValidation object*

Description

Plot CrossValidation object

Usage

```
## S3 method for class 'CrossValidation'
plot(x, y, ...)
```

Arguments

`x`	CrossValidation object
`y`	Not used
`...`	Not used

`plot.LearningCurve` *Plot LearningCurve object*

Description

Plot LearningCurve object

Usage

```
## S3 method for class 'LearningCurve'
plot(x, y, ...)
```

Arguments

`x`	LearningCurve object
`y`	Not used
`...`	Not used

`posterior` *Class Posteriors of a classifier*

Description

Class Posteriors of a classifier

Usage

```
posterior(object, ...)

## S4 method for signature 'NormalBasedClassifier'
posterior(object, newdata)

## S4 method for signature 'LogisticRegression'
posterior(object, newdata)
```

Arguments

`object`	Classifier or Regression object
`...`	Additional parameters
`newdata`	matrix of dataframe of objects to be classified

predict,scaleMatrix-method
Predict for matrix scaling inspired by stdize from the PLS package

Description

Predict for matrix scaling inspired by stdize from the PLS package

Usage

```
## S4 method for signature 'scaleMatrix'
predict(object, newdata, ...)
```

Arguments

`object`	scaleMatrix object
`newdata`	data to be scaled
`...`	Not used

PreProcessing *Preprocess the input to a classification function*

Description

The following actions are carried out: 1. data.frames are converted to matrix form and labels converted to an indicator matrix 2. An intercept column is added if requested 3. centering and scaling is applied if requested.

Usage

```
PreProcessing(X, y, X_u = NULL, scale = FALSE, intercept = FALSE,
  x_center = FALSE, use_Xu_for_scaling = TRUE)
```

Arguments

`X`	Design matrix, intercept term is added within the function
`y`	Vector or factor with class assignments
`X_u`	Design matrix of the unlabeled observations
`scale`	If TRUE, apply a z-transform to the design matrix X
`intercept`	Whether to include an intercept in the design matrices
`x_center`	logical (default: TRUE); Whether the feature vectors should be centered
`use_Xu_for_scaling`	logical (default: TRUE); Should the unlabeled data be used to determine scaling?

Value

list object with the following objects:

`X`	design matrix of the labeled data
`y`	integer vector indicating the labels of the labeled data
`X_u`	design matrix of the unlabeled data
`classnames`	names of the classes corresponding to the integers in y
`scaling`	a scaling object used to scale the test observations in the same way as the training set
`modelform`	a formula object containing the used model

`PreProcessingPredict` *Preprocess the input for a new set of test objects for classifier*

Description

The following actions are carried out: 1. data.frames are converted to matrix form and labels converted to integers 2. An intercept column is added if requested 3. centering and scaling is applied if requested.

Usage

```
PreProcessingPredict(modelform, newdata, y = NULL, classnames = NULL,
  scaling = NULL, intercept = FALSE)
```

Arguments

`modelform`	Formula object with model
`newdata`	data.frame object with objects
`y`	Vector or factor with class assignments (default: NULL)
`classnames`	Vector with class names
`scaling`	Apply a given z-transform to the design matrix X (default: NULL)
`intercept`	Whether to include an intercept in the design matrices

Value

list object with the following objects:

`X`	design matrix of the labeled data
`y`	integer vector indicating the labels of the labeled data

print.CrossValidation *Print CrossValidation object*

Description

Print CrossValidation object

Usage

```
## S3 method for class 'CrossValidation'
print(x, ...)
```

Arguments

x	CrossValidation object
...	Not used

print.LearningCurve *Print LearningCurve object*

Description

Print LearningCurve object

Usage

```
## S3 method for class 'LearningCurve'
print(x, ...)
```

Arguments

x	LearningCurve object
...	Not used

`projection_simplex` *Project an n-dim vector y to the simplex Dn*

Description

Where $Dn = \{0 <= x <= 1, sum(x) = 1\}$. R translation of Loog's version of Xiaojing Ye's initial implementation. The algorithm works row-wise.

Usage

```
projection_simplex(y)
```

Arguments

`y`	matrix with vectors to be projected onto the simplex

Value

projection of y onto the simplex

References

Algorithm is explained in http://arxiv.org/abs/1101.6081

`QuadraticDiscriminantClassifier` *Quadratic Discriminant Classifier*

Description

Implementation of the quadratic discriminant classifier. Classes are modeled as Gaussians with different covariance matrices. The optimal covariance matrix and means for the classes are found using maximum likelihood, which, in this case, has a closed form solution.

Usage

```
QuadraticDiscriminantClassifier(X, y, prior = NULL, scale = FALSE, ...)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`prior`	A matrix with class prior probabilities. If NULL, this will be estimated from the data
`scale`	logical; Should the features be normalized? (default: FALSE)
`...`	Not used

Value

S4 object of class LeastSquaresClassifier with the following slots:

`modelform`	weight vector
`prior`	the prior probabilities of the classes
`mean`	the estimates means of the classes
`sigma`	The estimated covariance matrix
`classnames`	a vector with the classnames for each of the classes
`scaling`	scaling object used to transform new observations

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass`
`LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `S4VM`, `SVM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

`responsibilities` *Responsibilities assigned to the unlabeled objects*

Description

Responsibilities assigned to the unlabeled objects

Usage

```
responsibilities(object, ...)
```

Arguments

`object`	Classifier; Trained Classifier
`...`	additional parameters

Value

numeric; responsibilities on the unlabeled objects

rssl-formatting — *Show RSSL classifier*

Description

Show RSSL classifier

Show the contents of a classifier

Usage

```
## S4 method for signature 'Classifier'
show(object)

## S4 method for signature 'NormalBasedClassifier'
show(object)

## S4 method for signature 'scaleMatrix'
show(object)
```

Arguments

`object`	classifier

rssl-predict — *Predict using RSSL classifier*

Description

Predict using RSSL classifier

For the SelfLearning Classifier the Predict Method delegates prediction to the specific model object

Usage

```
## S4 method for signature 'LeastSquaresClassifier'
predict(object, newdata, ...)

## S4 method for signature 'NormalBasedClassifier'
predict(object, newdata)

## S4 method for signature 'LogisticRegression'
predict(object, newdata)

## S4 method for signature 'GRFClassifier'
responsibilities(object, newdata, ...)
```

```
## S4 method for signature 'GRFClassifier'
predict(object, newdata = NULL, ...)

## S4 method for signature 'KernelLeastSquaresClassifier'
predict(object, newdata, ...)

## S4 method for signature 'LinearSVM'
predict(object, newdata)

## S4 method for signature 'LogisticLossClassifier'
predict(object, newdata)

## S4 method for signature 'MajorityClassClassifier'
predict(object, newdata)

## S4 method for signature 'SVM'
predict(object, newdata)

## S4 method for signature 'SelfLearning'
predict(object, newdata, ...)

## S4 method for signature 'USMLeastSquaresClassifier'
predict(object, newdata, ...)

## S4 method for signature 'WellSVM'
predict(object, newdata, ...)

## S4 method for signature 'WellSVM'
decisionvalues(object, newdata)

## S4 method for signature 'svmlinClassifier'
predict(object, newdata, ...)
```

Arguments

`object`	classifier
`newdata`	objects to generate predictions for
`...`	Other arguments

`S4VM` *Safe Semi-supervised Support Vector Machine (S4VM)*

Description

R port of the MATLAB implementation of Li & Zhou (2011) of the Safe Semi-supervised Support Vector Machine.

Usage

```
S4VM(X, y, X_u = NULL, C1 = 100, C2 = 0.1, sample_time = 100,
  gamma = 0, x_center = FALSE, scale = FALSE, lambda_tradeoff = 3)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`C1`	double; Regularization parameter for labeled data
`C2`	double; Regularization parameter for unlabeled data
`sample_time`	integer; Number of low-density separators that are generated
`gamma`	double; Width of RBF kernel
`x_center`	logical; Should the features be centered?
`scale`	logical; Should the features be normalized? (default: FALSE)
`lambda_tradeoff`	numeric; Parameter that determines the amount of "risk" in obtaining a worse solution than the supervised solution, see Li & Zhou (2011)

Details

The method randomly generates multiple low-density separators (controlled by the sample_time parameter) and merges their predictions by solving a linear programming problem meant to penalize the cost of decreasing the performance of the classifier, compared to the supervised SVM. S4VM is a bit of a misnomer, since it is a transductive method that only returns predicted labels for the unlabeled objects. The main difference in this implementation compared to the original implementation is the clustering of the low-density separators: in our implementation empty clusters are not dropped during the k-means procedure. In the paper by Li (2011) the features are first normalized to [0,1], which is not automatically done by this function. Note that the solution may not correspond to a linear classifier even if the linear kernel is used.

Value

S4VM object with slots:

`predictions`	Predictions on the unlabeled objects
`labelings`	Labelings for the different clusters

References

Yu-Feng Li and Zhi-Hua Zhou. Towards Making Unlabeled Data Never Hurt. In: Proceedings of the 28th International Conference on Machine Learning (ICML'11), Bellevue, Washington, 2011.

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClassi LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantCl SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM, svmlin()

Examples

```
library(RSSL)
library(dplyr)
library(ggplot2)
library(tidyr)

set.seed(1)
df_orig <- generateSlicedCookie(100,expected=TRUE)
df <- df_orig %>% add_missinglabels_mar(Class~.,0.95)
g_s <- SVM(Class~.,df,C=1,scale=TRUE,x_center=TRUE)
g_s4 <- S4VM(Class~.,df,C1=1,C2=0.1,lambda_tradeoff = 3,scale=TRUE,x_center=TRUE)

labs <- g_s4@labelings[-c(1:5),]
colnames(labs) <- paste("Class",seq_len(ncol(g_s4@labelings)),sep="-")

# Show the labelings that the algorithm is considering
df %>%
  filter(is.na(Class)) %>%
  bind_cols(data.frame(labs,check.names = FALSE)) %>%
  select(-Class) %>%
  gather(Classifier,Label,-X1,-X2) %>%
  ggplot(aes(x=X1,y=X2,color=Label)) +
  geom_point() +
  facet_wrap(~Classifier,ncol=5)

# Plot the final labeling that was selected
# Note that this may not correspond to a linear classifier
# even if the linear kernel is used.
# The solution does not seem to make a lot of sense,
# but this is what the current implementation returns
df %>%
  filter(is.na(Class)) %>%
  mutate(prediction=g_s4@predictions) %>%
  ggplot(aes(x=X1,y=X2,color=prediction)) +
  geom_point() +
  stat_classifier(color="black", classifiers=list(g_s))
```

S4VM-class *LinearSVM Class*

Description

LinearSVM Class

`sample_k_per_level` *Sample k indices per levels from a factor*

Description

Sample k indices per levels from a factor

Usage

```
sample_k_per_level(y, k)
```

Arguments

`y`	factor; factor with levels
`k`	integer; number of indices to sample per level

Value

vector with indices for sample

`scaleMatrix` *Matrix centering and scaling*

Description

This function returns an object with a predict method to center and scale new data. Inspired by stdize from the PLS package

Usage

```
scaleMatrix(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	matrix to be standardized
`center`	TRUE if x should be centered
`scale`	logical; TRUE of x should be scaled by the standard deviation

`SelfLearning` *Self-Learning approach to Semi-supervised Learning*

Description

Use self-learning (also known as Yarowsky's algorithm or pseudo-labeling) to turn any supervised classifier into a semi-supervised method by iteratively labeling the unlabeled objects and adding these predictions to the set of labeled objects until the classifier converges.

Usage

```
SelfLearning(X, y, X_u = NULL, method, prob = FALSE, cautious = FALSE,
  max_iter = 100, ...)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`X_u`	matrix; Design matrix for unlabeled data
`method`	Supervised classifier to use. Any function that accepts as its first argument a design matrix X and as its second argument a vector of labels y and that has a predict method.
`prob`	Not used
`cautious`	Not used
`max_iter`	integer; Maximum number of iterations
`...`	additional arguments to be passed to method

References

McLachlan, G.J., 1975. Iterative Reclassification Procedure for Constructing an Asymptotically Optimal Rule of Allocation in Discriminant Analysis. Journal of the American Statistical Association, 70(350), pp.365-369.

Yarowsky, D., 1995. Unsupervised word sense disambiguation rivaling supervised methods. Proceedings of the 33rd annual meeting on Association for Computational Linguistics, pp.189-196.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass`
`LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl`
`S4VM`, `SVM`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

Examples

```
data(testdata)
t_self <- SelfLearning(testdata$X,testdata$y,testdata$X_u,method=NearestMeanClassifier)
t_sup <- NearestMeanClassifier(testdata$X,testdata$y)
# Classification Error
1-mean(predict(t_self, testdata$X_test)==testdata$y_test)
1-mean(predict(t_sup, testdata$X_test)==testdata$y_test)
loss(t_self, testdata$X_test, testdata$y_test)
```

`solve_svm` *SVM solve.QP implementation*

Description

SVM solve.QP implementation

Usage

```
solve_svm(K, y, C = 1)
```

Arguments

`K`	Kernel matrix
`y`	Output vector
`C`	Cost parameter

`split_dataset_ssl` *Create Train, Test and Unlabeled Set*

Description

Create Train, Test and Unlabeled Set

Usage

```
split_dataset_ssl(X, y, frac_train = 0.8, frac_ssl = 0.8)
```

Arguments

`X`	matrix; Design matrix
`y`	factor; Label vector
`frac_train`	numeric; Fraction of all objects to be used as training objects
`frac_ssl`	numeric; Fraction of training objects to used as unlabeled objects

See Also

Other RSSL utilities: `LearningCurveSSL()`, `SSLDataFrameToMatrices()`, `add_missinglabels_mar()`, `df_to_matrices()`, `measure_accuracy()`, `missing_labels()`, `split_random()`, `true_labels()`

split_random *Randomly split dataset in multiple parts*

Description

The data.frame should start with a vector containing labels, or formula should be defined.

Usage

```
split_random(df, formula = NULL, splits = c(0.5, 0.5), min_class = 0)
```

Arguments

df	data.frame; Data frame of interest
formula	formula; Formula to indicate the outputs
splits	numeric; Probability of of assigning to each part, automatically normalized, should be >1
min_class	integer; minimum number of objects per class in each part

Value

list of data.frames

See Also

Other RSSL utilities: LearningCurveSSL(), SSLDataFrameToMatrices(), add_missinglabels_mar(), df_to_matrices(), measure_accuracy(), missing_labels(), split_dataset_ssl(), true_labels()

Examples

```
library(dplyr)

df <- generate2ClassGaussian(200,d=2)
dfs <- df %>% split_random(Class~.,split=c(0.5,0.3,0.2),min_class=1)
names(dfs) <- c("Train","Validation","Test")
lapply(dfs,summary)
```

`SSLDataFrameToMatrices` *Convert data.frame to matrices for semi-supervised learners*

Description

Given a formula object and a data.frame, extract the design matrix X for the labeled observations, X_u for the unlabeled observations and y for the labels of the labeled observations. Note: always removes the intercept

Usage

```
SSLDataFrameToMatrices(model, D)
```

Arguments

`model`	Formula object with model
`D`	data.frame object with objects

Value

list object with the following objects:

`X`	design matrix of the labeled data
`X_u`	design matrix of the unlabeled data
`y`	integer vector indicating the labels of the labeled data
`classnames`	names of the classes corresponding to the integers in y

See Also

Other RSSL utilities: `LearningCurveSSL()`, `add_missinglabels_mar()`, `df_to_matrices()`, `measure_accuracy()`, `missing_labels()`, `split_dataset_ssl()`, `split_random()`, `true_labels()`

`stat_classifier` *Plot RSSL classifier boundaries*

Description

Plot RSSL classifier boundaries

Usage

```
stat_classifier(mapping = NULL, data = NULL, show.legend = NA,
  inherit.aes = TRUE, breaks = 0, precision = 50, brute_force = FALSE,
  classifiers = classifiers, ...)
```

Arguments

`mapping`	aes; aesthetic mapping
`data`	data.frame; data to be displayed
`show.legend`	logical; Whether this layer should be included in the legend
`inherit.aes`	logical; If FALSE, overrides the default aesthetics
`breaks`	double; decision value for which to plot the boundary
`precision`	integer; grid size to sketch classification boundary
`brute_force`	logical; If TRUE, uses numerical estimation even for linear classifiers
`classifiers`	List of Classifier objects to plot
`...`	Additional parameters passed to geom

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)

df <- generateCrescentMoon(200)

# This takes a couple of seconds to run
## Not run:
g_svm <- SVM(Class~.,df,kernel = kernlab::rbfdot(sigma = 1))
g_ls <- LeastSquaresClassifier(Class~.,df)
g_nm <- NearestMeanClassifier(Class~.,df)


df %>%
  ggplot(aes(x=X1,y=X2,color=Class,shape=Class)) +
  geom_point(size=3) +
  coord_equal() +
  scale_x_continuous(limits=c(-20,20), expand=c(0,0)) +
  scale_y_continuous(limits=c(-20,20), expand=c(0,0)) +
  stat_classifier(aes(linetype=..classifier..),
                  color="black", precision=50,
                  classifiers=list("SVM"=g_svm,"NM"=g_nm,"LS"=g_ls)
  )

## End(Not run)
```

`stderror` *Calculate the standard error of the mean from a vector of numbers*

Description

Calculate the standard error of the mean from a vector of numbers

Usage

```
stderror(x)
```

Arguments

`x`	numeric; vector for which to calculate standard error

`summary.CrossValidation` *Summary of Crossvalidation results*

Description

Summary of Crossvalidation results

Usage

```
## S3 method for class 'CrossValidation'
summary(object, measure = NULL, ...)
```

Arguments

`object`	CrossValidation object
`measure`	Measure of interest
`...`	Not used

`svdinv` *Inverse of a matrix using the singular value decomposition*

Description

Inverse of a matrix using the singular value decomposition

Usage

```
svdinv(X)
```

Arguments

`X`	matrix; square input matrix

Value

Y matrix; inverse of the input matrix

svdinvsqrtm *Taking the inverse of the square root of the matrix using the singular value decomposition*

Description

Taking the inverse of the square root of the matrix using the singular value decomposition

Usage

```
svdinvsqrtm(X)
```

Arguments

X	matrix; square input matrix

Value

Y matrix; inverse of the square root of the input matrix

svdsqrtm *Taking the square root of a matrix using the singular value decomposition*

Description

Taking the square root of a matrix using the singular value decomposition

Usage

```
svdsqrtm(X)
```

Arguments

X	matrix; square input matrix

Value

Y matrix; square root of the input matrix

SVM *SVM Classifier*

Description

Support Vector Machine implementation using the quadprog solver.

Usage

```
SVM(X, y, C = 1, kernel = NULL, scale = TRUE, intercept = FALSE,
  x_center = TRUE, eps = 1e-09)
```

Arguments

`X`	matrix; Design matrix for labeled data
`y`	factor or integer vector; Label vector
`C`	numeric; Cost variable
`kernel`	kernlab::kernel to use
`scale`	logical; Should the features be normalized? (default: FALSE)
`intercept`	logical; Whether an intercept should be included
`x_center`	logical; Should the features be centered?
`eps`	numeric; Small value to ensure positive definiteness of the matrix in the QP formulation

Details

This implementation will typically be slower and use more memory than the svmlib implementation in the e1071 package. It is, however, useful for comparisons with the `TSVM` implementation.

Value

S4 object of type SVM

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass`
`LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl`
`S4VM`, `SelfLearning`, `TSVM`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

svmlin *svmlin implementation by Sindhwani & Keerthi (2006)*

Description

R interface to the svmlin code by Vikas Sindhwani and S. Sathiya Keerthi for fast linear transductive SVMs.

Usage

```
svmlin(X, y, X_u = NULL, algorithm = 1, lambda = 1, lambda_u = 1,
  max_switch = 10000, pos_frac = 0.5, Cp = 1, Cn = 1,
  verbose = FALSE, intercept = TRUE, scale = FALSE, x_center = FALSE)
```

Arguments

`X`	Matrix or sparseMatrix containing the labeled feature vectors, without intercept
`y`	factor containing class assignments
`X_u`	Matrix or sparseMatrix containing the unlabeled feature vectors, without intercept
`algorithm`	integer; Algorithm choice, see details (default:1)
`lambda`	double; Regularization parameter lambda (default 1)
`lambda_u`	double; Regularization parameter lambda_u (default 1)
`max_switch`	integer; Maximum number of switches in TSVM (default 10000)
`pos_frac`	double; Positive class fraction of unlabeled data (default 0.5)
`Cp`	double; Relative cost for positive examples (only available with algorithm 1)
`Cn`	double; Relative cost for positive examples (only available with algorithm 1)
`verbose`	logical; Controls the verbosity of the output
`intercept`	logical; Whether an intercept should be included
`scale`	logical; Should the features be normalized? (default: FALSE)
`x_center`	logical; Should the features be centered?

Details

The codes to select the algorithm are the following: 0. Regularized Least Squares Classification 1. SVM (L2-SVM-MFN) 2. Multi-switch Transductive SVM (using L2-SVM-MFN) 3. Deterministic Annealing Semi-supervised SVM (using L2-SVM-MFN).

References

Vikas Sindhwani and S. Sathiya Keerthi. Large Scale Semi-supervised Linear SVMs. Proceedings of ACM SIGIR, 2006 @references V. Sindhwani and S. Sathiya Keerthi. Newton Methods for Fast Solution of Semi-supervised Linear SVMs. Book Chapter in Large Scale Kernel Machines, MIT Press, 2006

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClass: LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantCl S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, WellSVM

Examples

```
data(svmlin_example)
t_svmlin_1 <- svmlin(svmlin_example$X_train[1:50,],
                 svmlin_example$y_train,X_u=NULL, lambda = 0.001)
t_svmlin_2 <- svmlin(svmlin_example$X_train[1:50,],
                       svmlin_example$y_train,
                       X_u=svmlin_example$X_train[-c(1:50),],
                       lambda = 10,lambda_u=100,algorithm = 2)

# Calculate Accuracy
mean(predict(t_svmlin_1,svmlin_example$X_test)==svmlin_example$y_test)
mean(predict(t_svmlin_2,svmlin_example$X_test)==svmlin_example$y_test)

data(testdata)

g_svm <- SVM(testdata$X,testdata$y)
g_sup <- svmlin(testdata$X,testdata$y,testdata$X_u,algorithm = 3)
g_semi <- svmlin(testdata$X,testdata$y,testdata$X_u,algorithm = 2)

mean(predict(g_svm,testdata$X_test)==testdata$y_test)
mean(predict(g_sup,testdata$X_test)==testdata$y_test)
mean(predict(g_semi,testdata$X_test)==testdata$y_test)
```

---

svmlin_example *Test data from the svmlin implementation*

Description

Useful for testing the svmlin interface and to serve as an example

---

svmproblem *Train SVM*

Description

Train SVM

Usage

```
svmproblem(K)
```

Arguments

`K`	kernel

Value

alpha, b, obj

`testdata` *Example semi-supervised problem*

Description

A list containing a sample from the `GenerateSlicedCookie` dataset for unit testing and examples.

`threshold` *Refine the prediction to satisfy the balance constraint*

Description

Refine the prediction to satisfy the balance constraint

Usage

```
threshold(y1, options)
```

Arguments

`y1`	predictions
`options`	options passed

Value

y2

true_labels — *Access the true labels when they are stored as an attribute in a data frame*

Description

Access the true labels when they are stored as an attribute in a data frame

Usage

```
true_labels(df)
```

Arguments

df	data.frame; data.frame with y_true attribute

See Also

Other RSSL utilities: `LearningCurveSSL()`, `SSLDataFrameToMatrices()`, `add_missinglabels_mar()`, `df_to_matrices()`, `measure_accuracy()`, `missing_labels()`, `split_dataset_ssl()`, `split_random()`

TSVM — *Transductive SVM classifier using the convex concave procedure*

Description

Transductive SVM using the CCCP algorithm as proposed by Collobert et al. (2006) implemented in R using the quadprog package. The implementation does not handle large datasets very well, but can be useful for smaller datasets and visualization purposes.

Usage

```
TSVM(X, y, X_u, C, Cstar, kernel = kernlab::vanilladot(),
  balancing_constraint = TRUE, s = 0, x_center = TRUE, scale = FALSE,
  eps = 1e-09, max_iter = 20, verbose = FALSE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
X_u	matrix; Design matrix for unlabeled data
C	numeric; Cost parameter of the SVM
Cstar	numeric; Cost parameter of the unlabeled objects
kernel	kernlab::kernel to use

`balancing_constraint`
: logical; Whether a balancing constraint should be enforced that causes the fraction of objects assigned to each label in the unlabeled data to be similar to the label fraction in the labeled data.

`s`
: numeric; parameter controlling the loss function of the unlabeled objects (generally values between -1 and 0)

`x_center`
: logical; Should the features be centered?

`scale`
: If TRUE, apply a z-transform to all observations in X and X_u before running the regression

`eps`
: numeric; Stopping criterion for the maximinimization

`max_iter`
: integer; Maximum number of iterations

`verbose`
: logical; print debugging messages, only works for vanilladot() kernel (default: FALSE)

Details

C is the cost associated with labeled objects, while Cstar is the cost for the unlabeled objects. s control the loss function used for the unlabeled objects: it controls the size of the plateau for the symmetric ramp loss function. The balancing constraint makes sure the label assignments of the unlabeled objects are similar to the prior on the classes that was observed on the labeled data.

References

Collobert, R. et al., 2006. Large scale transductive SVMs. Journal of Machine Learning Research, 7, pp.1687-1712.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass` `LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl` `S4VM`, `SVM`, `SelfLearning`, `USMLeastSquaresClassifier`, `WellSVM`, `svmlin()`

Examples

```
library(RSSL)

# Simple example with a few objects
X <- matrix(c(0,0.001,1,-1),nrow=2)
X_u <- matrix(c(-1,-1,-1,0,0,0,-0.4,-0.5,-0.6,1.2,1.3,1.25),ncol=2)
y <- factor(c(-1,1))

g_sup <- SVM(X,y,scale=FALSE)
g_constraint <- TSVM(X=X,y=y,X_u=X_u,
                     C=1,Cstar=0.1,balancing_constraint = TRUE)

g_noconstraint <- TSVM(X=X,y=y,X_u=X_u,
```

```
                       C=1,Cstar=0.1,balancing_constraint = FALSE)

g_lin <- LinearTSVM(X=X,y=y,X_u=X_u,C=1,Cstar=0.1)

w1 <- g_sup@alpha %*% X
w2 <- g_constraint@alpha %*% rbind(X,X_u,X_u,colMeans(X_u))
w3 <- g_noconstraint@alpha %*% rbind(X,X_u,X_u)
w4 <- g_lin@w

plot(X[,1],X[,2],col=factor(y),asp=1,ylim=c(-3,3))
points(X_u[,1],X_u[,2],col="darkgrey",pch=16,cex=1)
abline(-g_sup@bias/w1[2],-w1[1]/w1[2],lty=2)
abline(((1-g_sup@bias)/w1[2]),-w1[1]/w1[2],lty=2) # +1 Margin
abline(((-1-g_sup@bias)/w1[2]),-w1[1]/w1[2],lty=2) # -1 Margin
abline(-g_constraint@bias/w2[2],-w2[1]/w2[2],lty=1,col="green")
abline(-g_noconstraint@bias/w3[2],-w3[1]/w3[2],lty=1,col="red")
abline(-w4[1]/w4[3],-w4[2]/w4[3],lty=1,lwd=3,col="blue")

# An example
set.seed(42)
data <- generateSlicedCookie(200,expected=TRUE,gap=1)
X <- model.matrix(Class~.-1,data)
y <- factor(data$Class)

problem <- split_dataset_ssl(X,y,frac_ssl=0.98)

X <- problem$X
y <- problem$y
X_u <- problem$X_u
y_e <- unlist(list(problem$y,problem$y_u))
Xe<-rbind(X,X_u)

g_sup <- SVM(X,y,x_center=FALSE,scale=FALSE,C = 10)
g_constraint <- TSVM(X=X,y=y,X_u=X_u,
                     C=10,Cstar=10,balancing_constraint = TRUE,
                     x_center = FALSE,verbose=TRUE)

g_noconstraint <- TSVM(X=X,y=y,X_u=X_u,
                       C=10,Cstar=10,balancing_constraint = FALSE,
                       x_center = FALSE,verbose=TRUE)

g_lin <- LinearTSVM(X=X,y=y,X_u=X_u,C=10,Cstar=10,
                    verbose=TRUE,x_center = FALSE)

g_oracle <- SVM(Xe,y_e,scale=FALSE)

w1 <- c(g_sup@bias,g_sup@alpha %*% X)
w2 <- c(g_constraint@bias,g_constraint@alpha %*% rbind(X,X_u,X_u,colMeans(X_u)))
w3 <- c(g_noconstraint@bias,g_noconstraint@alpha %*% rbind(X,X_u,X_u))
w4 <- g_lin@w
w5 <- c(g_oracle@bias, g_oracle@alpha %*% Xe)
print(sum(abs(w4-w3)))
```

```
plot(X[,1],X[,2],col=factor(y),asp=1,ylim=c(-3,3))
points(X_u[,1],X_u[,2],col="darkgrey",pch=16,cex=1)
abline(-w1[1]/w1[3],-w1[2]/w1[3],lty=2)
abline(((1-w1[1])/w1[3]),-w1[2]/w1[3],lty=2) # +1 Margin
abline(((-1-w1[1])/w1[3]),-w1[2]/w1[3],lty=2) # -1 Margin

# Oracle:
abline(-w5[1]/w5[3],-w5[2]/w5[3],lty=1,col="purple")

# With balancing constraint:
abline(-w2[1]/w2[3],-w2[2]/w2[3],lty=1,col="green")

# Linear TSVM implementation (no constraint):
abline(-w4[1]/w4[3],-w4[2]/w4[3],lty=1,lwd=3,col="blue")

# Without balancing constraint:
abline(-w3[1]/w3[3],-w3[2]/w3[3],lty=1,col="red")
```

USMLeastSquaresClassifier *Updated Second Moment Least Squares Classifier*

Description

This methods uses the closed form solution of the supervised least squares problem, except that the second moment matrix (X'X) is exchanged with a second moment matrix that is estimated based on all data. See for instance *Shaffer1991*, where in this implementation we use all data to estimate E(X'X), instead of just the labeled data. This method seems to work best when the data is first centered `x_center=TRUE` and the outputs are scaled using `y_scale=TRUE`.

Usage

```
USMLeastSquaresClassifier(X, y, X_u, lambda = 0, intercept = TRUE,
  x_center = FALSE, scale = FALSE, y_scale = FALSE, ...,
  use_Xu_for_scaling = TRUE)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
X_u	matrix; Design matrix for unlabeled data
lambda	numeric; L2 regularization parameter
intercept	logical; Whether an intercept should be included
x_center	logical; Should the features be centered?
scale	logical; Should the features be normalized? (default: FALSE)
y_scale	logical; whether the target vector should be centered

`...` Not used

`use_Xu_for_scaling`
logical; whether the unlabeled objects should be used to determine the mean and scaling for the normalization

References

Shaffer, J.P., 1991. The Gauss-Markov Theorem and Random Regressors. The American Statistician, 45(4), pp.269-273.

See Also

Other RSSL classifiers: `EMLeastSquaresClassifier`, `EMLinearDiscriminantClassifier`, `GRFClassifier`, `ICLeastSquaresClassifier`, `ICLinearDiscriminantClassifier`, `KernelLeastSquaresClassifier`, `LaplacianKernelLeastSquaresClassifier()`, `LaplacianSVM`, `LeastSquaresClassifier`, `LinearDiscriminantClass`
`LinearSVM`, `LinearTSVM()`, `LogisticLossClassifier`, `LogisticRegression`, `MCLinearDiscriminantClassifier`, `MCNearestMeanClassifier`, `MCPLDA`, `MajorityClassClassifier`, `NearestMeanClassifier`, `QuadraticDiscriminantCl`
`S4VM`, `SVM`, `SelfLearning`, `TSVM`, `WellSVM`, `svmlin()`

USMLeastSquaresClassifier-class *USMLeastSquaresClassifier*

Description

USMLeastSquaresClassifier

---

wdbc *wdbc data for unit testing*

Description

Useful for testing the S4VM and WellSVM implementations

WellSVM *WellSVM for Semi-supervised Learning*

Description

WellSVM is a minimax relaxation of the mixed integer programming problem of finding the optimal labels for the unlabeled data in the SVM objective function. This implementation is a translation of the Matlab implementation of Li (2013) into R.

Usage

```
WellSVM(X, y, X_u, C1 = 1, C2 = 0.1, gamma = 1, x_center = TRUE,
  scale = FALSE, use_Xu_for_scaling = FALSE, max_iter = 20)
```

Arguments

X	matrix; Design matrix for labeled data
y	factor or integer vector; Label vector
X_u	matrix; Design matrix for unlabeled data
C1	double; A regularization parameter for labeled data, default 1;
C2	double; A regularization parameter for unlabeled data, default 0.1;
gamma	double; Gaussian kernel parameter, i.e., k(x,y) = exp(-gamma^2||x-y||^2/avg) where avg is the average distance among instances; when gamma = 0, linear kernel is used. default gamma = 1;
x_center	logical; Should the features be centered?
scale	logical; Should the features be normalized? (default: FALSE)
use_Xu_for_scaling	logical; whether the unlabeled objects should be used to determine the mean and scaling for the normalization
max_iter	integer; Maximum number of iterations

References

Y.-F. Li, I. W. Tsang, J. T. Kwok, and Z.-H. Zhou. Scalable and Convex Weakly Labeled SVMs. Journal of Machine Learning Research, 2013.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research 6, 1889-1918, 2005.

See Also

Other RSSL classifiers: EMLeastSquaresClassifier, EMLinearDiscriminantClassifier, GRFClassifier, ICLeastSquaresClassifier, ICLinearDiscriminantClassifier, KernelLeastSquaresClassifier, LaplacianKernelLeastSquaresClassifier(), LaplacianSVM, LeastSquaresClassifier, LinearDiscriminantClass: LinearSVM, LinearTSVM(), LogisticLossClassifier, LogisticRegression, MCLinearDiscriminantClassifier, MCNearestMeanClassifier, MCPLDA, MajorityClassClassifier, NearestMeanClassifier, QuadraticDiscriminantCl S4VM, SVM, SelfLearning, TSVM, USMLeastSquaresClassifier, svmlin()

Examples

```
library(RSSL)
library(ggplot2)
library(dplyr)

set.seed(1)
df_orig <- generateSlicedCookie(200,expected=TRUE)
df <- df_orig %>%
  add_missinglabels_mar(Class~.,0.98)

classifiers <- list("Well"=WellSVM(Class~.,df,C1 = 1, C2=0.1,
                                    gamma = 0,x_center=TRUE,scale=TRUE),
                    "Sup"=SVM(Class~.,df,C=1,x_center=TRUE,scale=TRUE))

df %>%
  ggplot(aes(x=X1,y=X2,color=Class)) +
  geom_point() +
  coord_equal() +
  stat_classifier(aes(color=..classifier..),
                  classifiers = classifiers)
```

`wellsvm_direct` *wellsvm implements the wellsvm algorithm as shown in [1].*

Description

wellsvm implements the wellsvm algorithm as shown in [1].

Usage

```
wellsvm_direct(x, y, testx, testy, C1 = 1, C2 = 0.1, gamma = 1)
```

Arguments

`x`	A Nxd training data matrix, where N is the number of training instances and d is the dimension of instance;
`y`	A Nx1 training label vector, where y = 1/-1 means positive/negative, and y = 0 means unlabeled;
`testx`	A Mxd testing data matrix, where M is the number of testing instances;
`testy`	A Mx1 testing label vector
`C1`	A regularization parameter for labeled data, default 1;
`C2`	A regularization parameter for unlabeled data, default 0.1;
`gamma`	Gaussian kernel parameter, i.e., k(x,y) = exp(-gamma^2||x-y||^2/avg) where avg is the average distance among instances; when gamma = 0, linear kernel is used. default gamma = 1;

Value

prediction - A Mx1 predicted testing label vector; accuracy - The accuracy of prediction; cputime - cpu running time;

References

Y.-F. Li, I. W. Tsang, J. T. Kwok, and Z.-H. Zhou. Scalable and Convex Weakly Labeled SVMs. Journal of Machine Learning Research, 2013.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research 6, 1889-1918, 2005.

WellSVM_SSL *Convex relaxation of S3VM by label generation*

Description

Convex relaxation of S3VM by label generation

Usage

```
WellSVM_SSL(K0, y, opt, yinit = NULL)
```

Arguments

K0	kernel matrix
y	labels
opt	options
yinit	label initialization (not used)

WellSVM_supervised *A degenerated version of WellSVM where the labels are complete, that is, supervised learning*

Description

A degenerated version of WellSVM where the labels are complete, that is, supervised learning

Usage

```
WellSVM_supervised(K0, y, opt, ind_y)
```

Arguments

K0	kernel matrix
y	labels
opt	options
ind_y	Labeled/Unlabeled indicator

`wlda` *Implements weighted likelihood estimation for LDA*

Description

Implements weighted likelihood estimation for LDA

Usage

```
wlda(a, w)
```

Arguments

`a`	is the data set
`w`	is an indicator matrix for the K classes or, potentially, a weight matrix in which the fraction with which a sample belongs to a particular class is indicated

Value

m contains the means, p contains the class priors, iW contains the INVERTED within covariance matrix

`wlda_error` *Measures the expected error of the LDA model defined by m, p, and iW on the data set a, where weights w are potentially taken into account*

Description

Measures the expected error of the LDA model defined by m, p, and iW on the data set a, where weights w are potentially taken into account

Usage

```
wlda_error(m, p, iW, a, w)
```

Arguments

`m`	means
`p`	class prior
`iW`	is the inverse of the within covariance matrix
`a`	design matrix
`w`	weights

`wlda_loglik` *Measures the expected log-likelihood of the LDA model defined by m, p, and iW on the data set a, where weights w are potentially taken into account*

Description

Measures the expected log-likelihood of the LDA model defined by m, p, and iW on the data set a, where weights w are potentially taken into account

Usage

```
wlda_loglik(m, p, iW, a, w)
```

Arguments

m	means
p	class prior
iW	is the inverse of the within covariance matrix
a	design matrix
w	weights

Value

Average log likelihood

Index