PCA and k-means clustering

The C++ implmentation for PCA-based and k-means clustering

Generated by Doxygen 1.8.11

Contents

1 k-means 1

2 Class Index 3

2.1 Class List 3

3 File Index 5

3.1 File List 5

4 Class Documentation 7

4.1 AHC_node Struct Reference 7

4.1.1 Detailed Description 7

4.1.2 Constructor & Destructor Documentation 7

4.1.2.1 AHC_node(const int &index) 7

4.1.2.2 AHC_node() 8

4.1.3 Member Data Documentation 8

4.1.3.1 element 8

4.1.3.2 index 8

4.2 DistNode Struct Reference 8

4.2.1 Detailed Description 8

4.2.2 Constructor & Destructor Documentation 9

4.2.2.1 DistNode(const int &first, const int &second, const float &dist) 9

4.2.2.2 DistNode() 9

4.2.3 Member Data Documentation 9

4.2.3.1 distance 9

4.2.3.2 first 9

4.2.3.3 second 9

4.3 Ensemble Struct Reference 9

4.3.1 Detailed Description 10

4.3.2 Constructor & Destructor Documentation 10

4.3.2.1 Ensemble(const int &number, const int &index) 10

4.3.3 Member Function Documentation 10

4.3.3.1 operator<(const Ensemble &object) const 10

4.3.4 Member Data Documentation 10

4.3.4.1 newIndex 10

4.3.4.2 number 10

4.3.4.3 oldIndex 10

4.4 PCA_Cluster Class Reference 11

4.4.1 Detailed Description 11

4.4.2 Member Function Documentation 12

4.4.2.1 getDistAtNodes(const vector< int > &firstList, const vector< int > &secondList, const Eigen::MatrixXf &reduced_dist_matrix) 12

4.4.2.2 hierarchicalMerging(std::unordered_map< int, AHC_node > &nodeMap, std↩::vector< DistNode > &dNodeVec, std::vector< AHC_node > &nodeVec, const Eigen::MatrixXf &reduced_dist_matrix, const Eigen::MatrixXf &cArray, const int &numberOfClusters, TimeRecorder &tr) 12

4.4.2.3 perform_AHC(const MatrixXf &cArray, const int &PC_Number, const MatrixXf &↩SingVec, const VectorXf &meanTrajectory, std::vector< MeanLine > &mass↩Center, const int &Cluster, std::vector< int > &group, std::vector< int > &total↩Num, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const Eigen::MatrixXf &data, TimeRecorder &tr, Silhouette &sil) 14

4.4.2.4 performDirectK_Means(const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std↩::vector< ExtractedLine > &furthest, const int &normOption, TimeRecorder &tr, Silhouette &sil) 16

4.4.2.5 performDirectK_Means(const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std↩::vector< ExtractedLine > &furthest, const int &Cluster, const int &normOption, TimeRecorder &tr, Silhouette &sil) 16

4.4.2.6 performFullK_MeansByClusters(const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, const int &↩Cluster, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const int &normOption, TimeRecorder &tr, Silhouette &sil) 16

4.4.2.7 performPC_KMeans(const MatrixXf &cArray, const int &Row, const int &Column, const int &PC_Number, const MatrixXf &SingVec, const VectorXf &meanTrajectory, std::vector< MeanLine > &massCenter, const int &Cluster, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const Eigen::MatrixXf &data, TimeRecorder &tr, Silhouette &sil) 20

4.4.2.8 performPCA_Clustering(const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, TimeRecorder &tr, Silhouette &sil) 22

4.4.2.9 performPCA_Clustering(const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const int &Cluster, TimeRecorder &tr, Silhouette &sil) 23

4.4.2.10 performSVD(MatrixXf &cArray, const Eigen::MatrixXf &data, const int &Row, const int &Column, int &PC_Number, MatrixXf &SingVec, VectorXf &meanTrajectory, TimeRecorder &tr) 23

4.4.2.11 setLabel(const std::vector< AHC_node > &nodeVec, vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf &centroid, const Eigen::MatrixXf &cArray, std::vector< int > &recorder) 24

4.4.2.12 setValue(std::vector< DistNode > &dNodeVec, const Eigen::MatrixXf &reduced_data, const Eigen::MatrixXf &reduced_dist_matrix) 24

4.5 TimeRecorder Struct Reference 25

4.5.1 Detailed Description 25

4.5.2 Member Data Documentation 25

4.5.2.1 eventList 25

4.5.2.2 timeList 25

5 File Documentation 27

5.1 main.cpp File Reference 27

5.1.1 Function Documentation 28

5.1.1.1 featureExtraction(const int &argc, char **argv) 28

5.1.1.2 main(int argc, char *argv[]) 30

5.1.1.3 performK_Means(const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &cluster, const int &dimension, const string &fullName, const int &maxElements, const Eigen::MatrixXf &data, const int &normOption, TimeRecorder &tr, Silhouette &sil) 30

5.1.1.4 performPCA_Cluster(const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &cluster, const int &dimension, const string &fullName, const int &maxElements, const Eigen::MatrixXf &data, TimeRecorder &tr, Silhouette &sil) 31

5.1.2 Variable Documentation 32
5.1.2.1 initializationOption 32
5.1.2.2 isPBF 32
5.1.2.3 post_processing 32
5.1.2.4 readCluster 32
5.2 PCA_Cluster.cpp File Reference 33
5.2.1 Variable Documentation 33
5.2.1.1 CLUSTER 33
5.2.1.2 initializationOption 33
5.2.1.3 isPBF 33
5.2.1.4 post_processing 33
5.2.1.5 TOR_1 34
5.3 PCA_Cluster.h File Reference 34
5.4 Predefined.cpp File Reference 35
5.4.1 Function Documentation 35
5.4.1.1 deleteVecElements(std::vector< T > &original, const T &first, const T &second) 35
5.5 Predefined.h File Reference 36
5.5.1 Function Documentation 37
5.5.1.1 deleteVecElements(std::vector< T > &origine, const T &first, const T &second) . 37
5.6 README.md File Reference 37
Index **39**

Chapter 1

k-means

It includes the clustering algorithms,

- PCA-based clustering with default parameter suggested by relative paper
- k-means algorithm with all similarity measures

k-means initialization

It includes three types of initialization types

- From random coordinates to generate an initialized line
- From the input lines to act as the initialized line
- The k-means++ initialization based on uniform probability w.r.t. distance

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AHC_node 7
DistNode 8
Ensemble 9
PCA_Cluster 11
TimeRecorder 25

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

main.cpp 27
PCA_Cluster.cpp 33
PCA_Cluster.h 34
Predefined.cpp 35
Predefined.h 36

Chapter 4

Class Documentation

4.1 AHC_node Struct Reference

```
#include <Predefined.h>
```

Public Member Functions

- AHC_node (const int &index)
- AHC_node ()

Public Attributes

- int index = -1
- std::vector< int > element

4.1.1 Detailed Description

Definition at line 12 of file Predefined.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AHC_node::AHC_node (const int & *index*) [inline]

Definition at line 19 of file Predefined.h.

```
19                                  : index(index)
20     {}
```

4.1.2.2 AHC_node::AHC_node ( ) [inline]

Definition at line 22 of file Predefined.h.

```
23     {}
```

4.1.3 Member Data Documentation

4.1.3.1 std::vector<int> AHC_node::element

Definition at line 17 of file Predefined.h.

4.1.3.2 int AHC_node::index = -1

Definition at line 14 of file Predefined.h.

The documentation for this struct was generated from the following file:

- Predefined.h

4.2 DistNode Struct Reference

```
#include <Predefined.h>
```

Public Member Functions

- DistNode (const int &first, const int &second, const float &dist)
- DistNode ()

Public Attributes

- int first = -1
- int second = -1
- float distance = -1.0

4.2.1 Detailed Description

Definition at line 41 of file Predefined.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DistNode::DistNode (const int & *first,* const int & *second,* const float & *dist*) [inline]

Definition at line 46 of file Predefined.h.

```
46                                                                  :first(
      first), second(second), distance(dist)
47     {}
```

4.2.2.2 DistNode::DistNode ( ) [inline]

Definition at line 49 of file Predefined.h.

```
50     {}
```

4.2.3 Member Data Documentation

4.2.3.1 float DistNode::distance = -1.0

Definition at line 44 of file Predefined.h.

4.2.3.2 int DistNode::first = -1

Definition at line 43 of file Predefined.h.

4.2.3.3 int DistNode::second = -1

Definition at line 43 of file Predefined.h.

The documentation for this struct was generated from the following file:

- Predefined.h

4.3 Ensemble Struct Reference

```
#include <PCA_Cluster.h>
```

Public Member Functions

- Ensemble (const int &number, const int &index)
- bool operator< (const Ensemble &object) const

Public Attributes

- int number
- int newIndex
- int oldIndex

4.3.1 Detailed Description

Definition at line 20 of file PCA_Cluster.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Ensemble::Ensemble (const int & *number,* const int & *index*) [inline]

Definition at line 25 of file PCA_Cluster.h.

```
25                                                   :number(number),
      newIndex(-1), oldIndex(index)
26      {}
```

4.3.3 Member Function Documentation

4.3.3.1 bool Ensemble::operator< (const Ensemble & *object*) const [inline]

Definition at line 27 of file PCA_Cluster.h.

```
28      {
29          return number<object.number;
30      }
```

4.3.4 Member Data Documentation

4.3.4.1 int Ensemble::newIndex

Definition at line 23 of file PCA_Cluster.h.

4.3.4.2 int Ensemble::number

Definition at line 22 of file PCA_Cluster.h.

4.3.4.3 int Ensemble::oldIndex

Definition at line 24 of file PCA_Cluster.h.

The documentation for this struct was generated from the following file:

- PCA_Cluster.h

4.4 PCA_Cluster Class Reference

```
#include <PCA_Cluster.h>
```

Static Public Member Functions

- static void performPCA_Clustering (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, TimeRecorder &tr, Silhouette &sil)
- static void performPCA_Clustering (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const int &Cluster, TimeRecorder &tr, Silhouette &sil)
- static void performDirectK_Means (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const int &normOption, TimeRecorder &tr, Silhouette &sil)
- static void performDirectK_Means (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const int &Cluster, const int &normOption, TimeRecorder &tr, Silhouette &sil)

Static Private Member Functions

- static void performSVD (MatrixXf &cArray, const Eigen::MatrixXf &data, const int &Row, const int &Column, int &PC_Number, MatrixXf &SingVec, VectorXf &meanTrajectory, TimeRecorder &tr)
- static void performPC_KMeans (const MatrixXf &cArray, const int &Row, const int &Column, const int &PC_Number, const MatrixXf &SingVec, const VectorXf &meanTrajectory, std::vector< MeanLine > &massCenter, const int &Cluster, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const Eigen::MatrixXf &data, TimeRecorder &tr, Silhouette &sil)
- static void performFullK_MeansByClusters (const Eigen::MatrixXf &data, const int &Row, const int &Column, std::vector< MeanLine > &massCenter, const int &Cluster, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const int &normOption, TimeRecorder &tr, Silhouette &sil)
- static void perform_AHC (const MatrixXf &cArray, const int &PC_Number, const MatrixXf &SingVec, const VectorXf &meanTrajectory, std::vector< MeanLine > &massCenter, const int &Cluster, std::vector< int > &group, std::vector< int > &totalNum, std::vector< ExtractedLine > &closest, std::vector< ExtractedLine > &furthest, const Eigen::MatrixXf &data, TimeRecorder &tr, Silhouette &sil)
- static void hierarchicalMerging (std::unordered_map< int, AHC_node > &nodeMap, std::vector< DistNode > &dNodeVec, std::vector< AHC_node > &nodeVec, const Eigen::MatrixXf &reduced_dist_matrix, const Eigen::MatrixXf &cArray, const int &numberOfClusters, TimeRecorder &tr)
- static float getDistAtNodes (const vector< int > &firstList, const vector< int > &secondList, const Eigen::MatrixXf &reduced_dist_matrix)
- static void setValue (std::vector< DistNode > &dNodeVec, const Eigen::MatrixXf &reduced_data, const Eigen::MatrixXf &reduced_dist_matrix)
- static void setLabel (const std::vector< AHC_node > &nodeVec, vector< vector< int > > &neighborVec, vector< int > &storage, Eigen::MatrixXf &centroid, const Eigen::MatrixXf &cArray, std::vector< int > &recorder)

4.4.1 Detailed Description

Definition at line 47 of file PCA_Cluster.h.

4.4.2 Member Function Documentation

4.4.2.1 float PCA_Cluster::getDistAtNodes (const vector< int > & *firstList*, const vector< int > & *secondList*, const Eigen::MatrixXf & *reduced_dist_matrix*) [static], [private]

Definition at line 1156 of file PCA_Cluster.cpp.

```
{
    const int& m = firstList.size();
    const int& n = secondList.size();
    assert(m!=0);
    assert(n!=0);

    float result, value;
    result = 0;
#pragma omp parallel for reduction(+:result) num_threads(8)
    for(int i=0;i<m;++i)
    {
        for(int j=0;j<n;++j)
        {
            value = reduced_dist_matrix(i,j);
            result+=value;
        }
    }
    result/=m*n;
    return result;
}
```

4.4.2.2 void PCA_Cluster::hierarchicalMerging (std::unordered_map< int, AHC_node > & *nodeMap*, std::vector< DistNode > & *dNodeVec*, std::vector< AHC_node > & *nodeVec*, const Eigen::MatrixXf & *reduced_dist_matrix*, const Eigen::MatrixXf & *cArray*, const int & *numberOfClusters*, TimeRecorder & *tr*) [static], [private]

Definition at line 1027 of file PCA_Cluster.cpp.

```
{
    /* would store distance matrix instead because it would save massive time */
    struct timeval start, end;
    double timeTemp;
    gettimeofday(&start, NULL);

    const int Row = cArray.rows();

    for(int i=0;i<Row;++i)
    {
        nodeMap[i].element.push_back(i);
    }

    DistNode poped;

    /* find node-pair with minimal distance */
    float minDist = FLT_MAX;
    int target = -1;
    for (int i = 0; i < dNodeVec.size(); ++i)
    {
        if(dNodeVec[i].distance<minDist)
        {
            target = i;
            minDist = dNodeVec[i].distance;
        }
    }
    poped = dNodeVec[target];

    int index = Row, currentNumber;
    do
    {
        //create new node merged and input it into hash map
        vector<int> first = (nodeMap[poped.first]).element;
        vector<int> second = (nodeMap[poped.second]).element;

        /* index would be starting from Row */
        AHC_node newNode(index);
        newNode.element = first;
        newNode.element.insert(newNode.element.end(), second.begin(), second.end());
        nodeMap.insert(make_pair(index, newNode));

        //delete two original nodes
        nodeMap.erase(poped.first);
        nodeMap.erase(poped.second);

        /* the difficulty lies how to update the min-heap with linkage
         * This would take 2NlogN.
         * Copy all node-pairs that are not relevant to merged nodes to new vec.
         * For relevant, would update the mutual distance by linkage
         */

        /* how many clusters exist */
        currentNumber = nodeMap.size();

        target = -1, minDist = FLT_MAX;

        std::vector<DistNode> tempVec(currentNumber*(currentNumber-1)/2);
        int current = 0, i_first, i_second;
        for(int i=0;i<dNodeVec.size();++i)
        {
            i_first=dNodeVec[i].first, i_second=dNodeVec[i].second;
            /* not relevant, directly copied to new vec */
            if(i_first!=poped.first&&i_first!=poped.second&&i_second!=poped.first&&i_second!=poped.second)
            {
                tempVec[current]=dNodeVec[i];
                if(tempVec[current].distance<minDist)
                {
                    target = current;
                    minDist = tempVec[current].distance;
                }
                ++current;
            }
        }

        for (auto iter=nodeMap.begin();iter!=nodeMap.end();++iter)
        {
            if((*iter).first!=newNode.index)
            {
                tempVec[current].first = (*iter).first;
                tempVec[current].second = newNode.index;
                tempVec[current].distance=getDistAtNodes(newNode.element,(*iter).second.element, reduced_dist_matrix);
                if(tempVec[current].distance<minDist)
                {
                    target = current;
                    minDist = tempVec[current].distance;
                }
                ++current;
            }
        }
        poped = tempVec[target];

        /* judge whether current is assigned to right value */
        assert(current==tempVec.size());
        dNodeVec.clear();
        dNodeVec = tempVec;
        tempVec.clear();
        ++index;
    }while(nodeMap.size()!=numberOfClusters);   //merging happens whenever requested cluster is not met

    nodeVec=std::vector<AHC_node>(nodeMap.size());
    int tag = 0;
    for(auto iter=nodeMap.begin();iter!=nodeMap.end();++iter)
        nodeVec[tag++]=(*iter).second;

    gettimeofday(&end, NULL);
    timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;

    tr.eventList.push_back("Hirarchical clustering for "+to_string(numberOfClusters)+" groups takes: ");
    tr.timeList.push_back(to_string(timeTemp)+" s");
    /* task completed, would delete memory contents */
    dNodeVec.clear();
    nodeMap.clear();
    /* use alpha function to sort the group by its size */
    std::sort(nodeVec.begin(), nodeVec.end(), [](const AHC_node& e1, const AHC_node& e2)
    {return e1.element.size()<e2.element.size()||(e1.element.size()==e2.element.size()&&e1.index<e2.index);});
}
```

4.4.2.3 void PCA_Cluster::perform_AHC (const MatrixXf & *cArray*, const int & *PC_Number*, const MatrixXf & *SingVec*, const VectorXf & *meanTrajectory*, std::vector< MeanLine > & *massCenter*, const int & *Cluster*, std::vector< int > & *group*, std::vector< int > & *totalNum*, std::vector< ExtractedLine > & *closest*, std::vector< ExtractedLine > & *furthest*, const Eigen::MatrixXf & *data*, TimeRecorder & *tr*, Silhouette & *sil*) [static], [private]

Definition at line 846 of file PCA_Cluster.cpp.

```
{
    std::unordered_map<int, AHC_node> nodeMap;
    std::vector<DistNode> dNodeVec;
    std::vector<AHC_node> nodeVec;
    const int& Row = cArray.rows();
    const int& Column = cArray.cols();

    /* compute distance matrix for reduced_space */
    Eigen::MatrixXf reduced_dist_matrix = Eigen::MatrixXf::Zero(Row, Row);
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Row; ++i)
    {
        for (int j = 0; j < Row; ++j)
        {
            /* don't wish to waste computation on diagonal element */
            if(i==j)
                continue;
            else
                reduced_dist_matrix(i,j) = (cArray.row(i)-cArray.row(j)).norm();
        }
    }
    /* set the ditNode vector */
    setValue(dNodeVec, cArray, reduced_dist_matrix);

    /* perform hirarchical clustering where within each step would merge two nodes */
    hierarchicalMerging(nodeMap, dNodeVec, nodeVec, reduced_dist_matrix, cArray, Cluster
  , tr);

    vector<vector<int> > neighborVec(Cluster);

    // element size for all groups
    vector<int> storage(Cluster);

    // geometric center
    Eigen::MatrixXf centroid = Eigen::MatrixXf::Zero(Cluster,Column);

    std::vector<int> recorder(Row);
    // set label information
    setLabel(nodeVec, neighborVec, storage, centroid, cArray, recorder);

    nodeVec.clear();

    struct timeval start, end;
    double delta;
    std::multimap<int,int> groupMap;

    float entropy = 0.0;
    float probability;

    for (int i = 0; i < Cluster; ++i)
    {
        groupMap.insert(std::pair<int,int>(storage[i],i));
        if(storage[i]>0)
        {
            probability = float(storage[i])/float(Row);
            entropy += probability*log2f(probability);
        }
    }

    int groupNo = 0;
    int increasingOrder[Cluster];
    for (multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
    {
        if(it->first>0)
        {
            increasingOrder[it->second] = (groupNo++);
        }
    }

    /* calculate the balanced entropy */
    entropy = -entropy/log2f(groupNo);
    Eigen::MatrixXf clusterCenter(Cluster, Column);

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Row; ++i)
    {
        group[i] = increasingOrder[recorder[i]];
        totalNum[i] = storage[recorder[i]];
    }

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Cluster; ++i)
    {
        clusterCenter.row(increasingOrder[i]) = centroid.row(i);
    }

    float shortest, farDist, toCenter;
    int shortestIndex = 0, fartestIndex = 0, tempIndex = 0;
    std::vector<int> neighborTemp;

    for (int i = 0; i < Cluster; ++i)
    {
        if(storage[i]>0 && !neighborVec[i].empty())
        {
            neighborTemp = neighborVec[i];
            shortest = FLT_MAX;
            farDist = FLT_MIN;

            for (int j = 0; j < storage[i]; ++j)
            {
                tempIndex = neighborTemp[j];
                toCenter = (clusterCenter.row(i)-cArray.row(tempIndex)).norm();

                if(toCenter<shortest)
                {
                    shortest = toCenter;
                    shortestIndex = tempIndex;
                }
                if(toCenter>farDist)
                {
                    farDist = toCenter;
                    fartestIndex = tempIndex;
                }
            }
            closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
            furthest.push_back(ExtractedLine(fartestIndex,increasingOrder[i]));
        }
    }
    MatrixXf pcSing(PC_Number,Column);

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < PC_Number; ++i)
    {
        pcSing.row(i) = SingVec.row(i);
    }

    MatrixXf massPos = clusterCenter*pcSing;

    for (int i = 0; i < Cluster; ++i)
    {
        if(storage[i]>0)
        {
            massPos.row(i) += meanTrajectory.transpose();
            std::vector<float> vecTemp;
            for (int j = 0; j < Column; ++j)
            {
                vecTemp.push_back(massPos(i,j));
            }
            massCenter.push_back(MeanLine(vecTemp,increasingOrder[i]));
        }
    }

    ValidityMeasurement vm;
    vm.computeValue(cArray, group);

    tr.eventList.push_back("PCA Validity measure is: ");
    stringstream fc_ss;
    fc_ss << vm.f_c;
    tr.timeList.push_back(fc_ss.str());

    /* Silhouette effect */
    gettimeofday(&start, NULL);

    sil.computeValue(cArray,group,groupNo,isPBF);

    gettimeofday(&end, NULL);
    delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;

    tr.eventList.push_back("Clustering evaluation computing takes: ");
    tr.timeList.push_back(to_string(delta)+"s");

    /* write value of the silhouette class */
    IOHandler::writeReadme(entropy, sil, "");

}
```

4.4.2.4 void PCA_Cluster::performDirectK_Means (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MeanLine > & *massCenter*, std::vector< int > & *group*, std::vector< int > & *totalNum*, std::vector< ExtractedLine > & *closest*, std::vector< ExtractedLine > & *furthest*, const int & *normOption*, TimeRecorder & *tr*, Silhouette & *sil*) [static]

Definition at line 426 of file PCA_Cluster.cpp.

```
{

    performFullK_MeansByClusters(data, Row, Column, massCenter,
  CLUSTER, group,
                                 totalNum, closest, furthest, normOption, tr, sil);
}
```

4.4.2.5 void PCA_Cluster::performDirectK_Means (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MeanLine > & *massCenter*, std::vector< int > & *group*, std::vector< int > & *totalNum*, std::vector< ExtractedLine > & *closest*, std::vector< ExtractedLine > & *furthest*, const int & *Cluster*, const int & *normOption*, TimeRecorder & *tr*, Silhouette & *sil*) [static]

Definition at line 504 of file PCA_Cluster.cpp.

```
{
    performFullK_MeansByClusters(data, Row, Column, massCenter, Cluster, group,

                                 totalNum, closest, furthest, normOption, tr, sil);
}
```

4.4.2.6 void PCA_Cluster::performFullK_MeansByClusters (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MeanLine > & *massCenter*, const int & *Cluster*, std::vector< int > & *group*, std::vector< int > & *totalNum*, std::vector< ExtractedLine > & *closest*, std::vector< ExtractedLine > & *furthest*, const int & *normOption*, TimeRecorder & *tr*, Silhouette & *sil*) [static], [private]

Definition at line 537 of file PCA_Cluster.cpp.

```
{
    MetricPreparation object(Row, Column);
    object.preprocessing(data, Row, Column, normOption);

    MatrixXf clusterCenter;

    switch(initializationOption)
    {
    case 1:
        Initialization::generateRandomPos(clusterCenter, Column, data, Cluster);
        break;

    case 2:
        Initialization::generateFromSamples(clusterCenter, Column, data, Cluster);
        break;

    case 3:
        Initialization::generateFarSamples(clusterCenter, Column, data, Cluster,
                                           normOption, object);
        break;
    }

    float moving=1000, tempMoving,/* dist, tempDist, */before;
    int *storage = new int[Cluster]; // used to store number inside each cluster
    MatrixXf centerTemp;
    int tag = 0;
    std::vector< std::vector<int> > neighborVec(Cluster, std::vector<int>());

/* perform K-means with different metrics */
    std::cout << "K-means start!" << std::endl;
    struct timeval start, end;
    gettimeofday(&start, NULL);
    std::vector<int> recorder(Row); //use to record which cluster the row belongs to

    do
    {
    /* reset storage number and weighted mean inside each cluster*/
        before=moving;
        memset(storage,0,sizeof(int)*Cluster);
        centerTemp = MatrixXf::Zero(Cluster,Column);

    /* clear streamline indices for each cluster */
    #pragma omp parallel for schedule(static) num_threads(8)
        for (int i = 0; i < Cluster; ++i)
        {
            neighborVec[i].clear();
        }

    #pragma omp parallel num_threads(8)
        {
        #pragma omp for nowait
            for (int i = 0; i < Row; ++i)
            {
                int clusTemp;
                float dist = FLT_MAX;
                float tempDist;
                for (int j = 0; j < Cluster; ++j)
                {
                    tempDist = getDisimilarity(clusterCenter.row(j),data,i,normOption,object);
                    if(tempDist<dist)
                    {
                        dist = tempDist;
                        clusTemp = j;
                    }
                }
                recorder[i] = clusTemp;

            #pragma omp critical
                {
                    storage[clusTemp]++;
                    neighborVec[clusTemp].push_back(i);
                    centerTemp.row(clusTemp)+=data.row(i);
                }
            }
        }
        moving = FLT_MIN;

    /* measure how much the current center moves from original center */
    #pragma omp parallel for reduction(max:moving) num_threads(8)
        for (int i = 0; i < Cluster; ++i)
        {
            if(storage[i]>0)
            {
                centerTemp.row(i)/=storage[i];
                tempMoving = (centerTemp.row(i)-clusterCenter.row(i)).norm();
                clusterCenter.row(i) = centerTemp.row(i);
                if(moving<tempMoving)
                    moving = tempMoving;
            }
        }
        std::cout << "K-means iteration " << ++tag << " completed, and moving is " << moving << "!" << std::endl;
    }while(abs(moving-before)/before >= 1.0e-2 && tag < 20 && moving > 0.01);

    double delta;

    std::multimap<int,int> groupMap;

    float entropy = 0.0, probability;
    int increasingOrder[Cluster];

    int nonZero = 0;
    for (int i = 0; i < Cluster; ++i)
    {
        groupMap.insert(std::pair<int,int>(storage[i],i));
        if(storage[i]>0)
        {
            probability=float(storage[i])/float(Row);
            entropy+=probability*log2f(probability);
            ++nonZero;
        }
    }
    entropy = -entropy/log2f(nonZero);

    int groupNo = 0;
    for (std::multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
    {
        if(it->first>0)
        {
            increasingOrder[it->second] = (groupNo++);
        }
    }
    std::cout << "There are " << groupNo << " groups generated!" << std::endl;
    /* finish tagging for each group */

    /* record labeling information */
    // IOHandler::generateGroups(neighborVec);

    // set cluster group number and size number
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Row; ++i)
    {
        group[i] = increasingOrder[recorder[i]];
        totalNum[i] = storage[recorder[i]];
    }

    float shortest, toCenter, farDist;
    int shortestIndex = 0, tempIndex = 0, furthestIndex = 0;
    std::vector<int> neighborTemp;

    /* choose cloest and furthest streamlines to centroid streamlines */
    for (int i = 0; i < Cluster; ++i)
    {
        if(storage[i]>0)
        {

            neighborTemp = neighborVec[i];
            shortest = FLT_MAX;
            farDist = FLT_MIN;

            for (int j = 0; j < storage[i]; ++j)
            {
                // j-th internal streamlines
                tempIndex = neighborTemp[j];
                toCenter = getDisimilarity(clusterCenter.row(i),data,tempIndex,normOption,object);

                /* update the closest index to centroid */
                if(toCenter<shortest)
                {
                    shortest = toCenter;
                    shortestIndex = tempIndex;
                }

                /* update the farthest index to centroid */
                if(toCenter>farDist)
                {
                    farDist = toCenter;
                    furthestIndex = tempIndex;
                }
            }
            closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
            furthest.push_back(ExtractedLine(furthestIndex,increasingOrder[i]));
            //distFile << std::endl;
        }
    }
    //distFile.close();

    std::vector<float> closeSubset;
    /* based on known cluster centroid, save them as vector for output */
    for (int i = 0; i < Cluster; ++i)
    {
        if(storage[i]>0)
        {
            for (int j = 0; j < Column; ++j)
            {
                closeSubset.push_back(clusterCenter(i,j));
            }
            massCenter.push_back(MeanLine(closeSubset,increasingOrder[i]));
            closeSubset.clear();
        }
    }
    delete[] storage;

    //groupNo record group numbers */

    if(groupNo<=1)
        return;

    /* if the dataset is not PBF, then should record distance matrix for Gamma matrix compution */
    if(!isPBF)
    {
        deleteDistanceMatrix(data.rows());

        std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
        if(distFile.fail())
        {
            distFile.close();
            getDistanceMatrix(data, normOption, object);
            std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
            for(int i=0;i<data.rows();++i)
            {
                for(int j=0;j<data.rows();++j)
                {
                    distFileOut << distanceMatrix[i][j] << " ";
                }
                distFileOut << std::endl;
            }
            distFileOut.close();
        }
        else
        {
            std::cout << "read distance matrix..." << std::endl;

            distanceMatrix = new float*[data.rows()];
        #pragma omp parallel for schedule(static) num_threads(8)
            for (int i = 0; i < data.rows(); ++i)
            {
                distanceMatrix[i] = new float[data.rows()];
            }
            int i=0, j;
            string line;
            stringstream ss;
            while(getline(distFile, line))
            {
                j=0;
                ss.str(line);
                while(ss>>line)
                {
                    if(i==j)
                        distanceMatrix[i][j]=0;
                    else
                        distanceMatrix[i][j] = std::atof(line.c_str());
                    ++j;
                }
                ++i;
                ss.str("");
                ss.clear();
            }
            distFile.close();
        }

        std::cout << "Distance between 0 and 1 is " << distanceMatrix[0][1] << std::endl;
    }

    gettimeofday(&start, NULL);

    sil.computeValue(normOption,data,Row,Column,group,object,groupNo,isPBF);

    gettimeofday(&end, NULL);
    delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;

    tr.eventList.push_back("For norm ");
    tr.timeList.push_back(to_string(normOption)+"\n");

    tr.eventList.push_back("Clustering evaluation computing takes: ");
    tr.timeList.push_back(to_string(delta)+"s");

    ValidityMeasurement vm;
    vm.computeValue(normOption, data, group, object, isPBF);

    tr.eventList.push_back("kmeans Validity measure is: ");
    stringstream fc_ss;
    fc_ss << vm.f_c;
    tr.timeList.push_back(fc_ss.str());

    /* write value of the silhouette class */
    IOHandler::writeReadme(entropy, sil, "For norm "+to_string(normOption));

}
```

4.4.2.7 void PCA_Cluster::performPC_KMeans (const MatrixXf & *cArray*, const int & *Row*, const int & *Column*, const int & *PC_Number*, const MatrixXf & *SingVec*, const VectorXf & *meanTrajectory*, std::vector< MeanLine > & *massCenter*, const int & *Cluster*, std::vector< int > & *group*, std::vector< int > & *totalNum*, std::vector< ExtractedLine > & *closest*, std::vector< ExtractedLine > & *furthest*, const Eigen::MatrixXf & *data*, TimeRecorder & *tr*, Silhouette & *sil*) [static], [private]

Definition at line 176 of file PCA_Cluster.cpp.

```
{
    MetricPreparation object(Row, Column);
    object.preprocessing(data, Row, Column, 0);
/* perform K-means clustering */
    MatrixXf clusterCenter;

    switch(initializationOption)
    {
    case 1:
        Initialization::generateRandomPos(clusterCenter, PC_Number, cArray, Cluster);
        break;

    case 2:
        Initialization::generateFromSamples(clusterCenter, PC_Number, cArray, Cluster);
        break;

    case 3:
        Initialization::generateFarSamples(clusterCenter, PC_Number, cArray,
                                           Cluster, 0, object);
        break;
    }

    float moving=1000, tempMoving, before;
    int storage[Cluster];

    MatrixXf centerTemp;  //store provisional center coordinate

    int tag = 0;

    std::vector< std::vector<int> > neighborVec(Cluster, std::vector<int>());

    double PCA_KMeans_delta, KMeans_delta;
    struct timeval start, end;

    gettimeofday(&start, NULL);

    std::vector<int> recorder(Row);
    do
    {
        before = moving;
        /* preset cluster number recorder */
        memset(storage,0,sizeof(int)*Cluster);
        centerTemp = MatrixXf::Zero(Cluster, PC_Number);

    #pragma omp parallel for schedule(static) num_threads(8)
        for (int i = 0; i < Cluster; ++i)
        {
            neighborVec[i].clear();
        }

    #pragma omp parallel num_threads(8)
        {
        #pragma omp for nowait
            for (int i = 0; i < Row; ++i)
            {
                float dist = FLT_MAX;
                float temp;
                int clusTemp;
                for (int j = 0; j < Cluster; ++j)
                {
                    temp = (cArray.row(i)-clusterCenter.row(j)).norm();
                    if(temp<dist)
                    {
                        dist = temp;
                        clusTemp = j;
                    }
                }

            #pragma omp critical
                {
                    storage[clusTemp]++;
                    neighborVec[clusTemp].push_back(i);
                    recorder[i] = clusTemp;
                    centerTemp.row(clusTemp)+=cArray.row(i);
                }
            }
        }

        moving = FLT_MIN;

    #pragma omp parallel for reduction(max:moving) num_threads(8)
        for (int i = 0; i < Cluster; ++i)
        {
            if(storage[i]>0)
            {
                centerTemp.row(i)/=storage[i];
                tempMoving = (centerTemp.row(i)-clusterCenter.row(i)).norm();
                clusterCenter.row(i) = centerTemp.row(i);
                if(moving<tempMoving)
                    moving = tempMoving;
            }
        }
        std::cout << "K-means iteration " << ++tag << " completed, and moving is " << moving << "!" << std::endl;
    }while(abs(moving-before)/before >= 1.0e-2 && tag < 20 && moving>0.01);

    gettimeofday(&end, NULL);

    float delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;

    tr.eventList.push_back("k-means iteration for PC takes: ");
    tr.timeList.push_back(to_string(delta)+"s");

    std::multimap<int,int> groupMap;

    float entropy = 0.0;
    float probability;


    for (int i = 0; i < Cluster; ++i)
    {
        groupMap.insert(std::pair<int,int>(storage[i],i));
        if(storage[i]>0)
        {
            probability = float(storage[i])/float(Row);
            entropy += probability*log2f(probability);
        }
    }

    int groupNo = 0;
    int increasingOrder[Cluster];
    for (multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
    {
        if(it->first>0)
        {
            increasingOrder[it->second] = (groupNo++);
        }
    }

    /* calculate the balanced entropy */
    entropy = -entropy/log2f(groupNo);


#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Row; ++i)
    {
        group[i] = increasingOrder[recorder[i]];
        totalNum[i] = storage[recorder[i]];
    }

    float shortest, farDist, toCenter;
    int shortestIndex = 0, fartestIndex = 0, tempIndex = 0;
    std::vector<int> neighborTemp;

    for (int i = 0; i < Cluster; ++i)
    {
        if(storage[i]>0 && !neighborVec[i].empty())
        {
            neighborTemp = neighborVec[i];
            shortest = FLT_MAX;
            farDist = FLT_MIN;

            for (int j = 0; j < storage[i]; ++j)
            {
                tempIndex = neighborTemp[j];
                toCenter = (clusterCenter.row(i)-cArray.row(tempIndex)).norm();

                if(toCenter<shortest)
                {
                    shortest = toCenter;
                    shortestIndex = tempIndex;
                }
                if(toCenter>farDist)
                {
                    farDist = toCenter;
                    fartestIndex = tempIndex;
                }
            }
            closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
            furthest.push_back(ExtractedLine(fartestIndex,increasingOrder[i]));
        }
    }
    MatrixXf pcSing(PC_Number,Column);

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < PC_Number; ++i)
    {
        pcSing.row(i) = SingVec.row(i);
    }

    MatrixXf massPos = clusterCenter*pcSing;

    for (int i = 0; i < Cluster; ++i)
    {
        if(storage[i]>0)
        {
            massPos.row(i) += meanTrajectory.transpose();
            std::vector<float> vecTemp;
            for (int j = 0; j < Column; ++j)
            {
                vecTemp.push_back(massPos(i,j));
            }
            massCenter.push_back(MeanLine(vecTemp,increasingOrder[i]));
        }
    }

    ValidityMeasurement vm;
    vm.computeValue(cArray, group);

    tr.eventList.push_back("PCA Validity measure is: ");
    stringstream fc_ss;
    fc_ss << vm.f_c;
    tr.timeList.push_back(fc_ss.str());

    /* Silhouette effect */
    gettimeofday(&start, NULL);

    sil.computeValue(cArray,group,groupNo,isPBF);

    gettimeofday(&end, NULL);
    delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;

    tr.eventList.push_back("Clustering evaluation computing takes: ");
    tr.timeList.push_back(to_string(delta)+"s");

    /* write value of the silhouette class */
    IOHandler::writeReadme(entropy, sil, "");

}
```

4.4.2.8 void PCA_Cluster::performPCA_Clustering (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*, std::vector< MeanLine > & *massCenter*, std::vector< int > & *group*, std::vector< int > & *totalNum*, std::vector< ExtractedLine > & *closest*, std::vector< ExtractedLine > & *furthest*, TimeRecorder & *tr*, Silhouette & *sil*) [static]

Definition at line 53 of file PCA_Cluster.cpp.

```
{
    MatrixXf cArray, SingVec;
    VectorXf meanTrajectory(Column);
    int PC_Number;

    performSVD(cArray, data, Row, Column, PC_Number, SingVec, meanTrajectory, tr);

    if(post_processing==1)
        performPC_KMeans(cArray, Row, Column, PC_Number, SingVec, meanTrajectory,
                         massCenter, CLUSTER, group, totalNum, closest, furthest, data, tr, sil);
    else if(post_processing==2)
        perform_AHC(cArray, PC_Number, SingVec, meanTrajectory,
                    massCenter, CLUSTER, group, totalNum, closest, furthest, data, tr, sil);
}
```

4.4.2.9 void PCA_Cluster::performPCA_Clustering (const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column,* std::vector< MeanLine > & *massCenter,* std::vector< int > & *group,* std::vector< int > & *totalNum,* std::vector< ExtractedLine > & *closest,* std::vector< ExtractedLine > & *furthest,* const int & *Cluster,* TimeRecorder & *tr,* Silhouette & *sil*) [static]

Definition at line 462 of file PCA_Cluster.cpp.

```
{
    MatrixXf cArray, SingVec;
    VectorXf meanTrajectory(Column);
    int PC_Number;

    performSVD(cArray, data, Row, Column, PC_Number, SingVec, meanTrajectory, tr);
    if(post_processing==1)
        performPC_KMeans(cArray, Row, Column, PC_Number, SingVec, meanTrajectory,
                         massCenter, Cluster, group, totalNum, closest, furthest, data, tr, sil);
    else if(post_processing==2)
        perform_AHC(cArray, PC_Number, SingVec, meanTrajectory,
                    massCenter, Cluster, group, totalNum, closest, furthest, data, tr, sil);
}
```

4.4.2.10 void PCA_Cluster::performSVD (MatrixXf & *cArray,* const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column,* int & *PC_Number,* MatrixXf & *SingVec,* VectorXf & *meanTrajectory,* TimeRecorder & *tr*) [static], [private]

Definition at line 94 of file PCA_Cluster.cpp.

```
{
    Eigen::MatrixXf temp = data;

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Column; ++i)
    {
        meanTrajectory(i) = temp.transpose().row(i).mean();
    }
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Row; ++i)
    {
        temp.row(i) = temp.row(i)-meanTrajectory.transpose();
    }

    struct timeval start, end;
    gettimeofday(&start, NULL);
    /* perform SVD decomposition for temp */
    JacobiSVD<MatrixXf> svd(temp, ComputeThinU | ComputeThinV);
    //const VectorXf& singValue = svd.singularValues();
    SingVec = svd.matrixV();
    gettimeofday(&end, NULL);
    const double& delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;

    tr.eventList.push_back("SVD takes: ");
    tr.timeList.push_back(to_string(delta)+"s");

    /* compute new attribute space based on principal component */
    MatrixXf coefficient = temp*SingVec;
    /*  decide first r dorminant PCs with a threshold */
    const float& varianceSummation = coefficient.squaredNorm();
    float tempSum = 0.0;
    const float& threshold = TOR_1*varianceSummation;

    for (int i = 0; i < Column; ++i)
    {
        tempSum+=(coefficient.transpose().row(i)).squaredNorm();
        if(tempSum>threshold)
        {
            PC_Number = i;
            break;
        }
    }

    cArray = MatrixXf(Row, PC_Number);
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < PC_Number; ++i)
    {
        cArray.transpose().row(i) = coefficient.transpose().row(i);
    }

    std::cout << "SVD completed!" << std::endl;

    SingVec.transposeInPlace();
}
```

4.4.2.11 void PCA_Cluster::setLabel (const std::vector< AHC_node > & *nodeVec*, vector< vector< int > > & *neighborVec*, vector< int > & *storage*, Eigen::MatrixXf & *centroid*, const Eigen::MatrixXf & *cArray*, std::vector< int > & *recorder*) [static], [private]

Definition at line 1217 of file PCA_Cluster.cpp.

```
{
// group tag by increasing order
    int groupID = 0;

    // element list for each group
    vector<int> eachContainment;

    // find group id and neighboring vec
    for(auto iter = nodeVec.begin(); iter!=nodeVec.end();++iter)
    {
        eachContainment = (*iter).element;
        neighborVec[groupID] = eachContainment;
    #pragma omp parallel num_threads(8)
        {
        #pragma omp for nowait
            for(int i=0;i<eachContainment.size();++i)
            {
                recorder[eachContainment[i]] = groupID;
            #pragma omp critical
                centroid.row(groupID) += cArray.row(eachContainment[i]);
            }
        }
        storage[groupID] = (*iter).element.size();
        centroid.row(groupID)/=eachContainment.size();
        ++groupID;
        eachContainment.clear();
    }
}
```

4.4.2.12 void PCA_Cluster::setValue (std::vector< DistNode > & *dNodeVec*, const Eigen::MatrixXf & *reduced_data*, const Eigen::MatrixXf & *reduced_dist_matrix*) [static], [private]

Definition at line 1187 of file PCA_Cluster.cpp.

```
{
    const int& Row = reduced_data.rows();
    dNodeVec = std::vector<DistNode>(Row*(Row-1)/2);
    int tag = 0;
    for(int i=0;i<Row-1;++i)
    {
        for(int j=i+1;j<Row;++j)
        {
            dNodeVec[tag].first = i;
            dNodeVec[tag].second = j;
            dNodeVec[tag].distance = reduced_dist_matrix(i, j);
            ++tag;
        }
    }
    assert(tag==dNodeVec.size());
}
```

The documentation for this class was generated from the following files:

- PCA_Cluster.h
- PCA_Cluster.cpp

4.5 TimeRecorder Struct Reference

```
#include <PCA_Cluster.h>
```

Public Attributes

- std::vector< string > eventList
- std::vector< string > timeList

4.5.1 Detailed Description

Definition at line 37 of file PCA_Cluster.h.

4.5.2 Member Data Documentation

4.5.2.1 std::vector<string> TimeRecorder::eventList

Definition at line 39 of file PCA_Cluster.h.

4.5.2.2 std::vector<string> TimeRecorder::timeList

Definition at line 40 of file PCA_Cluster.h.

The documentation for this struct was generated from the following file:

- PCA_Cluster.h

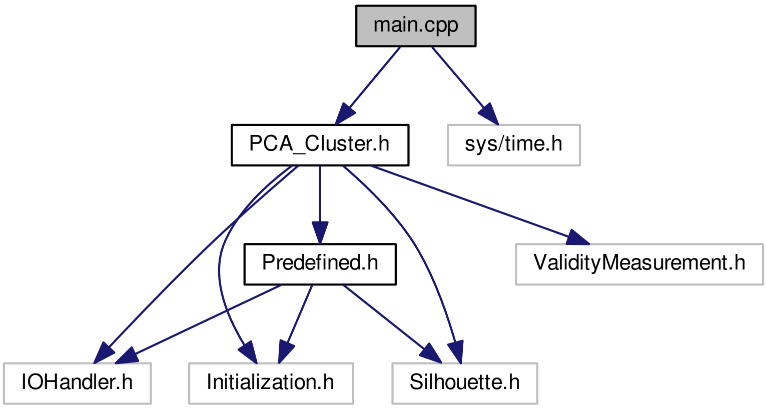
Chapter 5

File Documentation

5.1 main.cpp File Reference

```
#include "PCA_Cluster.h"
#include <sys/time.h>
```

Include dependency graph for main.cpp:



Functions

- void featureExtraction (const int &argc, char ∗∗argv)
- void performPCA_Cluster (const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &cluster, const int &dimension, const string &fullName, const int &maxElements, const Eigen::MatrixXf &data, TimeRecorder &tr, Silhouette &sil)
- void performK_Means (const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &cluster, const int &dimension, const string &fullName, const int &maxElements, const Eigen::MatrixXf &data, const int &normOption, TimeRecorder &tr, Silhouette &sil)
- int main (int argc, char ∗argv[ ])

Variables

- int initializationOption
- bool isPBF
- int post_processing
- bool readCluster

5.1.1 Function Documentation

5.1.1.1 void featureExtraction (const int & *argc,* char ∗∗ *argv*)

Definition at line 105 of file main.cpp.

```
{
    while(number!=3)
    {
        std::cout << "Input argument should have 3!" << endl
                  << "./cluster inputFile_name(in dataset folder) "
                  << "data_dimension(3)" << endl;
        exit(1);
    }
    const string& strName = string("../dataset/")+string(argv[1]);
    //const string& strName = "../dataset/pbfDataset";
    const int& dimension = atoi(argv[2]);
    //const string& pbfPath = "/media/lieyu/Seagate Backup Plus Drive/PBF_2013Macklin/pbf_velocitySeparate/source_data/Frame ";

    //std::cout << strName << std::endl;
    //fullName = "../dataset/streamlines_cylinder_9216_full.vtk";
    //const string& strName = "../dataset/streamlines_cylinder_9216";
    //const string& strName = "../dataset/pbf_data";

    std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
    int PBFjudgement;
    std::cin >> PBFjudgement;
    assert(PBFjudgement==1||PBFjudgement==0);
    isPBF = (PBFjudgement==1);

    /* input for judge whether it is a pathline data set so that MCP can be called */
    bool isPathlines;
    std::cout << "It is pathlines? 1.Yes, 0.No" << std::endl;
    std::cin >> PBFjudgement;
    assert(PBFjudgement==1||PBFjudgement==0);
    isPathlines = (PBFjudgement==1);

    /* set how many clusters and max vertex count of the data set */
    int cluster, vertexCount;

    /* choose k-means initialization method, 2 is often adopted providing better visualization effect */
    std::cout << "Please choose initialization option for seeds:" << std::endl
              << "1.chose random positions, 2.Chose from samples, 3.k-means++ sampling" << endl;
    std::cin >> initializationOption;
    assert(initializationOption==1 || initializationOption==2
           || initializationOption==3);

    int samplingMethod;
    if(isPathlines)
        samplingMethod = 1;

    else
    {
        /* select sampling strategy, and 2 is often for geometric clustering */
        std::cout << "Please choose sampling strategy: " << std::endl
                  << "1.directly filling, 2.uniformly sampling" << std::endl;
        std::cin >> samplingMethod;
    }

    assert(samplingMethod==1 || samplingMethod==2);

    /* whether number of clusters is read from user input or from ../dataset/cluster_number */
    std::cout << "Please choose cluster number method, 0.user input, 1.read clustering: " << std::endl;
    int clusterInput;
    std::cin >> clusterInput;
    assert(clusterInput==0 || clusterInput==1);
    readCluster = (clusterInput==1);

```

```
  std::unordered_map<int,int> clusterMap;
  if(readCluster)
  {
      IOHandler::readClusteringNumber(clusterMap, "cluster_number");
  }


  TimeRecorder tr;

  Silhouette sil;

  struct timeval start, end;
  double timeTemp;
  int maxElements;

  gettimeofday(&start, NULL);
  std::vector< std::vector<float> > dataVec;
  IOHandler::readFile(strName, dataVec, vertexCount, dimension, maxElements);
  gettimeofday(&end, NULL);
  timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
             + end.tv_usec - start.tv_usec) / 1.e6;
  tr.eventList.push_back("I-O file reader takes: ");
  tr.timeList.push_back(to_string(timeTemp)+"s");

  if(!readCluster)
  {
      std::cout << "Please input a cluster number (>=2) among [2, " << dataVec.size() << "]: " << std::endl;
      std::cin >> cluster;
  }
  else
  {
      cluster = clusterMap[0];
  }

  stringstream ss;
  ss << strName << "_differentNorm_full.vtk";
  const string& fullName = ss.str();
  IOHandler::printVTK(ss.str(), dataVec, vertexCount, dimension);
  ss.str("");

  Eigen::MatrixXf data;

  /* PCA computation is always using brute-force filling arrays by last point */
  IOHandler::expandArray(data, dataVec, dimension, maxElements);

  std::cout << "PCA-based clustering starts..." << std::endl;
  ss << strName << "_PCAClustering";
  gettimeofday(&start, NULL);
  performPCA_Cluster(ss.str(), dataVec, cluster, dimension, fullName, maxElements, data, tr, sil);
  std::cout << "Max element is " << maxElements << std::endl;
  ss.str("");
  ss.clear();
  gettimeofday(&end, NULL);
  timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
             + end.tv_usec - start.tv_usec) / 1.e6;

  tr.eventList.push_back("PCA+KMeans takes: ");
  tr.timeList.push_back(std::to_string(timeTemp)+"s");

  IOHandler::writeReadme(tr.eventList, tr.timeList, cluster);

  tr.eventList.clear();
  tr.timeList.clear();


  sil.reset();

  /*  0: Euclidean Norm
      1: Fraction Distance Metric
      2: piece-wise angle average
      3: Bhattacharyya metric for rotation
      4: average rotation
      5: signed-angle intersection
      6: normal-direction multivariate distribution
      7: Bhattacharyya metric with angle to a fixed direction
      8: Piece-wise angle average \times standard deviation
      9: normal-direction multivariate un-normalized distribution
      10: x*y/|x||y| borrowed from machine learning
      11: cosine similarity
      12: Mean-of-closest point distance (MCP)
      13: Hausdorff distance min_max(x_i,y_i)
      14: Signature-based measure taken from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
      15: Procrustes distance taken from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
      16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
```

```
        17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/S0097849318300128
            for pathlines only
    */
    if(samplingMethod==2)
        IOHandler::sampleArray(data, dataVec, dimension, maxElements);

    for(int i = 0;i<=17;i++)
    {
        if(isPathlines)
        {
            if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15 && i!=17)
                continue;
        }

        else
        {
            if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
                continue;
        }

        if(readCluster)
            cluster = clusterMap[i];
        else
        {
            std::cout << "Please input a cluster number (>=2) for norm " << i << " in [2, "
                    << dataVec.size() << "]: " << std::endl;
            std::cin >> cluster;
        }

        std::cout << "Kmeans on norm " << i << " starts..." << std::endl;
        gettimeofday(&start, NULL);
        ss << strName << "_KMeans";
        performK_Means(ss.str(), dataVec, cluster, dimension, fullName, maxElements, data,i, tr, sil);

        ss.str("");
        gettimeofday(&end, NULL);
        timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
                    + end.tv_usec - start.tv_usec) / 1.e6;
        tr.eventList.push_back("K-means on norm "+to_string(i)+" takes: ");
        tr.timeList.push_back(to_string(timeTemp)+"s");

        IOHandler::writeReadme(tr.eventList, tr.timeList, cluster);

        tr.eventList.clear();
        tr.timeList.clear();
        sil.reset();
    }
}
```

5.1.1.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 92 of file main.cpp.

```
{
    featureExtraction(argc, argv);
    return 0;
}
```

5.1.1.3 void performK_Means (const string & *fileName,* const std::vector< std::vector< float > > & *dataVec,* const int & *cluster,* const int & *dimension,* const string & *fullName,* const int & *maxElements,* const Eigen::MatrixXf & *data,* const int & *normOption,* TimeRecorder & *tr,* Silhouette & *sil*)

Definition at line 405 of file main.cpp.

```
{
    std::vector<MeanLine> centerMass;
    std::vector<ExtractedLine> closest;
    std::vector<ExtractedLine> furthest;
    std::vector<int> group(dataVec.size());
420     std::vector<int> totalNum(dataVec.size());
421     PCA_Cluster::performDirectK_Means(data, dataVec.size(), maxElements,
422                                       centerMass, group, totalNum,
423                                       closest, furthest, cluster, normOption, tr, sil);
424 
425     std::vector<std::vector<float> > closestStreamline, furthestStreamline;
426     std::vector<int> closestCluster, furthestCluster, meanCluster;
427     int closestPoint, furthestPoint;
428     IOHandler::assignVec(closestStreamline, closestCluster, closest, closestPoint, dataVec);
429     IOHandler::assignVec(furthestStreamline, furthestCluster, furthest,
430                          furthestPoint, dataVec);
431 
432 /* get the average rotation of the extraction */
433     std::vector<float> closestRotation, furthestRotation;
434     const float& closestAverage = getRotation(closestStreamline, closestRotation);
435     const float& furthestAverage = getRotation(furthestStreamline, furthestRotation);
436 
437     tr.eventList.push_back("Average rotation of closest for K-means clustering on norm "
438                             + to_string(normOption) + " is: ");
439     tr.timeList.push_back(to_string(closestAverage));
440 
441     tr.eventList.push_back("Average rotation of furthest for K-means clustering on norm "
442                             + to_string(normOption) + " is: ");
443     tr.timeList.push_back(to_string(furthestAverage));
444 /* finish the rotation computation */
445 
446 
447     IOHandler::assignVec(meanCluster, centerMass);
448     IOHandler::printVTK(fileName+string("_norm")+to_string(normOption)+string("_mean.vtk"),
449                         centerMass,
450                         centerMass.size()*centerMass[0].minCenter.size()/dimension,
451                         dimension, sil.sCluster);
452     IOHandler::printVTK(fileName+"_norm"+to_string(normOption)+"_closest.vtk",
453                         closestStreamline, closestPoint/dimension, dimension,
454                         closestCluster, sil.sCluster);
455     IOHandler::printVTK(fileName+"_norm"+to_string(normOption)+"_furthest.vtk",
456                         furthestStreamline, furthestPoint/dimension,
457                         dimension, furthestCluster, sil.sCluster);
458     std::cout << "Finish printing vtk for k-means clustering result!" << std::endl;
459 
460     IOHandler::printToFull(dataVec, group, totalNum, string("norm")+to_string(normOption)
461                             +string("_KMeans"), fullName, dimension);
462 
463     //IOHandler::writeReadme(closest, furthest, normOption);
464 
465     IOHandler::printToFull(dataVec, sil.sData, "norm"+to_string(normOption)+"_SValueLine",
466                             fullName, 3);
467     IOHandler::printToFull(dataVec, group, sil.sCluster, "norm"+to_string(normOption)+"_SValueCluster",
      fullName, 3);
468 
469     centerMass.clear();
470     closest.clear();
471     furthest.clear();
472     group.clear();
473     totalNum.clear();
474 }
```

5.1.1.4 void performPCA_Cluster (const string & *fileName,* const std::vector< std::vector< float > > & *dataVec,* const int & *cluster,* const int & *dimension,* const string & *fullName,* const int & *maxElements,* const Eigen::MatrixXf & *data,* TimeRecorder & *tr,* Silhouette & *sil*)

Definition at line 314 of file main.cpp.

```
323 {
324 
325     std::vector<MeanLine> centerMass;
326     std::vector<int> group(dataVec.size());
327     std::vector<ExtractedLine> closest;
328     std::vector<ExtractedLine> furthest;
329     std::vector<int> totalNum(dataVec.size());
330 
331     // choose an appropriate post processing technique for PCA rank space
332     std::cout << "Please select a post-processing: 1. k-means, 2. AHC-average." << std::endl;
333     std::cin >> post_processing;
334     assert(post_processing==1 || post_processing==2);
335 
336     PCA_Cluster::performPCA_Clustering(data, dataVec.size(), maxElements,
       centerMass,
337                         group, totalNum, closest, furthest, cluster, tr, sil);
```

```

    std::vector<std::vector<float> > closestStreamline;
    std::vector<std::vector<float> > furthestStreamline;
    std::vector<int> closestCluster, furthestCluster, meanCluster;
    int closestPoint, furthestPoint;

    IOHandler::assignVec(closestStreamline, closestCluster, closest, closestPoint, dataVec);

    IOHandler::assignVec(furthestStreamline, furthestCluster, furthest,
                         furthestPoint, dataVec);

/* get the average rotation of the extraction */
    std::vector<float> closestRotation, furthestRotation;
    const float& closestAverage = getRotation(closestStreamline, closestRotation);
    const float& furthestAverage = getRotation(furthestStreamline, furthestRotation);

    tr.eventList.push_back("Average rotation of closest for PCA clustering is: ");
    tr.timeList.push_back(to_string(closestAverage));

    tr.eventList.push_back("Average rotation of furthest for PCA clustering is: ");
    tr.timeList.push_back(to_string(furthestAverage));
/* finish the rotation computation */

    IOHandler::assignVec(meanCluster, centerMass);

    IOHandler::printVTK(fileName+string("_PCA_closest.vtk"), closestStreamline,
                        closestPoint/dimension, dimension, closestCluster,
                        sil.sCluster);

    IOHandler::printVTK(fileName+string("_PCA_furthest.vtk"), furthestStreamline,
                        furthestPoint/dimension, dimension, furthestCluster,
                        sil.sCluster);

    IOHandler::printVTK(fileName+string("_PCA_mean.vtk"), centerMass,
                        centerMass.size()*centerMass[0].minCenter.size()/dimension,
                        dimension, sil.sCluster);

    std::cout << "Finish printing vtk for pca-clustering result!" << std::endl;

    if(post_processing==1)
        IOHandler::printToFull(dataVec, group, totalNum, string("PCA_KMeans"), fullName, dimension);
    else if(post_processing==2)
        IOHandler::printToFull(dataVec, group, totalNum, string("PCA_AHC"), fullName, dimension);

    //IOHandler::writeReadme(closest, furthest);

    IOHandler::printToFull(dataVec, sil.sData, "PCA_SValueLine", fullName, 3);

    IOHandler::printToFull(dataVec, group, sil.sCluster, "PCA_SValueCluster",
                           fullName, 3);
}
```

5.1.2 Variable Documentation

5.1.2.1 int initializationOption

Definition at line 74 of file main.cpp.

5.1.2.2 bool isPBF

Definition at line 79 of file main.cpp.

5.1.2.3 int post_processing

Definition at line 84 of file main.cpp.

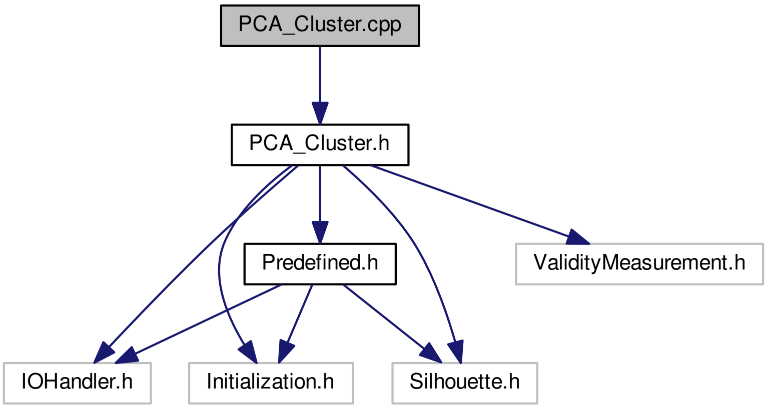
5.1.2.4 bool readCluster

Definition at line 89 of file main.cpp.

5.2 PCA_Cluster.cpp File Reference

```
#include "PCA_Cluster.h"
```

Include dependency graph for PCA_Cluster.cpp:



Variables

- const float & TOR_1 = 0.999
- const int & CLUSTER = 8
- int initializationOption
- int post_processing
- bool isPBF

5.2.1 Variable Documentation

5.2.1.1 const int& CLUSTER = 8

Definition at line 18 of file PCA_Cluster.cpp.

5.2.1.2 int initializationOption

Definition at line 74 of file main.cpp.

5.2.1.3 bool isPBF

Definition at line 79 of file main.cpp.

5.2.1.4 int post_processing

Definition at line 84 of file main.cpp.

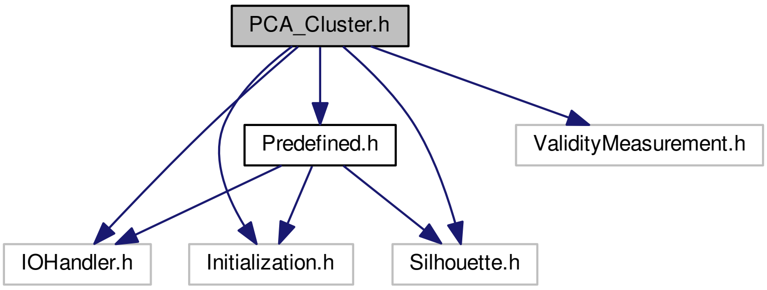
5.2.1.5 const float& TOR_1 = 0.999

Definition at line 13 of file PCA_Cluster.cpp.

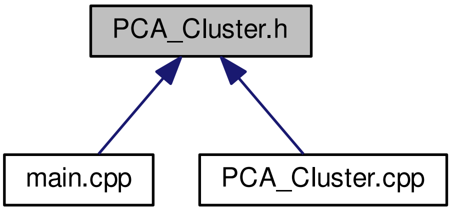
5.3 PCA_Cluster.h File Reference

```
#include "IOHandler.h"
#include "Initialization.h"
#include "Silhouette.h"
#include "ValidityMeasurement.h"
#include "Predefined.h"
```

Include dependency graph for PCA_Cluster.h:



This graph shows which files directly or indirectly include this file:



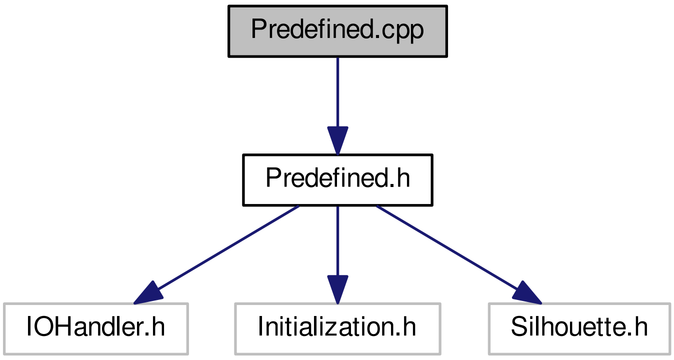
Classes

- struct Ensemble
- struct TimeRecorder
- class PCA_Cluster

5.4 Predefined.cpp File Reference

`#include "Predefined.h"`

Include dependency graph for Predefined.cpp:



Functions

- template<class T >
  void deleteVecElements (std::vector< T > &original, const T &first, const T &second)

5.4.1 Function Documentation

5.4.1.1 template<class T > void deleteVecElements (std::vector< T > & *original,* const T & *first,* const T & *second*)

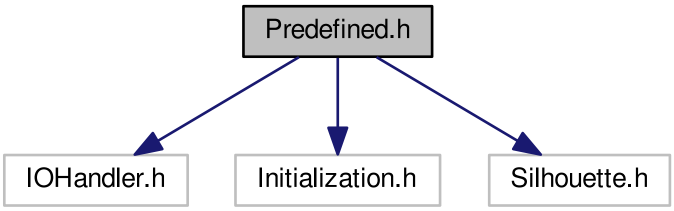
Definition at line 19 of file Predefined.cpp.

```
{
    std::size_t size = original.size();
    assert(size>2);
    vector<T> result(size-2);
    int tag = 0;
    for(int i=0;i<size;++i)
    {
        //meet with target elements, not copied
        if(original[i]==first || original[i]==second)
            continue;
        result[tag++]=original[i];
    }
    assert(tag==size-2);
    original = result;
}
```

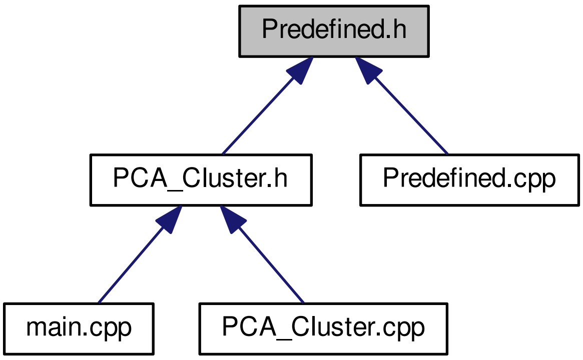
5.5 Predefined.h File Reference

```
#include "IOHandler.h"
#include "Initialization.h"
#include "Silhouette.h"
```

Include dependency graph for Predefined.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct AHC_node
- struct DistNode

Functions

- template<class T >
  void deleteVecElements (std::vector< T > &origine, const T &first, const T &second)

5.5.1 Function Documentation

5.5.1.1 template< class T > void deleteVecElements (std::vector< T > & *origine,* const T & *first,* const T & *second*)

Definition at line 19 of file Predefined.cpp.

```
{
    std::size_t size = original.size();
    assert(size>2);
    vector<T> result(size-2);
    int tag = 0;
    for(int i=0;i<size;++i)
    {
        //meet with target elements, not copied
        if(original[i]==first || original[i]==second)
            continue;
        result[tag++]=original[i];
    }
    assert(tag==size-2);
    original = result;
}
```

5.6 README.md File Reference

Index

AHC_node, 7
- AHC_node, 7
- element, 8
- index, 8

CLUSTER
- PCA_Cluster.cpp, 33

deleteVecElements
- Predefined.cpp, 35
- Predefined.h, 37

DistNode, 8
- DistNode, 9
- distance, 9
- first, 9
- second, 9

distance
- DistNode, 9

element
- AHC_node, 8

Ensemble, 9
- Ensemble, 10
- newIndex, 10
- number, 10
- oldIndex, 10
- operator<, 10

eventList
- TimeRecorder, 25

featureExtraction
- main.cpp, 28

first
- DistNode, 9

getDistAtNodes
- PCA_Cluster, 12

hierarchicalMerging
- PCA_Cluster, 12

index
- AHC_node, 8

initializationOption
- main.cpp, 32
- PCA_Cluster.cpp, 33

isPBF
- main.cpp, 32
- PCA_Cluster.cpp, 33

main
- main.cpp, 30

main.cpp, 27
- featureExtraction, 28
- initializationOption, 32
- isPBF, 32
- main, 30
- performK_Means, 30
- performPCA_Cluster, 31
- post_processing, 32
- readCluster, 32

newIndex
- Ensemble, 10

number
- Ensemble, 10

oldIndex
- Ensemble, 10

operator<
- Ensemble, 10

PCA_Cluster, 11
- getDistAtNodes, 12
- hierarchicalMerging, 12
- perform_AHC, 13
- performDirectK_Means, 16
- performFullK_MeansByClusters, 16
- performPC_KMeans, 19
- performPCA_Clustering, 22
- performSVD, 23
- setLabel, 24
- setValue, 24

PCA_Cluster.cpp, 33
- CLUSTER, 33
- initializationOption, 33
- isPBF, 33
- post_processing, 33
- TOR_1, 33

PCA_Cluster.h, 34

perform_AHC
- PCA_Cluster, 13

performDirectK_Means
- PCA_Cluster, 16

performFullK_MeansByClusters
- PCA_Cluster, 16

performK_Means
- main.cpp, 30

performPC_KMeans
- PCA_Cluster, 19

performPCA_Cluster

main.cpp, 31
performPCA_Clustering
PCA_Cluster, 22
performSVD
PCA_Cluster, 23
post_processing
main.cpp, 32
PCA_Cluster.cpp, 33
Predefined.cpp, 35
deleteVecElements, 35
Predefined.h, 36
deleteVecElements, 37

README.md, 37
readCluster
main.cpp, 32

second
DistNode, 9
setLabel
PCA_Cluster, 24
setValue
PCA_Cluster, 24

TOR_1
PCA_Cluster.cpp, 33
timeList
TimeRecorder, 25
TimeRecorder, 25
eventList, 25
timeList, 25