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 > &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) . . . 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 &mean↩ Trajectory, 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 &mean↩ Trajectory, 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::Matrix↩ Xf &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 &full↩ Name, 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.

```
1158 {
1159     const int& m = firstList.size();
1160     const int& n = secondList.size();
1161     assert(m!=0);
1162     assert(n!=0);
1163
1164     float result, value;
1165     result = 0;
1166 #pragma omp parallel for reduction(+:result) num_threads(8)
1167     for(int i=0;i<m;++i)
1168     {
1169         for(int j=0;j<n;++j)
1170         {
1171             value = reduced_dist_matrix(i,j);
1172             result+=value;
1173         }
1174     }
1175     result/=m*n;
1176     return result;
1177 }
```

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.

```
1030 {
1031     /* would store distance matrix instead because it would save massive time */
1032     struct timeval start, end;
1033     double timeTemp;
1034     gettimeofday(&start, NULL);
1035
1036     const int Row = cArray.rows();
1037
1038     for(int i=0;i<Row;++i)
1039     {
1040         nodeMap[i].element.push_back(i);
1041     }
1042
1043     DistNode poped;
1044
1045     /* find node-pair with minimal distance */
1046     float minDist = FLT_MAX;
1047     int target = -1;
1048     for (int i = 0; i < dNodeVec.size(); ++i)
1049     {
1050         if(dNodeVec[i].distance<minDist)
1051         {
1052             target = i;
1053             minDist = dNodeVec[i].distance;
1054         }
1055     }
1056     poped = dNodeVec[target];
1057
1058     int index = Row, currentNumber;
1059     do
1060     {
1061         //create new node merged and input it into hash map
1062         vector<int> first = (nodeMap[poped.first]).element;
1063         vector<int> second = (nodeMap[poped.second]).element;
1064
1065         /* index would be starting from Row */
1066         AHC_node newNode(index);
1067         newNode.element = first;
1068         newNode.element.insert(newNode.element.end(), second.begin(), second.end());
```

```
1069         nodeMap.insert(make_pair(index, newNode));
1070 
1071         //delete two original nodes
1072         nodeMap.erase(poped.first);
1073         nodeMap.erase(poped.second);
1074 
1075         /* the difficulty lies how to update the min-heap with linkage
1076          * This would take 2NlogN.
1077          * Copy all node-pairs that are not relevant to merged nodes to new vec.
1078          * For relevant, would update the mutual distance by linkage
1079          */
1080 
1081         /* how many clusters exist */
1082         currentNumber = nodeMap.size();
1083 
1084         target = -1, minDist = FLT_MAX;
1085 
1086         std::vector<DistNode> tempVec(currentNumber*(currentNumber-1)/2);
1087         int current = 0, i_first, i_second;
1088         for(int i=0;i<dNodeVec.size();++i)
1089         {
1090             i_first=dNodeVec[i].first, i_second=dNodeVec[i].second;
1091             /* not relevant, directly copied to new vec */
1092             if(i_first!=poped.first&&i_first!=poped.second&&i_second!=poped.
     first&&i_second!=poped.second)
1093             {
1094                 tempVec[current]=dNodeVec[i];
1095                 if(tempVec[current].distance<minDist)
1096                 {
1097                     target = current;
1098                     minDist = tempVec[current].distance;
1099                 }
1100                 ++current;
1101             }
1102         }
1103 
1104         for (auto iter=nodeMap.begin();iter!=nodeMap.end();++iter)
1105         {
1106             if((*iter).first!=newNode.index)
1107             {
1108                 tempVec[current].first = (*iter).first;
1109                 tempVec[current].second = newNode.index;
1110                 tempVec[current].distance=getDistAtNodes(newNode.element,(*iter).second.
     element, reduced_dist_matrix);
1111                 if(tempVec[current].distance<minDist)
1112                 {
1113                     target = current;
1114                     minDist = tempVec[current].distance;
1115                 }
1116                 ++current;
1117             }
1118         }
1119         poped = tempVec[target];
1120 
1121         /* judge whether current is assigned to right value */
1122         assert(current==tempVec.size());
1123         dNodeVec.clear();
1124         dNodeVec = tempVec;
1125         tempVec.clear();
1126         ++index;
1127     }while(nodeMap.size()!=numberOfClusters);   //merging happens whenever requested cluster is not met
1128 
1129     nodeVec=std::vector<AHC_node>(nodeMap.size());
1130     int tag = 0;
1131     for(auto iter=nodeMap.begin();iter!=nodeMap.end();++iter)
1132         nodeVec[tag++]=(*iter).second;
1133 
1134     gettimeofday(&end, NULL);
1135     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
1136 
1137     tr.eventList.push_back("Hirarchical clustering for "+to_string(numberOfClusters)+" groups
      takes: ");
1138     tr.timeList.push_back(to_string(timeTemp)+" s");
1139     /* task completed, would delete memory contents */
1140     dNodeVec.clear();
1141     nodeMap.clear();
1142     /* use alpha function to sort the group by its size */
1143     std::sort(nodeVec.begin(), nodeVec.end(), [](const AHC_node& e1, const
     AHC_node& e2)
1144     {return e1.element.size()<e2.element.size()||(e1.element.size()==e2.element.size()&&e1.
     index<e2.index);});
1145 }
```

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.

```
850 {
851     std::unordered_map<int, AHC_node> nodeMap;
852     std::vector<DistNode> dNodeVec;
853     std::vector<AHC_node> nodeVec;
854     const int& Row = cArray.rows();
855     const int& Column = cArray.cols();
856 
857     /* compute distance matrix for reduced_space */
858     Eigen::MatrixXf reduced_dist_matrix = Eigen::MatrixXf::Zero(Row, Row);
859 #pragma omp parallel for schedule(static) num_threads(8)
860     for (int i = 0; i < Row; ++i)
861     {
862         for (int j = 0; j < Row; ++j)
863         {
864             /* don't wish to waste computation on diagonal element */
865             if(i==j)
866                 continue;
867             else
868                 reduced_dist_matrix(i,j) = (cArray.row(i)-cArray.row(j)).norm();
869         }
870     }
871     /* set the ditNode vector */
872     setValue(dNodeVec, cArray, reduced_dist_matrix);
873 
874     /* perform hirarchical clustering where within each step would merge two nodes */
875     hierarchicalMerging(nodeMap, dNodeVec, nodeVec, reduced_dist_matrix, cArray, Cluster
    , tr);
876 
877     vector<vector<int> > neighborVec(Cluster);
878 
879     // element size for all groups
880     vector<int> storage(Cluster);
881 
882     // geometric center
883     Eigen::MatrixXf centroid = Eigen::MatrixXf::Zero(Cluster,Column);
884 
885     std::vector<int> recorder(Row);
886     // set label information
887     setLabel(nodeVec, neighborVec, storage, centroid, cArray, recorder);
888 
889     nodeVec.clear();
890 
891     struct timeval start, end;
892     double delta;
893     std::multimap<int,int> groupMap;
894 
895     float entropy = 0.0;
896     float probability;
897 
898     for (int i = 0; i < Cluster; ++i)
899     {
900         groupMap.insert(std::pair<int,int>(storage[i],i));
901         if(storage[i]>0)
902         {
903             probability = float(storage[i])/float(Row);
904             entropy += probability*log2f(probability);
905         }
906     }
907 
908     int groupNo = 0;
909     int increasingOrder[Cluster];
910     for (multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
911     {
912         if(it->first>0)
913         {
914             increasingOrder[it->second] = (groupNo++);
915         }
916     }
917 
918     /* calculate the balanced entropy */
919     entropy = -entropy/log2f(groupNo);
920     Eigen::MatrixXf clusterCenter(Cluster, Column);
921 
922 #pragma omp parallel for schedule(static) num_threads(8)
923     for (int i = 0; i < Row; ++i)
```

```
924     {
925         group[i] = increasingOrder[recorder[i]];
926         totalNum[i] = storage[recorder[i]];
927     }
928 
929 #pragma omp parallel for schedule(static) num_threads(8)
930     for (int i = 0; i < Cluster; ++i)
931     {
932         clusterCenter.row(increasingOrder[i]) = centroid.row(i);
933     }
934 
935     float shortest, farDist, toCenter;
936     int shortestIndex = 0, fartestIndex = 0, tempIndex = 0;
937     std::vector<int> neighborTemp;
938 
939     for (int i = 0; i < Cluster; ++i)
940     {
941         if(storage[i]>0 && !neighborVec[i].empty())
942         {
943             neighborTemp = neighborVec[i];
944             shortest = FLT_MAX;
945             farDist = FLT_MIN;
946 
947             for (int j = 0; j < storage[i]; ++j)
948             {
949                 tempIndex = neighborTemp[j];
950                 toCenter = (clusterCenter.row(i)-cArray.row(tempIndex)).norm();
951 
952                 if(toCenter<shortest)
953                 {
954                     shortest = toCenter;
955                     shortestIndex = tempIndex;
956                 }
957                 if(toCenter>farDist)
958                 {
959                     farDist = toCenter;
960                     fartestIndex = tempIndex;
961                 }
962             }
963             closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
964             furthest.push_back(ExtractedLine(fartestIndex,increasingOrder[i]));
965         }
966     }
967     MatrixXf pcSing(PC_Number,Column);
968 
969 #pragma omp parallel for schedule(static) num_threads(8)
970     for (int i = 0; i < PC_Number; ++i)
971     {
972         pcSing.row(i) = SingVec.row(i);
973     }
974 
975     MatrixXf massPos = clusterCenter*pcSing;
976 
977     for (int i = 0; i < Cluster; ++i)
978     {
979         if(storage[i]>0)
980         {
981             massPos.row(i) += meanTrajectory.transpose();
982             std::vector<float> vecTemp;
983             for (int j = 0; j < Column; ++j)
984             {
985                 vecTemp.push_back(massPos(i,j));
986             }
987             massCenter.push_back(MeanLine(vecTemp,increasingOrder[i]));
988         }
989     }
990 
991     ValidityMeasurement vm;
992     vm.computeValue(cArray, group);
993 
994     tr.eventList.push_back("PCA Validity measure is: ");
995     stringstream fc_ss;
996     fc_ss << vm.f_c;
997     tr.timeList.push_back(fc_ss.str());
998 
999     /* Silhouette effect */
1000     gettimeofday(&start, NULL);
1001 
1002     sil.computeValue(cArray,group,groupNo,isPBF);
1003 
1004     gettimeofday(&end, NULL);
1005     delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
1006 
1007     tr.eventList.push_back("Clustering evaluation computing takes: ");
1008     tr.timeList.push_back(to_string(delta)+"s");
1009 
1010     /* write value of the silhouette class */
```

```
1011     IOHandler::writeReadme(entropy, sil, "");
1012
1013 }
```

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.

```
437 {
438
439     performFullK_MeansByClusters(data, Row, Column, massCenter,
    CLUSTER, group,
440                                  totalNum, closest, furthest, normOption, tr, sil);
441 }
```

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.

```
516 {
517     performFullK_MeansByClusters(data, Row, Column, massCenter, Cluster, group,

518                                  totalNum, closest, furthest, normOption, tr, sil);
519 }
```

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.

```
549 {
550     MetricPreparation object(Row, Column);
551     object.preprocessing(data, Row, Column, normOption);
552
553     MatrixXf clusterCenter;
554
555     switch(initializationOption)
556     {
557     case 1:
558         Initialization::generateRandomPos(clusterCenter, Column, data, Cluster);
559         break;
560
561     case 2:
562         Initialization::generateFromSamples(clusterCenter, Column, data, Cluster);
563         break;
564
565     case 3:
566         Initialization::generateFarSamples(clusterCenter, Column, data, Cluster,
567                                            normOption, object);
568         break;
569     }
570
571     float moving=1000, tempMoving,/* dist, tempDist, */before;
```

```
572     int *storage = new int[Cluster]; // used to store number inside each cluster
573     MatrixXf centerTemp;
574     int tag = 0;
575     std::vector< std::vector<int> > neighborVec(Cluster, std::vector<int>());
576 
577 /* perform K-means with different metrics */
578     std::cout << "K-means start!" << std::endl;
579     struct timeval start, end;
580     gettimeofday(&start, NULL);
581     std::vector<int> recorder(Row); //use to record which cluster the row belongs to
582 
583     do
584     {
585     /* reset storage number and weighted mean inside each cluster*/
586         before=moving;
587         memset(storage,0,sizeof(int)*Cluster);
588         centerTemp = MatrixXf::Zero(Cluster,Column);
589 
590     /* clear streamline indices for each cluster */
591     #pragma omp parallel for schedule(static) num_threads(8)
592         for (int i = 0; i < Cluster; ++i)
593         {
594             neighborVec[i].clear();
595         }
596 
597     #pragma omp parallel num_threads(8)
598         {
599         #pragma omp for nowait
600             for (int i = 0; i < Row; ++i)
601             {
602                 int clusTemp;
603                 float dist = FLT_MAX;
604                 float tempDist;
605                 for (int j = 0; j < Cluster; ++j)
606                 {
607                     tempDist = getDisimilarity(clusterCenter.row(j),data,i,normOption,object);
608                     if(tempDist<dist)
609                     {
610                         dist = tempDist;
611                         clusTemp = j;
612                     }
613                 }
614                 recorder[i] = clusTemp;
615 
616             #pragma omp critical
617                 {
618                     storage[clusTemp]++;
619                     neighborVec[clusTemp].push_back(i);
620                     centerTemp.row(clusTemp)+=data.row(i);
621                 }
622             }
623         }
624         moving = FLT_MIN;
625 
626     /* measure how much the current center moves from original center */
627     #pragma omp parallel for reduction(max:moving) num_threads(8)
628         for (int i = 0; i < Cluster; ++i)
629         {
630             if(storage[i]>0)
631             {
632                 centerTemp.row(i)/=storage[i];
633                 tempMoving = (centerTemp.row(i)-clusterCenter.row(i)).norm();
634                 clusterCenter.row(i) = centerTemp.row(i);
635                 if(moving<tempMoving)
636                     moving = tempMoving;
637             }
638         }
639         std::cout << "K-means iteration " << ++tag << " completed, and moving is " << moving << "!" <<
    std::endl;
640     }while(abs(moving-before)/before >= 1.0e-2 && tag < 20 && moving > 0.01);
641 
642     double delta;
643 
644     std::multimap<int,int> groupMap;
645 
646     float entropy = 0.0, probability;
647     int increasingOrder[Cluster];
648 
649     int nonZero = 0;
650     for (int i = 0; i < Cluster; ++i)
651     {
652         groupMap.insert(std::pair<int,int>(storage[i],i));
653         if(storage[i]>0)
654         {
655             probability=float(storage[i])/float(Row);
656             entropy+=probability*log2f(probability);
657             ++nonZero;
```

```
658         }
659     }
660     entropy = -entropy/log2f(nonZero);
661 
662     int groupNo = 0;
663     for (std::multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
664     {
665         if(it->first>0)
666         {
667             increasingOrder[it->second] = (groupNo++);
668         }
669     }
670     std::cout << "There are " << groupNo << " groups generated!" << std::endl;
671     /* finish tagging for each group */
672 
673     /* record labeling information */
674     // IOHandler::generateGroups(neighborVec);
675 
676     // set cluster group number and size number
677 #pragma omp parallel for schedule(static) num_threads(8)
678     for (int i = 0; i < Row; ++i)
679     {
680         group[i] = increasingOrder[recorder[i]];
681         totalNum[i] = storage[recorder[i]];
682     }
683 
684     float shortest, toCenter, farDist;
685     int shortestIndex = 0, tempIndex = 0, furthestIndex = 0;
686     std::vector<int> neighborTemp;
687 
688     /* choose cloest and furthest streamlines to centroid streamlines */
689     for (int i = 0; i < Cluster; ++i)
690     {
691         if(storage[i]>0)
692         {
693 
694             neighborTemp = neighborVec[i];
695             shortest = FLT_MAX;
696             farDist = FLT_MIN;
697 
698             for (int j = 0; j < storage[i]; ++j)
699             {
700                 // j-th internal streamlines
701                 tempIndex = neighborTemp[j];
702                 toCenter = getDisimilarity(clusterCenter.row(i),data,tempIndex,normOption,object);
703 
704                 /* update the closest index to centroid */
705                 if(toCenter<shortest)
706                 {
707                     shortest = toCenter;
708                     shortestIndex = tempIndex;
709                 }
710 
711                 /* update the farthest index to centroid */
712                 if(toCenter>farDist)
713                 {
714                     farDist = toCenter;
715                     furthestIndex = tempIndex;
716                 }
717             }
718             closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
719             furthest.push_back(ExtractedLine(furthestIndex,increasingOrder[i]));
720             //distFile << std::endl;
721         }
722     }
723     //distFile.close();
724 
725     std::vector<float> closeSubset;
726     /* based on known cluster centroid, save them as vector for output */
727     for (int i = 0; i < Cluster; ++i)
728     {
729         if(storage[i]>0)
730         {
731             for (int j = 0; j < Column; ++j)
732             {
733                 closeSubset.push_back(clusterCenter(i,j));
734             }
735             massCenter.push_back(MeanLine(closeSubset,increasingOrder[i]));
736             closeSubset.clear();
737         }
738     }
739     delete[] storage;
740 
741     //groupNo record group numbers */
742 
743     if(groupNo<=1)
744         return;
```

```
745
746     /* if the dataset is not PBF, then should record distance matrix for Gamma matrix compution */
747     if(!isPBF)
748     {
749         deleteDistanceMatrix(data.rows());
750
751         std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
752         if(distFile.fail())
753         {
754             distFile.close();
755             getDistanceMatrix(data, normOption, object);
756             std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
757             for(int i=0;i<data.rows();++i)
758             {
759                 for(int j=0;j<data.rows();++j)
760                 {
761                     distFileOut << distanceMatrix[i][j] << " ";
762                 }
763                 distFileOut << std::endl;
764             }
765             distFileOut.close();
766         }
767         else
768         {
769             std::cout << "read distance matrix..." << std::endl;
770
771             distanceMatrix = new float*[data.rows()];
772         #pragma omp parallel for schedule(static) num_threads(8)
773             for (int i = 0; i < data.rows(); ++i)
774             {
775                 distanceMatrix[i] = new float[data.rows()];
776             }
777             int i=0, j;
778             string line;
779             stringstream ss;
780             while(getline(distFile, line))
781             {
782                 j=0;
783                 ss.str(line);
784                 while(ss>>line)
785                 {
786                     if(i==j)
787                         distanceMatrix[i][j]=0;
788                     else
789                         distanceMatrix[i][j] = std::atof(line.c_str());
790                     ++j;
791                 }
792                 ++i;
793                 ss.str("");
794                 ss.clear();
795             }
796             distFile.close();
797         }
798
799         std::cout << "Distance between 0 and 1 is " << distanceMatrix[0][1] << std::endl;
800     }
801
802     gettimeofday(&start, NULL);
803
804     sil.computeValue(normOption,data,Row,Column,group,object,groupNo,isPBF);
805
806     gettimeofday(&end, NULL);
807     delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
808
809     tr.eventList.push_back("For norm ");
810     tr.timeList.push_back(to_string(normOption)+"\n");
811
812     tr.eventList.push_back("Clustering evaluation computing takes: ");
813     tr.timeList.push_back(to_string(delta)+"s");
814
815     ValidityMeasurement vm;
816     vm.computeValue(normOption, data, group, object, isPBF);
817
818     tr.eventList.push_back("kmeans Validity measure is: ");
819     stringstream fc_ss;
820     fc_ss << vm.f_c;
821     tr.timeList.push_back(fc_ss.str());
822
823     /* write value of the silhouette class */
824     IOHandler::writeReadme(entropy, sil, "For norm "+to_string(normOption));
825
826 }
```

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.

```
191 {
192     MetricPreparation object(Row, Column);
193     object.preprocessing(data, Row, Column, 0);
194 /* perform K-means clustering */
195     MatrixXf clusterCenter;
196 
197     switch(initializationOption)
198     {
199     case 1:
200         Initialization::generateRandomPos(clusterCenter, PC_Number, cArray, Cluster);
201         break;
202 
203     case 2:
204         Initialization::generateFromSamples(clusterCenter, PC_Number, cArray, Cluster);
205         break;
206 
207     case 3:
208         Initialization::generateFarSamples(clusterCenter, PC_Number, cArray,
209                                            Cluster, 0, object);
210         break;
211     }
212 
213     float moving=1000, tempMoving, before;
214     int storage[Cluster];
215 
216     MatrixXf centerTemp;  //store provisional center coordinate
217 
218     int tag = 0;
219 
220     std::vector< std::vector<int> > neighborVec(Cluster, std::vector<int>());
221 
222     double PCA_KMeans_delta, KMeans_delta;
223     struct timeval start, end;
224 
225     gettimeofday(&start, NULL);
226 
227     std::vector<int> recorder(Row);
228     do
229     {
230         before = moving;
231         /* preset cluster number recorder */
232         memset(storage,0,sizeof(int)*Cluster);
233         centerTemp = MatrixXf::Zero(Cluster, PC_Number);
234 
235     #pragma omp parallel for schedule(static) num_threads(8)
236         for (int i = 0; i < Cluster; ++i)
237         {
238             neighborVec[i].clear();
239         }
240 
241     #pragma omp parallel num_threads(8)
242         {
243         #pragma omp for nowait
244             for (int i = 0; i < Row; ++i)
245             {
246                 float dist = FLT_MAX;
247                 float temp;
248                 int clusTemp;
249                 for (int j = 0; j < Cluster; ++j)
250                 {
251                     temp = (cArray.row(i)-clusterCenter.row(j)).norm();
252                     if(temp<dist)
253                     {
254                         dist = temp;
255                         clusTemp = j;
256                     }
257                 }
258 
259             #pragma omp critical
260                 {
261                     storage[clusTemp]++;
262                     neighborVec[clusTemp].push_back(i);
263                     recorder[i] = clusTemp;
264                     centerTemp.row(clusTemp)+=cArray.row(i);
```

```
265                 }
266             }
267         }
268 
269         moving = FLT_MIN;
270 
271     #pragma omp parallel for reduction(max:moving) num_threads(8)
272         for (int i = 0; i < Cluster; ++i)
273         {
274             if(storage[i]>0)
275             {
276                 centerTemp.row(i)/=storage[i];
277                 tempMoving = (centerTemp.row(i)-clusterCenter.row(i)).norm();
278                 clusterCenter.row(i) = centerTemp.row(i);
279                 if(moving<tempMoving)
280                     moving = tempMoving;
281             }
282         }
283         std::cout << "K-means iteration " << ++tag << " completed, and moving is " << moving << "!" <<
    std::endl;
284     }while(abs(moving-before)/before >= 1.0e-2 && tag < 20 && moving>0.01);
285 
286     gettimeofday(&end, NULL);
287 
288     float delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
289 
290     tr.eventList.push_back("k-means iteration for PC takes: ");
291     tr.timeList.push_back(to_string(delta)+"s");
292 
293     std::multimap<int,int> groupMap;
294 
295     float entropy = 0.0;
296     float probability;
297 
298 
299     for (int i = 0; i < Cluster; ++i)
300     {
301         groupMap.insert(std::pair<int,int>(storage[i],i));
302         if(storage[i]>0)
303         {
304             probability = float(storage[i])/float(Row);
305             entropy += probability*log2f(probability);
306         }
307     }
308 
309     int groupNo = 0;
310     int increasingOrder[Cluster];
311     for (multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
312     {
313         if(it->first>0)
314         {
315             increasingOrder[it->second] = (groupNo++);
316         }
317     }
318 
319     /* calculate the balanced entropy */
320     entropy = -entropy/log2f(groupNo);
321 
322 
323 #pragma omp parallel for schedule(static) num_threads(8)
324     for (int i = 0; i < Row; ++i)
325     {
326         group[i] = increasingOrder[recorder[i]];
327         totalNum[i] = storage[recorder[i]];
328     }
329 
330     float shortest, farDist, toCenter;
331     int shortestIndex = 0, fartestIndex = 0, tempIndex = 0;
332     std::vector<int> neighborTemp;
333 
334     for (int i = 0; i < Cluster; ++i)
335     {
336         if(storage[i]>0 && !neighborVec[i].empty())
337         {
338             neighborTemp = neighborVec[i];
339             shortest = FLT_MAX;
340             farDist = FLT_MIN;
341 
342             for (int j = 0; j < storage[i]; ++j)
343             {
344                 tempIndex = neighborTemp[j];
345                 toCenter = (clusterCenter.row(i)-cArray.row(tempIndex)).norm();
346 
347                 if(toCenter<shortest)
348                 {
349                     shortest = toCenter;
350                     shortestIndex = tempIndex;
```

```
351                 }
352                 if(toCenter>farDist)
353                 {
354                     farDist = toCenter;
355                     fartestIndex = tempIndex;
356                 }
357             }
358             closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
359             furthest.push_back(ExtractedLine(fartestIndex,increasingOrder[i]));
360         }
361     }
362     MatrixXf pcSing(PC_Number,Column);
363
364 #pragma omp parallel for schedule(static) num_threads(8)
365     for (int i = 0; i < PC_Number; ++i)
366     {
367         pcSing.row(i) = SingVec.row(i);
368     }
369
370     MatrixXf massPos = clusterCenter*pcSing;
371
372     for (int i = 0; i < Cluster; ++i)
373     {
374         if(storage[i]>0)
375         {
376             massPos.row(i) += meanTrajectory.transpose();
377             std::vector<float> vecTemp;
378             for (int j = 0; j < Column; ++j)
379             {
380                 vecTemp.push_back(massPos(i,j));
381             }
382             massCenter.push_back(MeanLine(vecTemp,increasingOrder[i]));
383         }
384     }
385
386     ValidityMeasurement vm;
387     vm.computeValue(cArray, group);
388
389     tr.eventList.push_back("PCA Validity measure is: ");
390     stringstream fc_ss;
391     fc_ss << vm.f_c;
392     tr.timeList.push_back(fc_ss.str());
393
394     /* Silhouette effect */
395     gettimeofday(&start, NULL);
396
397     sil.computeValue(cArray,group,groupNo,isPBF);
398
399     gettimeofday(&end, NULL);
400     delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
401
402     tr.eventList.push_back("Clustering evaluation computing takes: ");
403     tr.timeList.push_back(to_string(delta)+"s");
404
405     /* write value of the silhouette class */
406     IOHandler::writeReadme(entropy, sil, "");
407
408 }
```

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.

```
63 {
64     MatrixXf cArray, SingVec;
65     VectorXf meanTrajectory(Column);
66     int PC_Number;
67
68     performSVD(cArray, data, Row, Column, PC_Number, SingVec, meanTrajectory, tr);
69
70     if(post_processing==1)
71         performPC_KMeans(cArray, Row, Column, PC_Number, SingVec, meanTrajectory,
72                          massCenter, CLUSTER, group, totalNum, closest, furthest, data, tr, sil);
73     else if(post_processing==2)
74         perform_AHC(cArray, PC_Number, SingVec, meanTrajectory,
75                     massCenter, CLUSTER, group, totalNum, closest, furthest, data, tr, sil);
76 }
```

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.

```
473 {
474     MatrixXf cArray, SingVec;
475     VectorXf meanTrajectory(Column);
476     int PC_Number;
477 
478     performSVD(cArray, data, Row, Column, PC_Number, SingVec, meanTrajectory, tr);
479     if(post_processing==1)
480         performPC_KMeans(cArray, Row, Column, PC_Number, SingVec, meanTrajectory,
481                          massCenter, Cluster, group, totalNum, closest, furthest, data, tr, sil);
482     else if(post_processing==2)
483         perform_AHC(cArray, PC_Number, SingVec, meanTrajectory,
484                     massCenter, Cluster, group, totalNum, closest, furthest, data, tr, sil);
485 }
```

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.

```
102 {
103     Eigen::MatrixXf temp = data;
104 
105 #pragma omp parallel for schedule(static) num_threads(8)
106     for (int i = 0; i < Column; ++i)
107     {
108         meanTrajectory(i) = temp.transpose().row(i).mean();
109     }
110 #pragma omp parallel for schedule(static) num_threads(8)
111     for (int i = 0; i < Row; ++i)
112     {
113         temp.row(i) = temp.row(i)-meanTrajectory.transpose();
114     }
115 
116     struct timeval start, end;
117     gettimeofday(&start, NULL);
118     /* perform SVD decomposition for temp */
119     JacobiSVD<MatrixXf> svd(temp, ComputeThinU | ComputeThinV);
120     //const VectorXf& singValue = svd.singularValues();
121     SingVec = svd.matrixV();
122     gettimeofday(&end, NULL);
123     const double& delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
124 
125     tr.eventList.push_back("SVD takes: ");
126     tr.timeList.push_back(to_string(delta)+"s");
127 
128     /* compute new attribute space based on principal component */
129     MatrixXf coefficient = temp*SingVec;
130     /*  decide first r dorminant PCs with a threshold */
131     const float& varianceSummation = coefficient.squaredNorm();
132     float tempSum = 0.0;
133     const float& threshold = TOR_1*varianceSummation;
134 
135     for (int i = 0; i < Column; ++i)
136     {
137         tempSum+=(coefficient.transpose().row(i)).squaredNorm();
138         if(tempSum>threshold)
139         {
140             PC_Number = i;
141             break;
142         }
143     }
144 
145     cArray = MatrixXf(Row, PC_Number);
146 #pragma omp parallel for schedule(static) num_threads(8)
147     for (int i = 0; i < PC_Number; ++i)
148     {
```

```
149         cArray.transpose().row(i) = coefficient.transpose().row(i);
150     }
151
152     std::cout << "SVD completed!" << std::endl;
153
154     SingVec.transposeInPlace();
155 }
```

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.

```
1219 {
1220 // group tag by increasing order
1221     int groupID = 0;
1222
1223     // element list for each group
1224     vector<int> eachContainment;
1225
1226     // find group id and neighboring vec
1227     for(auto iter = nodeVec.begin(); iter!=nodeVec.end();++iter)
1228     {
1229         eachContainment = (*iter).element;
1230         neighborVec[groupID] = eachContainment;
1231     #pragma omp parallel num_threads(8)
1232         {
1233         #pragma omp for nowait
1234             for(int i=0;i<eachContainment.size();++i)
1235             {
1236                 recorder[eachContainment[i]] = groupID;
1237             #pragma omp critical
1238                 centroid.row(groupID) += cArray.row(eachContainment[i]);
1239             }
1240         }
1241         storage[groupID] = (*iter).element.size();
1242         centroid.row(groupID)/=eachContainment.size();
1243         ++groupID;
1244         eachContainment.clear();
1245     }
1246 }
```

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.

```
1189 {
1190     const int& Row = reduced_data.rows();
1191     dNodeVec = std::vector<DistNode>(Row*(Row-1)/2);
1192     int tag = 0;
1193     for(int i=0;i<Row-1;++i)
1194     {
1195         for(int j=i+1;j<Row;++j)
1196         {
1197             dNodeVec[tag].first = i;
1198             dNodeVec[tag].second = j;
1199             dNodeVec[tag].distance = reduced_dist_matrix(i, j);
1200             ++tag;
1201         }
1202     }
1203     assert(tag==dNodeVec.size());
1204 }
```

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

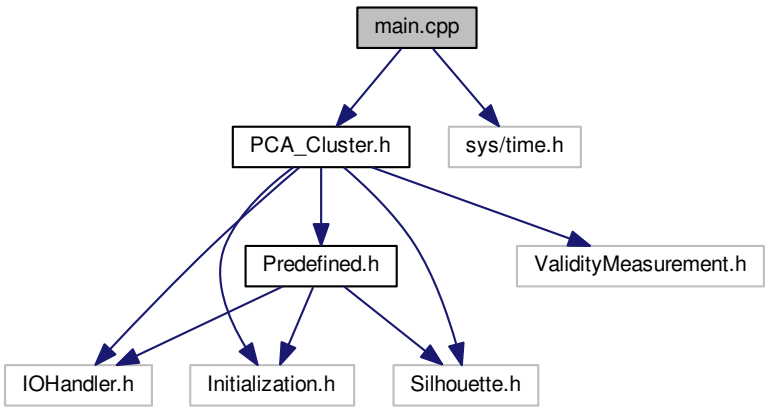
Chapter 5

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.

```
107 {
108     while(number!=3)
109     {
110         std::cout << "Input argument should have 3!" << endl
111                   << "./cluster inputFile_name(in dataset folder) "
112                   << "data_dimension(3)" << endl;
113         exit(1);
114     }
115     const string& strName = string("../dataset/")+string(argv[1]);
116     //const string& strName = "../dataset/pbfDataset";
117     const int& dimension = atoi(argv[2]);
118     //const string& pbfPath = "/media/lieyu/Seagate Backup Plus
      Drive/PBF_2013Macklin/pbf_velocitySeparate/source_data/Frame ";
119
120     //std::cout << strName << std::endl;
121     //fullName = "../dataset/streamlines_cylinder_9216_full.vtk";
122     //const string& strName = "../dataset/streamlines_cylinder_9216";
123     //const string& strName = "../dataset/pbf_data";
124
125     std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
126     int PBFjudgement;
127     std::cin >> PBFjudgement;
128     assert(PBFjudgement==1||PBFjudgement==0);
129     isPBF = (PBFjudgement==1);
130
131     /* input for judge whether it is a pathline data set so that MCP can be called */
132     bool isPathlines;
133     std::cout << "It is pathlines? 1.Yes, 0.No" << std::endl;
134     std::cin >> PBFjudgement;
135     assert(PBFjudgement==1||PBFjudgement==0);
136     isPathlines = (PBFjudgement==1);
137
138     /* set how many clusters and max vertex count of the data set */
139     int cluster, vertexCount;
140
141     /* choose k-means initialization method, 2 is often adopted providing better visualization effect */
142     std::cout << "Please choose initialization option for seeds:" << std::endl
143               << "1.chose random positions, 2.Chose from samples, 3.k-means++ sampling" << endl;
144     std::cin >> initializationOption;
145     assert(initializationOption==1 || initializationOption==2
146            || initializationOption==3);
147
148     int samplingMethod;
149     if(isPathlines)
150         samplingMethod = 1;
151
152     else
153     {
154         /* select sampling strategy, and 2 is often for geometric clustering */
155         std::cout << "Please choose sampling strategy: " << std::endl
156                   << "1.directly filling, 2.uniformly sampling" << std::endl;
157         std::cin >> samplingMethod;
158     }
159
160     assert(samplingMethod==1 || samplingMethod==2);
161
162     /* whether number of clusters is read from user input or from ../dataset/cluster_number */
163     std::cout << "Please choose cluster number method, 0.user input, 1.read clustering: " << std::endl;
164     int clusterInput;
165     std::cin >> clusterInput;
166     assert(clusterInput==0 || clusterInput==1);
167     readCluster = (clusterInput==1);
168
```

```
169     std::unordered_map<int,int> clusterMap;
170     if(readCluster)
171     {
172         IOHandler::readClusteringNumber(clusterMap, "cluster_number");
173     }
174 
175 
176     TimeRecorder tr;
177 
178     Silhouette sil;
179 
180     struct timeval start, end;
181     double timeTemp;
182     int maxElements;
183 
184     gettimeofday(&start, NULL);
185     std::vector< std::vector<float> > dataVec;
186     IOHandler::readFile(strName, dataVec, vertexCount, dimension, maxElements);
187     gettimeofday(&end, NULL);
188     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
189                 + end.tv_usec - start.tv_usec) / 1.e6;
190     tr.eventList.push_back("I-O file reader takes: ");
191     tr.timeList.push_back(to_string(timeTemp)+"s");
192 
193     if(!readCluster)
194     {
195         std::cout << "Please input a cluster number (>=2) among [2, " << dataVec.size() << "]: " <<
    std::endl;
196         std::cin >> cluster;
197     }
198     else
199     {
200         cluster = clusterMap[0];
201     }
202 
203     stringstream ss;
204     ss << strName << "_differentNorm_full.vtk";
205     const string& fullName = ss.str();
206     IOHandler::printVTK(ss.str(), dataVec, vertexCount, dimension);
207     ss.str("");
208 
209     Eigen::MatrixXf data;
210 
211     /* PCA computation is always using brute-force filling arrays by last point */
212     IOHandler::expandArray(data, dataVec, dimension, maxElements);
213 
214     std::cout << "PCA-based clustering starts..." << std::endl;
215     ss << strName << "_PCAClustering";
216     gettimeofday(&start, NULL);
217     performPCA_Cluster(ss.str(), dataVec, cluster, dimension, fullName, maxElements, data
    , tr, sil);
218     std::cout << "Max element is " << maxElements << std::endl;
219     ss.str("");
220     ss.clear();
221     gettimeofday(&end, NULL);
222     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
223                 + end.tv_usec - start.tv_usec) / 1.e6;
224 
225     tr.eventList.push_back("PCA+KMeans takes: ");
226     tr.timeList.push_back(std::to_string(timeTemp)+"s");
227 
228     IOHandler::writeReadme(tr.eventList, tr.timeList, cluster);
229 
230     tr.eventList.clear();
231     tr.timeList.clear();
232 
233 
234     sil.reset();
235 
236     /*  0: Euclidean Norm
237         1: Fraction Distance Metric
238         2: piece-wise angle average
239         3: Bhattacharyya metric for rotation
240         4: average rotation
241         5: signed-angle intersection
242         6: normal-direction multivariate distribution
243         7: Bhattacharyya metric with angle to a fixed direction
244         8: Piece-wise angle average \times standard deviation
245         9: normal-direction multivariate un-normalized distribution
246         10: x*y/|x||y| borrowed from machine learning
247         11: cosine similarity
248         12: Mean-of-closest point distance (MCP)
249         13: Hausdorff distance min_max(x_i,y_i)
250         14: Signature-based measure taken from http://ieeexplore.ieee.org/stamp/
    stamp.jsp?tp=&arnumber=6231627
251         15: Procrustes distance taken from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
252         16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
```

```
253         17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
    S0097849318300128
254             for pathlines only
255     */
256     if(samplingMethod==2)
257         IOHandler::sampleArray(data, dataVec, dimension, maxElements);
258 
259     for(int i = 0;i<=17;i++)
260     {
261         if(isPathlines)
262         {
263             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15 && i!=17)
264                 continue;
265         }
266 
267         else
268         {
269             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
270                 continue;
271         }
272 
273         if(readCluster)
274             cluster = clusterMap[i];
275         else
276         {
277             std::cout << "Please input a cluster number (>=2) for norm " << i << " in [2, "
278                     << dataVec.size() << "]: " << std::endl;
279             std::cin >> cluster;
280         }
281 
282         std::cout << "Kmeans on norm " << i << " starts..." << std::endl;
283         gettimeofday(&start, NULL);
284         ss << strName << "_KMeans";
285         performK_Means(ss.str(), dataVec, cluster, dimension, fullName, maxElements, data,i,
    tr, sil);
286 
287         ss.str("");
288         gettimeofday(&end, NULL);
289         timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
290                     + end.tv_usec - start.tv_usec) / 1.e6;
291         tr.eventList.push_back("K-means on norm "+to_string(i)+" takes: ");
292         tr.timeList.push_back(to_string(timeTemp)+"s");
293 
294         IOHandler::writeReadme(tr.eventList, tr.timeList, cluster);
295 
296         tr.eventList.clear();
297         tr.timeList.clear();
298         sil.reset();
299     }
300 }
```

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

Definition at line 92 of file main.cpp.

```
93 {
94     featureExtraction(argc, argv);
95     return 0;
96 }
```

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.

```
415 {
416     std::vector<MeanLine> centerMass;
417     std::vector<ExtractedLine> closest;
418     std::vector<ExtractedLine> furthest;
419     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);
```

```
338
339     std::vector<std::vector<float> > closestStreamline;
340     std::vector<std::vector<float> > furthestStreamline;
341     std::vector<int> closestCluster, furthestCluster, meanCluster;
342     int closestPoint, furthestPoint;
343
344     IOHandler::assignVec(closestStreamline, closestCluster, closest, closestPoint, dataVec);
345
346     IOHandler::assignVec(furthestStreamline, furthestCluster, furthest,
347                          furthestPoint, dataVec);
348
349 /* get the average rotation of the extraction */
350     std::vector<float> closestRotation, furthestRotation;
351     const float& closestAverage = getRotation(closestStreamline, closestRotation);
352     const float& furthestAverage = getRotation(furthestStreamline, furthestRotation);
353
354     tr.eventList.push_back("Average rotation of closest for PCA clustering is: ");
355     tr.timeList.push_back(to_string(closestAverage));
356
357     tr.eventList.push_back("Average rotation of furthest for PCA clustering is: ");
358     tr.timeList.push_back(to_string(furthestAverage));
359 /* finish the rotation computation */
360
361     IOHandler::assignVec(meanCluster, centerMass);
362
363     IOHandler::printVTK(fileName+string("_PCA_closest.vtk"), closestStreamline,
364                         closestPoint/dimension, dimension, closestCluster,
365                         sil.sCluster);
366
367     IOHandler::printVTK(fileName+string("_PCA_furthest.vtk"), furthestStreamline,
368                         furthestPoint/dimension, dimension, furthestCluster,
369                         sil.sCluster);
370
371     IOHandler::printVTK(fileName+string("_PCA_mean.vtk"), centerMass,
372                         centerMass.size()*centerMass[0].minCenter.size()/dimension,
373                         dimension, sil.sCluster);
374
375     std::cout << "Finish printing vtk for pca-clustering result!" << std::endl;
376
377     if(post_processing==1)
378         IOHandler::printToFull(dataVec, group, totalNum, string("PCA_KMeans"), fullName, dimension);
379     else if(post_processing==2)
380         IOHandler::printToFull(dataVec, group, totalNum, string("PCA_AHC"), fullName, dimension);
381
382     //IOHandler::writeReadme(closest, furthest);
383
384     IOHandler::printToFull(dataVec, sil.sData, "PCA_SValueLine", fullName, 3);
385
386     IOHandler::printToFull(dataVec, group, sil.sCluster, "PCA_SValueCluster",
387                            fullName, 3);
388 }
```

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.

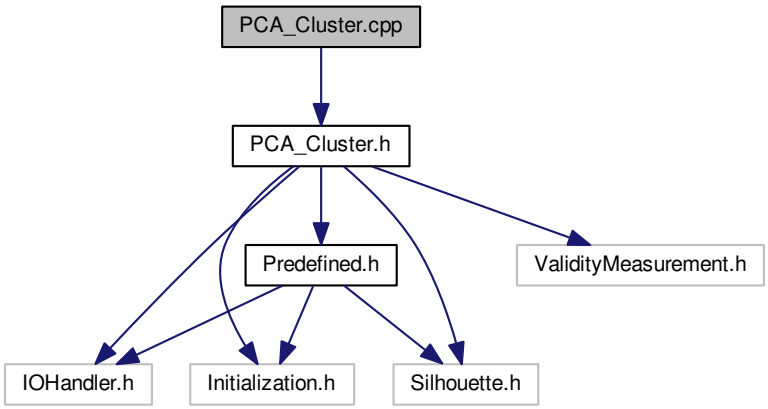
5.1.2.4 bool readCluster

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.

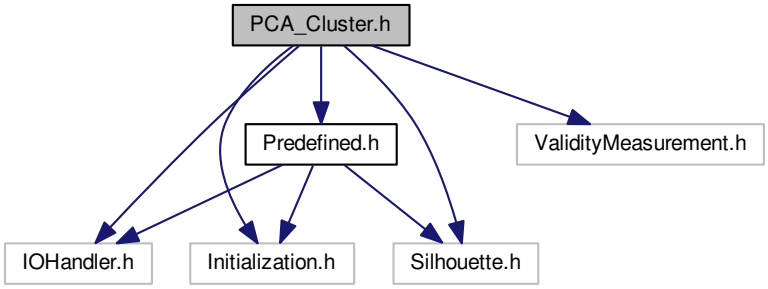
5.2.1.5 const float& TOR_1 = 0.999

Definition at line 13 of file PCA_Cluster.cpp.

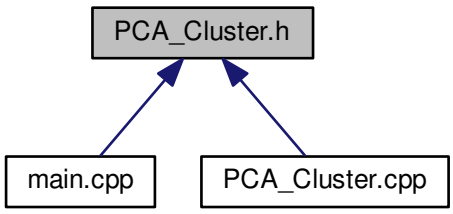
5.3 PCA_Cluster.h File Reference

```
#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:



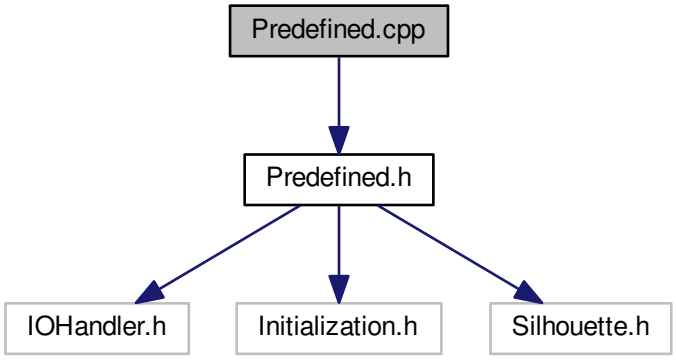
Classes

- 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*)

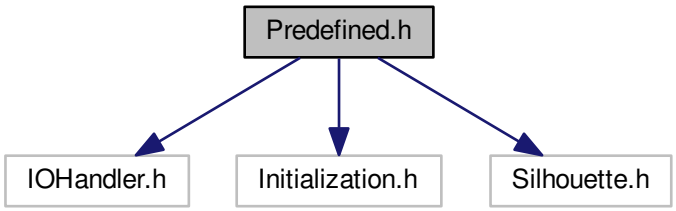
Definition at line 19 of file Predefined.cpp.

```
20 {
21     std::size_t size = original.size();
22     assert(size>2);
23     vector<T> result(size-2);
24     int tag = 0;
25     for(int i=0;i<size;++i)
26     {
27         //meet with target elements, not copied
28         if(original[i]==first || original[i]==second)
29             continue;
30         result[tag++]=original[i];
31     }
32     assert(tag==size-2);
33     original = result;
34 }
```

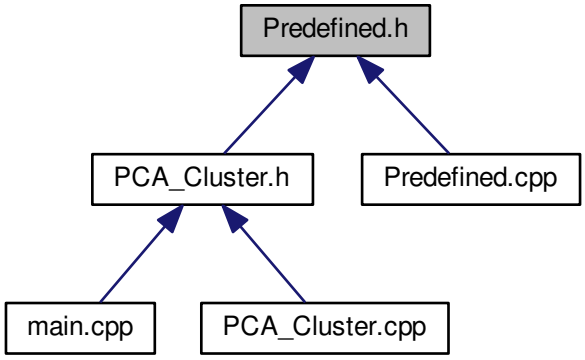
5.5 Predefined.h File Reference

```
#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.

```
20 {
21     std::size_t size = original.size();
22     assert(size>2);
23     vector<T> result(size-2);
24     int tag = 0;
25     for(int i=0;i<size;++i)
26     {
27         //meet with target elements, not copied
28         if(original[i]==first || original[i]==second)
29             continue;
30         result[tag++]=original[i];
31     }
32     assert(tag==size-2);
33     original = result;
34 }
```

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