# OpenPR Manual

# OpenPR Manual

# Table of Contents

I. OpenPR Manual ..... 1
aggloms ..... 2
ahclustering ..... 4
backward_selection ..... 6
balanced_winnow ..... 7
buildcart ..... 8
combinations ..... 9
competitive_learning ..... 10
confmatrix2ni_ce ..... 12
confmatrix2ni_id ..... 14
confmatrix2ni_mi ..... 16
createkernel ..... 18
emparams ..... 19
empredict ..... 20
emtrain ..... 21
exhaustive_selection ..... 23
fda ..... 24
forward_selection ..... 25
fuzzy_kmeans ..... 26
generalizedda ..... 27
hdr ..... 28
hmm_backward ..... 29
hmm_baum ..... 31
hmm_forward ..... 32
hmm_viterbi ..... 34
ica ..... 35
kerneldensitycovering ..... 36
kfda ..... 37
kmeans ..... 38
kmeans_sci ..... 39
kmedoids ..... 40
knearest ..... 41
knearest_sci ..... 42
kpca ..... 43
lbg_vq ..... 44
mlda ..... 45
leader_follower ..... 46
least_squares ..... 47
lms ..... 48
mds ..... 49
ml_gaussian ..... 50
msclustering ..... 51
mseclustering ..... 52
nbayespredict ..... 53
nbayestrain ..... 54
parzen_classify ..... 55
pca2 ..... 56
perceptron_batch ..... 57
perceptron_bvi ..... 58
perceptron_fis ..... 59
perceptron_vim ..... 60
plsiread ..... 61
plsitrain ..... 62
pnn_classify ..... 63
pnn_train ..... 64

qdmatch .......... 65
randperm .......... 66
rce_classify .......... 67
rce_train .......... 68
readsparse .......... 69
relaxation_brm .......... 70
relaxation_srm .......... 71
sohclustering .......... 72
sphereconstruction .......... 73
svmpredict .......... 74
svmtrain .......... 76
twodpca .......... 79
usecart .......... 80
whitening .......... 81
wpca .......... 82

# Part I. OpenPR Manual

# Name

aggloms — Agglomerative Mean-Shift Clustering

```
[cluster_centers, cluster_id] = aggloms(Data, sigma, ite_num)
```

# Parameters

Data
: Data matrix. Each column vector is a data point.

sigma
: Bandwidth of Gaussian kernel.

ite_num
: Number of iterations.

cluster_centers
: Cluster center matrix. Each column vector is a cluster center point.

cluster_id
: Cluster index vector.

# Description

Mean-Shift (MS) is a powerful non-parametric clustering method. Although good accuracy can be achieved, its computational cost is particularly expensive even on moderate data sets. This function uses an agglomerative MS clustering method called Agglo-MS, along with its mode-seeking ability and convergence property analysis, for the purpose of algorithm speedup. The method is built upon an iterative query set compression mechanism which is motivated by the quadratic bounding optimization nature of MS. The whole framework can be efficiently implemented in linear running time complexity.

# Examples

```
//This is an example program to show the performance of our Agglomerative Mean-S

Data = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/aggloms_data',2,1810);
Label = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/aggloms_label',1,1810);

sigma = 0.4; // kernel bandwidth
ite_num = 60; // iteration times

[cluster_centers, cluster_id]=aggloms(Data', sigma, ite_num);

//-------------------- draw clusters --------------------------------

cluster_count = size(cluster_centers,1);
for (i=1:cluster_count)
    my_color = rand(1,3);
    plot(Data(1,find(cluster_id==i)), Data(2,find(cluster_id==i)),'o','marker',
end
plot(cluster_centers(:,1), cluster_centers(:,2), 'gs','marker','o','markersize'
```

# Author

Xiao-Tong Yuan <xtyuan@nlpr.ia.ac.cn>

# Bibliography

Xiao-Tong Yuan, Bao-Gang Hu and Ran He, Agglomerative Mean-Shift Clustering via Query Set Compression, SDM 2009

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

kerneldensitycovering, sphereconstruction

## Name

ahclustering — agglomerative hierarchical clustering

```
[centers, labels] = ahclustering(train_samples, cluster_num, dist_type)
```

## Parameters

train_samples
: data matrix of size dim*num; each column is a data point

cluster_num
: number of desired clusters

dist_type
: type of distance function with default value 'min'; it could be 'min', 'max', 'avg' or 'mean'

centers
: centers of the formed clusters

labels
: labels of each trainning sample belonging to the formed clusters

## Description

The function implements the bottom-up agglomerative hierarchical clustering. The algorithm starts with every training sample being a singleton cluster, then it iteratively merges the nearest clusters until the desired number of clusters are formed. The distance between two clusters are measured by a distance function.

## Examples

```
samples = rand(2,30);
cluster_num = 3;
dist_type = 'avg';
[centers, labels] = ahclustering(samples, cluster_num, dist_type);
//show figures
scf(0);
plot(samples(1,:), samples(2,:), 'b.', 'MarkerSize', 3);
scf(1);
clusters = unique(labels);
plot(samples(1,find(labels==clusters(1))),samples(2,find(labels==clusters(1))),
set(gca(),"auto_clear","off");
plot(samples(1,find(labels==clusters(2))),samples(2,find(labels==clusters(2))),
set(gca(),"auto_clear","off");
plot(samples(1,find(labels==clusters(3))),samples(2,find(labels==clusters(3))),
set(gca(),"auto_clear","off");
plot(centers(1,:), centers(2,:), 'r.', 'MarkerSize', 4);
```

## Authors

Jia Wu <jiawu83@gmail.com>

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

## See Also

sohclustering

# Name

backward_selection — backward feature selection

```
[new_samples, feature_idx] = backward_selection(samples, labels, final_dim, fol
```

# Parameters

samples
: dim*num data matrix; each column is a data point, each row is a feature

labels
: a 1*num vector of labels for each data point

final_dim
: the number of output dimension after feature selection

fold
: number of folds for cross validation

new_samples
: final_dim*num data matrix after feature selection

feature_idx
: selected feature indices

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

forward_selection, exhaustive_selection

# Name

balanced_winnow — Balanced Winnow Algorithm (for two-category cases)

```
[a_plus, a_minus, test_labels] = balanced_winnow(train_samples, train_labels, p
```

# Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations, the convergence rate and the promotion parameter: [iterm,eta,alpha]; The default value is [1000,0.01,2].

test_samples
: test data matrix of size dim*num_te; each column is a data point

a_plus
: positive weight vector

a_minus
: negative weight vector

test_labels
: predicted labels for the test samples

# Description

The function finds positive weight vector and negative weight vector using balanced Winnow algorithm.

# Authors

Jia WU <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# Name

buildcart — Create a classification and regression tree (CART).

```
cart = buildcart(train_samples, train_labels, impurity_type)
```

# Parameters

train_samples
: training-sample matrix of size dim*num; each column is a sample point

train_labels
: class labels of the input training samples

impurity_type
: impurity type for splitting; it can be Entropy, Gini, or Misclassification

cart
: the trained classification and regression tree; it is a struct variable

# Examples

```
train_samples=[0.15 0.09 0.29 0.38 0.52 0.57 0.73 0.47 0.1  0.08 0.23 0.7  0.62
               0.83 0.55 0.35 0.7  0.48 0.73 0.75 0.06 0.29 0.15 0.16 0.19 0.47
train_labels=[1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0];
impurity_type='Gini';
cart=buildcart(train_samples,train_labels,impurity_type)
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

usecart

# Name

combinations — find all the combinations of given number of indices

```
comb_mat = combinations(indices, number)
```

# Parameters

indices
: input indices and should be a vector

number
: the required number of indices

comb_mat
: all the combinations of the required number of indices; the size is comb_num*number, comb_num is the number of all possible combinations, and num is the required number of indices

# Description

The function is to find all the combinations of a sequence given a number of elements.

# Examples

```
comb_mat = combinations([1 4 6 2 9 8 3 5], 3)
```

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

competitive_learning — competitve learning clustering

```
[centers, labels, W] = competitive_learning(train_samples, c, eta, alpha, max_i
```

# Parameters

train_samples
: data matrix of size dim*num; each column is a data point

c
: number of clusters

eta
: learning rate with default value 0.01

alpha
: decay coefficient of learning rate with default value 0.99

max_iter
: maximal number of iterations with default value 1000

eps
: threshold for change in weight vector with default value 1e-5

centers
: cluster centers

labels
: cluster indices for each training sample point

W
: weight vectors

# Description

The function implements competitive learning clustering. Cluster center adjustment is confined to the single cluster center most similar to the given training pattern.

# Examples

```
samples = [rand(2,10), -1*rand(2,10)];
c = 2;
[centers, labels, W] = competitive_learning(samples, c);
scf(1);
plot(samples(1,:), samples(2,:), 'b.', 'MarkerSize', 3);
scf(2);
style = ['g.', 'c.', 'y.', 'k.', 'm.'];
ul = unique(labels)
for i = 1:length(ul),
  plot(samples(1,find(labels==ul(i))), samples(2,find(labels==ul(i))), style(i)
  set(gca(), "auto_clear", "off");
end
plot(centers(1,:), centers(2,:), 'r.', 'MarkerSize', 4);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

kmeans_sci, fuzzy_kmeans, leader_follower

# Name

confmatrix2ni_ce — Calculate normalized mutual information based on cross entropy definition.

```
[NI,A,Rej,P,R]=confmatrix2ni_ce(c)
```

# Parameters

c

Confusion matrix in size of m by (m+1): row for exact labels, column for prediction labels, the (m+1)th column for rejection (or unknown) class. This matrix has to follow the constraints: c_ij >=0, and C_i>0 (the ith class number).

NI

Normalized Information listed from NI_21 to NI_24. NI_i= inf standing for singularity result.

A

Accuracy.

Rej

Rejection.

P

Precision for a binary classifier.

R

Recall for a binary classifier.

# Description

The function calculates Normalized Mutual Information from a given m by (m+1) confusion matrix for evaluating a classifier. All NIs are calculated base on cross entropy definition.

# Examples

```
//    Numerical examples in the reference (cirfirmed on all, but need change on
//    Examples of binary classification, Table 4
M1=[90   0   0 ; 1   9   0];
M2=[89   1   0 ; 0  10   0];
M3=[90   0   0 ; 0   9   1];
M4=[89   0   1 ; 0  10   0];
M5=[57  38   0 ; 3   2   0];
M6=[89   1   0 ; 1   9   0];
//    Examples of three-class classification, Table 7
M7 =[80  0  0  0; 0  15  0  0; 1  0  4  0 ];
M8 =[80  0  0  0; 0  15  0  0; 0  1  4  0 ];
M9 =[80  0  0  0; 0  15  0  0; 0  0  4  1 ];
M10=[80  0  0  0; 1  14  0  0; 0  0  5  0 ];
M11=[80  0  0  0; 0  14  1  0; 0  0  5  0 ];
M12=[80  0  0  0; 0  14  0  1; 0  0  5  0 ];
M13=[79  1  0  0; 0  15  0  0; 0  0  5  0 ];
M14=[79  0  1  0; 0  15  0  0; 0  0  5  0 ];
M15=[79  0  0  1; 0  15  0  0; 0  0  5  0 ];
c=M6;
format('v',6);
[NI,A,Rej,P,R]=confmatrix2ni_ce(c)
```

# Authors

Baogang Hu <hubg@nlpr.ia.ac.cn>

# Bibliography

Hu, B.-G., He, R., and Yuan, X.-T., Information-Theoretic Measures for Objective Evaluation of Classifiers, submitted to a journal (2009)

Hu, B.-G., Information Measure Toolbox for Classifier Evaluation on Open Source Software Scilab, submitted to OSSC-2009.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

confmatrix2ni_id, confmatrix2ni_mi

# Name

confmatrix2ni_id — Calculate normalized mutual information based on information divergence definition.

```
[NI,A,Rej,P,R]=confmatrix2ni_id(c)
```

# Parameters

c
: Confusion matrix in size of m by (m+1): row for exact labels, column for prediction labels, the (m+1)th column for rejection (or unknown) class. This matrix has to follow the constraints: c_ij >=0, and C_i>0 (the ith class number).

NI
: Normalized Information listed from NI_10 to NI_20. NI_i= inf standing for singularity result.

A
: Accuracy.

Rej
: Rejection.

P
: Precision for a binary classifier.

R
: Recall for a binary classifier.

# Description

The function calculates Normalized Mutual Information from a given m by (m+1) confusion matrix for evaluating a classifier. All NIs are calculated base on information divergence definition.

# Examples

```
//    Numerical examples in the reference (cirfirmed on all, but need change on
//    Examples of binary classification, Table 4
M1=[90   0   0 ; 1   9   0];
M2=[89   1   0 ; 0  10   0];
M3=[90   0   0 ; 0   9   1];
M4=[89   0   1 ; 0  10   0];
M5=[57  38   0 ; 3   2   0];
M6=[89   1   0 ; 1   9   0];
//    Examples of three-class classification, Table 7
M7 =[80  0  0  0; 0  15  0  0; 1  0  4  0 ];
M8 =[80  0  0  0; 0  15  0  0; 0  1  4  0 ];
M9 =[80  0  0  0; 0  15  0  0; 0  0  4  1 ];
M10=[80  0  0  0; 1  14  0  0; 0  0  5  0 ];
M11=[80  0  0  0; 0  14  1  0; 0  0  5  0 ];
M12=[80  0  0  0; 0  14  0  1; 0  0  5  0 ];
M13=[79  1  0  0; 0  15  0  0; 0  0  5  0 ];
M14=[79  0  1  0; 0  15  0  0; 0  0  5  0 ];
M15=[79  0  0  1; 0  15  0  0; 0  0  5  0 ];
c=M1;
format('v',7);
[NI,A,Rej,P,R]=confmatrix2ni_id(c)
```

# Authors

Baogang Hu <hubg@nlpr.ia.ac.cn>

# Bibliography

Hu, B.-G., He, R., and Yuan, X.-T., Information-Theoretic Measures for Objective Evaluation of Classifiers, submitted to a journal (2009)

Hu, B.-G., Information Measure Toolbox for Classifier Evaluation on Open Source Software Scilab, submitted to OSSC-2009.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

confmatrix2ni_ce, confmatrix2ni_mi

# Name

confmatrix2ni_mi — Calculate normalized mutual information based on mutual informtin definition.

```
[NI,A,Rej,P,R]=confmatrix2ni_mi(c)
```

# Parameters

c
: Confusion matrix in size of m by (m+1): row for exact labels, column for prediction labels, the (m+1)th column for rejection (or unknown) class. This matrix has to follow the constraints: c_ij >=0, and C_i>0 (the ith class number).

NI
: Normalized Information listed from NI_1 to NI_9. NI_i= inf standing for singularity result.

A
: Accuracy.

Rej
: Rejection.

P
: Precision for a binary classifier.

R
: Recall for a binary classifier.

# Description

The function calculates Normalized Mutual Information from a given m by (m+1) confusion matrix for evaluating a classifier. All NIs are calculated base on mutual informtin definition.

# Examples

```
//    Numerical examples in the reference (cirfirmed on all)
//    Examples of binary classification, Table 4 in Ref 1
M1=[90   0   0 ; 1   9   0];
M2=[89   1   0 ; 0  10   0];
M3=[90   0   0 ; 0   9   1];
M4=[89   0   1 ; 0  10   0];
M5=[57  38   0 ; 3   2   0];
M6=[89   1   0 ; 1   9   0];
//    Examples of three-class classification, Table 7 in Ref 1
M7 =[80  0  0  0; 0  15  0  0; 1  0  4  0 ];
M8 =[80  0  0  0; 0  15  0  0; 0  1  4  0 ];
M9 =[80  0  0  0; 0  15  0  0; 0  0  4  1 ];
M10=[80  0  0  0; 1  14  0  0; 0  0  5  0 ];
M11=[80  0  0  0; 0  14  1  0; 0  0  5  0 ];
M12=[80  0  0  0; 0  14  0  1; 0  0  5  0 ];
M13=[79  1  0  0; 0  15  0  0; 0  0  5  0 ];
M14=[79  0  1  0; 0  15  0  0; 0  0  5  0 ];
M15=[79  0  0  1; 0  15  0  0; 0  0  5  0 ];
c=M1;
format('v',6);
[NI,A,Rej,P,R]=confmatrix2ni_mi(c)
```

# Authors

Baogang Hu <hubg@nlpr.ia.ac.cn>

# Bibliography

Hu, B.-G., He, R., and Yuan, X.-T., Information-Theoretic Measures for Objective Evaluation of Classifiers, submitted to a journal (2009)

Hu, B.-G., Information Measure Toolbox for Classifier Evaluation on Open Source Software Scilab, submitted to OSSC-2009.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

confmatrix2ni_id, confmatrix2ni_ce

# Name

createkernel — Kernel Function.

```
K = createkernel(x, y, param)
```

# Parameters

x
: dim*numx data matrix. Each column is a data point.

y
: dim*numy data matrix. Each column is a data point.

param
: A struct variable with the following fields:

- `typ`

  Gaussian: exp(-|x-y|^2/2t^2)

  Polynomial: (c*x'*y+r)^d

  Linear: x'*y

  Sigmoid: tanh(c*x'*y+r)

- `t` - kernel parameter
- `c` - kernel parameter
- `r` - kernel parameter
- `d` - kernel parameter

K
: numx*numy kernel matrix.

# Examples

```
x = rand(2,10);
y = rand(2,15);
param = struct('typ', 'Gaussian', 't', 1);
K = createkernel(x, y, param);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

emparams — Create a struct variable containing parameter values for EM algorithm.

```
params = emparams(nclusters, cov_mat_type, start_step, iter, eps, probs, weight
```

# Parameters

nclusters
: number of Gaussian distributions

cov_mat_type
: type of covariance matrix. 0 - spherical, 1 - diagonal, 2 - generic

start_step
: initial step EM starts from. 0 - Auto-step, 1 - E-step, 2 - M-step

iter/eps
: termination criteria of the procedure. iter for iteration times; eps for difference of change

probs
: initial probabilities (Pi,k); used(must be not NULL) only when EM starts from M-step

weights
: initial weights for each distribution; used(if not NULL) only when EM starts from E-step

means
: initial means of each distribution; used(must be not NULL) only when EM starts from E-step

covs
: initial covariance matrix of each distribution; used(if not NULL) only when EM starts from E-step

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

emtrain, empredict

# Name

empredict — Use trained parameter values of Gaussian mixtures by EM algorithm to predict the labels of the input data points.

```
[labels, probs] = empredict(test_samples, em_model)
```

# Parameters

test_samples
: n*dim data matrix. Each row is a data point.

em_model
: The model trained by emtrain.

labels
: The predicted labels of the input data.

probs
: Probabilites of data point belonging to each class.

# Examples

```
params = emparams(4, 0, 0, 10, 0.1);
train_samples = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/em_data',100,2);
[em_model, labels] = emtrain(train_samples, params);
dim = size(train_samples, 2);
test_samples = rand(50, dim)+100*ones(50, dim);
[labels, probs] = empredict(test_samples, em_model);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

J. A. Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report TR-97-021, International Computer Science Institute and Computer Science Division, University of California at Berkeley, April 1998.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

emparams, emtrain

# Name

emtrain — Use EM algorithm to stimate Gaussian mixture parameters from the sample set.

```
[model, labels] = emtrain(samples, params)
```

# Parameters

samples
: num*dim data matrix. Each row is a sample point.

params
: A struct variable containing parameter values for training. It has the following fields:

  - nclusters
  - cov_mat_type
  - start_step
  - iter
  - eps
  - probs
  - weights
  - means
  - covs

  Use function emparams to create this variable.

model
: The trained model containing Gaussian mixture parameters for the sample set.

labels
: Labels of the input samples calculated by the training.

# Examples

```
params = emparams(4, 0, 0, 10, 0.1);
train_samples = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/em_data',100,2);
[em_model, labels] = emtrain(train_samples, params);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

J. A. Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report TR-97-021, International Computer Science Institute and Computer Science Division, University of California at Berkeley, April 1998.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

## See Also

emparams, empredict

# Name

exhaustive_selection — exhaustive feature selection

```
[new_samples, feature_idx] = exhaustive_selection(samples, labels, final_dim, f
```

# Parameters

samples
: dim*num data matrix; each column is a data point, each row is a feature

labels
: a 1*num vector of lables for each data point

final_dim
: the number of output dimension after feature selection

fold
: number of folds for cross validation

new_samples
: final_dim*num data matrix after feature selection

feature_idx
: selected feature indices

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

forward_selection, backward_selection

# Name

fda — Fisher's Linear Discriminant Analysis

```
[nx, w] = fda(x, c)
```

# Parameters

x
: Data matrix of size dim*num. Each column is a data point.

c
: Class label vector of size 1*num or num*1.

nx
: New data vector.

w
: Weight vector.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

forward_selection — forward feature selection

```
[new_samples, feature_idx] = forward_selection(samples, labels, final_dim, fold
```

# Parameters

samples
: dim*num data matrix; each column is a data point, each row is a feature

labels
: a 1*num vector of labels for each data point

final_dim
: the number of output dimension after feature selection

fold
: number of folds for cross validation

new_samples
: final_dim*num data matrix after feature selection

feature_idx
: selected feature indices

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

backward_selection, exhaustive_selection

# Name

fuzzy_kmeans — Fuzzy K-Means Algorithm

```
[labels, centroids] = fuzzy_kmeans(data, k, b)
```

# Parameters

data
: dim*num data matrix; each column is a data point.

k
: number of nearest neighbors.

b
: b > 1 is a free parameter chosen to adjust the “blending” of different clusters

labels
: labels of the input data.

centroids
: cluster centroids

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

kmeans_sci

# Name

generalizedda — Generalized Discriminant Analysis(GDA)

```
norAlpha = generalizedda(x, c, ker)
```

# Parameters

x
: dim*num data matrix. Each column is a data point.

c
: Class label vector of size 1*num or num*1.

ker
: A struct variable for creating kernel matrix.

norAlpha
: Alpha normalized vector.

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

G. Baudat, F. Anouar, "Generalized Discriminant Analysis Using a Kernel Approach", Neural Computation, 12:2385-2404, 2000.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

hdr — hierarchical dimensionality reduction (hdr)

```
new_samples = hdr(train_samples, dimension)
```

# Parameters

train_samples
: data matrix of size dim*num; each column is a sample

dimension
: required output dimension

new_samples
: new data matrix of size dimension*num after dimensionality reduction

# Description

The function merges similar features in terms of the correlation matrix of the features to reduce the input data dimensionality.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

mds, pca2

# Name

hmm_backward — backward algorithm for computing the probabilty of an observation sequence given a hidden markov model

```
[P, be] = hmm_backward(tran_prob, emit_prob, ob_seq, start_prob)
```

# Parameters

tran_prob
: transition probability matrix of size N*N

emit_prob
: emission probability matrix of size N*M

ob_seq
: observation sequence in numerics

start_prob
: initial probability vector of size 1*N

P
: probability of an observation sequence generated by the given model

be
: probability of matrix of size T*N be(t,i) represents the probability of remainder partial observation sequence O(t+1)O(t+1)...O(T) given hidden state Si at time t

# Examples

```
tran_prob = [1.  0.  0.  0.;
             0.2 0.3 0.1 0.4;
             0.2 0.5 0.2 0.1;
             0.8 0.1 0.  0.1];
emit_prob = [1.  0.  0.  0.  0. ;
             0.  0.3 0.4 0.1 0.2;
             0.  0.1 0.1 0.7 0.1;
             0.  0.5 0.2 0.1 0.2];
ob_seq = [2 4 3 1];
start_prob = [0.2 0.3 0.1 0.4];

[P, be] = hmm_backward(tran_prob, emit_prob, ob_seq, start_prob)
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

A tutorial on Hidden Markov Models and selected applications in speech recognition, L. Rabiner, 1989, Proc. IEEE 77(2):257--286.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

## See Also

hmm_forward, hmm_viterbi, hmm_baum

# Name

hmm_baum — Baum-Welch algorithm for estimating the parameters of a hidden markov model given an observation sequence

```
[tran_prob, emit_prob, start_prob] = hmm_baum(num_states, num_symbols, ob_seq)
```

# Parameters

num_states
: number of hidden states

num_symbols
: number of visible states

ob_seq
: observation sequence in numerics

tran_prob
: transition probability matrix

emit_prob
: emission probability matrix

start_prob
: initial probability vector

# Examples

```
num_states = 3;
num_symbols = 2;
ob_seq = [2 1 2 1 2 2 1 2 1 2 1 1 1 2 1 1 2 1 2 1 2 1 1 2 2 2 2 1 2 1 1 2 1 1 2

[tran_prob, emit_prob, start_prob] = hmm_baum(num_states, num_symbols, ob_seq)
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

A tutorial on Hidden Markov Models and selected applications in speech recognition, L. Rabiner, 1989, Proc. IEEE 77(2):257--286.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

hmm_forward, hmm_backward, hmm_viterbi

# Name

hmm_forward — foward algorithm for computing the probabilty of an observation sequence given a hidden markov model

```
[P, alpha] = hmm_forward(tran_prob, emit_prob, ob_seq, start_prob)
```

# Parameters

tran_prob
: transition probability matrix of size N*N

emit_prob
: emission probability matrix of size N*M

ob_seq
: observation sequence in numerics

start_prob
: initial probability vector of size 1*N

P
: probability of an observation sequence generated by the given model

alpha
: probability matrix of size T*N alpha(t,i) represents the probability of partial observation sequence O(1)O(2)...O(t) and hidden state being Si at time t

# Examples

```
tran_prob = [1.  0.  0.  0.;
             0.2 0.3 0.1 0.4;
             0.2 0.5 0.2 0.1;
             0.8 0.1 0.  0.1];
emit_prob = [1.  0.  0.  0.  0. ;
             0.  0.3 0.4 0.1 0.2;
             0.  0.1 0.1 0.7 0.1;
             0.  0.5 0.2 0.1 0.2];
ob_seq = [2 4 3 1];
start_prob = [0.2 0.3 0.1 0.4];

[P, alpha] = hmm_forward(tran_prob, emit_prob, ob_seq, start_prob)
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

A tutorial on Hidden Markov Models and selected applications in speech recognition, L. Rabiner, 1989, Proc. IEEE 77(2):257--286.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

hmm_backward, hmm_viterbi, hmm_baum

# Name

hmm_viterbi — Viterbi algorithm for computing the optimal hidden state sequence given an observation sequence and the hidden markov model

```
state_seq = hmm_viterbi(tran_prob, emit_prob, ob_seq, start_prob)
```

# Parameters

tran_prob
: transition probability matrix of size N*N

emit_prob
: emission probability matrix of size N*M

ob_seq
: observation sequence in numerics

start_prob
: initial probability vector of size 1*N

state_sep
: optimal hidden state sequence given the observation sequence *ob_seq*

# Examples

```
tran_prob = [1.  0.  0.  0.;
             0.2 0.3 0.1 0.4;
             0.2 0.5 0.2 0.1;
             0.8 0.1 0.  0.1];
emit_prob = [1.  0.  0.  0.  0. ;
             0.  0.3 0.4 0.1 0.2;
             0.  0.1 0.1 0.7 0.1;
             0.  0.5 0.2 0.1 0.2];
ob_seq = [2 4 3 1];
start_prob = [0.2 0.3 0.1 0.4];

state_seq = hmm_viterbi(tran_prob, emit_prob, ob_seq, start_prob)
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

A tutorial on Hidden Markov Models and selected applications in speech recognition, L. Rabiner, 1989, Proc. IEEE 77(2):257--286.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

hmm_forward, hmm_backward, hmm_baum

# Name

ica — Independent Component Analysis

```
[source, W, Aw] = ica(mixture, K)
```

# Parameters

mixture
: Observed signals; each column is a data point.

K
: Number of source components.

source
: Source signals.

W
: Weighted matrix.

Aw
: Whitening matrix.

# Description

The function *ica* seek those directions that show the independence of signals.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

pca2, mlda

# Name

kerneldensitycovering — Construct a set of hyperspheres to cover the entire query dataset.

```
[sph_cov, sph_cov_id] = kerneldensitycovering(Data_Query, Data_Ref, sigma)
```

# Parameters

Data_Query
: Query data matrix. Each column vector is a data point.

Data_Ref
: Reference data matrix. Each column vector is a data point.

sigma
: Bandwidth of Gaussian kernel.

sph_cov
: Constructed hyperspheres that cover the query data set.

sph_cov_id
: Vector of covering hypersphere index. Each data point receives one index.

# Description

This function is used to construct a set of hyperspheres to cover the query dataset.

# Author

Xiao-Tong Yuan <xtyuan@nlpr.ia.ac.cn>

# Bibliography

Xiao-Tong Yuan, Bao-Gang Hu and Ran He, Agglomerative Mean-Shift Clustering via Query Set Compression, SDM 2009

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

aggloms, sphereconstruction

# Name

kfda — Kernel Fisher Discriminant Analysis

```
alpha = kfda(x, c, ker)
```

# Parameters

x
: Data matrix of size dim*num. Each column is a data point.

c
: Class label vector of size 1*num or num*1.

ker
: Kernel, a struct variable.

alpha
: The principal eigenvector.

# Description

Note the function is used for two-class problems.

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Müller, “Fisher discriminant analysis with kernels,” in Neural Networks for Signal Processing IX, Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, Eds. Piscataway, NJ: IEEE, 1999, pp. 41–48.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

kmeans — K-Means clustering

```
IDX = kmeans(X, k)
```

# Parameters

X
: n-by-p data matrix of input samples. Rows of X correspond to a sample, columns correspond to variables.

k
: Number of clusters to split the samples by.

IDX
: n-by-1 vector containing cluster indices of each sample.

# Description

The *kmeans* function implements the K-Means algorithm that finds centers of *k* clusters and groups the input samples around the clusters. *IDX(i)* contains the cluster index of the sample stored in the i-th row of *X*.

# Examples

```
x = [rand(100, 2)+ones(100, 2);rand(100, 2)-ones(100, 2)];
idx = kmeans(x, 2);
plot(x(idx==1, 1), x(idx==1, 2), 'r.', 'MarkerSize', 5);
set(gca(),"auto_clear","off")
plot(x(idx==2, 1), x(idx==2, 2), 'b.', 'MarkerSize', 5);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

kmeans_sci, kmedoids

# Name

kmeans_sci — K-Means clustering

```
[labels, centroids] = kmeans_sci(data, k, stop)
```

# Parameters

data
: dim*num data matrix; each column is a data point.

k
: number of nearest neighbors.

stop
: a scalar in (0, 1). Stop iterations if improved rate is less than this value.

labels
: labels of the input data.

centroids
: cluster centroids

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

kmeans, fuzzy_kmeans

# Name

kmedoids — basic k-medoids clustering algorithm

```
[medoids, labels] = kmedoids(samples, knum)
```

# Parameters

samples
: dim*snum data matrix; each column is a data point

knum
: number of the medoids

medoids
: dim*knum matrix; each column is a cluster center

labels
: 1*snum vector assigning each data point to a cluster

# Description

The function partitions the *samples* into *knum* groups and minize the squared error. It chooses data-points as centers.

# Examples

```
motor = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/kmedoids_data',2,133);
c=4;
[medoids, labels]=kmedoids(motor,c);

set(gca(),"auto_clear","off");

col = ['r*','g+','bx','y.'];
cen = ['ro','go','bo','yo'];
for i=1:c,
  idx=find(labels==i);
  cluster=motor(:,idx);
  plot(cluster(1,:),cluster(2,:),col(i));
  plot(medoids(1,i),medoids(2,i),cen(i));
end
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

kmeans

# Name

knearest — Use the K Nearest Neighbors algorithm to classifiy a new sample.

```
test_labels = knearest(train_labels, train_data, test_data, k[, is_regression])
```

# Parameters

train_labels
: A 1d vector (a row or a column) of integer or floating-point data type.

train_data
: A m by n floating-point matrix of training instances with n features. Each row corresponds to a data sample, and each column a feature variable.

test_data
: A M by n floating-point matrix of testing instances with n features. Each row corresponds to a data sample, and each column a feature variable.

k
: An integer number of the nearest neighbors to be found.

is_regression
: Flag for classification or regression - 0(default) for classification and 1 for regression.

test_labels
: A 1d vector of integer(in case of classification) of floating-point data type. It contains as much elements as the row number of 'test_data'.

# Description

The function uses the K-Nearest Neighbors algorithm to predict to response for a new sample. When classifying a new point, it looks up its K nearest points and then labels the new point according to which sets containing the majority of its K neighbors.

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

svmtrain, svmpredict, nbayestrain, knearest_sci

# Name

knearest_sci — K-Nearest Neighbors

```
test_labels = knearest_sci(train_data, train_labels, test_data, k)
```

# Parameters

train_data
: dim*num data matrix; each column is a data point.

train_labels
: labels of each train_data.

test_data
: dim*t_num data matrix; each column is a data point.

k
: number of nearest neighbors

test_labels
: labels of each test_data.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

knearest

# Name

kpca — Kernel Principal Component Analysis

```
[eig_vec, eig_val, new_patterns] = kpca(patterns, ker, dimension)
```

# Parameters

patterns
: Data matrix. Each column is a data point.

ker
: Kernel, a struct variable.

dimension
: Number of dimension for the new patterns.

eig_vec
: Sorted eigenvector. Each column is an eigenvector. eig_vec'*eig_vec=I.

eig_val
: The sorted eigenvalue.

new_patterns
: New patterns.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

createkernel

# Name

lbg_vq — Linde-Buzo-Gray Vector Quantization Algorithm (LBG Design Algorithm)

```
[code_vec,labels,Q] = lbg_vq(training_vec,codevec_num)
```

# Parameters

training_vec
: k*M data matrix; each column is a data point

codevec_num
: number of code vectors

code_vec
: k*code_vec matrix; each column is a code vector

labels
: 1*M vector assigning each data point to a code vector

Q

# Description

The function realizes vector quantization(VQ) proposed by Linde,Buzo,Gray in 1980, based on a training sequence.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

mlda — Multiple Linear Discriminant Analysis

```
w = mlda(x, c)
```

# Parameters

x
: Data matrix of size dim*num. Each column is a data point

c
: Class label vector of size 1*num or num*1.

w
: Weight matrix of size dim*(class_num-1).

# Description

The function *lda* finds the linear combination of features to best separate two or more classes. The output *w* consists of the number of class-1 LDA components.

# Examples

```
x = [(rand(2,5)-ones(2,5)),(rand(2,3)+ones(2,3)),(rand(2,2)-[1 1; 0 0])];
c = [zeros(1,5),ones(1,3),2*ones(1,2)];

w = lda(x, c);
new_x = w'*x;
plot(x(1,:),x(2,:),'r.');
set(gca(),"auto_clear","off")
plot(new_x(1,:),new_x(2,:),'b.');
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

pca2, wpca

# Name

leader_follower — basic leader-follower clustering

```
[centers, labels] = leader_follower(train_samples, theta, eta)
```

# Parameters

train_samples
: data matrix of size dim*num; each column is a data point

theta
: threshold distance

eta
: rate of convergence with default value 0.1

centers
: cluster centers

labels
: the index of cluster assigned to each training sample

# Description

The function implements the basic leader-follower clustering algorithm. It generates a new cluster center when an new data sample differs from the existing clusters more than the threshold distance *theta*.

# Examples

```
samples = [rand(2,10), -1*rand(2,10)];
theta = 1;
[centers, labels] = leader_follower(samples, theta);
scf(1);
plot(samples(1,:), samples(2,:), 'b.', 'MarkerSize', 3);
scf(2);
style = ['g.', 'c.', 'y.', 'k.', 'm.'];
ul = unique(labels)
for i = 1:length(ul),
  plot(samples(1,find(labels==ul(i))), samples(2,find(labels==ul(i))), style(i)
  set(gca(), "auto_clear", "off");
end
plot(centers(1,:), centers(2,:), 'r.', 'MarkerSize', 4);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

competitive_learning

# Name

least_squares — use least-squares algorithm to classify

```
[test_labels, a] = least_squares(train_samples, train_labels, test_samples)
```

# Parameters

train_samples
: dim*num data matrix; each column is a data sapmle, each row is a feature

train_labels
: 1*num vector of labels for each data sample

test_samples
: dim*test_num data matrix

test_labels
: 1*test_num vector of predicted labels for test_samples

a
: weighted vector

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

## Name

lms — Least-Mean-Squared Rule(Widrow-Hoff) (for two-category cases)

```
[a, test_labels] = lms(train_samples, train_labels, param, test_samples)
```

## Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations, the convergence criterion and the convergence rate: [iterm,theta,eta]; The default value is [1000,0.01,0.1]

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: weight vector

test_labels
: predicted labels for the test samples

## Description

The function finds a linear classifier using least mean squared rule.

## Authors

Jia WU <jiawu83@gmail.com>

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# Name

mds — multidimensional scaling (MDS)

```
new_samples = mds(train_samples, dimension, criterion, rate, eps, max_iter)
```

# Parameters

train_samples
: data matrix of size dim*num; each column is a sample

dimension
: output dimension with default value 2

criterion
: criterion function with default 'ee'; the criterion can be:

  'ee' emphasize the largest errors

  'ff' emphasize the largest fractional errors

  'ef' emphasize the largest product of error and fractional error

rate
: convergence rate with default value 0.1

eps
: accuracy with default value 0.01

max_iter
: maximal number of iteration with default value 1000

new_samples
: new samples after multidimensional scaling

# Description

The function represent the sample points in a lower-dimensional space such that distances between points in that space correspond to the dissimilarities between points in the original space. It now supports 3 criterion functions and finally finds an optimal representation which minimizes the criterion function.

# Examples

```
train_samples = rand(4, 20);
new = mds(train_samples);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

pca2, hdr

# Name

ml_gaussian — Maximum Likelihood Estimation for Gaussian Cases

```
[m, cov] = ml_gaussian(x, c)
```

# Parameters

x
: Train samples of size dim*num. Each column is a sample.

c
: Class label vector of size 1*num or num*1.

m
: The estimate mean, one column for a class.

cov
: The estimate covariance matrix of each class.

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

msclustering — Mean-Shift Clustering

```
[cluster_centers, sample_id] = msclustering(samples, h)
```

# Parameters

samples
: dim*num data matrix; each column is a data point

h
: parameter for bandwidth

cluster_centers
: cluster centers

sample_id
: label for each sample to which cluster it belongs to

# Description

Mean-Shift clustering algorithm is a nonparametric clustering technique which does not require the prior knowledge of the number of clusters, and does not constrain the shape of the clusters. It's a practical application of the mode finding procedure. This function *msclustering* realizes the mean-shift clustering algorithm with a flat kernel.

# Examples

```
//create samples points
samples = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/msc_data',2,750);
h=0.75;
[cluster_centers, sample_id]=msclustering(samples,h);
//plot the samples and the cluster centers
set(gca(),'auto_clear','off');
for i=1:size(cluster_centers,2),
  colors=rand(1,3);
  plot(samples(1,find(sample_id==i)),samples(2,find(sample_id==i)),'.','markers
  plot(cluster_centers(1,i),cluster_centers(2,i),'o','markersize',8,'markfor','
end
```

# Author

Jia Wu <jiawu83@gmail.com>

# Bibliography

Fukunaga, K.; Hostetler, L.; , "The estimation of the gradient of a density function, with applications in pattern recognition," Information Theory, IEEE Transactions on , vol.21, no.1, pp. 32- 40, Jan 1975

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

aggloms

## Name

mseclustering — basic iterative minimum-squared-error clustering

```
[centers, labels] = mseclustering(train_samples, cluster_num)
```

## Parameters

train_samples
: data matrix of size dim*num; each column is a data point

cluster_num
: number of desired clusters

centers
: centers of the formed clusters

labels
: labels of each trainning sample belonging to the formed clusters

## Description

The function implements the basic iterative minimum-squared-error clustering algorithm. It iteratively searches for the desired number of clusters by minimizing the sum of squared error of the training samples with respect to the nearest cluster center. The initial cluster centers are selected from the training samples, and the initial partition is made according to nearest distance between sample data and cluster centers.

## Examples

```
samples = rand(2,30);
cluster_num = 3;
[centers, labels] = mseclustering(samples, cluster_num);
scf(0);
plot(samples(1,:), samples(2,:), 'b.', 'MarkerSize', 3);
scf(1);
clusters = unique(labels);
plot(samples(1,find(labels==clusters(1))),samples(2,find(labels==clusters(1))),
set(gca(),"auto_clear","off");
plot(samples(1,find(labels==clusters(2))),samples(2,find(labels==clusters(2))),
set(gca(),"auto_clear","off");
plot(samples(1,find(labels==clusters(3))),samples(2,find(labels==clusters(3))),
set(gca(),"auto_clear","off");
plot(centers(1,:), centers(2,:), 'r.', 'MarkerSize', 4);
```

## Authors

Jia Wu <jiawu83@gmail.com>

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

## See Also

sohclustering

# Name

nbayespredict — Predict the responses for input samples.

```
test_labels = nbayespredict(test_data, model)
```

# Parameters

test_data
: Data samples, each row corresponding to a data sample and column a feature variable.

model
: Normal Bayes model trained using *nbayestrain*.

test_labels
: A 1d vector of the input data samples' labels(class).

# Description

The function estimates the most probable classes for the input data samples using the pre-trained Normal Bayes model.

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

K. Fukunaga, "Introduction to Statistical Pattern Recognition. second ed.", New York: Academic Press, 1990.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

nbayestrain, knearest, svmtrain, svmpredict

# Name

nbayestrain — Create a Normal Bayes model.

```
model = nbayestrain(train_data, train_labels[, var_idx[, sample_idx]])
```

# Parameters

train_data
: Data samples, each row corresponding to a data sample and column a feature variable.

train_labels
: A 1d vector of the input data samples' labels(class), with the length of the number of input data samples.

var_idx
: Matrix identifies variables of interest.

sample_idx
: Matrix identifies samples of interest.

model
: The trained Normal Bayes model.

# Description

The function creates a Normal Bayes model. It assumes that feature vectors from each class are normally distributed(though not neccesarily independently distributed), so the whole data distribution function is assumed to be a Guassian mixture, one component per class. Then it estimates mean vectors and covariance matrices for every class, using the training data.

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

K. Fukunaga, "Introduction to Statistical Pattern Recognition. second ed.", New York: Academic Press, 1990.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

nbayespredict, knearest, svmtrain, svmpredict

# Name

parzen_classify — classification using simple parzen-window estimation

```
test_labels = parzen_classify(train_samples, train_labels, test_samples, h)
```

# Parameters

train_samples
: dim*num data matrix; each column is a data point

train_labels
: 1*num vector of labels of each training sample

test_samples
: dim*numt data matrix; each column is a data point

h
: window width

test_labels
: 1*numt vector of labels of each test sample

# Description

Parzen window density estimation is a non-parametric way of estimating the probability density function. This function classifies the samples according to the maximal posterior probabilities.

# Examples

```
train_samples=[rand(2,500),rand(2,500)+5.0,(-1)*rand(2,500)];
train_labels=[ones(1,500),2*ones(1,500),3*ones(1,500)];
test_samples=[(-1)*rand(2,10),rand(2,10),rand(2,10)+20];
h=0.8;
test_labels=parzen_classify(train_samples,train_labels,test_samples,h)
```

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

pnn_train, pnn_classify, rce_train, rce_classify

# Name

pca2 — Principal component analysis on data.

```
[new_patterns, m, eig_val, eig_vec] = pca2(patterns, dimension)
```

# Parameters

patterns
: Data matrix; each column corresponds to a data point.

dimension
: Number of dimension for the new patterns.

m
: Data centers.

eig_val
: The sorted eigenvalues of covariance matrix.

eig_vec
: The sorted eigenvectors of covariance matrix; each column corresponds to an eigenvector.

# Description

The function *pca2* implements PCA analysis on input data matrix *patterns*, returning the data center vector(mean vector) *m*, eigenvalues of covariance matrix *eig_val*, eigenvectors of the covariance matrix *eig_vec*, and new data matrix *new_patterns* for the input *patterns*.

# Examples

```
patterns=rand(2, 100);
dimension=2;
[new_patterns, m, eig_val, eig_vec]=pca2(patterns, dimension);
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

wpca, mlda, twodpca

# Name

perceptron_batch — Batch Perceptron Criterion Function (for two-category cases)

```
[a, test_labels] = perceptron_batch(train_samples, train_labels, param, test_sam
```

# Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations, the convergence criterion and the convergence rate: [iterm, theta, eta]. The default values are 1000, 0.01 and 0.01.

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: perceptron weights (weight vector)

test_labels
: predicted labels for the test samples

# Description

The function finds a linear perceptron classifier in batch mode. A large group of samples is used when computing each weight update.

# Examples

```
patterns = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_data',2,40);
targets = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_label',1,40);
train_samples=[patterns(:,1:7),patterns(:,11:17)];
train_labels=[targets(:,1:7),targets(:,11:17)];
param=[1000,0.01,0.01];
test_samples=[patterns(:,8:10),patterns(:,18:20)];
[a, test_labels]=perceptron_batch(train_samples,train_labels,param,test_samples
```

# Authors

Jia WU <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

perceptron_fis, perceptron_vim, perceptron_bvi, balanced_winnow

# Name

perceptron_bvi — Batch Variable Increment Perceptron Criterion Function (for two-category cases)

```
[a, test_labels] = perceptron_bvi(train_samples, train_labels, param, test_samp
```

# Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations and the convergence rate: [iterm,eta]. The default value is [1000,0.1].

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: perceptron weights (weight vector)

test_labels
: predicted labels for the test samples

# Description

The function finds a linear perceptron classifier in batch mode. The learning rate is variable.

# Examples

```
patterns = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_data',2,40);
targets = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_label',1,40);
train_samples=[patterns(:,1:7),patterns(:,11:17)];
train_labels=[targets(:,1:7),targets(:,11:17)];
param=[1000,0.1];
test_samples=[patterns(:,8:10),patterns(:,18:20)];
[a, test_labels]=perceptron_bvi(train_samples,train_labels,param,test_samples)
```

# Authors

Jia WU <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

perceptron_fis, perceptron_vim, perceptron_batch, balanced_winnow

# Name

perceptron_fis — Fixed-Increment Single-Sample Perceptron Criterion Function (for two-category cases)

```
[a, test_labels] = perceptron_fis(train_samples, train_labels, param, test_samp
```

# Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion: the maximum number of iterations with default value 4000

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: perceptron weights (weight vector)

test_labels
: predicted labels for the test samples

# Description

The function finds a linear perceptron classifier. It updates the weight vector whenever a single sample is misclassified.

# Examples

```
patterns = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_data',2,40);
targets = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_label',1,40);
train_samples=[patterns(:,1:7),patterns(:,11:17)];
train_labels=[targets(:,1:7),targets(:,11:17)];
param=4000;
test_samples=[patterns(:,8:10),patterns(:,18:20)];
[a, test_labels]=perceptron_fis(train_samples,train_labels,param,test_samples)
```

# Authors

Jia WU <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

perceptron_batch, perceptron_vim, perceptron_bvi, balanced_winnow

## Name

perceptron_vim — Variable-Increment Perceptron with Margin Criterion Function (for two-category cases)

```
[a, test_labels] = perceptron_vim(train_samples, train_labels, param, test_samp
```

## Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations, the convergence rate and the margin: [iterm,eta,b]. The default value is [1000,0.1,0.1].

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: perceptron weights (weight vector)

test_labels
: predicted labels for the test samples

## Description

The function finds a linear perceptron classifier with a margin.

## Examples

```
patterns = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_data',2,40);
targets = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_label',1,40);
train_samples=[patterns(:,1:7),patterns(:,11:17)];
train_labels=[targets(:,1:7),targets(:,11:17)];
param=[1000,0.1,0.1];
test_samples=[patterns(:,8:10),patterns(:,18:20)];
[a, test_labels]=perceptron_vim(train_samples,train_labels,param,test_samples)
```

## Authors

Jia WU <jiawu83@gmail.com>

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

## See Also

perceptron_fis, perceptron_batch, perceptron_bvi, balanced_winnow

# Name

plsiread — Read data from a file for training for the probabilistic latent semantic indexing approach.

```
[docnum wordnum matrix] = plsiread(filename, issparse)
```

# Parameters

filename
: Name of the file containing the training data.

issparse
: Whether the data in the file uses a sparse way (i.e.: wordid:count). 1 if the file is in the sparse style; 0 if not.

docnum
: Number of the documents.

wordnum
: Number of the words.

matrix
: The term-document matrix whose size is (docnum by wordnum).

# Description

plsiread reads training data from a given file into a Scilab matrix. The outputs can be used as inputs of plsitrain.

# Authors

Mingbo Wang <mbwang@nlpr.ia.ac.cn>
Jia Wu <jiawu83@gmail.com>

# Bibliography

Thomas Hofmann, Probabilistic latent semantic indexing, Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval(SIGIR-99), 1999

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

plsitrain

## Name

plsitrain — probabilistic latent semantic indexing training

```
[pz pwz pdz foldnums]=plsitrain(docnum,wordnum,matrix,nz,beta,itenum,frate,eta,
```

## Parameters

docnum
: Number of the documents.

wordnum
: Number of the words.

matrix
: The term-document matrix.

## Examples

```
        [docnum wordnum matrix]=plsiread(SCI+'/contrib/OpenPR-0.0.2/etc/data/pl
        [pz pwz pdz foldnums]=plsitrain(docnum,wordnum,matrix,20,1,2,0.2,0.8,0)
```

## Authors

Mingbo Wang <mbwang@nlpr.ia.ac.cn>
Jia Wu <jiawu83@gmail.com>

## Bibliography

Thomas Hofmann, Probabilistic latent semantic indexing, Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval(SIGIR-99), 1999

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

## See Also

plsiread

# Name

pnn_classify — classification using a Parzen probabilistic neural network

```
test_labels = pnn_classify(net, test_samples, sigma)
```

# Parameters

net
: a trained Parzen probabilistic neural network

test_samples
: dim*numt data matrix; each column is a data point

sigma
: window width of the transfer function(default 2)

test_labels
: 1*numt vector of labels of each test sample

# Description

Probability neural network(PNN) is a hardware implementation of the Parzen windows approach. This function classifies the samples according to the maximal activation of the network.

# Examples

```
train_samples = [100*rand(2,100), (-100)*rand(2,30)];
train_labels = [ones(1,100), 2*ones(1,30)];
net = pnn_train(train_samples, train_labels);

test_samples = [100*rand(2,5), (-100)*rand(2,5)];
test_labels = pnn_classify(net, test_samples)
```

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

parzen_classify, pnn_train, rce_train, rce_classify

# Name

pnn_train — create a Parzen probabilistic neural network

```
net = pnn_train(train_samples, train_labels)
```

# Parameters

train_samples
: dim*num data matrix; each column is a data point

train_labels
: 1*num vector of labels of each training sample

net
: Parzen probabilistic neural network

# Description

Probability neural network(PNN) is a hardware implementation of the Parzen windows approach. This function creates a Parzen probability neural network.

# Examples

```
train_samples = [100*rand(2,100), (-100)*rand(2,30)];
train_labels = [ones(1,100), 2*ones(1,30)];
net = pnn_train(train_samples, train_labels)
```

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

parzen_classify, pnn_classify, rce_train, rce_classify

# Name

qdmatch — Find point matches between two images.

```
matchAB = qdmatch(imA, imB)
[matchBC, matchABC] = qdmatch(imB, imC, matchAB)
```

# Parameters

imA
: The input gray image.

imB
: The input gray image.

imC
: The input gray image.

matchAB
: N by 5 matrix. It contains the point matches between imA and imB, and has the format of [y1 x1 y2 x2 zncc; ... ...]. N is the number of matches.

matchBC
: N by 5 matrix. It contains the point matches between imB and imC except those already recorded in matchABC, and has the same format as matchAB. N is the number of matches.

matchABC
: N by 6 matrix. It contains the point matches between imA, imB and imC, and has the format of [y1 x1 y2 x2 y3 x3; ... ...]. N is the number of matches.

# Description

The function is used to find point matches between two images. It first uses SIFT matching algorithm to find sparse point matches between the two images, and then uses "quasi-dense propagation" algorithm to get "quasi-dense" point matches.

# Authors

Zhenhui Xu <zhxu@nlpr.ia.ac.cn>
Jia Wu <jiawu83@gmail.com>

# Bibliography

D. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 2(60):91-110, 2004.

LHUILLIER, M., and QUAN, L. 2005. A quasi-dense approach to surface reconstruction from uncalibrated images. IEEE Transactions on Pattern Analysis andMachine Intelligence 27, 3, 418–433.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

randperm — Random permutation.

```
p = randperm(n)
```

# Parameters

n
: an integer

p
: a random permutation of the integers 1:n

# Description

The function returns a random permutaion of the integers 1:n.

# Examples

```
randperm(5)
ans =
2. 5. 4. 1. 3.
```

# Name

rce_classify — classification using a RCE(reduced coulomb energy) network

```
test_labels = rce_classify(net, test_samples)
```

# Parameters

net
: a trained reduced coulomb energy network

test_samples
: dim*numt data matrix; each column is a data point

test_labels
: 1*numt vector of labels of each test sample

# Description

Reduced coulomb energy of RCE network is one representative method of *potential functions*, which is the simplest method of some relaxation techniques. This function classifies the samples using a trained RCE network.

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

parzen_classify, pnn_train, pnn_classify, rce_train

# Name

rce_train — create a reduced coulomb energy or RCE network

```
net = rce_train(train_samples, train_labels, lambdam, epsilon)
```

# Parameters

train_samples
: dim*num data matrix; each column is a data point

train_labels
: 1*num vector of labels of each training sample

lambdam
: maximal radius

epsilon
: parameter used in D(x,x')-epsilon with defaul value 1e-4

net
: a reduced coulomb energy network

# Description

Reduced coulomb energy of RCE network is one representative method of *potential functions*, which is the simplest method of some relaxation techniques. This function is to create a RCE network.

# Author

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

parzen_classify, pnn_train, pnn_classify, rce_classify

# Name

readsparse — Read files in LIBSVM format.

```
[label_vector, instance_matrix] = readsparse(fname)
```

# Parameters

fname
: Data file name. The file must be in LIBSVM format which is:

```
label index1:value1 index2:value2 ...
                  .
                  .
                  .
```

label_vector
: An m by 1 vector (type must be double).

instance_matrix
: An m by n matrix. It can be dense or sparse (type must be double).

# Description

Read files in LIBSVM format. Two outputs are label_vector and instance_matrix, which can then be used as inputs of svmtrain or svmpredict.

# Authors

Chih-Jen Lin <cjlin[at]csie.ntu.edu.tw>
Chih-Chung Chang
Jun-Cheng Chen
Kuan-Jen Peng
Chih-Yuan Yang
Chih-Huai Cheng
Rong-En Fan
Ting-Fan Wu
Jia Wu <jiawu83[at]gmail.com>

# Bibliography

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

svmtrain, svmpredict

# Name

relaxation_brm — Batch Relaxation with Margin Criterion Function (for two-category cases)

```
[a, test_labels] = relaxation_brm(train_samples, train_labels, param, test_samp
```

# Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations, the convergence rate and the margin: [iterm,eta,b]. The default value is [1000,0.1,0.1].

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: weights (weight vector)

test_labels
: predicted labels for the test samples

# Description

The function finds a linear classifier with a margin using relaxation rule in a batch mode.

# Examples

```
patterns = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_data',2,40);
targets = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_label',1,40);
train_samples=[patterns(:,1:7),patterns(:,11:17)];
train_labels=[targets(:,1:7),targets(:,11:17)];
param=[1000,0.01,0.01];
test_samples=[patterns(:,8:10),patterns(:,18:20)];
[a, test_labels]=relaxation_brm(train_samples,train_labels,param,test_samples)
```

# Authors

Jia WU <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

relaxation_srm

# Name

relaxation_srm — Single-Sample Relaxation with Margin Criterion Function (for two-category cases)

```
[a, test_labels] = relaxation_srm(train_samples, train_labels, param, test_samp
```

# Parameters

train_samples
: training data matrix of size dim*num_tr; each column is a data point

train_labels
: 1*num_tr vector of labels for the training samples

param
: parameters for the criterion, including the maximum number of iterations, the convergence rate and the margin: [iterm,eta,b]. The default value is [1000,0.1,0.1].

test_samples
: test data matrix of size dim*num_te; each column is a data point

a
: weights (weight vector)

test_labels
: predicted labels for the test samples

# Description

The function finds a linear classifier with a margin, using a single-sample relaxation rule with margin.

# Examples

```
patterns = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_data',2,40);
targets = read(SCI+'/contrib/OpenPR-0.0.2/etc/data/perceptron_label',1,40);
train_samples=[patterns(:,1:7),patterns(:,11:17)];
train_labels=[targets(:,1:7),targets(:,11:17)];
param=[1000,0.01,0.01];
test_samples=[patterns(:,8:10),patterns(:,18:20)];
[a, test_labels]=relaxation_srm(train_samples,train_labels,param,test_samples)
```

# Authors

Jia WU <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

relaxation_brm

## Name

sohclustering — stepwise optimal hierarchical clustering

```
[centers, labels] = sohclustering(train_samples, cluster_num)
```

## Parameters

train_samples
: data matrix of size dim*num; each column is a data point

cluster_num
: number of desired clusters

centers
: centers of the formed clusters

labels
: labels of each trainning sample belonging to the formed clusters

## Description

The function implements the bottom-up stepwise optimal hierarchical clustering. The algorithm starts with every training sample being a singleton cluster, then it iteratively merges two clusters which change a certain criterion function the least, until the desired number of clusters are formed. In this routine a sum-of-squared-error criterion function is used.

## Examples

```
samples = rand(2,30);
cluster_num = 3;
[centers, labels] = sohclustering(samples, cluster_num);
scf(0);
plot(samples(1,:), samples(2,:), 'b.', 'MarkerSize', 3);
scf(1);
clusters = unique(labels);
plot(samples(1,find(labels==clusters(1))),samples(2,find(labels==clusters(1))),
set(gca(),"auto_clear","off");
plot(samples(1,find(labels==clusters(2))),samples(2,find(labels==clusters(2))),
set(gca(),"auto_clear","off");
plot(samples(1,find(labels==clusters(3))),samples(2,find(labels==clusters(3))),
set(gca(),"auto_clear","off");
plot(centers(1,:), centers(2,:), 'r.', 'MarkerSize', 4);
```

## Authors

Jia Wu <jiawu83@gmail.com>

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

## See Also

ahclustering

# Name

sphereconstruction — nstruct hypersphere from a given point by Mean-Shift.

```
[s_center, radius] = sphereconstruction(p_start, data, sigma)
```

# Parameters

p_start
: Given point from which hypersphere to be constructed.

data
: Data matrix. Each column vector is a data point.

sigma
: Bandwidth of Gaussian kernel.

s_center
: Center of the constructed hypersphere.

radius
: Radius of the cosntructed hypersphere.

# Description

This function is used to construct a hypersphere from a given point by running one-iteration of Mean-Shift on the data set.

# Author

Xiao-Tong Yuan <xtyuan@nlpr.ia.ac.cn>

# Bibliography

Xiao-Tong Yuan, Bao-Gang Hu and Ran He, Agglomerative Mean-Shift Clustering via Query Set Compression, SDM 2009

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

aggloms, kerneldensitycovering

## Name

svmpredict — Predict class of the new input data according to a pre-trained model.

```
[predicted_label,accuracy,decision_values/prob_estimates]=svmpredict(testing_la
```

## Parameters

testing_label_vector
: An m by 1 vector of prediction labels. If labels of test data are unknown, simply use any random values. (type must be double)

testing_instance_matrix
: An m by n matrix of m testing instances with n features. It can be dense or sparse. (type must be double)

model
: The output of svmtrain.

libsvm_options
: A character string of testing options.

  -b probability_estimates
  : whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM only 0 is supported

predicted_label
: A vector of predicted labels.

accuracy
: A vector including accuracy (for classification), mean squared error, and squared correlation coefficient (for regression).

decision_values/prob_estimates
: A matrix containing decision values or probability estimates (if '-b 1' is specified). If k is the number of classes, for decision values, each row includes results of predicting k(k-1/2) binary-class SVMs. For probabilities, each row contains k values indicating the probability that the testing instance is in each class. The order of classes here is the same as 'Label' field in the model structure.

## Description

Predict class of the new input data according to a pre-trained model.

## Examples

```
        Probability estimates (need '-b 1' for training and testing):

             [label_vector, instance_vector] = readsparse(SCI+'/contrib/OpenPR-
             model = svmtrain(label_vector, instance_vector, '-c 1 -g 0.07 -b 1
             [label_vector, instance_vector] = readsparse(SCI+'/contrib/OpenPR-
             [predict_label, accuracy, prob_estimates] = svmpredict(label_vecto
```

## Authors

Chih-Jen Lin cjlin@csie.ntu.edu.tw
Chih-Chung Chang
Jun-Cheng Chen

Kuan-Jen Peng
Chih-Yuan Yang
Chih-Huai Cheng
Rong-En Fan
Ting-Fan Wu
Jia Wu jiawu83@gmail.com

# Bibliography

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

svmtrain, readsparse

# Name

svmtrain — Train data.

```
model=svmtrain(training_label_vector, training_instance_matrix [, libsvm_option
```

# Parameters

training_label_vector
: An m by 1 vector of training labels (type must be double).

training_instance_matrix
: An m by n matrix of m training instances with n features. It can be dense or sparse (type must be double).

libsvm_options
: A character string of training options.

-s svm_type
: set type of SVM (default 0)

```
0 - C-SVC
1 - nu-SVC
2 - one-class SVM
3 - epsilon SVR
4 - nu-SVR
```

-t kernel_type
: set type of kernel function (default 2)

```
0 - linear - u'*v
1 - polynomial - gamma*u'*v + coef0)^degree
2 - radial basis function - exp(-gamma*|u-v|^2)
3 - sigmoid - tanh(gamma*u'*v + coef0)
4 - precomputed kernel (kernel values in training_set_file)
```

-d degree
: set degree in kernel function (default 3)

-g gamma
: set gamma in kernel function (default 1/k). The k means the number of attributes in the input data

-r coef0
: set coef0 in kernel function (default 0)

-c cost
: set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

-n nu
: set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)

-p epsilon
: set the epsilon in loss function of epsilon-SVR (default 0.1)

-m cachesize
: set cache memory size in MB (default 100)

-e epsilon
: set tolerance of termination criterion (default 0.001)

-h shrinking
: whether to use the shrinking heuristics, 0 or 1 (default 1)

-b probability_estimates
: whether to train an SVC or SVR model for probability estimates, 0 or 1 (default 0)

-wi weight
: set the parameter C of class i to weight*C in C-SVC (default 1)

-v n
: n-fold cross validation mode. option -v randomly splits the data into n parts and calculates cross validation accuracy/mean squared error on them

model
: The returned model structure for future prediction by 'svmtrain'. It is a structure organized as [Parameters, nr_class, totalSV, rho, Label, ProbA, ProbB, nSV, sv_coef, SVs].

Parameters
: parameters

nr_class
: number of classes; = 2 for regression/one-class svm

totalSV
: total #SV

rho
: -b of the decision function(s) wx+b

Label
: label of each class; empty for regression/one-class SVM

ProbA
: pairwise probability information; empty if -b 0 or in one-class SVM

ProbB
: pairwise probability information; empty if -b 0 or in one-class SVM

nSV
: number of SVs for each class; empty for regression/one-class SVM

sv_coef
: coefficients for SVs in decision functions

SVs
: support vectors

# Description

Train data and generate a 'Modle' file which could be thought as a storage format for the internal data of SVM.

# Examples

```
[label_vector, instance_vector] = readsparse(SCI+'/contrib/OpenPR-0.0.2/
modle = svmtrain(label_vector, instance_vector, '-c 1 -g 0.07')
```

# Authors

Chih-Jen Lin <cjlin@csie.ntu.edu.tw>

Chih-Chung Chang
Jun-Cheng Chen
Kuan-Jen Peng
Chih-Yuan Yang
Chih-Huai Cheng
Rong-En Fan
Ting-Fan Wu
Jia Wu <jiawu83@gmail.com>

# Bibliography

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

svmpredict, readsparse

# Name

twodpca — Two-Dimensional PCA

```
[Y, X] = twodpca(A, K)
```

# Parameters

A
: m*n*d hypermat; m*n is the size of the sample matrix; d is the number of the sample

K
: number of projection axis Xk; K <= n

Y
: m*K*d hypermat; m*K is the size of the feature matrix; each column is a projected feature vector Yk = A*Xk; d is the number of the sample

X
: The set of projection axes; each column is a projection axis.

# Description

Two-Dimensional principal component analysis (2DPCA) is to project an m*n sample matrix *Ai* onto an n-dimensional unitary column vector *Xk* by a linear transformation `Yk = AiXk`, to get an m-dimensional feature vector *Yk*. *K* principal component vectors of sample *Ai* can be get from `Yk = AiXk k=1,2,...,K`. An m*K matrix `B = [Y1, Y2, ..., YK]` which is the feature matrix of *Ai* can be obtained.

# Authors

Jia Wu <jiawu83@gmail.com>

# Bibliography

J. Yang, D. Zhang, A. Frangi, and J. Yang. Two-dimensional pca: A new approach to appearance-based face representation and recognition. IEEE Trans. on Pattern Analysis and Machine Intelligence, 26(1):131–137, 2004.

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

pca2

# Name

usecart — Use a trained classification and regression tree to predict the labels of given samples.

```
labels = usecart(samples, indices, cart)
```

# Parameters

samples
: input sample matrix; each column is a sample point

indices
: indices of selected samples

cart
: the trained classification and regression tree by function buildcart

labels
: predicted labels of the input samples

# Examples

```
train_samples=[0.15 0.09 0.29 0.38 0.52 0.57 0.73 0.47 0.1  0.08 0.23 0.7  0.62
               0.83 0.55 0.35 0.7  0.48 0.73 0.75 0.06 0.29 0.15 0.16 0.19 0.47
train_labels=[1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0];
impurity_type='Gini';
cart=buildcart(train_samples,train_labels,impurity_type);

samples=rand(2,10);
indices=1:size(samples,2);
labels = usecart(samples, indices, cart)
```

# Authors

Jia Wu <jiawu83@gmail.com>

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn

# See Also

buildcart

## Name

whitening — Whitening transformation.

```
[new_mat, Aw] = whitening(mat)
```

## Parameters

mat
: Input matrix.

new_mat
: New matrix after whitening transformation.

Aw
: Whitening matrix.

## Description

The function *whitening* converts the covariance matrix of the input *mat* into the identity matrix.

## Authors

Jia Wu <jiawu83@gmail.com>

## Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# Name

wpca — Weighted Principal Component Analysis

```
[eig_vec, m_vec, eig_val] = wpca(Train_Patterns, weight)
```

# Parameters

Train_Patterns
: Data matrix. Each column vector is a data point.

weight
: A column vector whose length is equal to number of data points.

eig_vec
: Each column is an eigvector. eig_vec'*eig_vec=I.

m_vec
: Data center.

eig_val
: he sorted eigvalue of WPCA algorithm.

# Description

The function is used for weighted principal component analysis.

# Examples

```
       e.g.1: learn the principal component of a dataset
             Train_Patterns = rand(2,100);   //number of dimension is 2 number o
             weight = ones(100,1);           //A column vector of weight
             [eig_vec,m_vec,eig_val] = wpca(Train_Patterns,weight);

       e.g.2: a graph example of principal component
             rotation = [7 -cos(3.14/4);sin(3.14/4) 1];
             Train_Patterns = rand(2,100);
             Train_Patterns = rotation* Train_Patterns;
             weight = ones(100,1) ;                                    //equal w
             plot2d(Train_Patterns(1,:),Train_Patterns(2,:),style=-4); //plot th
             [eig_vec,m_vec,eig_val] = wpca(Train_Patterns,weight);
             plot2d(m_vec(1),m_vec(2),style=-5);                       //plot th
             //plot the first principal component
             x=-1:0.1:7;
             y=(eig_vec(2,1)/eig_vec(1,1))*(x-m_vec(1))+m_vec(2);
             plot2d(x,y,style=2);
             legends(["data";"data center";"eigen vector"],[-4,-5,2], with_box=%
```

# Authors

Ran He <rhe@nlpr.ia.ac.cn>

# Bibliography

C.M. Bishop. Pattern Recognition and Machine Learning. Information Science and Statistics, 2006

# Availability

The latest version of OpenPR can be found at http://www.openpr.org.cn.

# See Also

pca2, mlda