K-Medoids Clustering

The C++ implmentation for K-Medoids clustering

Generated by Doxygen 1.8.11

Contents

Chapter 1

k-medoids

It includes the iterative or median-based medoid refinement similar to what k-medoids defined. To confirm to the regulations of other clustering techniques and cluster representatives, we choose the **iterative medoid refinement**.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 KMedoids Class Reference

`#include <KMedoids.h>`

Public Member Functions

- KMedoids (const Parameter &pm, const Eigen::MatrixXf &data, const int &numOfClusters)
- ~KMedoids ()
- void getMedoids (FeatureLine &fline, const int &normOption, Silhouette &sil, TimeRecorder &tr) const

Public Attributes

- int numOfClusters

Private Member Functions

- void getInitCenter (MatrixXf &initialCenter, const MetricPreparation &object, const int &normOption) const
- void computeMedoids (MatrixXf &centerTemp, const vector< vector< int > > &neighborVec, const int &normOption, const MetricPreparation &object) const

Private Attributes

- int initialStates
- bool isSample
- Eigen::MatrixXf data

4.1.1 Detailed Description

Definition at line 47 of file KMedoids.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 KMedoids::KMedoids (const Parameter & *pm,* const Eigen::MatrixXf & *data,* const int & *numOfClusters*)

Definition at line 17 of file KMedoids.cpp.

```
20          :initialStates(pm.initialization), isSample(pm.
      isSample),
21           data(data), numOfClusters(numOfClusters)
22 {
23
24 }
```

4.1.2.2 KMedoids::~KMedoids ( )

Definition at line 30 of file KMedoids.cpp.

```
31 {
32
33 }
```

4.1.3 Member Function Documentation

4.1.3.1 void KMedoids::computeMedoids (MatrixXf & *centerTemp,* const vector< vector< int > > & *neighborVec,* const int & *normOption,* const MetricPreparation & *object*) const [private]

Definition at line 360 of file KMedoids.cpp.

```
364 {
365     centerTemp = MatrixXf(numOfClusters,data.cols());
366     if(isSample)//centroid is from samples with minimal L1 summation
367                 //use Voronoi iteration https://en.wikipedia.org/wiki/K-medoids
368     {
369     #pragma omp parallel for schedule(static) num_threads(8)
370         for(int i=0;i<neighborVec.size();++i)
371         {
372             const vector<int>& clusMember = neighborVec[i];
373             const int& clusSize = clusMember.size();
374             MatrixXf mutualDist = MatrixXf::Zero(clusSize, clusSize);
375             /*mutualDist to store mutual distance among lines of each cluster */
376             for(int j=0;j<clusSize;++j)
377             {
378                 for(int k=j+1;k<clusSize;++k)
379                 {
380                     mutualDist(j,k) = getDisimilarity(data,clusMember[j],
381                         clusMember[k],normOption,object);
382                     mutualDist(k,j) = mutualDist(j,k);
383                 }
384             }
385
386             float minL1_norm = FLT_MAX, rowSummation;
387             int index = -1;
388             for(int j=0;j<clusSize;++j)
389             {
390                 rowSummation = mutualDist.row(j).sum();
391                 if(rowSummation<minL1_norm)
392                 {
393                     minL1_norm = rowSummation;
394                     index = j;
395                 }
396             }
397             centerTemp.row(i)=data.row(clusMember[index]);
398         }
399     }
400
401     else//use Weiszfeld's algorithm to get geometric median
```

```
402         //reference at https://en.wikipedia.org/wiki/Geometric_median
403     {
404         MatrixXf originCenter = centerTemp;
405     #pragma omp parallel for schedule(static) num_threads(8)
406         for(int i=0;i<numOfClusters;++i)
407         {
408             const vector<int>& clusMember = neighborVec[i];
409             const int& clusSize = clusMember.size();
410             float distToCenter, distInverse, percentage = 1.0;
411             int tag = 0;
412             while(tag<=10&&percentage>=0.02)
413             {
414                 VectorXf numerator = VectorXf::Zero(data.cols());
415                 VectorXf previous = centerTemp.row(i);
416                 float denominator = 0;
417                 for(int j=0;j<clusSize;++j)
418                 {
419                     distToCenter = getDisimilarity(centerTemp.row(i),
420                             data,clusMember[j],normOption,object);
421                     distInverse = (distToCenter>1.0e-8)?1.0/distToCenter:1.0e8;
422                     numerator += data.row(clusMember[j])*distInverse;
423                     denominator += distInverse;
424                 }
425                 centerTemp.row(i) = numerator/denominator;
426                 percentage = (centerTemp.row(i)-previous).norm()/previous.norm();
427                 tag++;
428             }
429         }
430     }
431 }
```

4.1.3.2 void KMedoids::getInitCenter (MatrixXf & *initialCenter,* const MetricPreparation & *object,* const int & *normOption*) const [private]

Definition at line 329 of file KMedoids.cpp.

```
332 {
333     switch(initialStates)
334     {
335     case 1:
336         Initialization::generateRandomPos(initialCenter, data.cols(), data,
    numOfClusters);
337         break;
338
339     default:
340     case 2:
341         Initialization::generateFromSamples(initialCenter, data.cols(), data,
    numOfClusters);
342         break;
343
344     case 3:
345         Initialization::generateFarSamples(initialCenter, data.cols(), data,
    numOfClusters, normOption, object);
346         break;
347     }
348     std::cout << "Initialization completed!" << std::endl;
349 }
```

4.1.3.3 void KMedoids::getMedoids (FeatureLine & *fline,* const int & *normOption,* Silhouette & *sil,* TimeRecorder & *tr*) const

Definition at line 44 of file KMedoids.cpp.

```
48 {
49     MetricPreparation object(data.rows(), data.cols());
50     object.preprocessing(data, data.rows(), data.cols(), normOption);
51
52     MatrixXf clusterCenter;
53     getInitCenter(clusterCenter, object, normOption);
54
55     float moving=1000, tempMoving,/* dist, tempDist, */before;
```

```
56      int *storage = new int[numOfClusters]; // used to store number inside each cluster
57      MatrixXf centerTemp;
58      int tag = 0;
59      std::vector< std::vector<int> > neighborVec(numOfClusters,
60                                                  std::vector<int>());
61
62  /* perform K-means with different metrics */
63      std::cout << "K-medoids start!" << std::endl;
64      const int& Row = data.rows();
65      const int& Column = data.cols();
66      struct timeval start, end;
67      gettimeofday(&start, NULL);
68      std::vector<int> recorder(Row); //use to record which cluster the row belongs to
69
70      do
71      {
72      /* reset storage number and weighted mean inside each cluster*/
73          before=moving;
74          memset(storage,0,sizeof(int)*numOfClusters);
75          centerTemp = clusterCenter;
76
77      /* clear streamline indices for each cluster */
78      #pragma omp parallel for schedule(static) num_threads(8)
79          for (int i = 0; i < numOfClusters; ++i)
80          {
81              neighborVec[i].clear();
82          }
83
84      #pragma omp parallel num_threads(8)
85          {
86          #pragma omp for nowait
87              for (int i = 0; i < Row; ++i)
88              {
89                  int clusTemp;
90                  float dist = FLT_MAX;
91                  float tempDist;
92                  for (int j = 0; j < numOfClusters; ++j)
93                  {
94                      tempDist = getDisimilarity(clusterCenter.row(j),
95                                 data,i,normOption,object);
96                      if(tempDist<dist)
97                      {
98                          dist = tempDist;
99                          clusTemp = j;
100                     }
101                 }
102                 recorder[i] = clusTemp;
103
104             #pragma omp critical
105                 {
106                     storage[clusTemp]++;
107                     neighborVec[clusTemp].push_back(i);
108                 }
109             }
110         }
111
112         computeMedoids(centerTemp, neighborVec, normOption, object);
113
114         moving = FLT_MIN;
115
116     /* measure how much the current center moves from original center */
117     #pragma omp parallel for reduction(max:moving) num_threads(8)
118         for (int i = 0; i < numOfClusters; ++i)
119         {
120             if(storage[i]>0)
121             {
122                 tempMoving = (centerTemp.row(i)-clusterCenter.row(i)).norm();
123                 clusterCenter.row(i) = centerTemp.row(i);
124                 if(moving<tempMoving)
125                     moving = tempMoving;
126             }
127         }
128         std::cout << "K-means iteration " << ++tag << " completed, and moving is " << moving
129                   << "!" << std::endl;
130     }while(abs(moving-before)/before >= 1.0e-2 && tag < 20/* && moving > 5.0*/);
131
132     double delta;
133
134     tr.eventList.push_back("For norm ");
135     tr.timeList.push_back(to_string(normOption)+"\n");
136
137     std::multimap<int,int> groupMap;
138     float entropy = 0.0;
139     float probability;
140     vector<int> increasingOrder(numOfClusters);
141     for (int i = 0; i < numOfClusters; ++i)
142     {
```

```
143         groupMap.insert(std::pair<int,int>(storage[i],i));
144         if(storage[i]>0)
145         {
146             probability = float(storage[i])/float(Row);
147             entropy += probability*log2f(probability);
148         }
149     }
150 
151     int groupNo = 0;
152     for (std::multimap<int,int>::iterator it = groupMap.begin(); it != groupMap.end(); ++it)
153     {
154         if(it->first>0)
155         {
156             increasingOrder[it->second] = (groupNo++);
157         }
158     }
159 
160     entropy = -entropy/log2f(groupNo);
161     /* finish tagging for each group */
162 
163     /* record labeling information */
164     // IOHandler::generateGroups(neighborVec);
165 
166 
167     // set cluster group number and size number
168 #pragma omp parallel for schedule(static) num_threads(8)
169     for (int i = 0; i < Row; ++i)
170     {
171         fline.group[i] = increasingOrder[recorder[i]];
172         fline.totalNum[i] = storage[recorder[i]];
173     }
174 
175     float shortest, toCenter, farDist;
176     int shortestIndex = 0, tempIndex = 0, furthestIndex = 0;
177     std::vector<int> neighborTemp;
178 
179     /* choose cloest and furthest streamlines to centroid streamlines */
180     for (int i = 0; i < numOfClusters; ++i)
181     {
182         if(storage[i]>0)
183         {
184 
185             neighborTemp = neighborVec[i];
186             shortest = FLT_MAX;
187             farDist = FLT_MIN;
188 
189             for (int j = 0; j < storage[i]; ++j)
190             {
191                 // j-th internal streamlines
192                 tempIndex = neighborTemp[j];
193                 toCenter = getDisimilarity(clusterCenter.row(i),data,
194                                            tempIndex,normOption,object);
195                 if(!isSample)
196                 {
197                     if(toCenter<shortest)
198                     {
199                         shortest = toCenter;
200                         shortestIndex = tempIndex;
201                     }
202                 }
203                 /* update the farthest index to centroid */
204                 if(toCenter>farDist)
205                 {
206                     farDist = toCenter;
207                     furthestIndex = tempIndex;
208                 }
209             }
210             if(!isSample)
211                 fline.closest.push_back(ExtractedLine(shortestIndex,increasingOrder[i]));
212             fline.furthest.push_back(ExtractedLine(furthestIndex,increasingOrder[i]));
213         }
214     }
215 
216     std::vector<float> closeSubset;
217     /* based on known cluster centroid, save them as vector for output */
218     for (int i = 0; i < numOfClusters; ++i)
219     {
220         if(storage[i]>0)
221         {
222             for (int j = 0; j < Column; ++j)
223             {
224                 closeSubset.push_back(clusterCenter(i,j));
225             }
226             fline.centerMass.push_back(MeanLine(closeSubset,increasingOrder[i]));
227             closeSubset.clear();
228         }
229     }
```

```
230     delete[] storage;
231     std::cout << "Has taken closest and furthest out!" << std::endl;
232 
233 
234 /* Silhouette computation started */
235 
236     std::cout << "The finalized cluster size is: " << groupNo << std::endl;
237     if(groupNo<=1)
238         return;
239 
240     /* if the dataset is not PBF, then should record distance matrix for Gamma matrix compution */
241     if(!isPBF)
242     {
243         deleteDistanceMatrix(data.rows());
244 
245         std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
246         if(distFile.fail())
247         {
248             distFile.close();
249             getDistanceMatrix(data, normOption, object);
250             std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
251             for(int i=0;i<data.rows();++i)
252             {
253                 for(int j=0;j<data.rows();++j)
254                 {
255                     distFileOut << distanceMatrix[i][j] << " ";
256                 }
257                 distFileOut << std::endl;
258             }
259             distFileOut.close();
260         }
261         else
262         {
263             std::cout << "read distance matrix..." << std::endl;
264 
265             distanceMatrix = new float*[data.rows()];
266         #pragma omp parallel for schedule(static) num_threads(8)
267             for (int i = 0; i < data.rows(); ++i)
268             {
269                 distanceMatrix[i] = new float[data.rows()];
270             }
271             int i=0, j;
272             string line;
273             stringstream ss;
274             while(getline(distFile, line))
275             {
276                 j=0;
277                 ss.str(line);
278                 while(ss>>line)
279                 {
280                     if(i==j)
281                         distanceMatrix[i][j]=0;
282                     else
283                         distanceMatrix[i][j] = std::atof(line.c_str());
284                     ++j;
285                 }
286                 ++i;
287                 ss.str("");
288                 ss.clear();
289             }
290             distFile.close();
291         }
292     }
293 
294     tr.eventList.push_back("Final cluster number is : ");
295     tr.timeList.push_back(to_string(groupNo));
296 
297     ValidityMeasurement vm;
298     vm.computeValue(normOption, data, fline.group, object, isPBF);
299 
300     tr.eventList.push_back("Kmedoids Validity measure is: ");
301     stringstream fc_ss;
302     fc_ss << vm.f_c;
303     tr.timeList.push_back(fc_ss.str());
304 
305     //groupNo record group numbers */
306     gettimeofday(&start, NULL);
307 
308     sil.computeValue(normOption,data,Row,Column,fline.group,object,groupNo,
    isPBF);
309     std::cout << "Silhouette computation completed!" << std::endl;
310 
311     gettimeofday(&end, NULL);
312     delta = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
313 
314     tr.eventList.push_back("Evaluation analysis would take: ");
315     tr.timeList.push_back(to_string(delta)+"s");
```

```
316
317     /* write value of the silhouette class */
318     IOHandler::writeReadme(entropy, sil, "For norm "+to_string(normOption));
319 }
```

4.1.4 Member Data Documentation

4.1.4.1 Eigen::MatrixXf KMedoids::data [private]

Definition at line 107 of file KMedoids.h.

4.1.4.2 int KMedoids::initialStates [private]

Definition at line 95 of file KMedoids.h.

4.1.4.3 bool KMedoids::isSample [private]

Definition at line 102 of file KMedoids.h.

4.1.4.4 int KMedoids::numOfClusters

Definition at line 85 of file KMedoids.h.

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

- KMedoids.h
- KMedoids.cpp

4.2 Parameter Struct Reference

`#include <KMedoids.h>`

Public Member Functions

- Parameter (const int &initialization, const bool &isSample)
- Parameter ()
- ~Parameter ()

Public Attributes

- int initialization
- bool isSample

4.2.1 Detailed Description

Definition at line 20 of file KMedoids.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Parameter::Parameter (const int & *initialization,* const bool & *isSample*) [inline]

Definition at line 24 of file KMedoids.h.

```
25     : initialization(initialization), isSample(
       isSample)
26     {}
```

4.2.2.2 Parameter::Parameter ( ) [inline]

Definition at line 27 of file KMedoids.h.

```
28     {}
```

4.2.2.3 Parameter::~Parameter ( ) [inline]

Definition at line 29 of file KMedoids.h.

```
30     {}
```

4.2.3 Member Data Documentation

4.2.3.1 int Parameter::initialization

Definition at line 22 of file KMedoids.h.

4.2.3.2 bool Parameter::isSample

Definition at line 23 of file KMedoids.h.

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

- KMedoids.h

4.3 TimeRecorder Struct Reference

```
#include <KMedoids.h>
```

Public Attributes

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

4.3.1 Detailed Description

Definition at line 37 of file KMedoids.h.

4.3.2 Member Data Documentation

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

Definition at line 39 of file KMedoids.h.

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

Definition at line 40 of file KMedoids.h.

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

- KMedoids.h

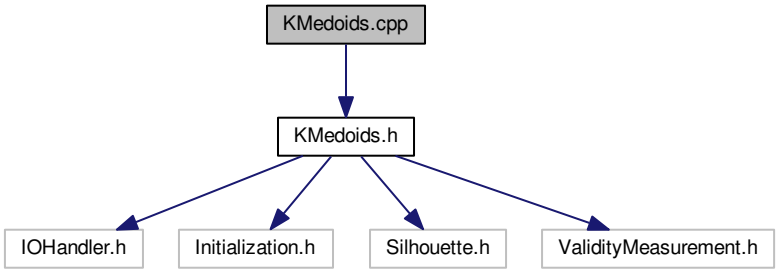
Chapter 5

File Documentation

5.1 KMedoids.cpp File Reference

```
#include "KMedoids.h"
```

Include dependency graph for KMedoids.cpp:



Variables

- bool isPBF

5.1.1 Variable Documentation

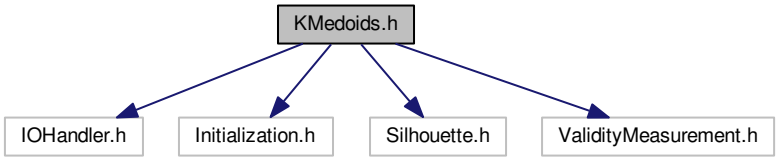
5.1.1.1 bool isPBF

Definition at line 16 of file main.cpp.

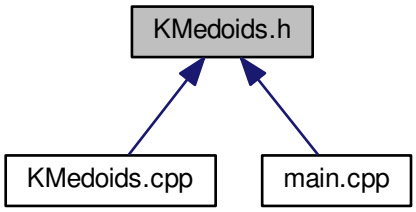
5.2 KMedoids.h File Reference

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

Include dependency graph for KMedoids.h:



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



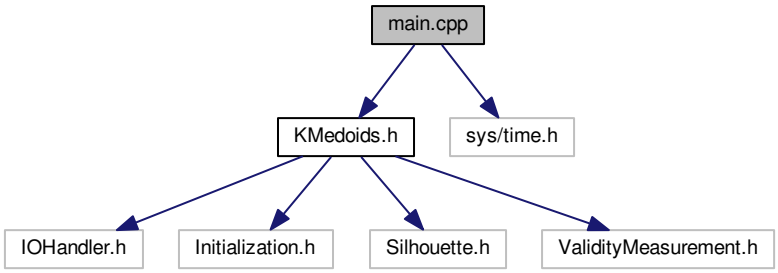
Classes

- struct Parameter
- struct TimeRecorder
- class KMedoids

5.3 main.cpp File Reference

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

Include dependency graph for main.cpp:



Functions

- void featureExtraction (const int &argc, char **argv)
- void performKMedoids (const string &fileName, const std::vector< std::vector< float > > &dataVec, const int &dimension, const string &fullName, const KMedoids &kmedoid, const int &normOption, Silhouette &sil, TimeRecorder &tr)
- void recordInitilization (const Parameter &pm, const int &sampleOption)
- int main (int argc, char *argv[])

Variables

- bool isPBF
- bool readCluster

5.3.1 Function Documentation

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

Definition at line 78 of file main.cpp.

```
80 {
81     while(number!=3)
82     {
83         std::cout << "Input argument should have 3!" << endl
84                   << "./cluster inputFile_name(in dataset folder) "
85                   << "data_dimension(3)" << endl;
86         exit(1);
87     }
88     const string& strName = string("../dataset/")+string(argv[1]);
89     const int& dimension = atoi(argv[2]);
90
91     /* check whether it is a PBF data set */
92     std::cout << "It is a PBF dataset? 1. Yes, 0. No." << std::endl;
93     int isPBFInput;
94     std::cin >> isPBFInput;
95     assert(isPBFInput==1||isPBFInput==0);
96     isPBF = (isPBFInput==1);
97
98     /* check whether it is a Pathline data set or not */
99     bool isPathlines;
100     std::cout << "It is a Pathline? 1.Yes, 0. No" << std::endl;
101     std::cin >> isPBFInput;
```

```
102     assert(isPBFInput==1||isPBFInput==0);
103     isPathlines = (isPBFInput==1);
104 
105     int vertexCount;
106 
107 /*-------------------------------------Input parameter choice-------------------------*/
108     Parameter pm;
109 
110     std::cout << "Please choose initialization option for seeds:" << std::endl
111               << "1.chose random positions, 2.Chose from samples, 3.k-means++ sampling" << endl;
112     std::cin >> pm.initialization;
113     assert(pm.initialization==1||pm.initialization==2||pm.
    initialization==3);
114 
115     int sampleOption;
116     std::cout << "Please choose sample strategy option for centroids: " << std::endl
117               << "1.from sample, 2.by iterations" << std::endl;
118     std::cin >> sampleOption;
119     assert(sampleOption==1||sampleOption==2);
120     if(sampleOption==1)
121         pm.isSample = true;
122     else if(sampleOption==2)
123         pm.isSample = false;
124 
125     if(isPathlines)
126         sampleOption = 1;
127     else
128     {
129         std::cout << "choose a sampling method for the dataset?" << std::endl
130                   << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
131         std::cin >> sampleOption;
132     }
133     assert(sampleOption==1||sampleOption==2);
134 
135     std::cout << "Please choose cluster number method, 0.user input, 1.read clustering: " << std::endl;
136     int clusterInput;
137     std::cin >> clusterInput;
138     assert(clusterInput==0 || clusterInput==1);
139     readCluster = (clusterInput==1);
140 
141 /*-------------------------------------Finish parameter choice-------------------------*/
142 
143     TimeRecorder tr;
144 
145     std::unordered_map<int,int> clusterMap;
146     if(readCluster)
147     {
148         IOHandler::readClusteringNumber(clusterMap, "cluster_number");
149     }
150 
151     /* a Silhouette method to estimate the clustering effect */
152     Silhouette silhou;
153 
154     struct timeval start, end;
155     double timeTemp;
156     int maxElements;
157 
158     gettimeofday(&start, NULL);
159     std::vector< std::vector<float> > dataVec;
160     IOHandler::readFile(strName, dataVec, vertexCount, dimension, maxElements);
161     //IOHandler::readFile(pbfPath, dataVec, vertexCount, dimension, 128000, 1500);
162     gettimeofday(&end, NULL);
163     timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
164               + end.tv_usec - start.tv_usec) / 1.e6;
165     tr.eventList.push_back("I-O file reader takes: ");
166     tr.timeList.push_back(to_string(timeTemp)+"s");
167 
168     stringstream ss;
169     ss << strName << "_differentNorm_full.vtk";
170     const string& fullName = ss.str();
171     IOHandler::printVTK(ss.str(), dataVec, vertexCount, dimension);
172     ss.str("");
173 
174     Eigen::MatrixXf data;
175     std::vector<float> averageS;
176 
177     if(sampleOption==1)
178         IOHandler::expandArray(data, dataVec, dimension, maxElements);
179     else if(sampleOption==2)
180         IOHandler::sampleArray(data, dataVec, dimension, maxElements);
181 
182     /*  0: Euclidean Norm
183         1: Fraction Distance Metric
184         2: piece-wise angle average
185         3: Bhattacharyya metric for rotation
186         4: average rotation
187         5: signed-angle intersection
```

```
188         6: normal-direction multivariate distribution
189         7: Bhattacharyya metric with angle to a fixed direction
190         8: Piece-wise angle average \times standard deviation
191         9: normal-direction multivariate un-normalized distribution
192         10: x*y/|x||y| borrowed from machine learning
193         11: cosine similarity
194         12: Mean-of-closest point distance (MCP)
195         13: Hausdorff distance min_max(x_i,y_i)
196         14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
197         15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
198         16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
199         17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/
    S0097849318300128
200             for pathlines only
201     */
202
203     KMedoids kmedoid(pm, data, -1);
204
205     recordInitilization(pm, sampleOption);
206
207     for(int i = 0;i<=17;i++)
208     {
209         if(isPathlines)
210         {
211             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15 && i!=17)
212                 continue;
213         }
214         else
215         {
216             if(i!=0 && i!=1 && i!=2 && i!=4 && i!=12 && i!=13 && i!=14 && i!=15)
217                 continue;
218         }
219
220         if(readCluster)
221             kmedoid.numOfClusters = clusterMap[i];
222         else
223         {
224             std::cout << "Please input a cluster number (>=2) for norm " << i << " in [2, "
225                     << dataVec.size() << "]: " << std::endl;
226             std::cin >> kmedoid.numOfClusters;
227         }
228
229         gettimeofday(&start, NULL);
230         ss << strName << "_KMedoids";
231         performKMedoids(ss.str(), dataVec, dimension, fullName, kmedoid, i, silhou, tr);
232         ss.str("");
233         gettimeofday(&end, NULL);
234         timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
235         tr.eventList.push_back("Direct K_Means operation time for norm "+to_string(i)+" takes: ");
236         tr.timeList.push_back(to_string(timeTemp)+"s");
237         if(silhou.sData.empty())
238             silhou.sAverage = 0;
239
240         IOHandler::writeReadme(tr.eventList, tr.timeList, kmedoid.numOfClusters);
241         tr.eventList.clear();
242         tr.timeList.clear();
243         silhou.reset();
244     }
245 }
```

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

Definition at line 65 of file main.cpp.

```
66 {
67     featureExtraction(argc, argv);
68     return 0;
69 }
```

5.3.1.3 void performKMedoids (const string & *fileName*, const std::vector< std::vector< float > > & *dataVec*, const int & *dimension*, const string & *fullName*, const KMedoids & *kmedoid*, const int & *normOption*, Silhouette & *sil*, TimeRecorder & *tr*)

Definition at line 260 of file main.cpp.

```
268 {
269     FeatureLine fl(dataVec);
270     kmedoid.getMedoids(fl, normOption, sil, tr);
271 
272     std::vector<std::vector<float> > closestStreamline, furthestStreamline;
273     std::vector<int> closestCluster, furthestCluster, meanCluster;
274     int closestPoint, furthestPoint;
275     IOHandler::assignVec(closestStreamline, closestCluster, fl.closest,
276                          closestPoint, dataVec);
277     IOHandler::assignVec(furthestStreamline, furthestCluster, fl.furthest,
278                          furthestPoint, dataVec);
279 
280 
281 /* get the average rotation of the extraction */
282     std::vector<float> closestRotation, furthestRotation;
283     const float& closestAverage = getRotation(closestStreamline, closestRotation);
284     const float& furthestAverage = getRotation(furthestStreamline, furthestRotation);
285 
286     tr.eventList.push_back("Average rotation of closest for K-medoids clustering on norm "
287                            + to_string(normOption) + " is: ");
288     tr.timeList.push_back(to_string(closestAverage));
289 
290     tr.eventList.push_back("Average rotation of furthest for K-medoids clustering on norm "
291                            + to_string(normOption) + " is: ");
292     tr.timeList.push_back(to_string(furthestAverage));
293 /* finish the rotation computation */
294 
295     IOHandler::assignVec(meanCluster, fl.centerMass);
296     IOHandler::printVTK(fileName+string("_norm")+to_string(normOption)+string("_mean.vtk"),
297                         fl.centerMass,
298                         fl.centerMass.size()*fl.centerMass[0].minCenter.size()/dimension,
299                         dimension, sil.sCluster);
300     IOHandler::printVTK(fileName+"_norm"+to_string(normOption)+"_closest.vtk",
301                         closestStreamline, closestPoint/dimension, dimension,
302                         closestCluster, sil.sCluster);
303     IOHandler::printVTK(fileName+"_norm"+to_string(normOption)+"_furthest.vtk",
304                         furthestStreamline, furthestPoint/dimension,
305                         dimension, furthestCluster, sil.sCluster);
306     std::cout << "Finish printing vtk for k-means clustering result!" << std::endl;
307 
308     IOHandler::printToFull(dataVec, fl.group, fl.totalNum, string("norm")+to_string(normOption)
309                            +string("_KMedoids"), fullName, dimension);
310     //IOHandler::writeReadme(fl.closest, fl.furthest, normOption);
311 
312     IOHandler::printToFull(dataVec, sil.sData, "norm"+to_string(normOption)+"_SValueLine",
313                            fullName, 3);
314     IOHandler::printToFull(dataVec, fl.group, sil.sCluster,
315               "norm"+to_string(normOption)+"_SValueCluster", fullName, 3);
316 }
```

5.3.1.4 void recordInitilization (const Parameter & *pm*, const int & *sampleOption*)

Definition at line 325 of file main.cpp.

```
327 {
328     std::ofstream readme("../dataset/README", ios::out | ios::app);
329     if(readme.fail())
330     {
331         std::cout << "cannot create README file!" << std::endl;
332         exit(1);
333     }
334 
335     readme << std::endl;
336     readme << "Initial centroid is: ";
337     if(pm.initialization==1)
338         readme << pm.initialization << ".random initialization"
339                << std::endl;
340     else if(pm.initialization==2)
341         readme << pm.initialization << ".sample initialization"
342                << std::endl;
343     else if(pm.initialization==3)
344         readme << pm.initialization << ".kmedoids++ initialization"
345                << std::endl;
346 
347     readme << "Medoid is: ";
348     if(pm.isSample)
349         readme << pm.isSample << ".inside samples" << std::endl;
350     else
351         readme << pm.isSample << ".from iterations" << std::endl;
352 
```

```
353     readme << "Sampling is: ";
354     if(sampleOption==1)
355         readme << sampleOption << ".directly filling" << std::endl;
356     else if(sampleOption==2)
357         readme << sampleOption << ".uniformly sampling" << std::endl;
358 
359     readme.close();
360 
361 }
```

5.3.2 Variable Documentation

5.3.2.1 bool isPBF

Definition at line 16 of file main.cpp.

5.3.2.2 bool readCluster

Definition at line 21 of file main.cpp.

5.4 README.md File Reference

Index