Package ‘SSL’

May 14, 2016

Type Package
Title Semi-Supervised Learning
Version 0.1
Date 2016-05-01
Author Junxiang Wang
Maintainer Junxiang Wang <xianggebenben@163.com>
Description Semi-supervised learning has attracted the attention of machine learning community because of its high accuracy with less annotating effort compared with supervised learning. The question that semi-supervised learning wants to address is: given a relatively small labeled dataset and a large unlabeled dataset, how to design classification algorithms learning from both? This package is a collection of some classical semi-supervised learning algorithms in the last few decades.
License GPL (>= 3)
LazyData TRUE
RoxygenNote 5.0.1
Depends R (>= 3.2)
Imports NetPreProc (>= 1.1), Rcpp (>= 0.12.2), caret (>= 6.0-52), proxy (>= 0.4-15), xgboost (>= 0.4), klaR (>= 0.6-12), e1071 (>= 1.6-7), stats (>= 3.2)
LinkingTo Rcpp
NeedsCompilation yes
Repository CRAN
Date/Publication 2016-05-14 23:12:09

R topics documented:

  sslCoTrain .......................................................... 2
  sslGmmEM ........................................................... 3
  sslLabelProp ......................................................... 5
  sslLapRLS ............................................................ 6
  sslLDS ................................................................. 9
sslCoTrain

Description

Co-Training

Usage

sslCoTrain(xl, yl, xu, method1 = "nb", method2 = "nb", nrounds1, nrounds2, portion = 0.5, n = 10, seed = 0, ...)

Arguments

x1 a n * p matrix or data.frame of labeled data
yl a n * 1 integer vector of labels.
xu a m * p matrix or data.frame of unlabeled data
method1, method2 a string which specifies the first and second classification model to use. xgb means extreme gradient boosting, please refer to xgb.train. For other options, see more in train.
nrounds1, nrounds2 parameter needed when method1 or method2 = xgb. See more in xgb.train
portion the percentage of data to split into two parts.
n the number of unlabeled examples to add into label data in each iteration.
seed an integer specifying random number generation state for data split
... other parameters

Details

sslCoTrain divides labeled data into two parts, each part is trained with a classifier, then it chooses some unlabeled examples for prediction and adds them into labeled data. These new labeled data help the other classifier improve performance.

Value

a m * 1 integer vector representing the predictions of unlabeled data.
sslGmmEM

**Author(s)**

Junxiang Wang

**References**


**See Also**

train xgb.train

**Examples**

data(iris)
xl<-iris[,1:4]
# Suppose we know the first twenty observations of each class
# and we want to predict the remaining with co-training
# 1 setosa, 2 versicolor, 3 virginica
yl<-rep(1:3,each=20)
known.label <-c(1:20,51:70,101:120)
xu<-xl[-known.label,]
xl<-xl[known.label,]
yu<-sslCoTrain(xl,yl,xu,method1="xgb",nrounds1 = 100,method2="xgb",nrounds2 = 100,n=60)

---

**sslGmmEM**

*Gaussian Mixture Model with an EM Algorithm*

**Description**

sslGmmEM implements Gaussian Mixture Model with an EM algorithm, and weights the unlabeled data by introducing lambda-EM technique.

**Usage**

sslGmmEM(xl, yl, xu, seed = 0, improvement = 1e-04, p = 0.3)

**Arguments**

- **xl**: a n * p matrix or data.frame of labeled data
- **yl**: a n * 1 integer vector of labels.
- **xu**: a m * p matrix or data.frame of unlabeled data
- **seed**: an integer specifying random number generation state for splitting labeled data into training set and cross-validation set.
- **improvement**: numeric. Minimal allowed improvement of parameters.
- **p**: percentage of labeled data are splitted into cross-validation set.
Details

sslGmmEM introduces unlabeled data into parameter estimation process. The weight lambda is chosen by cross-validation. The Gaussian Mixture Model is estimated based on maximum log likelihood function with an EM algorithm. The E-step computes the probabilities of each class for every observation. The M-step computes parameters based on probabilities obtained in the E-step.

Value

a list of values is returned:

Fields

- para: a numeric estimated parameter matrix in which the column represents variables and the row represents estimated means and standard deviation of each class. For example, the first and second row represents the mean and standard deviation of the first class, the third and fourth row represents the mean and standard deviation of the second class, etc.
- classProb: the estimated class probabilities
- yu: the predicted label of unlabeled data
- optLambda: the optimal lambda chosen by cross-validation

Author(s)

Junxiang Wang

References

Kamal Nigam, Andrew McCallum, Sebastian Thrun, Tom Mitchell(1999) Text Classification from Labeled and Unlabeled Documents using EM

Examples

data(iris)
x1<-iris[,,-5]
#Suppose we know the first twenty observations of each class
#and we want to predict the remaining with Gaussian Mixture Model
#1 setosa, 2 versicolor, 3 virginica
yu<-rep(1:3,each=20)
known.label<-c(1:20,51:70,101:120)
xu<-x1[-known.label,]
x1<-x1[known.label,]
l<-sslGmmEM(x1,y1,xu)
sslLabelProp

Label Propagation

Description

sslLabelProp propagates a few known labels to a large number of unknown labels according to their proximities to neighboring nodes. It supports many kinds of distance measurements and graph representations.

Usage

sslLabelProp(x, y, known.label, graph.type = "exp", dist.type = "Euclidean", alpha, alpha1, alpha2, k, epsilon, iter = 1000)

Arguments

- **x**
a n * p matrix or data.frame of n observations and p predictors

- **y**
a vector of k known labels. The rows of y must be the same as the length of known.label.

- **known.label**
a vector indicating the row index of known labels in matrix x.

- **graph.type**
character string; which type of graph should be created? Options include knn, enn, tanh and exp.

  - **knn** : kNN graphs. Nodes i, j are connected by an edge if i is in j’s k-nearest-neighborhood. k is a hyperparameter that controls the density of the graph.

  - **enn** : epsilon-NN graphs. Nodes i, j are connected by an edge if the distance d(i, j) < epsilon. The hyperparameter epsilon controls neighborhood radius.

  - **tanh** : tanh-weighted graphs. w(i, j) = (tanh(alpha1(d(i, j) - alpha2))) + 1/2. where d(i, j) denotes the distance between point i and j. Hyperparameters alpha1 and alpha2 control the slope and cutoff value respectively.

  - **exp** : exp-weighted graphs. w(i, j) = exp(-d(i, j)^2/alpha^2),where d(i, j) denotes the distance between point i and j. Hyperparameter alpha controls the decay rate.

- **dist.type**
character string; this parameter controls the type of distance measurement. (see dist or pr_DB).

- **alpha**
umeric parameter needed when graph.type = exp

- **alpha1**
umeric parameter needed when graph.type = tanh

- **alpha2**
umeric parameter needed when graph.type = tanh

- **k**
integer parameter needed when graph.type = knn

- **epsilon**
umeric parameter needed when graph.type = enn

- **iter**
iteration
Details

sslLabelProp implements label propagation algorithm in iter iterations. It supports many kinds of distance measurements and four types of graph creations.

Value

A n * 1 vector indicating the predictions of n observations in C class

Author(s)

Junxiang Wang

References

Xiaojin Zhu(2005), Semi-Supervised Learning with Graphs

See Also

dist, pr_DB

Examples

data(iris)
x<-iris[,1:4]
# Suppose we know the first twenty observations of each class and we want to propagate
# these labels to unlabeled data.
# 1 setosa, 2 versicolor, 3 virginica
y<-rep(1:3,each =20)
known.label <-c(1:20,51:70,101:120)
f1<-sslLabelProp(x,y,known.label,graph.type="enn",epsilon = 0.5)
f2<-sslLabelProp(x,y,known.label,graph.type="knn",k =10)
f3<-sslLabelProp(x,y,known.label,graph.type="tanh",alpha1=-2, alpha2=1)
f4<-sslLabelProp(x,y,known.label,graph.type="exp",alpha = 1)

----------

sslLapRLS

Laplacian Regularized Least Squares

Description

Laplacian Regularized Least Squares

Usage

sslLapRLS(xl, yl, xu, graph.type = "exp", dist.type = "Euclidean", alpha, alpha1, alpha2, k, epsilon, kernel = "gaussian", c1, c2, c3, deg, gamma, alpha3, alpha4, gammaA = 1, gammaI = 1)
sslLapRLS

Arguments

- **x1**: a \( n \times p \) matrix or data.frame of labeled data.
- **y1**: a \( n \times 1 \) binary labels (1 or -1).
- **xu**: a \( m \times p \) matrix or data.frame of unlabeled data.
- **graph.type**: character string; which type of graph should be created? Options include knn, enn, tanh and exp.
  - knn: \( k \)-NN graphs. Nodes \( i, j \) are connected by an edge if \( i \) is in \( j \)'s \( k \)-nearest-neighborhood. \( k \) is a hyperparameter that controls the density of the graph.
  - enn: \( \epsilon \)-NN graphs. Nodes \( i, j \) are connected by an edge, if the distance \( d(i, j) < \epsilon \). The hyperparameter \( \epsilon \) controls neighborhood radius.
  - tanh: \( \tanh \)-weighted graphs. \( w(i, j) = (\tanh(\alpha_1(d(i, j) - \alpha_2)) + 1)/2 \). where \( d(i, j) \) denotes the distance between point \( i \) and \( j \). Hyperparameters \( \alpha_1 \) and \( \alpha_2 \) control the slope and cutoff value respectively.
  - exp: \( \exp \)-weighted graphs. \( w(i, j) = \exp(-d(i, j)^2/\alpha^2) \). where \( d(i, j) \) denotes the distance between point \( i \) and \( j \). Hyperparameter \( \alpha \) controls the decay rate.
- **dist.type**: character string; this parameter controls the type of distance measurement. (see `dist` or `pr_DB`).
- **alpha**: numeric parameter needed when graph.type = exp
- **alpha1**: numeric parameter needed when graph.type = tanh
- **alpha2**: numeric parameter needed when graph.type = tanh
- **k**: integer parameter needed when graph.type = knn
- **epsilon**: numeric parameter needed when graph.type = enn
- **kernel**: character string; it controls four types of common kernel functions: linear, polynomial, gaussian and sigmoid.
  - linear: Linear kernel; \( k(x, y) = \text{dot}(x, y) + c_1 \). where \( \text{dot}(x, y) \) is the dot product of vector \( x \) and \( y \). \( c_1 \) is a constant term.
  - polynomial: Polynomial kernel; \( k(x, y) = (\alpha_3 \times \text{dot}(x, y) + c_2)^\text{deg} \). where \( \text{dot}(x, y) \) is the dot product of vector \( x \) and \( y \). Adjustable parameters are the slope \( \alpha_3 \), the constant term \( c_2 \) and the polynomial degree \( \text{deg} \).
  - gaussian: Gaussian kernel; \( k(x, y) = \exp(-\gamma d(x, y)^2) \). where \( d(x, y) \) is Euclidean distance between vector \( x \) and \( y \). \( \gamma \) is a slope parameter.
  - sigmoid: Hyperbolic Tangent (Sigmoid) Kernel; \( k(x, y) = \tanh(\alpha_4 \times \text{dot}(x, y) + c_3) \). where \( \text{dot}(x, y) \) is dot product of vector \( x \) and \( y \). There are two adjustable parameters in the sigmoid kernel, the slope \( \alpha_4 \) and the intercept constant \( c_3 \).
- **c1**: numeric parameter needed when kernel = linear
- **c2**: numeric parameter needed when kernel = polynomial
- **c3**: numeric parameter needed when kernel = sigmoid
- **deg**: integer parameter needed when kernel = sigmoid
- **gamma**: numeric parameter needed when kernel = polynomial
- **alpha3**: numeric parameter needed when kernel = polynomial
alphaA numeric parameter needed when kernel = sigmoid

gammaA numeric; model parameter.

gammaI numeric; model parameter.

Value

A m * 1 integer vector representing the predicted labels of unlabeled data(1 or -1).

Author(s)

Junxiang Wang

References


See Also

pr_db dist

Examples

data(iris)
x1<-iris[c(1:20,51:70),,5]
xu<-iris[c(21:50,71:100),,5]
y1<-rep(c(1,-1),each=20)
# combinations of different graph types and kernel types
# graph.type = knn, kernel = linear
yu1<-sslLapRLS(x1,y1,xu,graph.type="knn",k=10,kernel="linear",c1=1)
# graph.type = knn, kernel = polynomial
yu2<-sslLapRLS(x1,y1,xu,graph.type="knn",k=10,kernel="polynomial",c2=1,deg=2,alpha3=1)
# graph.type = knn, kernel = gaussian
yu3<-sslLapRLS(x1,y1,xu,graph.type="knn",k=10,kernel="gaussian",gamma=1)
# graph.type = knn, kernel = sigmoid
yu4<-sslLapRLS(x1,y1,xu,graph.type="knn",k=10,kernel="sigmoid",c3=-10, alpha4=0.001,gammaI = 0.05,gammaA = 0.05)
# graph.type = enn, kernel = linear
yu5<-sslLapRLS(x1,y1,xu,graph.type="enn",epsilon=1,kernel="linear",c1=1)
# graph.type = enn, kernel = polynomial
yu6<-sslLapRLS(x1,y1,xu,graph.type="enn",epsilon=1,kernel="polynomial",c2=1,deg=2,alpha3=1)
# graph.type = enn, kernel = gaussian
yu7<-sslLapRLS(x1,y1,xu,graph.type="enn",epsilon=1,kernel="gaussian",gamma=1)
# graph.type = enn, kernel = sigmoid
yu8<-sslLapRLS(x1,y1,xu,graph.type="enn",epsilon=1,kernel="sigmoid",c3=-10, alpha4=0.001,gammaI = 0.05,gammaA = 0.05)
# graph.type = tanh, kernel = linear
yu9<-sslLapRLS(x1,y1,xu,graph.type="tanh",alpha1=-2,alpha2=1,kernel="linear",c1=1)
# graph.type = tanh, kernel = polynomial
yu10<-sslLapRLS(x1,y1,xu,graph.type="tanh",alpha1=-2,alpha2=1, kernel="polynomial",c2=1,deg=2,alpha3=1)
sslLDS

**Low Density Separation**

**Description**

sslLDS implements low density separation with Transductive Support Vector Machines (TSVM) for semi-supervised binary classification.

**Usage**

```r
sslLDS(xl, yl, xu, rho = 1, C = 1, dist.type = "Euclidean", p = 0.3,
        improvement = 1e-04, seed = 0, delta = 0.01, alpha = 0.01)
```

**Arguments**

- `xl` a n * p matrix or data.frame of labeled data.
- `yl` a n * 1 binary labels (1 or -1).
- `xu` a m * p matrix or data.frame of unlabeled data.
- `rho` numeric; a parameter for connectivity kernel. It defines minimal rho-path distances.
- `C` numeric; a parameter in the TSVM training model.
- `dist.type` character string; this parameter controls the type of distance measurement. (see `dist` or `pr_DB`).
- `p` the percentage of data used for cross-validation set.
- `improvement` numeric; minimal allowed improvement of parameters.
- `seed` an integer specifying random number generation state for splitting labeled data into training set and cross-validation set.
- `delta` numeric; a allowed cutoff for the cumulative percent of variance to lose by multidimensional scaling.
- `alpha` numeric; a learning rate in the gradient descent algorithm.
Details

sslLDS constructs a low density graph with connectivity kernel. It implements multidimensional scaling for dimensionality reduction and chooses optimal C.star by cross-validation. Finally, it trains the TSVM model with gradient descent algorithm.

Value

a list of values is returned:

Fields

yu the predicted label of unlabeled data
optC.star the optimal C.star chosen by cross-validation. C.star weights the unlabeled data in the TSVM model.
para estimated parameters of TSVM, including w and b

Author(s)

Junxiang Wang

References


Examples

data(iris)
x1< iris[c(1:20,51:70),-5]
xu< iris[c(21:50,71:100),-5]
yl< rep(c(1,-1),each=20)
l< sslLDS(x1,yl,xu,alpha=0.1)

ssILLGC  

Local and Global Consistency

Description

Local and Global Consistency

Usage

ssILLGC(x1, yl, xu, dist.type = "Euclidean", alpha = 0.01, gamma = 1, iter = 10000)
Arguments

\( \text{x1} \)  
\( \text{a n * p matrix or data.frame of labeled data} \)

\( \text{yl} \)  
\( \text{a n X C matrix representing labels of n observations in C classes. If observation i belongs to class j, then yl(i,j)=1, and other elements in the same row equal 0.} \)

\( \text{xu} \)  
\( \text{a m * p matrix or data.frame of unlabeled data.} \)

\( \text{dist.type} \)  
\( \text{character string; this parameter controls the type of distance measurement.(see dist or pr_DB).} \)

\( \text{alpha} \)  
\( \text{a numeric parameter controls convergence rate.} \)

\( \text{gamma} \)  
\( \text{a numeric parameter in the affinity matrix} \)

\( \text{iter} \)  
\( \text{the number of iteration.} \)

Value

\( \text{a m * 1 integer vector representing the predicted labels of unlabeled data.} \)

Author(s)

Junxiang Wang

References


See Also

pr_DB dist

Examples

data(iris)
x1<-iris[c(1:20,51:70,101:120),-5]
yl<-matrix(0,ncol=3,nrow=60)
yl[1:20,1]<-1
yl[21:40,2]<-1
yl[41:60,3]<-1
xu<-iris[-c(1:20,51:70,101:120),-5]
yu<-ssllgc(xl,yl,xu)
sslMarkovRandomWalks  

\textit{t-step Markov Random Walks}

\textbf{Description}

\textit{t-step Markov Random Walks}

\textbf{Usage}

\begin{verbatim}
sslMarkovRandomWalks(xl, yl, xu, t = 10, dist.type = "Euclidean", k = 10,
gamma = 1, improvement = 1e-04)
\end{verbatim}

\textbf{Arguments}

- \textit{xl}  
a \textit{n} * \textit{p} matrix or data.frame of labeled data.
- \textit{yl}  
a \textit{n} * 1 binary labels (1 or -1).
- \textit{xu}  
a \textit{m} * \textit{p} matrix or data.frame of unlabeled data.
- \textit{t}  
step size.
- \textit{dist.type}  
character string; this parameter controls the type of distance measurement. (see \texttt{dist} or \texttt{pr_DB}).
- \textit{k}  
an integer parameter controls a k-nearest neighbor graph.
- \textit{gamma}  
a numeric parameter in the affinity matrix.
- \textit{improvement}  
numeric. Maximum allowed distance between computed parameters in two successive iterations at the steady state.

\textbf{Details}

\textit{sslMarkovRandomWalks} transmits known labels to unlabeled data by \textit{t-step Markov random walks}. Parameters are estimated by an EM algorithm.

\textbf{Value}

- a \textit{m} * 1 integer vector representing the predicted labels of unlabeled data.

\textbf{Author(s)}

Junxiang Wang

\textbf{References}


\textbf{See Also}

\texttt{pr_DB dist}
sslMincut

**Examples**

```r
data(iris)
xl <- iris[1:20, 51:70, -5]
xu <- iris[21:50, 71:100, -5]
yl <- rep(c(1, -1), each=20)
yu <- sslMarkovRandomWalks(xl, yl, xu)
```

**sslMincut**

**Mincut**

**Description**

sslMincut implements the Mincut algorithm for maxflow graph partition in the k-nearest neighbor graph.

**Usage**

```r
sslMincut(xl, yl, xu, simil.type = "correlation", k = 10)
```

**Arguments**

- `xl`:
  a n * p matrix or data.frame of labeled data
- `yl`:
  a n * 1 binary labels (1 or -1).
- `xu`:
  a m * p matrix or data.frame of unlabeled data.
- `simil.type`:
  character string; this parameter controls the type of similarity measurement (see `simil` or `pr_DB`).
- `k`:
  an integer parameter controls a k-nearest neighbor graph.

**Details**

sslMincut creates a k-nearest neighbor graph and finds a maxflow from the first positive observation to the first negative one based on MPLA algorithm. This maxflow partitions the graph into positive labels and negative ones.

**Value**

a m * 1 integer vector representing the predicted labels of unlabeled data.

**Author(s)**

Junxiang Wang

**References**

sslRegress

**Description**

sslRegress develops a regularization framework on graphs. It supports many kinds of distance measurements and graph representations. However, it only supports binary classifications.

**Usage**

```
sslRegress(xl, yl, xu, graph.type = "exp", dist.type = "Euclidean", alpha, alpha1, alpha2, p = 2, method = "Tikhonov", gamma = 1)
```

**Arguments**

- `xl`: a $n \times p$ matrix or data.frame of labeled data.
- `yl`: a $n \times 1$ binary labels (1 or -1).
- `xu`: a $m \times p$ matrix or data.frame of unlabeled data.
- `graph.type`: character string; which type of graph should be created? Options include `tanh` and `exp`.
  - `tanh`: tanh-weighted graphs. $w(i,j) = \frac{(\tanh(\alpha_1(d(i,j) - \alpha_2)) + 1)/2$ where $d(i,j)$ denotes the distance between point $i$ and $j$. Hyperparameters $\alpha_1$ and $\alpha_2$ control the slope and cutoff value respectively.
  - `exp`: exp-weighted graphs. $w(i,j) = \exp(-\frac{d(i,j)^2}{\alpha^2})$. where $d(i,j)$ denotes the distance between point $i$ and $j$. Hyperparameter $\alpha$ controls the decay rate.
- `dist.type`: character string; this parameter controls the type of distance measurement. (see `dist` or `pr_DB`).
- `alpha`: numeric parameter needed when `graph.type = exp`.
- `alpha1`: numeric parameter needed when `graph.type = tanh`.
- `alpha2`: numeric parameter needed when `graph.type = tanh`.
- `p`: an integer parameter controls the power of Laplacian for regularization.
- `method`: character string; this parameter choose two possible algorithms: "Tikhonov" means Tikhonov regularization; "Interpolated" means Interpolated regularization.
- `gamma`: a parameter of Tikhonov regularization.

**Examples**

```r
data(iris)
xl <- iris[c(1:20, 51:70), -5]
xu <- iris[c(21:50, 71:100), -5]
yl <- rep(c(1, -1), each = 20)
yu <- sslMinCut(xl, yl, xu)
sslRegress(xl, yl, xu, graph.type = "exp", dist.type = "Euclidean", alpha, alpha1, alpha2, p = 2, method = "Tikhonov", gamma = 1)
```

See Also

`pr_DB simil`
Self-Training

Usage

sslSelfTrain(xl, yl, xu, n = 10, nrounds, ...)

Arguments

xl a n * p matrix or data.frame of labeled data
yl a n * 1 integer vector of labels (begin from 1).
ux a m * p matrix or data.frame of unlabeled data
n number of unlabeled examples to add into labeled data in each iteration
nrounds the maximal number of iterations, see more in xgb.train
... other parameters
Details

In self-training a classifier is first trained with the small amount of labeled data using extreme gradient boosting. The classifier is then used to classify the unlabeled data. The most confident unlabeled points, together with their predicted labels, are added to the training set. The classifier is re-trained and the procedure repeats.

Value

a m * 1 integer vector representing the predictions of unlabeled data.

Author(s)

Junxiang Wang

References


See Also

xgb.train

Examples

data(iris)
xl<-iris[,1:4]
#Suppose we know the first twenty observations of each class
#and we want to predict the remaining with self-training
# 1 setosa, 2 versicolor, 3 virginica
yl<-rep(1:3,each = 20)
known.label <-c(1:20,51:70,101:120)
xu<-xl[-known.label,]
x<-xl[known.label,]
yu<-sslSelfTrain(xl,yl,xu,nrounds = 100,n=30)
Index

dist, 5–9, 11, 12, 14, 15

pr_DB, 5–9, 11–15

simil, 13, 14
sslCoTrain, 2
sslGmmEM, 3
sslLabelProp, 5
sslLapRLS, 6
sslLDS, 9
sslLLGC, 10
sslMarkovRandomWalks, 12
sslMincut, 13
sslRegress, 14
sslSelfTrain, 15

train, 2, 3

xgb.train, 2, 3, 15, 16