POSTERIOR AGREEMENT FOR CLUSTERING IN THE PROBABILITY SIMPLEX

Author: Nico Stephan Gorbach , Institute of Machine Learning, ETHZ, email: nico.gorbach@inf.ethz.ch

Implementation of " **Pipeline Validation for Connectivity-based Cortex Parcellation** " by Nico S. Gorbach, Marc Tittgemeyer and Joachim M. Buhmann

Simulation for validation by posterior agreeement for clustering in the probability simplex.

Contents


```
% clear workspace, close figures
clear all; close all; clc;
```

Introduction

In this instructional code we demonstrate validation by posterior agreement for clustering in the probability simplex whereby the connectivity scores are sampled from the Dirichlet distribution. Additionally, we validate clustering across more degrees of freedom different potential clusters by determining the maximum generalization capacity for different potential clusters. We refer to the maximum generalization capacity as the information content of the algorithm.

Input

Set the number of objects to cluster, the number of true clusters and the number of potential clusters for estimation.

```
n = 2500;      % number of objects
ktrue = 9;     % true number of clusters
K = 2:15;      % number of potential clusters
```

Generate data from the probability simplex

Sample data points from the Dirichlet distribution.

```
dim = 3;       % number of dimensions

% sample Dirichlet parameters
for i = 1:ktrue
    dirichlet_param(i,:) = dirichletRnd(1,ones(1,3)./3);
end

% sample probability measurements from Dirichlet distribution
for k = 1:size(dirichlet_param,1)
    for i = 1:floor(n/size(dirichlet_param,1))
        cluster1{k}(i,:) = dirichletRnd(5e1,dirichlet_param(k,:));
        cluster2{k}(i,:) = dirichletRnd(5e1,dirichlet_param(k,:));
    end
end

% pack data
data1 = [cluster1{:}]; data1 = reshape(data1',dim,[])';
data2 = [cluster2{:}]; data2 = reshape(data2',dim,[])';

% sample very noisy data
for i = 1:500
    corruption1(i,:) = dirichletRnd(3e0,ones(1,3)./3);
    corruption2(i,:) = dirichletRnd(3e0,ones(1,3)./3);
end

% pack and scale data
data1 = 500*[data1;corruption1];
data2 = 500*[data2;corruption2];

%h = setup_plots;
```

```
% preprocessing for Bayesian evidence

alpha{1} = full(sum(data1,1));
```

```
alpha{2} = full(sum(data1,1))/100;
```

```
% start timer
tic;
```

Histogram clustering

Deterministic annealing

Determine global minimizer.

```
% Annealing settings
beta_init = 1e-3;     % starting inverse temperature
beta_step = 1.1;      % inverse temperature step
beta_stop = 1e0;      % stopping inverse temperature
perturb_sd = 1e-6;    % centroid perturbation

% Initialization of information content
info_content.hc = zeros(1,max(K));

for k = K
```

```
    % Initialization of Gibbs distributions
    gibbs_dist1 = ones(size(data1,1),k) ./ k;
    gibbs_dist2 = ones(size(data2,1),k) ./ k;

    % Initialization of centroids
    centroid1 = gibbs_dist1'*data1;
    centroid1 = bsxfun(@rdivide,centroid1,sum(centroid1,2));
    centroid1(centroid1==0) = eps;
    centroid2 = gibbs_dist2'*data2;
    centroid2 = bsxfun(@rdivide,centroid2,sum(centroid2,2));
    centroid2(centroid2==0) = eps;

    j = 0; beta = beta_init;
    while beta <= beta_stop
```

Perturb centroids

Avoid local minimum by perturbing centroids: $\phi_{kj}^{(\cdot)} = \phi_{kj}^{(\cdot)} + \epsilon$

```
        centroid1 = centroid1 + perturb_sd * rand(size(centroid1));
        centroid1 = bsxfun(@rdivide,centroid1,sum(centroid1,2)); % normalize
        centroid2 = centroid2 + perturb_sd * rand(size(centroid2));
        centroid2 = bsxfun(@rdivide,centroid2,sum(centroid2,2)); % normalize
```

Expectation maximization

Iterate between determining Gibbs distributions and maximzing variational lower bound w.r.t. centroids.

```
        for iter = 1:10
```

Costs for histogram clustering given instance 1

KL divergence between empirical probabilities (data) and centroid probabilities (up to proportionality constant): $R_{ik}^{(1)} = -\sum_j x_{ij}^{(1)} \log(\phi_{kj}^{(1)})$

```
            potential1 = -data1 * log(centroid1)';
```

Costs for histogram clustering given instance 2

KL divergence between empirical probabilities (data) and centroid probabilities (up to proportionality constant): $R_{ik}^{(2)} = -\sum_j x_{ij}^{(2)} \log(\phi_{kj}^{(2)})$

```
            potential2 = -data2 * log(centroid2)';
```

Gibbs distribution 1

Maximum entropy distribution: $p_{ik}^{(1)} = \exp\left(-\beta R_{ik}^{(1)}\right)/Z$

```
            gibbs_dist1 = exp(-beta * potential1);
            partition_sum1 = sum(gibbs_dist1,2);
            gibbs_dist1 = bsxfun(@rdivide,gibbs_dist1,partition_sum1);

            % avoid underflow
            idx = find(partition_sum1==0);
            if ~isempty(idx)
                [~,min_cost_idx] = min(potential1(idx,:),[],2);
                max_ind = sub2ind(size(gibbs_dist1),idx,min_cost_idx);
                gibbs_dist1(idx,:) = zeros(length(idx),k);
                gibbs_dist1(max_ind) = 1;
            end
```

Gibbs distribution 2

Maximum entropy distribution: $p_{ik}^{(2)} = \exp\left(-\beta R_{ik}^{(2)}\right)/Z$

```
            gibbs_dist2 = exp(-beta * potential2);
            partition_sum2 = sum(gibbs_dist2,2);
            gibbs_dist2 = bsxfun(@rdivide,gibbs_dist2,partition_sum2);

            % avoid underflow
            idx = find(partition_sum2==0);
            if ~isempty(idx)
                [~,min_cost_idx] = min(potential2(idx,:),[],2);
                max_ind = sub2ind(size(gibbs_dist2),idx,min_cost_idx);
                gibbs_dist2(idx,:) = zeros(length(idx),k);
                gibbs_dist2(max_ind) = 1;
            end
```

Joint Gibbs distribution

Maximum entropy distribution: $p_{ik}^{(1,2)} = \exp\left(-\beta(R_{ik}^{(1)} + R_{ik}^{(2)})\right)/Z$

```
            dist_joint = exp(-beta * (potential1 + potential2));
            joint_partition_sum = sum(dist_joint,2);
```

Centroids for instance 1

Probability prototype: $\phi_{kj}^{(1)} = \frac{\sum_i p_{ik}^{(1)} x_{ij}^{(1)}}{\sum_j \sum_{i'} p_{i'k}^{(1)} x_{i'j}^{(1)}}$

```
            centroid1 = gibbs_dist1'*data1;
            centroid1 = bsxfun(@rdivide,centroid1,sum(centroid1,2));
            centroid1(centroid1==0) = eps;
```

Centroids for instance 2

Probability prototype: $\phi_{kj}^{(2)} = \frac{\sum_i p_{ik}^{(2)} x_{ij}^{(2)}}{\sum_j \sum_{i'} p_{i'k}^{(2)} x_{i'j}^{(2)}}$

```
            centroid2 = gibbs_dist2'*data2;
            centroid2 = bsxfun(@rdivide,centroid2,sum(centroid2,2));
            centroid2(centroid2==0) = eps;

        end

        % increase inverse temperature:
        beta = beta * beta_step;
```

Match clusters across data instances

Use Hungarian algorithm to match clusters.

```
        match_clusters_idx = munkres(pdist2(centroid1,centroid2));
        potential2=potential2(:,match_clusters_idx);
```

Log partition sum for instance 1

Determine log partition sum while avoiding underflow: $\log Z_1 = \sum_i \log \sum_k \exp\left(-\beta R_{ik}^{(1)}\right)$

```
        scaled_cost1 = -beta * potential1;
        % log-sum-exp trick to prevent underflow
        max_scaled_cost1 = max(scaled_cost1,[],2);
        log_partition_sum1 = max_scaled_cost1 + log(sum(exp(scaled_cost1-max_scaled_cost1),2));
```

Log partition sum for instance 2

Determine log partition sum while avoiding underflow: $\log Z_2 = \sum_i \log \sum_k \exp\left(-\beta R_{ik}^{(2)}\right)$

```
        scaled_cost2 = -beta * potential2;
        % log-sum-exp trick to prevent underflow
        max_scaled_cost2 = max(scaled_cost2,[],2);
        log_partition_sum2 = max_scaled_cost2 + log(sum(exp(scaled_cost2-max_scaled_cost2),2));
```

Joint log partition sum

Determine joint log partition sum while avoiding underflow: $\log Z_{12} = \sum_i \log \sum_k \exp\left(-\beta (R_{ik}^{(1)} + R_{ik}^{(2)})\right)$

```
        joint_scaled_cost = -beta * (potential1 + potential2);
        % log-sum-exp trick to prevent underflow
        max_scaled_cost3 = max(joint_scaled_cost,[],2);
        log_joint_partition_sum = max_scaled_cost3 + log(sum(exp(joint_scaled_cost-max_scaled_cost3),2));

        j = j+1;
```

Generalization capacity

Resolution of the hypothesis space: $GC(\beta) = \log(k) + \frac{1}{n}\left(\log Z_{12} - \log Z_1 - \log Z_2\right)$

```
        gc.hc{k}(j) = log(k) + sum(log_joint_partition_sum - log_partition_sum1 ...
            - log_partition_sum2) ./ size(partition_sum1,1);

        % store results
        inv_temp.hc(j) = beta;
        gibbs_dist_packed1.hc{k}(:,:,j) = gibbs_dist1;
        gibbs_dist_packed2.hc{k}(:,:,j) = gibbs_dist2(:,match_clusters_idx);

        % plot results
        %plot(h{2},inv_temp,gc{k},'LineWidth',2); drawnow % plot generalization capacity
        %simplex(h{1},data1(:,1),data1(:,2),data1(:,3),gibbs_dist1); % display Gibbs distribution on probability simplex
```

```
    end
```

Number of equivariant transformations

Richness of the hypothesis space: $\frac{1}{n}\log|\{\tau\}| = H(n_1/n, \dots, n_k/n)$

```
    gibbs_dist1 = round(gibbs_dist1);
    d = sum(gibbs_dist1,1); d = d./sum(d); d(d==0) = 1;
    nTransformations = -d * log(d)';

     % correct generalization capacity
    gc.hc{k} = gc.hc{k}-log(k)+nTransformations;

    gc.hc{k} = gc.hc{k} * log2(exp(1));  % transforming units from nats to bits
```

Information content

Quality of algorithm: $\mathcal{I} = \max_\beta GC(\beta)$

```
    [info_content.hc(k),max_gc_idx.hc(k)] = max(gc.hc{k});
```

```
    % check if emprical risk minimizer (ERM) contains k clusters
    if sum(logical(sum(gibbs_dist1,1)))==k
```

Bayesian Information Criterion (BIC)

$BIC := m \times k \times ln(n) + 2R(c^\perp, \mathbf{X})$ where m is the number of bins and $c^\perp$ is the empirical risk minimizer

```
    cost(k) = sum(sum(gibbs_dist1 .* (-bsxfun(@rdivide,data1,sum(data1,2)) * log(centroid1)')));
    %cost = sum(sum(gibbs_dist1 .* (-data1 * log(centroid1)')));
    BIC(k) = size(data1,2) * k * log(size(data1,1)) + 2 * cost(k);
```

Akaike Information Criterion (AIC)

$AIC := 2 \times m \times k + 2R(c^\perp, \mathbf{X})$ where m is the number of bins and $c^\perp$ is the empirical risk minimizer

```
    AIC(k) = 2 * size(data1,2) * k + 2 * cost(k);
```

```
    else
        BIC(k) = NaN;
        AIC(k) = NaN;
    end
```

Bayesian evidence

$p(\mathbf{X} \mid \alpha) = \int p(\mathbf{X} \mid \phi) p(\theta \mid \alpha) d\phi = \prod_k \frac{B(\alpha_k + \mathbf{n}_k)}{B(\alpha_k)}$ where $\mathbf{n}_k := (n_{k1}, n_{k2}, \ldots, n_{kd})$ and $n_{kj} := \sum_{i:c(i)=k} x_{ij}$

```
    [~,labels] = max(gibbs_dist1,[],2);
    log_bayes_evidence{1}(k) = 0;
    log_bayes_evidence{2}(k) = 0;
    for c = unique(labels)'
        ncj = full(sum(data1(labels==c,:),1));
        alpha_tmp{1} = alpha{1}/k; alpha_tmp{1}(ncj==0) = [];
        alpha_tmp{2} = alpha{2}; alpha_tmp{2}(ncj==0) = [];
        ncj(ncj==0) = [];

        log_bayes_evidence{1}(k) = log_bayes_evidence{1}(k) + sum(gammaln(alpha_tmp{1} + ncj)) ...
            + gammaln(sum(alpha_tmp{1})) - gammaln(sum(alpha_tmp{1} + ncj)) - sum(gammaln(alpha_tmp{1}));
        log_bayes_evidence{2}(k) = log_bayes_evidence{2}(k) + sum(gammaln(alpha_tmp{2} + ncj)) ...
            + gammaln(sum(alpha_tmp{2})) - gammaln(sum(alpha_tmp{2} + ncj)) - sum(gammaln(alpha_tmp{2}));
    end
```

```
end
```

Kmeans

Deterministic annealing

Determine global minimizer.

```
% Annealing settings
beta_init = 1e-6;     % starting inverse temperature
beta_step = 1.1;      % inverse temperature step
beta_stop = 5e-3;      % stopping inverse temperature
perturb_sd = 1e-6;    % centroid perturbation

% Initialization of information content
info_content.kmeans = zeros(1,max(K));

for k = K
```

```
    % Initialization of Gibbs distributions
    gibbs_dist1 = ones(size(data1,1),k) ./ k;
    gibbs_dist2 = ones(size(data2,1),k) ./ k;

    % Initialization of centroids
    centroid1 = gibbs_dist1'*data1;
    centroid1 = bsxfun(@rdivide,centroid1,sum(gibbs_dist1,1)');
    centroid2 = gibbs_dist2'*data2;
    centroid2 = bsxfun(@rdivide,centroid2,sum(gibbs_dist2,1)');

    j = 0; beta = beta_init;
    while beta <= beta_stop
```

Perturb centroids

Avoid local minimum by perturbing centroids: $\phi_{kj}^{(\cdot)} = \phi_{kj}^{(\cdot)} + \epsilon$

```
        centroid1 = centroid1 + perturb_sd * rand(size(centroid1));
        centroid2 = centroid2 + perturb_sd * rand(size(centroid2));
```

Expectation maximization

Iterate between determining Gibbs distributions and maximzing variational lower bound w.r.t. centroids.

```
        for iter = 1:10
```

Costs for histogram clustering given instance 1

KL divergence between empirical probabilities (data) and centroid probabilities (up to proportionality constant): $R_{ik}^{(1)} = -\sum_j x_{ij}^{(1)} \log(\phi_{kj}^{(1)})$

```
            potential1 = pdist2(data1,centroid1,'squaredeuclidean');
```

Costs for histogram clustering given instance 2

KL divergence between empirical probabilities (data) and centroid probabilities (up to proportionality constant): $R_{ik}^{(2)} = -\sum_j x_{ij}^{(2)} \log(\phi_{kj}^{(2)})$

```
            potential2 = pdist2(data2,centroid2,'squaredeuclidean');
```

Gibbs distribution 1

Maximum entropy distribution: $p_{ik}^{(1)} = \exp\left(-\beta R_{ik}^{(1)}\right)/Z$

```
            gibbs_dist1 = exp(-beta * potential1);
            partition_sum1 = sum(gibbs_dist1,2);
            gibbs_dist1 = bsxfun(@rdivide,gibbs_dist1,partition_sum1);

            % avoid underflow
            idx = find(partition_sum1==0);
            if ~isempty(idx)
                [~,min_cost_idx] = min(potential1(idx,:),[],2);
                max_ind = sub2ind(size(gibbs_dist1),idx,min_cost_idx);
                gibbs_dist1(idx,:) = zeros(length(idx),k);
                gibbs_dist1(max_ind) = 1;
            end
```

Gibbs distribution 2

Maximum entropy distribution: $p_{ik}^{(2)} = \exp\left(-\beta R_{ik}^{(2)}\right)/Z$

```
            gibbs_dist2 = exp(-beta * potential2);
            partition_sum2 = sum(gibbs_dist2,2);
            gibbs_dist2 = bsxfun(@rdivide,gibbs_dist2,partition_sum2);

            % avoid underflow
            idx = find(partition_sum2==0);
            if ~isempty(idx)
                [~,min_cost_idx] = min(potential2(idx,:),[],2);
                max_ind = sub2ind(size(gibbs_dist2),idx,min_cost_idx);
                gibbs_dist2(idx,:) = zeros(length(idx),k);
                gibbs_dist2(max_ind) = 1;
            end
```

Centroids for instance 1

Probability prototype: $\phi_{kj}^{(1)} = \frac{\sum_i p_{ik}^{(1)} x_{ij}^{(1)}}{\sum_j \sum_{i'} p_{i'k}^{(1)} x_{i'j}^{(1)}}$

```
            centroid1 = gibbs_dist1'*data1;
            centroid1 = bsxfun(@rdivide,centroid1,sum(gibbs_dist1,1)');
```

Centroids for instance 2

Probability prototype: $\phi_{kj}^{(2)} = \frac{\sum_i p_{ik}^{(2)} x_{ij}^{(2)}}{\sum_j \sum_{i'} p_{i'k}^{(2)} x_{i'j}^{(2)}}$

```
            centroid2 = gibbs_dist2'*data2;
            centroid2 = bsxfun(@rdivide,centroid2,sum(gibbs_dist2,1)');

        end

        % increase inverse temperature:
        beta = beta * beta_step;
```

Match clusters across data instances

Use Hungarian algorithm to match clusters.

```
        match_clusters_idx = munkres(pdist2(centroid1,centroid2));
        potential2=potential2(:,match_clusters_idx);
```

Log partition sum for instance 1

Determine log partition sum while avoiding underflow: $\log Z_1 = \sum_i \log \sum_k \exp\left(-\beta R_{ik}^{(1)}\right)$

```
        scaled_cost1 = -beta * potential1;
        % log-sum-exp trick to prevent underflow
        max_scaled_cost1 = max(scaled_cost1,[],2);
        log_partition_sum1 = max_scaled_cost1 + log(sum(exp(scaled_cost1-max_scaled_cost1),2));
```

Log partition sum for instance 2

Determine log partition sum while avoiding underflow: $\log Z_2 = \sum_i \log \sum_k \exp\left(-\beta R_{ik}^{(2)}\right)$

```
        scaled_cost2 = -beta * potential2;
        % log-sum-exp trick to prevent underflow
        max_scaled_cost2 = max(scaled_cost2,[],2);
        log_partition_sum2 = max_scaled_cost2 + log(sum(exp(scaled_cost2-max_scaled_cost2),2));
```

Joint log partition sum

Determine joint log partition sum while avoiding underflow: $\log Z_{12} = \sum_i \log \sum_k \exp\left(-\beta (R_{ik}^{(1)} + R_{ik}^{(2)})\right)$

```
        joint_scaled_cost = -beta * (potential1 + potential2);
        % log-sum-exp trick to prevent underflow
        max_scaled_cost3 = max(joint_scaled_cost,[],2);
        log_joint_partition_sum = max_scaled_cost3 + log(sum(exp(joint_scaled_cost-max_scaled_cost3),2));

        j = j+1;
```

Generalization capacity

Resolution of the hypothesis space: $GC(\beta) = \log(k) + \frac{1}{n}\left(\log Z_{12} - \log Z_1 - \log Z_2\right)$

```
        gc.kmeans{k}(j) = log(k) + sum(log_joint_partition_sum - log_partition_sum1 ...
            - log_partition_sum2) ./ size(partition_sum1,1);

        % store results
        inv_temp.kmeans(j) = beta;
        gibbs_dist_packed1.kmeans{k}(:,:,j) = gibbs_dist1;
        gibbs_dist_packed2.kmeans{k}(:,:,j) = gibbs_dist2(:,match_clusters_idx);

        % plot results
        %plot(h{2},inv_temp,gc{k},'LineWidth',2); drawnow % plot generalization capacity
        %simplex(h{1},data1(:,1),data1(:,2),data1(:,3),gibbs_dist1); % display Gibbs distribution on probability simplex
```

```
    end
```

Number of equivariant transformations

Richness of the hypothesis space: $\frac{1}{n}\log|\{\tau\}| = H(n_1/n, \ldots, n_k/n)$

```
    gibbs_dist1 = round(gibbs_dist1);
    d = sum(gibbs_dist1,1); d = d./sum(d); d(d==0) = 1;
    nTransformations = -d * log(d)';

     % correct generalization capacity
    gc.kmeans{k} = gc.kmeans{k}-log(k)+nTransformations;

    gc.kmeans{k} = gc.kmeans{k} * log2(exp(1));  % transforming units from nats to bits
```

Information content

Quality of algorithm: $\mathcal{I} = \max_\beta GC(\beta)$

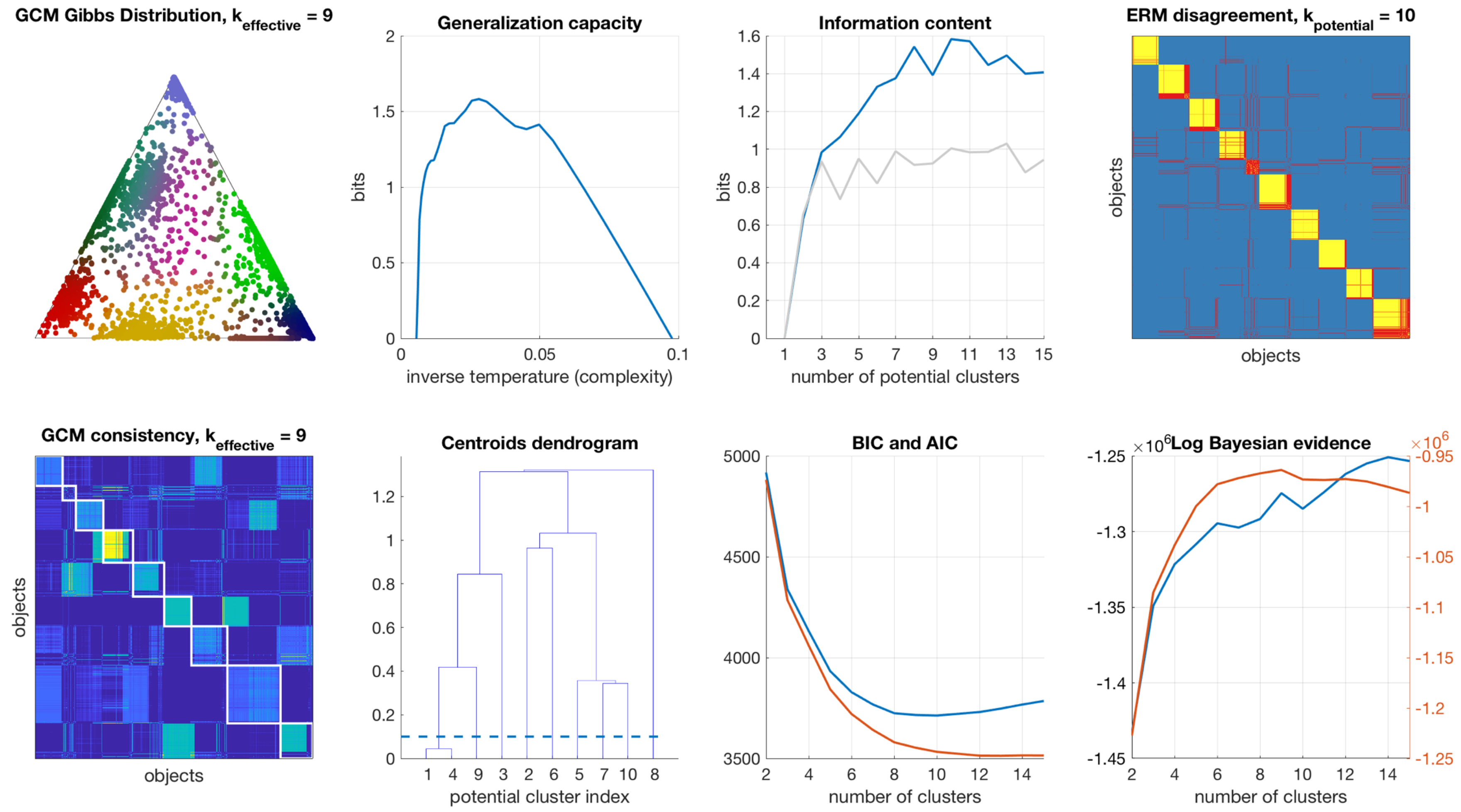
```
    [info_content.kmeans(k),max_gc_idx.kmeans(k)] = max(gc.kmeans{k});
```

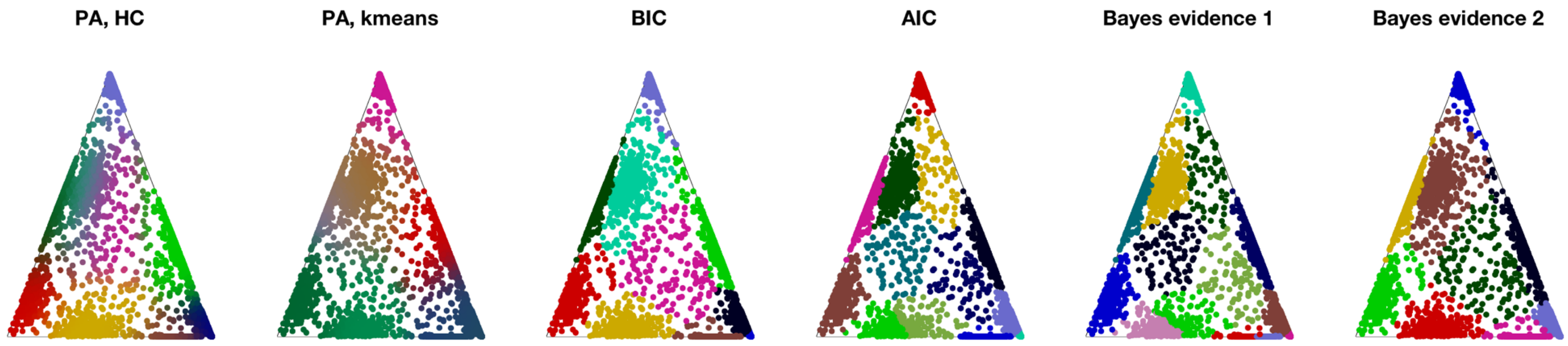
```
end
```

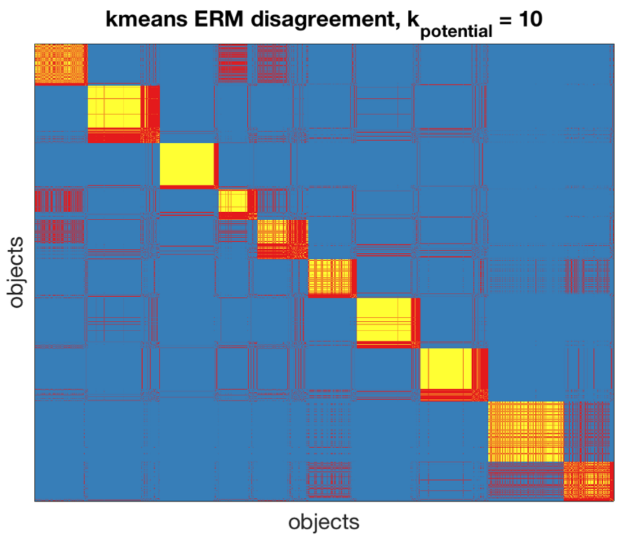
Results

```
display_result(info_content,gc,max_gc_idx,inv_temp,BIC,AIC,log_bayes_evidence,...
    gibbs_dist_packed1,gibbs_dist_packed2,data1,k,K)
```

```
runtime = 0.85587 minutes
```







Discussion

The posterior agreement is compared to alternative validation criteria including BIC, AIC and Bayesian evidence We additionally determine Bayesian evidence for two different hyperparameter settings in order to investigate the sensitivity of ranking by Bayesian evidence on the hyperparameters.

Published with MATLAB® R2017a