**OpenCV3.0中机器学习模块的使用**

         在OpenCV3.0中，所有的机器学习类都派生于StatModel类，该类规定了所有机器学习类的通用方法的调用形式，例如：训练和预测。OpenCV3.0实现了归一化贝叶斯分类器、K最近邻算法、支持向量机、决策树、自适应Boost算法、GBT( gradient boosted trees )、随机林木、人工神经网络、EM算法这些机器学习算法，它们都有相同的创建、训练、分类调用形式，比OpenCV2.0更加规范，易于学习。

一、训练数据的准备

具体参数的说明可以参考源代码。典型的创建示例如下：

TrainData::create(samples, ROW\_SAMPLE, responses));

该代码返回一个Ptr<TrainData>，即TrainData指针对象。代码中，samples类型必须为CV\_32F，每一行为一个样本，每一列对应样本的一个特征。responses为与样本对应的类别矩阵，类型必须为CV\_32F或CV\_32S。

二、训练

调用train函数，原型为：

bool train( const Ptr<TrainData>& trainData, int flags=0 );

只要将创建的训练数据直接传递给train函数即可。

三、识别

调用predict函数，原型为：

float predict( InputArray samples, OutputArray results=noArray(), int flags=0 )

将单个样本传递给该函数，即可返回预测结果。

以下的代码，详细演示了各种分类器的创建、设置和使用方法：

#include "opencv2/core.hpp"

#include "opencv2/imgproc.hpp"

#include "opencv2/ml.hpp"

#include "opencv2/highgui.hpp"

#ifdef HAVE\_OPENCV\_OCL

#define \_OCL\_KNN\_ 1 // select whether using ocl::KNN method or not, default is using

#define \_OCL\_SVM\_ 1 // select whether using ocl::svm method or not, default is using

#include "opencv2/ocl/ocl.hpp"

#endif

#include <stdio.h>

using namespace std;

using namespace cv;

using namespace cv::ml;

const Scalar WHITE\_COLOR = Scalar(255,255,255);

const string winName = "points";

const int testStep = 5;

Mat img, imgDst;

RNG rng;

vector<Point>  trainedPoints;

vector<int>    trainedPointsMarkers;

const int MAX\_CLASSES = 2;

vector<Vec3b>  classColors(MAX\_CLASSES);

int currentClass = 0;

vector<int> classCounters(MAX\_CLASSES);

#define \_NBC\_ 1 // normal Bayessian classifier

#define \_KNN\_ 1 // k nearest neighbors classifier

#define \_SVM\_ 1 // support vectors machine

#define \_DT\_  1 // decision tree

#define \_BT\_  1 // ADA Boost

#define \_GBT\_ 0 // gradient boosted trees

#define \_RF\_  1 // random forest

#define \_ANN\_ 1 // artificial neural networks

#define \_EM\_  1 // expectation-maximization

static void on\_mouse( int event, int x, int y, int /\*flags\*/, void\* )

{

    if( img.empty() )

        return;

    int updateFlag = 0;

    if( event == EVENT\_LBUTTONUP )

    {

        trainedPoints.push\_back( Point(x,y) );

        trainedPointsMarkers.push\_back( currentClass );

        classCounters[currentClass]++;

        updateFlag = true;

    }

    //draw

    if( updateFlag )

    {

        img = Scalar::all(0);

        // draw points

        for( size\_t i = 0; i < trainedPoints.size(); i++ )

        {

            Vec3b c = classColors[trainedPointsMarkers[i]];

            circle( img, trainedPoints[i], 5, Scalar(c), -1 );

        }

        imshow( winName, img );

   }

}

static Mat prepare\_train\_samples(const vector<Point>& pts)

{

    Mat samples;

    Mat(pts).reshape(1, (int)pts.size()).convertTo(samples, CV\_32F);

    return samples;

}

static Ptr<TrainData> prepare\_train\_data()

{

    Mat samples = prepare\_train\_samples(trainedPoints);

    return TrainData::create(samples, ROW\_SAMPLE, Mat(trainedPointsMarkers));

}

static void predict\_and\_paint(const Ptr<StatModel>& model, Mat& dst)

{

    Mat testSample( 1, 2, CV\_32FC1 );

    for( int y = 0; y < img.rows; y += testStep )

    {

        for( int x = 0; x < img.cols; x += testStep )

        {

            testSample.at<float>(0) = (float)x;

            testSample.at<float>(1) = (float)y;

            int response = (int)model->predict( testSample );

            dst.at<Vec3b>(y, x) = classColors[response];

        }

    }

}

#if \_NBC\_

static void find\_decision\_boundary\_NBC()

{

    // learn classifier

    Ptr<NormalBayesClassifier> normalBayesClassifier = StatModel::train<NormalBayesClassifier>(prepare\_train\_data());

    predict\_and\_paint(normalBayesClassifier, imgDst);

}

#endif

#if \_KNN\_

static void find\_decision\_boundary\_KNN( int K )

{

    Ptr<KNearest> knn = KNearest::create();

    knn->setDefaultK(K);

    knn->setIsClassifier(true);

    knn->train(prepare\_train\_data());

    predict\_and\_paint(knn, imgDst);

}

#endif

#if \_SVM\_

static void find\_decision\_boundary\_SVM( double C )

{

    Ptr<SVM> svm = SVM::create();

    svm->setType(SVM::C\_SVC);

    svm->setKernel(SVM::POLY); //SVM::LINEAR;

    svm->setDegree(0.5);

    svm->setGamma(1);

    svm->setCoef0(1);

    svm->setNu(0.5);

    svm->setP(0);

    svm->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER+TermCriteria::EPS, 1000, 0.01));

    svm->setC(C);

    svm->train(prepare\_train\_data());

    predict\_and\_paint(svm, imgDst);

    Mat sv = svm->getSupportVectors();

    for( int i = 0; i < sv.rows; i++ )

    {

        const float\* supportVector = sv.ptr<float>(i);

        circle( imgDst, Point(saturate\_cast<int>(supportVector[0]),saturate\_cast<int>(supportVector[1])), 5, Scalar(255,255,255), -1 );

    }

}

#endif

#if \_DT\_

static void find\_decision\_boundary\_DT()

{

    Ptr<DTrees> dtree = DTrees::create();

    dtree->setMaxDepth(8);

    dtree->setMinSampleCount(2);

    dtree->setUseSurrogates(false);

    dtree->setCVFolds(0); // the number of cross-validation folds

    dtree->setUse1SERule(false);

    dtree->setTruncatePrunedTree(false);

    dtree->train(prepare\_train\_data());

    predict\_and\_paint(dtree, imgDst);

}

#endif

#if \_BT\_

static void find\_decision\_boundary\_BT()

{

    Ptr<Boost> boost = Boost::create();

    boost->setBoostType(Boost::DISCRETE);

    boost->setWeakCount(100);

    boost->setWeightTrimRate(0.95);

    boost->setMaxDepth(2);

    boost->setUseSurrogates(false);

    boost->setPriors(Mat());

    boost->train(prepare\_train\_data());

    predict\_and\_paint(boost, imgDst);

}

#endif

#if \_GBT\_

static void find\_decision\_boundary\_GBT()

{

    GBTrees::Params params( GBTrees::DEVIANCE\_LOSS, // loss\_function\_type

                         100, // weak\_count

                         0.1f, // shrinkage

                         1.0f, // subsample\_portion

                         2, // max\_depth

                         false // use\_surrogates )

                         );

    Ptr<GBTrees> gbtrees = StatModel::train<GBTrees>(prepare\_train\_data(), params);

    predict\_and\_paint(gbtrees, imgDst);

}

#endif

#if \_RF\_

static void find\_decision\_boundary\_RF()

{

    Ptr<RTrees> rtrees = RTrees::create();

    rtrees->setMaxDepth(4);

    rtrees->setMinSampleCount(2);

    rtrees->setRegressionAccuracy(0.f);

    rtrees->setUseSurrogates(false);

    rtrees->setMaxCategories(16);

    rtrees->setPriors(Mat());

    rtrees->setCalculateVarImportance(false);

    rtrees->setActiveVarCount(1);

    rtrees->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER, 5, 0));

    rtrees->train(prepare\_train\_data());

    predict\_and\_paint(rtrees, imgDst);

}

#endif

#if \_ANN\_

static void find\_decision\_boundary\_ANN( const Mat&  layer\_sizes )

{

    Mat trainClasses = Mat::zeros( (int)trainedPoints.size(), (int)classColors.size(), CV\_32FC1 );

    for( int i = 0; i < trainClasses.rows; i++ )

    {

        trainClasses.at<float>(i, trainedPointsMarkers[i]) = 1.f;

    }

    Mat samples = prepare\_train\_samples(trainedPoints);

    Ptr<TrainData> tdata = TrainData::create(samples, ROW\_SAMPLE, trainClasses);

    Ptr<ANN\_MLP> ann = ANN\_MLP::create();

    ann->setLayerSizes(layer\_sizes);

    ann->setActivationFunction(ANN\_MLP::SIGMOID\_SYM, 1, 1);

    ann->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER+TermCriteria::EPS, 300, FLT\_EPSILON));

    ann->setTrainMethod(ANN\_MLP::BACKPROP, 0.001);

    ann->train(tdata);

    predict\_and\_paint(ann, imgDst);

}

#endif

#if \_EM\_

static void find\_decision\_boundary\_EM()

{

    img.copyTo( imgDst );

    Mat samples = prepare\_train\_samples(trainedPoints);

    int i, j, nmodels = (int)classColors.size();

    vector<Ptr<EM> > em\_models(nmodels);

    Mat modelSamples;

    for( i = 0; i < nmodels; i++ )

    {

        const int componentCount = 3;

        modelSamples.release();

        for( j = 0; j < samples.rows; j++ )

        {

            if( trainedPointsMarkers[j] == i )

                modelSamples.push\_back(samples.row(j));

        }

        // learn models

        if( !modelSamples.empty() )

        {

            Ptr<EM> em = EM::create();

            em->setClustersNumber(componentCount);

            em->setCovarianceMatrixType(EM::COV\_MAT\_DIAGONAL);

            em->trainEM(modelSamples, noArray(), noArray(), noArray());

            em\_models[i] = em;

        }

    }

    // classify coordinate plane points using the bayes classifier, i.e.

    // y(x) = arg max\_i=1\_modelsCount likelihoods\_i(x)

    Mat testSample(1, 2, CV\_32FC1 );

    Mat logLikelihoods(1, nmodels, CV\_64FC1, Scalar(-DBL\_MAX));

    for( int y = 0; y < img.rows; y += testStep )

    {

        for( int x = 0; x < img.cols; x += testStep )

        {

            testSample.at<float>(0) = (float)x;

            testSample.at<float>(1) = (float)y;

            for( i = 0; i < nmodels; i++ )

            {

                if( !em\_models[i].empty() )

                    logLikelihoods.at<double>(i) = em\_models[i]->predict2(testSample, noArray())[0];

            }

            Point maxLoc;

            minMaxLoc(logLikelihoods, 0, 0, 0, &maxLoc);

            imgDst.at<Vec3b>(y, x) = classColors[maxLoc.x];

        }

    }

}

#endif

int main()

{

    cout << "Use:" << endl

         << "  key '0' .. '1' - switch to class #n" << endl

         << "  left mouse button - to add new point;" << endl

         << "  key 'r' - to run the ML model;" << endl

         << "  key 'i' - to init (clear) the data." << endl << endl;

    cv::namedWindow( "points", 1 );

    img.create( 480, 640, CV\_8UC3 );

    imgDst.create( 480, 640, CV\_8UC3 );

    imshow( "points", img );

    setMouseCallback( "points", on\_mouse );

    classColors[0] = Vec3b(0, 255, 0);

    classColors[1] = Vec3b(0, 0, 255);

    for(;;)

    {

        uchar key = (uchar)waitKey();

        if( key == 27 ) break;

        if( key == 'i' ) // init

        {

            img = Scalar::all(0);

            trainedPoints.clear();

            trainedPointsMarkers.clear();

            classCounters.assign(MAX\_CLASSES, 0);

            imshow( winName, img );

        }

        if( key == '0' || key == '1' )

        {

            currentClass = key - '0';

        }

        if( key == 'r' ) // run

        {

            double minVal = 0;

            minMaxLoc(classCounters, &minVal, 0, 0, 0);

            if( minVal == 0 )

            {

                printf("each class should have at least 1 point\n");

                continue;

            }

            img.copyTo( imgDst );

#if \_NBC\_

            find\_decision\_boundary\_NBC();

            imshow( "NormalBayesClassifier", imgDst );

#endif

#if \_KNN\_

            find\_decision\_boundary\_KNN( 3 );

            imshow( "kNN", imgDst );

            find\_decision\_boundary\_KNN( 15 );

            imshow( "kNN2", imgDst );

#endif

#if \_SVM\_

            //(1)-(2)separable and not sets

            find\_decision\_boundary\_SVM( 1 );

            imshow( "classificationSVM1", imgDst );

            find\_decision\_boundary\_SVM( 10 );

            imshow( "classificationSVM2", imgDst );

#endif

#if \_DT\_

            find\_decision\_boundary\_DT();

            imshow( "DT", imgDst );

#endif

#if \_BT\_

            find\_decision\_boundary\_BT();

            imshow( "BT", imgDst);

#endif

#if \_GBT\_

            find\_decision\_boundary\_GBT();

            imshow( "GBT", imgDst);

#endif

#if \_RF\_

            find\_decision\_boundary\_RF();

            imshow( "RF", imgDst);

#endif

#if \_ANN\_

            Mat layer\_sizes1( 1, 3, CV\_32SC1 );

            layer\_sizes1.at<int>(0) = 2;

            layer\_sizes1.at<int>(1) = 5;

            layer\_sizes1.at<int>(2) = (int)classColors.size();

            find\_decision\_boundary\_ANN( layer\_sizes1 );

            imshow( "ANN", imgDst );

#endif

#if \_EM\_

            find\_decision\_boundary\_EM();

            imshow( "EM", imgDst );

#endif

        }

    }

    return 0;

}