Module 5
Clustering

VPAC Education Grant Round 2
EDPNMO006/2001
David Taniar and Kate Smith
Monash University
Clustering- Introduction

- **Clustering analysis** - Finding groups in data; formal study of algorithms and methods for grouping, or classifying, objects (Jain, Dubes 1988).
- **Cluster** - comprised of a number of *similar* objects collected or grouped together from *data space*.
- Does not use category labels that tag objects with prior identifiers (thus unsupervised training).
- **Discriminant analysis** - given unlabeled pattern, assign it to a pre-classified patterns.
- Human are unable to cluster *data space* with high dimensionality (e.g. > 3).

- Clustering is the process of finding groups for data. The formed groups are normally called clusters.
- Clustering is an unsupervised training method, where the target label of a training example is unknown.
- The clustering algorithm try to form groups from these data points characteristics.
- After forming the clusters, new data point can be added to a cluster my comparing the new point with the characteristic of the existing clusters.
Clustering - Introduction

Examples,
- Cluster customer according to their buying behaviors.
- Cluster students based on their exam marks, gender, height etc.
- Cluster pattern of protein/DNA/RNA sequences.

2 clusters formed by weight and height.

- Above gave an illustration on the application area of clustering.
- In the diagram, the clustering algorithm formed the boundaries for the two classes. A good cluster creates a boundaries that does not overlap with other cluster and contains most of the it members.
Clustering (1)

**Clustering process**

1. Pattern representation (optionally including feature extraction and/or selection),
2. Definition of a pattern proximity measure appropriate to the data domain,
3. Clustering or grouping,
4. Data abstraction (optional),
5. Assessment of output (validity analysis).

- The clustering process can be generally divided into 5 steps. The first step is the preprocessing step which involves features extraction and feature selection. The purpose of features extraction is to extract suitable representative values for the problem domain. The feature selection is to select the most useful and non-redundant features among the extracted features.
Clustering (2)

- **Pattern representation** refers to the number of classes, the number of available patterns, and the number, *type*, and *scale* of the features available to the clustering algorithm.

- **Feature selection** - process of identifying the most effective subset of the original features to use in clustering. Some of the features may be redundant (e.g. constant value).

- **Feature extraction** - transformations of the input features into new representation features. Can use for dimension reduction.
Clustering (3)

- **Pattern proximity** - measured the similarity of dissimilarity between two pattern pairs (using some distance measures).
- **Data abstraction** - the process of extracting a simple and compact representation of a data set.
Clustering - definitions (1)

- **Pattern** - a single data item \( x \) used by the clustering algorithm. It consists of a vector of \( d \) measurements: 
  \[ x = (x_1, x_2, \ldots, x_d). \] 
  \( d \) is the dimensionality of the pattern space.

- **Features** - an individual measurement from a pattern \( x \) (i.e. \( x_i \)).

- **Pattern matrix** - an \( n \times d \) matrix that consisted of a set of \( n \) patterns.
Clustering - definitions (2)

- **Distance measures** - a metric on the feature space used to quantify the similarity of patterns.
- **Proximity matrix** - an $n \times n$ matrix accumulates the pairwise distance measure between two patterns in which each row and column represents a pattern.
The input features measurement needs to be scaled properly before apply to the clustering algorithm. This ensure that the features are represent in a meaningful way that can enhance the resulting cluster.

The features can be either qualitative of quantitative measure. Example of qualitative measures are Nominal (e.g. color) and Ordinal (e.g. cool, hot). Examples of Quantitative measures are Interval (e.g. 1 to 7), Ratio (e.g. temperature with reference value 0) and discrete numbers.

The scaling methods results in qualitative or quantitative measurements. Different data type may needs different scaling treatment.
Clustered - pattern representation (2)

- Continuous values - positive or negative real numbers, such as height, weight, temperature, age etc.
- Interval values - continuous value limited by upper and lower values (e.g. time measure).
- Discrete values - finite, usually small, number of possible values (e.g. 8 bits pixel values).
- Nominal - number values use as a name, order is not important (e.g. 1-red, 2-blue etc).
- Ordinal - similar to nominal, but the ordered is meaningful (e.g. rate a food taste by scale 1 to 5).
Clustering - pattern representation (3)

- Ratio - positive numbers that have an absolute meaning (does not depend on the measurement units).
- Most clustering techniques (e.g. hierarchical and partitioning) only applicable to numerical data.
- Different measurement unit may have different weight on the clustering result (e.g. height measured in centimeters and meter).
Clustering - pattern representation (4)

- Needs to standardize the data by converting the original measurements to unitless variables (sometimes may not be desirable, because it may distort the natural cluster that may existed in the data).
- \( z\)-scores - standardized value of a feature in a pattern.
Clustering - pattern representation (5)

\[
\begin{align*}
(x_{1}, \ldots, x_{r}, \ldots, x_{p}) & \quad \bar{m}_r = \frac{1}{n}(x_{1r} + x_{2r} + \ldots + x_{nr}) \quad (1) \\
& \quad s_r = \frac{1}{n-1} \left[ (x_{1r} - \bar{m}_r)^2 + (x_{2r} - \bar{m}_r)^2 + \ldots + (x_{nr} - \bar{m}_r)^2 \right] \quad (2) \\
& \quad z_{iq} = \frac{x_{iq} - \bar{m}_q}{s_q} \quad (3)
\end{align*}
\]

(1) - means for variable \( f \),
(2) - standard deviation for variable \( f \), and
(3) - z-score for variable \( f \) of pattern \( i \).
Clustering - Pattern similarity (1)

- Measure similarity between two patterns drawn from the same feature space.
- Given a distance measure $d$ between two patterns $i_{th}$ and $j_{th}$, the properties of $d(i,j)$ that must satisfy are:
  1. (a) For a dissimilarity: $d(i, i) = 0$, all $i$,
     (b) For a similarity: $d(i, i) \geq \max d(i, j)$, all $i, i \neq j$,
  2. $d(j, k) = d(j, i)$, all $(i, j)$,
  3. $d(i, j) \geq 0$, all $(i, k)$.

- All clustering algorithm must use some kind of distance measure between two patterns. The distance measure use must comply with the three properties outlined above.
- In the three conditions above, $d$ is the distance measure method.
Clustering - Pattern similarity (2)

Euclidean distance

\[ d_2(i, j) = \left( \sum_{k=1}^{p} (x_{ik} - x_{jk})^2 \right)^{1/2} \quad (4) \]

\[ = \left\| x_i - x_j \right\|_2 \quad p\text{-input dimension} \]

Manhattan distance

\[ d_2(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ip} - x_{jp}| \quad (5) \]

- Two commonly use methods for distance measure is the Euclidean distance and the Manhattan distance. Other distance measure may better suit some problem domain.
Clustering - Pattern similarity (1)

Binary data type

If all the features are of binary type, we can use Jaccard coefficient for distance measure,

\[
\begin{array}{cccc}
 & 1 & 0 \\
1 & a_{11} & a_{10} \\
0 & a_{01} & a_{00} \\
\end{array}
\]

\[
d_2(i, j) = \frac{a_{11}}{a_{11} + a_{01} + a_{10}} = \frac{a_{11}}{d - a_{00}} \tag{6}
\]

- The Jaccard coefficient can be used to calculate distance measure for binary input features.
Clustering - Pattern similarity (2)

<table>
<thead>
<tr>
<th>Feature Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{x}_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \mathbf{x}_2 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Jaccard coefficient:

\[
\frac{8}{13} = 0.615
\]

Above gave an example of using Jaccard distance measure for two input patterns.
Two main types

a) Hierarchical - agglomerative or divisive

b) Partitional - squared error (k-means), graph theoretic, mixture resolving (Expectation resolving), mode seeking.

Hierarchical

- do not construct a single partition with k clusters, but they deal with all values of k in the same run. That is, the partition with k=1 is part of the output, and also the situation with k = n (k is number of cluster, n is number of input pattern).

• Two main category of clustering algorithm are the Hierarchical and Partition method.
• The hierarchical clustering method construct the cluster either by merging (agglomerative) the clusters or by splitting (divisive) the clusters in successive steps.
• The hierarchical agglomerative method can be either the single-link, complete-link or minimum-variance. The single-link and complete link algorithms differ in the way they characterize the similarity between a pair of clusters.
• The partitional algorithm is an iterative method with a stopping criterion. It start with an initial clusters and assign each pattern to a cluster. At each iteration the clusters center are updated and the patterns are reassigning to the nearest cluster.
Clustering - Clustering algorithms  

- in \( k = r + 1 \), one of the \( r \) clusters in \( k = r \) is split up to obtain \( r + 1 \) clusters.
- in agglomerative methods, start with \( k=n \) clusters. Then in each step two clusters are merge until only one cluster left.
- divisive- start with \( k=0 \) and in each following step a cluster is split up, until there are \( n \) of them.

• Figure below shows the Dendrogram diagram result from clustering the 7 input patterns using single-link hierarchical agglomerative algorithm. Three clusters were formed (dotted line).

• The Hierarchical Agglomerative clustering algorithm are performed through the following steps
  
  (1) Compute the proximity matrix containing the distance between each pair of patterns. Three each pattern as a cluster.
  
  (2) Find the most similar pair of clusters using the proximity matrix. Merge these two clusters into one cluster. Update the proximity matrix to reflect this merge operation.
  
  (3) If all patterns are in one cluster, stop. Otherwise, go to step 2.
### Clustering - Clustering algorithms (3)

**Partitional**
- Constructs $k$ clusters from input pattern; $k$ is supplied by the users.
- Each cluster must contain at least one object.
- Each object must belong to exactly one group.
- Difficult to determine the value of $k$ ($k$ might not be the same as the available “natural” clusters).

- The number of clusters formed by the partitional algorithm is fixed by the user. Some partitional algorithms allow dynamic determination of the number of clusters by setting the merging or splitting threshold.
Clustering - Clustering algorithms (4)

Square-Error clustering

- objective - minimize squared error with fixed k partition.

- Given k clusters \{C_1, C_2, ..., C_k\} such that cluster \(C_i\), \(1 \leq i \leq k\) has \(n_i\) patterns and each pattern is in exactly one cluster. The mean \((m_i)\) vector \((n_i \times 1)\) for cluster \(C_i\) is defined as the centroid of the cluster,

\[
m^i = \left(\frac{1}{n_i}\right) \sum_{j=1}^{n_i} x^i_j, \quad x^i \in C_i \quad (7)
\]

\(x^i\) is the \(j\)th pattern belongs to cluster \(i\).

- The Squared Error (SE) clustering method is as follows,

  1. Select an initial partition of the patterns with a fixed number of clusters and cluster centers.

  2. Assign each pattern to its closest cluster center and compute the new cluster centers as the centroids of the clusters. Repeat this step until convergence is achieved.

  3. Merge and split clusters based on some heuristic information, optionally repeating step 2.

- The stopping criterion for SE is shown in the equation (8) and (9).
Clustering - Clustering algorithms (5)

- The squared error $e_i$ for cluster $C_i$ is the sum of the squared Euclidean distances between each pattern $(x_j)$ in $C_i$ and its cluster center $m^i$.

$$e_i = \sum_{j=1}^{n_i} (x_j - m_i^T)(x_j - m_i), \quad x_j \in C_i \quad (8)$$

$$E_k = \sum_{i=1}^{k} e_i \quad (9)$$

Equation (8) is the squared error for each cluster.

Equation (9) is the total squared error for all cluster.
Example:

- Above is an example of two clusters created using partition method. The clusters are center at m1 and m2. xi is the patterns that belongs to each cluster respectively.
- The straight line of the input patterns from the cluster center represents the distance in the feature space.
K-means clustering (1)

- One type of partitioning algorithm.
- Simplest and the most commonly used algorithm employing a squared error criterion (complexity is $O(n)$, $n$ is the number of pattern).
- Iterative algorithm until a convergence criterion is met.
- Minimize total squared error $E$ in (9) in each iteration.
K-means clustering (2)

- Many variation of implementation (e.g. FORGY-dynamic number of cluster, CLARANS, PAM).
- Objectives - minimize distance between patterns within a cluster and maximize inter-cluster distance.
K-means clustering (3)

Steps,

(1) Choose k clusters centers to coincide with k randomly-chosen patterns or k randomly defined points inside the hypervolume containing the pattern set.

(2) Assign each pattern to the closest cluster center.

(3) Re-compute the cluster centers using the current cluster memberships.

(4) If a convergence criterion is not met, go to step 2. The convergence criterion are: no (or minimal) reassignment of patterns to new cluster centers, or minimal decrease in squared error.
K-means clustering - Serial Algorithm

<table>
<thead>
<tr>
<th>Initialize each cluster $C_i$ with centroid $m_i^t$, $1 \leq i \leq k$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize $\text{TotalE}$ to large number</td>
</tr>
<tr>
<td>do {</td>
</tr>
<tr>
<td>\hspace{1cm} $\text{OldTotalE} = \text{TotalE}$;</td>
</tr>
<tr>
<td>\hspace{1cm} $\text{tempTotalE} = 0$;</td>
</tr>
<tr>
<td>\hspace{1cm} for $j = 1$ to $k$</td>
</tr>
<tr>
<td>\hspace{2cm} varsum$<em>{j} = 0$; count$</em>{j} = 0$;</td>
</tr>
<tr>
<td>\hspace{1cm} end of</td>
</tr>
<tr>
<td>\hspace{1cm} for each input pattern $x_j$</td>
</tr>
<tr>
<td>\hspace{2cm} for $i = 1$ to $k$</td>
</tr>
<tr>
<td>\hspace{3cm} compute Squared Euclidean distance $d^2(x_j, m^t)$;</td>
</tr>
<tr>
<td>\hspace{2cm} end for</td>
</tr>
<tr>
<td>\hspace{1cm} find the closest centroid $m^t$ to $x_j$;</td>
</tr>
<tr>
<td>\hspace{1cm} //sum all point in a cluster</td>
</tr>
<tr>
<td>\hspace{2cm} varsum$_s = \text{varsum}_s + x_j$;</td>
</tr>
<tr>
<td>\hspace{2cm} count$_s = \text{count}_s + 1$;</td>
</tr>
<tr>
<td>\hspace{1cm} $\text{tempTotalE} = \text{tempTotalE} + d^2(x_j, m^t)$;</td>
</tr>
<tr>
<td>\hspace{1cm} end for</td>
</tr>
<tr>
<td>//calc new centroid</td>
</tr>
<tr>
<td>\hspace{1cm} for $i = 1$ to $k$</td>
</tr>
<tr>
<td>\hspace{2cm} count$_i = \text{max}(\text{count}_i, 1)$;</td>
</tr>
<tr>
<td>\hspace{2cm} $m^t = \frac{\text{varsum}_i}{\text{count}_i}$;</td>
</tr>
<tr>
<td>\hspace{1cm} end for</td>
</tr>
<tr>
<td>$\text{TotalE} = \text{tempTotalE};$</td>
</tr>
<tr>
<td>} while ($\text{TotalE} &lt; \text{OldTotalE}$)</td>
</tr>
</tbody>
</table>

Note: $m$ and varsum are maxtrix of size $d \times 1$, $d$ is the input pattern dimension

- Above is the pseudocode for the k-means clustering algorithm. $k$ is the total number of cluster. The $\text{do}\{\ldots\}\text{while}$ loop is the iterative step that assign and re-assign the input patterns to the clusters.
- At the end of each iterative the squared error are computed. This new error is compared with previous iteration error. The while loop stop when the new error is greater than the last error.
K-means clustering-select initial partitions (1)

- K-means algorithm is very sensitive to the choice initial centroid values.
- Different initialization may result in different final clustering because algorithms can converge to local minima.
- To overcome local minima, one can run the algorithm with several different initial partitions. If they lead to the (nearly) same final partition, then it is more confidence that the global minimum of square error has been achieved.
- The initial partition can be formed by random seed patterns from the user or from the input patterns itself.
K-means clustering-select initial partitions (2)

- Several methods to derive centroid values from input patterns
  - Sequential partition - select k contiguous partition from input patterns and assign each partition to a cluster.
  - Random partition - randomly assign all input pattern to a cluster.
  - Samples based - choose $m$ samples and assign them to a cluster.

• Few methods to determine the initial clusters center are given above.
• The initial cluster centers will effect greatly the final clusters formed.
K-means clustering-Updating the partition

- Reassigning patterns to clusters in an attempt to reduce the square-error.
- A pattern is assign to it closest cluster centroid (using distance measure in equation (4) or (5)).
- At each pass the centroid of clusters are update to minimize the squared error.
K-means clustering-Convergence

- Stopping criteria
  - K-means terminate when the criterion function cannot be improved. User may specify a minimum fraction \( t \) (%) change of total mean squared error between successive iteration; stop when total MSE change is less than or equal to this fraction.
  - Patterns in the clusters does not change between successive iteration.
  - User specify maximum iteration.
K-means clustering - parallel algorithm

- Employed data parallelism concept, where training data is partitions into \( p \) partition (\( p \) is the number of processor).
- Each processor will independently running the k-means serial algorithm on its local data partition \( D \).
- At the end of an iteration, all processor are synchronized to exchange local clusters statistical information (e.g. number of points in each cluster, total mse, and clusters variables sum).

(Dhillon 98, et. al.)

• The parallel k-means algorithm employed the shared nothing architecture and using data parallelism concept.
• Each processor will working on it local data partition. in a serial way. The collective communications are perform to exchange local information with all processors.
K-means clustering- parallel algorithm (1)

if (proc_rank == 0)
    read number of clusters from user
    partition_training_data(p);
end if
MPI_Barrier(comm_group);
MPI_Bcast(&number_clusters);
Assign initial data points to all clusters C_i and calculate cluster means m_i:
MPI_Barrier(comm_group);
MPI_Allreduce local clusters variables sum and number of points in each cluster.
do {
    OldTotE = TotE;
    tempTotE = 0;
    for j = 1 to k
        varsum_j = 0, count_j = 0
    end for

• Above is the pseudocode for the parallel k-means algorithm.
• Initially all processors initial the clusters centers with it local data partition. The data points at each processor are assign to the initial clusters.
• These initial clusters information are exchange between all processor to form uniform cluster centers. That is, all processor will have the same clusters center.
K-means clustering- parallel algorithm (2)

```plaintext
for each pattern \( x_j \) in local partition \( D^r \)
    for \( i = 1 \) to \( k \)
        compute squared Euclidean distance \( d_2(x_j, m^i) \)
    end for
    find the closest centroid \( m^s \) to \( x_j \);
    varsums = varsums + \( x_j \);
    counts = counts + 1
    tempTotalE = tempTotalE + \( d_2(x_j, m^s) \)
end for
Allocate sendbuffer and receivebuffer of contiguous mem size \( d \times k + k + 1 \)
Load tempTotalE, and varsum, count from each clusters into sendbuffer.
MPI_BARRIER(comm_group);
MPI_ALLREDUCE(sendbuffer, receivebuffer,...);
```

- In serial, each processor reassign it local data points to the clusters and updating the cluster means and number of data points.
- At the end of an iteration, all processor synchronized to exchange the local clusters information.
K-means clustering- parallel algorithm (3)

Update varsum of each clusters
Update count of each clusters
for i = 0 to k
  \[ m_i = \frac{\text{varsum}_i}{\text{count}_i} \]
end for
Update tempTotE using receivebuffer
\[ \text{TotE} = \text{tempTotE}; \]
while((TotE < OldTotE));
Parallel K-means - choosing initial partition

- The i-th cluster centers in each processor are initial with the i-subset from the local data partition. Other initialization method can also be used.
The MPI_Allreduce function with MPI_SUM operator is used to exchange local clusters information.

The minimize communication overhead, all the clusters information (clusters means and total points) are loaded into an array. So, the total information is exchanged by using only one MPI statement.

In the example above, the i-th cluster information in a processor is stored in the i-th portion of the array with the total point in a cluster appended to the \((d+1)\) index of the i-th cluster, where \(d\) is the data dimensions.

The MPI_Allreduce() function will sum up all the clusters information.
Parallel K-means - inputs

- **User inputs:**
  - a) Number of cluster
  - b) Stopping parameters (e.g. maximum iteration).

- **2 input files to root process**
  - a) config.txt
  - b) input.txt

<table>
<thead>
<tr>
<th>#NUMPATTERN 100</th>
<th>#DIMENSION 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>73.66 74.56 27.48</td>
<td></td>
</tr>
<tr>
<td>86.76 78.53 20.90</td>
<td></td>
</tr>
<tr>
<td>62.95 74.40 7.19</td>
<td></td>
</tr>
<tr>
<td>78.12 71.08 15.70</td>
<td></td>
</tr>
<tr>
<td>64.30 69.51 13.60</td>
<td></td>
</tr>
<tr>
<td>74.14 79.32 32.50</td>
<td></td>
</tr>
</tbody>
</table>

- There are two input files config.txt and input.txt.
- config.txt file contains the number of training pattern NUMPATTERN, and data dimension DIMENSION.
- The “input.txt” file contains the training data. The data section start with the “@data” keyword.
- The file name is hard coded to provide easier understanding.
Output Format

- User inputs are
  - Number of clusters, and
  - Maximum number of iteration

- Output
  - KMEANS CLUSTERING
    - Number of cluster: 10
    - Maximum iteration: 5
    - Iteration: 1 MSErr: 3.38e+05
    - Iteration: 2 MSErr: 9.94e+04
    - Iteration: 3 MSErr: 5.47e+04
    - Iteration: 4 MSErr: 4.94e+04
    - Iteration: 5 MSErr: 4.73e+04

Output Files

- The only output file is clusters.xx, where xx is the number of cluster.
- Sample file output

  =========== Cluster #1 ===========
  Num of points: 40
  Cluster center
  10.56, 72.55, 39.41
  =========== Cluster #2 ===========
  Num of points: 40
  Cluster center
  33.87, 73.39, 15.21
Objects Classes (1)

- class Cluster {...}
  - store a cluster information such as number of points, cluster center, and squared error.
- class kmeans {...}
  - k-means clustering algorithm
  - keep all formed clusters
- class Dataset {...}
  - data input functions

class Cluster {
    private:
    float *stdDev;
    double *sum;
    double *sumSquared;
    double sumSqError;
    float *centroid;
    long numOfPoints;
    int dimensions;
    int id;
    public:
    Cluster(int, float *, int);
    ~Cluster();
    float eucdistance(float * x);
    float eucdistance(Cluster * & c);
    void clear();
    void updateMeans();
    void updatevarsum(double *);
    double * getvarsum() { return sum; }
    long &getPointssize() { return numOfPoints; }
}
double squaredError() { return sumSqError; }
void calcStdDev(void);
void addPoint(float * pt);
friend ostream & operator<<(ostream &, Cluster &);
};

class kmeans {
private:
  unsigned long maxIteration; /* maximum iteration */
  unsigned long trainingIter; /* training iteration */
  unsigned int numCluster; /* number of cluster */
  double convThreshold; /* convergence threshold */
  double E; /* global squared error */
  Cluster ** clusters; /* kmeans cluster */
  Dataset *db; /* pointer to training data file */
  int closestCluster(float * & pt);
void kmeans_initial_clusters(MPI_Comm & comm, int);
public:
  kmeans(unsigned long m, unsigned int k, double th,
         Dataset * &); /* constructor */
~kmeans(); /* destructor */
void formClusters(MPI_Comm & comm, int);
void printClusters();
};
Parallel k-means

- Implementations
  1. in main() function, calls partition() function to partition the input data into p partition.
  2. initialize Dataset object instance to read configuration file and local data partition.
  3. initialize kmeans object instance to initialize the k-means clustering algorithm
  4. calls formcluster() method in kmeans class.

```c
int main(int argc, char *argv[])
{
    ... /*variables declaration*/
    ... /* MPI initialization*/
    if(myrank == 0) /*root processor */
    {
        /*get inputs from user*/
        cout << "Number of cluster:"; cin >> numcluster;
        cout << "Maximum iteration:"; cin >> maxiter;
        partition("input.txt","config.txt", proc_size);
    }
    MPI_Barrier(io_comm);
    MPI_Bcast(&numcluster,1, MPI_INT, 0,io_comm);
    MPI_Bcast(&maxiter, 1, MPI_UNSIGNED_LONG,0,io_comm);
    ...
    db = new Dataset(dbfilename, configfilename);
    km = new kmeans(maxiter, numcluster, 0.0, db);
    km->formclusters(io_comm,myrank);
    ...
}
```
1. void kmeans::formclusters(MPI_Comm &comm, int rankno)
   { DATAPoint * buffer;
    float oldsumError;
    float cursumError;
  
5.     int closeclus;
    unsigned long i,k,l;
    unsigned int j;
    double *varsum, * recvbuffer, * sendbuffer;
    const unsigned int dimensions = db->dimension();

10.    kmeans_initial_clusters(comm, rankno); /*initialize k-means model*/
    db->reset();
    oldsumError = 3.4 * pow(10,38);
    /*buffer size = each cluster variable sum x dimensions
     + mse + number of points in each cluster*/
    int bufsize = numcluster * dimensions + 1 + numcluster;
    recvbuffer = new double[bufsize];

15.    sendbuffer = new double[bufsize];
    for(i = 0; i < maxiteration; i++)
    {
       cursumError = 0.0;
       /*scan input patterns*/
       while((buffer = db->next_pattern()) != NULL)
       {
         /*find the closest cluster for current pattern*/
         closeclus = closestCluster(buffer->varvalue);
         /*add current pattern to the closest cluster*/
         clusters[closeclus]->addPoint(buffer->varvalue);
         delete [] buffer->varvalue;
         delete buffer;
       }

20.    k=0;
    /*total up mean squared errors from all cluster*/
    for (j = 0; j < numcluster; j++)
    {
       cursumError += clusters[j]->squaredError();
    }
    for(j =0; j < numcluster; j++)
    {
       varsum = clusters[j]->getvarsum();
       for(l=0; l < dimensions; l++)
       {
          sendbuffer[k++] = varsum[l];
       }
    }
sendbuffer[k++] = (double)clusters[j]->getPointssize();

sendbuffer[k] = cursumError; /*load mse*/
MPI_Barrier(comm); //synchronous all processor
MPI_Allreduce(sendbuffer, recvbuffer, bufsize, MPI_DOUBLE, MPI_SUM, comm);

k=0;
/*update each processor local variables sum value */

for(j =0; j< numcluster; j++)
{      clusters[j]->updavalue(recvbuffer + k);
        k += dimensions;
      clusters[j]->getPointssize() = (long)recvbuffer[k];
      clusters[j]->updateMeans(); /*update centroid*/

      k++;
}

cursumError = recvbuffer[k]; /*global mse*/
trainingiter++;
//check stopping conditions

if (cursumError >= oldsumError)
    break;

oldsumError = cursumError;
for(j=0; j < numcluster; j++)
    clusters[j]->clear();

  }

  db->reset();
  }
//for

• Lines 2 – 9: Local variables declarations.
• Lines 10: Initializing kmeans algorithm.
• Lines 14-15: allocate buffer to receive clusters information from other processor.
• Lines 16: K-means main loop start and end at lines 56.
• Lines 18-24: scan the database and assign to the closest cluster.
• Lines 26-27: Calculate squared error in each cluster.
• Lines 28-35: save the clusters center and number of point in each cluster into the sendbuffer. The sum squared error also save in the buffer in line 36.
• Lines 37-38: Synchronize all processor and perform total exchange.
• Lines 40-46: Update local clusters information to global cluster information by updating the cluster center and the total number of point in the cluster.
• Lines 53: Checking the stopping criterion.
• Lines 56-57- delete the total points information in all clusters and create a new cluster.
Clusters Initialization (1)

- Divide the training data into k partitions.
- Initialize each cluster with one of the partition.
- Exchange to local clusters means with other processors.

1. ```
   void kmeans::kmeans_initial_clusters(MPI_Comm &io_comm, int rank)
   {
     unsigned long i;
     unsigned j, k;
     unsigned long * count;
     unsigned long patsize = db->size();
     float * varsum;
     DATAPoint *buffer;
     unsigned int dimensions = db->dimension();
     MPI_Comm_rank(io_comm, &rank);
     db->reset();
     varsum = new float[dimensions * numcluster];
     count = new unsigned long[numcluster];
     for (i=0; i < numcluster; i++)
       count[i] = 0;
     for (i=0; i < (dimensions * numcluster); i++)
       varsum[i] = 0;
     i=0; j=0;
     while ((buffer = db->next_pattern()) != NULL)
     {
```
20.   j = i * numcluster / patsize; i++; 
    if (j >= numcluster) 
      break; 
    for(k=0;k < dimensions;k++) 
      varsum[j * dimensions + k] += buffer->varvalue[k]; 
25.   count[j]++; /*increment count*/ 
    delete [] buffer->varvalue; 
    delete buffer; 
} 
clusters = new Cluster*[numcluster]; /*allocate cluster*/ 
30.   float * globalmeans = new float[numcluster * dimensions]; 
    unsigned long * globalcount = new unsigned long[numcluster]; 
    MPI_BARRIER(io_comm);        /*synchronize all processor*/ 
    MPI_Allreduce(varsum, globalmeans, numcluster*dimensions, 
                   MPI_FLOAT, MPI_SUM, io_comm); 
    MPI_Allreduce(count, globalcount, numcluster, 
                   MPI_UNSIGNED_LONG, MPI_SUM, io_comm); 
35.   delete [] varsum; 
    delete [] count; 
    for(i=0;i < numcluster; i++) 
    {  for(j=0;j < dimensions; j++) 
       globalmeans[i * dimensions + j] /= (double)globalcount[i]; 
40.   } 
    delete [] globalcount; 
    int dim = db->dimension(); 
    for(j=0;j < numcluster; j++) 
    {  clusters[j] = new Cluster(dim, globalmeans + (j*dim), j+1); 
45.   } 
    delete [] globalmeans; 
} 
• Lines 18-28: Initialize each cluster with N/k training data, where N is the total number of 
  training data and k is number of clusters. 
• Lines 29 create the actual array of cluster. 
• Lines 33: Obtain the global means value of each clusters. 
• Lines 34: Obtain the global number of points in each cluster from all processors. 
• Lines 37-40: Recalculate the new cluster means. 
• Lines 43-45: Create cluster instance using the means/cluster center.
1. `Cluster::Cluster(int dim, float * cent, int idnum=-1)`
   
   ```
   
   {  
      dimensions = dim;  
      numOfPoints = 0;  
     
     5.  
      stddev = new float[dimensions];  
      sum = new double[dimensions];  
      sumSquared = new double[dimensions];  
      centroid = new float[dimensions];  
      sumSqError = 0.0;  
     
     10.  
      for (int i=0; i < dimensions; i++)  
         
         centroid[i] = cent[i];  
         
         id = idnum;  
         
         clear();  
   }
   ```

- Cluster class constructor.
1. void Cluster::addPoint(float * pt) 
   
   int i; 
   double *ss = sumSquared; 
5. double *s = sum; 
   for (i=0;i < dimensions;i++) 
   { 
      sumSqError += eucdistance(pt); /*total sum squared error */ 
      *s += pt[i]; 
10. *ss += pt[i] * pt[i];/* square of point.To calculate std dev */ 
      s++; ss++; 
   } 
   numOfPoints++; 
   }

• Add a data point to a cluster and update their sum squared errors. 
• Lines 6-14: Calculate the Euclidean distance to the cluster center/mean. and update the total number of points in the clusters.

1. void Cluster::updateMeans() 
   
   float * oldmeans = centroid; 
   double * varssum = sum; 
5. float newmeans; 
   if (numOfPoints <= 0) 
      numOfPoints = 1; 
   for(int i=0;i < dimensions;i++) 
   { 
      /*mean value for variable i */ 
10. newmeans = *varssum / numOfPoints; 
      *oldmeans = newmeans; /*new centroid value*/ 
      oldmeans++; varssum++; /*next variable */ 
   } 
   }

• Update the cluster means. 
• Lines 8: For each input features. 
• Lines 10: calculate new means using the sum of all data points for each data features.
End of Module 5
Clustering