```
int main(int argc, char *argv[])
{


   QApplication a(argc, argv);

   // Instantiate the viewer.
   Viewer viewer1;
   Viewer viewer1_learned;
   Viewer viewer2;
   Viewer viewer2_learned;

   viewer1.setWindowTitle("raw data 1");
   viewer1_learned.setWindowTitle("learned data 1");

   viewer2.setWindowTitle("raw data 2");
   viewer2_learned.setWindowTitle("learned data 2");

   viewer1.drawMode = 0; //raw data
   viewer1_learned.drawMode = 1;//learned

   viewer2.drawMode = 0; //raw data
   viewer2_learned.drawMode = 1;//learned

   readPointClouds1("oakland_part3_am_rf_no_label.node_features");
   readPointClouds2("oakland_part3_an_rf_no_label.node_features");

   findCentroid1();
   findCentroid2();

   //add noise of dataset2
   addNoise2();

   viewer1.drawMode = 0; //raw data
   viewer1_learned.drawMode = 1;//learned

   viewer1.drawingData = 1; //draw first data set
   viewer1_learned.drawingData = 1;

   viewer2.drawMode = 0; //raw data
   viewer2_learned.drawMode = 1;//learned

   viewer2.drawingData = 2; //draw second data set
   viewer2_learned.drawingData = 2;

   unsigned long pcInd = 0;
   unsigned long inPCInd = 0;
   int i,j;

   double W[5][10];
   PointCloud* pc;

   //initialize weights
   for(i=0;i<5;i++){
      for(j=0;j<10;j++){
         W[i][j] = 1.0;
      }
   }

//-------------------------------------------------------------------------
   unsigned short nrepeats =10;
   unsigned short irepeats =0;
   double learningRate;

   //learn both data sets
   while (irepeats <nrepeats){
      qDebug() << irepeats;

     //the first set
      npoints = npoints1;
      learningRate =  1/sqrt(npoints*nrepeats);

      //reshuffle the first set
      inPCInd = 0;
      while ( inPCInd < npoints){
         inpPointCloud[inPCInd].node_label = 0;
         inPCInd++;
      }

      pcInd = 0;
      while ( pcInd < npoints ){
         pc =  &pointCloud1[pcInd];

         inPCInd = (qrand()+ qrand() +qrand())%npoints;

         if (inpPointCloud[inPCInd].node_label == 0){//not taken yet
            for(i=0;i<3;i++){
               inpPointCloud[inPCInd].pos[i] = pc->pos[i];
            }

            inpPointCloud[inPCInd].node_label = pc->node_label;

            for(i=0;i<5;i++){
               inpPointCloud[inPCInd].node_vec[i] = pc->node_vec[i];
            }

            for(i=0;i<10;i++){
               inpPointCloud[inPCInd].features[i] = pc->features[i];
            }
            pcInd++;
         }
      }
      //end reshuffle the data

      //-------------------------------------------------------------------------
      //do the learning
      for (pcInd = 0; pcInd < npoints;pcInd++){
         pc =  &inpPointCloud[pcInd];

         double f[10][1];
         double Wf[5][1];
         double Wf_y[5][1];
         double y[5][1];
         double Wf_yT[1][5];
         double f2[10][1];
         double f2Wf_yT[10][5];
         double dLoss[5][10];

         for (i=0;i<5;i++){
            y[i][0] = pc->node_vec[i];
         }
         for (i=0;i<10;i++){
            f[i][0]  = pc->features[i];
            f2[i][0] = 2.0*f[i][0];
         }

         mat_mult((double *) W, 5,10,(double *)f,10,1,(double *)Wf);
         mat_sub ((double *) Wf,5, 1,(double *)y, (double *)Wf_y);
         mat_transpose((double *) Wf_y ,5,1,(double *) Wf_yT);
         mat_mult((double *)f2,10,1,(double *)Wf_yT,1,5,(double *)f2Wf_yT);
         mat_transpose((double *) f2Wf_yT ,10,5,(double *) dLoss);

         for(i=0;i<5;i++){
            for(j=0;j<10;j++){
               W[i][j] -= learningRate*dLoss[i][j];
            }
         }
      }

   #if 1
      //option to learn with second set also

      npoints = npoints2;
      learningRate =  1/sqrt(npoints*nrepeats);

      //reshuffle the second set
      inPCInd = 0;
      while ( inPCInd < npoints){
         inpPointCloud[inPCInd].node_label = 0;
         inPCInd++;
      }

      pcInd = 0;
      while ( pcInd < npoints ){
         pc =  &pointCloud2[pcInd];

         inPCInd = (qrand()+ qrand() +qrand())%npoints;

         if (inpPointCloud[inPCInd].node_label == 0){//not taken yet
            for(i=0;i<3;i++){
               inpPointCloud[inPCInd].pos[i] = pc->pos[i];
            }

            inpPointCloud[inPCInd].node_label = pc->node_label;

            for(i=0;i<5;i++){
               inpPointCloud[inPCInd].node_vec[i] = pc->node_vec[i];
            }

            for(i=0;i<10;i++){
               inpPointCloud[inPCInd].features[i] = pc->features[i];
            }

            pcInd++;
         }
      }
      //end reshuffle the data

      //-------------------------------------------------------------------------
      //do the learning
      for (pcInd = 0; pcInd < npoints;pcInd++){
         pc =  &inpPointCloud[pcInd];

         double f[10][1];
         double Wf[5][1];
         double Wf_y[5][1];
         double y[5][1];
         double Wf_yT[1][5];
         double f2[10][1];
         double f2Wf_yT[10][5];
         double dLoss[5][10];

         for (i=0;i<5;i++){
            y[i][0] = pc->node_vec[i];
         }
         for (i=0;i<10;i++){
            f[i][0]  = pc->features[i];
            f2[i][0] = 2.0*f[i][0];
         }

         mat_mult((double *) W, 5,10,(double *)f,10,1,(double *)Wf);
         mat_sub ((double *) Wf,5, 1,(double *)y, (double *)Wf_y);
         mat_transpose((double *) Wf_y ,5,1,(double *) Wf_yT);
         mat_mult((double *)f2,10,1,(double *)Wf_yT,1,5,(double *)f2Wf_yT);
         mat_transpose((double *) f2Wf_yT ,10,5,(double *) dLoss);

         for(i=0;i<5;i++){
            for(j=0;j<10;j++){
               W[i][j] -= learningRate*dLoss[i][j];
            }
         }
      }
      //end learning
      //--------------------------
#endif
      irepeats++;
   }

   //-----------------------------------------------------------------
   //learning is completed
   //test performance

   //test first data set
   npoints = npoints1;
   for (pcInd = 0; pcInd < npoints;pcInd++){
      PointCloud* pc =  &pointCloud1[pcInd];
      double Wf[5][1];
      double f[10][1];
      for (i=0;i<10;i++){
         f[i][0]  = pc->features[i];
      }
      mat_mult((double *) W, 5,10,(double *)f,10,1,(double *)Wf);

      //which node does it predict?
      int max_element = 6;
      double max_element_value  =0.0;
      for (i=0;i<5;i++){
         if (Wf[i][0] > max_element_value){
            max_element_value = Wf[i][0];
            max_element = i;
         }
      }

      switch (max_element){
      case 0:
         pc->learned_label = NODE_VEG;
         break;
      case 1:
         pc->learned_label = NODE_WIRE;
         break;
      case 2:
         pc->learned_label = NODE_POLE;
         break;
      case 3:
         pc->learned_label = NODE_GROUND;
         break;
      case 4:
         pc->learned_label = NODE_FACADE;
         break;
      }
   }

   //test second data set
   npoints = npoints2;
   for (pcInd = 0; pcInd < npoints;pcInd++){
      PointCloud* pc =  &pointCloud2[pcInd];
      double Wf[5][1];
      double f[10][1];
      for (i=0;i<10;i++){
         f[i][0]  = pc->features[i];
      }
      mat_mult((double *) W, 5,10,(double *)f,10,1,(double *)Wf);

      //which node does it predict?
      int max_element = 6;
      double max_element_value  =0.0;
      for (i=0;i<5;i++){
         if (Wf[i][0] > max_element_value){
            max_element_value = Wf[i][0];
            max_element = i;
         }
      }

      switch (max_element){
      case 0:
         pc->learned_label = NODE_VEG;
         break;
      case 1:
         pc->learned_label = NODE_WIRE;
         break;
      case 2:
         pc->learned_label = NODE_POLE;
         break;
      case 3:
         pc->learned_label = NODE_GROUND;
         break;
```

```
      case 4:
         pc->learned_label = NODE_FACADE;
         break;
      }
   }

   viewer1.show();
   viewer1_learned.show();

   viewer2.show();
   viewer2_learned.show();

   return a.exec();
}

void findCentroid1(){
   unsigned long pcInd = 0;
   double sum_pos[3];
   PointCloud *pc;
   sum_pos[0] = 0.0; sum_pos[1] = 0.0; sum_pos[2] = 0.0;
   for (pcInd=0; pcInd < npoints1; pcInd++ ){
      pc = &pointCloud1[pcInd];
      sum_pos[0] += pc->pos[0];
      sum_pos[1] += pc->pos[1];
      sum_pos[2] += pc->pos[2];
   }

   pos_cent1[0] = sum_pos[0]/npoints1;
   pos_cent1[1] = sum_pos[1]/npoints1;
   pos_cent1[2] = sum_pos[2]/npoints1;
}

void findCentroid2(){
   unsigned long pcInd = 0;
   double sum_pos[3];
   PointCloud *pc;
   sum_pos[0] = 0.0; sum_pos[1] = 0.0; sum_pos[2] = 0.0;
   for (pcInd=0; pcInd < npoints2; pcInd++ ){
      pc = &pointCloud2[pcInd];
      sum_pos[0] += pc->pos[0];
      sum_pos[1] += pc->pos[1];
      sum_pos[2] += pc->pos[2];
   }

   pos_cent2[0] = sum_pos[0]/npoints2;
   pos_cent2[1] = sum_pos[1]/npoints2;
   pos_cent2[2] = sum_pos[2]/npoints2;
}

void readPointClouds1(char *fileName){

   FILE *fp;
   if((fp = fopen(fileName, "rt")) == NULL) {
      fprintf(stderr, "# Could not open file %s\n", fileName);
      return ;
   }

   unsigned long pcInd = 0;
   unsigned short dummy;
   int done =0;
   PointCloud *pc;
   while(!done){

      pc = &pointCloud1[pcInd];

      fscanf(fp,"%f %f %f %d %d %f %f %f %f %f %f %f %f %f %f",
           &pc->pos[0],&pc->pos[1],&pc->pos[2],&dummy,&pc->node_label,
            &pc->features[0],&pc->features[1],&pc->features[2],
            &pc->features[3],&pc->features[4],&pc->features[5],
            &pc->features[6],&pc->features[7],&pc->features[8],&pc->features[9]);

      switch (pc->node_label){
      case NODE_VEG:
         pc->node_vec[0] = 1.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_WIRE:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 1.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_POLE:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 1.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_GROUND:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 1.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_FACADE:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 1.0;
         break;
      }

      pcInd++;

      if (pcInd > 89821){ //am rf
         done = 1;
      }
   }

   npoints1 = pcInd;
}

void readPointClouds2(char *fileName){

   FILE *fp;
   if((fp = fopen(fileName, "rt")) == NULL) {
      fprintf(stderr, "# Could not open file %s\n", fileName);
      return ;
   }

   unsigned long pcInd = 0;
   unsigned short dummy;
   int done =0;
   PointCloud *pc;
   while(!done){

      pc = &pointCloud2[pcInd];

      fscanf(fp,"%f %f %f %d %d %f %f %f %f %f %f %f %f %f %f",
           &pc->pos[0],&pc->pos[1],&pc->pos[2],&dummy,&pc->node_label,
            &pc->features[0],&pc->features[1],&pc->features[2],
            &pc->features[3],&pc->features[4],&pc->features[5],
            &pc->features[6],&pc->features[7],&pc->features[8],&pc->features[9]);

      switch (pc->node_label){
      case NODE_VEG:
         pc->node_vec[0] = 1.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_WIRE:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 1.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_POLE:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 1.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_GROUND:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 1.0; pc->node_vec[4] = 0.0;
         break;
      case NODE_FACADE:
         pc->node_vec[0] = 0.0; pc->node_vec[1] = 0.0; pc->node_vec[2] = 0.0; pc-
>node_vec[3] = 0.0; pc->node_vec[4] = 1.0;
         break;
      }

      pcInd++;

      if (pcInd > 36396){ //an rf
         done = 1;
      }
   }

   npoints2 = pcInd;
}

//add noise to second dataset
void addNoise2(){
//first find mean of the features
   double sum_features2[10];
   unsigned short i;
   unsigned long pcInd = 0;
   PointCloud *pc;

   //initialize to 0
   for (i=0;i<10;i++){
      sum_features2[i] = 0.0;
   }

   //cumulative sum
   for (pcInd=0; pcInd < npoints2; pcInd++ ){
      pc = &pointCloud2[pcInd];
      for (i=0;i<10;i++){
         sum_features2[i] += pc->features[i];
      }
   }

   //find the mean
   for (i=0;i<10;i++){
      mean_features2[i] =  sum_features2[i]/npoints2;
      //qDebug() << mean_features2[i];
   }

   //initialize to 0
   for (i=0;i<10;i++){
      sum_features2[i] = 0.0;
   }

   //cumulative sum of variance
   for (pcInd=0; pcInd < npoints2; pcInd++ ){
      pc = &pointCloud2[pcInd];
      for (i=0;i<10;i++){
         sum_features2[i] += (pc->features[i] - mean_features2[i])*(pc->features[i]
- mean_features2[i]) ;
      }
   }

   //find the variance
  for (i=0;i<10;i++){
     stddev_features2[i] =  sqrt(sum_features2[i]/npoints2);
  }

  //now...add the noise
  for (pcInd=0; pcInd < npoints2; pcInd++ ){
     pc = &pointCloud2[pcInd];
     for (i=0;i<10;i++){
        std::normal_distribution<double> distribution1(
mean_features2[i],stddev_features2[i]);
        pc->features[i] += distribution1(generator);
     }
  }

}
// Draws a spiral
void Viewer::draw()
{
   unsigned long npoints;
   short drawingLabel;
   // Draw an axis using the QGLViewer static function
   glClearColor (0.0,0.0,0.0,1.0);

   PointCloud *pc;
   unsigned long pcInd = 0;

   glPointSize(2.0f);

   glBegin(GL_POINTS);

   switch (drawingData){
   case 1:
      npoints = npoints1;
      break;
   case 2:
      npoints = npoints2;
      break;
   }

   for (pcInd=0; pcInd < npoints; pcInd++ ){

      switch (drawingData){
      case 1:
         pc = &pointCloud1[pcInd];
         break;
      case 2:
         pc = &pointCloud2[pcInd];
         break;
      }

      switch(drawMode){
      case 0:
      drawingLabel = pc->node_label;
         break;
      case 1:
      drawingLabel = pc->learned_label;
         break;
      }

      switch (drawingLabel){
      case NODE_VEG:
         glColor4f(0.0, 1.0f , 0.0f,1.0f);
         break;
      case NODE_WIRE:
         glColor4f(0.2, 0.2f , 0.2f,1.0f);
         break;
      case NODE_POLE:
         glColor4f(1.0, 1.0f , 1.0f,1.0f);
         break;
      case NODE_GROUND:
         glColor4f(0.5, 0.27f , 0.07f,1.0f);
         break;
      case NODE_FACADE:
         glColor4f(1.0, 0.89f , 0.77f,1.0f);
         break;
      }

      switch (drawingData){
      case 1:
         glVertex3f(pc->pos[0]-pos_cent1[0], pc->pos[1]-pos_cent1[1], pc->pos[2]-
pos_cent1[2]);
         break;
      case 2:
         glVertex3f(pc->pos[0]-pos_cent2[0], pc->pos[1]-pos_cent2[1], pc->pos[2]-
pos_cent2[2]);
         break;
      }
         }
   glEnd();

}

#ifndef LAB2_H
#define LAB2_H

#define NODE_VEG 1004
```

```
#define NODE_WIRE 1100
#define NODE_POLE 1103
#define NODE_GROUND 1200
#define NODE_FACADE 1400

extern struct PointCloud{
  float pos[3];
  short node_label;
  float node_vec[5];
  float learned_label;
  float features[10];
} pointCloud1[90000],pointCloud2[37000],inpPointCloud[90000];

extern unsigned long npoints1;
extern unsigned long npoints2;
extern double pos_cent1[3];
extern double pos_cent2[3];
void readPointClouds1(char *fileName);
void readPointClouds2(char *fileName);
void findCentroid1();
void findCentroid2();
void addNoise2();

void mat_init( double *in, int rows, int cols, double init_val );
void mat_transpose( double *A, int na, int ma, double *C );
void mat_mult( double *A, int na, int ma,
              double *B, int nb, int mb,
              double *C );
void mat_sub( double *A, int na, int ma, double *B, double *C );


#ifndef VIEWER_H
#define VIEWER_H

#include <QGLViewer/qglviewer.h>
#include <QGLViewer/manipulatedFrame.h>

class Viewer : public QGLViewer
{
public :
    int drawMode; //0 original 1 learned
    int drawingData; //1 am 2 an
protected :
  virtual void draw();
  virtual void init();
  virtual QString helpString() const;
  virtual void postDraw();
 private :
    void drawCornerAxis();
};

#endif // VIEWER_H
```