1 Introduction

This lab explores the use of K-Means algorithm for classifying images of lower-case printed characters. Questions of interest include:

1. Is K-Means good enough for this task?
2. What is the effect of PCA on the overall K-Means performance?
3. What is a good set of features?
4. What is a good way for calculating cluster centers?
5. What are good stopping criteria for K-Means?
6. What is a good way to initialize the cluster centers?
7. What is a good way to select training and testing data?

2 Method

K-Means is an unsupervised algorithm that takes in the number of classes K, initializes cluster centers to random data points/feature points, and then iteratively assigns each point to the class of its nearest cluster center. This iterative process is repeated until a stopping criterion is reached, after which the clusters remain stationary and represent corresponding classes. The primary input data used in this document consists of 18 feature dimensions (Set #1). Section 2.1 explains these features in detail. It also describes another feature Set #2, comprising of low-medium level features. Another Set #3 comprising of low-high level features is given by Prof. Tom Henderson for Adaboost classifier and is used for comparison too, but is not listed here.

2.1 Features

2.1.1 Set#1 - Slope Change Characteristics

This feature encodes the derivative of border pixels. Basically, for each side of the image viz., left, right, top and bottom, the border pixels as marked by the red pixels in the Figure 1, are extracted. For each such border pixel obtained, the non-varying pixel coordinate is stored. For example, when processing the left border, the column coordinates are stored and similarly, for the top border pixels, the row coordinates are stored. From each of the four 1D arrays of values thus obtained, the derivative arrays are computed by replacing an element with difference between the current and previous element. For thus obtained derivative array, the counts of positive, negative and zero values are then stored as the features. A fourth value encodes the sign/neutralness of the first value encountered in the derivative array. As there are 4 vectors, $4^*4 = 16$ feature values are obtained from the whole image under this category. A full explanation of this feature using an example is given in the Verification Section listed later.

2.1.2 Set#1 - Full/Empty Row Characteristic

This characteristic is computed as follows:

\[ \text{feature} = \text{number of fully filled rows} - \text{number of empty rows} \]
Figure 1: Figure showing image of character $a$ with its border pixels marked in red
2.1.3 Set#1 - Full/Empty Column Characteristic
This characteristic is computed as follows:

\[ \text{feature} = \text{number of fully filled columns} - \text{number of empty columns} \]

2.1.4 Set#2 - Euler Number
This attribute is computed using the \textit{bweuler} function in Matlab as follows:

Euler Number = \text{number of objects in the image} - \text{total number of holes in the objects}

2.1.5 Set#2 - Number of Full Rows
This attribute is computed as follows:

\[ \text{feature} = \text{number of fully filled rows} \]

2.1.6 Set#2 - Number of Empty Rows
This attribute is computed as follows:

\[ \text{feature} = \text{number of empty rows} \]

2.1.7 Set#2 - Number of Full Columns
This attribute is computed as follows:

\[ \text{feature} = \text{number of fully filled columns} \]

2.1.8 Set#2 - Number of Empty Columns
This attribute is computed as follows:

\[ \text{feature} = \text{number of empty columns} \]

2.1.9 Set#2 - Top or Bottom?
This attribute is a Boolean that decides where more number of 1s occur (in a binary image) among the top and bottom halves of the pixels. And is computed as:

\[ \text{feature} = \text{count of 1s in the upper half} > \text{count of 1s in the lower half} \]

2.1.10 Set#2 - Left or Right?
This attribute is a Boolean that decides where more number of 1s occur (in a binary image) among the left and right halves of the pixels. And is computed as:

\[ \text{feature} = \text{count of 1s in the left half} > \text{count of 1s in the right half} \]

2.1.11 Set#2 - Count of protruding terminal pixels
This feature essentially decides how many pixels in the image are terminal. That is, it gives the count of pixels, for which, the number of 1s in its $3 \times 3$ neighborhood is 2 (including itself). Thus, it is computed as follows:

\[ \text{feature} = \text{pixels with } 3 \times 3 \text{ neighborhood counts 2} \]
2.1.12 Set#2 - Count of island pixels

This feature essentially decides how many pixels in the image are isolated. That is, it gives the count of pixels, for which, the number of 1s in its $3 \times 3$ neighborhood is 1 (including itself). Thus, it is computed as follows:

\[
\text{feature} = \text{pixels with } 3 \times 3 \text{ neighborhood counts } 1
\]

3 Pseudo Code

The primary code used in this assignment is listed in Appendix A. The code that is used from previous assignments or is given by Prof. Tom Henderson for his course CS 6350/5350 class is not listed in this document. This section gives pseudo codes for two of the major approaches employed by this assignment. Rest of the approaches are fairly straightforward and are not listed here. See Appendix A for full codes of these approaches.

PCA:

Center the feature points at their mean

Compute Covariance Matrix

Compute Eigen Values and Eigen Vectors

Transform the input data points into a lower dimensional space using the Eigen vectors

Best of Multiple Runs:

Run K-Means for $N$ independent iterations
For each classified training sample $S$
  For each of the $N$ iterations
    Extract the original class index to which it is classified
  End For
End For

Reset all cluster centers to zeros
For each classified training sample $S$
  NewClass = Mode of all the original class indices across the $N$ iterations
  Cluster(SNewClass).center += $S$
  Cluster(SNewClass).count += 1
End

For each new cluster
  Cluster(SNewClass).center /= Cluster(SNewClass).count
End

Intra And Inter Cluster Selection:

Run K-Means for $N$ iterations
For each of the $N$ iterations
  For each cluster $C$
    Intra-dist($C$) = sum of distances between each sample and center of $C$
  End For
  For each cluster pair $C_1$, $C_2$
Inter-dist(C1,C2) = distance between centers of C1 and C2
End For
End For

Choose the iteration that minimizes the \( \text{Sum(Intra-dist)}/\text{Sum(Inter-dist)}() \) to decide the final clustering

4 Verification of Program

As most of the code used in this document is verified in CS6350, Fall 2012 course lectures, no additional verification is provided. The following output shows the eigen values obtained after applying PCA on the discussed 18 feature dimensions of Feature Set #1 (see Section 2.1):

\[
> \text{diag(D)}
\]
\[
\text{ans} =
\]
\[
1.0e+03 *
\]
\[
6.5573
5.9237
3.1832
1.3994
0.9647
0.6583
0.3828
0.2426
0.1917
0.1314
0.0925
0.0740
0.0637
0.0407
0.0319
0.0209
0.0000
-0.0000
\]

5 Data

This section gives graphs that illustrate the overall performance of K-Means classifier and also show the factors affecting its performance.

PCA Reduction:

The following are the stats obtained with 90% confidence interval comparing the original percentages P with PCA percentages PPCA over K=9 runs.

\[
\text{Mean(P - PPCA)} = -2.9915
\]
\[
\text{Std(P - PPCA)} = 8.1084
\]
\[
\text{Confidence Interval} = [-8.0187, 2.0358]
\]
Figure 2: Figure shows average classification performance of K-Means on the original feature set vs PCA-reduced dimensionality feature set over K=9 runs. It also shows confidence intervals with 90% confidence.

Figure 3: Figure shows average classification performance of K-Means on different feature sets mentioned in Section 2.1 for K=9 runs. It also shows confidence intervals with 90% confidence.
Different Feature Sets:

The following are the stats obtained with 90% confidence interval comparing the each pair of feature sets f1, f2 and f3 over K=9 runs.

\[
\text{Mean}(p_{f1} - p_{f2}) = 17.9487 \\
\text{Std}(p_{f1} - p_{f2}) = 12.0096 \\
\text{Confidence Interval} = [10.5027, 25.3947]
\]

\[
\text{Mean}(p_{f1} - p_{f3}) = 14.9573 \\
\text{Std}(p_{f1} - p_{f3}) = 5.2470 \\
\text{Confidence Interval} = [11.7042, 18.2104]
\]

\[
\text{Mean}(p_{f2} - p_{f3}) = -2.9915 \\
\text{Std}(p_{f2} - p_{f3}) = 12.5778 \\
\text{Confidence Interval} = [-10.7897, 4.8067]
\]

where \(p_{f1}, p_{f2}\) and \(p_{f3}\) are the arrays of testing percentage accuracies of \(f_{1}, f_{2}\) and \(f_{3}\) respectively, obtained over \(K=9\) runs.

Cluster Center Selection Methods:

![KMeans Performance With Mean Vs Median Center Calculation](image)

Figure 4: Figure shows the comparison of classification percentage accuracies of K-Means with different cluster center calculation methods, namely, mean and median for \(K=9\) runs.

The following are the stats obtained with 90% confidence interval comparing the pair of cluster center methods mean and median over \(K=9\) runs.

\[
\text{Mean}(p_{\text{mean}} - p_{\text{median}}) = -1.5790e-15 \\
\text{Std}(p_{\text{mean}} - p_{\text{median}}) = 6.3781 \\
\text{Confidence Interval} = [-3.9544, 3.9544]
\]

\[
\text{Mean}(p_{\text{meantime}} - p_{\text{medianime}}) = -71.1111 \\
\text{Std}(p_{\text{meantime}} - p_{\text{medianime}}) = 33.3333
\]
Figure 5: Figure shows the comparison of training times of K-Means with different cluster center calculation methods, namely, mean and median for K=9 runs.

Confidence Interval = [-91.7778, -50.4444]

where pmean and pmedian are the arrays of testing percentage accuracies, pmeantime and pmedian time are the arrays of training times (in milli seconds) for mean and median methods respectively, obtained over K=9 runs.

**Best of Multiple Iterations:**

**Intra-Inter Cluster Distance Based Cluster Selection:**

The following are the stats obtained with 90% confidence interval comparing the pair of cluster selection methods, default and best of Intra-Inter Cluster Distances selected over K=9 runs.

\[
\begin{align*}
\text{Mean}(p_{\text{default}} - p_{\text{intraandinter}}) &= -0.8547 \\
\text{Std}(p_{\text{default}} - p_{\text{intraandinter}}) &= 5.3632 \\
\text{Confidence Interval} &= [-4.1799, 2.4705]
\end{align*}
\]

where pdefault and pintraandinter are the arrays of testing percentage accuracies obtained over K=9 runs for the default and Intra-Inter cluster selection methods respectively.

**Cluster Center Initialization:**

The following are the stats obtained with 90% confidence interval comparing the pair of cluster center initialization methods, random data space points and random feature space points over K=9 runs.

\[
\begin{align*}
\text{Mean}(\text{pranddata} - \text{prandfeature}) &= 10.2564 \\
\text{Std}(\text{pranddata} - \text{prandfeature}) &= 24.7024 \\
\text{Confidence Interval} &= [-5.0591, 25.5719]
\end{align*}
\]
Figure 6: Figure shows the average classification performance of K-Means based on the best of multiple iterations. This is repeated for K=9 runs. It also shows confidence intervals with 90% confidence.

Figure 7: Figure shows the comparison of classification graphs of samples with normal K-Means vs enhanced K-Means by taking best of 100 iterations for a given (one of K=9 runs) run.
Figure 8: Figure shows the performance comparisons of K-Means based on default clustering vs the clustering with best of Intra and Inter Clusters obtained over 10 iterations. This is repeated for K=9 runs. It also shows confidence 90% confidence intervals.

Figure 9: Figure shows the average classification performance of K-Means based on the type of cluster center Initialization Method for K=9 runs. X=1 represents random data point, while X=2 represents random feature point. It also shows confidence intervals with 90% confidence.
where randdata and randfeature are the arrays of testing percentage accuracies obtained over K=9 runs for the random data space and random feature space center intialization methods respectively.

Data Representation:

![K-Means Performance With Data Representation](image)

Figure 10: Figure shows the average classification performance performance of K-Means based on the input training data representation for K=9 runs. X=1 represents entire training data is considered, while X=2 represents average point of the training set is used. It also shows confidence intervals with 90% confidence.

The following are the stats obtained with 90% confidence interval comparing the pair of data selection methods, entire data set (default) and average representational set over K=9 runs.

Mean(p - pavg) = -17.0940
Std(p - pavg) = 4.3476
Confidence Interval = [-19.7896, -14.3985]

6 Analysis

Figure 2 shows the comparison between Non-PCA and PCA based K-Means. One can see that PCA does not detriment the performance of K-Means. In fact, the greater lower end of the comparison confidence interval suggests that, with 90% confidence that PCA will perform at least as good (if not better) as the Non-PCA case.

Figure 3 shows the comparison of the three different feature sets described in Section 2.1. One can see from the graph and the comparison confidence intervals that the Feature Set #1 outruns the other two feature sets, which is kind of expected as a combination of higher level features are expected to perform better than a mixture of lower to higher level features.

Figure 4 shows the accuracy comparison of the mean and median cluster center calculation methods, while Figure 5 shows the training time comparison of both the methods. One can see from the first graph that and the comparison confidence interval that we cannot statistically say one method is better than the other. But, the later graph suggests that we can say with 90% confidence that the Median method is slower than
the Mean method, which is expected as median method requires sorting the dimensional values every time a center is computed.

Figure 6 shows the comparison of getbest cluster selection method for different number of iterations. One can clearly see from the graph that there is a gradual increase in the average performance of the K-Means with increase in number of iterations. Also, as low as 10 iterations are sufficient to get a very good performance (consistently in the 90s). Figure 7 demonstrates this fact. As, we know that the ideal classification graph for our data set is a step function, the blue lines show the classification graph of a best of 100 instance, while the red lines show a typical default run with only 1 iteration. One can clearly see that the blue curve is almost ideal except for a couple of misclassifications, while the red curve has significant number of misclassifications, suggesting again the improved result with multiple iterations using the getbest approach.

Figure 8 shows the comparison of intraandinter cluster selection method over 10 iterations vs the default selection method over a single iteration. One can see from the confidence intervals that there is no statistically significant improvement with the intraandinter cluster method over the default method. Though, the average performance seems to be a little better, it looks more inconsistent with higher standard deviation. This is kind of expected as the method does not change the actual clusters of any run. So, the best one can expect out of this method is to select the run with the maximally performing cluster set, but cannot be expected to improve over any of the individual runs by too much.

Figure 9 shows the comparison of two different cluster center initialization methods (random data vs random features). In the random data method, k random data points are selected as cluster centers, while in the random feature method, k random feature points are selected as cluster centers. That is, for each dimension, a random feature value is selected and this whole process is repeated to get k random feature points. One can clearly see that, the average performance of random data point method is better than that of the random feature method. Also, the former is more consistent, while the latter suffers from a high standard deviation. The larger positive range of the comparison confidence interval also suggests that selecting a data point is mostly better than selecting a feature point as cluster center. This can be explained by the fact that, selecting a random feature point can result in completely random values that might not even occur in any of the classes.

Figure 10 shows the comparison of the default case in which entire data set is selected vs what we can call the ideal case in which the average representational set is selected as the training data. That is because, in the latter case, only the means of all training samples are part of the input data set, making the k=26 cluster centers to be one of the 26 input mean samples. One can see the results of such an experiment in the graph and the reported comparison confidence interval, and can clearly notice that, as expected, average representational set outruns the default representational set.

7 Interpretation

The answers to the questions posed are as follows:

1. Is K-Means good enough for this task?
   K-Means by itself is just acceptable for this task. But, with the adaptations discussed in the Section 6, it can perform very well for this task.

2. What is the effect of PCA on the overall K-Means performance?
   PCA seems to increase the overall performance in general and also reduces the redundant feature dimensions in the training data. So, we can say that PCA is actually very useful before applying K-Means.

3. What is a good set of features?
   A set of high level features seems to be very good for this task.
4. What is a good way for calculating cluster centers?
   Though the classification performance doesn’t differ much between mean and median cluster center methods, mean method is faster and probably more feasible for this task.

5. What are good stopping criteria for K-Means?
   Getting best of multiple runs seems to be a very good stopping criteria (or final cluster selection method) over other methods.

6. What is a good way to initialize the cluster centers?
   Selecting random data points as initial cluster centers seems to perform well.

7. What is a good way to select training and testing data?
   If the training data is available, selecting the average representation does very well. As this is often not the case for K-Means, the default case is not that bad either.

8 Critique

Overall, I learnt that, despite being unsupervised, K-Means can perform very well if a proper feature set is chosen. Additional measures like reducing feature dimensionality using PCA, selecting the best of multiple runs will augment the classification performance. If I had more time, I would have explored when median will do better than mean as the cluster center method. Probably, for more noisy data sets with outlier points, median method will do better than the mean case. Also, a better approximation of calculating the median can also improve the observed results. I would also have tried to improve the intraandinter cluster selection method by doing a crossing over of clusters across multiple iterations, instead of just selecting the unchanged cluster set from one of the iterations. More things to try would be the other stopping criteria like distance to the Voronoi boundary, hyper-parallelepiped based cluster center initialization method etc.

9 Log

Time spent on coding:
- November 2nd, 2012 from 5 PM to 8 PM

Time spent on writing the report, generating the results, analyzing the various relationships, coding and conducting experiments:
- November 4th, 2012 from 6 PM to 9 PM
- November 5th, 2012 from 6 PM to 9 PM
- November 6th, 2012 from 9 PM to 12 AM
- November 7th, 2012 from 3:30 PM to 5 PM

A Code

Only the code most relevant to this assignment is listed here. Rest of the code not listed here is already listed in the previous reports or is from Prof. Tom Henderson’s code base for CS 6350/5350 class, Fall 2012.

%File Name: kmeans_main_multiple_runs.m
%Date: November 06, 2012

function [percent_correct] = kmeans_main_multiple_runs(clusterMethod, stopMode,...
n_runs, initMethod, pcaEnabled)
%
% kmeans_main_multiple_runs - customizable kmeans classifier
% On input:
%   clusterMethod: ’mean’ vs ’median’ that specifies
how cluster centers are calculated
stopMode: string specifying the stopping criteria for the classification
'getbest' – indicates get best out of multiple runs
'intraandinter' – indicates choose the cluster set that
minimizes the ratio
sum of intra cluster distances of all clusters /
sum of inter cluster distances for all cluster pairs
n_runs: integer specifying the number of iterations
KMeans has to be carried for
initMethod: string indicating the type of cluster center initialization
'random_data' – indicates random point in data space
'random_feature' – indicates random point in feature space
pcaEnabled: Boolean value indicating if PCA has to be used to reduce
the dimensionality of input data
percent_correct: array of real numbers indicating the percentage testing
accuracy at each of the K=9 runs
Call:
p = kmeans_main_multiple_runs('mean','getbest',1,'random_data',0);
Author:
Srivishnu Kaushik Satyavolu
University of Utah
Fall 2012

Create some features on the data
features = cs6350_NN_image_features;

Get the size of the feature vector
[m, n, p] = size(features);

Organize features as training data
[X, targets] = CS5350_MLP_data_prep(features);

PCA Reduction
if pcaEnabled
X0 = 0*X;
for f = 1:p
X0(:,f) = X(:,f) - mean(X(:,f));
end
sum(X0);
C = X0'*X0;
[V,D] = eig(C);
sort(diag(D),'descend')
V;
V*V';
X0_rot = V'*X0';
X0_rotation_point = X0_rot';
X = X0_rotation_point(:,3:p);
X(1:18,:);
p = p - 2;
end
[X,targets] = CS6350_DT_image_features_mod;
% [X,targets] = CS5350_Adaboost_image_features;

%m = 26;
%n = 9;
%p = size(X,2);

% Split Data into Training and Testing Data sets
data = split_data(X, targets, m, size(X,1)/m, p);
n_trials = length(data);
h_waitbar = waitbar(0,'Ensemble Learning');

for trial_ind = 1:n_trials
    waitbar(trial_ind/n_trials);

    curTrainingData = data(trial_ind).train_X;
    curTrainTargetData = data(trial_ind).train_targets;
    curTestTargetData = data(trial_ind).test_targets;

    curTrainData = data(trial_ind).train_X;
    curTrainTargetData = data(trial_ind).train_targets;
    curTestTargetData = data(trial_ind).test_targets;

    % Perform training
    nSamples = size(curTrainingData,1)/m;
    clear run_stats;

    % Collect Run statistics for each run
    for run_number = 1:n_runs
        if strcmp(clusterMethod, 'mean')
            % train_time(trial_ind,1) = cputime;
            [run_stats(run_number).segs,...
             run_stats(run_number).clusters] = ...;
            CV_kmeans(curTrainingData, 'CV_Euclidean', m,...
                       initMethod);
            % train_time(trial_ind,1) = cputime -
            % train_time(trial_ind,1);
            elseif strcmp(clusterMethod,'median')
                % train_time(trial_ind,1) = cputime;
                [run_stats(run_number).segs,...
                 run_stats(run_number).clusters] = ...;
                CV_kmedians(curTrainingData, 'CV_Euclidean', m);
                % train_time(trial_ind,1) = cputime -
                train_time(trial_ind,1);
            else
                disp('ERROR: Choose a valid cluster method');
                break;
            end
        end

        %plot(sort(run_stats(run_number).segs));
for k = 1:m
    run_stats(run_number).clusterData(k).indices = []; 
    run_stats(run_number).clusterData(k).targetIndices... 
    = []; 
end 

for k = 1:m*nSamples
    run_stats(run_number).clusterData(... 
    run_stats(run_number).segs(k,1)).indices(end+1,1) = k; 
    run_stats(run_number).clusterData(... 
    run_stats(run_number).segs(k,1)).targetIndices(... 
    end+1,1) = curTrainTargetData(k,1); 
end 

for k = 1:m
    run_stats(run_number).clusterData(k).classIndex =... 
    mode(run_stats(run_number).clusterData(k).targetIndices... 
); 
end 
end 

for k = 1:m
    clusters(k).center = zeros(1,p); 
    clusters(k).classSize = 0; 
end 

if strcmp(stopMode, 'getbest')

    for sample_ind = 1:m*nSamples
        temp_array = []; 
        for run_number = 1:n_runs 
            temp_array(run_number,1) = ... 
            run_stats(run_number).clusterData(run_stats(... 
            run_number).segs(sample_ind,1)).classIndex; 
        end 
        segs(sample_ind,1).finalClassIndex = mode(temp_array); 
        %debugArray(sample_ind,1) = ... 
        %segs(sample_ind,1).finalClassIndex; 
    end 

    plot(debugArray,'b'); 

    for sample_ind = 1:m*nSamples
        clusters(segs(sample_ind,1).finalClassIndex).center... 
        =clusters(segs(sample_ind,1).finalClassIndex).center... 
        +curTrainingData(sample_ind,:); 
        clusters(segs(sample_ind,1).finalClassIndex... 
        ).classSize = clusters(... 
        segs(sample_ind,1).finalClassIndex).classSize + 1; 
        end 
    for k = 1:m
clusters(k).center = clusters(k).center /...
end

elseif strcmp(stopMode, 'intraandinter')

% Choose the clustering that minimizes the ratio
% % sum of intra cluster distances of all clusters / % sum of inter cluster distances for all cluster pairs

minInd = -1;

for run_number = 1:n_runs

intraDistAvg = 0.0;
interDistAvg = 0.0;

for sample_ind = 1:m*nSamples
clusterIndex =...
run_stats(run_number).segs(sample_ind,1);
intraDistAvg = intraDistAvg +...
end

end

intraDistAvg = intraDistAvg/m*nSamples;

for k1 = 1:m
for k2 = k1+1:m
interDistAvg = interDistAvg +...
end
end

interDistAvg = interDistAvg/(m*(m+1)/2);
curFraction = intraDistAvg / interDistAvg;

if minInd == -1 || curFraction < minFraction + eps
minFraction = curFraction;
end
end
end

for k = 1:m
clusters(run_stats(minInd).clusterData(k).classIndex...).
center = run_stats(minInd).clusters(k,:);
end

else
disp('ERROR: Choose a valid Stopping criteria');
end
% Perform Testing
n_test_samples = size(curTestingData,1);
percent_correct(trial_ind,1) = 0;
for k = 1:n_test_samples
    clusterInd = classify_sample2(clusters, curTestingData(k,:));
    if clusterInd == curTestTargetData(k,1)
        percent_correct(trial_ind,1) =...
    end
end
percent_correct(trial_ind,1) = percent_correct(trial_ind,1) *100 /...
end
close(h_waitbar);
end

% File Name: classify_sample2.m
% Date: November 06, 2012

function minInd = classify_sample2(clusters, sample_X)
%
% classify_sample2 - recall function that classifies a given sample
% On input:
% clusters: array of structures that contain cluster centers
% sample_X: 1 by p dimension test sample
% On output:
% minInd: index of the cluster center closest to sample_X
% Call:
% classIndex = classify_sample2(clusters, sample_X);
% Author:
% Srivishnu Kaushik Satyavolu
% University of Utah
% Fall 2012

nClasses = length(clusters);

minD = norm(clusters(1).center - sample_X);
minInd = 1;

% Find the closest cluster center
for k = 2:nClasses
    curD = norm(clusters(k).center - sample_X);
    if(curD < minD + eps)
        minD = curD;
        minInd = k;
    end
end
end

% File Name: CV_kmeans.m
% Date: November 06, 2012

function [segs,clusters] = CV_kmeans(pts,f_dist_func,k,initMethod)
CV_kmeans - K-means clustering

On input:
%  pts (mxn array): data points
%  f_dist_func (string): name of distance function
%  k (int): number of clusters

On output:
%  segs (mx1 vector): segmented (distinct integer for each class)
%  clusters (kxn array): center value (mean) for each cluster

Call:
%  [s,c] = CV_kmeans(pts,CV_Euclidean,4);

Author:
%  Tom Henderson
%  UU
%  Fall 2004
%  Modified: Fall 2012
%

MAX_D = 2;

[num_pts,num_dim] = size(pts);
clusters = zeros(k,num_dim);

if strcmp(initMethod,'random_data')
  indexes = randperm(num_pts);
  clusters = pts(indexes(1:k),:);
elseif strcmp(initMethod,'random_feature')
  % Choose k random feature points as cluster centers
  for cInd = 1:k
    indexes = randperm(num_pts);
    for dim = 1:num_dim
      clusters(cInd, dim) = pts(indexes(dim),dim);
    end
  end
else
  disp('ERROR: Choose a valid Init Method for choosing cluster centers');
  exit;
end

unchanged = 0;

while (unchanged==0)
  counts = zeros(k,1);
  sums  = zeros(k,num_dim);
  for p = 1:num_pts
    dist = zeros(k,1);
    for cl = 1:k
      dist(cl) = norm(clusters(cl,:)-pts(p,:));
    end
    [val,index] = min(dist);
    sums(index,:) = sums(index,:) + pts(p,:);
    counts(index) = counts(index)+1;
  end
  for cl = 1:k
    % Update cluster centers
    % ...
if (counts(cl)==0)
    counts(cl) = 1;
    sums(cl,:) = pts(max(1,min(num_pts,round(rand*num_pts))),:);
end
for cl = 1:k
    new_clusters(cl,:) = sums(cl,:)/counts(cl);
end
d = sum(norm(new_clusters-clusters));
if (d<MAX_D)
    unchanged = 1;
else
    clusters = new_clusters;
end
end

segs = zeros(num_pts,1);
for p = 1:num_pts
    dist = zeros(k,1);
    for cl = 1:k
        dist(cl) = norm(clusters(cl,:)-pts(p,:));
    end
    [val,index] = min(dist);
    segs(p) = index;
end

% File Name: CV_kmedians.m
% Date: November 06, 2012

function [segs,clusters] = CV_kmedians(pts,f_dist_func,k)
    
    CV_kmeans - K-Medians clustering
    On input:
    pts (mxn array): data points
    f_dist_func (string): name of distance function
    k (int): number of clusters
    On output:
    segs (mx1 vector): segmented (distinct integer for each class)
    clusters (kxn array): center value (median) for each cluster
    Call:
    [s,c] = CV_kmedians(pts,CV_Euclidean,4);
    Author:
    Srivishnu Kaushik Satyavolu
    University of Utah
    Fall 2012
    Original Author:
    Tom Henderson
    University of Utah
    Fall 2004
    Modified: Fall 2012
%

MAX_D = 2;

[num_pts,num_dim] = size(pts);
clusters = zeros(k,num_dim);

indexes = randperm(num_pts);
clusters = pts(indexes(1:k),:);

unchanged = 0;

for cl = 1:k
    clusterData(cl).points = [];
end

while (unchanged==0)
    counts = zeros(k,1);
    sums = zeros(k,num_dim);

    for p = 1:num_pts
        dist = zeros(k,1);
        for cl = 1:k
            dist(cl) = norm(clusters(cl,:)-pts(p,:));
        end
        [val,index] = min(dist);
        sums(index,:) = sums(index,:) + pts(p,:);
        clusterData(index).points(end+1,:) = pts(p,:);
        counts(index) = counts(index)+1;
    end
    for cl = 1:k
        if (counts(cl)==0)
            counts(cl) = 1;
            sums(cl,:) = pts(max(1,min(num_pts,round(rand*num_pts)),:),:);
            clusterData(cl).points(end+1,:) = sums(cl,:);
        end
    end
    for cl = 1:k
        new_clusters(cl,:) = sums(cl,:)/counts(cl);
    end
    for dim = 1:num_dim
        new_clusters(cl,dim) = median(clusterData(cl).points(:,dim));
    end
    d = sum(norm(new_clusters-clusters));
    if (d<MAX_D)
        unchanged = 1;
    else
        clusters = new_clusters;
    end
end

segs = zeros(num_pts,1);
for p = 1:num_pts
    dist = zeros(k,1);
    for cl = 1:k
        dist(cl) = norm(clusters(cl,:)-pts(p,:));
    end
    [val,index] = min(dist);
    segs(p) = index;
end

References