Weka[9] SimpleKMeans 源代码分析

作者：Koala++/屈伟

再看 SimpleKMeans，从 moveCentroid 开始：

```
double[] vals = new double[members.numAttributes()];

// used only for Manhattan Distance
Instances sortedMembers = null;
int middle = 0;
boolean dataIsEven = false;

if (m_DistanceFunction instanceof ManhattanDistance) {
    middle = (members.numInstances() - 1) / 2;
    dataIsEven = ((members.numInstances() % 2) == 0);
    if (m_PreserveOrder) {
        sortedMembers = members;
    } else {
        sortedMembers = new Instances(members);
    }
}
```

注释上也写了，这段代码仅用于 Manhattan Distance，Manhattan Distance 就是|x-y|这样的，这里得到样本数的中间值，和样本数是不是一个偶数。

```
for (int j = 0; j < members.numAttributes(); j++) {

    // in case of Euclidian distance the centroid is the mean point
    // in case of Manhattan distance the centroid is the median point
    // in both cases, if the attribute is nominal, the centroid is the
    // mode
    if (m_DistanceFunction instanceof EuclideanDistance
            || members.attribute(j).isNominal()) {
        vals[j] = members.meanOrMode(j);
    } else if (m_DistanceFunction instanceof ManhattanDistance) {
        // singleton special case
        if (members.numInstances() == 1) {
            vals[j] = members.instance(0).value(j);
        } else {
            sortedMembers.kthSmallestValue(j, middle + 1);
            vals[j] = sortedMembers.instance(middle).value(j);
            if (dataIsEven) {
                sortedMembers.kthSmallestValue(j, middle + 2);
                vals[j] = (vals[j] + sortedMembers.instance(middle + 1)
                        .value(j)) / 2;
            }
        }
    }

    if (updateClusterInfo) {
        m_ClusterMissingCounts[centroidIndex][j] = members
                .attributeStats(j).missingCount;
        m_ClusterNominalCounts[centroidIndex][j] = members
                .attributeStats(j).nominalCounts;
        if (members.attribute(j).isNominal()) {
            if (m_ClusterMissingCounts[centroidIndex][j] >
```

```
                m_ClusterNominalCounts[centroidIndex][j][Utils
                .maxIndex(m_ClusterNominalCounts[centroidIndex][j])]) {
                vals[j] = Instance.missingValue(); // mark mode as
                                                   // missing
            }
        } else {
            if (m_ClusterMissingCounts[centroidIndex][j] == members
                    .numInstances()) {
                vals[j] = Instance.missingValue(); // mark mean as
                                                   // missing
            }
        }
    }
}
```

注释中的 mean，median，mode，这均值，中位数，众数。先看一下 meanOrMode 这个函数：

```
/**
 * Returns the mean (mode) for a numeric (nominal) attribute as a
 * floating-point value. Returns 0 if the attribute is neither nominal
 * nor numeric. If all values are missing it returns zero.
 * */
public/* @pure@ */double meanOrMode(int attIndex) {

    double result, found;
    int[] counts;

    if (attribute(attIndex).isNumeric()) {
        result = found = 0;
        for (int j = 0; j < numInstances(); j++) {
            if (!instance(j).isMissing(attIndex)) {
                found += instance(j).weight();
                result += instance(j).weight()
                        * instance(j).value(attIndex);
            }
        }
        if (found <= 0) {
            return 0;
        } else {
            return result / found;
        }
    } else if (attribute(attIndex).isNominal()) {
        counts = new int[attribute(attIndex).numValues()];
        for (int j = 0; j < numInstances(); j++) {
            if (!instance(j).isMissing(attIndex)) {
                counts[(int) instance(j).value(attIndex)] += instance(j)
                        .weight();
            }
        }
        return (double) Utils.maxIndex(counts);
    } else {
        return 0;
    }
}
```

注释写到如果是连续值，就是平均值，如果是离散值就是众数。当不是连续值和离散值的时候返回 0。代码是很简单的，如果是连续值，用的是加权平均的公式，如果是离散值，那就找出出现就多的值。

回到刚才的函数，我拷贴贝一点上次写的：有点需要解释的是为什么偶数的是时候用的是 middle+2，这是因为这个 coder 在求 middle 的时候用的是(members.numInstances() - 1) / 2; 这样如果是偶数实际求出来的 middle 就小 1，另一点是因为数数是从 0 数起（讲这个有点污辱人了），所以是+2。这也是我吐血的一点，不就多写两行代码吗？何必把代码写的这么古怪。kthSmallestValue 找出第 kth 个最小值，就是中位数了。

再看 if(updateClusterInfo)下面的代码，得到每个属性的缺失值计数和离散值计数，如果属性是离散值，如果缺失值比最多出现的离散值都多，那么标记众数为缺失值，如果是连续值，如果都是缺失值，那么标记平均数为缺失值。最后把这个值加入到 m_ClusterCentroids 中。

再看 buildClusterer 的代码：

```
m_FullMissingCounts = new int[instances.numAttributes()];
if (m_displayStdDevs) {
    m_FullStdDevs = new double[instances.numAttributes()];
}
m_FullNominalCounts = new int[instances.numAttributes()][0];

m_FullMeansOrMediansOrModes = moveCentroid(0, instances, false);
for (int i = 0; i < instances.numAttributes(); i++) {
    m_FullMissingCounts[i] = instances.attributeStats(i).missingCount;
    if (instances.attribute(i).isNumeric()) {
        if (m_displayStdDevs) {
            m_FullStdDevs[i] = Math.sqrt(instances.variance(i));
        }
        if (m_FullMissingCounts[i] == instances.numInstances()) {
            m_FullMeansOrMediansOrModes[i] = Double.NaN; // mark missing
                                                         // as mean
        }
    } else {
        m_FullNominalCounts[i] = instances.attributeStats(i).
            nominalCounts;
        if (m_FullMissingCounts[i] > m_FullNominalCounts[i][Utils
                .maxIndex(m_FullNominalCounts[i])]) {
            m_FullMeansOrMediansOrModes[i] = -1; // mark missing as most
                                                 // common value
        }
    }
}
```

这里调用了刚才看的 moveCentroid 代码，这里有一个是不是显示标准差的一个 boolean 变量，得到数据集的第 i 个属性的方差开方，如果这个属性全是缺失值，就把它标记为 NaN。再下来，如果是缺失值比别的有的离散值还多，标志为-1，这和刚才看的代码是一样的，这应该也写成 missingValue 的。

```
m_ClusterCentroids = new Instances(instances, m_NumClusters);
int[] clusterAssignments = new int[instances.numInstances()];

if (m_PreserveOrder)
    m_Assignments = clusterAssignments;

m_DistanceFunction.setInstances(instances);

Random RandomO = new Random(getSeed());
int instIndex;
HashMap initC = new HashMap();
DecisionTableHashKey hk = null;
```

```
Instances initInstances = null;
if (m_PreserveOrder)
    initInstances = new Instances(instances);
else
    initInstances = instances;

for (int j = initInstances.numInstances() - 1; j >= 0; j--) {
    instIndex = RandomO.nextInt(j + 1);
    hk = new DecisionTableHashKey(initInstances.instance(instIndex),
           initInstances.numAttributes(), true);
    if (!initC.containsKey(hk)) {
        m_ClusterCentroids.add(initInstances.instance(instIndex));
        initC.put(hk, null);
    }
    initInstances.swap(j, instIndex);

    if (m_ClusterCentroids.numInstances() == m_NumClusters) {
        break;
    }
}
```

m_ClusterCentroids 初始化大小为 m_NumClusters，这里可不是初始化为前 n_NumClusters 个样本，下面的 for 是产生随机点的代码，用 DecisionTableHashKey 产生随机得到的 instance，如果这个样本以前就被加入过中心点集合中，当然就不再加了，如果不是就加入，并加入它的 key，循环直到用户指定的中心点数的中心点都被初始指定。

While(!converged)的代码内：

```
emptyClusterCount = 0;
m_Iterations++;
converged = true;
for (i = 0; i < instances.numInstances(); i++) {
    Instance toCluster = instances.instance(i);
    int newC = clusterProcessedInstance(toCluster, true);
    if (newC != clusterAssignments[i]) {
        converged = false;
    }
    clusterAssignments[i] = newC;
}
```

m_Iterations 是记录迭代了多少次，后面后判断是不是到了指定的最大迭代次数，这里对所有的数据进行循环，如果 clusterProcessedInstance 得到的新的簇和以前得到的不一样，那么就没有收敛。把这个簇赋给 clusterAssignments[i]。

```
// update centroids
m_ClusterCentroids = new Instances(instances, m_NumClusters);
for (i = 0; i < m_NumClusters; i++) {
    tempI[i] = new Instances(instances, 0);
}
for (i = 0; i < instances.numInstances(); i++) {
    tempI[clusterAssignments[i]].add(instances.instance(i));
}
for (i = 0; i < m_NumClusters; i++) {
    if (tempI[i].numInstances() == 0) {
        // empty cluster
        emptyClusterCount++;
    } else {
        moveCentroid(i, tempI[i], true);
    }
```

```
}

if (emptyClusterCount > 0) {
    m_NumClusters -= emptyClusterCount;
    if (converged) {
        Instances[] t = new Instances[m_NumClusters];
        int index = 0;
        for (int k = 0; k < tempI.length; k++) {
            if (tempI[k].numInstances() > 0) {
                t[index++] = tempI[k];
            }
        }
        tempI = t;
    } else {
        tempI = new Instances[m_NumClusters];
    }
}
```

tempI 是记录每个簇里的样本的，tempI[clusterAssignments[i]]就是第 clusterAssignments[i]个簇的样本集，第三个 for 是如果一个簇是空的，将记录空簇的变量 emptyClusterCount 累加，或用 moveCentroid 移动中心点。再下面，如果有空簇，改变 m_NumClusters 的数量，如果收敛了，那么就把不是空数据集的数据集放到 tempI 中，没有收敛就只是改变 tempI 的大小。

```
if (m_Iterations == m_MaxIterations)
    converged = true;

if (!converged) {
    m_squaredErrors = new double[m_NumClusters];
    m_ClusterNominalCounts = new int[m_NumClusters][instances
            .numAttributes()][0];
}
```

如果已经达到了最大迭代次数 m_MaxIterations，如果没有收敛，重置这两个变量。

```
/**
 * clusters an instance that has been through the filters
 **/
private int clusterProcessedInstance(Instance instance, boolean
updateErrors) {
    double minDist = Integer.MAX_VALUE;
    int bestCluster = 0;
    for (int i = 0; i < m_NumClusters; i++) {
        double dist = m_DistanceFunction.distance(instance,
                m_ClusterCentroids.instance(i));
        if (dist < minDist) {
            minDist = dist;
            bestCluster = i;
        }
    }
    if (updateErrors) {
        if (m_DistanceFunction instanceof EuclideanDistance) {
            // Euclidean distance to Squared Euclidean distance
            minDist *= minDist;
        }
        m_squaredErrors[bestCluster] += minDist;
    }
    return bestCluster;
}
```

在 m_NumClusters 中找与这个样本最近的中心点，返回。如果要更新误差，就将误差累

加到 squaredErrors 上。

```
public int clusterInstance(Instance instance) throws Exception {
    Instance inst = null;
    if (!m dontReplaceMissing) {
        m ReplaceMissingFilter.input(instance);
        m ReplaceMissingFilter.batchFinished();
        inst = m ReplaceMissingFilter.output();
    } else {
        inst = instance;
    }

    return clusterProcessedInstance(inst, false);
}
```

这里就是简单地得到样本是哪个簇的代码了。