OPTICS Clustering

The C++ implmentation for OPTICS clustering

Generated by Doxygen 1.8.11

Contents

Chapter 1

OPTICS

We implemented the basic OPTICS as described by `wikepedia`. The OPTICS is claimed to have better performance than the DBSCAN with more natural clusters generated while solving the leading drawback of DBSCAN, at the cost of more complicated parameter setting.

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 <Predefined.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 193 of file Predefined.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DataSet::DataSet ( ) [inline]

Definition at line 204 of file Predefined.h.

```
{}
```

4.1.2.2 DataSet::∼DataSet ( ) [inline]

Definition at line 207 of file Predefined.h.

```
208     {}
```

4.1.3 Member Data Documentation

4.1.3.1 Eigen::MatrixXf DataSet::dataMatrix

Definition at line 196 of file Predefined.h.

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

Definition at line 195 of file Predefined.h.

4.1.3.3 int DataSet::dimension

Definition at line 199 of file Predefined.h.

4.1.3.4 string DataSet::fullName

Definition at line 202 of file Predefined.h.

4.1.3.5 int DataSet::maxElements

Definition at line 197 of file Predefined.h.

4.1.3.6 string DataSet::strName

Definition at line 201 of file Predefined.h.

4.1.3.7 int DataSet::vertexCount

Definition at line 198 of file Predefined.h.

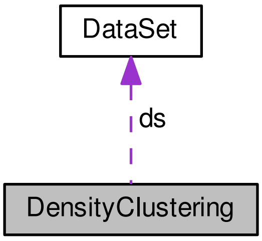
The documentation for this struct was generated from the following file:

- Predefined.h

4.2 DensityClustering Class Reference

```
#include <OPTICS.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 OPTICS (const float &radius_eps, const int &minPts)
- void update (const int &index, const vector< int > &neighbor, LinkedList &seeds, 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)
- const float getReachability (const int &first, const int &target, const int &minPts)
- void extractFeatures (const float &radius_eps, const int &minPts)
- void computeCoredDistance (const float &radius_eps, const int &minPts)
- void getGroup (const float &radius_eps)
- void writeReachability ()
- const float getMinPt_thDist (const int &minPts)

Private Attributes

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

4.2.1 Detailed Description

Definition at line 20 of file OPTICS.h.

4.2.2 Constructor & Destructor Documentation

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

Definition at line 32 of file OPTICS.cpp.

```
{
    struct timeval start, end;
    double timeTemp;
    gettimeofday(&start, NULL);

    setDataset(argc, argv);
    setNormOption();

    object = MetricPreparation(ds.dataMatrix.rows(), ds.dataMatrix.cols());
    object.preprocessing(ds.dataMatrix, ds.dataMatrix.rows(),
   ds.dataMatrix.cols(), normOption);

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

        getDistanceMatrix(ds.dataMatrix, normOption, object);

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

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

    gettimeofday(&end, NULL);
    timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
               + end.tv_usec - start.tv_usec) / 1.e6;
    activityList.push_back("Distance matrix for norm "+to_string(normOption)+" takes: ");
    timeList.push_back(to_string(timeTemp)+" s");

    nodeVec = vector<PointNode>(ds.dataMatrix.rows(), PointNode());
}
```

4.2.2.2 DensityClustering::∼DensityClustering ( )

Definition at line 114 of file OPTICS.cpp.

```
{

}
```

4.2.3 Member Function Documentation

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

Definition at line 684 of file OPTICS.cpp.

```
{
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < ds.dataMatrix.rows(); ++i)
    {
        vector<float> distRecord;   //record distance value
        for (int j = 0; j < ds.dataMatrix.rows(); ++j)
        {
            if(j==i)
                continue;
            float tempDist;
            if(distanceMatrix)
                tempDist = distanceMatrix[i][j];
            else
                tempDist = getDisimilarity(ds.dataMatrix.row(i),ds.dataMatrix.row(j),i, j, normOption, object);
            if(tempDist<=radius_eps)
            {
                nodeVec[i].neighbor.push_back(j);
                distRecord.push_back(tempDist);
            }
        }
        /* find minPts-th smallest element in vector by linear traversal */
        if(distRecord.size()>=minPts)
        {
            /* A k*n complex algorithm */
            /*
            vector<float> smallestRange(minPts,FLT_MAX);    //update to get minPts-th smallest
            for(int k=0;k<distRecord.size();++k)
            {
                if(distRecord[k]<smallestRange[minPts-1])
                    smallestRange[minPts-1]=distRecord[k];
                for(int l=minPts-1;l>=1;--l)
                {
                    if(smallestRange[l]>smallestRange[l-1])
                        std::swap(smallestRange[l], smallestRange[l-1]);
                }
            }
            nodeVec[i].core_distance = smallestRange[minPts-1];
            */

            /* instead we shall apply a n*logk algorithm */
            std::priority_queue<float> smallestRange;
            for(int k=0;k<distRecord.size();++k)
            {
                smallestRange.push(distRecord[k]);
                if(smallestRange.size()>minPts)
                    smallestRange.pop();
            }
            nodeVec[i].core_distance = smallestRange.top();
        }
    }

    std::cout << "Precomputing for cored-distance is done!" << std::endl;
}
```

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

Definition at line 473 of file OPTICS.cpp.

```
{
    int maxGroup = -INT_MAX+1;
#pragma omp parallel num_threads(8)
    {
    #pragma omp for nowait
        for (int i = 0; i < nodeVec.size(); ++i)
        {
            int groupID = nodeVec[i].group;
        #pragma omp critical
            {
                if(groupID!=-1 && groupID>maxGroup)
                    maxGroup = groupID;
            }
        }
    }
    std::cout << "Max group is: " << maxGroup << std::endl;

/* re-index the group id by increasing number */
    int numClusters = maxGroup+1;
    std::vector<int> container(numClusters,0);
    for (int i = 0; i < nodeVec.size(); ++i)
    {
        if(nodeVec[i].group!=-1)
            ++container[nodeVec[i].group];
    }

    int increasingOrder[numClusters];
    std::multimap<int,int> groupMap;

    for (int i = 0; i < numClusters; ++i)
        groupMap.insert(std::pair<int,int>(container[i],i));

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

    numClusters = groupNo+1;    /* plus -1 as group */

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < nodeVec.size(); ++i)
    {
        if(nodeVec[i].group!=-1)
            nodeVec[i].group=increasingOrder[nodeVec[i].group];
    }

    /* in case -1, we use 0 to record number of -1 as noise */

    std::vector<int> item_cids(nodeVec.size());
    std::vector<std::vector<int> > storage(numClusters);
    /* -1 group as group[0] */
    for (int i = 0; i < nodeVec.size(); ++i)
    {
        item_cids[i] = nodeVec[i].group;
        storage[nodeVec[i].group+1].push_back(i);
    }

    container.insert(container.begin(),storage[0].size());

    const int& Row = ds.dataMatrix.rows();
    float entropy = 0.0, probability;
    for(int i=0;i<container.size();++i)
    {
        probability = float(container[i])/float(Row);
        entropy+=probability*log2f(probability);
    }
    entropy = -entropy/log2f(numClusters);


    IOHandler::printClustersNoise(ds.dataVec,item_cids,container,
         "norm"+to_string(normOption),ds.fullName,ds.dimension);

    struct timeval start, end;
    double timeTemp;

    numClusters-=1;

    const int& numNoise = storage[0].size();
    storage.erase(storage.begin());


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

    /* compute the centroid coordinates of each clustered group */

    gettimeofday(&start, NULL);

    Eigen::MatrixXf centroid = MatrixXf::Zero(numClusters,ds.dataMatrix.cols());
    vector<vector<float> > cenVec(numClusters);
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i=0;i<numClusters;++i)
    {
        const std::vector<int>& groupRow = storage[i];
        for (int j = 0; j < groupRow.size(); ++j)
        {
            centroid.row(i)+=ds.dataMatrix.row(groupRow[j]);
        }
        centroid.row(i)/=groupRow.size();
        const Eigen::VectorXf& vec = centroid.row(i);
        cenVec[i] = vector<float>(vec.data(), vec.data()+ds.dataMatrix.cols());
    }

    vector<vector<float> > closest(numClusters);
    vector<vector<float> > furthest(numClusters);

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i=0;i<numClusters;++i)
    {
        float minDist = FLT_MAX;
        float maxDist = -10;
        int minIndex = -1, maxIndex = -1;
        const std::vector<int>& groupRow = storage[i];
        const Eigen::VectorXf& eachCentroid = centroid.row(i);
        for (int j = 0; j < groupRow.size(); ++j)
        {
            float distance = getDisimilarity(eachCentroid,ds.dataMatrix,groupRow[j],normOption,object);
            if(minDist>distance)
            {
                minDist = distance;
                minIndex = groupRow[j];
            }
            if(maxDist<distance)
            {
                maxDist = distance;
                maxIndex = groupRow[j];
            }
        }
        closest[i] = ds.dataVec[minIndex];
        furthest[i] = ds.dataVec[maxIndex];
    }

/* measure closest and furthest rotation */
    std::vector<float> closestRot, furthestRot;
    const float& closestAverage = getRotation(closest, closestRot);
    const float& furthestAverage = getRotation(furthest, furthestRot);

    gettimeofday(&end, NULL);
    timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u + end.tv_usec - start.tv_usec) / 1.e6;
    activityList.push_back("Feature extraction takes: ");
    timeList.push_back(to_string(timeTemp)+" s");

    ValidityMeasurement vm;
    vm.computeValue(normOption, ds.dataMatrix, item_cids, object,
isPBF);
    activityList.push_back("Optics Validity measure is: ");
    stringstream fc_ss;
    fc_ss << vm.f_c;
    timeList.push_back(fc_ss.str());

    gettimeofday(&start, NULL);
    Silhouette sil;
    sil.computeValue(normOption,ds.dataMatrix,ds.dataMatrix.rows(),ds.dataMatrix.cols(),
                     item_cids,object,numClusters, isPBF);

    gettimeofday(&end, NULL);
    timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
               + end.tv_usec - start.tv_usec) / 1.e6;
    activityList.push_back("Silhouette calculation takes: ");
    timeList.push_back(to_string(timeTemp)+" s");

    std::cout << "Finishing extracting features!" << std::endl;
    IOHandler::printFeature("norm"+to_string(normOption)+"_closest.vtk",
            closest, sil.sCluster, ds.dimension);
    IOHandler::printFeature("norm"+to_string(normOption)+"_furthest.vtk",
            furthest, sil.sCluster, ds.dimension);
    IOHandler::printFeature("norm"+to_string(normOption)+"_centroid.vtk",
            cenVec, sil.sCluster,ds.dimension);

    IOHandler::printToFull(ds.dataVec, sil.sData,
            "norm"+to_string(normOption)+"_SValueLine", ds.fullName, ds.dimension);
    IOHandler::printToFull(ds.dataVec, item_cids, sil.sCluster,
              "norm"+to_string(normOption)+"_SValueCluster", ds.fullName, ds.dimension);

    activityList.push_back("Norm option is: ");
    timeList.push_back(to_string(normOption));

    activityList.push_back("numCluster is: ");
    timeList.push_back(to_string(numClusters));

    activityList.push_back("Noise number is: ");
    timeList.push_back(to_string(numNoise));

    activityList.push_back("radius eps is: ");
    timeList.push_back(to_string(multiTimes));

    activityList.push_back("MinPts is: ");
    timeList.push_back(to_string(minPts));

    IOHandler::generateReadme(activityList,timeList);

    IOHandler::writeReadme(closestAverage, furthestAverage);

    IOHandler::writeReadme(entropy, sil, "For norm "+to_string(normOption));
}
```

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

Definition at line 386 of file OPTICS.cpp.

```
{
    const float& Percentage = 0.1;
    const int& Rows = ds.dataMatrix.rows();
    const int& chosen = int(Percentage*Rows);
    minDist = FLT_MAX;
    maxDist = -1.0;
#pragma omp parallel num_threads(8)
    {
    #pragma omp for nowait
        for (int i = 0; i < chosen; ++i)
        {
            for (int j = 0; j < Rows; ++j)
            {
                if(i==j)
                    continue;
                float dist;
                if(distanceMatrix)
                    dist = distanceMatrix[i][j];
                else
                    dist = getDisimilarity(ds.dataMatrix.row(i),ds.dataMatrix.row(j),i,j,normOption,object);
            #pragma omp critical
                {
                    if(dist<minDist)
                        minDist=dist;
                    if(dist>maxDist)
                        maxDist=dist;
                }
            }
        }
    }
}
```

4.2.3.4 void DensityClustering::getGroup (const float & *radius_eps*) [private]

Definition at line 771 of file OPTICS.cpp.

```
{
    std::cout << "----Parameter regime----" << std::endl;
    int continueOption;
    do
    {
        /* group tag information */
        int tag = 0;
        std::cout << "Input threshold for OPTICS reachability-plot: ";
        float threshold;
        std::cin >> threshold;
        threshold*=radius_eps;
        std::cout << threshold << std::endl;
        bool findSummit = false;
        for(int i=0;i<orderedList.size();++i)
        {
            if(nodeVec[orderedList[i]].reachabilityDist>=threshold)
            {
                findSummit = true;
            }
            else
            {
                if(findSummit)
                {
                    findSummit = false;
                    ++tag;
                }
                nodeVec[orderedList[i]].group = tag;
            }
        }
        std::cout << "Finally it forms " << (tag+1) << " clusters!" << std::endl;
        std::cout << "Want to continue with parameter? 1. Yes, 0. No." << std::endl;
        std::cin >> continueOption;
        assert(continueOption==1||continueOption==0);
    }while(continueOption==1);
    writeReachability();
}
```

4.2.3.5 const float DensityClustering::getMinPt_thDist (const int & *minPts*) [private]

Definition at line 836 of file OPTICS.cpp.

```
{
    float result = 0.0;
    const int& rowSize = ds.dataMatrix.rows();
    const int& seletedRow = 0.2*rowSize;
#pragma omp parallel num_threads(8)
    {
    #pragma omp for nowait
        for (int i = 0; i < seletedRow; ++i)
        {
            /* a linear k*n implementation by directly using a vector for linear mapping */
            /*
            std::vector<float> minDistVec(minPts, FLT_MAX);
            float tempDist;
            for (int j=0;j<rowSize;++j)
            {
                if(i==j)
                    continue;
                if(distanceMatrix)
                    tempDist = distanceMatrix[i][j];
                else
                    tempDist=getDisimilarity(ds.dataMatrix.row(i), ds.dataMatrix.row(j),i,j,normOption, object);

                if(tempDist<minDistVec[minPts-1])
                    minDistVec[minPts-1]=tempDist;
                for(int l=minPts-1;l>=1;--l)
                {
                    if(minDistVec[l]>minDistVec[l-1])
                        std::swap(minDistVec[l], minDistVec[l-1]);
                }
            }*/

            /* use a priority_queue<float> with n*logk time complexity */
            std::priority_queue<float> minDistArray;
            float tempDist;
            for (int j=0;j<rowSize;++j)
            {
                if(i==j)
                    continue;
                if(distanceMatrix)
                    tempDist = distanceMatrix[i][j];
                else
                    tempDist=getDisimilarity(ds.dataMatrix.row(i), ds.dataMatrix.row(j),i,j,normOption, object);

                minDistArray.push(tempDist);
                if(minDistArray.size()>minPts)
                    minDistArray.pop();

            }

        #pragma omp critical
            result += minDistArray.top();
        }
    }
    return result/seletedRow;
}
```

4.2.3.6 const float DensityClustering::getReachability (const int & *first*, const int & *target*, const int & *minPts*) [private]

Definition at line 749 of file OPTICS.cpp.

```
{
    const int& size = nodeVec[target].neighbor.size();
    if(size<minPts)
        return -1.0;
    float dist;

    if(distanceMatrix)
        dist = distanceMatrix[first][target];
    else
        dist = getDisimilarity(ds.dataMatrix.row(first),ds.dataMatrix.row(target),first, target, normOption, object);
    return std::max(nodeVec[target].core_distance, dist);
}
```

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

Definition at line 172 of file OPTICS.cpp.

```
{
    computeCoredDistance(radius_eps, minPts);

    for(int i=0;i<ds.dataMatrix.rows();++i)
    {
        if(nodeVec[i].visited)
```

```
            continue;
        const vector<int>& neighbor = nodeVec[i].neighbor;
              /*regionQuery(i,radius_eps)*/;
        nodeVec[i].visited = true;
        orderedList.push_back(i);
        if(nodeVec[i].core_distance!=-1.0)
        {
            /* linkedList is good, but would be O(n), so delete it */
             LinkedList seeds;

            /* should use priority queue */
            update(i, neighbor, seeds, radius_eps, minPts);
            pointNode *temp = seeds.start;
            while(temp)
            {
                if(nodeVec[temp->value.index].visited)
                {
                    temp = temp->next;
                    continue;
                }
                const vector<int>& neighborChild = nodeVec[temp->value.index].neighbor;
                      /*regionQuery(temp->value.index, radius_eps)*/;
                nodeVec[temp->value.index].visited = true;
                orderedList.push_back(temp->value.index);
                if(nodeVec[temp->value.index].core_distance!=-1.0)
                    update(temp->value.index, neighborChild, seeds, radius_eps, minPts);
                temp = seeds.start;
            }
        }
    }
}
```

4.2.3.8 void DensityClustering::performClustering ( )

Definition at line 123 of file OPTICS.cpp.

```
{
    float radius_eps;

    int minPts = setMinPts();

    int epsOption = 2;
    /*std::cout << "Choose eps selection method. 1. user input of multiplication, 2. minPt-th dist." << std::endl;
    std::cin >> epsOption;
    assert(epsOption==1||epsOption==2);*/

    if(epsOption==1)
    {
        float minDist, maxDist;
        getDistRange(minDist, maxDist);
        std::cout << "Distance range is [" << minDist << ", "
                  << maxDist << "]." << std::endl;
        multiTimes = setTimesMin(minDist, maxDist);

        radius_eps = maxDist*multiTimes;
    }
    else if(epsOption==2)
    {
        radius_eps = getMinPt_thDist(minPts);
    }

    struct timeval start, end;
    double timeTemp;
    gettimeofday(&start, NULL);

    OPTICS(radius_eps, minPts);

    gettimeofday(&end, NULL);
    timeTemp = ((end.tv_sec  - start.tv_sec) * 1000000u
               + end.tv_usec - start.tv_usec) / 1.e6;
    activityList.push_back("OPTICS clustering for norm "+to_string(normOption)+" takes: ");
    timeList.push_back(to_string(timeTemp)+" s");

    getGroup(radius_eps);
    extractFeatures(radius_eps, minPts);
}
```

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

Definition at line 264 of file OPTICS.cpp.

```
{
    vector<int> neighborArray;
    neighborArray.push_back(index);
    float tempDist;
    for (int i = 0; i < ds.dataMatrix.rows(); ++i)
    {
        if(i==index)
            continue;
        if(distanceMatrix)
            tempDist = distanceMatrix[index][i];
        else
            tempDist=getDisimilarity(ds.dataMatrix.row(index),ds.dataMatrix.row(i),index,i,normOption, object);
        if(tempDist<=radius_eps)
            neighborArray.push_back(i);
    }
    return neighborArray;
}
```

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

Definition at line 291 of file OPTICS.cpp.

```
{
    if(argc!=3)
    {
        std::cout << "Input argument should have 3!" << endl
                  << "./cluster inputFile_name(in dataset folder) "
                  << "data_dimension(3)" << endl;
        exit(1);
    }
    ds.strName = string("../dataset/")+string(argv[1]);
    ds.dimension = atoi(argv[2]);

    /* need to judge whether it is a PBF dataset or not */
    std::cout << "It is a PBF dataset? 1.Yes, 0.No." << std::endl;
    int PBFInput;
    std::cin >> PBFInput;
    assert(PBFInput==1||PBFInput==0);
    isPBF = (PBFInput==1);

    std::cout << "It is a pathlines dataset? 1.Yes, 0.No." << std::endl;
    std::cin >> PBFInput;
    assert(PBFInput==1||PBFInput==0);
    isPathlines = (PBFInput==1);

    int sampleOption;

    if(isPathlines)
        sampleOption = 1;
    else
    {
        std::cout << "choose a sampling method for the dataset?" << std::endl
                  << "1.directly filling with last vertex; 2. uniform sampling." << std::endl;
        std::cin >> sampleOption;
    }
    assert(sampleOption==1||sampleOption==2);

    IOHandler::readFile(ds.strName,ds.dataVec,ds.vertexCount,ds.dimension,ds.maxElements);

    ds.fullName = ds.strName+"_differentNorm_full.vtk";
    IOHandler::printVTK(ds.fullName, ds.dataVec, ds.vertexCount, ds.dimension);

    if(sampleOption==1)
        IOHandler::expandArray(ds.dataMatrix,ds.dataVec,ds.dimension,ds.maxElements);
    else if(sampleOption==2)
        IOHandler::sampleArray(ds.dataMatrix,ds.dataVec,ds.dimension,ds.maxElements);
}
```

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

Definition at line 424 of file OPTICS.cpp.

```
{
    /*std::cout << std::endl;
    std::cout << "Input the minPts for OPTICS in [0" << ", "
              << ds.dataMatrix.rows() << "], 6 is recommended:" << std::endl;
              */
    int minPts = 6;
    //std::cin >> minPts;

    if(minPts<=0 || minPts>=ds.dataMatrix.rows())
    {
        std::cout << "Error for out-of-range minPts!" << std::endl;
        exit(1);
    }
    return minPts;
}
```

4.2.3.12 void DensityClustering::setNormOption ( ) [private]

Definition at line 343 of file OPTICS.cpp.

```
{
    if(isPathlines)
    {
        std::cout << "Choose a norm from 0-17!" << std::endl;
        std::cin >> normOption;
        assert(normOption>=0 && normOption<=17);
    }
    else
    {
        std::cout << "Choose a norm from 0-15!" << std::endl;
        std::cin >> normOption;
        assert(normOption>=0 && normOption<=15);
    }
    /*  0: Euclidean Norm
        1: Fraction Distance Metric
        2: piece-wise angle average
        3: Bhattacharyya metric for rotation
        4: average rotation
        5: signed-angle intersection
        6: normal-direction multivariate distribution
        7: Bhattacharyya metric with angle to a fixed direction
        8: Piece-wise angle average \times standard deviation
        9: normal-direction multivariate un-normalized distribution
        10: x*y/|x||y| borrowed from machine learning
        11: cosine similarity
        12: Mean-of-closest point distance (MCP)
        13: Hausdorff distance min_max(x_i,y_i)
        14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
        15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
        16: entropy-based distance metric taken from http://vis.cs.ucdavis.edu/papers/pg2011paper.pdf
        17: time-series MCP distance from https://www.sciencedirect.com/science/article/pii/S0097849318300128
            for pathlines only
    */
}
```

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

Definition at line 449 of file OPTICS.cpp.

```
{
    std::cout << std::endl;
    float lowerBound = minDist/maxDist;
    std::cout << "Input the multiplication for OPTICS radius in ["
              << lowerBound << ",1.0]:" << std::endl;
    float multiTimes;
    std::cin >> multiTimes;
    if(multiTimes>=1.0 || multiTimes<=lowerBound)
    {
        std::cout << "Error for out-of-range minPts!" << std::endl;
        exit(1);
    }
    return multiTimes;
}
```

4.2.3.14 void DensityClustering::update (const int & *index*, const vector< int > & *neighbor*, LinkedList & *seeds*, const float & *radius_eps*, const int & *minPts*) [private]

Definition at line 222 of file OPTICS.cpp.

```
{
    const float& coredist = nodeVec[index].core_distance;
    for(int i=0;i<neighbor.size();++i)
    {
        if(!nodeVec[neighbor[i]].visited)
        {
            float dist_toCenter;
            if(distanceMatrix)
                dist_toCenter = distanceMatrix[neighbor[i]][index];
            else
                dist_toCenter = getDisimilarity(ds.dataMatrix.row(neighbor[i]),
                  ds.dataMatrix.row(index), neighbor[i], index, normOption, object);
            const float& biggerDist = std::max(coredist, dist_toCenter);
            if(nodeVec[neighbor[i]].reachabilityDist==-1.0)
            {
                nodeVec[neighbor[i]].reachabilityDist=biggerDist;
                seeds.insertNode(new pointNode(OrderedPoint(neighbor[i], biggerDist)));
            }
            else
            {
                float& reachDist = nodeVec[neighbor[i]].reachabilityDist;
                if(biggerDist<reachDist)
                {
                    reachDist = biggerDist;
                    seeds.updateNode(neighbor[i], reachDist);
                }
            }
        }
    }
}
```

4.2.3.15 void DensityClustering::writeReachability ( ) [private]

Definition at line 813 of file OPTICS.cpp.

```
{
    ofstream ofile("../dataset/reachability.txt", ios::out);
    if(!ofile)
    {
        std::cout << "Error creating the file!" << std::endl;
        exit(1);
    }
    for (int i = 0; i < orderedList.size(); ++i)
    {
        ofile << orderedList[i] << " " << nodeVec[orderedList[i]].reachabilityDist
        << " " << nodeVec[orderedList[i]].group << std::endl;
    }
    ofile << std::endl;
    ofile.close();
}
```

4.2.4 Member Data Documentation

4.2.4.1 DataSet DensityClustering::ds [private]

Definition at line 70 of file OPTICS.h.

4.2.4.2 bool DensityClustering::isPathlines [private]

Definition at line 80 of file OPTICS.h.

4.2.4.3 bool DensityClustering::isPBF [private]

Definition at line 75 of file OPTICS.h.

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

Definition at line 55 of file OPTICS.h.

4.2.4.5 int DensityClustering::normOption [private]

Definition at line 65 of file OPTICS.h.

4.2.4.6 MetricPreparation DensityClustering::object [private]

Definition at line 60 of file OPTICS.h.

4.2.4.7 vector< int > DensityClustering::orderedList [private]

Definition at line 50 of file OPTICS.h.

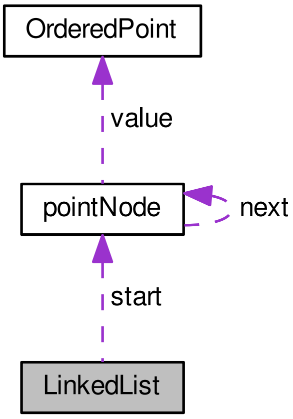
The documentation for this class was generated from the following files:

- OPTICS.h
- OPTICS.cpp

4.3 LinkedList Class Reference

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

Collaboration diagram for LinkedList:



Public Member Functions

- LinkedList ()
- LinkedList (const OrderedPoint &value)
- ∼LinkedList ()
- void insertNode (pointNode ∗vNode)
- void updateNode (const int &index, const float &newDist)

Public Attributes

- pointNode ∗ start

Private Member Functions

- void deleteNode (pointNode ∗&pNode)

4.3.1 Detailed Description

Definition at line 93 of file Predefined.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 LinkedList::LinkedList ( ) [inline]

Definition at line 98 of file Predefined.h.

```
98                    : start(NULL)
99      {}
```

4.3.2.2 LinkedList::LinkedList (const OrderedPoint & *value*) [inline]

Definition at line 101 of file Predefined.h.

```
101                                             : start(new pointNode(value))
102      {}
```

4.3.2.3 LinkedList::∼LinkedList ( ) [inline]

Definition at line 104 of file Predefined.h.

```
105      {
106          deleteNode(start);
107      }
```

4.3.3 Member Function Documentation

4.3.3.1 void LinkedList::deleteNode (pointNode *& *pNode*) [inline], [private]

Definition at line 175 of file Predefined.h.

```
{
    if(pNode==NULL)
        return;
    else if(pNode && pNode->next==NULL)
    {
        delete pNode;
        pNode = NULL;
        return;
    }
    deleteNode(pNode->next);
}
```

4.3.3.2 void LinkedList::insertNode (pointNode * *vNode*) [inline]

Definition at line 109 of file Predefined.h.

```
{
    if(start==NULL)
    {
        start = vNode;
        return;
    }
    else if(vNode->value.reachabilityDist<start->value.reachabilityDist)
    {
        vNode->next = start;
        start = vNode;
        return;
    }
    pointNode *temp = start;
    while(temp->next && temp->next->value.reachabilityDist<vNode->value.reachabilityDist)
    {
        temp=temp->next;
    }
    vNode->next = temp->next;
    temp->next = vNode;
}
```

4.3.3.3 void LinkedList::updateNode (const int & *index*, const float & *newDist*) [inline]

Definition at line 132 of file Predefined.h.

```
{
    pointNode *temp;
    if(start==NULL)
    {
        std::cout << "Linked list is empty!" << std::endl;
        return;
    }
    if(start->value.index==index)
    {
        temp = start;
        start = start->next;
        temp->next = NULL;
        temp->value.reachabilityDist = newDist;
        insertNode(temp);
        return;
    }

    pointNode *before = start;
```

```
        temp = before->next;
        while(temp)
        {
            if(temp->value.index==index)
                break;
            temp=temp->next;
            before=before->next;
        }
        if(temp==NULL)
        {
            std::cout << "Indexed node is not inside the linked list!" << std::endl;
            return;
        }
        else
        {
            temp->value.reachabilityDist = newDist;
            before->next = temp->next;
            temp->next = NULL;
            insertNode(temp);
        }
    }
```

4.3.4 Member Data Documentation

4.3.4.1 pointNode∗ LinkedList::start

Definition at line 96 of file Predefined.h.

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

- Predefined.h

4.4 OrderedPoint Struct Reference

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

Public Member Functions

- OrderedPoint (const int &index, const float &reachabilityDist)
- OrderedPoint ()

Public Attributes

- int index
- float reachabilityDist

4.4.1 Detailed Description

Definition at line 49 of file Predefined.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 OrderedPoint::OrderedPoint (const int & *index*, const float & *reachabilityDist*) [inline]

Definition at line 54 of file Predefined.h.

```
54                                                                       :
55      index(index), reachabilityDist(reachabilityDist)
56      {}
```

4.4.2.2 OrderedPoint::OrderedPoint ( ) [inline]

Definition at line 58 of file Predefined.h.

```
58                    : index(-1), reachabilityDist(-1.0)
59      {}
```

4.4.3 Member Data Documentation

4.4.3.1 int OrderedPoint::index

Definition at line 51 of file Predefined.h.

4.4.3.2 float OrderedPoint::reachabilityDist

Definition at line 52 of file Predefined.h.

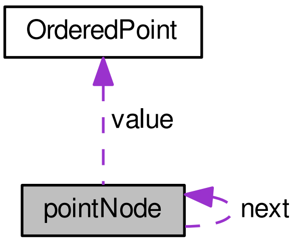
The documentation for this struct was generated from the following file:

- Predefined.h

4.5 pointNode Struct Reference

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

Collaboration diagram for pointNode:



Public Member Functions

- pointNode (const OrderedPoint &value)
- pointNode ()
- ∼pointNode ()

Public Attributes

- OrderedPoint value
- pointNode ∗ next

4.5.1 Detailed Description

Definition at line 66 of file Predefined.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 pointNode::pointNode (const **OrderedPoint** & *value*) [inline]

Definition at line 71 of file Predefined.h.

```
71                                      : value(value), next(NULL)
72     {
73     }
```

4.5.2.2 pointNode::pointNode ( ) [inline]

Definition at line 75 of file Predefined.h.

```
75                : value(OrderedPoint()), next(NULL)
76     {
77     }
```

4.5.2.3 pointNode::∼pointNode ( ) [inline]

Definition at line 79 of file Predefined.h.

```
80     {
81         if(next)
82         {
83             delete next;
84             next = NULL;
85         }
86     }
```

4.5.3 Member Data Documentation

4.5.3.1 pointNode∗ pointNode::next

Definition at line 69 of file Predefined.h.

4.5.3.2 OrderedPoint pointNode::value

Definition at line 68 of file Predefined.h.

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

- Predefined.h

4.6 PointNode Struct Reference

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

Public Member Functions

- PointNode ()
- ∼PointNode ()

Public Attributes

- int type
- bool visited
- int group
- float reachabilityDist
- float core_distance
- vector< int > neighbor

4.6.1 Detailed Description

Definition at line 30 of file Predefined.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 PointNode::PointNode ( ) [inline]

Definition at line 38 of file Predefined.h.

```
38                :type(-1), visited(false), group(-1),
      reachabilityDist(-1.0), core_distance(-1.0)
39     {}
```

4.6.2.2 PointNode::∼PointNode ( ) [inline]

Definition at line 41 of file Predefined.h.

```
42     {}
```

4.6.3 Member Data Documentation

4.6.3.1 float PointNode::core_distance

Definition at line 36 of file Predefined.h.

4.6.3.2 int PointNode::group

Definition at line 34 of file Predefined.h.

4.6.3.3 vector<int> PointNode::neighbor

Definition at line 37 of file Predefined.h.

4.6.3.4 float PointNode::reachabilityDist

Definition at line 35 of file Predefined.h.

4.6.3.5 int PointNode::type

Definition at line 32 of file Predefined.h.

4.6.3.6 bool PointNode::visited

Definition at line 33 of file Predefined.h.

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

- Predefined.h

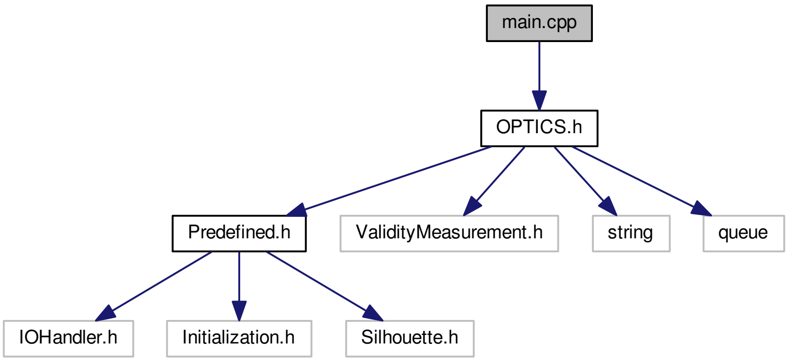
Chapter 5

File Documentation

5.1 main.cpp File Reference

`#include "OPTICS.h"`

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.1.1 Function Documentation

5.1.1.1 int main (int *argc*, char ∗∗ *argv*)

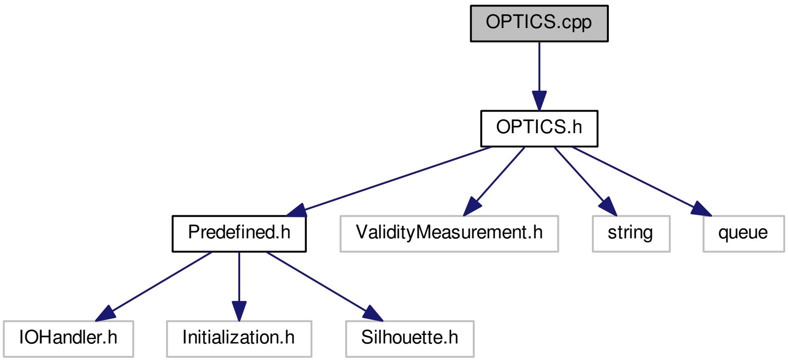
Definition at line 9 of file main.cpp.

```
10 {
11     DensityClustering dclustering(argc, argv);
12     dclustering.performClustering();
13     return 0;
14 }
```

5.2 OPTICS.cpp File Reference

```
#include "OPTICS.h"
```

Include dependency graph for OPTICS.cpp:



Variables

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

5.2.1 Variable Documentation

5.2.1.1 std::vector<string> activityList

Definition at line 11 of file OPTICS.cpp.

5.2.1.2 int minPts

Definition at line 23 of file OPTICS.cpp.

5.2.1.3 float multiTimes

Definition at line 18 of file OPTICS.cpp.

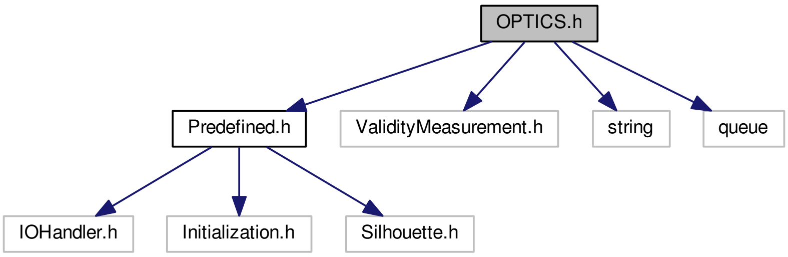
5.2.1.4 std::vector<string> timeList

Definition at line 12 of file OPTICS.cpp.

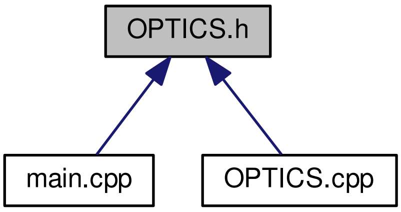
5.3 OPTICS.h File Reference

```
#include "Predefined.h"
#include "ValidityMeasurement.h"
#include <string>
#include <queue>
```

Include dependency graph for OPTICS.h:



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



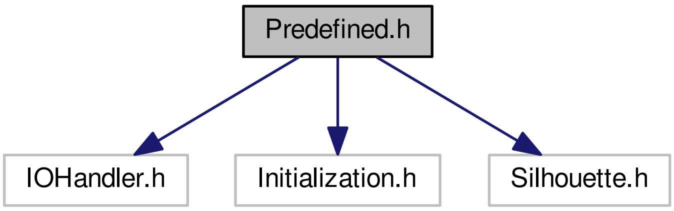
Classes

- class DensityClustering

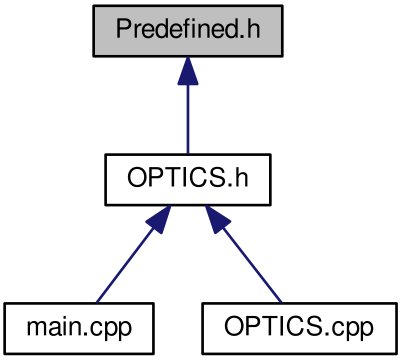
5.4 Predefined.h File Reference

```
#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 PointNode
- struct OrderedPoint
- struct pointNode
- class LinkedList
- struct DataSet

Enumerations

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

5.4.1 Enumeration Type Documentation

5.4.1.1 enum PointType

Enumerator

- ***CORE***
- ***BORDER***
- ***NOISE***

Definition at line 19 of file Predefined.h.

```
{
    CORE = 0,
    BORDER,
    NOISE
};
```

5.5 README.md File Reference

Index