算法设计与分析 project 报告

刘敏行 尚明月 王越

一、项目内容概述

实现一个手写数字识别系统，即对一个写有数字的图片，识别出这张图片上对应的数字。我们通过利用不同算法实现这一功能，比较了不同算法针对这一问题上准确率、复杂度的不同，并初步分析了原因。

二、使用算法

两种机器学习算法。

第一种：K-临近（KNN）算法，核心思路是如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别，从而完成分类。

第二种：支持向量机（SVM）算法，核心思路是在数据可以被线性可分时，算法通过最大化与支持向量的距离从而找到一条最有效的分界线；在数据线性不可分时，算法通过选取合适的函数将数据映射到更高维空间使之变为线性可分的，从而完成分类。

三、project 程序内容及分工

我们的工程（NumberParser）分为 5 个 cpp 文件：

MyKNN.cpp：用我们自己实现的 KNN 算法实现的数字识别系统

StandardKNN.cpp：以 OpenCV 自带的 KNN 算法为基础实现的数字识别系统

StandardSVM.cpp：以 OpenCV 自带的 SVM 算法为基础实现的数字识别系统

Parse.cpp：以 OpenCV 图像处理函数为基础的图片预处理函数

main.cpp：主程序，显示用户界面和手写板

其中，刘敏行主要负责 StandardKNN.cpp，StandardSVM.cpp，main.cpp 的编写，同时参与了 MyKNN.cpp 的编写；尚明月主要负责 MyKNN.cpp 的编写，同时参与了 main.cpp 的编写；王越主要负责 Parse.cpp 的编写，同时参与了 MyKNN.cpp 的编写，同时负责搜集图片测试样例。

四、程序细节介绍

1. MyKNN.cpp

MyKNN 采用 KNN 算法，其中在选取 K 个临近点时采用了 KD 树结构。

readFile 函数将机器学习所需数据库读入，并进行图片预处理，将图片特征值转换为 0-1 一维向量，作为 KNN 算法的参数。核心代码如下：

```
readin(data);     //以二进制只读方式打开 BMP 文件
changeto10();     //转换为 0-1 向量
remove_ilegalpoint();       //去除干扰点
n = atoi(data);
Number[n].num = n;
for (int i = 0; i < WEIGHT; i++)
    for (int j = 0; j < HEIGHT; j++)
        tem[i *WEIGHT + j] = DATAMASS[i][j];
Number[n].DATABIT.push_back(tem);
```

然后依据从训练集数据库中得到的数据为节点建立 kd 树。即 build_tree 函数。

kdd 函数读入一个测试集图片转换得到的向量，得到一个返回结果。

```
build_tree(node,0,leaf-1,1,0);
Node temp;
for(int j=0;j<dimen;++j)
    temp.x[j]=given[j];
search_tree(temp,1,0);
for(int j=0;j<m;++j)
{
    res[m-1-j]=pq.top();
    pq.pop();
}
return res[0].num;
```

其中 **search_tree 函数**是查找 kd 树的结点，是 kd 树数据结构的标准用法，此处不再赘述。

testcode 函数用来计算测试集的准确率。

2、StandardKNN.cpp

程序主要由四个函数支撑。

getdata 函数将机器学习所需的训练数据读入，并且进行图像预处理，同时将图像特征值转换为 0-1 一维向量，作为 KNN 算法的参数。核心代码如下：

```
src_image = cvLoadImage(file,0); //读入数据
prs_image = preprocessing(src_image, size, size,1); //预处理
//cvGetRow函数：将row和trainClasses联系起来（改一个会同时改另一个），下同
cvGetRow(trainClasses, &row, i*train_samples + j);
cvSet(&row, cvRealScalar(i));
cvGetRow(trainData, &row, i*train_samples + j);
IplImage* img = cvCreateImage( cvSize( size, size ), IPL_DEPTH_32F, 1 );
//将图片prs_image缩放后存进img
//0.0039215=1/255
cvConvertScale(&prs_image, img, 0.0039215, 0);
//将所需大小的图片放进data中
cvGetSubRect(img, &data, cvRect(0,0, size,size));
CvMat row_header, *row1;
//转换成1维向量
row1 = cvReshape( &data, &row_header, 0, 1 );
//存入数据矩阵中
cvCopy(row1, &row, NULL);
```

其中读入数据调用的OpenCV的图片加载函数，预处理则是用到了定义在Parse.cpp中的preproceesing函数，后面的建立特征矩阵的部分也是调用了OpenCV的相关图像处理函数。

train 函数读入训练数据集进行机器学习，由于主要的参数准备工作已经在之前做好了，此处只有一句话：

```
knn=new CvKNearest( trainData, trainClasses, 0, false, K );
```

此处是OpenCV自带的KNN算法实现函数，第一个参数是每个图片对应的特征矩阵，第二参数是每个图片的标签（即对应数字），中间两个参数是默认的，最后一个参数是考察的邻居数，此处我选择的是K=5

classify函数针对一个输入的图片，首先进行预处理，再返回识别的结果：

```
prs_image = preprocessing(img, size, size,3);
result=knn->find_nearest(row1,K,0,0,nearest,0);
```

其中返回结果调用的OpenCV的KNN类的find_nearest函数，得到预测结果。

test 函数读入测试数据集进行识别准确率测试，结构与 train 函数类似，只不过这里加入了 classify 函数的调用和统计错误率的部分：

```
prs_image = preprocessing(src_image, size, size,1);
float r=classify(&prs_image,0);
if((int)r!=i)
    error++;
testCount++;
```

3、StandardSVM.cpp

本程序的流程与 StandardKNN 流程极为相似，只不过我们在这里直接用了一份已经训练好的 XML 文档（OpenCV 的 SVM 要求载入数据格式是 XML），因此相对 KNN 算法部分省去了训练部分，而只有测试部分和识别分类部分

classify函数针对一个输入的图片，首先进行预处理，再返回识别的结果，与StandardKNN很相似：

```
cvZero(trainTempImg);
cvResize(test,trainTempImg);
HOGDescriptor *hog=new
  HOGDescriptor(cvSize(28,28),cvSize(14,14),cvSize(7,7),cvSize(7,7),9);
vector<float>descriptors;//存放结果
hog->compute(trainTempImg, descriptors,Size(1,1), Size(0,0)); //Hog特征计算
CvMat* SVMtrainMat=cvCreateMat(1,descriptors.size(),CV_32FC1);
int n=0;
for(vector<float>::iterator iter=descriptors.begin();iter!=descriptors.end();iter++)
```

```
{
    cvmSet(SVMtrainMat,0,n,*iter);
    n++;
}
float ret = svm.predict(SVMtrainMat);//检测结果
```

与KNN算法不同，SVM算法需要提取图片本身的特征，描述图片特征的一个很有效的特征是HOG特征，对输入图片进行Hog特征提取后，利用SVM库的predict函数直接进行识别预测。

test 函数功能与结构与 StandardKNN.cpp 中的 test 函数十分相似，这里不再赘述。

4、 Parse.cpp

这个程序主要是对输入的图片进行预处理，这个预处理比较有针对性，它保证输入给机器学习算法的图片中的数字是位于图片中央的，而不会蜷缩在图片的一角导致识别算法出错：

```
IplImage* result;
IplImage* scaledResult;

CvMat data;
CvMat dataA;
CvRect bb;

//找到边界框
bb=findBB(imgSrc);

cvGetSubRect(imgSrc, &data, cvRect(bb.x, bb.y, bb.width, bb.height));

int size=(bb.width>bb.height)?bb.width:bb.height;
result=cvCreateImage( cvSize( size, size ), 8, type );
cvSet(result,CV_RGB(255,255,255),NULL);
//将图像放在正中间，大小归一化
int x=(int)floor((float)(size-bb.width)/2.0f);
```

```
int y=(int)floor((float)(size-bb.height)/2.0f);
cvGetSubRect(result, &dataA, cvRect(x,y,bb.width, bb.height));
cvCopy(&data, &dataA, NULL);

scaledResult=cvCreateImage( cvSize( new_width, new_height ), 8, type );
cvResize(result, scaledResult, CV_INTER_NN);

return *scaledResult;
```

这个程序有三个函数，主函数 preprocessing 会调用寻找图片有效边框的子函数 findBB, 其再调用两个子函数，寻找 X 边界 findX 和寻找 Y 边界的 findY, 之后程序将裁切出来的有效区域扩展为原大小，并保证数字处于图片中央位置。需要声明的是，这个程序的大部分代码是借鉴了网上与此相关的程序，因此在此更多的细节不再涉及。

5、 main.cpp

这是工程的主函数，主要显示用户界面，显示手写画板，并根据用户输入不同调用对应的机器学习算法。

cpp 文件主要分为 3 大部分，第一大部分是基本环境设置，包括设置画板，提示输入等：

```
drawing=0;     //画画状态
r=5; //画笔半径
last_x=last_y=red=green=blue=0;
//创建窗口，大小 128*128
imagen=cvCreateImage(cvSize(128,128),8,3);
cvSet(imagen, CV_RGB(255,255,255),NULL);
screenBuffer=cvCloneImage(imagen);
cvNamedWindow( "手写板", 0 );
cvResizeWindow("手写板", 512,512);
//加入鼠标事件
cvSetMouseCallback("手写板",&on_mouse, 0 );
PrintWelcome();          //提示欢迎信息
```

第二大部分是响应鼠标事件，在画板上画图，此处主要借鉴了网上的相关内容，此处只写出与此相关的函数名，不再具体解释：

```
void draw(int x,int y)      //画图
void drawCursor(int x, int y)
void on_mouse( int event, int x, int y, int flags, void* param )      //鼠标事件
```

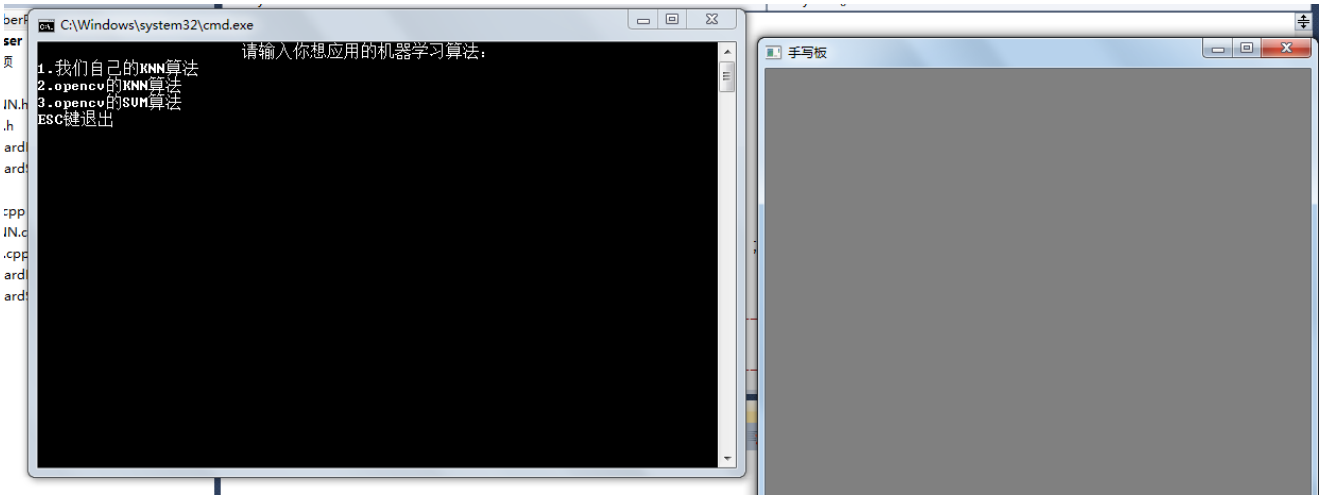
第三大部分是对三个机器学习算法的调用，三个函数 doMyKNN, doStandardKNN, doStandardSVM 分别调用三个机器学习算法，三个函数框架几乎相同，故在此不赘述过多细节。不过有一点要提到的是，自己实现的 KNN 算法和 OpenCV 库的 SVM 算法用的图片样例都是 3 通道的（bmp），而 OpenCV 的 KNN 算法用的图片样例是 1 通道的（pbm），另一方面，因此在三个函数中会针对图片格式和大小进行一些转换，如：

```
IplImage* temp;
temp = cvCreateImage( cvGetSize( imagen ), imagen->depth, 1 );
cvCvtColor( imagen, temp, CV_BGR2GRAY );
```

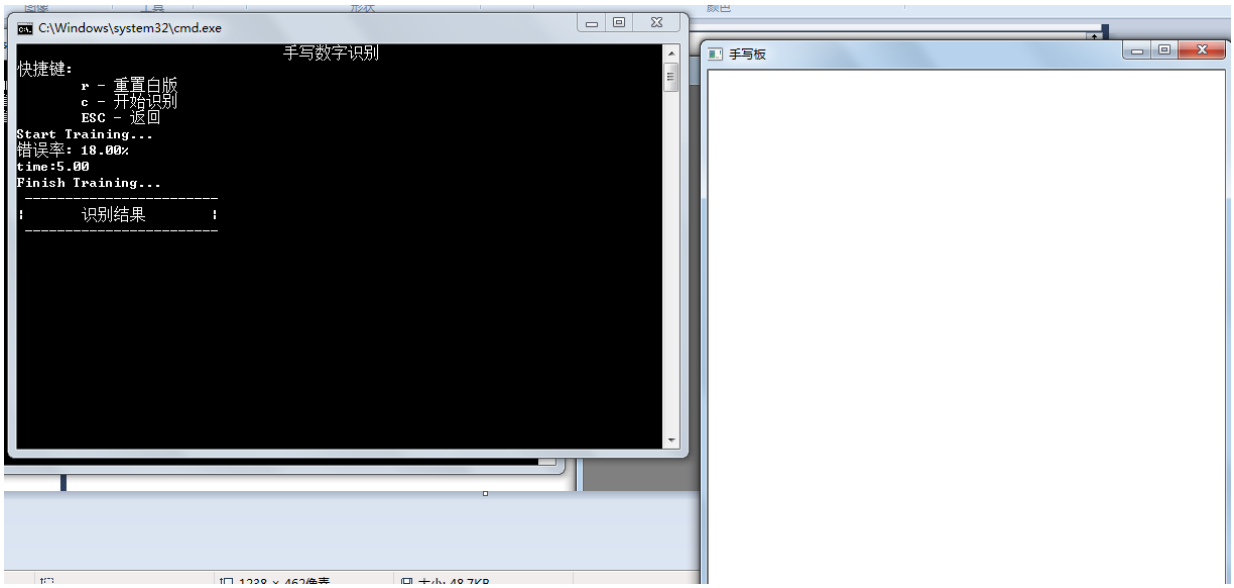
便将 3 通道 bmp 图片转换成了 1 通道 pbm 图片。

五、应用使用方法

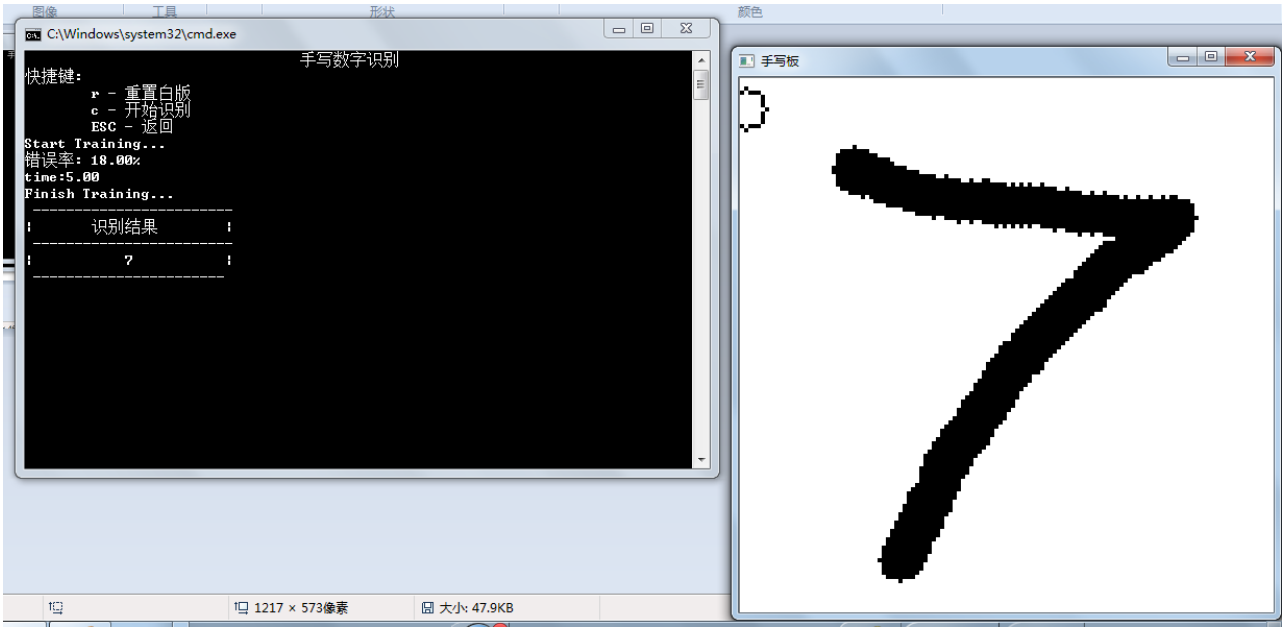
运行 NumberParser.exe，会进入如下界面：



根据提示输入想测试的算法，程序会先进行训练和内部数据的测试，之后，画板可以使用：



用户可以在画板上画一个数字，按‘c’键则进行识别，按’r’键则清除所画内容，按 ESC 键退出当前算法，可以选择测试其他算法。在画的过程中，可以通过’+’,’-‘键控制画笔的粗细。



六、程序运行情况及相关结论

评判一个机器学习算法，最重要的两个指标是识别时间和识别确率，我们也就这一点对三个算法做了评测比较。进行测试的运行环境是 Window7 系统，core i3 芯片，主频 2.3GHz，32 位系统，内存 4G。测试数据集是一定数量的 128*128 的写有数字的图片。通过变换训练数据集和测试数据集的比例，并同时针对三个算法测试识别时间和识别错误率，得到表格如下：（注：StandardSVM 训练集途径与两个 KNN 不同，因此它的训练集大小并没有改变，之

后会对此做解释）

训练集：测试集	测试数据识别时间(s)/识别错误率		
	MyKNN	StandardKNN	StandardSVM
200：800	79/24.00%	37/14.25%	14/16.00%
400：600	111/16.33%	45/9.00%	11/16.17%
500：500	114/13.10%	55/7.40%	9/16.10%
600：400	113/10.58%	41/7.15%	7/16.50%
800：200	79/10.50%	27/7.00%	4/15.80%

综合以上数据，我们可以得出一些初步的结论。对于任何一个比例的测试集，都可以发现是 MyKNN 算法耗时最长，StandardSVM 耗时最短，但准确率最差，而另一方面，StandardKNN 正确率最高。而随着测试集相对于训练集的比例越来越小，MyKNN 和 StandardKNN 在识别正确率上都会一致地提高，而识别时间则是先增加后减少，成对称态；反观 StandardSVM，时间逐渐减少，正确率几乎保持不变。

针对以上情况，我们做出了一些分析。首先从横向评测结果来看，StandardSVM 在时间上表现最优，印证了之前提到的 SVM 算法具有时间复杂度低的特点。而之所以 StandardSVM 在识别准确率上表现不好，我们认为有一部分原因是这个算法的训练集和另两个算法不一样。前两个算法用的是完全的交叉测试法，即从图片库中抽取一定张数张作为训练用，再用剩下的作为测试用，这两组图片会有一定的相关性，而 StandardSVM 算法中的训练集和测试集的相关性没有那么大，导致识别准确度不高。因此总的来说，这里测量 StandardSVM 的识别准确率意义并不是很大。不过值得一提的是，用手写板画数字进行非静态测试，StandardSVM 效果最好，可达到 95%以上的正确率。

再看一下 MyKNN 和 StandardKNN 两个本质相同的算法的横向比较情况，从数据可以看出，我们的 KNN 算法从性能上几乎是完败于 OpenCV 的 KNN 算法，我们对此进行了简单的分析：时间复杂度方面，我们虽然用了 KD 树这种高效的空间数据结构，但实际上 KD 树也许更适合点的维数不太多（这里是 128*128 维）、每一维数据比较分散（这里每一维不是 0 就是 1）的情况，因此用在手写数字识别的分类器上效果并不是十分理想，时间复杂度并不会有显著改善。识别准确率方面，MyKNN 中对图像处理并没有利用 Parse.cpp 中的函数（因为那个是针对 OpenCV 的），而是自己写的图像处理函数，这个函数只是对图像进行了一些锐化处理，并没有像 Parse.cpp 中的函数那样会把核心图像提取，所以对一些样子稍微有些

奇怪的数字就无法识别了。

其实说实话，我们是首先编写完成了 MyKNN，一开始并没有想做出图形界面而是决定主攻优化 MyKNN 的效率，不过后来我们突然发现 OpenCV 自带有 KNN 库和 SVM 库，我们才临时决定再编写两份功能相同的 cpp 文件，并且做出图形界面，这个占去了我们最后的时间，我们也没有进一步去对我们自己编写的 KNN 算法进行优化，算是一个小遗憾，也可以算是给未来的进一步提高留有了空间。

对横向的比较分析之后，我们还可以看看纵向比较的结果。两个 KNN 算法都会在测试集比例变大时，测试时间先增加后减少，这是由 KNN 算法的特点决定的。当测试集较少时，训练集会比较大，对于每一个测试样例，检索出“最近邻”的时间要变长；当测试集增加时，识别每个样例的时间会变少，但是要测试的样例变多了。进一步可以推出 KNN 算法的检索复杂度和数据规模呈线性关系，即 O(n)，而实际上 KD 树的检索复杂度是 O(n^(1-1/k)+m)，其中 k 是维数，m 是邻居数，在这个例子中，就是 O(n)的，也解释了 KNN 算法在时间上的表现。至于识别准确率的表现，则是训练集越多，正确率越高，这一点不难理解。而至于 SVM 算法，之前提过，它用到的训练集是固定的，无法调整它的大小，因此针对这个算法只变动了测试集大小，所以说它的用时逐渐减少、识别率几乎不变的表现是比较显然的。实际上，SVM 算法的纵向比较意义不大，主要是用来横向比较的。

综上，我们对 KNN 算法和 SVM 算法有了更深一步的了解，也对我们如何在之后进一步提高我们的 KNN 算法的途径有了打算，更为我们今后学习机器学习算法打下了基础。

七、感想与体会

我们最初是先决定研究 KNN 算法，并根据该算法做出相应的应用。在网上查询资料发现 KNN 算法常用来做文本分类器，但是小组内讨论觉得针对文本分类器并没有什么好的应用。最终讨论决定用 KNN 算法做数字图像识别。

因为 KNN 算法本身理解起来非常简单，代码的实现工作也不难，我们在实现了 KNN 算法之后决定再学习一下 SVM 算法，因为 SVM 算法对于数字识别的精确度似乎更高一些。

在实现 KNN 算法的时候，我们对 opencv 库并不了解，因此自己实现了对图片的转换向量处理和去干扰点处理。在查找学习 SVM 算法的时候，我们发现了 OpenCV 自带的 KNN 算法和 SVM 算法，因此我们将这三种算法做了比较，有 KNN 算法和 SVM 之间的比较，也有两个 KNN 之间的比较。

学习新的算法是一个很愉快的过程，尤其是把理论转化为实际应用并得出结果的时候。

不足之处可能在于我们对于训练集的样例仍然不是很充足，另外我们也没有很充足的时间在针对三个算法进行了评测之后对某一些算法继续改进。希望今后还有机会对此进行进一步改进。

非常高兴能有这样一个小组合作做一个项目的机会，我们第一次感受到了做一个实际一点的应用的挑战与乐趣，感谢小班课给了我们这样一个机会，也感谢老师和助教对我们一直以来的的帮助！