ПРИЛОЖЕНИЕ В

(*обязательное*)

Исходный текст типа ROCDialog

ROCDialog::ROCDialog(QString fileName, QVector<Seed> seedVect, QWidget \*parent) :

QDialog(parent),

ui(new Ui::ROCDialog)

{

ui->setupUi(this);

this->fileName = QString(fileName);

seedVector = QVector<Seed>(seedVect);

this->TP = QVector<double>();

this->FP = QVector<double>();

}

ROCDialog::~ROCDialog()

{

delete ui;

}

void ROCDialog::drawRocCurve(int posclass)

{

double labels[27] = { 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0 };

float MAXprobability = 0, MINprobability = seedVector[0].probability;

int n = seedVector.length();

float N=0, P=0;

for(int i = 0; i < n ; i++) {

if(labels[i] == posclass ) P++;

else N++;

}

if( N == 0 || P == 0 )

cout << "I only found class 1 in the labels vector ...\n";

for( int i = 0; i < n; i++ )

{

if(MAXprobability < seedVector[i].probability)

MAXprobability = seedVector[i].probability;

if(MINprobability > seedVector[i].probability)

MINprobability = seedVector[i].probability;

}

float t = MINprobability;

float TP = 0, FP = 0;

do

{

TP = 0, FP = 0;

for(int i = 0; i < n; i++)

{

if(seedVector[i].probability >= t)

{

if(seedVector[i].GetCluster() == posclass)

TP++;

else

FP++;

}

}

float tp = TP/P, fp = FP/N;

this->TP.push\_back(tp);

this->FP.push\_back(fp);

t += 0.01;

} while(t < MAXprobability);

std::reverse(this->TP.begin(), this->TP.end());

std::reverse(this->FP.begin(), this->FP.end());

for(int j = 0; j < this->TP.length(); j++)

{

cout << this->FP[j] << " " << this->TP[j] << "\n";

}

makePlot();

}

void ROCDialog::calculateROCparemeters()

{

int \*idealClusterData = new int[seedVector.length()];

if(fileName.count("sds.png") > 0)

{

int idealClusterDataStatic[27] = { 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0 };

for(int i = 0; i < seedVector.length(); i++)

{

idealClusterData[i] = idealClusterDataStatic[i];

}

}

if(fileName.count("lampSeed.png") > 0)

{

int idealClusterDataStatic[81] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1};

for(int i = 0; i < seedVector.length(); i++)

{

idealClusterData[i] = idealClusterDataStatic[i];

}

}

if(fileName.count("sds.png") > 0 || fileName.count("lampSeed.png") > 0)

{

ROCforCluster(idealClusterData);

this->show();

}

}

void ROCDialog::ROCforCluster(int \*labels)

{

int n = seedVector.length();

double scores[n];

double scores2[n];

for(int j = 0; j < n; j++)

{

scores[j] = seedVector[j].probability;

scores2[j] = 1 - seedVector[j].probability;

}

calcAUC(labels, scores, seedVector.length(), 0);

calcAUC(labels, scores2, seedVector.length(), 1);

makePlot();

}

double ROCDialog::calcAUC(int\* labels, double \* scores,int n,int posclass)

{

typedef QPair<float,int> mypair;

QVector<mypair> L(n);

for(int i = 0; i < n; i++) {

L[i].first = scores[i];

L[i].second = labels[i];

}

qSort(L.begin(),L.end());

std::reverse(L.begin(), L.end());

/\* Count number of positive and negative examples first \*/

float N=0, P=0;

for(int i = 0; i < n ; i++) {

if(labels[i] == posclass ) P++;

else N++;

}

if( N == 0 || P == 0 )

cout << "I only found class 1 in the labels vector ...\n";

/\* Then calculate the actual are under the ROC curve \*/

double A = 0;

double fprev = INT\_MIN; //-infinity

double FP = 0,

TP = 0,

FPprev = 0,

TPprev = 0;

for(int i = 0 ; i < n; i++) {

double fi = L[i].first;

double label= L[i].second;

if(fi != fprev) {

/\* Divide area here already : a bit slower, but gains in precision and avoids overflows \*/

A = A + (trapezoidArea(FP\*1.0/N,FPprev\*1.0/N,TP\*1.0/P,TPprev\*1.0/P));

fprev = fi;

FPprev = FP;

TPprev = TP;

if(posclass == 0)

{

this->TP.push\_back(TP/P);

this->FP.push\_back(FP/N);

}

else {

this->TP2.push\_back(TP/P);

this->FP2.push\_back(FP/N);

}

}

if(label == posclass)

TP = TP + 1;

else

FP = FP + 1;

}

if(posclass == 0)

{

this->TP.push\_back(TP/P);

this->FP.push\_back(FP/N);

}

else {

this->TP2.push\_back(TP/P);

this->FP2.push\_back(FP/N);

}

QVector<QPair<float, float> > a = QVector<QPair<float, float> >(this->TP.length());

for(int j = 0; j < this->TP.length(); j++)

{

if(posclass == 0)

{

a[j].first = this->TP[j];

a[j].second = this->FP[j];

}

else {

a[j].first = this->TP2[j];

a[j].second = this->FP2[j];

}

}

smoothing(a);

qSort(a.begin(),a.end());

std::reverse(a.begin(), a.end());

if(posclass == 0)

{

this->a1 = QVector<QPair<float, float> > (a);

} else {

this->a2 = QVector<QPair<float, float> > (a);

}

if(posclass == 0)

{

this->TP.clear();

this->FP.clear();

}

else {

this->TP2.clear();

this->FP2.clear();

}

for(int j = 0; j < a.length(); j++)

{

if(posclass == 0)

{

this->TP.push\_back(a[j].first);

this->FP.push\_back(a[j].second);

}

else {

this->TP2.push\_back(a[j].first);

this->FP2.push\_back(a[j].second);

}

}

A = A + trapezoidArea(1.0,FPprev\*1.0/N,1.0,TPprev\*1.0/P);

return A;

}

double ROCDialog::trapezoidArea(double X1, double X2, double Y1, double Y2) {

double base = std::abs(X1-X2);

double height = (Y1+Y2)/2.0;

return (base \* height);

}

void ROCDialog::smoothing(QVector<QPair<float, float> > &a)

{

//How many neighbours to smooth

int NO\_OF\_NEIGHBOURS=3;

QVector<QPair<float, float> > tmp=a;

for(int i=0;i<a.size()-1;i++)

{

if(i+NO\_OF\_NEIGHBOURS+1<a.size())

{

for(int j=1;j<NO\_OF\_NEIGHBOURS;j++)

{

a[i].first += a[i+j].first;

a[i].second += a[i+j].second;

}

a[i].first /= NO\_OF\_NEIGHBOURS;

a[i].second /= NO\_OF\_NEIGHBOURS;

}

else

{

for(int j=1;j<NO\_OF\_NEIGHBOURS;j++)

{

a[i].first += tmp[i-j].first;

a[i].second += tmp[i-j].second;

}

a[i].first /= NO\_OF\_NEIGHBOURS;

a[i].second /= NO\_OF\_NEIGHBOURS;

}

}

for(int i=1;i<a.size()-1;i++)

{

if(a[i].first < a[i-1].first)

a[i].first = a[i-1].first+0.1;

if(a[i].second < a[i-1].second)

a[i].second = a[i-1].second+0.1;

}

}