ПРИЛОЖЕНИЕ А

(*обязательное*)

Исходный текст типа SVMclassifier

SVMclassifier::SVMclassifier(QVector<Seed> seedVector, QVector<int> featVector, int clusters, QVector<QVector<int> > trainDataObj) : QObject(0)

{

seedVect = QVector<Seed>(seedVector);

featVect = QVector<int>(featVector);

trainigDataObjs = QVector<QVector<int> >(trainDataObj);

}

void SVMclassifier::FillTrainingMat()

{

// AREA = 0, PERIMETR, COMPACTNESS, ELONGATION, LUMA, CONTRAST, HOMOGENEITY, DISSIMILARITY, ENERGY, ENTROPY, CORRELATION, MATEXPECT, DISPERTION

int indexOfObject = 0;

int countOfOb = 0;

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

for(int j = 0; j < trainigDataObjs[i].length(); j++)

countOfOb++;

float labels[countOfOb];

float trainingData[countOfOb][featVect.length()]; //= new float\*[countOfOb];//[featVect.length()]

int k = 0;

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

for(int j = 0; j < trainigDataObjs[i].length(); j++)

{

labels[indexOfObject++] = i;

fillObject(trainingData[k++], trainigDataObjs[i][j]);

}

Mat labelsMat(countOfOb, 1, CV\_32FC1, labels);

Mat trainingDataMat(countOfOb, featVect.length(), CV\_32FC1, trainingData);

// Set up SVM's parameters

CvSVMParams params;

params.svm\_type = CvSVM::C\_SVC;

params.kernel\_type = CvSVM::LINEAR;

params.term\_crit = cvTermCriteria(CV\_TERMCRIT\_ITER, 100, 1e-6);

// Train the SVM

CvSVM SVM;

SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);

int featureLength = featVect.length();

float object[featureLength];

float confidence;

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

{

fillObject(object, i);

Mat objectMat(1, featureLength, CV\_32FC1, object);

float cl2 = SVM.predict(objectMat, true);

confidence = 1.0 / (1.0 + exp(-cl2));

float cl = SVM.predict(objectMat);

seedVect[i].SetCluster(cl);

seedVect[i].probability = confidence;

}

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

// cout << "area = " <<seedVect[i].GetArea() <<" luma = " <<seedVect[i].GetLuma() << " matexp = " << seedVect[i].matExpect

// <<" centre = " <<seedVect[i].centerMass << " ellong = " << seedVect[i].elongation << " cluster = " <<seedVect[i].GetCluster() <<"\n ";

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

cout << seedVect[i].GetCluster() << ", ";

}

void SVMclassifier::fillObject(float \*arr, int numberOfSeed)

{

int l = 0;

if((featVect.indexOf(AREA)) != -1)

arr[l++] = seedVect[numberOfSeed].GetArea();

if((featVect.indexOf(COMPACTNESS)) != -1)

arr[l++] = seedVect[numberOfSeed].GetCompactness();

if((featVect.indexOf(PERIMETR)) != -1)

arr[l++] = seedVect[numberOfSeed].GetPerimetr();

if((featVect.indexOf(ELONGATION)) != -1)

arr[l++] = seedVect[numberOfSeed].elongation;

if((featVect.indexOf(LUMA)) != -1)

arr[l++] = seedVect[numberOfSeed].GetLuma();

if((featVect.indexOf(CONTRAST)) != -1)

arr[l++] = seedVect[numberOfSeed].contrast;

if((featVect.indexOf(HOMOGENEITY)) != -1)

arr[l++] = seedVect[numberOfSeed].homogeneity;

if((featVect.indexOf(DISSIMILARITY)) != -1)

arr[l++] = seedVect[numberOfSeed].dissimilarity;

if((featVect.indexOf(ENERGY)) != -1)

arr[l++] = seedVect[numberOfSeed].energy;

if((featVect.indexOf(ENTROPY)) != -1)

arr[l++] = seedVect[numberOfSeed].entropy;

if((featVect.indexOf(CORRELATION)) != -1)

arr[l++] = seedVect[numberOfSeed].correlation;

if((featVect.indexOf(MATEXPECT)) != -1)

arr[l++] = seedVect[numberOfSeed].matExpect;

if((featVect.indexOf(DISPERTION)) != -1)

arr[l++] = seedVect[numberOfSeed].dispersion;

}

QVector<Seed> SVMclassifier::GetSeedVector()

{

return seedVect;

}