Weka[8] ID3 源代码分析

作者：Koala++/屈伟

这次介绍一下 Id3 源码，这次用 Weka 的源码介绍一下。首先 Id3 是继承于 Classifier 的：

```
public class Id3 extends Classifier
```

Id3[]成员变量是递归保存树的变量，数据中每一个元素都是当前结点的子结点。

```
/** The node's successors. */
private Id3[] m_Successors;
```

Attribute 是属性类，m_Attribute 是分裂属性

```
/** Attribute used for splitting. */
private Attribute m_Attribute;
```

如果当前结果是叶子结点，m_ClassValue 是类别，到底 double 代表什么，以前讲过了，一会再讲

```
/** Class value if node is leaf. */
private double m_ClassValue;
```

Distribution 表示的是这个结点属于某个类别的概率，如 m_Distribution[0] == 0.1 表示当前结点属于类别 0 的概率为 0.1

```
/** Class distribution if node is leaf. */
private double[] m_Distribution;
```

数据集的类别，以前也讲过了

```
/** Class attribute of dataset. */
private Attribute m_ClassAttribute;
```

以前也讲过，继承自 Classifier 类的类都要实现 buildClassifier 函数。

```
public void buildClassifier(Instances data) throws Exception {

    // can classifier handle the data?
    getCapabilities().testWithFail(data);

    // remove instances with missing class
    data = new Instances(data);
    data.deleteWithMissingClass();

    makeTree(data);
}
```

getCapabilities().testWithFail(data)是判断是否 ID3 能处理选择的数据集，比如什么连续属性，类别索引没有设置等等。

而 data.deleteWithMissingClass 则是删除有缺失样本的函数，具体代码如下：

```
for (int i = 0; i < numInstances(); i++) {
    if (!instance(i).isMissing(attIndex)) {
        newInstances.addElement(instance(i));
    }
}
    m_Instances = newInstances;
```

简单一点，不用为上面的东西分心，关注 makeTree(data)就行了：

```
// Compute attribute with maximum information gain.
double[] infoGains = new double[data.numAttributes()];
Enumeration attEnum = data.enumerateAttributes();
while (attEnum.hasMoreElements()) {
```

```
    Attribute att = (Attribute) attEnum.nextElement();
    infoGains[att.index()] = computeInfoGain(data, att);
}
    m_Attribute = data.attribute(Utils.maxIndex(infoGains));
```

infoGains 保存每个属性的信息增益（IG），枚举每个属性，用 computeInfoGain 函数计算信息增益值。Util.maxIndex 返回信息增益最大的下标，这个属性作为分裂属性保存在 m_Attribute 成员变量中。

```
// Make leaf if information gain is zero.
if (Utils.eq(infoGains[m_Attribute.index()], 0)) {
    m_Attribute = null;
    m_Distribution = new double[data.numClasses()];
    Enumeration instEnum = data.enumerateInstances();
    while (instEnum.hasMoreElements()) {
        Instance inst = (Instance) instEnum.nextElement();
        m_Distribution[(int) inst.classValue()]++;
    }
        Utils.normalize(m_Distribution);
        m_ClassValue = Utils.maxIndex(m_Distribution);
        m_ClassAttribute = data.classAttribute();
}
```

当信息增益等于 0 时为叶子结点（这为什么再讲，就没完了）。

m_Attribute = null，已经不用再分裂了，所以为 null。刚才也说过 m_Distribution 保存的是当前结点属于类别的概率，所以数组大小于 data.numClasses()。枚举当前结点中的每一个样本，inst.classValue()就是 inst 样本的类别值，请注意不要以为类别都是”good”,”bad”之类的值，而是第一个类别就是 0，第二个类别就是 1 的 double 型。

m_Distribution[(int) inst.classValue()]++;也就是将每个样本的相应的下标加 1，比如当前叶子结点有 10 个样本，9 个属于第一个类别，1 个属于第五个类别，则 m_Distribution[0]=9,m_Distribution[4]=1。

Utils.normalize(m_Distribution); 简单地理解为归一化吧，刚在例子也就是 m_Distribution[0]=9,m_Distribution[4]=0.1。（提醒一下，这个例子应该是不会发生的，包括开始的那个例子，为什么就不解释了）

m_ClassValue = Utils.maxIndex(m_Distribution);属于哪个类别的概率最高，那当然就是哪个类别，在刚才失败的例子中也就是 m_ClassValue == 0。

m_ClassAtrribute 也就是类别属性。

```
// Otherwise create successors.
else {
    Instances[] splitData = splitData(data, m_Attribute);
    m_Successors = new Id3[m_Attribute.numValues()];
    for (int j = 0; j < m_Attribute.numValues(); j++) {
        m_Successors[j] = new Id3();
        m_Successors[j].makeTree(splitData[j]);
    }
}
```

如果信息增益不是 0 那么分裂当前结点，在 splitData(data, m_Attribute)函数中，data 被分裂成 m_Attribute 离散值个子结点，比如 m_Attribute 有三种取值”green”，”red”，”blue”，则 splitData 将 data 分成 3 部分到 splitData 中。

在例子中当前结点有 3 个子结点，则 m_Attribute.numValues()==3，递归地 makeTree。（再讲清楚点，也就是每一个子树都是一个决策树，所以 new Id3()）

这里有一个问题，如果在例中，当前结点的 green 子结点没有样本，那么：

```
// Check if no instances have reached this node.
if (data.numInstances() == 0) {
```

```
    m_Attribute = null;
    m_ClassValue = Instance.missingValue();
    m_Distribution = new double[data.numClasses()];
    return;
}
```

这就没什么好解释的了。

对于分类一个样本：

```
if (m_Attribute == null) {
    return m_ClassValue;
} else {
    return m_Successors[(int) instance.value(m_Attribute)]
        .classifyInstance(instance);
}
```

当然也是递归了，如果已经到达叶子结点了，那直接返回类别值。不是叶子结点，那么根据样本属性值到相应的子结点，再调用 classifyInstance(instance)。

还有 distributionForInstance 也差不多，不作解释了：

```
if (m_Attribute == null) {
    return m_Distribution;
} else {
    return m_Successors[(int) instance.value(m_Attribute)]
            .distributionForInstance(instance);
}
```

computeInfoGain 和 computeEntropy 两个函数也不解释了，不过还是应该看一下，有时候还能用的着（最重要的其实是，这都不懂，也过份了点吧）。

分裂结点的函数：

```
private Instances[] splitData(Instances data, Attribute att) {
    Instances[] splitData = new Instances[att.numValues()];
    for (int j = 0; j < att.numValues(); j++) {
        splitData[j] = new Instances(data, data.numInstances());
    }
    Enumeration instEnum = data.enumerateInstances();
    while (instEnum.hasMoreElements()) {
        Instance inst = (Instance) instEnum.nextElement();
        splitData[(int) inst.value(att)].add(inst);
    }
    for (int i = 0; i < splitData.length; i++) {
        splitData[i].compactify();
    }
    return splitData;
}
```

将 data 分裂成 att.numValues()个子结点，inst.value(att)就是根据 inst 样本 att 属性值将 inst 样本分成相应的子结点中。（确切点，也不是子结点，一个 Instances 数组元素）