Data mining: Homework 4

vishvAs vAsuki

November 13, 2009

1 1

```
function [partition] = kmeans(X, numClasses)
    % Get random paritioning
    numPoints = size(X, 1);
    partition = getRandomPartitioning(numPoints,
        numClasses);
    obj = getObjective(X, partition);
    improvement = 1;

    while improvement > .001
        partition = repartition(X, partition);
        newObj = getObjective(X, partition);
        improvement = obj - newObj;
        obj = newObj;
%           fprintf(1,'Obj, improvement: %d %d\n', obj,
    improvement);

    end
    partition = ClusterUtils.renameClusters(partition);
end

function partition = repartition(X, partition)
    numClasses = numel(unique(partition));
    numPoints = size(X, 1);
    DistancesFromMean = zeros(numPoints, numClasses);
    for c=1:numClasses
        clusterMembers = find(partition == c);
        numClusterMembers = numel(clusterMembers);
        clusterMean = sum(X(clusterMembers,:), 1)/
            numClusterMembers ;
        for i=1:numPoints
            DistancesFromMean(i, c) = Metrics.
                klDivergence(X(i,:), clusterMean);
```

```
        end
    end
    [y, partition] = min(DistancesFromMean,[],2);
end

function obj = getObjective(X, partition)
    numClasses = numel(unique(partition));
    numPoints = size(X, 1);
    obj = 0;
    for c=1:numClasses
        clusterMembers = find(partition == c);
        numClusterMembers = numel(clusterMembers);
        clusterMean = sum(X(clusterMembers,:), 1)/
            numClusterMembers ;
%           fprintf(1,'Cluster: %d\n', c);
        for i=1:numClusterMembers
            obj = obj + Metrics.klDivergence(X(i,:),
                clusterMean);
%               fprintf(1,'  Update: %d %d\n', Metrics.
    klDivergence(X(i,:), clusterMean), obj)
        end
    end
end

function partition = getRandomPartitioning(numPoints,
    numClasses)
    partition = zeros(numPoints, 1);
    for i=1:numClasses
        unPartitionedPts = find(partition == 0);
        selectedPts = randomizationUtils.Sample.
            sampleWithoutReplacement(unPartitionedPts,
            numPoints/numClasses);
        partition(selectedPts) = i;
    end
end
```

1.1 Related Functions

```
classdef Sample
methods(Static = true)
    function sample=sampleWithoutReplacement(inputVector,
        sampleSize)
        % Input: 1. the vector to sample from. 2. the
            size of the sample to draw.
        % Output: the sample drawn.
        inputVector = MatrixTransformer.getColumnVector(
```

```
            inputVector);
        inputVector = inputVector';
        perm = randperm(numel(inputVector));
        sample = inputVector(perm(1:sampleSize));
    end
end
end
```

2 2

```
classdef Agglomerative
methods(Static=true)
    function partition = singleLink(X, numClasses)
        numPoints = size(X, 1);
        partition = 1:numPoints;
        numPartitions = numel(unique(partition));
        while numPartitions > numClasses
%             fprintf(1, '%d ', partition');
%             display ''
%             Calculate distances
            InterPointDistance = ClusterUtils.
                getInterPointDistance(X);
            InterClusterDistance = ClusterUtils.
                getInterClusterDistanceMin(
                InterPointDistance, partition);

%             Identify clusters to merge
%             Fix InterClusterDistance to have large
    diagonal elements.
            InterClusterDistance = InterClusterDistance +
                diag(sum(InterClusterDistance));
            [minValue, linIndex] = min(
                InterClusterDistance(:));
            [cluster_i, cluster_j] = ind2sub([
                numPartitions, numPartitions], linIndex);

%             Merge clusters.
            classes = unique(partition);
            clusterLabel_i = classes(cluster_i);
            clusterLabel_j = classes(cluster_j);
            clusterMembers_i = find(partition==
                clusterLabel_i);
            partition(clusterMembers_i) = clusterLabel_j;
%             fprintf(1, 'Merged Clusters %d and %d \n',
     clusterLabel_i, clusterLabel_j);
```

```
            numPartitions = numel(unique(partition));
        end
        partition = ClusterUtils.renameClusters(partition
            );
    end

end
end
```

2.1 Related Functions

```
classdef ClusterUtils
methods(Static=true)
    function [confusionMatrix] = getConfusionMatrix(
        partition1, partition2)
        numClasses = numel(unique(partition1));
        confusionMatrix = zeros(numClasses, numClasses);
        for i=1:numClasses
            for j=1:numClasses
                confusionMatrix(i, j) = sum((partition1
                    == i).*(partition2 == j));
            end
        end
        display 'Columns may need to be transposed
            appropriately.';
    end

    function InterPointDistance = getInterPointDistance(X
        )
        numPoints = size(X, 1);
        InterPointDistance = zeros(numPoints,numPoints);
        for i = 1:numPoints
        for j = 1:numPoints
            InterPointDistance(i, j) = Metrics.
                klDivergence(X(i,:), X(j,:));
        end
        end
    end

    function InterClusterDistance =
        getInterClusterDistanceMin(InterPointDistance,
        partition)
        numPoints = size(InterPointDistance, 1);
        classes = unique(partition);
        numClasses=numel(classes);
        InterClusterDistance = zeros(numClasses,
```

```
            numClasses);
        for i = 1:numClasses
            clusterLabel_i = classes(i);
            clusterMembers_i = find(partition ==
                clusterLabel_i);
            for j = 1:numClasses
                clusterLabel_j = classes(j);
                clusterMembers_j = find(partition ==
                    clusterLabel_j);
%                   fprintf(1, 'InterClusterDistance:
    comparing: %d %d %d %d \n', i, j, clusterLabel_i,
    clusterLabel_j);
                InterClusterDistance(i, j) = min(min(
                    InterPointDistance(clusterMembers_i,
                    clusterMembers_j)));
            end
        end
    end

    function [partition] = renameClusters(oldPartition)
        classes = unique(oldPartition);
        numClasses=numel(classes);
        partition = oldPartition;
        for i = 1:numClasses
            clusterLabel_i = classes(i);
            clusterMembers_i = find(oldPartition ==
                clusterLabel_i);
            partition(clusterMembers_i) = i;
        end
    end

end
end
```

3 3

3.1 Notation

Let number of classes = c. Let number of data points = N. Let dimension of the data = D. Let number of iterations = k.

3.2 Some observations

Time taken to calculate distance between 2 D dimensional vectors is O(D).

Time taken to find minimum of n numbers: O(n).

3.3 Experiment code

```
%  load iris;
%
%  %  Normalize data
%  X = inv(diag(sum(X, 2)))*X;
%  numClasses = numel(unique(classid));

tic
partitionKMeans = kmeans(X, numClasses);
toc
display 'kmeans confusion matrix'
confusionMatrixKMeans = ClusterUtils.getConfusionMatrix(
    partitionKMeans, classid)

%  tic
%  partitionAgglomerative = Agglomerative.singleLink(X,
    numClasses);
%  toc
%  display 'Agglomerative clustering confusion matrix'
%  confusionMatrixAgg = ClusterUtils.getConfusionMatrix(
    partitionAgglomerative', classid)
%
```

3.4 k-means clustering

k-means is highly sensitive to the initial conditions. So, different initial partitionings lead to clusterings of different qualities.

3.4.1 Confusion matrix

```
confusionMatrixKMeans =

50.0000e+000     0.0000e-003     0.0000e-003
0.0000e-003    13.0000e+000     0.0000e-003
0.0000e-003    37.0000e+000    50.0000e+000
```

3.4.2 Observed Running time

0.171957 seconds.

3.4.3 Theoretical running time

A worst case analysis follows. Time taken to find mean of a cluster: O(ND). Time taken to find means of c clusters: O(NDc). Time taken to find distances of each point to c means: O(NDc).

The above is repeated during all of the k iterations. So, the total running time is O(kNDc).

3.5 Agglomerative clustering

3.5.1 Confusion matrix

```
confusionMatrixAgg =

49.0000e+000      0.0000e-003      0.0000e-003
1.0000e+000      0.0000e-003      0.0000e-003
0.0000e-003     50.0000e+000     50.0000e+000
```

3.5.2 Observed Running time

421.530759 seconds.

3.5.3 Theoretical running time

A worst case analysis follows.

Time taken to compute the inter-point distance matrix = $O(N^2D)$.

Calculating distance between all possible pairs of clusters, in any iteration, involves finding the minimum amongst $\leq N^2$ numbers, so, this operation costs $O(N^2)$ time. Number of iterations is k $= N - c$.

So, total operation count is $O(N^2D + N^2(N - c)) = O(N^2(N + D - c))$.

4 4

$S_T = \sum_x d_f(x, y), S_B = \sum_{c=1}^{K} N_c d_f(m_c, m), S_W = \sum_{c=1}^{K} \sum_{x \in \pi_c} d_f(x, m_c)$
where $m = N^{-1} \sum_x x,\ m_c = N_c^{-1} \sum_{x \in \pi_c} x$.

$$
\begin{aligned}
m &= N^{-1} \sum_{x} x \\
&= N^{-1} \sum_{c=1}^{K} \sum_{x \in \pi_c} x \\
&= N^{-1} \sum_{c=1}^{K} N_c m_c \\
&= N^{-1} \sum_{c=1}^{K} \sum_{x \in \pi_c} m_c \\
\therefore \sum_{x \in \pi_c} \sum_{c=1}^{K} (m_c - m) &= 0 \\
\sum_{x \in \pi_c} (x - m_c)^T &= N_c(m_c - m_c) = 0 \\
\sum_{x} (x - m)^T &= N(m - m) = 0
\end{aligned}
$$

We use this below.

$$
\begin{aligned}
S_B + S_W &= \sum_{c=1}^{K} N_c d_f(m_c, m) + \sum_{c=1}^{K} \sum_{x \in \pi_c} d_f(x, m_c) \\
&= \sum_{c=1}^{K} \sum_{x \in \pi_c} d_f(m_c, m) + \sum_{c=1}^{K} \sum_{x \in \pi_c} d_f(x, m_c) \\
&= \sum_{x \in \pi_c} \sum_{c=1}^{K} (d_f(m_c, m) + d_f(x, m_c)) \\
&= \sum_{x \in \pi_c} \sum_{c=1}^{K} f(m_c) - f(m) + f(x) - f(m_c) \\
&\quad -(m_c - m)^T \nabla f(m) - (x - m_c)^T \nabla f(m_c) \\
&= \sum_{x \in \pi_c} \sum_{c=1}^{K} f(x) - f(m) \\
&= \sum_{x} f(x) - f(m) \\
&= \sum_{x} f(x) - f(m) - (x - m)^T \nabla f(m) \\
&= \sum_{x} d_f(x, m) \\
&= S_T
\end{aligned}
$$