模式识别
遥感影像非监督分类
实习报告

（遥感信息工程学院 2012 级）

班　　级：1201

姓　　名：陈卉君

学　　号：2012302590011

指导教师：马洪超

一、 算法步骤

1.1 K 均值算法

K 均值分类方法假定被用来表示样本空间的聚类中心的个数是预先知道的。它使聚类域中所有样本到聚类中心的距离平方和最小，这是在误差平方和准则的基础上得来的。

K 平均算法由下述步骤组成：

(1) 任意选择k个初始聚类中心$Z_1(1)$、$Z_2(1)$、…、$Z_g(1)$，一般选择给定样本集的前 k 个样本作为初始聚类中心。

(2) 第k次迭代，$\| X - Z_j(k) \| < \| X - Z_i(k) \|$，式中$i = 1,2,\dots,k$，$i \neq j$，则 $X \in f_j(k)$，$f_j(k)$ 为聚类中心是$Z_j(k)$的样本集。于是分配各样本X到k个聚类域。

(3) 由(2)的结果,计算新的聚类中心

$$Z_j(k+1) = \frac{1}{n_j}\sum_{X \in f_j(k)} X，\ j = 1,2,3,\dots,k$$

这样使$f_j(k)$中的所有点到新的聚类中心的距离平方和最小。

(4) 若$Z_j(k+1) < Z_j(k)$，$j = 1,2,\dots,k$，算法收敛，程序结束。否则转入第二步。

1.2 ISODATA 算法

ISODATA(Iterative Self-organizing Data Analysis Techniques Algorithm)算法亦称迭代自组数据分析算法，它与 K-均值算法有两点不同。第一，它不是每调整一个样本的类别就重新计算一次各类样本的均值，而是在每次把所有样本都调整完毕之后才重新计算一次各类样本的均值，前者称为逐个样本修正法,后者称为成批样本修正法：第二，ISODATA 算法不仅可以通过调整样本所属类别完成样本的聚类分析，而且可以自动地进行类别的“合并”和“分裂”，从而得到类数比较合理的聚类结果。

ISODATA 算法描述如下：

第一步:给出下列控制参数:

K：希望得到的类别数（近似值）；

θ_N：所希望的一个类中样本的最小数目；

θ_S：关于类的分散程度的参数（如标准差）；

θ_C：关于类间距离的参数（如最小距离）；

L：每次允许合并的类的对数；
I：允许迭代的次数。

第二步：适当地选取N_C个类的初始中心$\{Z_i，\ i=1,2,\dots,N_C\}$。

第三步：把所有样本X按如下的方法分到N_C个类别中的某一类中去：对于所有的$i\neq j，\ i=1,2,\dots,N_C$，如果$\| X-Z_j \|<\| X-Z_i \|$，则$X\in S_j$其中$S_j$是以$Z_j$为中心的类。

第四步：如果S_j类中的样本数$N_j<\theta_N$，则去掉S_j类，$N_C=N_C-1$，返回“第三步”。

第五步：按下式重新计算各类的中心：

$$Z_j=\frac{1}{N_j}\sum_{X\in S_j}X，\ j=1,2,\dots,N_C$$

第六步：计算S_j类内的平均距离：

$$\bar{D_j}=\frac{1}{N_j}\sum_{X\in S_j}\| X-Z_j \|，\ j=1,2,\dots,N_C$$

第七步：计算所有样本离开其相应的聚类中心的平均距离：

$$\bar{D}=\frac{1}{N_j}\sum_{j=1}^{N_C}N_j\,\bar{D_j}，\ j=1,2,\dots,N_C$$

式中，N为样本总数。

第八步：如果迭代次数大于I，则转向“第十二步”，检查类间最小距离，判断是否进行合并。

如果$N_C<\frac{K}{2}$，则转向“第九步”，检查每类中各分量的标准差（分裂）。

如果迭代次数为偶数，或$N_C<2K$，则转向“第十二步”，检查类间最小距离，判断是否进行合并。

否则则转向“第九步”。

第九步：计算每类中各分量的标准差δ_{ij}：

$$\delta_{ij}=\sqrt{\frac{1}{N_j}\sum_{X\in S_j}\left(X_{ik}-Z_{ij}\right)^2}$$

式中，$i=1,2,\dots,n$，n为样本X的维数；

$j=1,2,\dots,N_C$，N_C为类别数；

$k=1,2,\dots,N_j$，k为S_j类中的样本数；

X_{ik}为第k个样本的第i个分量；

Z_{ij}为第j个聚类中心Z_j的第i个分量；

第十步：对每一个聚类S_j，找出标准差的最大的分量δ_{jmax}：

$$\delta_{jmax} = \max(\delta_{1j,}\delta_{2j,} \dots, \delta_{nj})，\ j = 1,2, \dots, N_C$$

第十一步：如果条件 1 和条件 2 有一个成立，则把S_j分裂成两个聚类，两个新类的中心分别为$Z_j{}^+$和$Z_j{}^-$，原来的Z_j取消，使$N_C = N_C + 1$，然后转向“第三步”，重新分配样本。其中，

条件 1：$\delta_{jmax} > \theta_S$且$\overline{D_j} > \overline{D}$且$N_j > 2(\theta_N + 1)$

条件 2：$\delta_{jmax} > \theta_S$且$N_C \leq \frac{K}{2}$

$$Z_j{}^+ = Z_j + \gamma_j$$

$$Z_j{}^- = Z_j - \gamma_j$$

$\gamma_j = \delta_{jmax} \cdot k$，$k$是认为给定的常数，且$0 < k \leq 1$

第十二步：计算所有聚类中心之间的两两距离：

$$D_{ij} = \| X - Z_i \|，\ i = 1,2, \dots, N_C - 1，\ j = i + 1, \dots, N_C$$

第十三步：比较D_{ij}和θ_C，把小于θ_C的D_{ij}按从小到大的顺序排列：

$$D_{i_1 j_1} < D_{i_2 j_2} < \cdots < D_{i_L j_L}$$

其中，L 为每次允许合并的类的对数

第十四步:按照$l = 1,2, \dots, L$的顺序,把$D_{i_l j_l}$所对应的两个聚类中心Z_{i_l}和Z_{j_l}合并成一个新的聚类中心$Z_l{}^*$，并使$N_C = N_C - 1$：

$$Z_l{}^* = \frac{1}{N_{i_l} \cdot N_{j_l}} (N_{i_l} Z_{i_l} + N_{j_l} Z_{j_l})$$

在对$D_{i_l j_l}$所对应的两个聚类中心Z_{i_l}和Z_{j_l}进行合并时，如果其中至少有一个聚类中心已经被合并过，则越过该项，继续进行后面的合并处理。

第十五步：若迭代次数大于I，或者迭代中的参数的变化在差限以内，则迭代结束，否则转向“第三步”继续进行迭代处理。

ISODATA 法的实质是以初始类别为“种子”实行自动迭代聚类的过程。迭代结束标志着分类所依据的基准类别已经确定，它们的分布参数也在不断的“聚类训练”中逐渐确定，并最终用于构建所需要的判决函数。从这个意义上讲，其基准类别参数的确定过程,也正是对判决函数的不断调整和“训练”过程。只不过这种“训练”的依据，不是来自外来的先验知识，而是来自类别光谱特征本身的统计性质。

二、 代码实现

2.1 K 均值聚类算法

```
void CBayesView::OnK1()
{
    //线性存储
    CBayesDoc *pDoc=GetDocument();
    DWORD lHeight = pDoc->lHeight;
    DWORD lWidth = pDoc->lWidth;
    DWORD bandnum = pDoc->nbands;

    BYTE** data = (BYTE**)malloc(sizeof(BYTE*)*bandnum);
    double *Result = (double*)malloc(sizeof(double)
        *bandnum*lHeight*lWidth);

    for(DWORD i=0;i<bandnum;i++)
        data[i] = (BYTE*)malloc(sizeof(BYTE)*lHeight*lWidth);

    for (DWORD j=0;j<lHeight;j++)
    {
        for (DWORD i=0;i<lWidth;i++)
        {
            //读取每个像素点的每个波段DN值
            for(DWORD k=0;k<bandnum;k++)
                data[k][i+j*lWidth] = *(pDoc->pData+i+
                j*lWidth+k*lWidth*lHeight);

        }
    }

    //计算均值、方差、特征值、特征向量

    double *avg = (double*)malloc(sizeof(double)*bandnum);
    //计算各波段均值
    for(DWORD k=0;k<bandnum;k++)
    {
        avg[k] = 0;
        for (i=0;i<lHeight*lWidth;i++)
        {
            avg[k] += (double)data[k][i];
        }
        avg[k] /= (double)(lHeight*lWidth);
    }
    //计算波段之间的协方差矩阵
    double **cov = create_array(bandnum,bandnum);

    for(i=0;i<bandnum;i++)
    {
        for(DWORD j=0;j<bandnum;j++)
        {
            cov[i][j] = 0.0;
```

```
            double a = 0,b = 0;
            for (DWORD k=0;k<lHeight*lWidth;k++)
            {
                a += data[i][k]*data[j][k];
            }
            a /= (lWidth*lHeight);
            b = avg[i]*avg[j];
            cov[i][j] = a - b;
        }
    }
    //计算协方差矩阵的特征值

     int maxI=60;
     double **q,*b,*c;
     b = (double*)malloc(sizeof(double)*bandnum);
     c = (double*)malloc(sizeof(double)*bandnum);
     q = create_array(bandnum,bandnum);
    const double eps=0.000001;
    strq(cov,bandnum,q,b,c);
    sstq(bandnum,b,c,q,eps,maxI);

    //特征值降序排序
    for (j=0;j<bandnum;j++)
    {

         for (int i=0;i<bandnum-j;i++)
         {
             if (b[i]<b[i+1])
             {
                 double t;
                 t=b[i];b[i]=b[i+1];b[i+1]=t;
                 liehuhuan(q,bandnum,bandnum,i,i+1);
             }
         }
     }

     //进行变换
     double *t=(double*)malloc(sizeof(double)*bandnum);
     for (j=0;j<lHeight;j++)
     {
         for (DWORD i=0;i<lWidth;i++)
         {
             for(DWORD k=0;k<bandnum;k++)
             {
                 t[k]=*(pDoc->pData+i+j*lWidth+k*lWidth*lHeight)-avg[k];
                 *(Result+i+j*lWidth+k*lWidth*lHeight) = 0.0;
             }

             for(k=0;k<bandnum;k++)
```

```
            {
                for(DWORD num=0;num<bandnum;num++)
                {
                    *(Result+i+j*lWidth+k*lWidth*lHeight) +=
                        q[num][k]*t[num];
                }
            }
        }
    }
    delete[] t;

    //显示图像
    for (j=0;j<lHeight;j++)
    {
        for (DWORD i=0;i<lWidth;i++)
        {
            for(DWORD k=0;k<bandnum;k++)
            {
                *(pDoc->pData+i+j*lWidth+k*lWidth*lHeight) =
                    *(Result+i+j*lWidth+k*lWidth*lHeight);
            }
        }
    }

    //输出图像
```

2.2 ISODATA 算法 C 语言实现

```
void CBayesView::OnIsodata()
{
    CBayesDoc *pDoc=GetDocument();

    DWORD lHeight = pDoc->lHeight;
    DWORD lWidth = pDoc->lWidth;
    DWORD bandnum = pDoc->nbands;

    DWORD kindnum = 0;
    DWORD maxiteration = 0;
    DWORD SigmaN,L,K;
    double SigmaC,SigmaS;
    CIsodata dlg;
    if (dlg.DoModal()==IDOK)
    {
        K = dlg.m_K; //获取分类数
        maxiteration = dlg.m_I; //最大迭代次数
        SigmaN = dlg.m_SigmaN;
        SigmaC = dlg.m_SigmaC;
        SigmaS = dlg.m_SigmaS;
        L = dlg.m_L;
    }
    delete dlg;
    double** Result = create_array(lHeight,lWidth); //分类结果
```

```
    double min = 10000000; int minID;
    DWORD count = 0;

    int MAXKINDNUM = 100;
    double **Z = create_array(MAXKINDNUM,bandnum); //K个聚类中心
    double *d = (double*)malloc(sizeof(double)*MAXKINDNUM);
    int *num = (int*)malloc(sizeof(int)*MAXKINDNUM);

    int oldkindnum;
    double *maxsigma;
    int *maxsigmaID ;
    double **sigma;
    double** dij;
    double dij_ongoing;
    int i_ongoing=0;
    int j_ongoing=0;

step1:
    //选取C个初始聚类中心 C不一定等于K 这里C=2*K
    kindnum = 2*K;
```

```
    DWORD x0,y0;
    srand((unsigned)time(NULL));
    for(DWORD i=0;i<kindnum;i++)
    {
        x0 = rand()/((DWORD)(RAND_MAX)/lHeight);
        y0 = rand()/((DWORD)(RAND_MAX)/lWidth);
        for(DWORD k=0;k<bandnum;k++)
        {
            Z[i][k] =  *(pDoc->pData+y0+x0*lWidth+k*lWidth*lHeight);
        }
    }

    BYTE *x = (BYTE*)malloc(sizeof(BYTE)*bandnum);

step2:
        //将样本集的每个样本按照最小距离原则分配给kindnum个聚类中心

    for(i=0;i<kindnum;i++)
            num[i]=0;

        for (DWORD j=0;j<lHeight;j++)
        {
            for (DWORD i=0;i<lWidth;i++)
            {
                //读取每个像素点的每个波段DN值
                for(DWORD k=0;k<bandnum;k++)
```

```
                x[k] = *(pDoc->pData+i+j*lWidth+k*lWidth*lHeight);

            //求出与各聚类中心的距离
            for(k=0;k<kindnum;k++)
            {
                d[k] = 0.0;
                for(DWORD band=0;band<bandnum;band++)
                {
                    double t1 = x[band]*Z[k][band];
                    double t2 = Z[k][band]*Z[k][band]/2;
                    d[k] += (t2-t1);
                }
            }
            //找出最小的d
            for(k=0;k<kindnum;k++)
            {
                if(d[k]<min)
                {
                    min=d[k];
                    minID=k;
                }
            }
            Result[j][i] = minID;
            num[minID]++;
                min = 1000000;

            }
        }
        count++;

step3:
        //对于某个聚类域，若其样本数小于SigmaN
        //则取消这个子集，类别数量-1，重新分配样本
        int Oldkindnum = kindnum;
        for (i=0;i<Oldkindnum;i++)
        {
            if(num[i]<SigmaN)
            {
                kindnum--;
                for (int j=i;j<Oldkindnum-1;j++)
                {
                    hanghuhuan(Z,MAXKINDNUM,bandnum,j,j+1);
                }
                goto step2;
            }
        }

step4:
```

```
        //选取新的聚类中心
            for(DWORD k=0;k<kindnum;k++)
            {
                for(DWORD band=0;band<bandnum;band++)
                {
                    if(num[k]!=0)
                        Z[k][band] = 0.0;
                }
            }


            for (j=0;j<lHeight;j++)
            {
                for (DWORD i=0;i<lWidth;i++)
                {
                    for(DWORD band=0;band<bandnum;band++)
                    {
                        BYTE t = *(pDoc->pData+i+
                            j*lWidth+band*lWidth*lHeight);
                        Z[(int)Result[j][i]][band] +=
                            (double)t/(double)num[(int)Result[j][i]];
                    }
                }
            }
```

step5:

```
            //计算所有样本到其聚类中心的距离的平均值
            double *avr_dj = (double*)malloc(sizeof(double)*kindnum);
            for(k=0;k<kindnum;k++)
            {
                avr_dj[k] = 0.0;
            }

            for (j=0;j<lHeight;j++)
            {
                for (DWORD i=0;i<lWidth;i++)
                {
                    int k = (int)Result[j][i];
                    for(DWORD band=0;band<bandnum;band++)
                    {

                        BYTE t = *(pDoc->pData+i+j*lWidth+
                            band*lWidth*lHeight);
                        double t1 = t*Z[k][band];
                        double t2 = Z[k][band]*Z[k][band]/2;
                        avr_dj[k] += (t2-t1);

                    }
                    avr_dj[k] /= (double)num[k];
                }
            }
            delete[] avr_dj;
```

```
step6:
            //计算所有样本到其相应聚类中心的平均值
            double avr_d = 0.0;
            long int Nnum = 0;
            for(k=0;k<kindnum;k++)
            {
                avr_d += num[k]*avr_dj[k];
                Nnum += num[k];
            }
            avr_d/=Nnum;


step7:      //判断去向
            if (count>=maxiteration)
                goto step14;
            if(kindnum<=K/2)//跳过合并,进行分裂
                goto step8;
            else if(kindnum>=2*K)//跳过分裂，进行合并
                goto step11;
            else //奇数分裂，偶数合并
            {
                if((kindnum/2) == (kindnum/2)*2)
                    goto step11;
                else
                    goto step8;
            }
```

```
step8:
            //计算各类中样本与聚类中心之间距离的标准差sigma
            sigma = create_array(kindnum,bandnum);
            memset(sigma,0,sizeof(double)*bandnum*kindnum);

            for (j=0;j<lHeight;j++)
            {
                for (DWORD i=0;i<lWidth;i++)
                {
                    int k = (int)Result[j][i];
                    for(DWORD band=0;band<bandnum;band++)
                    {
                        BYTE t = *(pDoc->pData+i+j*lWidth+
                            band*lWidth*lHeight);
                        sigma[k][band] += (t-Z[k][band])*
                            (t-Z[k][band])/num[k];
                    }
                }
            }
            for (i=0;i<kindnum;i++)
            {
                for (j=0;j<bandnum;j++)
                {
                    sigma[i][j] = sqrt(sigma[i][j]);
                }
            }
```

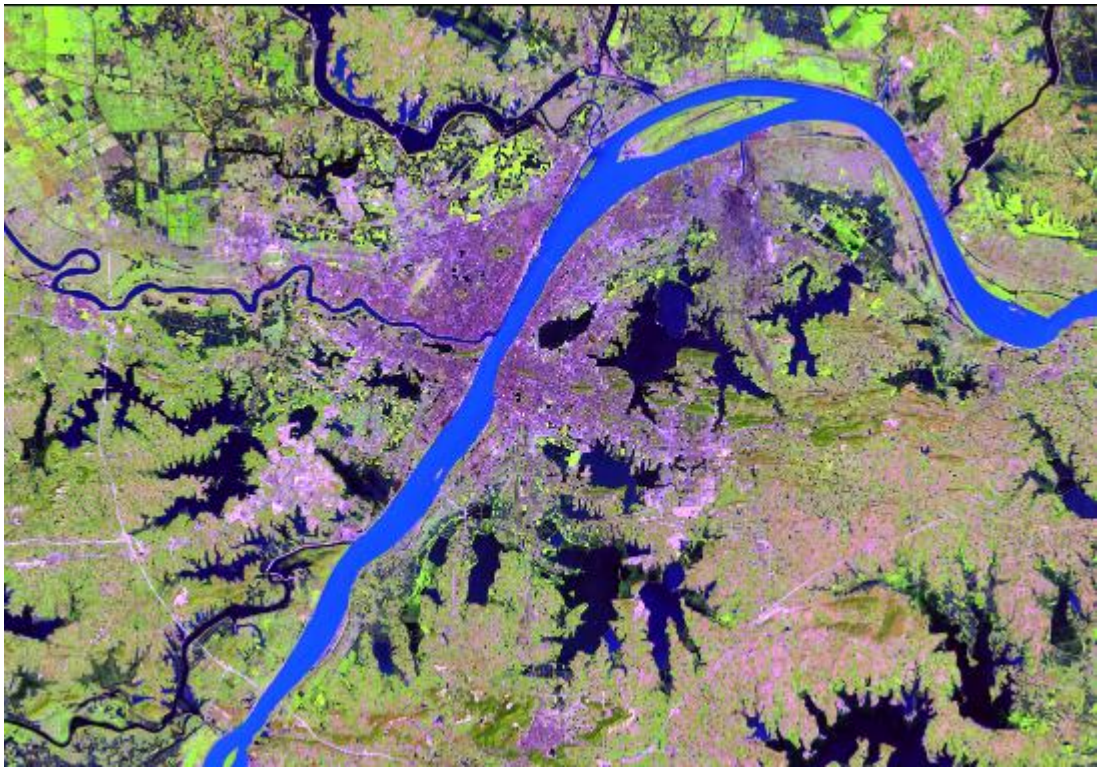
```
step9:
            //求每个标准差中的最大分量
            maxsigma = (double*)malloc(sizeof(double)*kindnum);
            maxsigmaID = (int*)malloc(sizeof(int)*kindnum);

            for (i=0;i<kindnum;i++)
            {
                maxsigma[i]=sigma[i][0];
                for (j=0;j<bandnum;j++)
                {
                    if (maxsigma[i]<sigma[i][j])
                    {
                        maxsigma[i]=sigma[i][j];
                        maxsigmaID[i]=j;
                    }
                }
            }

            delete[] maxsigma;
            delete[] maxsigmaID;

step10:   //进行分裂判断
          //若有最大标准差分类大于SigmaS，且avr_dj>avr_d
          //则分裂,类别数+1

                oldkindnum = kindnum;

                for (k=0;k<oldkindnum;k++)
                {
                    if((num[k]>2*(SigmaN-1) && avr_dj[k]>avr_d
                        || kindnum<=K/2) && maxsigma[k]>SigmaS)
                    {
                        //分裂
                        kindnum++;
                        const double M = 0.5;
                        double r = M*sigma[k][(int)maxsigmaID[k]];
                        Z[kindnum][(int)maxsigmaID[k]] =
                            Z[k][(int)maxsigmaID[k]] + r;
                        Z[k][(int)maxsigmaID[k]] -= r;

                        free_array(sigma,kindnum);

                        goto step2;
                    }
                }

step11:  //计算全部聚类中心之间的距离
                dij = create_array(kindnum,kindnum);
                for (i=0;i<kindnum;i++)
```

```
                {
                    for (j=0;j<kindnum;j++)
                    {
                        dij[i][j] = 0.0;
                        for (k=0;k<bandnum;k++)
                        {
                            dij[i][j] += (Z[i][k]-Z[j][k])*
                                (Z[i][k]-Z[j][k]);
                        }
                    }
                }


step12:  //合并
        //取出dij中小于SigmaC的，合并
                dij_ongoing=dij[0][0];


                for (i=0;i<kindnum;i++)
                {
                    for (j=0;j<kindnum;j++)
                    {
                        if(dij[i][j]<dij_ongoing)
                        {
                            dij_ongoing=dij[i][j];
                            i_ongoing=i;
                            j_ongoing=j;
                        }
                    }
                }
                free_array(dij,kindnum);


step13:  //合并不符合要求的类别
                if (dij_ongoing<SigmaC)
                {
                    kindnum--;
                    for (k=0;k<bandnum;k++)
                    {
                    Z[i_ongoing][k] = (num[i_ongoing]*Z[i_ongoing][k]
                        +num[j_ongoing]+Z[j_ongoing][k])/
                        (num[j_ongoing]+num[i_ongoing]);
                    }
                    for (int i=j_ongoing;i<Oldkindnum-1;i++)
                    {
                        hanghuhuan(Z,MAXKINDNUM,bandnum,i,i+1);
                    }
                }
```

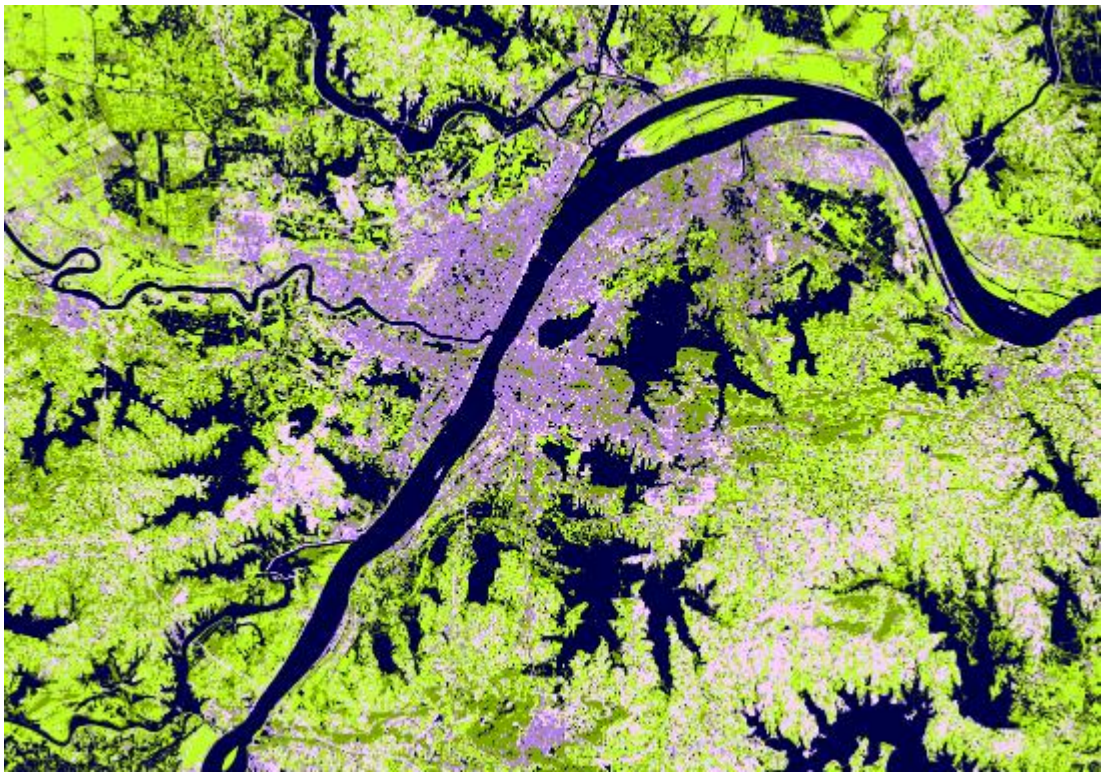
```
                }

step14:  //判断是否继续迭代
                if(count>=maxiteration)
                {
                    count=0;
                }
                else
                {
                    goto step2;
                }

    //输出图像
```

三、 结果分析

用于分类的影像是在地理空间数据云上下载的武汉市的 TM 影像，使用 ERDAS 软件将 TM 的 1 到 5 波段以及第 7 波段进行合成，合成后影像格式为 img，下图为 543 波段作为 RGB 显示的影像。



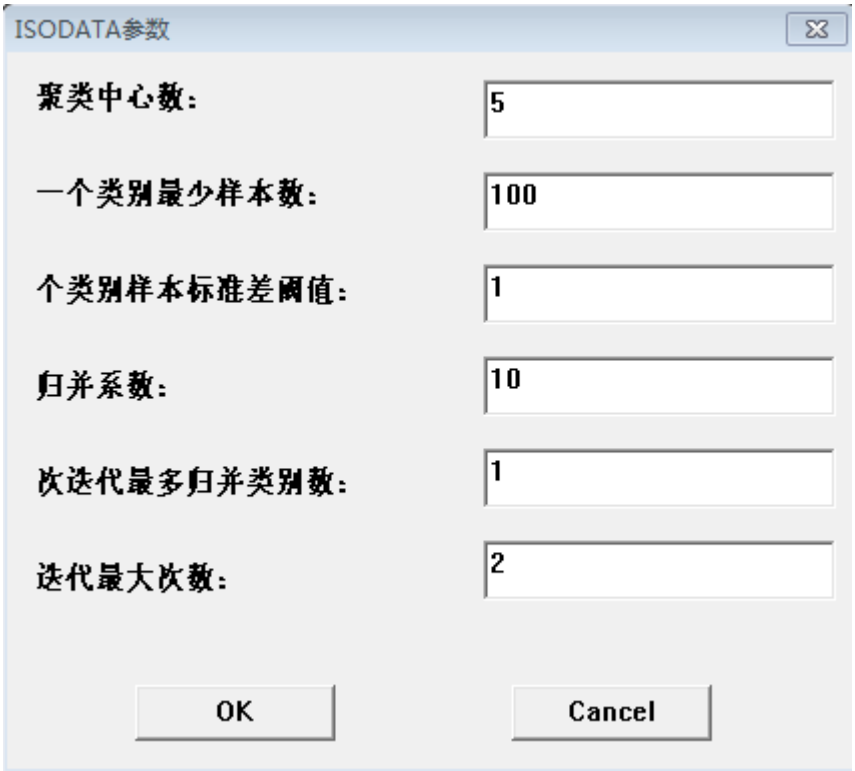
1. K 均值算法聚类结果

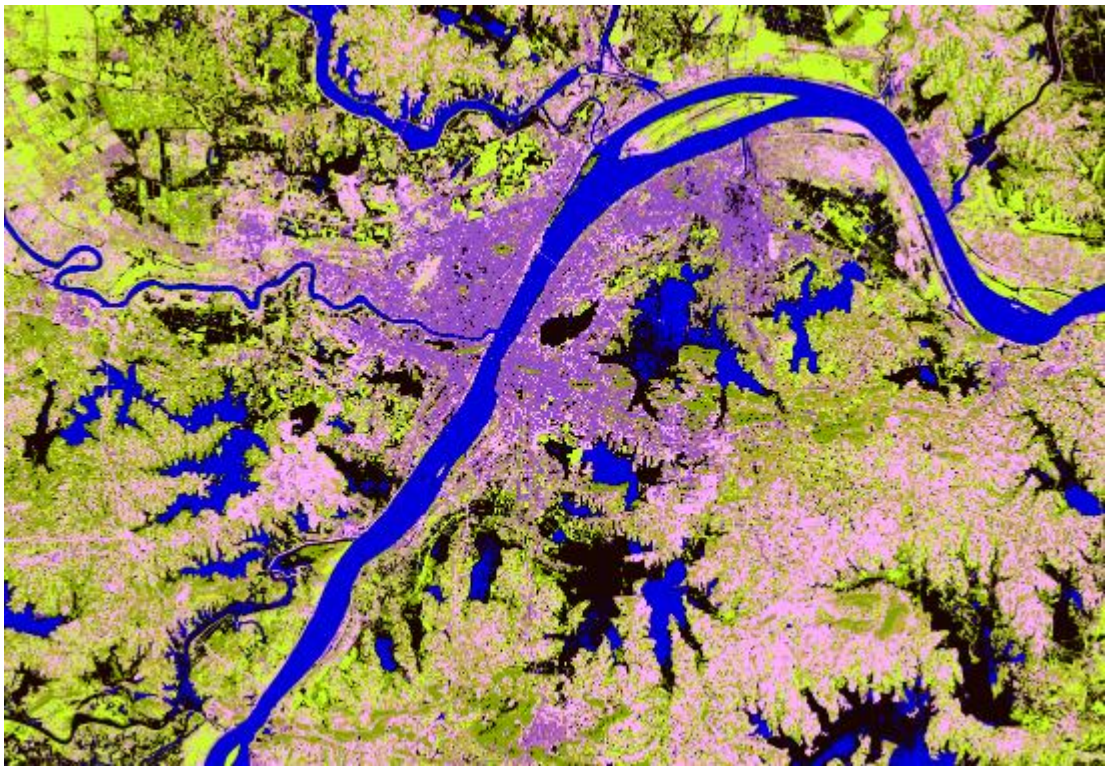


可以看出，地物大概分为了城镇用地、草地与农田、水系、裸地和林地，总体来说还比较准确，只是林地和农田混淆比较严重。

不过 K 均值算法的分类结果和初始点有很大的关系，在本程序中初始点是随机点，按照同样的参数再分一次，效果如下图，可以看出城镇用地不少被分成了林地，这是极大的错误！K 均值算法虽然计算量较小，但是分类精度具有较大的随机性，不是很精确。

2. ISODATA 聚类结果





由于代表类别的颜色我取的是该类别像素的平均值（543 波段代表 RGB），而从色调上来看，ISODATA 分出来的更接近原图像，可以判定大体上 ISODATA 分类结果更好，从细节上来看也是如此。

四、心得体会

1. 对算法的理解是写好程序的基础和关键，对算法理解透了才能更有效率地设计程序
2. 需要了解一下遥感图像处理中常用的开源库，如 GDAL
3. 对于误操作要有相应处理方法和警告
4. 把程序分成几个功能块，写一步测试一步调试效率较高