Weka[23] PART 源代码分析

作者：Koala++/屈伟

rose 璐问我这个算法，我才去看它的论文和算法的，因为个人时间有限，分析的有些粗糙。

请先把论文 Generating Accurate Rule Sets Without Global Optimization 看一下。PART 在 classifiers.rules 包下面，我们直接从 buildClassifier 开始。

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

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

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

    ModelSelection modSelection;

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

如果以前看过 J48 的代码，相信看到这段代码不会陌生，不同的地方是以前是 C45PruneableClassifierTree，现在是 MakeDecList。

因为讲树分类器的次数太多(ID3，J48，NBTree，REPTree)，所以就不想再讲的太细了，我们直接看 m_root.buildClassifier(instances)这句话。

```
theRules = new Vector();
if ((reducedErrorPruning) && !(unpruned)) {
```

```
    Random random = new Random(m_seed);
    data.randomize(random);
    data.stratify(numSetS);
    oldGrowData = data.trainCV(numSetS, numSetS - 1, random);
    oldPruneData = data.testCV(numSetS, numSetS - 1);
} else {
    oldGrowData = data;
    oldPruneData = null;
                                    }
```

如果要剪枝，就用 trainCV 和 testCV 把数据集分成 oldGrowData 和 oldPruneData，如果不需要剪枝，那么 oldGrowData 就等于 Data，这已经在 REPTree 中讲过了。

```
while (Utils.gr(oldGrowData.numInstances(), 0)) {

    // Create rule
    if (unpruned) {
       currentRule = new ClassifierDecList(toSelectModeL, minNumObj);
       ((ClassifierDecList) currentRule).buildRule(oldGrowData);
    } else if (reducedErrorPruning) {
       currentRule = new PruneableDecList(toSelectModeL, minNumObj);
       ((PruneableDecList) currentRule).buildRule(oldGrowData,
             oldPruneData);
    } else {
       currentRule = new C45PruneableDecList(toSelectModeL, CF,
             minNumObj);
       ((C45PruneableDecList) currentRule).buildRule(oldGrowData);
    }
    numRules++;

    // Remove instances from growing data
    newGrowData = new Instances(oldGrowData,
       oldGrowData.numInstances());
    Enumeration enu = oldGrowData.enumerateInstances();
    while (enu.hasMoreElements()) {
       Instance instance = (Instance) enu.nextElement();
       currentWeight = currentRule.weight(instance);
       if (Utils.sm(currentWeight, 1)) {
           instance.setWeight(instance.weight() * (1 - currentWeight));
           newGrowData.add(instance);
       }
    }
    newGrowData.compactify();
    oldGrowData = newGrowData;

    // Remove instances from pruning data
```

```
    if ((reducedErrorPruning) && !(unpruned)) {
       newPruneData = new Instances(oldPruneData, oldPruneData
              .numInstances());
       enu = oldPruneData.enumerateInstances();
       while (enu.hasMoreElements()) {
           Instance instance = (Instance) enu.nextElement();
           currentWeight = currentRule.weight(instance);
           if (Utils.sm(currentWeight, 1)) {
              instance.setWeight(instance.weight()
                     * (1 - currentWeight));
              newPruneData.add(instance);
           }
       }
       newPruneData.compactify();
       oldPruneData = newPruneData;
    }
    theRules.addElement(currentRule);
                                    }
```

我们可以看到一前几行，知道一共有三种规则(Rule)产生的函数，我们看一个最简单的，也就是第一个，不剪枝的。ClassifierDecList 这个类的 buildRule 函数：

```
public void buildRule(Instances data) throws Exception {
    buildDecList(data, false);

    cleanup(new Instances(data, 0));
                                    }
```

不用想又是一个递归算法，我们看 buildDecList 吧。我还是把这个函数拆开：

```
sumOfWeights = data.sumOfWeights();
noSplit = new NoSplit(new Distribution((Instances) data));
if (leaf)
    m_localModel = noSplit;
else
            m_localModel = m_toSelectModel.selectModel(data);
```

如果是传进来的参数 leaf 为真，表示已经是一个叶子结点了，就不分裂了(noSplit)，如果不是叶子结点，就用 selectModel 函数，这个函数已经在 J48 中详细讲过了，不讲了。

```
if (m_localModel.numSubsets() > 1) {
    localInstances = m_localModel.split(data);
    data = null;
    m_sons = new ClassifierDecList[m_localModel.numSubsets()];
    i = 0;
    do {
       i++;
       ind = chooseIndex();
       if (ind == -1) {
           for (j = 0; j < m_sons.length; j++)
```

```
                if (m_sons[j] == null)
                    m_sons[j] = getNewDecList(localInstances[j], true);
            if (i < 2) {
                m_localModel = noSplit;
                m_isLeaf = true;
                m_sons = null;
                if (Utils.eq(sumOfWeights, 0))
                    m_isEmpty = true;
                return;
            }
            ind = 0;
            break;
        } else
            m_sons[ind] = getNewDecList(localInstances[ind], false);
    } while ((i < m_sons.length) && (m_sons[ind].m_isLeaf));

    // Choose rule
    indeX = chooseLastIndex();
} else {
    m_isLeaf = true;
    if (Utils.eq(sumOfWeights, 0))
        m_isEmpty = true;
                                    }
```

如果子集数大于 1(numSubsets() > 1)，就将 data 分裂(split)，但是这里我们看到了一个很陌生的函数 chooseIndex()：

```
public final int chooseIndex() {

    int minIndex = -1;
    double estimated, min = Double.MAX_VALUE;
    int i, j;

    for (i = 0; i < m_sons.length; i++)
        if (son(i) == null) {
            if (Utils.sm(localModel().distribution().perBag(i),
                    (double) m_minNumObj))
                estimated = Double.MAX_VALUE;
            else {
                estimated = 0;
                for (j = 0; j < localModel().distribution().numClasses();
                    j++)
                    estimated -= m_splitCrit.logFunc(localModel()
                            .distribution().perClassPerBag(i, j));
                estimated += m_splitCrit.logFunc(localModel()
                        .distribution().perBag(i));
```

```
                estimated /= localModel().distribution().perBag(i);
            }
            if (Utils.smOrEq(estimated, 0))
                return i;
            if (Utils.sm(estimated, min)) {
                min = estimated;
                minIndex = i;
            }
        }

    return minIndex;
                                    }
```

这个函数也就是论文图 3 中所讲的那样，找到最小熵的结点进行分裂，看第一个 if，希望大家还知道，小于 m_minNumObj 样本数的结点是无法分裂的，所以也选不到它去，else 那里面是计算熵的算法，如果你真是不知道，知道这一点也就足够了，最后再下来一个 if，都小于等于 0 了，没法再小了，直接返回了。最后一个 if 如果这次计算的熵值小于 min，那么替换它，并且最后返回有最小熵值的结点下标。

回到 buildRule 函数，如果 chooseIndex 返回的(ind)是-1，那么就把那么 m_son 中为空的结点全部设为根结点，再向下，i<2 意味着，第一次就没找到一个可以分裂的结点，只好把当前的这个结点设为根结点。如果 ind 不是-1，那么 m_sons[ind] =这句话就开始递归了。我们再看一下我们陌生的一个函数

```
public final int chooseLastIndex() {

    int minIndex = 0;
    double estimated, min = Double.MAX_VALUE;

    if (!m_isLeaf)
       for (int i = 0; i < m_sons.length; i++)
           if (son(i) != null) {
              if (Utils.grOrEq(localModel().distribution().perBag(i),
                     (double) m_minNumObj)) {
                  estimated = son(i).getSizeOfBranch();
                  if (Utils.sm(estimated, min)) {
                     min = estimated;
                     minIndex = i;
                  }
              }
           }

    return minIndex;
                                    }
```

这个函数是返回子结点中有最多样本的下标，原论文中说的是 Our implementation aims at the most general rule by choosing the leaf that covers the greatest number of instances。其中 getSizeOfBranch 的代码如下：

```
protected double getSizeOfBranch() {

    if (m_isLeaf) {
       return -localModel().distribution().total();
    } else
       return son(indeX).getSizeOfBranch();
                                        }
```

刚才说是最多样本的下标，你可能有点意外，明明是 Utils.sm，可是你看 getSizeOfBranch 中的第一个 return 后面是有负号的，我想这种写法是不是想到样本权重之和为 0 这种情况考虑进去。

```
public double classifyInstance(Instance instance) throws Exception {

    double maxProb = -1;
    double currentProb;
    int maxIndex = 0;
    int j;

    for (j = 0; j < instance.numClasses(); j++) {
       currentProb = getProbs(j, instance, 1);
       if (Utils.gr(currentProb, maxProb)) {
           maxIndex = j;
           maxProb = currentProb;
       }
    }
    if (Utils.eq(maxProb, 0))
       return -1.0;
    else
       return (double) maxIndex;
                                        }
```

看起来与 J48 蛮相似的，我们也知道还需要看的函数就是 getProbs 了（总不至于下面几行看不懂吧）。

```
private double getProbs(int classIndex, Instance instance, double weight)
       throws Exception {

    double[] weights;
    int treeIndex;

    if (m_isLeaf) {
       return weight * localModel().classProb(classIndex, instance, -1);
    } else {
       treeIndex = localModel().whichSubset(instance);
       if (treeIndex == -1) {
           weights = localModel().weights(instance);
           return son(indeX).getProbs(classIndex, instance,
```

```
                    weights[indeX] * weight);
        } else {
            if (treeIndex == indeX) {
               return son(indeX).getProbs(classIndex, instance,
                    weight);
            } else {
               return 0;
            }
        }
    }
                                        }
```

这里其实也与 J48 没有什么区别，我们看一下 else 中的第一个 if，treeIndex==-1 的时候是这表示，在分裂属性上这个样本的属性值是缺失的，weights 数是计算每个叶子结点除总权重的权重，而下面就是递归了。再看 else，如果 treeIndex==index 那么是可以递归的，不然就不符合归则，就返回 0。