MATLAB实例：聚类初始化方法与数据归一化方法

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

初始化方法有：随机初始化，K-means初始化，FCM初始化。。。。。。

归一化方法有：不归一化，z-score归一化，最大最小归一化。。。。。。

1. 聚类初始化方法

init_methods.m

```
function label=init_methods(data, K, choose)
% 输入：无标签数据，聚类数，选择方法
% 输出：聚类标签
if choose==1
    %随机初始化，随机选K行作为聚类中心，并用欧氏距离计算其他点到其聚类，将数据集分为K类，输出每个样例的类标签
    [X_num, ~]=size(data);
    rand_array=randperm(X_num);    %产生1~X_num之间整数的随机排列
    para_miu=data(rand_array(1:K), :);  %随机排列取前K个数，在X矩阵中取这K行作为初始聚类中心
    %欧氏距离，计算（X-para_miu）^2=X^2+para_miu^2-2*X*para_miu'，矩阵大小为X_num*K
    distant=repmat(sum(data.*data,2),1,K)+repmat(sum(para_miu.*para_miu,2)',X_num,1)-2*data*para_miu';
    %返回distant每行最小值所在的下标
    [~,label]=min(distant,[],2);
elseif choose==2
    %用kmeans进行初始化聚类，将数据集聚为K类，输出每个样例的类标签
    label=kmeans(data, K);
elseif choose==3
    %用FCM算法进行初始化
    options=[NaN, NaN, NaN, 0];
    [~, responsivity]=fcm(data, K, options);   %用FCM算法求出隶属度矩阵
    [~, label]=max(responsivity', [], 2);
elseif choose==4
    label = litekmeans(data, K,'Replicates',20);
end
```

litekmeans.m

```
function [label, center, bCon, sumD, D] = litekmeans(X, k, varargin)
% [IDX, C] = litekmeans(data, K,'Replicates',20);
%LITEKMEANS K-means clustering, accelerated by matlab matrix operations.
%
%   label = LITEKMEANS(X, K) partitions the points in the N-by-P data matrix
%   X into K clusters.  This partition minimizes the sum, over all
%   clusters, of the within-cluster sums of point-to-cluster-centroid
%   distances.  Rows of X correspond to points, columns correspond to
%   variables.  KMEANS returns an N-by-1 vector label containing the
%   cluster indices of each point.
%
%   [label, center] = LITEKMEANS(X, K) returns the K cluster centroid
%   locations in the K-by-P matrix center.
%
%   [label, center, bCon] = LITEKMEANS(X, K) returns the bool value bCon to
%   indicate whether the iteration is converged.
%
%   [label, center, bCon, SUMD] = LITEKMEANS(X, K) returns the
%   within-cluster sums of point-to-centroid distances in the 1-by-K vector
%   sumD.
%
%   [label, center, bCon, SUMD, D] = LITEKMEANS(X, K) returns
%   distances from each point to every centroid in the N-by-K matrix D.
%
%   [ ... ] = LITEKMEANS(..., 'PARAM1',val1, 'PARAM2',val2, ...) specifies
%   optional parameter name/value pairs to control the iterative algorithm
%   used by KMEANS.  Parameters are:
%
%   'Distance' - Distance measure, in P-dimensional space, that KMEANS
%      should minimize with respect to.  Choices are:
%            {'sqEuclidean'} - Squared Euclidean distance (the default)
%             'cosine'       - One minus the cosine of the included angle
%                              between points (treated as vectors). Each
%                              row of X SHOULD be normalized to unit. If
%                              the intial center matrix is provided, it
%                              SHOULD also be normalized.
%
%   'Start' - Method used to choose initial cluster centroid positions,
%      sometimes known as "seeds".  Choices are:
%         {'sample'}  - Select K observations from X at random (the default)
%          'cluster' - Perform preliminary clustering phase on random 10%
%                      subsample of X.  This preliminary phase is itself
%                      initialized using 'sample'. An additional parameter
%                      clusterMaxIter can be used to control the maximum
%                      number of iterations in each preliminary clustering
%                      problem.
%           matrix   - A K-by-P matrix of starting locations; or a K-by-1
%                      indicate vector indicating which K points in X
%                      should be used as the initial center.  In this case,
%                      you can pass in [] for K, and KMEANS infers K from
%                      the first dimension of the matrix.
%
%   'MaxIter'    - Maximum number of iterations allowed.  Default is 100.
%
%   'Replicates' - Number of times to repeat the clustering, each with a
%                  new set of initial centroids. Default is 1. If the
%                  initial centroids are provided, the replicate will be
%                  automatically set to be 1.
%
% 'clusterMaxIter' - Only useful when 'Start' is 'cluster'. Maximum number
%                    of iterations of the preliminary clustering phase.
%                    Default is 10.
%
%
%    Examples:
%
%       fea = rand(500,10);
%       [label, center] = litekmeans(fea, 5, 'MaxIter', 50);
%
%       fea = rand(500,10);
%       [label, center] = litekmeans(fea, 5, 'MaxIter', 50, 'Replicates', 10);
%
%       fea = rand(500,10);
%       [label, center, bCon, sumD, D] = litekmeans(fea, 5, 'MaxIter', 50);
%       TSD = sum(sumD);
%
%       fea = rand(500,10);
%       initcenter = rand(5,10);
%       [label, center] = litekmeans(fea, 5, 'MaxIter', 50, 'Start', initcenter);
%
%       fea = rand(500,10);
%       idx=randperm(500);
%       [label, center] = litekmeans(fea, 5, 'MaxIter', 50, 'Start', idx(1:5));
%
%
%   See also KMEANS
%
%    [Cite] Deng Cai, "Litekmeans: the fastest matlab implementation of
%           kmeans," Available at:
%           http://www.zjucadcg.cn/dengcai/Data/Clustering.html, 2011.
%
%   version 2.0 --December/2011
%   version 1.0 --November/2011
%
```

```
%   Written by Deng Cai (dengcai AT gmail.com)

if nargin < 2
    error('litekmeans:TooFewInputs','At least two input arguments required.');
end

[n, p] = size(X);

pnames = {   'distance' 'start'   'maxiter'  'replicates' 'onlinephase' 'clustermaxiter'};
dflts =  {'sqeuclidean' 'sample'       []        []        'off'              []        };
[eid,errmsg,distance,start,maxit,reps,online,clustermaxit] = getargs(pnames, dflts, varargin{:});
if ~isempty(eid)
    error(sprintf('litekmeans:%s',eid),errmsg);
end

if ischar(distance)
    distNames = {'sqeuclidean','cosine'};
    j = strcmpi(distance, distNames);
    j = find(j);
    if length(j) > 1
        error('litekmeans:AmbiguousDistance', ...
            'Ambiguous ''Distance'' parameter value:  %s.', distance);
    elseif isempty(j)
        error('litekmeans:UnknownDistance', ...
            'Unknown ''Distance'' parameter value:  %s.', distance);
    end
    distance = distNames{j};
else
    error('litekmeans:InvalidDistance', ...
        'The ''Distance'' parameter value must be a string.');
end

center = [];
if ischar(start)
    startNames = {'sample','cluster'};
    j = find(strncmpi(start,startNames,length(start)));
    if length(j) > 1
        error(message('litekmeans:AmbiguousStart', start));
    elseif isempty(j)
        error(message('litekmeans:UnknownStart', start));
    elseif isempty(k)
        error('litekmeans:MissingK', ...
            'You must specify the number of clusters, K.');
    end
    if j == 2
        if floor(.1*n) < 5*k
            j = 1;
        end
    end
    start = startNames{j};
elseif isnumeric(start)
    if size(start,2) == p
        center = start;
    elseif (size(start,2) == 1 || size(start,1) == 1)
        center = X(start,:);
    else
        error('litekmeans:MisshapedStart', ...
            'The ''Start'' matrix must have the same number of columns as X.');
    end
    if isempty(k)
        k = size(center,1);
    elseif (k ~= size(center,1))
        error('litekmeans:MisshapedStart', ...
            'The ''Start'' matrix must have K rows.');
    end
    start = 'numeric';
else
    error('litekmeans:InvalidStart', ...
        'The ''Start'' parameter value must be a string or a numeric matrix or array.');
end

% The maximum iteration number is default 100
if isempty(maxit)
    maxit = 100;
end

% The maximum iteration number for preliminary clustering phase on random
% 10% subsamples is default 10
if isempty(clustermaxit)
    clustermaxit = 10;
end


% Assume one replicate
if isempty(reps) || ~isempty(center)
    reps = 1;
end

if ~(isscalar(k) && isnumeric(k) && isreal(k) && k > 0 && (round(k)==k))
    error('litekmeans:InvalidK', ...
        'X must be a positive integer value.');
elseif n < k
    error('litekmeans:TooManyClusters', ...
        'X must have more rows than the number of clusters.');
```

```
end

bestlabel = [];
sumD = zeros(1,k);
bCon = false;

for t=1:reps
    switch start
        case 'sample'
            center = X(randsample(n,k),:);
        case 'cluster'
            Xsubset = X(randsample(n,floor(.1*n)),:);
            [dump, center] = litekmeans(Xsubset, k, varargin{:}, 'start','sample', 'replicates',1 ,'MaxIter',clustermaxit);
        case 'numeric'
    end

    last = 0;label=1;
    it=0;

    switch distance
        case 'sqeuclidean'
            while any(label ~= last) && it<maxit
                last = label;

                bb = full(sum(center.*center,2)');
                ab = full(X*center');
                D = bb(ones(1,n),:) - 2*ab;

                [val,label] = min(D,[],2); % assign samples to the nearest centers
                ll = unique(label);
                if length(ll) < k
                    %disp([num2str(k-length(ll)),' clusters dropped at iter ',num2str(it)]);
                    missCluster = 1:k;
                    missCluster(ll) = [];
                    missNum = length(missCluster);

                    aa = sum(X.*X,2);
                    val = aa + val;
                    [dump,idx] = sort(val,1,'descend');
                    label(idx(1:missNum)) = missCluster;
                end
                E = sparse(1:n,label,1,n,k,n);  % transform label into indicator matrix
                center = full((E*spdiags(1./sum(E,1)',0,k,k))'*X);    % compute center of each cluster
                it=it+1;
            end
            if it<maxit
                bCon = true;
            end
            if isempty(bestlabel)
                bestlabel = label;
                bestcenter = center;
                if reps>1
                    if it>=maxit
                        aa = full(sum(X.*X,2));
                        bb = full(sum(center.*center,2));
                        ab = full(X*center');
                        D = bsxfun(@plus,aa,bb') - 2*ab;
                        D(D<0) = 0;
                    else
                        aa = full(sum(X.*X,2));
                        D = aa(:,ones(1,k)) + D;
                        D(D<0) = 0;
                    end
                    D = sqrt(D);
                    for j = 1:k
                        sumD(j) = sum(D(label==j,j));
                    end
                    bestsumD = sumD;
                    bestD = D;
                end
            else
                if it>=maxit
                    aa = full(sum(X.*X,2));
                    bb = full(sum(center.*center,2));
                    ab = full(X*center');
                    D = bsxfun(@plus,aa,bb') - 2*ab;
                    D(D<0) = 0;
                else
                    aa = full(sum(X.*X,2));
                    D = aa(:,ones(1,k)) + D;
                    D(D<0) = 0;
                end
                D = sqrt(D);
                for j = 1:k
                    sumD(j) = sum(D(label==j,j));
                end
                if sum(sumD) < sum(bestsumD)
                    bestlabel = label;
                    bestcenter = center;
                    bestsumD = sumD;
                    bestD = D;
                end
            end
        case 'cosine'
```

```
while any(label ~= last) && it<maxit
    last = label;
    W=full(X*center');
    [val,label] = max(W,[],2); % assign samples to the nearest centers
    ll = unique(label);
    if length(ll) < k
        missCluster = 1:k;
        missCluster(ll) = [];
        missNum = length(missCluster);
        [dump,idx] = sort(val);
        label(idx(1:missNum)) = missCluster;
    end
    E = sparse(1:n,label,1,n,k,n);  % transform label into indicator matrix
    center = full((E*spdiags(1./sum(E,1)',0,k,k))'*X);    % compute center of each cluster
    centernorm = sqrt(sum(center.^2, 2));
    center = center ./ centernorm(:,ones(1,p));
    it=it+1;
end
if it<maxit
    bCon = true;
end
if isempty(bestlabel)
    bestlabel = label;
    bestcenter = center;
    if reps>1
        if any(label ~= last)
            W=full(X*center');
        end
        D = 1-W;
        for j = 1:k
            sumD(j) = sum(D(label==j,j));
        end
        bestsumD = sumD;
        bestD = D;
    end
else
    if any(label ~= last)
        W=full(X*center');
    end
    D = 1-W;
    for j = 1:k
        sumD(j) = sum(D(label==j,j));
    end
    if sum(sumD) < sum(bestsumD)
        bestlabel = label;
        bestcenter = center;
        bestsumD = sumD;
```

```
                    bestD = D;
                end
            end
    end
end

label = bestlabel;
center = bestcenter;
if reps>1
    sumD = bestsumD;
    D = bestD;
elseif nargout > 3
    switch distance
        case 'sqeuclidean'
            if it>=maxit
                aa = full(sum(X.*X,2));
                bb = full(sum(center.*center,2));
                ab = full(X*center');
                D = bsxfun(@plus,aa,bb') - 2*ab;
                D(D<0) = 0;
            else
                aa = full(sum(X.*X,2));
                D = aa(:,ones(1,k)) + D;
                D(D<0) = 0;
            end
            D = sqrt(D);
        case 'cosine'
            if it>=maxit
                W=full(X*center');
            end
            D = 1-W;
    end
    for j = 1:k
        sumD(j) = sum(D(label==j,j));
    end
end


function [eid,emsg,varargout]=getargs(pnames,dflts,varargin)
%GETARGS Process parameter name/value pairs
%   [EID,EMSG,A,B,...]=GETARGS(PNAMES,DFLTS,'NAME1',VAL1,'NAME2',VAL2,...)
%   accepts a cell array PNAMES of valid parameter names, a cell array
%   DFLTS of default values for the parameters named in PNAMES, and
%   additional parameter name/value pairs.  Returns parameter values A,B,...
%   in the same order as the names in PNAMES.  Outputs corresponding to
%   entries in PNAMES that are not specified in the name/value pairs are
%   set to the corresponding value from DFLTS.  If nargout is equal to
%   length(PNAMES)+1, then unrecognized name/value pairs are an error.  If
%   nargout is equal to length(PNAMES)+2, then all unrecognized name/value
%   pairs are returned in a single cell array following any other outputs.
%
%   EID and EMSG are empty if the arguments are valid.  If an error occurs,
%   EMSG is the text of an error message and EID is the final component
%   of an error message id.  GETARGS does not actually throw any errors,
%   but rather returns EID and EMSG so that the caller may throw the error.
%   Outputs will be partially processed after an error occurs.
%
%   This utility can be used for processing name/value pair arguments.
%
%   Example:
%       pnames = {'color' 'linestyle', 'linewidth'}
%       dflts  = {    'r'         '_'          '1'}
%       varargin = {{'linew' 2 'nonesuch' [1 2 3] 'linestyle' ':'}
%       [eid,emsg,c,ls,lw] = statgetargs(pnames,dflts,varargin{:})    % error
%       [eid,emsg,c,ls,lw,ur] = statgetargs(pnames,dflts,varargin{:}) % ok

% We always create (nparams+2) outputs:
%    one each for emsg and eid
%    nparams varargs for values corresponding to names in pnames
% If they ask for one more (nargout == nparams+3), it's for unrecognized
% names/values

%   Original Copyright 1993-2008 The MathWorks, Inc.
%   Modified by Deng Cai (dengcai@gmail.com) 2011.11.27

% Initialize some variables
emsg = '';
eid = '';
nparams = length(pnames);
varargout = dflts;
unrecog = {};
nargs = length(varargin);

% Must have name/value pairs
if mod(nargs,2)~=0
    eid = 'WrongNumberArgs';
    emsg = 'Wrong number of arguments.';
else
    % Process name/value pairs
    for j=1:2:nargs
        pname = varargin{j};
        if ~ischar(pname)
            eid = 'BadParamName';
            emsg = 'Parameter name must be text.';
            break;
        end
        i = strcmpi(pname,pnames);
        i = find(i);
        if isempty(i)
            % if they've asked to get back unrecognized names/values, add this
            % one to the list
            if nargout > nparams+2
                unrecog((end+1):(end+2)) = {varargin{j} varargin{j+1}};
                % otherwise, it's an error
            else
                eid = 'BadParamName';
                emsg = sprintf('Invalid parameter name:  %s.',pname);
                break;
            end
        elseif length(i)>1
            eid = 'BadParamName';
            emsg = sprintf('Ambiguous parameter name:  %s.',pname);
            break;
        else
            varargout{i} = varargin{j+1};
        end
    end
end

varargout{nparams+1} = unrecog;
```

2. 数据归一化方法：normlization.m

```
function data = normlization(data, choose)
% 数据归一化
if choose==0
    % 不归一化
    data = data;
elseif choose==1
    % Z-score归一化
    data = bsxfun(@minus, data, mean(data));
    data = bsxfun(@rdivide, data, std(data));
elseif choose==2
    % 最大-最小归一化处理
    [data_num,~]=size(data);
    data=(data-ones(data_num,1)*min(data))./(ones(data_num,1)*(max(data)-min(data)));
end
```

注意：可以在elseif后面添加自己的方法。