DBSCAN Clustering

The C++ implmentation for DBSCAN clustering

Generated by Doxygen 1.8.11

Contents

Chapter 1

DBSCAN Description

Parameter selection

Two critical parameters for DBSCAN clustering

- The **minPts** which describes *how many neighbor candidates needed to create a core point*
- The **radius** which describes *how large the searched area can be to define a core point* which are, however, pretty difficult for parameter tuning varying on different data sets.

Our implementation details

We only use one parameter, **minPts**, and the other parameter, **radius**, is set to be the minPts-th smallest distance to the candidate line from all its neighbors.

minPts can be totally user defined, or to be default, set by 6 in all the data sets and it performs pretty well to generate around 10-100 clusters for fair comparisons of clustering combinations in our paper.

From this perspective, DBSCAN is not suitable for scientific data visualization depsite it has benefit of lower overhead and resource requirement for calculation.

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 DataSet Struct Reference

`#include <DensityClustering.h>`

Public Member Functions

- DataSet ()
- ~DataSet ()

Public Attributes

- vector< vector< float > > dataVec
- Eigen::MatrixXf dataMatrix
- int maxElements
- int vertexCount
- int dimension
- string strName
- string fullName

4.1.1 Detailed Description

Definition at line 48 of file DensityClustering.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DataSet::DataSet ( ) [inline]

Definition at line 59 of file DensityClustering.h.

```
60     {}
```

4.1.2.2 DataSet::~DataSet ( ) [inline]

Definition at line 62 of file DensityClustering.h.

```
63     {}
```

4.1.3 Member Data Documentation

4.1.3.1 Eigen::MatrixXf DataSet::dataMatrix

Definition at line 51 of file DensityClustering.h.

4.1.3.2 vector<vector<float> > DataSet::dataVec

Definition at line 50 of file DensityClustering.h.

4.1.3.3 int DataSet::dimension

Definition at line 54 of file DensityClustering.h.

4.1.3.4 string DataSet::fullName

Definition at line 57 of file DensityClustering.h.

4.1.3.5 int DataSet::maxElements

Definition at line 52 of file DensityClustering.h.

4.1.3.6 string DataSet::strName

Definition at line 56 of file DensityClustering.h.

4.1.3.7 int DataSet::vertexCount

Definition at line 53 of file DensityClustering.h.

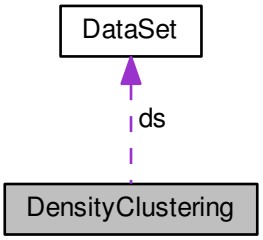
The documentation for this struct was generated from the following file:

- DensityClustering.h

4.2 DensityClustering Class Reference

`#include <DensityClustering.h>`

Collaboration diagram for DensityClustering:



Public Member Functions

- DensityClustering (const int &argc, char **argv)
- ~DensityClustering ()
- void performClustering ()

Private Member Functions

- void setDataset (const int &argc, char **argv)
- void setNormOption ()
- void DBSCAN (const float &radius_eps, const int &minPts)
- void expandCluster (const int &index, vector< int > &neighbor, const int &cluster_id, const float &radius_eps, const int &minPts)
- const vector< int > regionQuery (const int &index, const float &radius_eps)
- void getDistRange (float &minDist, float &maxDist)
- const int setMinPts ()
- const float setTimesMin (const float &minDist, const float &maxDist)
- void extractFeatures (const float &radius_eps, const int &minPts)
- const float getDistThreshold (const int &minPts)
- const float getAverageDist (const int &minPts)

Private Attributes

- vector< PointNode > nodeVec
- MetricPreparation object
- int normOption
- DataSet ds
- bool isPBF
- bool isPathlines

4.2.1 Detailed Description

Definition at line 70 of file DensityClustering.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DensityClustering::DensityClustering (const int & *argc*, char ** *argv*)

Definition at line 29 of file DensityClustering.cpp.

```
30 {
31     struct timeval start, end;
32     double timeTemp;
33     gettimeofday(&start, NULL);
34 
35     // set data set and norm option
36     setDataset(argc, argv);
37     setNormOption();
38 
39     // create the object for distance matrix computation
40     object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
41     object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
42             ds.dataMatrix.cols(), normOption);
43 
44     /* if the dataset is not PBF, then should record distance matrix for Gamma matrix compution */
45     if (!isPBF)
46     {
47         deleteDistanceMatrix(ds.dataMatrix.rows());
48 
49         std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
50         if(distFile.fail()) // the distance matrix file not exists, will create new one
51         {
52             distFile.close();
53             getDistanceMatrix(ds.dataMatrix, normOption, object);
54             std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
55             for(int i=0;i<ds.dataMatrix.rows();++i)
56             {
57                 for(int j=0;j<ds.dataMatrix.rows();++j)
58                 {
59                     distFileOut << distanceMatrix[i][j] << " ";
60                 }
61                 distFileOut << std::endl;
62             }
63             distFileOut.close();
64         }
65         else    // distance matrix file exists, directly read from the file
66         {
67             std::cout << "read distance matrix..." << std::endl;
68 
69             distanceMatrix = new float*[ds.dataMatrix.rows()];
70         #pragma omp parallel for schedule(static) num_threads(8)
71             for (int i = 0; i < ds.dataMatrix.rows(); ++i)
72             {
73                 distanceMatrix[i] = new float[ds.dataMatrix.rows()];
74             }
75 
76             int i=0, j;
77             string line;
78             stringstream ss;
79             while(getline(distFile, line))
80             {
81                 j=0;
82                 ss.str(line);
83                 while(ss>>line)
84                 {
85                     if(i==j)
86                         distanceMatrix[i][j] = 0;
87                     else
88                         distanceMatrix[i][j] = std::atof(line.c_str());
89                     ++j;
90                 }
91                 ++i;
92                 ss.str("");
93                 ss.clear();
94             }
95             distFile.close();
96         }
```

```
97      }
98
99      gettimeofday(&end, NULL);
100      timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
101               - start.tv_usec) / 1.e6;
102      activityList.push_back("Computing distance matrix for norm "+to_string(
       normOption)+" takes: ");
103      timeList.push_back(to_string(timeTemp) + " s");
104
105      nodeVec = vector<PointNode>(ds.dataMatrix.rows(),
       PointNode());
106 }
```

4.2.2.2 DensityClustering::~DensityClustering ( )

Definition at line 112 of file DensityClustering.cpp.

```
112                                     {
113
114 }
```

4.2.3 Member Function Documentation

4.2.3.1 void DensityClustering::DBSCAN (const float & *radius_eps,* const int & *minPts*) [private]

Definition at line 217 of file DensityClustering.cpp.

```
217                                                                     {
218     int C = 0;
219     for (int i = 0; i < ds.dataMatrix.rows(); ++i)
220     {
221         if (nodeVec[i].visited)
222             continue;
223         nodeVec[i].visited = true;
224         vector<int> neighbor = regionQuery(i, radius_eps);
225         if (neighbor.size() < minPts)
226             nodeVec[i].type = NOISE;
227         else
228         {
229             expandCluster(i, neighbor, C, radius_eps, minPts);
230             ++C;
231         }
232     }
233 }
```

4.2.3.2 void DensityClustering::expandCluster (const int & *index,* vector< int > & *neighbor,* const int & *cluster_id,* const float & *radius_eps,* const int & *minPts*) [private]

Definition at line 245 of file DensityClustering.cpp.

```
247 {
248     nodeVec[index].group = cluster_id;
249     int insideElement;
250     for (int i = 0; i < neighbor.size(); ++i) {
251         insideElement = neighbor[i];
252         if (!nodeVec[insideElement].visited) {
253             nodeVec[insideElement].visited = true;
254             vector<int> newNeighbor = regionQuery(insideElement, radius_eps);
255             if (newNeighbor.size() >= minPts) {
256                 neighbor.insert(neighbor.end(), newNeighbor.begin(),
257                         newNeighbor.end());
258             }
259         }
260         if (nodeVec[insideElement].group == -1)
261             nodeVec[insideElement].group = cluster_id;
262     }
263 }
```

4.2.3.3 void DensityClustering::extractFeatures (const float & *radius_eps*, const int & *minPts*) [private]

Definition at line 480 of file DensityClustering.cpp.

```
481 {
482     // find the maximal cluster labels with openmp critical operation, could be disabled
483     int maxGroup = -INT_MAX + 1;
484 #pragma omp parallel num_threads(8)
485     {
486     #pragma omp for nowait
487         for (int i = 0; i < nodeVec.size(); ++i) {
488             int groupID = nodeVec[i].group;
489         #pragma omp critical
490             {
491                 if (groupID != -1 && groupID > maxGroup)
492                     maxGroup = groupID;
493             }
494         }
495     }
496     std::cout << "Max group is: " << maxGroup << std::endl;
497
498     /* re-index the group id by increasing number */
499     int numClusters = maxGroup + 1;
500     std::vector<int> container(numClusters, 0);
501     for (int i = 0; i < nodeVec.size(); ++i) {
502         if (nodeVec[i].group != -1)
503             ++container[nodeVec[i].group];
504     }
505
506     int increasingOrder[numClusters];
507     std::multimap<int, int> groupMap;
508
509     for (int i = 0; i < numClusters; ++i)
510         groupMap.insert(std::pair<int, int>(container[i], i));
511
512     std::fill(container.begin(), container.end(), 0);
513     int groupNo = 0;
514     for (std::multimap<int, int>::iterator it = groupMap.begin();
515             it != groupMap.end(); ++it) {
516         if (it->first > 0) {
517             increasingOrder[it->second] = groupNo;
518             container[groupNo] = it->first;
519             ++groupNo;
520         }
521     }
522
523     numClusters = groupNo + 1; /* plus -1 as group */
524
525 #pragma omp parallel for schedule(static) num_threads(8)
526     for (int i = 0; i < nodeVec.size(); ++i) {
527         if (nodeVec[i].group != -1)
528             nodeVec[i].group = increasingOrder[nodeVec[i].group];
529     }
530
531     /* in case -1, we use 0 to record number of -1 as noise */
532     std::vector<int> item_cids(nodeVec.size());
533     std::vector<std::vector<int> > storage(numClusters);
534     /* -1 group as group[0] */
535     for (int i = 0; i < nodeVec.size(); ++i) {
536         item_cids[i] = nodeVec[i].group;
537         storage[nodeVec[i].group + 1].push_back(i);
538     }
539
540     container.insert(container.begin(), storage[0].size());
541
542     /* compute balanced Entropy value for the clustering algorithm */
543     const int& Row = ds.dataMatrix.rows();
544     float entropy = 0.0, probability;
545     for (int i = 0; i < container.size(); ++i) {
546         probability = float(container[i]) / float(Row);
547         entropy += probability * log2f(probability);
548     }
549     entropy = -entropy / log2f(numClusters);
550
551     IOHandler::printClustersNoise(ds.dataVec, item_cids, container, "norm" + to_string(
    normOption),
552             ds.fullName, ds.dimension);
553
554     struct timeval start, end;
555     double timeTemp;
556
557     numClusters -= 1;
558
559     const int& numNoise = storage[0].size();
```

```
560     storage.erase(storage.begin());
561
562     /* record labeling information */
563     // IOHandler::generateGroups(storage);
564
565     /* compute the centroid coordinates of each clustered group */
566
567     gettimeofday(&start, NULL);
568
569     // compute the centroid coordinates for the clusters
570     Eigen::MatrixXf centroid = MatrixXf::Zero(numClusters, ds.dataMatrix.cols());
571     vector<vector<float> > cenVec(numClusters);
572 #pragma omp parallel for schedule(static) num_threads(8)
573     for (int i = 0; i < numClusters; ++i)
574     {
575         const std::vector<int>& groupRow = storage[i];
576         for (int j = 0; j < groupRow.size(); ++j)
577         {
578             centroid.row(i) += ds.dataMatrix.row(groupRow[j]);
579         }
580         centroid.row(i) /= groupRow.size();
581         const Eigen::VectorXf& vec = centroid.row(i);
582         cenVec[i] = vector<float>(vec.data(), vec.data() + ds.dataMatrix.cols());
583     }
584
585     // extract the streamlines closest and furthest to the centroids for each cluster
586     vector<vector<float> > closest(numClusters);
587     vector<vector<float> > furthest(numClusters);
588
589 #pragma omp parallel for schedule(static) num_threads(8)
590     for (int i = 0; i < numClusters; ++i) {
591         float minDist = FLT_MAX;
592         float maxDist = -10;
593         int minIndex = -1, maxIndex = -1;
594         const std::vector<int>& groupRow = storage[i];
595         const Eigen::VectorXf& eachCentroid = centroid.row(i);
596         for (int j = 0; j < groupRow.size(); ++j) {
597             float distance = getDisimilarity(eachCentroid, ds.dataMatrix,
598                     groupRow[j], normOption, object);
599             if (minDist > distance) {
600                 minDist = distance;
601                 minIndex = groupRow[j];
602             }
603             if (maxDist < distance) {
604                 maxDist = distance;
605                 maxIndex = groupRow[j];
606             }
607         }
608         closest[i] = ds.dataVec[minIndex];
609         furthest[i] = ds.dataVec[maxIndex];
610     }
611
612     gettimeofday(&end, NULL);
613     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
614             - start.tv_usec) / 1.e6;
615     activityList.push_back("Feature extraction takes: ");
616     timeList.push_back(to_string(timeTemp) + " s");
617
618     // calculate the normalized validity measurement
619     ValidityMeasurement vm;
620     vm.computeValue(normOption, ds.dataMatrix, item_cids, object,
    isPBF);
621     activityList.push_back("Validity measure is: ");
622     stringstream fc_ss;
623     fc_ss << vm.f_c;
624     timeList.push_back(fc_ss.str());
625
626     // calculate the silhouette, db index and gamma statistics for the evaluation
627     gettimeofday(&start, NULL);
628     Silhouette sil;
629     sil.computeValue(normOption, ds.dataMatrix, ds.
    dataMatrix.rows(), ds.dataMatrix.cols(),
630             item_cids, object, numClusters, isPBF);
631     gettimeofday(&end, NULL);
632     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
633             - start.tv_usec) / 1.e6;
634     activityList.push_back("Silhouette calculation takes: ");
635     timeList.push_back(to_string(timeTemp) + " s");
636
637     // print the cluster representatives
638     std::cout << "Finishing extracting features!" << std::endl;
639     IOHandler::printFeature("norm" + to_string(normOption) + "_closest.vtk",
640             closest, sil.sCluster, ds.dimension);
641     IOHandler::printFeature("norm" + to_string(normOption) + "_furthest.vtk",
642             furthest, sil.sCluster, ds.dimension);
643     IOHandler::printFeature("norm" + to_string(normOption) + "_centroid.vtk",
644             cenVec, sil.sCluster, ds.dimension);
```

```
645
646     IOHandler::printToFull(ds.dataVec, sil.sData,
647             "norm" + to_string(normOption) + "_SValueLine", ds.
    fullName,
648             ds.dimension);
649     IOHandler::printToFull(ds.dataVec, item_cids, sil.sCluster,
650             "norm" + to_string(normOption) + "_SValueCluster", ds.
    fullName,
651             ds.dimension);
652
653     // record some time for readme
654     activityList.push_back("Norm option is: ");
655     timeList.push_back(to_string(normOption));
656
657     activityList.push_back("numCluster is: ");
658     timeList.push_back(to_string(numClusters));
659
660     activityList.push_back("Noise number is: ");
661     timeList.push_back(to_string(numNoise));
662
663     activityList.push_back("radius eps is: ");
664     timeList.push_back(to_string(multiTimes));
665
666     activityList.push_back("MinPts is: ");
667     timeList.push_back(to_string(minPts));
668
669     IOHandler::generateReadme(activityList, timeList);
670
671     IOHandler::writeReadme(entropy, sil, "For norm "+to_string(normOption));
672
673     /* measure closest and furthest rotation */
674     std::vector<float> closestRot, furthestRot;
675     const float& closestAverage = getRotation(closest, closestRot);
676     const float& furthestAverage = getRotation(furthest, furthestRot);
677
678     IOHandler::writeReadme(closestAverage, furthestAverage);
679
680 }
```

4.2.3.4 const float DensityClustering::getAverageDist (const int & *minPts*) [private]

Definition at line 178 of file DensityClustering.cpp.

```
179 {
180     float result = 0.0;
181     const int& rowSize = ds.dataMatrix.rows();
182 #pragma omp parallel num_threads(8)
183     {
184     #pragma omp for nowait
185         for (int i = 0; i < rowSize; ++i) {
186             /* use a priority_queue<float> with n*logk time complexity */
187             std::priority_queue<float> minDistArray;
188             float tempDist;
189             for (int j = 0; j < rowSize; ++j) {
190                 if (i == j)
191                     continue;
192                 if (distanceMatrix)
193                     tempDist = distanceMatrix[i][j];
194                 else
195                     tempDist = getDisimilarity(ds.dataMatrix.row(i),
196                             ds.dataMatrix.row(j), i, j, normOption,
    object);
197
198                 minDistArray.push(tempDist);
199                 if (minDistArray.size() > minPts)
200                     minDistArray.pop();
201             }
202
203         #pragma omp critical
204             result += minDistArray.top();
205         }
206     }
207     return result / rowSize;
208 }
```

4.2.3.5 void DensityClustering::getDistRange (float & *minDist*, float & *maxDist*) [private]

Definition at line 400 of file DensityClustering.cpp.

```
401 {
402     const float& Percentage = 0.1;
403     const int& Rows = ds.dataMatrix.rows();
404     const int& chosen = int(Percentage * Rows);
405     minDist = FLT_MAX;
406     maxDist = -1.0;
407 #pragma omp parallel num_threads(8)
408     {
409     #pragma omp for nowait
410         for (int i = 0; i < chosen; ++i) {
411             float tempDist;
412             for (int j = 0; j < Rows; ++j) {
413                 if (i == j)
414                     continue;
415                 if (distanceMatrix)
416                     tempDist = distanceMatrix[i][j];
417                 else
418                     tempDist = getDisimilarity(ds.dataMatrix.row(i),
419                             ds.dataMatrix.row(j), i, j, normOption,
    object);
420                 #pragma omp critical
421                 {
422                     if (tempDist < minDist)
423                         minDist = tempDist;
424                     if (tempDist > maxDist)
425                         maxDist = tempDist;
426                 }
427             }
428         }
429     }
430     std::cout << minDist << " " << maxDist << std::endl;
431 }
```

4.2.3.6 const float DensityClustering::getDistThreshold (const int & *minPts*) [private]

Definition at line 149 of file DensityClustering.cpp.

```
150 {
151     int distOption = 2; // set the default parameter type
152     /*
153      std::cout << "Choose distThreshold setup option: 1.user input, 2.minPts-th dist." << std::endl;
154      std::cin >> distOption;*/
155
156     assert(distOption == 1 || distOption == 2);
157
158     if (distOption == 1)    // if input for the radius, should let the user know the distance range
159     {
160         float minDist, maxDist;
161         getDistRange(minDist, maxDist);
162         std::cout << "Distance range is [" << minDist << ", " << maxDist << "]."
163                 << std::endl;
164         multiTimes = setTimesMin(minDist, maxDist);
165         return multiTimes * maxDist;
166     } else if (distOption == 2) {   // otherwise, directly get the average distance
167         /* should be pointed as average distance of minPts-th dist */
168         return getAverageDist(minPts);
169     }
170 }
```

4.2.3.7 void DensityClustering::performClustering ( )

Definition at line 120 of file DensityClustering.cpp.

```
120                                         {
121
122     struct timeval start, end;
123     double timeTemp;
124     gettimeofday(&start, NULL);
125
126     // read in the minPts as a parameter
127     minPts = setMinPts();
128     float distThreshold = getDistThreshold(minPts);
129
130     // perform DBSCAN clustering
131     DBSCAN(distThreshold, minPts);
132
133     gettimeofday(&end, NULL);
134     timeTemp = ((end.tv_sec - start.tv_sec) * 1000000u + end.tv_usec
135             - start.tv_usec) / 1.e6;
136     activityList.push_back("DBSCAN clustering for norm "+to_string(
      normOption)+" takes: ");
137     timeList.push_back(to_string(timeTemp) + " s");
138
139     // extract features and calculate the clustering evaluation metrics
140     extractFeatures(distThreshold, minPts);
141 }
```

4.2.3.8 const vector< int > DensityClustering::regionQuery (const int & *index*, const float & *radius_eps*) [private]

Definition at line 273 of file DensityClustering.cpp.

```
274                                     {
275     vector<int> neighborArray;
276     neighborArray.push_back(index);
277     float tempDist;
278     for (int i = 0; i < ds.dataMatrix.rows(); ++i) {
279         if (i == index)
280             continue;
281
282         /* in case somebody uses distance matrix */
283         if (distanceMatrix)
284             tempDist = distanceMatrix[index][i];
285         else
286             tempDist = getDisimilarity(ds.dataMatrix.row(index),
287                     ds.dataMatrix.row(i), index, i, normOption,
      object);
288         if (tempDist <= radius_eps)
289             neighborArray.push_back(i);
290     }
291     return neighborArray;
292 }
```

4.2.3.9 void DensityClustering::setDataset (const int & *argc*, char ** *argv*) [private]

Definition at line 301 of file DensityClustering.cpp.

```
301                                                                 {
302     if (argc != 3) {
303         std::cout << "Input argument should have 3!" << endl
304                 << "./cluster inputFile_name(in dataset folder) "
305                 << "data_dimension(3)" << endl;
306         exit(1);
307     }
308     ds.strName = string("../dataset/") + string(argv[1]);
309     ds.dimension = atoi(argv[2]);
310
311     /* get the bool tag for isPBF */
312     std::cout << "It is a PBF dataset? 1.Yes, 0.No" << std::endl;
313     int PBFjudgement;
314     std::cin >> PBFjudgement;
315     assert(PBFjudgement == 1 || PBFjudgement == 0);
316     isPBF = (PBFjudgement == 1);
317
318     // check whether it is pathlines or not
319     std::cout << "It is a pathlines dataset? 1.Yes, 0.No" << std::endl;
```

```
320     std::cin >> PBFjudgement;
321     assert(PBFjudgement == 1 || PBFjudgement == 0);
322     isPathlines = (PBFjudgement == 1);
323 
324     // decide the sampling strategy and operation for the given data sets
325     int sampleOption;
326 
327     if(isPathlines)
328         sampleOption = 1;
329     else
330     {
331         std::cout << "choose a sampling method for the dataset?" << std::endl
332                 << "1.directly filling with last vertex; 2. uniform sampling (recommended!); "
333                 << "3. equal-arc sampling. " << std::endl;
334         std::cin >> sampleOption;
335     }
336     assert(sampleOption == 1 || sampleOption == 2 || sampleOption == 3);
337 
338     // read the coordinates from the file
339     IOHandler::readFile(ds.strName, ds.dataVec, ds.
    vertexCount, ds.dimension, ds.maxElements);
340 
341     ds.fullName = ds.strName + "_differentNorm_full.vtk";
342     IOHandler::printVTK(ds.fullName, ds.dataVec, ds.
    vertexCount, ds.dimension);
343 
344     if (sampleOption == 1)
345         IOHandler::expandArray(ds.dataMatrix, ds.dataVec,
    ds.dimension, ds.maxElements);
346     else if (sampleOption == 2)
347         IOHandler::sampleArray(ds.dataMatrix, ds.dataVec,
    ds.dimension, ds.maxElements);
348     else if (sampleOption == 3)
349         IOHandler::uniformArcSampling(ds.dataMatrix, ds.dataVec,
    ds.dimension, ds.maxElements);
350 }
```

4.2.3.10 const int DensityClustering::setMinPts ( ) [private]

Definition at line 437 of file DensityClustering.cpp.

```
437                                     {
438     /*std::cout << std::endl;
439     std::cout << "Input the minPts for DBSCAN in [0" << ", "
440             << ds.dataMatrix.rows() << "], 6 preferred: " << std::endl;*/
441     int minPts = 6;
442     //std::cin >> minPts;
443     if (minPts <= 0 || minPts >= ds.dataMatrix.rows()) {
444         std::cout << "Error for out-of-range minPts!" << std::endl;
445         exit(1);
446     }
447     return minPts;
448 }
```

4.2.3.11 void DensityClustering::setNormOption ( ) [private]

Definition at line 356 of file DensityClustering.cpp.

```
356                                     {
357 
358     if(isPathlines)
359     {
360         std::cout << "Choose a norm from 0-17!" << std::endl;
361         std::cin >> normOption;
362         assert(normOption>=0 && normOption<=17);
363     }
364     else
365     {
366         std::cout << "Choose a norm from 0-15!" << std::endl;
367         std::cin >> normOption;
368         assert(normOption>=0 && normOption<=15);
369     }
```

```
370
371     /*  0: Euclidean Norm
372      1: Fraction Distance Metric
373      2: piece-wise angle average
374      3: Bhattacharyya metric for rotation
375      4: average rotation
376      5: signed-angle intersection
377      6: normal-direction multivariate distribution
378      7: Bhattacharyya metric with angle to a fixed direction
379      8: Piece-wise angle average \times standard deviation
380      9: normal-direction multivariate un-normalized distribution
381      10: x*y/|x||y| borrowed from machine learning
382      11: cosine similarity
383      12: Mean-of-closest point distance (MCP)
384      13: Hausdorff distance min_max(x_i,y_i)
385      14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
386      15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
387      16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
388      17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/S0097849318300128
389          for pathlines only
390      */
391 }
```

4.2.3.12 const float DensityClustering::setTimesMin (const float & *minDist*, const float & *maxDist*) [private]

Definition at line 458 of file DensityClustering.cpp.

```
459 {
460     std::cout << std::endl;
461     float lowerBound = minDist / maxDist;
462     std::cout << "Input the multiplication for DBSCAN radius in [" << lowerBound
463             << ",1.0]:" << std::endl;
464     float multiTimes;
465     std::cin >> multiTimes;
466     if (multiTimes >= 1.0 || multiTimes <= lowerBound) {
467         std::cout << "Error for out-of-range minPts!" << std::endl;
468         exit(1);
469     }
470     return multiTimes;
471 }
```

4.2.4 Member Data Documentation

4.2.4.1 DataSet DensityClustering::ds [private]

Definition at line 116 of file DensityClustering.h.

4.2.4.2 bool DensityClustering::isPathlines [private]

Definition at line 126 of file DensityClustering.h.

4.2.4.3 bool DensityClustering::isPBF [private]

Definition at line 121 of file DensityClustering.h.

4.2.4.4 vector<PointNode> DensityClustering::nodeVec [private]

Definition at line 101 of file DensityClustering.h.

4.2.4.5 int DensityClustering::normOption [private]

Definition at line 111 of file DensityClustering.h.

4.2.4.6 MetricPreparation DensityClustering::object [private]

Definition at line 106 of file DensityClustering.h.

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

- DensityClustering.h
- DensityClustering.cpp

4.3 PointNode Struct Reference

`#include <DensityClustering.h>`

Public Member Functions

- PointNode ()
- ~PointNode ()

Public Attributes

- int type
- bool visited
- int group

4.3.1 Detailed Description

Definition at line 32 of file DensityClustering.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 PointNode::PointNode ( ) [inline]

Definition at line 37 of file DensityClustering.h.

```
37                 :type(-1), visited(false), group(-1)
38     {}
```

4.3.2.2 PointNode::~PointNode ( ) [inline]

Definition at line 40 of file DensityClustering.h.

```
41     {}
```

4.3.3 Member Data Documentation

4.3.3.1 int PointNode::group

Definition at line 36 of file DensityClustering.h.

4.3.3.2 int PointNode::type

Definition at line 34 of file DensityClustering.h.

4.3.3.3 bool PointNode::visited

Definition at line 35 of file DensityClustering.h.

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

- DensityClustering.h

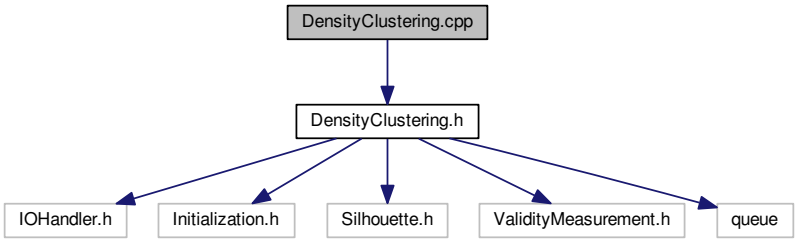
Chapter 5

File Documentation

5.1 DensityClustering.cpp File Reference

```
#include "DensityClustering.h"
```

Include dependency graph for DensityClustering.cpp:



Variables

- std::vector< string > activityList
- std::vector< string > timeList
- float multiTimes
- int minPts

5.1.1 Variable Documentation

5.1.1.1 std::vector<string> activityList

Definition at line 8 of file DensityClustering.cpp.

5.1.1.2 int minPts

Definition at line 20 of file DensityClustering.cpp.

5.1.1.3 float multiTimes

Definition at line 15 of file DensityClustering.cpp.

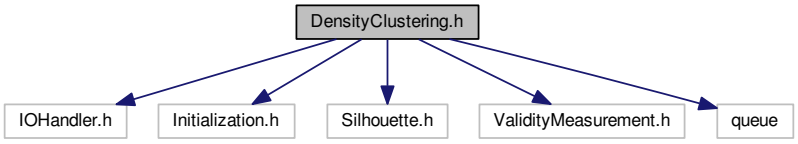
5.1.1.4 std::vector<string> timeList

Definition at line 9 of file DensityClustering.cpp.

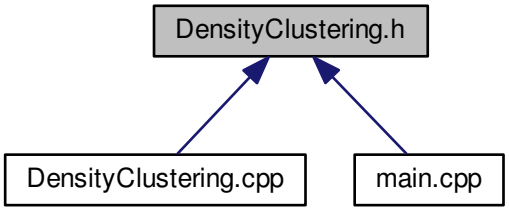
5.2 DensityClustering.h File Reference

```
#include "IOHandler.h"
#include "Initialization.h"
#include "Silhouette.h"
#include "ValidityMeasurement.h"
#include <queue>
```

Include dependency graph for DensityClustering.h:



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



Classes

- struct PointNode
- struct DataSet
- class DensityClustering

Enumerations

- enum PointType { CORE = 0, BORDER, NOISE }

5.2.1 Enumeration Type Documentation

5.2.1.1 enum PointType

Enumerator

CORE

BORDER

NOISE

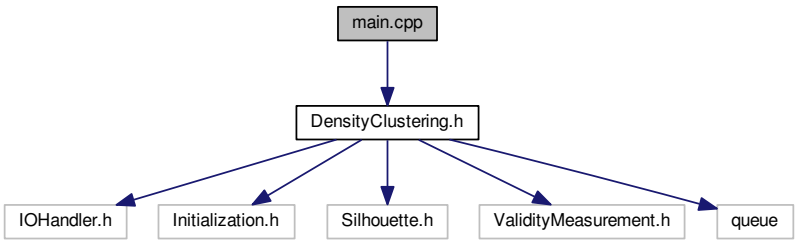
Definition at line 21 of file DensityClustering.h.

```
22 {
23     CORE = 0,
24     BORDER,
25     NOISE
26 };
```

5.3 main.cpp File Reference

```
#include "DensityClustering.h"
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char **argv)

5.3.1 Function Documentation

5.3.1.1 int main (int *argc*, char ** *argv*)

Definition at line 16 of file main.cpp.

```
17 {
18     DensityClustering dclustering(argc, argv);
19     dclustering.performClustering();
20     return 0;
21 }
```

5.4 README.md File Reference

Index