Weka[11] J48 源代码分析

作者：Koala++/屈伟

这次介绍一下 J48 的源码，分析 J48 的源码似乎真还是有用的，同学改造 J48 写过 VFDT，我自己用 J48 进行特征选择（当然很失败）。

J48 的 buildClassfier 函数：

```
public void buildClassifier(Instances instances) throws Exception {
    ModelSelection modSelection;

    if (m_binarySplits)
        modSelection = new BinC45ModelSelection(m_minNumObj, instances);
    else
        modSelection = new C45ModelSelection(m_minNumObj, instances);
    if (!m_reducedErrorPruning)
        m_root = new C45PruneableClassifierTree(modSelection,
            !m_unpruned, m_CF, m_subtreeRaising, !m_noCleanup);
    else
        m_root = new PruneableClassifierTree(modSelection, !m_unpruned,
                    m_numFolds, !m_noCleanup, m_Seed);
    m_root.buildClassifier(instances);
    if (m_binarySplits) {
        ((BinC45ModelSelection) modSelection).cleanup();
    } else {
        ((C45ModelSelection) modSelection).cleanup();
    }
}
```

在 NBTree 中已经介绍过了，ModelSelection 是决定决策树的模型类，前面两个 if，一个是判断连续属性是否只分出两个子结点，另一个判断是否最后剪枝。m_root 是一个 ClassifierTree 对象，它调用 buildClassifier 函数。这里列出这个函数：

```
public void buildClassifier(Instances data) throws Exception {
    // can classifier tree handle the data?
    getCapabilities().testWithFail(data);

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

    buildTree(data, false);
}
```

有注释也没什么好说的，直接看最后一个函数 buildTree：

```
public void buildTree(Instances data, boolean keepData) throws Exception
{
    Instances[] localInstances;

    if (keepData) {
        m_train = data;
    }
    m_test = null;
    m_isLeaf = false;
    m_isEmpty = false;
    m_sons = null;
    m_localModel = m_toSelectModel.selectModel(data);
```

```
    if (m_localModel.numSubsets() > 1) {
        localInstances = m_localModel.split(data);
        data = null;
        m_sons = new ClassifierTree[m_localModel.numSubsets()];
        for (int i = 0; i < m_sons.length; i++) {
            m_sons[i] = getNewTree(localInstances[i]);
            localInstances[i] = null;
        }
    } else {
        m_isLeaf = true;
        if (Utils.eq(data.sumOfWeights(), 0))
            m_isEmpty = true;
        data = null;
    }
}
```

这里的 selectModel 函数，如果看过 NBTree 一篇的读者应该不会太陌生，selectModel 简单地说就是如果不符合分裂的条件就返回 NoSplit，如果符合分裂的条件，则从 currentModel 数组中选出 bestModel 返回。

这最要注意的是 selectModel 也不只是决定哪个属性分裂，其实到底如何分裂已经在这个函数里算里出来了。

我把 selectModel 拆开来讲解

```
// Check if all Instances belong to one class or if not
// enough Instances to split.
checkDistribution = new Distribution(data);
noSplitModel = new NoSplit(checkDistribution);
if (Utils.sm(checkDistribution.total(), 2 * m_minNoObj)
        || Utils.eq(checkDistribution.total(), checkDistribution
                .perClass(checkDistribution.maxClass())))
                return noSplitModel;
```

2 * m_minNoObj 表示至有有这么多样本才可以分裂，原因很简单，因为一个结点至少分出两个子结点，每个子结点至少有 m_minNoObj 个样本，第二个是或条件是表示是否这个结点上所有的样本都属于同一类别，也就是这个结点总的权重是否等于这个最多类别的权重。

```
// Check if all attributes are nominal and have a lot of values.
if (m_allData != null) {
    Enumeration enu = data.enumerateAttributes();
    while (enu.hasMoreElements()) {
        attribute = (Attribute) enu.nextElement();
        if ((attribute.isNumeric())
            || (Utils.sm((double) attribute.numValues(),
                (0.3 * (double) m_allData.numInstances())))) {
                    multiVal = false;
                    break;
                }
        }
}
```

判断是否有很多不同的属性值，标准就是如果有一个属性的属性值小多于总样本数*0.3，那么就是不是 multiVal。

```
currentModel = new C45Split[data.numAttributes()];
sumOfWeights = data.sumOfWeights();

// For each attribute.
for (i = 0; i < data.numAttributes(); i++) {
```

```
    // Apart from class attribute.
    if (i != (data).classIndex()) {

    // Get models for current attribute.
    currentModel[i] = new C45Split(i, m minNoObj, sumOfWeights);
    currentModel[i].buildClassifier(data);

    // Check if useful split for current attribute
    // exists and check for enumerated attributes with
    // a lot of values.
    if (currentModel[i].checkModel())
        if (m allData != null) {
            if ((data.attribute(i).isNumeric())
                || (multiVal || Utils.sm((double) data
                    .attribute(i).numValues(),
                     (0.3 * (double) m allData.numInstances()))))) {
                          averageInfoGain = averageInfoGain
                                 + currentModel[i].infoGain();
                          validModels++;
                      }
                } else {
                      averageInfoGain = averageInfoGain
                          + currentModel[i].infoGain();
                      validModels++;
                }
        } else
            currentModel[i] = null;
    }
```

里面重要的两句就是：

```
// Get models for current attribute.
currentModel[i] = new C45Split(i, m minNoObj, sumOfWeights);
currentModel[i].buildClassifier(data);
```

其它的也没有什么，求一下 averageInfoGain 和 validModels。checkModel 如果可以分出子结点则为真。

这里是 C45Split 类的成员函数 buildClassfier 被调用，列出它的代码：

```
public void buildClassifier(Instances trainInstances) throws Exception
{
    // Initialize the remaining instance variables.
    m numSubsets = 0;
    m splitPoint = Double.MAX_VALUE;
    m infoGain = 0;
    m gainRatio = 0;

    // Different treatment for enumerated and numeric
    // attributes.
    if (trainInstances.attribute(m attIndex).isNominal()) {
        m complexityIndex = trainInstances.attribute(m attIndex)
            .numValues();
      m index = m complexityIndex;
      handleEnumeratedAttribute(trainInstances);
    }else{
      m complexityIndex = 2;
      m index = 0;
      trainInstances.sort(trainInstances.attribute(m attIndex));
      handleNumericAttribute(trainInstances);
    }
```

```
  }
```

这里 handleEnumerateAttribute 和 handleNumericAttribute 是决定到底是哪一个属性分裂(m_attIndex)和分裂出几个子结点的函数(m_numSubsets)。这里的 m_comlexity 就是指分可以分裂出多少子结点。如果是连续属性就是 2。再看一下 handleEnumeratedAttribute 函数：

```
private void handleEnumeratedAttribute(Instances trainInstances)
throws Exception {

    Instance instance;

    m distribution = new Distribution(m complexityIndex,
        trainInstances.numClasses());

    // Only Instances with known values are relevant.
    Enumeration enu = trainInstances.enumerateInstances();
    while (enu.hasMoreElements()) {
        instance = (Instance) enu.nextElement();
        if (!instance.isMissing(m attIndex))
            m distribution.add((int) instance.value(m attIndex),
                instance);
    }

    // Check if minimum number of Instances in at least two
    // subsets.
    if (m distribution.check(m minNoObj)) {
        m numSubsets = m complexityIndex;
        m infoGain = infoGainCrit.splitCritValue(m distribution,
                m sumOfWeights);
        m gainRatio = gainRatioCrit.splitCritValue(m distribution,
                m sumOfWeights, m infoGain);
    }
}
// Current attribute is a numeric attribute.
m distribution = new Distribution(2, trainInstances.numClasses());

// Only Instances with known values are relevant.
Enumeration enu = trainInstances.enumerateInstances();
i = 0;
while (enu.hasMoreElements()) {
    instance = (Instance) enu.nextElement();
    if (instance.isMissing(m attIndex))
        break;
    m distribution.add(1, instance);
    i++;
}

firstMiss = i;
```

已经讲过了，如果是连续属性就分出两个子结点，也就是 Distribution 的第一个参数。枚举所有样本，因为在调用 HandleNumericAttribute 之间已经对数据集根据 m_attIndex 排序过，所以缺失数据都在最后。也就是 firstMiss 是在 m_attIndex 上有确定值的样本个数＋1。在 while 循环中，把所有的样本都先放到 bag 1 中(add(1,instance))。还是列出来一下吧。

```
public final void add(int bagIndex, Instance instance) throws Exception
{
    int classIndex;
    double weight;
```

```
    classIndex = (int) instance.classValue();
    weight = instance.weight();
    m_perClassPerBag[bagIndex][classIndex] =
        m_perClassPerBag[bagIndex][classIndex] + weight;
    m_perBag[bagIndex] = m_perBag[bagIndex] + weight;
    m_perClass[classIndex] = m_perClass[classIndex] + weight;
    totaL = totaL + weight;
}
```

也就这个函数也就是根据参数 bagIndex 和样本的类别值 classIndex，三个成员变量 m_perBag, m_perClass, m_perClassPerBag 分别加上样本的权重。

```
// Compute minimum number of Instances required in each subset.
minSplit = 0.1 * (m_distribution.total())
       / ((double) trainInstances.numClasses());
if (Utils.smOrEq(minSplit, m_minNoObj))
    minSplit = m_minNoObj;
else if (Utils.gr(minSplit, 25))
    minSplit = 25;

// Enough Instances with known values?
if (Utils.sm((double) firstMiss, 2 * minSplit))
    return;
```

计算分最小分裂需要的样本数，这些涉及的值在 Quinlan 的论文中没有提到，可能也没有太多的道理，就是如果样本数的 1/10 小于 m_minNoObj 那么最小分裂样本数就是 m_minNoObj，如果大于 25，最小分裂样本数就是 25。

如果 firstMiss 小于 2*minSplit 表示已经不可以再分裂了（为什么刚才已经讲过了）。

```
// Compute values of criteria for all possible split indices.
defaultEnt = infoGainCrit.oldEnt(m_distribution);
while (next < firstMiss) {

    if (trainInstances.instance(next - 1).value(m_attIndex)
        + 1e-5 < trainInstances.instance(next).value(m_attIndex)) {

        // Move class values for all Instances up to next
        // possible split point.
        m_distribution.shiftRange(1, 0, trainInstances, last, next);

        // Check if enough Instances in each subset and compute
        // values for criteria.
        if (Utils.grOrEq(m_distribution.perBag(0), minSplit)
            && Utils.grOrEq(m_distribution.perBag(1), minSplit)) {
            currentInfoGain = infoGainCrit.splitCritValue(
                m_distribution, m_sumOfWeights, defaultEnt);
            if (Utils.gr(currentInfoGain, m_infoGain)) {
                m_infoGain = currentInfoGain;
                splitIndex = next - 1;
            }
            m_index++;
        }
        last = next;
    }
    next++;
}
```

oldEnt 计算没有分裂的信息增益，得到 defaultEnt 注意，刚才是把样本放在了一个 bag 中。然后对所有有确定值的样本进行循环。第一个 if，如果两个属性值太接近，那么选择的分裂点不会有太大的区别，就不进行处理。shiftRange 是把第一个 bag 中下标从 last 到 next-1

的样本移到第 0 个 bag。shiftRange 代码如下：

```
public final void shiftRange(int from, int to, Instances source,
        int startIndex, int lastPlusOne) throws Exception {
    int classIndex;
    double weight;
    Instance instance;
    int i;

    for (i = startIndex; i < lastPlusOne; i++) {
        instance = (Instance) source.instance(i);
        classIndex = (int) instance.classValue();
        weight = instance.weight();
        m_perClassPerBag[from][classIndex] -= weight;
        m_perClassPerBag[to][classIndex] += weight;
        m_perBag[from] -= weight;
        m_perBag[to] += weight;
    }
}
```

很简单就是把对应样本的样本权重从 from bag 中减去，再加到 to bag 中。

转回来，如果 bag 1 和 bag 0 都满足最小分裂样本数，计算在当前分裂点上的信息增益值。如果比上一个最好的分裂点的信息增益高，那么记录下当前的信息增益值为最高信息增益值 m_infoGain，和当前分裂点 splitIndex。

```
// Was there any useful split?
if (m_index == 0)
    return;

// Compute modified information gain for best split.
m_infoGain = m_infoGain - (Utils.log2(m_index) / m_sumOfWeights);
if (Utils.smOrEq(m_infoGain, 0))
    return;

// Set instance variables' values to values for best split.
m_numSubsets = 2;
m_splitPoint = (trainInstances.instance(splitIndex + 1).value(
    m_attIndex) + trainInstances.instance(splitIndex).value(
    m_attIndex)) / 2;
```

如果没有找到任何分裂点，返回，接下来的 m_infoGain 自己到 J.R.Quinlan 的 Improved use of continuous Attributes in C4.5 论文中的第 4 页第二段中找。最后设置有两个结点，分裂点在刚才找到的最好的分裂点与下一个属性值的中点。

```
// In case we have a numerical precision problem we need to choose the
// smaller value
if (m_splitPoint == trainInstances.instance(splitIndex + 1).value(
    m_attIndex)) {
    m_splitPoint = trainInstances.instance(splitIndex).value(
        m_attIndex);
}

// Restore distributioN for best split.
m_distribution = new Distribution(2, trainInstances.numClasses());
m_distribution.addRange(0, trainInstances, 0, splitIndex + 1);
m_distribution.addRange(1, trainInstances, splitIndex + 1, firstMiss);

// Compute modified gain ratio for best split.
m_gainRatio = gainRatioCrit.splitCritValue(m_distribution,
        m_sumOfWeights, m_infoGain);
```

if 是处理精度的细节问题。然后重新通过 addRange 计算 m_distribution，最后计算增益率(Gain Ratio)。

这里看到又有一个新类 Distribution 类，还是要把 Distribution 类讲一下，Distribution 类中有一个 bag 成员变量，它的意思是能有几个子结点。从下面的构造函数看出来的，第一个参数在上面调用它的时候用的就是 m_complexityIndex.

```
public Distribution(int numBags, int numClasses) {
    int i;

    m_perClassPerBag = new double[numBags][0];
    m_perBag = new double[numBags];
    m_perClass = new double[numClasses];
    for (i = 0; i < numBags; i++)
        m_perClassPerBag[i] = new double[numClasses];
    totaL = 0;
}
```

Distribution 的 add 函数就是在相应的属性值上进行统计，太简单了，略过。

回到刚才的 buildTree 函数，如果 numSubsets 返回 1，则表示当前结点不再分裂为叶子结点，如果大于 1，那么调用 split 函数，split 函数只是根据有上次得到的子结点数，并根据 WhichSubset 返回值，把当前结点的样本分到几个子结点去。再对每一个子结点训练一个新子树，到这已经与以前讲的 ID3 有很大的相似了。

可能大家学习的时候都对理论很感兴趣，但看了半天也没看到，有点不解，其实也很好找，当然应该在 handleEnumerateAttribute 和 handleNumericAttribute 中了，也就是 InfoGainSplitCrit 和 GainRatioSplitCrit 两个类。

分裂一个样本与 NBTree 相似，这里不再赘述。

```
// Current attribute is a numeric attribute.
m_distribution = new Distribution(2, trainInstances.numClasses());

// Only Instances with known values are relevant.
Enumeration enu = trainInstances.enumerateInstances();
i = 0;
while (enu.hasMoreElements()) {
    instance = (Instance) enu.nextElement();
    if (instance.isMissing(m_attIndex))
        break;
    m_distribution.add(1, instance);
    i++;
}

firstMiss = i;
```

已经讲过了，如果是连续属性就分出两个子结点，也就是 Distribution 的第一个参数。枚举所有样本，因为在调用 HandleNumericAttribute 之间已经对数据集根据 m_attIndex 排序过，所以缺失数据都在最后。也就是 firstMiss 是在 m_attIndex 上有确定值的样本个数＋1。在 while 循环中，把所有的样本都先放到 bag 1 中(add(1,instance))。还是列出来一下吧。

```
public final void add(int bagIndex, Instance instance) throws Exception
{
    int classIndex;
    double weight;

    classIndex = (int) instance.classValue();
    weight = instance.weight();
    m_perClassPerBag[bagIndex][classIndex] =
```

```
        m_perClassPerBag[bagIndex][classIndex] + weight;
    m_perBag[bagIndex] = m_perBag[bagIndex] + weight;
    m_perClass[classIndex] = m_perClass[classIndex] + weight;
    totaL = totaL + weight;
}
```

也就这个函数也就是根据参数 bagIndex 和样本的类别值 classIndex，三个成员变量 m_perBag, m_perClass, m_perClassPerBag 分别加上样本的权重。

```
// Compute minimum number of Instances required in each subset.
minSplit = 0.1 * (m_distribution.total())
       / ((double) trainInstances.numClasses());
if (Utils.smOrEq(minSplit, m_minNoObj))
    minSplit = m_minNoObj;
else if (Utils.gr(minSplit, 25))
    minSplit = 25;

// Enough Instances with known values?
if (Utils.sm((double) firstMiss, 2 * minSplit))
    return;
```

计算分最小分裂需要的样本数，这些涉及的值在 Quinlan 的论文中没有提到，可能也没有太多的道理，就是如果样本数的 1/10 小于 m_minNoObj 那么最小分裂样本数就是 m_minNoObj，如果大于 25，最小分裂样本数就是 25。

如果 firstMiss 小于 2*minSplit 表示已经不可以再分裂了（为什么刚才已经讲过了）。

```
// Compute values of criteria for all possible split indices.
defaultEnt = infoGainCrit.oldEnt(m_distribution);
while (next < firstMiss) {

    if (trainInstances.instance(next - 1).value(m_attIndex)
        + 1e-5 < trainInstances.instance(next).value(m_attIndex)) {

        // Move class values for all Instances up to next
        // possible split point.
        m_distribution.shiftRange(1, 0, trainInstances, last, next);

        // Check if enough Instances in each subset and compute
        // values for criteria.
        if (Utils.grOrEq(m_distribution.perBag(0), minSplit)
            && Utils.grOrEq(m_distribution.perBag(1), minSplit)) {
            currentInfoGain = infoGainCrit.splitCritValue(
                m_distribution, m_sumOfWeights, defaultEnt);
            if (Utils.gr(currentInfoGain, m_infoGain)) {
                m_infoGain = currentInfoGain;
                splitIndex = next - 1;
            }
            m_index++;
        }
        last = next;
    }
    next++;
}
```

oldEnt 计算没有分裂的信息增益，得到 defaultEnt 注意，刚才是把样本放在了一个 bag 中。然后对所有有确定值的样本进行循环。第一个 if，如果两个属性值太接近，那么选择的分裂点不会有太大的区别，就不进行处理。shiftRange 是把第一个 bag 中下标从 last 到 next-1 的样本移到第 0 个 bag。shiftRange 代码如下：

```
public final void shiftRange(int from, int to, Instances source,
        int startIndex, int lastPlusOne) throws Exception {
```

```
    int classIndex;
    double weight;
    Instance instance;
    int i;

    for (i = startIndex; i < lastPlusOne; i++) {
        instance = (Instance) source.instance(i);
        classIndex = (int) instance.classValue();
        weight = instance.weight();
        m_perClassPerBag[from][classIndex] -= weight;
        m_perClassPerBag[to][classIndex] += weight;
        m_perBag[from] -= weight;
        m_perBag[to] += weight;
    }
}
```

很简单就是把对应样本的样本权重从 from bag 中减去，再加到 to bag 中。

转回来，如果 bag 1 和 bag 0 都满足最小分裂样本数，计算在当前分裂点上的信息增益值。如果比上一个最好的分裂点的信息增益高，那么记录下当前的信息增益值为最高信息增益值 m_infoGain，和当前分裂点 splitIndex。

```
// Was there any useful split?
if (m_index == 0)
    return;

// Compute modified information gain for best split.
m_infoGain = m_infoGain - (Utils.log2(m_index) / m_sumOfWeights);
if (Utils.smOrEq(m_infoGain, 0))
    return;

// Set instance variables' values to values for best split.
m_numSubsets = 2;
m_splitPoint = (trainInstances.instance(splitIndex + 1).value(
    m_attIndex) + trainInstances.instance(splitIndex).value(
    m_attIndex)) / 2;
```

如果没有找到任何分裂点，返回，接下来的 m_infoGain 自己到 J.R.Quinlan 的 Improved use of continuous Attributes in C4.5 论文中的第 4 页第二段中找。最后设置有两个结点，分裂点在刚才找到的最好的分裂点与下一个属性值的中点。

```
// In case we have a numerical precision problem we need to choose the
// smaller value
if (m_splitPoint == trainInstances.instance(splitIndex + 1).value(
    m_attIndex)) {
    m_splitPoint = trainInstances.instance(splitIndex).value(
        m_attIndex);
}

// Restore distributioN for best split.
m_distribution = new Distribution(2, trainInstances.numClasses());
m_distribution.addRange(0, trainInstances, 0, splitIndex + 1);
m_distribution.addRange(1, trainInstances, splitIndex + 1, firstMiss);

// Compute modified gain ratio for best split.
m_gainRatio = gainRatioCrit.splitCritValue(m_distribution,
        m_sumOfWeights, m_infoGain);
```

if 是处理精度的细节问题。然后重新通过 addRange 计算 m_distribution，最后计算增益率(Gain Ratio)。