Weka[28] EM 源代码分析

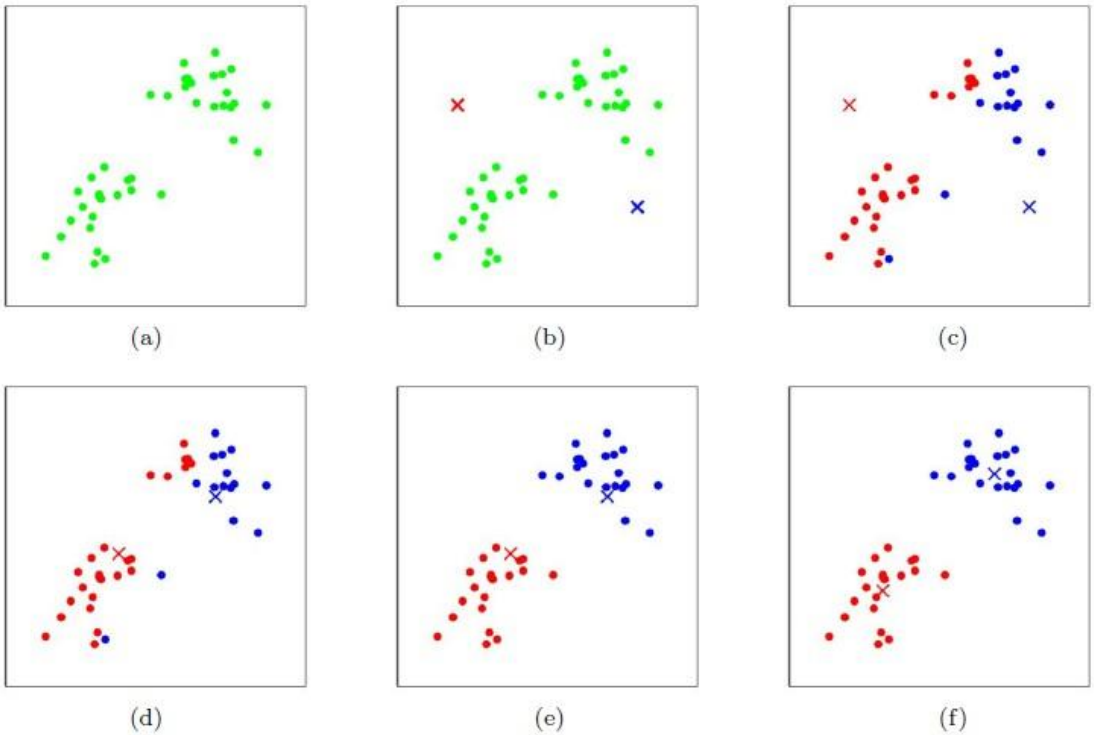
作者：Koala++/屈伟

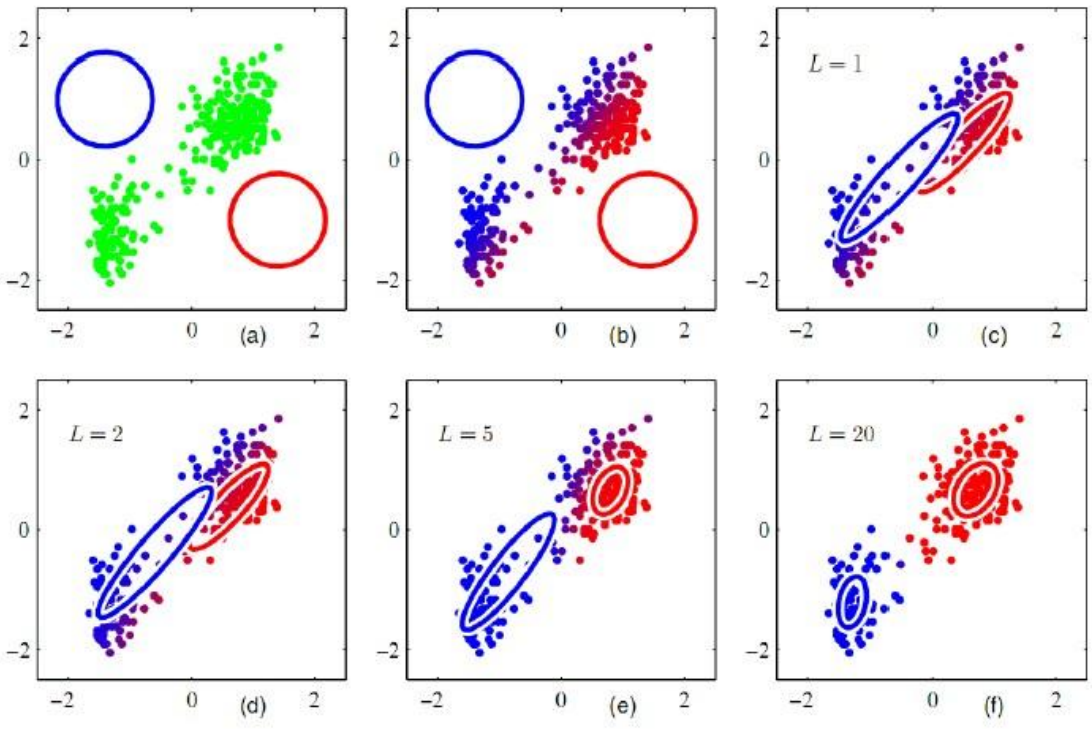
EM 算法在 clusterers 下面，提一下是因为我没有想到它竟然在这里，而且它的名字也太大了点，因为这里它只是与 SimpleKMeans 结合的算法。

引自 Andrew Ng 的 Lecture notes mixtures of Gaussians and the EM algorithm：The EM-algorithm is also reminiscent of the K-means clustering algorithm, except that instead of "hard" cluster assignment c(i), we instead have the "soft" assignment w_j^(i). Similar to K-means, it is also susceptible to local optima, so reinitializing at several different initial parameters may be a good idea。

Soft 指的是我们猜测是概率，取值在[0,1]区间，相反，"hard"猜测是指单个最好的猜测，可以取值在{0,1}或是{1,…,k}。英文原文：The term "soft" refers to our guesses being probabilities and taking values in [0,1]; in contrast, a "hard" guess is one that represents a single best guess(such as taking values in {0,1} or {1,…,k})

下面的图来自 Ng Andrew 和 Bishop Chistopher，第一组图 K-Means 的猜测是两个点，而第二组图 EM 是对概率的猜测。





另一点是刚才文中提到的，多个初始化点，在代码中也体现了。

Ng 在对 EM 算法收敛证明之后，解释如下：Hence, EM causes the likelihood to converge monotonically. In our description of the EM algorithm, we said we'd run it until convergence. Given the result that we just showed, one reasonable convergence test would be to check if the increase in l(theta) between successive iterations is smaller than some tolerance parameter, and to declare convergence if EM is improving l(theta) too slowly.

从 buildCluster 开始：

```
if (data.checkForStringAttributes()) {
    throw new Exception("Can't handle string attributes!");
}

m_replaceMissing = new ReplaceMissingValues();
Instances instances = new Instances(data);
instances.setClassIndex(-1);
m_replaceMissing.setInputFormat(instances);
data = weka.filters.Filter.useFilter(instances, m_replaceMissing);
instances = null;

m_theInstances = data;

// calculate min and max values for attributes
m_minValues = new double[m_theInstances.numAttributes()];
m_maxValues = new double[m_theInstances.numAttributes()];
for (int i = 0; i < m_theInstances.numAttributes(); i++) {
    m_minValues[i] = m_maxValues[i] = Double.NaN;
}
for (int i = 0; i < m_theInstances.numInstances(); i++) {
    updateMinMax(m_theInstances.instance(i));
}
```

ReplaceMissingValues 是将缺失值用平均值或中位数代替。m_minValues 和 m_maxValues 是每个属性的最小值与最大值数组。

```
private void updateMinMax(Instance instance) {
```

```
    for (int j = 0; j < m_theInstances.numAttributes(); j++) {
        if (!instance.isMissing(j)) {
            if (Double.isNaN(m_minValues[j])) {
                m_minValues[j] = instance.value(j);
                m_maxValues[j] = instance.value(j);
            } else {
                if (instance.value(j) < m_minValues[j]) {
                    m_minValues[j] = instance.value(j);
                } else {
                    if (instance.value(j) > m_maxValues[j]) {
                        m_maxValues[j] = instance.value(j);
                    }
                }
            }
        }
    }
}
```

Double.isNan 这里是判断是不是还没有一个真正的属性值来代替过它。其它的代码就是找第 j 个属性的最大值和最小值。

```
doEM();

// save memory
m_theInstances = new Instances(m_theInstances, 0);
```

doEM 之后就是释放空间了，那么所有的工作都是在 doEM 中完成的：

```
private void doEM() throws Exception {

    m_rr = new Random(m_rseed);

    // throw away numbers to avoid problem of similar initial numbers
    // from a similar seed
    for (int i = 0; i < 10; i++)
        m_rr.nextDouble();

    m_num_instances = m_theInstances.numInstances();
    m_num_attribs = m_theInstances.numAttributes();

    // setDefaultStdDevs(theInstances);
    // cross validate to determine number of clusters?
    if (m_initialNumClusters == -1) {
        if (m_theInstances.numInstances() > 9) {
            CVClusters();
            m_rr = new Random(m_rseed);
            for (int i = 0; i < 10; i++)
                m_rr.nextDouble();
        } else {
            m_num_clusters = 1;
        }
    }

    // fit full training set
    EM_Init(m_theInstances);
    m_loglikely = iterate(m_theInstances, m_verbose);
}
```

丢弃从同一个种子得到的随机数，这个与下面的代码有关？如果 m_initialNumClusters ==-1 表明没有指定要聚多少个类，那么要用 cross validate 来决定聚多少个类。如果样本数大于 9，用 CVClusters 函数来决定。如果小于 9 个样本，就认为就一个类。EM_Init 初始化，

然后迭代，先不去管 CVClusters，认为已经指定了 m_initialNumClusters，那么先看 EM_Init：

```
// run k means 10 times and choose best solution
SimpleKMeans bestK = null;
double bestSqE = Double.MAX_VALUE;
for (i = 0; i < 10; i++) {
    SimpleKMeans sk = new SimpleKMeans();
    sk.setSeed(m_rr.nextInt());
    sk.setNumClusters(m_num_clusters);
    sk.buildClusterer(inst);
    if (sk.getSquaredError() < bestSqE) {
        bestSqE = sk.getSquaredError();
        bestK = sk;
    }
}
```

这里是用不同的随机种子初始化，最后求得一个最好的 SimpleKMeans 对象。

```
// initialize with best k-means solution
m_num_clusters = bestK.numberOfClusters();
m_weights = new double[inst.numInstances()][m_num_clusters];
m_model = new DiscreteEstimator[m_num_clusters][m_num_attribs];
m_modelNormal = new double[m_num_clusters][m_num_attribs][3];
m_priors = new double[m_num_clusters];
Instances centers = bestK.getClusterCentroids();
Instances stdD = bestK.getClusterStandardDevs();
int[][][] nominalCounts = bestK.getClusterNominalCounts();
int[] clusterSizes = bestK.getClusterSizes();
```

centers 是聚类后的所有中心点，stdD 是标准差，而 nominalCounts 第一维大小为所聚类的个数，第二维属性数，第三级该维的取值数。

```
for (i = 0; i < m_num_clusters; i++) {
    Instance center = centers.instance(i);
    for (j = 0; j < m_num_attribs; j++) {
        if (inst.attribute(j).isNominal()) {
            m_model[i][j] = new DiscreteEstimator(m_theInstances
                    .attribute(j).numValues(), true);
            for (k = 0; k < inst.attribute(j).numValues(); k++) {
                m_model[i][j].addValue(k, nominalCounts[i][j][k]);
            }
        } else {
            double minStdD = (m_minStdDevPerAtt != null) ?
                m_minStdDevPerAtt[j] : m_minStdDev;
            double mean = (center.isMissing(j)) ? inst.meanOrMode(j)
                    : center.value(j);
            m_modelNormal[i][j][0] = mean;
            double stdv = (stdD.instance(i).isMissing(j)) ?
                ((m_maxValues[j] - m_minValues[j]) / (2 * m_num_clusters))
                    : stdD.instance(i).value(j);
            if (stdv < minStdD) {
                stdv = inst.attributeStats(j).numericStats.stdDev;
                if (Double.isInfinite(stdv)) {
                    stdv = minStdD;
                }
                if (stdv < minStdD) {
                    stdv = minStdD;
                }
            }
            if (stdv <= 0) {
                stdv = m_minStdDev;
```

```
            }

            m_modelNormal[i][j][1] = stdv;
            m_modelNormal[i][j][2] = 1.0;
        }
    }
}
```

这里 DiscreteEstimator 是针对离散数据进行统计的一个类，构造函数如下：

```
public DiscreteEstimator(int numSymbols, boolean laplace) {

    m_Counts = new double[numSymbols];
    m_SumOfCounts = 0;
    if (laplace) {
        for (int i = 0; i < numSymbols; i++) {
            m_Counts[i] = 1;
        }
    }
    m_SumOfCounts = (double) numSymbols;
}
```

这里使用了 laplace 平滑，m_Counts 初始为 1，也就是平常所见过的公式加上 1，并且 SumOfCounts 也初始化为取值的个数，也就是公式中分母最后加的那个数。

```
public void addValue(double data, double weight) {

    m_Counts[(int) data] += weight;
    m_SumOfCounts += weight;
}
```

addValue 函数很简单就是在第几个取值上，加上相应的权重。写这么麻烦是因为不能得到连续值的估计。minStdD 控制精度，平均值是中心点的取值，而 stdv 就是在 SimpleKMeans 中计算出的值。M_modelNormal[i][j][0] 是均值，M_modelNormal[i][j][1] 是方差，M_modelNormal[i][j][2]记录的是概率。

```
for (j = 0; j < m_num_clusters; j++) {
    //m_priors[j] += 1.0;
    m_priors[j] = clusterSizes[j];
}
Utils.normalize(m_priors);
```

通过每个所聚类的大小，算出先验概率。

```
private double iterate(Instances inst, boolean report) throws Exception
{
    int i;
    double llkold = 0.0;
    double llk = 0.0;

    boolean ok = false;
    int seed = m_rseed;
    int restartCount = 0;
    while (!ok) {
        try {
            for (i = 0; i < m_max_iterations; i++) {
                llkold = llk;
                llk = E(inst, true);

                if (i > 0) {
                    if ((llk - llkold) < 1e-6) {
                        break;
                    }
```

```
                }
                M(inst);
            }
            ok = true;
        } catch (Exception ex) {
        }
    }

    return llk;
}
```

可以看到有两种迭代中止的方法，第一种是达到了 m_max_iterations，第二种是 llk-llkold 小于阈值，llk 是 log likelihood 的缩写，EM 的目标就是最大化它，如果已经接近最优值了，所以就停止了。下面就是 E：

```
private double E(Instances inst, boolean change_weights) throws Exception
{

    double loglk = 0.0, sOW = 0.0;

    for (int l = 0; l < inst.numInstances(); l++) {

        Instance in = inst.instance(l);

        loglk += in.weight() * logDensityForInstance(in);
        sOW += in.weight();

        if (change_weights) {
            m_weights[l] = distributionForInstance(in);
        }
    }

    // reestimate priors
    if (change_weights) {
        estimate_priors(inst);
    }
    return loglk / sOW;
}
```

这里 logDensityForInstance 的代码：

```
public double logDensityForInstance(Instance instance) throws Exception
{
    double[] a = logJointDensitiesForInstance(instance);
    double max = a[Utils.maxIndex(a)];
    double sum = 0.0;

    for (int i = 0; i < a.length; i++) {
        sum += Math.exp(a[i] - max);
    }

    return max + Math.log(sum);
}
```

logJointDensitiesForInstance：

```
public double[] logJointDensitiesForInstance(Instance inst)
        throws Exception {

    double[] weights = logDensityPerClusterForInstance(inst);
    double[] priors = clusterPriors();
```

```
    for (int i = 0; i < weights.length; i++) {
        if (priors[i] > 0) {
            weights[i] += Math.log(priors[i]);
        } else {
            throw new IllegalArgumentException("Cluster empty!");
        }
    }
    return weights;
}
```

logDensityPerClusterForInstance：

```
public double[] logDensityPerClusterForInstance(Instance inst)
        throws Exception {

    int i, j;
    double logprob;
    double[] wghts = new double[m num clusters];

    m replaceMissing.input(inst);
    inst = m replaceMissing.output();

    for (i = 0; i < m num clusters; i++) {
        logprob = 0.0;

        for (j = 0; j < m num attribs; j++) {
            if (!inst.isMissing(j)) {
                if (inst.attribute(j).isNominal()) {
                    logprob += Math.log(m model[i][j].
                        getProbability(inst.value(j)));
                } else { // numeric attribute
                    logprob += logNormalDens(inst.value(j),
                            m modelNormal[i][j][0],
                            m modelNormal[i][j][1]);
                }
            }
        }

        wghts[i] = logprob;
    }
    return wghts;
}
```

对于离散型属性，这里计算它的概率的对数，它的概率计算很简单：

```
public double getProbability(double data) {

    if (m SumOfCounts == 0) {
        return 0;
    }
    return (double) m Counts[(int) data] / m SumOfCounts;
}
```

就是 Laplace 平滑后的概率，而对于连续属性：

```
private static double m normConst = Math.log(Math.sqrt(2 * Math.PI));

private double logNormalDens(double x, double mean, double stdDev) {

    double diff = x - mean;

    return -(diff * diff / (2 * stdDev * stdDev)) - m normConst
           - Math.log(stdDev);
```

```
}
```

Diff 就是高斯分布(正态分布)中的 x-mean，而下面的那一长串就是对高斯分布的公式对数化得到的。

回到 logJointDensitiesForInstance 中，clusterPriors 的代码如下：

```
public double[] clusterPriors() {

    double[] n = new double[m priors.length];

    System.arraycopy(m priors, 0, n, 0, n.length);
    return n;
}
```

只是复制一下，而在 clusterPriors 后，weights[i]+=Math.log(priors[i])还是取对数后，可以展开来，也就是 P(xi|zi)P(zi)的 P(zi)，log(P(xi|zi)P(zi))=log(P(xi|zi))+log(P(zi))。

再回到 logDensityForInstance 中，这里的 Math.exp(a[i]-max)这个看起来奇怪，公式里没有 a[i]-max，这里可以把 max 这个量想成要用别的 a[i]组合得到的，因为 sum(a[i])=1，可以参考一下 Andrew Ng 的 lecture notes 中的 supervise learing 第 28 页。其实也就是分母。

E 函数中，有 distributionForInstance 这个函数如下：

```
public double[] distributionForInstance(Instance instance) throws
Exception {

    return Utils.logs2probs(logJointDensitiesForInstance(instance));
}
```

函数 logs2probs 注释中写，将概率取自然对数后的数组转换回概率，概率的和为 1。Converts an array containing the natural logarithms of probabilities stored in a vector back into probabilities. The probabilities are assumed to sum to one.

```
private void estimate priors(Instances inst) throws Exception {

    for (int i = 0; i < m num clusters; i++) {
        m priors[i] = 0.0;
    }

    for (int i = 0; i < inst.numInstances(); i++) {
        for (int j = 0; j < m num clusters; j++) {
            m priors[j] += inst.instance(i).weight() * m weights[i][j];
        }
    }

    Utils.normalize(m priors);
}
```

计算每个类的先验概率，就是将每个样本的权重×属于某个类的概率。

下面是 M 函数的内容：

new_estimators 的代码如下：

```
private void new estimators() {
    for (int i = 0; i < m num clusters; i++) {
        for (int j = 0; j < m num attribs; j++) {
            if (m theInstances.attribute(j).isNominal()) {
                m model[i][j] = new DiscreteEstimator(m theInstances
                        .attribute(j).numValues(), true);
            } else {
                m modelNormal[i][j][0] = m modelNormal[i][j][1] =
                    m modelNormal[i][j][2] = 0.0;
            }
        }
```

```
    }
}
```

重新初始化 m_model 和 m_modelNormal。

```
for (i = 0; i < m_num_clusters; i++) {
    for (j = 0; j < m_num_attribs; j++) {
        for (l = 0; l < inst.numInstances(); l++) {
            Instance in = inst.instance(l);
            if (!in.isMissing(j)) {
                if (inst.attribute(j).isNominal()) {
                    m_model[i][j].addValue(in.value(j), in.weight()
                            * m_weights[l][i]);
                } else {
                    m_modelNormal[i][j][0] += (in.value(j)
                            * in.weight() * m_weights[l][i]);
                    m_modelNormal[i][j][2] += in.weight()
                            * m_weights[l][i];
                    m_modelNormal[i][j][1] += (in.value(j)
                            * in.value(j) * in.weight() * m_weights[l][i]);
                }
            }
        }
    }
}
```

这里的代码与初始化的时候是不一样的(当然不一样，一样不就没进步了吗？)，如果是离散值，那么比较简单就是将属于这个簇的权重×样本权重就可以了，如果是连续值，也不复杂，记录一下就可以了，当然这还只是累记。

```
// calcualte mean and std deviation for numeric attributes
for (j = 0; j < m_num_attribs; j++) {
    if (!inst.attribute(j).isNominal()) {
        for (i = 0; i < m_num_clusters; i++) {
            if (m_modelNormal[i][j][2] <= 0) {
                m_modelNormal[i][j][1] = Double.MAX_VALUE;
                //      m_modelNormal[i][j][0] = 0;
                m_modelNormal[i][j][0] = m_minStdDev;
            } else {

                // variance
                m_modelNormal[i][j][1] = (m_modelNormal[i][j][1] -
                    (m_modelNormal[i][j][0] * m_modelNormal[i][j][0] /
                     m_modelNormal[i][j][2])) / (m_modelNormal[i][j][2]);

                if (m_modelNormal[i][j][1] < 0) {
                    m_modelNormal[i][j][1] = 0;
                }

                // std dev
                double minStdD = (m_minStdDevPerAtt != null)
                        ? m_minStdDevPerAtt[j]
                        : m_minStdDev;

                m_modelNormal[i][j][1] = Math
                        .sqrt(m_modelNormal[i][j][1]);

                if ((m_modelNormal[i][j][1] <= minStdD)) {
                    m_modelNormal[i][j][1] =
                         inst.attributeStats(j).numericStats.stdDev;
```

```
                    if ((m_modelNormal[i][j][1] <= minStdD)) {
                        m_modelNormal[i][j][1] = minStdD;
                    }
                }
                if ((m_modelNormal[i][j][1] <= 0)) {
                    m_modelNormal[i][j][1] = m_minStdDev;
                }
                if (Double.isInfinite(m_modelNormal[i][j][1])) {
                    m_modelNormal[i][j][1] = m_minStdDev;
                }

                // mean
                m_modelNormal[i][j][0] /= m_modelNormal[i][j][2];
            }
        }
    }
}
```

这里还要继续对连续属性进行处理，因为上次只是进行了累加。如果 m_NormalModel <= 0 表示没有样本属于这个类，else 的代码很长，其实很简单，第一个是计算方差，然后还是得到有关精度的 minStdD，开方后得到标准差，再根据精度对标准差进行处理，最后是计算均值。

```
double CVLogLikely = -Double.MAX_VALUE;
double templl, tll;
boolean CVincreased = true;
m_num_clusters = 1;
int num_clusters = m_num_clusters;
int i;
Random cvr;
Instances trainCopy;
int numFolds = (m_theInstances.numInstances() < 10) ? m_theInstances
        .numInstances() : 10;

boolean ok = true;
int seed = m_rseed;
int restartCount = 0;
```

只是初始化一点内容。

```
CLUSTER_SEARCH: while (CVincreased) {
    // theInstances.stratify(10);

    CVincreased = false;
    cvr = new Random(m_rseed);
    trainCopy = new Instances(m_theInstances);
    trainCopy.randomize(cvr);
    templl = 0.0;
    for (i = 0; i < numFolds; i++) {
        Instances cvTrain = trainCopy.trainCV(numFolds, i, cvr);
        if (num_clusters > cvTrain.numInstances()) {
            break CLUSTER_SEARCH;
        }
        Instances cvTest = trainCopy.testCV(numFolds, i);
        m_rr = new Random(seed);
        for (int z = 0; z < 10; z++)
            m_rr.nextDouble();
        m_num_clusters = num_clusters;
        EM_Init(cvTrain);
        try {
```

```
            iterate(cvTrain, false);
        } catch (Exception ex) {
            // catch any problems - i.e. empty clusters occuring
            ex.printStackTrace();
            seed++;
            restartCount++;
            ok = false;
            if (restartCount > 5) {
                break CLUSTER SEARCH;
            }
            break;
        }
        try {
            tll = E(cvTest, false);
        } catch (Exception ex) {
            ex.printStackTrace();
            seed++;
            restartCount++;
            ok = false;
            if (restartCount > 5) {
                break CLUSTER SEARCH;
            }
            break;
        }

        templl += tll;
    }

    if (ok) {
        restartCount = 0;
        seed = m rseed;
        templl /= (double) numFolds;

        if (templl > CVLogLikely) {
            CVLogLikely = templl;
            CVincreased = true;
            num clusters++;
        }
    }
}
```

trainCopy 是待聚类的数据，如果没有什么意外那么 numFolds 等于 10，trainCV 和 testCV 相当于是 cross validation 中的第 i 次的训练和测试数据。如果要聚的类比样本还多，当然是 break。再用 EM_Init 初始化，iterate 进行迭代，然后用 E 在 cvTest 上测试得到 log likelihood，如果没有遇到异常，那么求得 templl 看是不是比以前更小的 num_clusters 能取得更好的结果，如果没有取得那么整个循环就结束了。