ReadClustering

The C++ implmentation for ReadClustering

Generated by Doxygen 1.8.11

Contents

Chapter 1

ReadClustering

The program will read from a generated .vtk file with clustering results (i.e., geometric coordinates and cluster labels of integral lines) to re-calculate the clustering evaluation metrics of the current result in case of miscalculated result.

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 <ReadClustering.h>
```

Public Attributes

- std::vector< std::vector< float > > dataVec
- Eigen::MatrixXf array
- unordered_map< string, std::vector< int > > groupAggregate
- unordered_map< string, int > maxGroup
- int numOfElements
- std::vector< std::vector< int > > neighborVec

4.1.1 Detailed Description

Definition at line 27 of file ReadClustering.h.

4.1.2 Member Data Documentation

4.1.2.1 Eigen::MatrixXf Dataset::array

Definition at line 33 of file ReadClustering.h.

4.1.2.2 std::vector<std::vector<float> > Dataset::dataVec

Definition at line 30 of file ReadClustering.h.

4.1.2.3 unordered_map<string, std::vector<int> > Dataset::groupAggregate

Definition at line 36 of file ReadClustering.h.

4.1.2.4 unordered_map< string, int > Dataset::maxGroup

Definition at line 39 of file ReadClustering.h.

4.1.2.5 std::vector< std::vector< int > > Dataset::neighborVec

Definition at line 44 of file ReadClustering.h.

4.1.2.6 int Dataset::numOfElements

Definition at line 42 of file ReadClustering.h.

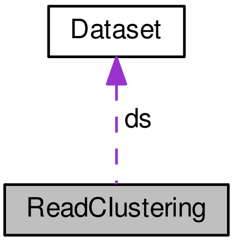
The documentation for this struct was generated from the following file:

- ReadClustering.h

4.2 ReadClustering Class Reference

```
#include <ReadClustering.h>
```

Collaboration diagram for ReadClustering:



Public Member Functions

- ReadClustering ()
- virtual ∼ReadClustering ()
- void getEvaluation (const char ∗fileName)

Private Member Functions

- void readData (const char ∗fileName)
- void computeEvaluation ()
- void writeAnalysis ()
- void computeEvaluation (std::unordered_map< string, std::vector< int > >::const_iterator &iter)
- void performSVD (MatrixXf &cArray, const Eigen::MatrixXf &data, const int &Row, const int &Column, int &PC_Number)

Private Attributes

- std::vector< string > activityList
- std::vector< string > timeList
- Dataset ds
- int maxElements
- bool isPBF

4.2.1 Detailed Description

Definition at line 52 of file ReadClustering.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ReadClustering::ReadClustering ( )

Definition at line 20 of file ReadClustering.cpp.

```
20                                  {
21      // TODO Auto-generated constructor stub
22
23 }
```

4.2.2.2 ReadClustering::∼ReadClustering ( ) [virtual]

Definition at line 29 of file ReadClustering.cpp.

```
29                                   {
30      // TODO Auto-generated destructor stub
31 }
```

4.2.3 Member Function Documentation

4.2.3.1 void ReadClustering::computeEvaluation ( ) [private]

Definition at line 257 of file ReadClustering.cpp.

```
258 {
259     std::unordered_map<string, std::vector<int> >::const_iterator iter;
260     for(iter=ds.groupAggregate.begin(); iter!=ds.
      groupAggregate.end(); ++iter)
261     {
262         std::cout << "Processing for " << iter->first << "..." << std::endl;
263         computeEvaluation(iter);
264     }
265 }
```

4.2.3.2 void ReadClustering::computeEvaluation (std::unordered_map< string, std::vector< int > >::const_iterator & *iter*) [private]

Definition at line 273 of file ReadClustering.cpp.

```
274 {
275     if(iter->second.empty())
276         return;
277 
278     ds.neighborVec.clear();
279     ds.neighborVec = std::vector<std::vector<int> >(ds.maxGroup[iter->first]);
280 
281     const std::vector<int>& groupOfNorm = iter->second;
282     const int& groupSize = groupOfNorm.size();
283 
284     Silhouette sil;
285     ValidityMeasurement vm;
286 
287     int totalNum = 0;
288     for(int i=0;i<groupSize;++i)
289     {
290         if(groupOfNorm[i]<0)
291             continue;
292         ds.neighborVec[groupOfNorm[i]].push_back(i);
293         ++totalNum;
294     }
295     std::cout << totalNum << std::endl;
296     /* the 'PCA' option */
297     if(strcmp("PCA", iter->first.c_str())==0)
298     {
299         IOHandler::expandArray(ds.array,ds.dataVec,3,maxElements);
300         std::cout << "expanded!" << std::endl;
301         Eigen::MatrixXf cArray;
302         int PC_number;
303         std::cout << ds.maxGroup[iter->first] << std::endl;
304         performSVD(cArray, ds.array, ds.array.rows(), ds.
      array.cols(), PC_number);
305         sil.computeValue(cArray,groupOfNorm,ds.maxGroup[iter->first],
      isPBF);
306         vm.computeValue(ds.array, groupOfNorm);
307     }
308     else
309     {
310         /* count from "norm" for norm option */
311         const int& normOption = std::atoi(iter->first.substr(4).c_str());
312         std::cout << "This is norm " << normOption << std::endl;
313         //if(normOption!=4 && normOption!=15)
314         //  return;
315 
316         if(normOption==17)
317             IOHandler::expandArray(ds.array,ds.dataVec,3,
      maxElements);
318         else
319             IOHandler::sampleArray(ds.array,ds.dataVec,3,
      maxElements);
320 
321         MetricPreparation object(ds.array.rows(), ds.array.cols());
322         object.preprocessing(ds.array, ds.array.rows(), ds.array.cols(), normOption);
323         /* if the dataset is not PBF, then should record distance matrix for Gamma matrix compution */
324         if(!isPBF)
325         {
326             deleteDistanceMatrix(ds.array.rows());
327 
328             std::ifstream distFile(("../dataset/"+to_string(normOption)).c_str(), ios::in);
329             if(distFile.fail())
330             {
331                 distFile.close();
332                 getDistanceMatrix(ds.array, normOption, object);
333                 std::ofstream distFileOut(("../dataset/"+to_string(normOption)).c_str(), ios::out);
334                 for(int i=0;i<ds.array.rows();++i)
335                 {
336                     for(int j=0;j<ds.array.rows();++j)
337                     {
338                         distFileOut << distanceMatrix[i][j] << " ";
339                     }
340                     distFileOut << std::endl;
341                 }
342                 distFileOut.close();
343             }
344             else
345             {
346                 std::cout << "read distance matrix..." << std::endl;
347 
```

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

            std::cout << "Distance between 0 and 1 is " << distanceMatrix[0][1] << std::endl;
        }
        sil.computeValue(normOption,ds.array,ds.array.rows(),ds.array.cols(),groupOfNorm,object,
                             ds.maxGroup[iter->first],isPBF, ds.neighborVec);
        vm.computeValue(normOption, ds.array, groupOfNorm, object, isPBF);
    }

    /* compute the entropy */
    float entropy = 0, prob;
    for (int i = 0; i < ds.neighborVec.size(); ++i)
    {
        if(ds.neighborVec[i].size()>0)
        {
            prob = float(ds.neighborVec[i].size())/float(totalNum);
            entropy+=prob*log2f(prob);
        }
    }

    entropy = -entropy/log2f(ds.maxGroup[iter->first]);
    std::cout << "Entropy is " << entropy << std::endl;

    activityList.push_back("Silhouette for "+iter->first+" is: ");
    timeList.push_back(to_string(sil.sAverage));

    activityList.push_back("Gamma statistic for "+iter->first+" is: ");
    timeList.push_back(to_string(sil.gammaStatistic));

    activityList.push_back("Entropy for "+iter->first+" is: ");
    timeList.push_back(to_string(entropy));

    activityList.push_back("DB Index for "+iter->first+" is: ");
    timeList.push_back(to_string(sil.dbIndex));

    activityList.push_back("Validity measurement on "+iter->first+" is: ");
    stringstream fc_ss;
    fc_ss << vm.f_c;
    timeList.push_back(fc_ss.str());

    /* record labeling information */
    // IOHandler::generateGroups(ds.neighborVec, iter->first+"_storage");
}
```

4.2.3.3 void ReadClustering::getEvaluation (const char * *fileName*)

Definition at line 39 of file ReadClustering.cpp.

```
{
    int isPBFInput;
    std::cout << "Is it a PBF dataset? 1.Yes, 0.No." << std::endl;
    std::cin >> isPBFInput;
    assert(isPBFInput==1||isPBFInput==0);
    isPBF = (isPBFInput==1);

    /* read data into ds */
    std::cout << fileName << std::endl;
    readData(fileName);

    /* compute the evaluation */
    computeEvaluation();

    /* output the result to text file */
    writeAnalysis();
}
```

4.2.3.4 void ReadClustering::performSVD (MatrixXf & *cArray,* const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column,* int & *PC_Number*) [private]

Definition at line 428 of file ReadClustering.cpp.

```
{
    MatrixXf SingVec;
    VectorXf meanTrajectory(Column);
    Eigen::MatrixXf temp = data;

#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Column; ++i)
    {
        meanTrajectory(i) = temp.transpose().row(i).mean();
    }
#pragma omp parallel for schedule(static) num_threads(8)
    for (int i = 0; i < Row; ++i)
    {
        temp.row(i) = temp.row(i)-meanTrajectory.transpose();
    }
    /* perform SVD decomposition for temp */
    JacobiSVD<MatrixXf> svd(temp, ComputeThinU | ComputeThinV);
    //const VectorXf& singValue = svd.singularValues();
    SingVec = svd.matrixV();

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

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

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

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

    SingVec.transposeInPlace();
}
```

4.2.3.5 void ReadClustering::readData (const char ∗ *fileName*) [private]

Definition at line 75 of file ReadClustering.cpp.

```
{
    ifstream fin(fileName, ios::in);
    if(!fin)
    {
        std::cout << "Error opening the file!" << std::endl;
        exit(1);
    }

    Eigen::MatrixXf vertexCoordinate;

    /* omit first four lines */
    string line;
    for(int i=0;i<5;++i)
    {
        getline(fin, line);
    }
    /* split the string into three parts */
    stringstream ss(line);
    ss >> line;
    ss >> line;

    ss.str(std::string());

    /* get how many vertex inside */
    const int& vertexCount = atoi(line.c_str());

    vertexCoordinate = Eigen::MatrixXf(vertexCount, 3);

    /* read in vertex coordinates */
    for(int i=0;i<vertexCount;++i)
    {
        /* read one line */
        getline(fin, line);
        /* split and analyze the string */
        ss.str(line);
        for(int j=0;j<3;++j)
        {
            ss >> line;
            vertexCoordinate(i,j) = atof(line.c_str());
        }
        ss.str(std::string());
    }

    /* get how many streamlines you'll have */
    getline(fin, line);

    ss.str(line);
    ss >> line;
    ss >> line;

    ss.str(std::string());

    ds.numOfElements = atoi(line.c_str());

    ds.dataVec = std::vector<std::vector<float> >(ds.numOfElements);

    /* read vertex coordinates into dataVec */

    maxElements = INT_MIN;

    int vertexNum, index;
    for(int i=0;i<ds.numOfElements;++i)
    {
        getline(fin,line);
        ss.str(line);

        /* explicate vertex count in each line */
        ss>>line;
        vertexNum = atoi(line.c_str());

        /* assign memory */
        std::vector<float>& tempVec = ds.dataVec[i];

        tempVec = std::vector<float>(vertexNum*3);

        maxElements = std::max(maxElements, vertexNum*3);

        for(int j=0;j<vertexNum;++j)
        {
            ss>>line;
            index = atoi(line.c_str());
            for(int k=0;k<3;++k)
                tempVec[3*j+k] = vertexCoordinate(index,k);
        }

        ss.str(std::string());
    }
    for(int i=0;i<3;++i)
    {
        getline(fin,line);
    }
    for(int i=0;i<vertexCount;++i)
    {
        getline(fin,line);
    }
    std::size_t found_int, found, found_scalars;

    int normOption, totalLine, groupIndex;
    while(getline(fin,line))
    {
        found_int = line.find("int");
        found_scalars = line.find("SCALARS");
        /* has int, should be group information */
        if(found_int!=std::string::npos && found_scalars!=std::string::npos)
        {
            ss.str(line);
            ss>>line;
            ss>>line;

            string norm_choice;
            if(strcmp(line.substr(0,3).c_str(), "PCA")==0)
                norm_choice="PCA";
            else
            {
                found = line.find("_");
                found_int = line.find("norm");
                if(found==std::string::npos)
                {
                    norm_choice = line;
                }
                else if(found_int>found)
                {
                    norm_choice = line.substr(found_int);
                }
                else if(found_int<found)
                {
                    norm_choice = line.substr(found_int, found);
                }
            }

            getline(fin,line);

            std::vector<int> tempGroup(ds.numOfElements);

            for(int i=0;i<ds.numOfElements;++i)
            {
                for(int j=0;j<ds.dataVec[i].size()/3;++j)
                    getline(fin,line);
                tempGroup[i] = atoi(line.c_str());
            }
            ds.groupAggregate.insert(std::make_pair(norm_choice, tempGroup));

            totalLine = vertexCount+2;
            for(int i=0; i<totalLine; ++i)
            {
                getline(fin,line);
            }
        }
    }

    fin.close();

    /* compute cluster number */

    std::vector<int> groupArray;
    int max_num;
    std::unordered_map<string, std::vector<int> >::const_iterator iter;
    for(iter=ds.groupAggregate.begin(); iter!=ds.groupAggregate.end(); ++iter)
    {
        groupArray = iter->second;
        max_num = -1;
        if(groupArray.empty())
            continue;

        index = groupArray.size();
        for(int j=0;j<index;++j)
        {
            max_num = std::max(max_num, groupArray[j]);
        }
        max_num+=1;
        ds.maxGroup.insert(make_pair(iter->first, max_num));
    }

```

```
}
```

4.2.3.6 void ReadClustering::writeAnalysis ( ) [private]

Definition at line 62 of file ReadClustering.cpp.

```
{
    /* write information */
    IOHandler::generateReadme(activityList,timeList);

}
```

4.2.4 Member Data Documentation

4.2.4.1 std::vector<string> ReadClustering::activityList [private]

Definition at line 79 of file ReadClustering.h.

4.2.4.2 Dataset ReadClustering::ds [private]

Definition at line 89 of file ReadClustering.h.

4.2.4.3 bool ReadClustering::isPBF [private]

Definition at line 99 of file ReadClustering.h.

4.2.4.4 int ReadClustering::maxElements [private]

Definition at line 94 of file ReadClustering.h.

4.2.4.5 std::vector<string> ReadClustering::timeList [private]

Definition at line 84 of file ReadClustering.h.

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

- ReadClustering.h
- ReadClustering.cpp

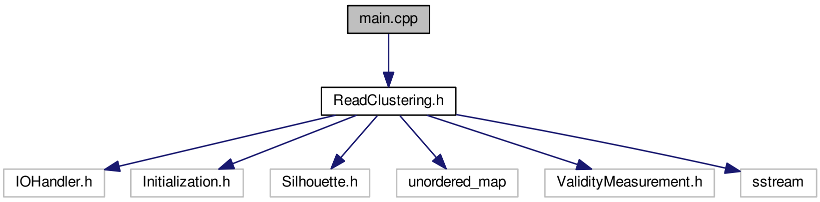
Chapter 5

File Documentation

5.1 main.cpp File Reference

`#include "ReadClustering.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[ ]*)

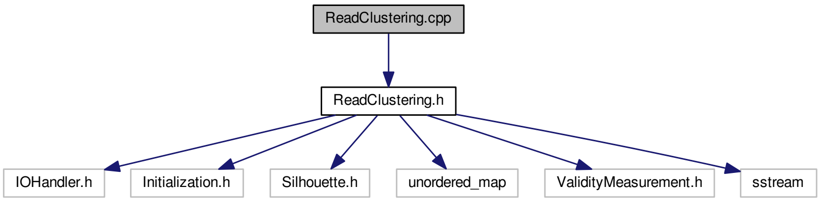
Definition at line 10 of file main.cpp.

```
{
    if(argc!=2)
    {
        std::cout << "Error for argument input!" << std::endl;
        exit(1);
    }

    /* create ReadClustering object and get evaluation */

    ReadClustering rc;

    rc.getEvaluation(argv[1]);

    return 0;
}
```

5.2 ReadClustering.cpp File Reference

```
#include "ReadClustering.h"
```

Include dependency graph for ReadClustering.cpp:



Variables

- const float & TOR_1 = 0.999

5.2.1 Variable Documentation

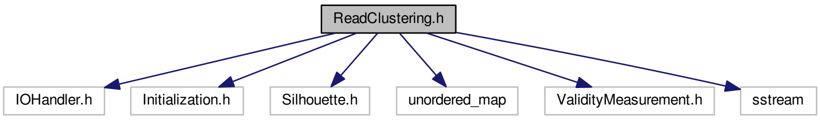
5.2.1.1 const float& TOR_1 = 0.999

Definition at line 14 of file ReadClustering.cpp.

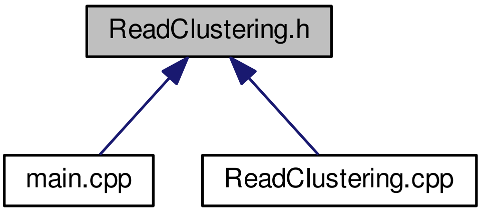
5.3 ReadClustering.h File Reference

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

Include dependency graph for ReadClustering.h:



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



Classes

- struct Dataset
- class ReadClustering

5.4 README.md File Reference

Index