ПРОГРАММА ДЛЯ ЭВМ

Программа кластеризации и обобщения данных для ассоциативной памяти робота АР-600

Фрагменты исходного текста программы

Правообладатель-автор: *Ширкин Александр Евгеньевич*

2018 г.

Листинг файла cluster.h

```
#pragma once
#include <list>
#include <memory>

#define vertex shared_ptr<Neuron>
#define neuronIterator list<vertex>::iterator


using namespace std;

class Neuron;

class Cluster
{

	private:

		int id;
		double meanDensity;

	public:
	Cluster(vertex delegatorOfCluster, int clusterId);
		~Cluster();
	list<vertex> neuronsList;
		int getId();
		void setId(int data);
		double getDensity();
	vertex findApex();
	vertex getApex();
	vertex setApex(vertex data);
		double calcMeanDensity();

	void setDensity(double density);
	vertex apex;
};
```

Листинг файла cluster.cpp

```
#include "neuron.h"
#include "cluster.h"

Cluster::Cluster(vertex delegatorOfCluster, int clusterId)
{
   apex = delegatorOfCluster;
   neuronsList.push_back(apex);
   id = clusterId;
}

Cluster::~Cluster()
{
   //for(auto it = neuronsList->begin(); it != neuronsList->end(); ++it) delete (*it);
   //delete neuronsList;
   //delete apex;
}

int Cluster::getId()
{
	return id;
}

void Cluster::setId(int data)
{
	id = data;
}

double Cluster::getDensity()
{
	return meanDensity;
}

vertex Cluster::getApex()
{
	return apex;
}

vertex Cluster::setApex(vertex data)
{
   apex = data;
   return apex;
}

double Cluster::calcMeanDensity()
{
   auto res = 0.0;
   for(auto &it : neuronsList)
	{
      res += it->getDensity();
   }
   res /= neuronsList.size();
	meanDensity = res;
	return meanDensity;
}

void Cluster::setDensity(double dens)
{
   meanDensity = dens;
}

vertex Cluster::findApex()
```

```
{
   if (neuronsList.size())
   {
      if (!apex) apex = *(neuronsList.begin());
      for (auto &it : neuronsList)
      {
         if (it->getDensity() > apex->getDensity()) apex = it;
      }
   }
         return apex;
}
```

Листинг файла connection.h

```
#pragma once

#include <memory>
#define vertex shared_ptr<Neuron>


//Also can be called as edge

using namespace std;

class Neuron;

class Connection
{

	private:

		int age;

	public:

	Connection(vertex first, vertex second);
		~Connection();
	vertex first;
	vertex second;
	void incAge();
	vertex getNeighbourNeuron(vertex node);
	void setAge(int age);
	int getAge();
};
```

Листинг файла connection.cpp

```
#include "connection.h"
#include "neuron.h"

Connection::Connection(vertex first, vertex second)
{
   this->first = first;
   this->second = second;
   //count++;
   //id = count;
   age = 0;
}

Connection::~Connection()
{
   //delete first;
   //delete second;
}
void Connection::incAge()
{
	age++;
}

void Connection::setAge(int age)
{
	this->age = age;
}

//int Connection::getId(){
//	return id;
//}

int Connection::getAge()
{
	return age;
}

vertex Connection::getNeighbourNeuron(vertex node)
{
	return first == node ? second : first;
}
```

Листинг файла neuron.h

```
#pragma once
#include <list>
#include <memory>
#include <time.h>
#define weight unique_ptr<double[]>
#define vertex shared_ptr<Neuron>
#define edge shared_ptr<Connection>
#define cluster shared_ptr<Cluster>

#define neuronIterator list<vertex>::iterator
#define edgeIterator list<edge>::iterator

class Cluster;
class Connection;


using namespace std;


class Neuron
{

	private:

		double density;
	int dimentionSize;
	int classId;
    bool classified;
	int winerTimesCount;
    cluster area;

  public:

    Neuron(int dimentionSize, double* weights);
    Neuron(vertex v);
		~Neuron();

    double similarityThreshold;
    double *weights;
    list<edge> neighboursList;
    double point;

    bool winInThisIter;
    int allTimeWin;

    void incSignal();
    cluster getCluster();
	double getDensity();
		int getDim();
    int getId();
    int getCountSignals();
    void setDensity(double data);
	void setId(int data);
    void setStatus(bool flag);
    bool isClassified();

    string getNeuronData();
    void setNeuronData(double simTh, double Point, bool winned, int allwins, double dens, int classid, int winCnt);
    cluster setArea(cluster buf);

    cluster setCluster(cluster buf);
};
```

Листинг файла neuron.cpp

```
#include "neuron.h"
#include "cluster.h"
#include "connection.h"
#include <iostream>
#include <string>


Neuron::Neuron(int dimentionSize, double * weights = NULL)
{
	srand((unsigned)time(NULL));
	this->dimentionSize = dimentionSize;
	this->weights = new double[dimentionSize];
	for (int i = 0; i < dimentionSize; i++)
			{
	   //TODO: in what borders random values?
	   if (weights == NULL)
	      this->weights[i] = rand() % 1000;
	   else this->weights[i] = weights[i];
	}
	this->area = nullptr;
	this->classId = -1;
	this->density = 0;
	this->winerTimesCount = 0;
	this->point = 0;
	this->winInThisIter = false;
	this->allTimeWin = 0;
	this->classified = false;
}

Neuron::Neuron(vertex v)
{
   srand((unsigned)time(NULL));
   this->dimentionSize = v->getDim();
   this->weights = new double[dimentionSize];
   for (int i = 0; i < dimentionSize; i++)
		{
	   //TODO: in what borders random values?
	   if (v->weights == NULL)
	   this->weights[i] = rand() % 1000;
	   else this->weights[i] = v->weights[i];
   }
   this->area = v->getCluster();
   this->classId = v->getId();
   this->density = v->getDensity();
   this->winerTimesCount = v->getCountSignals();
   this->point = 0;
   this->winInThisIter = false;
   this->allTimeWin = 0;
   this->classified = false;
}

Neuron::~Neuron()
{
   //for(auto it = neighboursList->begin(); it != neighboursList->end(); ++it) delete (*it);
   //delete area;
   //cout << "deleted" << endl;
   delete[] weights;
}

void Neuron::incSignal()
{
```

```
            winerTimesCount++;
    if (!winInThisIter){
        winInThisIter = true;
        allTimeWin++;
    }
}

cluster Neuron::setArea(cluster buf)
{
    this->area = buf;
    this->classId = area->getId();
    classified = true;
    return buf;
}

string Neuron::getNeuronData(){
    return to_string(similarityThreshold) + " " + to_string(point) + " " + to_string(winInThisIter) + to_string(allTimeWin) +
to_string(density) + to_string(classId) + to_string(winerTimesCount);
}

void Neuron::setNeuronData(double simTh, double Point, bool winned, int allwins, double dens, int classid, int winCnt){
    similarityThreshold = simTh;
    point = Point;
    winInThisIter = winned;
    allTimeWin = allwins;
    density = dens;
    classId = classid;
    winerTimesCount = winCnt;
}

void Neuron::setId(int data)
{
            classId = data;
}

void Neuron::setDensity(double data)
{
    this->density = data;
}

cluster Neuron::setCluster(cluster buf)
{
            this->area = buf;
    return buf;
}

void Neuron::setStatus(bool flag)
{
    classified = flag;
}

double Neuron::getDensity()
{
            return density;
}

int Neuron::getId()
{
            return classId;
}

int Neuron::getCountSignals()
{
```

```
	return winerTimesCount;
}

int Neuron::getDim()
{
	return dimentionSize;
}

cluster Neuron::getCluster()
{
	return area;
}

bool Neuron::isClassified()
{
   return classified;
}
```

Листинг файла esoinn.h

```
#pragma once

#include <list>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <stdio.h>
#include <memory>
#include <QDebug>

#define INF 1e15
#define weight unique_ptr<double[]>
#define vertex shared_ptr<Neuron>
#define edge shared_ptr<Connection>
#define cluster shared_ptr<Cluster>

#define neuronIterator list<vertex>::iterator
#define  edgeIterator list<edge>::iterator
#define clusterIterator list<cluster>::iterator



using namespace std;



class Cluster;
class Neuron;
class Connection;

class Esoinn
{

	private:

		//variables
	int dimensionSize;
	int maximalConnectionAge;
	int LT;
	int lambda;
	double c1;
	double c2;
	list<vertex> neuronsList;
	list<edge> connectionsList;
	list<cluster> clustersList;
	int clustersId;

		//methods
	double commonDistanceFunction(double * inputVector, double * checkDistanceVector);

		//TODO another params?


	vertex addNeuron(double * weights, double threshold);
	/*+*/vertex addNeuron(double *weights);
	vertex addNeuron(vertex neuronToAdd);
	/*+*/edge addConnection(vertex first, vertex second);
	cluster addCluster(vertex delegatorOfCluster);
	/*+*/void removeConnection(edge Edge);
	void removeConnection(edgeIterator &edgeToremove);
	/*+*/void removeConnection(vertex first, vertex second);
	/*+*/void removeNeuron(vertex neuronToRemove);
```

```
    /*+*/void removeNeuron(neuronIterator &neuronToRemove);
    /*+*/edge getConnection(vertex first, vertex second);

                //TODO another params?
    double calcDistance(double *weight1, double *weight2);
    double calcMeanDistance(vertex neuron);

    /*?*/bool findWiner(double * inputVector, vertex &winner, vertex &secondWinner);
    /*?*/bool needAddConnection(vertex first, vertex second);
    /*?*/bool needUniteClusters(vertex first, vertex second);
    /*?*/void uniteClusters(vertex a, vertex b);
    /*?*/double densityThreshold(double mean, double max);
    /*?*/double similarityThreshold(vertex neuron);
    /*?*/bool isWithinThreshold(vertex firstWinner, vertex secondWinner, double* inputSignal);
    /*?*/void updateDensity(vertex winner);
    /*?*/void adaptWeights(vertex &winner, double* inputVector);
    /*?*/void removeOldConnections();



    double calcPoint(vertex neuron);
    double calcEuclidNorm(double * vector1, double * vector2, int n);
    double calcHemmingNorm(double * vector1, double * vector2, int n);
    double externalCalcDistance(double * weight1, double * weight2);



                //TODO: params and implementation
                void updateClassLabels();
    void markClasses();
                void separateToSubclasses();
                void removeNoise();

    vertex getNeuron(int neuronIndex);
    edge getConnection(int connectionIndex);

    int getNeuronId(vertex neuron);
    vertex getNeuronById(int id);


  public:

                //Constructor for ESOINN
    //dimensionSize means the size of learning vectors
    //distanceFunction can not be set, it means the function, that calculate distance between vectors


                Esoinn(int dimensionSize, int maximalConnectionAge, int lambda, double c1, double c2, double
(*distanceFunction)(double *,double *));
                /*+*/Esoinn(int dimensionSize, int maximalConnectionAge, int lambda, double c1, double c2);
    Esoinn(string fileName);
                ~Esoinn();
                //method for input learning vectors as double values



    void inputSignal(double *inputVector);
    //double* interpolation(int* pixels, int w1, int h1, int w2, int h2);
    //void inputSignal(Neuron * inputVector);
                /*+*/void writeStructureToFile(string fileName);
                double ** getStructure();
    string getCurrentParams();
    double ** getTopVectors(int cnt);
```

```
		//returns main neuron that represent this input vector
	void clearWinners();
		//int neuronClassId(double * inputVector);

	void saveStateToFile(string fileName);
	void loadStateFromFile(string fileName);

	vertex getPattern(double * inputVector);
	vertex getNeuronPatern(vertex n);

		//bool saveNetworkData(string fileName);
	//bool loadNetworkData(string fileName);

};
```

Листинг файла esoinn.cpp

```
#include "neuron.h"
#include "connection.h"
#include "esoinn.h"
#include "cluster.h"

#include <iostream>
#include <fstream>
#include <QTime>
//#include "ESOINNLibSources/hasharray.h"


Esoinn::Esoinn(int dimensionSize, int maximalConnectionAge, int lambda, double c1, double c2, double
(*distanceFunction)(double *,double *))//= &commonDistanceFunction
{
   this->dimensionSize = dimensionSize;
   this->maximalConnectionAge = maximalConnectionAge;
   this->lambda = lambda;
   this->c1 = c1;
   this->c2 = c2;
   this->LT = 0;
   clustersId = 0;
   //this->externalCalcDistance = distanceFunction;
}

Esoinn::Esoinn(int dimensionSize, int maximalConnectionAge, int lambda, double c1, double c2)//=
&commonDistanceFunction
{
        this->dimensionSize = dimensionSize;
   this->maximalConnectionAge = maximalConnectionAge;
   this->lambda = lambda;
   this->c1 = c1;
   this->c2 = c2;
   this->LT = 0;
   clustersId = 0;
}

Esoinn::Esoinn(string fileName){
   clustersId = 0;
   LT = 0;
   loadStateFromFile(fileName);
}


Esoinn::~Esoinn()
{
   for (auto &it: neuronsList)
      removeNeuron(it);
   neuronsList.clear();
   connectionsList.clear();
   clustersList.clear();
}

vertex Esoinn::addNeuron(double *weights)
{
   vertex neuron = vertex(new Neuron(dimensionSize, weights));
   //vertex neuron = make_shared<Neuron>(Neuron(dimensionSize, weights));
   //cout << neuron->weights[0]  << endl;
   neuronsList.push_back(neuron);
   return neuron;
}

vertex Esoinn::addNeuron(double * weights, double threshold)
```

```
{
   auto neuron = make_shared<Neuron>(Neuron(dimensionSize, weights));
   neuronsList.push_back(neuron);
   neuron->similarityThreshold = threshold;
   return neuron;
}

vertex Esoinn::addNeuron(vertex neuronToAdd)
{
   neuronsList.push_back(neuronToAdd);
          return neuronToAdd;
}

void Esoinn::removeNeuron(vertex neuronToRemove)
{
   //cout << "BEG " <<  neuronToRemove.use_count() << " ";
   for(auto it = connectionsList.begin(); it != connectionsList.end();)
          {
                 if(((*it)->first == neuronToRemove) || ((*it)->second == neuronToRemove))
                 {
          //cout << "A";
          vertex neigh = (*it)->getNeighbourNeuron(neuronToRemove);
          neigh->neighboursList.remove(*it);
          neuronToRemove->neighboursList.remove(*it);
          (*it)->first = nullptr;
          (*it)->second = nullptr;
          it = connectionsList.erase(it);
                 }
                 else ++it;
           }

   cluster buf;
   for(auto &it : clustersList)
          {
     int size1, size2;
     size1 = it->neuronsList.size();
     it->neuronsList.remove(neuronToRemove);
     size2 = it->neuronsList.size();
     if(abs(size2 - size1))
     {
         buf = it;
         if (it->getApex() == neuronToRemove)
         {

            it->setApex(nullptr);
            if(it->findApex())
            {
               it->setId(it->getApex()->getId());
            }
            else
            {
               it->setId(-1);
            }


         }
         break;
      }
  }
   if(!buf->neuronsList.size()) clustersList.remove(buf);
   neuronToRemove->neighboursList.clear();
   neuronToRemove->setCluster(nullptr);
   //cout << neuronToRemove.use_count() << endl;
```

```
}

cluster Esoinn::addCluster(vertex delegatorOfCluster)
{
   clustersList.push_back(make_shared<Cluster>(Cluster(delegatorOfCluster, ++clustersId)));
   return clustersList.back();
}

edge Esoinn::addConnection(vertex first, vertex second)
{
   auto connection = make_shared<Connection>(Connection(first, second));
   connectionsList.push_back(connection);
   first->neighboursList.push_back(connection);
   second->neighboursList.push_back(connection);
          return connection;
}

void Esoinn::removeConnection(edge d)
{
   d->first->neighboursList.remove(d);
   d->second->neighboursList.remove(d);
   d->first = nullptr;
   d->second = nullptr;
}

/*void Esoinn::removeConnection(vertex first, vertex second)
{
   edge d;
   for(auto it = connectionsList.begin(); it != connectionsList.end();)
          {
      d = (*it);
      if(((d->first == first) && (d->second == second)) || ((d->first == second) && (d->second == first)))
                 {
         it = connectionsList.erase(it);
         for(auto j = d->first->neighboursList.begin(); j != d->first->neighboursList.end();)
                          {
            if ((*j) == d)
                                   {
               j = d->first->neighboursList.erase(j);
                                          break;
                                   }
                                   else ++j;
                          }
         for(auto j = d->second->neighboursList.begin(); j != d->second->neighboursList.end();)
                          {
            if((*j) == d)
                                   {
               j = d->second->neighboursList.erase(j);
                                          break;
                                   }
                                   else ++j;
                          }
                          break;
                 }
                 else ++it;

          }
}*/

edge Esoinn::getConnection(vertex first, vertex second)
{
   for (auto &it : connectionsList)
          {
```

```
        if(((it->first == first) && (it->second == second)) || ((it->first == second) && (it->second == first)))
                        {
            return it;
                }
            }
    return nullptr;
}

double Esoinn::calcEuclidNorm(double * vector1, double * vector2, int n)
{
    auto res = 0.0;
    for(int i = 0; i < n; i++)
            {
        res += pow(vector1[i] - vector2[i], 2);
        //if (vector1[i] != 0)
        //    cout << "v1=" << vector1[i] << " ";
        //if (vector2[i] != 0)
        //    cout << "v2=" << vector2[i] << " ";
            }
    //if (res != 0)
    //    cout << "res1=" << res << endl;
            res = sqrt(res);
    //if (res != 0)
    //    cout << "res2=" << res << endl;
            return res;
}

double Esoinn::calcHemmingNorm(double * vector1, double * vector2, int n)
{
    auto res = 0.0;
    for(int i = 0; i < n; ++i)
            {
                    res += (int)vector1[i] != (int)vector2[i] ? 1 : 0;
            }
            return res;
}

double Esoinn::calcDistance(double * a, double * b)
{
    return calcEuclidNorm(a, b, dimensionSize);
}

double Esoinn::calcMeanDistance(vertex neuron)
{
    auto res = 0.0;
    //if (neuron->getId() == -1){
        /*double min = INF;
        for (list<Neuron*>::iterator it = neuronsList->begin(); it != neuronsList->end(); ++it){
            if ((*it) != neuron){
                double temp = calcDistance(neuron->weights, (*it)->weights);
                if (temp < min)
                    min = temp;
            }
        }
        return min;*/
    //    return res;
    // }
    vertex tmp;
    for (auto &it : neuron->neighboursList)
            {
        tmp = it->getNeighbourNeuron(neuron);
        res += calcDistance(neuron->weights, tmp->weights);
    }
```

```
    if (!neuron->neighboursList.size()) res = 0;
    else res /= neuron->neighboursList.size();
            return res;
}

double Esoinn::calcPoint(vertex neuron)
{
            return 1.0 / pow(1 + calcMeanDistance(neuron), 2);
}

bool Esoinn::findWiner(double *inputVector, vertex &firstWinner, vertex &secondWinner)
{
    firstWinner = nullptr, secondWinner = nullptr;
    double dist, firstMinDist = INF, secondMinDist = INF;
    int neuronsCnt = (int)neuronsList.size();
    if (neuronsCnt < 2) return false;
    for(auto &it : neuronsList)
            {
        dist = calcDistance(it->weights, inputVector);//error here

        if(dist < firstMinDist)
                        {
                                secondWinner = firstWinner;
            secondMinDist = firstMinDist;
            firstMinDist = dist;
            firstWinner = it;
        }
        else
        if (dist < secondMinDist)
                        {
            secondMinDist = dist;
            secondWinner = it;
        }
    }
    return true;
}


double Esoinn::similarityThreshold(vertex neuron)
{
    auto dist = 0.0;
    if(!neuron->neighboursList.size())
            {
        dist = INF;
        for(auto &it : neuronsList)
                        {
            if(it != neuron)
                                {
                auto distCurrent = calcDistance(neuron->weights, it->weights);
                dist = min(dist, distCurrent);
            }
        }
    }
            else
            {
        dist = -INF;
        for(auto &it : neuron->neighboursList)
                        {
            auto distCurrent = calcDistance(neuron->weights, it->getNeighbourNeuron(neuron)->weights);
            dist =  max(dist, distCurrent);
        }
    }
    return dist;
```

```
}

bool Esoinn::isWithinThreshold(vertex firstWinner, vertex secondWinner, double *inputVector)
{
   if(calcDistance(inputVector, firstWinner->weights) > similarityThreshold(firstWinner))
           {
      return false;
   }
   if(calcDistance(inputVector, secondWinner->weights) > similarityThreshold(secondWinner))
           {
      return false;
   }
   return true;
}


double Esoinn::densityThreshold(double mean, double max)
{
   double threshold;
   if(2.0 * mean >= max)
           {
           threshold = 0.0;
           }
           else if(3.0 * mean >= max && max > 2.0 * mean)
           {
                   threshold = 0.5;
           }
   else threshold = 1.0;
   return threshold;
}

bool Esoinn::needUniteClusters(vertex first, vertex second)
{
   auto A = first->getCluster();
   auto B = second->getCluster();
   auto meanA = A->calcMeanDensity();
   auto meanB = B->calcMeanDensity();

   //qDebug() << "B";
   auto thresholdA = densityThreshold(meanA, A->findApex()->getDensity());
   auto thresholdB = densityThreshold(meanB, B->findApex()->getDensity());
   auto minAB = min(first->getDensity(), second->getDensity());
   //qDebug() << meanA << meanB << thresholdA << thresholdB << minAB;
   //if ((minAB > thresholdA * A->getApex()->getDensity() && minAB > thresholdB * B->getApex()->getDensity()) ==
true)
    //   qDebug() << "TRUE";
   return (minAB > thresholdA * A->getApex()->getDensity() && minAB > thresholdB * B->getApex()->getDensity());
}

void Esoinn::uniteClusters(vertex a, vertex b)
{
   auto A = a->getCluster();
   auto B = b->getCluster();

   for(auto &it : B->neuronsList)
   {
      A->neuronsList.push_back(it);
      it->setCluster(A);
      it->setId(A->getId());

   }
   B->neuronsList.clear();
   B->setApex(nullptr);
```

```
    clustersList.remove(B);

}

bool Esoinn::needAddConnection(vertex first, vertex second)
{
    if (!first->isClassified() || !second->isClassified()) return true;
    if (first->isClassified() && second->isClassified())
    {
        if (first->getId() == second->getId()) return true;
        if (first->getId() != second->getId() && needUniteClusters(first, second)) return true;
    }


            return false;
}

void Esoinn::updateDensity(vertex winner)
{
    winner->point += calcPoint(winner);//+= ?
    winner->incSignal();
    double density = winner->point / winner->allTimeWin;
    //qDebug() << winner->getDensity() << density;
    winner->setDensity(density);
    //qDebug() << winner->getDensity();
}

void Esoinn::adaptWeights(vertex &winner, double* inputVector)
{
            double e1 = 1.0 / winner->getCountSignals();
    double e2 = 1.0 / (100 * winner->getCountSignals());
    for(int i = 0; i < winner->getDim(); i++)
            {
        //qDebug() << "BEFORE";
        //qDebug() << winner->weights[i];
        winner->weights[i] += e1 * (inputVector[i] - winner->weights[i]);
        //qDebug() << winner->weights[i];
    }
    for(auto &it : winner->neighboursList)
            {
        for(int i = 0; i < winner->getDim(); i++)
                        {
            it->getNeighbourNeuron(winner)->weights[i] += e2 * (inputVector[i] - it->getNeighbourNeuron(winner)-
>weights[i]);
        }
    }
}

void Esoinn::removeOldConnections()
{
    for (auto it = connectionsList.begin(); it != connectionsList.end();)
            {
        if ((*it)->getAge() > maximalConnectionAge)
                        {
            removeConnection(*it);
            it = connectionsList.erase(it);
        }
        else ++it;
    }
}

void path(vertex top, cluster bag)
{
```

```
    top->setCluster(bag);
    top->setStatus(true);
    top->setId(bag->getId());
    if(top != bag->getApex()) bag->neuronsList.push_back(top);
    for(auto &it : top->neighboursList)
            {
        if(!it->getNeighbourNeuron(top)->isClassified() && it->getNeighbourNeuron(top)->getDensity() < top-
>getDensity())//?
                    {
            path(it->getNeighbourNeuron(top), bag);
                    }
            }
}

bool cmp_density(vertex a, vertex b)
{
            return a->getDensity() > b->getDensity();
}

void Esoinn::markClasses()
{
    list<vertex> vertexQueue;
    for(auto &it : neuronsList)
            {
        it->setStatus(false);
        it->setCluster(nullptr);
        it->setId(-1);
        vertexQueue.push_back(it);
            }
    for(auto &it : clustersList)
    {
        it->neuronsList.clear();
        it->setApex(nullptr);
    }
    clustersList.clear();
    //clustersId = 0;
            vertexQueue.sort(cmp_density);

    for(auto &it : vertexQueue)
            {
        if(!it->isClassified()) path(it, addCluster(it));
            }
}

void Esoinn::separateToSubclasses()
{
    vertex a, b;
    for(auto it = connectionsList.begin(); it != connectionsList.end();)
            {
        bool need_iter_inc = true;
                    a = (*it)->first, b = (*it)->second;
                    if(a->getId() != b->getId())
                    {
            if (needUniteClusters(a, b))
            {
                uniteClusters(a, b);
            }
            else
            {
                if (getConnection(a, b)) // можно искать edge
                {
                    removeConnection(*it);
                    it = connectionsList.erase(it);
```

```
                need_iter_inc = false;
            }
        }
                    }
    if (need_iter_inc) it++;
        }
}

void Esoinn::removeNoise()
{
    auto meanDensityA = 0.0;
    for (auto &it : neuronsList) meanDensityA += it->getDensity();
    meanDensityA /= neuronsList.size();
    /*int win_cnt_temp = 0;
    for (list<Neuron*>::iterator it = neuronsList->begin(); it != neuronsList->end(); it++){
        if ((*it)->getCountSignals() > 0)
            win_cnt_temp++;
    }
    qDebug() << "WIN:" << win_cnt_temp << (this->LT % this->lambda);*/

    //meanDensityA /= 2;
    //qDebug() << "SIZEb: " << neuronsList->size();

    for(auto it = neuronsList.begin(); it != neuronsList.end();)
            {
        if(((*it)->neighboursList.size() == 2) && ((*it)->getDensity() < c1 * meanDensityA))
                    {
            //qDebug() << (*it)->getCountSignals() << (*it)->getDensity();
            //qDebug() << c1 * meanDensityA << " " << (*it)->getDensity() << (*it)->getCountSignals();
            removeNeuron(*it);
            it = neuronsList.erase(it);
                    }
        else if(((*it)->neighboursList.size() == 1) && ((*it)->getDensity() < c2 * meanDensityA))
                    {

            //qDebug() << (*it)->getCountSignals() << (*it)->getDensity();
            //qDebug() << c2 * meanDensityA << " " << (*it)->getDensity()<< (*it)->getCountSignals();
            removeNeuron(*it);
            it = neuronsList.erase(it);
                    }

        else if((*it)->neighboursList.size() == 0)
        {
            //qDebug() << (*it)->getCountSignals();
            removeNeuron(*it);
            it = neuronsList.erase(it);
                    }
                    else ++it;
    }

    /*win_cnt_temp = 0;
    for (list<Neuron*>::iterator it = neuronsList->begin(); it != neuronsList->end(); it++){
        if ((*it)->getCountSignals() > 0)
            win_cnt_temp++;
    }
    qDebug() << "WIN:" << win_cnt_temp << (this->LT % this->lambda);*/
    //qDebug() << "SIZEe: " << neuronsList->size();
}

void Esoinn::updateClassLabels()
{
    markClasses();
```

```
    separateToSubclasses();
    removeNoise();

    for(auto &it : clustersList)
    {
        if(!it->neuronsList.size()) qDebug() << "ho" << "\n";
    }
}

//TODO: implement this function
void Esoinn::inputSignal(double* inputVector)
{
/*-----------------1.Initialize-set-of-2-neurons-with-2-first-weights-taken-from-input*/
    if (neuronsList.size() < 2)//better count of all input signals
    {
        addNeuron(inputVector);
        return;
    }
/*-----------------1.end-----------------------------------------------------------*/

/*-----------------2.Finding-first-and-second-winner-for-input-signal----------------*/
    vertex firstWinner = nullptr;
    vertex secondWinner = nullptr;
    findWiner(inputVector, firstWinner, secondWinner);
/*-----------------2.end-----------------------------------------------------------*/
/*-----------------3.add-neuron-if-the-distance-between-inputVector-and-winner-or-secondWinner-is-greater-than-threshold-
------*/
    if(!isWithinThreshold(firstWinner, secondWinner, inputVector))
    {
        addNeuron(inputVector);
            return;
    }
/*-----------------3.end-----------------------------------------------------------*/
/*-----------------4.increase-age-of-connection,-which-belongs-to-winner--------------*/
    for (auto &it : firstWinner->neighboursList) it->incAge();

/*----------------4.end.----------------------------------------------------------*/

/*----------------5.To-create-connections-between-winner-and-secondWinner-if necessary*/
    auto d = getConnection(firstWinner, secondWinner);
    if(needAddConnection(firstWinner, secondWinner))
            {
        if(!d)
                        {
            addConnection(firstWinner, secondWinner);
        }
        else d->setAge(0);
    }
    else
    if (d)
    {
        removeConnection(d);
        connectionsList.remove(d);
    }
            /*-----------------5.end.----------------------------------------------------*/

    /*-----------------6.Update the density of winner-----------------------------------*/
    updateDensity(firstWinner);
    /*-----------------6.end.----------------------------------------------------------*/

    /*-----------------7.Increase-signals-of--neuron-winner---------------------------*/
    //firstWinner->incSignal();
    /*-----------------7.end.----------------------------------------------------------*/
```

```
    /*-----------------8.Adapt weight vectors of winner and it's neighbours------------*/
    //qDebug() << firstWinner->weights[0] << firstWinner->weights[1];
    adaptWeights(firstWinner, inputVector);
    //qDebug() << firstWinner->weights[0] << firstWinner->weights[1];
    /*-----------------8.end.-----------------------------------------------------------*/

    /*-----------------9.Find-old-edges-and-remove-them---------------------------------*/
    removeOldConnections();
    /*-----------------9.end.-----------------------------------------------------------*/
    /*---------------- 10.Separate-all-classes-on--subclasses--------------------------*/
    if(!(this->LT % this->lambda))
    {
       updateClassLabels();

    }
            /*---------------- 10.end.----------------------------------------------------------*/
    /*-----------------11.-Delete-nodes-polluted-by-noise-------  ----------------------*/

    this->LT++;
    /*-----------------11.end.----------------------------------------------------------*/

}

 /*int* Esoinn::interpolation(int* pixels, int w1, int h1, int w2, int h2)
 {
    int* temp = new int[w2 * h2] ;
    // EDIT: added +1 to account for an early rounding problem
    int x_ratio = (int)((w1 << 16) / w2) + 1;
    int y_ratio = (int)((h1 << 16) / h2) + 1;

    int x2, y2;
    for (int i = 0; i < h2; ++i)
    {
       for (int j = 0; j < w2; ++j)
       {
          x2 = ((j * x_ratio) >> 16);
          y2 = ((i * y_ratio) >> 16);
          temp[(i * w2) + j] = pixels[(y2 * w1) + x2];
       }
    }
    return temp ;
}*/


void Esoinn::writeStructureToFile(string fileName)
{
    std::ofstream out(fileName.c_str(),  std::ofstream::out);
    out << neuronsList.size() << "\n";
    for(auto &it : neuronsList)
             {
       out << it->weights[0] << " " << it->weights[1] << " ";
       for(auto &it2 : it->neighboursList)
                       {
          vertex n;
          /*if ((*it2)->first != (*it)) n = (*it2)->first;
          else n = (*it2)->second;
          */
          n = it2->getNeighbourNeuron(it);
          int cnt = 0;
          for(auto &it3 : neuronsList)
                                  {
             if (n == it3)
```

```
                                        {
                out << cnt << " ";
            }
            cnt++;
        }
    }
    out << "\n";
  }
}

string Esoinn::getCurrentParams(){
  return "Neurons count: " + to_string(neuronsList.size()) + ", Connections count: " + to_string(connectionsList.size()) +
", Clusters count: " + to_string(clustersList.size());
}

double ** Esoinn::getStructure()
{
  //std::ofstream out(fileName.c_str(),  std::ofstream::out);
  //std::ofstream ofs ("test.txt"
  double ** structure = new double * [neuronsList.size() + 1];
  structure[0] = new double[2];
  structure[0][0] = neuronsList.size();
  structure[0][1] = dimensionSize;
  int i = 1;
  for(auto &it : neuronsList)
  {
    structure[i] = new double[it->neighboursList.size() + 2 + dimensionSize];
    if (it->isClassified())
      structure[i][0] = it->getCluster()->getId();
    else structure[i][0] = -1;
    for (int k = 0; k < dimensionSize; k++)
      structure[i][k + 1] = it->weights[k];
    //cout << structure[i][0] << " " << structure[i][1] << " " << i << " " << neuronsList->size() << endl;
    if (!it->neighboursList.size()) structure[i][1 + dimensionSize] = -1;

    int j = 1 + dimensionSize;
    for(auto &it2 : it->neighboursList){
      vertex n;
      if (it2->first != it) n = it2->first;
      else n = it2->second;
      int cnt = 0;
      for(auto &it3 : neuronsList){
        //cout << n << " " << *it3 << endl;
        if (n == it3){
          structure[i][j] = cnt;
          structure[i][j + 1] = -1;
          j++;
          break;
        }
        cnt++;
      }
    }
    i++;
  }
  return structure;
}

void Esoinn::clearWinners()
{
  for (auto &it : neuronsList) it->winInThisIter = false;
}

int Esoinn::getNeuronId(vertex neuron){
```

```
    int  neuron_ind = 0;
    for (auto &n : neuronsList){
        if (neuron == n)
            return neuron_ind;
        neuron_ind++;
    }
}

vertex Esoinn::getNeuronById(int id){
    int  neuron_ind = 0;
    for (auto &n : neuronsList)
    {
        if (neuron_ind == id) return n;
        neuron_ind++;
    }
}

void Esoinn::saveStateToFile(string fileName){
    ofstream file(fileName);
    file << dimensionSize << " " << maximalConnectionAge << " " << lambda << " " << c1 << " " << c2 << endl;
    file << neuronsList.size() << " " << connectionsList.size() << " " << clustersList.size() << endl;
    for(auto &it : neuronsList){
        for (int i = 0; i < dimensionSize; i++)
            file << it->weights[i] << " ";
        file << it->getNeuronData() << endl;
    }
    for (auto &it : connectionsList){
        file << it->getAge() << " " << getNeuronId(it->first) << " " << getNeuronId(it->second) << endl;
    }
    file << endl;
    for (auto &it : clustersList){
        file << it->getDensity() << " " << it->getId() << " " << getNeuronId(it->apex) << endl;
        file << it->neuronsList.size() << endl;
        for (auto &neuron : it->neuronsList)
            file << getNeuronId(neuron) << " ";
        file << endl;
    }
    file << endl;
}


void Esoinn::loadStateFromFile(string fileName){
    ifstream file(fileName);
    file >> dimensionSize >>  maximalConnectionAge >> lambda >> c1 >> c2;
    int neurons_list_size, connections_list_size, clusters_list_size;
    file >> neurons_list_size >> connections_list_size >> clusters_list_size;
    for(int i = 0; i < neurons_list_size; i++){
        double * w = new double[dimensionSize];
        for (int i = 0; i < dimensionSize; i++)
            file >> w[i];
        vertex n = addNeuron(w);
        double d1,d2,d3; int i1,i2,i3; bool b1;
        file >> d1 >> d2 >> b1 >> i1 >> d3 >> i2 >> i3;
        n->setNeuronData(d1, d2, b1, i1, d3 ,i2 ,i3);
    }
    for (int i = 0; i < connections_list_size; i++){
        int f_id, s_id, age;
        file >> age >> f_id >> s_id;
        vertex n1 = getNeuronById(f_id);
        vertex n2 = getNeuronById(s_id);
        auto con = addConnection(n1, n2);
        n1->neighboursList.push_back(con);
        n2->neighboursList.push_back(con);
```

```
        con->setAge(age);
    }
    for (int i = 0; i < clusters_list_size; i++){
        double dens;
        int id, neuronId, neuronsSize;
        file >> dens >> id >> neuronId >> neuronsSize;
        auto cl = addCluster(getNeuronById(neuronId));
        if (id == -1)
            id = clustersId++;
        cl->setId(id);
        cl->setDensity(dens);
        for (int j = 0; j < neuronsSize; j++){
            int n_id; file >> n_id;
            vertex neuron = getNeuronById(n_id);
            cl->neuronsList.push_back(neuron);
            neuron->setArea(cl);
        }
    }
}

bool comparator(cluster a, cluster b){
    return a->neuronsList.size() > b->neuronsList.size();
}

double ** Esoinn::getTopVectors(int cnt){
    vector<cluster> cl;
    for (auto &it: clustersList){
        cl.push_back(it);
    }
    sort(cl.begin(), cl.end(), comparator);
    double ** result = new double*[cnt];
    for (int i = 0; i < cnt; i++){
        result[i] = new double[27*27];
        if (cl[i]->apex != nullptr)
            for (int j = 0; j < 27*27; j++)
                result[i][j] = cl[i]->apex->weights[j];
        else result[i][0] = -1;
    }
    return result;
}

vertex Esoinn::getPattern(double * inputVector){
    vertex firstWinner = nullptr;
    vertex secondWinner = nullptr;
    findWiner(inputVector, firstWinner, secondWinner);
    if (firstWinner && firstWinner->getCluster())
        return firstWinner->getCluster()->getApex();
    else return nullptr;
}

vertex Esoinn::getNeuronPatern(vertex n){
    if (n->getCluster()){
        if (n->getCluster()->getApex() == n)
            return n;
        else return n->getCluster()->getApex();
    }
    else return nullptr;
}
```

Листинг файла dataExchanger.h

```
#ifndef PERSON_H
#define PERSON_H

#include <QObject>
#include <QQuickImageProvider>
#include <QImage>
#include <string>
#include "ESOINNLibSources/esoinn.h"
#include "ESOINNLibSources/simpleam.h"
//![0]

typedef QUrl imgType;

class dataExchanger : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString structureData READ structureData WRITE setStructureData)
    Q_PROPERTY(QList<QString> esoinnParams READ esoinnParams WRITE setEsoinnParams)

    Q_PROPERTY(QUrl im READ im WRITE sim)
    Q_PROPERTY(QUrl pointedImage READ pointedImage WRITE setPointedImage)

    Q_PROPERTY(QUrl loadStructure READ loadStructure WRITE setLoadStructure)
    Q_PROPERTY(QUrl saveStructure READ saveStructure WRITE setSaveStructure)

    Q_PROPERTY(QUrl loadVector READ loadVector WRITE setLoadVector)

    Q_PROPERTY(int dimensionsCnt READ dimensionsCnt WRITE setDimensionsCnt)

    Q_PROPERTY(QString currentNNparams READ currentNNparams WRITE setCurrentNNparams)

    Q_PROPERTY(QString saveMainVectors READ saveMainVectors WRITE setSaveMainVectors)
public:
    dataExchanger(QObject *parent = 0);
    //shared data
    QString structureData() const;
    void setStructureData(const QString &);

    QList<QString> esoinnParams() const;
    void setEsoinnParams(const QList<QString> &);

    imgType im() const;
    void sim(const imgType &);

    QUrl pointedImage() const;
    void setPointedImage(const QUrl &);

    QUrl loadStructure() const;
    void setLoadStructure(const QUrl &);

    QUrl saveStructure() const;
    void setSaveStructure(const QUrl &);

    QUrl loadVector() const;
    void setLoadVector(const QUrl &);

    int dimensionsCnt() const;
    void setDimensionsCnt(const int &);
    //При реализации проекта по созданию портала используются средства государственной поддержки,
выделенные в качестве гранта в соответствии с распоряжением Президента Российской Федерации от 17.01.2014
№ 11-рп и на основании конкурса, проведенного Фондом ИСЭПИ.
    QString currentNNparams() const;
```

```
    void setCurrentNNparams(const QString &);

    QString saveMainVectors() const;
    void setSaveMainVectors(const QString &);


    //local data
    Esoinn * es;
    QImage * image;
    double ** vectors;
    double ** normalizedVectors;
    int dimSize;
    int vectorsCnt;

    //SimpleAM am;

private:
    QString m_structureData;
    QList<QString> m_esoinnParams;
    imgType m_im;
    string m_loadStructure;
    int m_dimensionsCnt;

    QString m_currentNNparams;
};
//![0]

#endif // PERSON_H
```

Листинг файла dataExchanger.cpp

```
#include "dataExchanger.h"
#include <QtDebug>
#include <QTime>
#include <fstream>
//#include <random>

// ![0]
dataExchanger::dataExchanger(QObject *parent): QObject(parent)
{
    es = NULL;
    vectors = NULL;
    normalizedVectors = NULL;
}

QString dataExchanger::structureData() const
{
    return m_structureData;
}

void dataExchanger::setStructureData(const QString &n)
{
    m_structureData = n;
}

int dataExchanger::dimensionsCnt() const{
    return m_dimensionsCnt;

}
void dataExchanger::setDimensionsCnt(const int &number){
    m_dimensionsCnt = number;
}

QList<QString> dataExchanger::esoinnParams() const
{
    return m_esoinnParams;
}

imgType dataExchanger::im() const
{
    return m_im;
}

QString dataExchanger::currentNNparams() const
{
    return m_currentNNparams;
}

QString dataExchanger::saveMainVectors() const{
    return "";
}

void dataExchanger::setSaveMainVectors(const QString & str){
    double ** vects = es->getTopVectors(100);
    for (int i = 0; i < 100; i++){
        uchar * arr = new uchar[28*28*4];
        if (vects[i][0] != -1){
            for (int j = 0; j < 28*28*4; j+=4){
                arr[j + 3] = 255;
                arr[j + 2] = arr[j + 1] = arr[j] = vects[i][j/4];
            }
            QImage * qim = new QImage(arr,28,28,QImage::Format_ARGB32);
            qim->save("image" + QString::number(i) + ".png");
```

```
        }
    }
}

void dataExchanger::setCurrentNNparams(const QString &str){
    m_currentNNparams = str;
}

void dataExchanger::sim(const imgType &n)
{

    m_im = n;
    if (!image)
        delete image;
    image = new QImage(n.toLocalFile());

    double ** shuf_arr = new double*[image->width() * image->height()];
    int points_cnt = 0;
    for (int i = 0; i < image->height(); i++){
        for (int j = 0; j < image->width(); j++){
            QColor qc(image->pixel(j,i));
            if (qc.red() < 100 || qc.green() < 100 || qc.blue() < 100){
                shuf_arr[points_cnt] = new double[2];
                shuf_arr[points_cnt][0] = j;
                shuf_arr[points_cnt][1] = i;
                points_cnt++;
            }
        }
    }
    if (normalizedVectors){
        for (int i = 0; i < vectorsCnt; i++)
            delete[] normalizedVectors[i];
        delete[] normalizedVectors;
    }
    if (vectors){
        for (int i = 0; i < vectorsCnt; i++)
            delete[] vectors[i];
        delete[] vectors;
    }
    vectors = new double*[points_cnt];
    vectorsCnt = points_cnt;
    dimSize = 2;
    for (int i = 0; i < points_cnt; i++){
        vectors[i] = new double[2];
        vectors[i][0] = shuf_arr[i][0];
        vectors[i][1] = shuf_arr[i][1];
        delete[] shuf_arr[i];
    }
    delete[] shuf_arr;
    m_dimensionsCnt = dimSize;
}

QUrl dataExchanger::pointedImage() const
{
    return m_im;
}

void dataExchanger::setPointedImage(const QUrl &n)
{
    m_im = n;
    delete image;
    image = new QImage(n.toLocalFile());
    QString strs = "";
```

```
    for (int i = 0; i < image->height(); i++){
        for (int j = 0; j < image->width(); j++){
            QColor qc(image->pixel(j,i));
            if (qc.red() < 100 || qc.green() < 100 || qc.blue() < 100){
                strs += QString::number(j);
                strs += " ";
                strs += QString::number(i);
                strs += ",";
            }
        }
    }
    setStructureData(strs);
}

void dataExchanger::setLoadStructure(const QUrl &filePath){
    auto fileName = filePath.toLocalFile().toStdString();
    if (es == NULL){
        es = new Esoinn(fileName);
    }
    else {
        es->~Esoinn();
        es = new Esoinn(fileName);
    }
    QString qs = "";
    double ** str = es->getStructure();
    m_dimensionsCnt = str[0][1];
    for (int ii = 1; ii < str[0][0] + 1; ii++){
        for (int jj = 0; jj < str[0][0] + 4; jj++){
            if (jj > 1 && str[ii][jj] == -1)
                break;
            qs += QString::number(str[ii][jj]);
            qs += " " ;

        }
        qs += "/";
    }
    int n = str[0][0] + 1;
    for(int i = 0; i < n; ++i) delete[] str[i];
    delete[] str;
    qs += ";";
    setStructureData(qs);
}

QUrl dataExchanger::loadStructure() const{
    return m_im;
}

QUrl dataExchanger::saveStructure() const{
    return m_im;
}
void dataExchanger::setSaveStructure(const QUrl &filePath){
    auto fileName = filePath.toLocalFile().toStdString();
    es->saveStateToFile(fileName);
}

QUrl dataExchanger::loadVector() const{
    return m_im;
}
void dataExchanger::setLoadVector(const QUrl &filePath){
    auto fileName = filePath.toLocalFile().toStdString();


    if (normalizedVectors){
```

```
        for (int i = 0; i < vectorsCnt; i++)
            delete[] normalizedVectors[i];
        delete[] normalizedVectors;
    }
    if (vectors){
        for (int i = 0; i < vectorsCnt; i++)
            delete[] vectors[i];
        delete[] vectors;
    }
    list<double> oneVect;
    ifstream in(fileName.c_str());
    char c; double num; in >> c;
    while (c != ')'){
        in >> num >> c;
        oneVect.push_back(num);
    }
    dimSize = oneVect.size();
    list<double *> vects;
    double * d = new double[dimSize];
    int i = 0;
    for (auto &it: oneVect)
        d[i++] = it;
    vects.push_back(d);
    while (in >> c){
        d = new double[dimSize];
        for (int i = 0; i < dimSize; i++)
            in >> d[i] >> c;
        vects.push_back(d);
    }
    vectorsCnt = vects.size();
    vectors = new double*[vectorsCnt];
    i = 0;
    for (auto &it: vects){
        vectors[i++] = it;
    }




    m_dimensionsCnt = dimSize;
}

void dataExchanger::setEsoinnParams(const QList<QString> &n){
    m_esoinnParams = n;
    QString qs;
    qsrand(0);
    bool randomizeDataOrder = m_esoinnParams[1] == "true" ? true : false;
    bool visualizeEveryStep = m_esoinnParams[2] == "true" ? true : false;
    bool normalizeInput = m_esoinnParams[5] == "true" ? true : false;
    if (m_esoinnParams[3].toDouble() == 1 || es == NULL)
        es = new Esoinn(m_dimensionsCnt, m_esoinnParams[6].toDouble(), m_esoinnParams[7].toDouble(),
m_esoinnParams[8].toDouble(), m_esoinnParams[9].toDouble());

    double ** cur_vectors = vectors;
    if (normalizeInput){
        if (!normalizedVectors){
            double * max_vals = new double[dimSize];
            double * min_vals = new double[dimSize];
            for (int j = 0; j < dimSize; j++){
                max_vals[j] = vectors[0][j];
                min_vals[j] = vectors[0][j];
            }
            for (int i = 0; i < vectorsCnt; i++)
```

```
            for (int j = 0; j < dimSize; j++){
                if (vectors[i][j] > max_vals[j])
                    max_vals[j] = vectors[i][j];
                if (vectors[i][j] < min_vals[j])
                    min_vals[j] = vectors[i][j];
            }
        double * norma = new double[dimSize];
        double max = max_vals[0] - min_vals[0];
        for (int j = 0; j < dimSize; j++){
            norma[j] = max_vals[j] - min_vals[j];
            if (max < norma[j])
                max = norma[j];
        }
        for (int j = 0; j < dimSize; j++)
            norma[j] = max / norma[j];
        normalizedVectors = new double * [vectorsCnt];
        for (int i = 0; i < vectorsCnt; i++){
            normalizedVectors[i] = new double[dimSize];
            for (int j = 0; j < dimSize; j++)
                normalizedVectors[i][j] = (vectors[i][j] - min_vals[j]) * norma[j];
        }
    }
    cur_vectors = normalizedVectors;
}

//double values are situated in vectors array!
double ** shuf_arr = new double*[vectorsCnt];
for (int i = 0; i < vectorsCnt; i++)
    shuf_arr[i] = cur_vectors[i];
for (int iter = 0; iter < m_esoinnParams[4].toDouble(); iter++)
{
    if (randomizeDataOrder){
        for (int i = 0; i < vectorsCnt; i++){
            double * temp = shuf_arr[i];
            int swap_cell = qrand() % vectorsCnt;
            shuf_arr[i] = shuf_arr[swap_cell];
            shuf_arr[swap_cell] = temp;
        }
    }

    es->clearWinners();
    for (int i = 0; i < vectorsCnt; i++){
        double * w = new double[dimSize];
        for (int j = 0; j < dimSize; j++)
            w[j] = shuf_arr[i][j];
        es->inputSignal(w);
        delete[] w;
        if (visualizeEveryStep || (i == vectorsCnt - 1)){
            double ** str = es->getStructure();
            for (int ii = 1; ii < str[0][0] + 1; ii++){
                for (int jj = 0; jj < str[0][0] + 4; jj++){
                    //qDebug() << str[i] << " ";
                    if (jj > 1 && str[ii][jj] == -1)
                        break;
                    qs += QString::number(str[ii][jj]);
                    qs += " " ;
                }
                qs += "/";
            }
            int n = str[0][0] + 1;
            for(int i = 0; i < n; ++i) delete[] str[i];
            delete[] str;
            qs += ";";
```

```
            }
        }
    }
    m_dimensionsCnt = dimSize;
    setCurrentNNparams(QString::fromStdString(es->getCurrentParams()));



    //getting esoinn structure from double array
    delete[] shuf_arr;
    //Loading data to class. This data now will be available in QML
    setStructureData(qs);

}
```

Листинг файла main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QtDebug>


#include <QQuickView>
#include <QVariantList>
#include <QQmlComponent>
#include <QGuiApplication>
#include <QCoreApplication>
#include <QQmlEngine>
#include <QQmlComponent>
#include <QDebug>
#include <QtGlobal>
#include <QTime>
#include <QObject>
#include "ESOINNLibSources/esoinn.h"
#include "dataExchanger.h"
#include <QQmlContext>
#include <QVariant>

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    //Start of esoinn is in this class (dataExchanger) and it starts on click button LEARN
    dataExchanger dataEx;
    engine.rootContext()->setContextProperty("dataEx", &dataEx);
    //loading main qml file
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();

}
```

Листинг файла main.qml

```
import QtQuick 2.7
import QtQuick.Window 2.2
import QtQuick.Controls 1.2
import QtQuick.Dialogs 1.2
import QtGraphicalEffects 1.0

Window {
    visible: true
    width:1000
    height:800
    MainForm {
        id: mainForm
        color:"#FFFFFF"
        anchors.fill: parent
        //canvas is main control where i draw data
        FileDialog{
            id: fileDialog
            title: "Please choose a image file to load"
            nameFilters: [ "Image files (*.jpg *.png)" ]
            onAccepted: {
                 //console.log("You chose: " + fileDialog.fileUrls)
                imagePreview.source = fileDialog.fileUrl
                dataEx.im = imagePreview.source
            }
        }

        FileDialog{
            id: saveFileDialog
            title: "Please choose a image file to load"
            selectExisting:false
            nameFilters: [ "Text files (*.txt *.)" ]
            onAccepted: {
                dataEx.saveStructure = saveFileDialog.fileUrl.toString();
            }
        }
        FileDialog{
            id: openFileWithVector
            title: "Choose a file with input data in vectors"
            nameFilters: [ "Text files (*.txt *.)" ]
            onAccepted:{
                dataEx.loadVector = openFileWithVector.fileUrl.toString();
            }
        }

        FileDialog{
            id: loadFileDialog
            title: "Please choose a image file to load"
            nameFilters: [ "Text files (*.txt *.)" ]
            onAccepted: {
                dataEx.loadStructure = loadFileDialog.fileUrl.toString();
                settingsBar.currEsoinnData = dataEx.structureData.split(';');
                dataShowTimer.running = true;
            }
        }

        Rectangle{
            id: settingsBar
            height: 30
            color: "red"
            anchors.top: mainForm.top
            Button{
                id: loadImgButton
```

```
            text:"Load image"
            onClicked: {
                fileDialog.open()
            }
        }
        Button {
            id:loadInputVector
            anchors.leftMargin: 10
            anchors.left: loadImgButton.right
            text: "Load input vector"
            onClicked: {
                openFileWithVector.open();
            }
        }

        Button{
            id: showImgButton
            anchors.leftMargin: 10
            anchors.left: loadInputVector.right
            text:"Show pointed image"
            onClicked: {
                dataEx.pointedImage = imagePreview.source
                canvas.loadStructure();
            }
        }

        ComboBox{
            id: nnComboBox
            width:70
            currentIndex: 0
            anchors.left: showImgButton.right
            anchors.leftMargin: 10
            model:["ESOINN"]
        }
        property var visualizeIter : 0;
        property var currEsoinnData : [];
        function learn(fromBegin){
            var beg_time = new Date();
            var arr = [nnComboBox.currentText, randomInput.checked, fullVisualize.checked, fromBegin, iterEdit.text,
normalizeInput.checked, parseFloat(conAge.text), parseFloat(lambda.text), parseFloat(c1P.text), parseFloat(c2P.text)];
            dataEx.esoinnParams = arr;
            visualizeIter = 0;
            currEsoinnData = dataEx.structureData.split(';');
            var end_time = new Date();
            function time_ring(value, type){
                if (value >= 0){
                    if (type != "ms")
                        return value;
                    else {
                        if (value < 10)
                            return "00" + value;
                        else if (value < 100) return "0" + value;
                        else return value;
                    }
                }
                if (type == "ms"){
                    value = 1000 + value;
                    if (value < 10)
                        return "00" + value;
                    else if (value < 100) return "0" + value;
                    else return value;
                }
                else return 60 + value;
```

```
            }

            var time_diff = time_ring(end_time.getMinutes() - beg_time.getMinutes(), "m") + "m. " +
time_ring(end_time.getSeconds() - beg_time.getSeconds(), "s") + "." + time_ring(end_time.getMilliseconds() -
beg_time.getMilliseconds(), "ms") + "s.";
            learnResultsText.text = dataEx.currentNNparams + ", Time elapsed: " + time_diff;
            //dataEx.saveMainVectors = "a";
            dataShowTimer.running = true;
        }

        Timer{
            id: dataShowTimer
            interval: 10;
            repeat: true;
            running: false;
            onTriggered: {
                if (settingsBar.visualizeIter < settingsBar.currEsoinnData.length - 1){
                    canvas.loadStructure(settingsBar.currEsoinnData[settingsBar.visualizeIter]);
                    settingsBar.visualizeIter++;
                }
            }
        }

        Button{
            id: learnBeginButton
            anchors.left: nnComboBox.right
            anchors.leftMargin: 10
            text:"LEARN FROM BEGIN"
            onClicked: {
                settingsBar.learn("1");

            }
        }

        Button{
            id: learnButton
            anchors.left: learnBeginButton.right
            anchors.leftMargin: 10
            text:"LEARN ITERATIONS"
            onClicked: {
                settingsBar.learn("0");
            }
        }

        Button{
            id: saveNNStructure
            anchors.left: learnButton.right
            anchors.leftMargin: 20
            text:"Save structure"
            onClicked: {
                saveFileDialog.open();
            }
        }
        Button{
            id: loadNNStructure
            anchors.left: saveNNStructure.right
            anchors.leftMargin: 10
            text:"Load structure"
            onClicked: {
                loadFileDialog.open();
            }
        }
    }
```

```
Rectangle{
    id: settings2Bar
    height: 30
    color: "red"
    anchors.top: settingsBar.bottom
    anchors.leftMargin: 5
    Text{
        id: paramsText
        anchors.leftMargin: 10
        font.pointSize: 12
        text: "LEARN PARAMETRS:"
    }
    Text{
        id: iterText
        anchors.left: paramsText.right
        anchors.leftMargin: 5
        font.pointSize: 12
        text: "Iterations:"
    }
    TextEdit{
        id: iterEdit
        color:"blue"
        anchors.left: iterText.right
        anchors.leftMargin: 5
        font.pointSize: 12
        text:"1"
    }
    Text{
        id: text1
        anchors.left:iterEdit.right
        font.pointSize: 12
        anchors.leftMargin: 15
        text: "Max connection age:"
    }

    TextEdit{
        id: conAge
        color:"blue"
        anchors.left: text1.right
        anchors.leftMargin: 5
        font.pointSize: 12
        text:"100"
    }
    Text{
        id: text2
        anchors.left:conAge.right
        font.pointSize: 12
        anchors.leftMargin: 15
        text:"Lambda:"
    }
    TextEdit{
        id: lambda
        color:"blue"
        anchors.left: text2.right
        anchors.leftMargin: 5
        font.pointSize: 12
        text:"100"
    }
    Text{
        id: text3
        anchors.left:lambda.right
        anchors.leftMargin: 15
```

```
        font.pointSize: 12
        text:"c1:"
    }
    TextEdit{
        id: c1P
        color:"blue"
        anchors.left: text3.right
        anchors.leftMargin: 5
        font.pointSize: 12
        text:"0.01"
    }
    Text{
        id: text4
        anchors.left:c1P.right
        anchors.leftMargin: 15
        font.pointSize: 12
        text:"c2:"
    }
    TextEdit{
        id: c2P
        color:"blue"
        anchors.left: text4.right
        anchors.leftMargin: 5
        font.pointSize: 12
        text:"1"
    }
}
Rectangle{
    id: settings3Bar
    height: 30
    anchors.top: settings2Bar.bottom
    CheckBox{
        id: randomInput
        text:"Randomize input data"
        checked: true
    }
    CheckBox{
        id: fullVisualize
        anchors.leftMargin: 5
        anchors.left: randomInput.right
        text:"Visualize every step"
        checked: false
    }
    CheckBox{
        id: normalizeInput
        anchors.leftMargin: 5
        anchors.left: fullVisualize.right
        text: "Normalize input"
        checked: false
    }
    CheckBox{
        id: showClustersIds
        anchors.leftMargin: 5
        anchors.left: normalizeInput.right
        text: "Show clusters ids"
        checked: false
    }

    Text{
        id: dimsText
        anchors.leftMargin: 5
        anchors.left: showClustersIds.right
        font.pointSize: 12
```

```
        text:"Visualize dimensions ids: "
    }
    TextEdit{
        id: fstDimId
        anchors.leftMargin: 5
        anchors.left: dimsText.right
        font.pointSize: 12
        color:"blue"
        text : "1"
    }
    Text{
        id: dimsDefisText
        anchors.leftMargin: 5
        anchors.left: fstDimId.right
        font.pointSize: 12
        text:"-"
    }
    TextEdit{
        id: secondDimId
        anchors.leftMargin: 5
        anchors.left: dimsDefisText.right
        font.pointSize: 12
        color:"blue"
        text: "2"
    }
}
Rectangle{
    id: settings4Bar
    height: 30
    anchors.top: settings3Bar.bottom
    Text{
        id: learnResultsText
    }
}

Image{
    id: imagePreview
    width: mainForm.width / 2
    anchors.top: settings4Bar.bottom
    anchors.bottom: mainForm.bottom
    anchors.left: mainForm.left
}
Colorize {
        anchors.fill: imagePreview
        source: imagePreview
        hue: 0.0
        saturation: 0
        lightness: 0
   }

Canvas{
    id: canvas
    anchors.top: settings4Bar.bottom
    anchors.bottom: mainForm.bottom
    anchors.left: imagePreview.right
    anchors.right: mainForm.right
    //anchors.fill: parent
    property var ctx : canvas.getContext("2d")

    onPaint:{
        ctx = canvas.getContext('2d');
        //Draw vertical coordinates line
        /*ctx.lineWidth = 1
```

```
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.beginPath()
            ctx.moveTo(canvas.width / 2,0)
            ctx.lineTo(canvas.width / 2,canvas.height)
            ctx.stroke()
            //Draw horizontal coordinates line
            ctx.lineWidth = 1
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.beginPath()
            ctx.moveTo(0,canvas.height / 2)
            ctx.lineTo(canvas.width,canvas.height / 2)
            ctx.stroke()*/
        }
        function loadStructure(data) {
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            //data from esoin is in string in Person.name in format: "x y, x y, x y,"
            var arr = data.split('/');
            var showed_clusters = {};
            var cur_cluster_id = 0;

            var dimSize = dataEx.dimensionsCnt;
            var fstDim = parseInt(fstDimId.text);
            var secDim = parseInt(secondDimId.text);
            //console.log(arr)
            var min_x = 1000000, max_x = -1000000, min_y = 1000000, max_y = -1000000
            //finding the square, in that all data is lay
            for (var i = 0; i < arr.length; i++){
                var numz = arr[i].split(' ');
                numz[fstDim] = parseInt(numz[fstDim])
                numz[secDim] = parseInt(numz[secDim])
                if (numz[fstDim] > max_x)
                    max_x = numz[fstDim];
                if (numz[fstDim] < min_x)
                    min_x = numz[fstDim];
                if (numz[secDim] < min_y)
                    min_y = numz[secDim];
                if (numz[secDim] > max_y)
                    max_y = numz[secDim];
            }
            //coefficents to transform data coordinates to see input in full window size
            var offset_x = -min_x;
            var offset_y = -min_y;
            if (max_x - min_x == 0)
                max_x = min_x + 1;
            if (max_y - min_y == 0)
                max_y = min_y + 1;
            var scale_x = (canvas.width - 100) / (max_x - min_x);
            var scale_y = (canvas.height - 100)/ (max_y - min_y);
            //console.log(min_x + ' ' + max_x + ' ' + min_y + ' ' + max_y)

            //this function resizes input coordinates fit to canvas size
            function newCoords(offset, scale, num){
                return parseInt((parseInt(num) + offset) * scale + 50)
            }

            //draw points, that interpritates data
            for (var i = 0; i < arr.length; i++){
                var nums = arr[i].split(' ');
                 //console.log(((parseInt(nums[0]) + offset_x) * scale_x) + ' ' + parseInt((parseInt(nums[1]) + offset_y)  *
scale_y))
                ctx.beginPath();
```

```
    //ctx.fillStyle = "green"
    //ctx.strokeStyle = "blue"
    //if (nums[2] > 0)
      //  ctx.fillStyle = "red"
  function getCol(x){
  if (x == 0) return 255;
  var pow2 = Math.log(x+1) / Math.log(2);
  var kolco = parseInt((x+1) % Math.pow(2,Math.floor(pow2)) * 2 - 1);
  if (kolco == -1)
    kolco = Math.pow(2,Math.ceil(pow2)) - 1;
  return 256/Math.pow(2,Math.ceil(pow2))*kolco;
  }
  var color = getCol(nums[0]) / 255;
  if (!nums[0] || nums[0] == -1){
     color = 0;
  }

  var red = (nums[0] % 6 == 0 || nums[0] % 6 == 3 || nums[0] % 6 == 5) ? color : 0;
  var green = (nums[0] % 6 == 0 || nums[0] % 6 == 4 || nums[0] % 6 == 2) ? color : 0;
  var blue = (nums[0] % 6 == 5 || nums[0] % 6 == 4 || nums[0] % 6 == 1) ? color : 0;
    ctx.fillStyle = Qt.rgba(red, green, blue, 1);
  ctx.strokeStyle = Qt.rgba(red, green, blue, 1);
    //console.log(colors)
    //console.log(colors.length)

  var temp_x = newCoords(offset_x, scale_x, nums[fstDim]);
  var temp_y = newCoords(offset_y, scale_y, nums[secDim]);
    ctx.arc( temp_x, temp_y, 3, 0, 2*Math.PI, false)
    ctx.fill();
    ctx.stroke();

  //ctx.strokeStyle = "black"
  //ctx.fillStyle = "black"
  if (showClustersIds.checked && showed_clusters[nums[0]] != 1){
     ctx.font = "24px fantasy";
     //ctx.shadowColor = "white";
     //ctx.shadowOffsetX = "1px";
     //ctx.shadowOffsetX = "1px";
     ctx.text(cur_cluster_id, temp_x, temp_y);

     ctx.stroke();
     showed_clusters[nums[0]] = 1;
     cur_cluster_id++;
  }
  //console.log(nums[0] + "," + nums[1] + "," + nums[2] + "," + nums[3])
  //console.log(dimSize);
    //Draw connections
    for (var j = 1 + dimSize; j < nums.length; j++){
       if (nums[j] && nums[j] >= 0){
          var xy = (arr[parseInt(nums[j])]).split(' ')
          ctx.lineWidth = 1
          ctx.strokeStyle = "black"
          ctx.fillStyle = "black"
          ctx.beginPath()
          ctx.moveTo(newCoords(offset_x, scale_x, xy[fstDim]), newCoords(offset_y, scale_y, xy[secDim]))
          ctx.lineTo(newCoords(offset_x, scale_x, nums[fstDim]), newCoords(offset_y, scale_y, nums[secDim]))
          //console.log(newCoords(offset_x, scale_x, xy[0]) + ' ' + newCoords(offset_y, scale_y, xy[1]))
          //console.log(newCoords(offset_x, scale_x, nums[0])+ ' ' + newCoords(offset_y, scale_y, nums[1]))
          ctx.stroke()
       }
    }
}
//console.log(offset_x + ' ' + offset_y + ' ' + scale_x + ' ' + scale_y)
```

```
            //Draw vertical coordinates line
            /*ctx.lineWidth = 1
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.beginPath()
            ctx.moveTo( parseInt((offset_x) * scale_x) + 10,0)
            ctx.lineTo(parseInt((offset_x) * scale_x) + 10,canvas.height)
            ctx.stroke()
            //Draw horizontal coordinates line
            ctx.lineWidth = 1
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.beginPath()
            ctx.moveTo(0,parseInt((offset_y) * scale_y) + 10)
            ctx.lineTo(canvas.width,parseInt((offset_y) * scale_y) + 10)
            ctx.stroke()


            //Draw horizontal scale on coordinate
            ctx.lineWidth = 1
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.beginPath()
            ctx.moveTo(canvas.width - 10,canvas.height / 2 - 10)
            ctx.lineTo(canvas.width - 10,canvas.height / 2 + 10)
            ctx.stroke()
            ctx.beginPath();
            ctx.fillStyle = "black"
            ctx.strokeStyle = "black"
            ctx.font = "16px sans-serif";
            ctx.text(max_x, canvas.width - 30, canvas.height / 2 + 20);
            ctx.stroke();


            //Draw vertical scale on coordinate
            ctx.lineWidth = 1
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.beginPath()
            ctx.moveTo(canvas.width / 2 - 10, 10)
            ctx.lineTo(canvas.width / 2 + 10, 10)
            ctx.stroke()
            ctx.beginPath();
            ctx.strokeStyle = "black"
            ctx.fillStyle = "black"
            ctx.font = "16px sans-serif";
            ctx.text(max_y, canvas.width / 2 - 10, 25);
            ctx.stroke();*/
            canvas.requestPaint();
        }
    }
  }
}
```