Weka[16] OneR 源代码介绍

作者：Koala++/屈伟

OneR 是一个很简单的算法，出自论文：Very simple classification rules perform well on most commonly used datasets，由于论文的风格过于奔放，并且很长，所以我也就没怎么看。基本思想就是对每一个属性都建一个单层的分类器，对这些分类器进行比较，谁分类效果好就作为最终的分类器。

下面还是看 buildClassifier 的代码（删除了部分代码），首先判断是不是就一个属性（一个属性意味着只有一个类别特征），如果是，那就用 ZeroR 算法（如果不知道为什么，看我上一篇）。下面枚举每一个属性，在每一个属性上产生一个 OneRRule 对象 r，下面判断这个 r 是比以前产生的正确的样本数多，如果是则替换。

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

    boolean noRule = true;

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

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

    // only class? -> build ZeroR model
    if (data.numAttributes() == 1) {
        m_ZeroR = new weka.classifiers.rules.ZeroR();
        m_ZeroR.buildClassifier(data);
        return;
    } else {
        m_ZeroR = null;
    }

    // for each attribute ...
    Enumeration enu = instances.enumerateAttributes();
    while (enu.hasMoreElements()) {
        try {
            OneRRule r = newRule((Attribute) enu.nextElement(), data);

            // if this attribute is the best so far, replace the rule
            if (noRule || r.m_correct > m_rule.m_correct) {
                m_rule = r;
            }
            noRule = false;
```

```
        } catch (Exception ex) {
        }
    }
}
```

下面看一下刚才的 newRule 函数，初始化一个 missingValueCounts 数组，数组大小为类别集合的大小。如果当前这个类别是离散的调用 newNominalRule，如果是连续的调用 newNumericRule。下面的几行代码现在可能还有点难理解（理解不了，看完下面的再转回来看），missingValueCounts 保存的是对这个属性缺失值类别值的读数，而 maxIndex 函数返回的就是这个属性缺失时最有时候的类别 Index。再下来 If 判断是否训练集中如果这个属性值缺失的样本，那么 r.m_missingValueClass = -1;如果有，r.m_correct 加上当这个属性缺失情况下最多出现的类别值的出现次数（没办法就是这么难表达）。

```
public OneRRule newRule(Attribute attr, Instances data) throws Exception
{
    OneRRule r;

    // ... create array to hold the missing value counts
    int[] missingValueCounts = new
        int[data.classAttribute().numValues()];

    if (attr.isNominal()) {
        r = newNominalRule(attr, data, missingValueCounts);
    } else {
        r = newNumericRule(attr, data, missingValueCounts);
    }
    r.m_missingValueClass = Utils.maxIndex(missingValueCounts);
    if (missingValueCounts[r.m_missingValueClass] == 0) {
        r.m_missingValueClass = -1; // signal for no missing value class
    } else {
        r.m_correct += missingValueCounts[r.m_missingValueClass];
    }
    return r;
}
```

先看一下离散的情况，初始化一个二维数组，第一维属性的个数，第二维类别值集合的大小。下面对样本进行枚举，如果当前样本该属性值是缺失的，那么 missingValuleCounts 在相应的类别值下标上记数。如果不是缺失的，那种就在这个样本在该属性值的类别值下标上记数（说起来很糊涂，想通了很简单）。接下面这段代码刚开始看的时候，我也糊涂了，其实也很简单。best 就是当一个样本在该属性取值为 value 时，最有可能的类别值。m_correct 就是对这种情况的记数，即在全部样本中，当属性为 attr 时，属性值为 value，类别值是 value 这种情况一共出现了多少次。

```
public OneRRule newNominalRule(Attribute attr, Instances data,
        int[] missingValueCounts) throws Exception {

    // ... create arrays to hold the counts
    int[][] counts = new int[attr.numValues()][data.classAttribute()
```

```
            .numValues()];

    // ... calculate the counts
    Enumeration enu = data.enumerateInstances();
    while (enu.hasMoreElements()) {
        Instance i = (Instance) enu.nextElement();
        if (i.isMissing(attr)) {
            missingValueCounts[(int) i.classValue()]++;
        } else {
            counts[(int) i.value(attr)][(int) i.classValue()]++;
        }
    }

    OneRRule r = new OneRRule(data, attr); // create a new rule
    for (int value = 0; value < attr.numValues(); value++) {
        int best = Utils.maxIndex(counts[value]);
        r.m_classifications[value] = best;
        r.m_correct += counts[value][best];
    }
    return r;
}
```

如果看上面一段都感觉有难度的人，就不要看这一段了。我也很反感属性是连续值的情况，不管怎么说，看都看完了(我晕了一会，查了一下作者的论文里关于连续值的情况，反正他也写的很难懂)。Classifications 里存的是每一段属性取值的类别值（这里用的是段这个词，因为连续值要分成几个段来处理），breakpoints 里存的是在哪个值开始分段，counts 是统计一段上类别值记数。lastInstance 是样本数。

data.sort(attr)至关重要，注释写的是该属性有缺失值的样本将被排到最后，下面的 while 循环先统计该属性缺失值类别值，lastInstance 相应减少。

Bucket(桶)就是我上面讲的相应的段，下面这个比较长的 while 主要是处理分段。看第一个 for，没什么特别的，清空上一次的 counts（注意：前面也说了，是一段的记数值），下一个 do/while 循环，注意一下它的循环结束条件，它的意思要被分段（也就是形成一个桶），至少要有一个类的样本超过 m_minBucketSize。下一个 while，注释说的是如果下面的样本属于如果刚才超过 m_minBucketSize 的类别值，继续加入这个段(装桶)。再下一个 while 注释：如果样本属性值和上面的样本一样，继续加入这个段（装桶）。为什么以上两个 while 这么做？因为样本集已经根据该属性值排过序。第一个 while 可以少分段，第二个 while 因为它们值都是一样的，当然要属于一段。

下面一个 for 循环，选出这段出现最多的类别值为 it（下面代码下面解释）。

```
public OneRRule newNumericRule(Attribute attr, Instances data,
        int[] missingValueCounts) throws Exception {

    // ... can't be more than numInstances buckets
    int[] classifications = new int[data.numInstances()];
    double[] breakpoints = new double[data.numInstances()];
```

```
// create array to hold the counts
int[] counts = new int[data.classAttribute().numValues()];
int correct = 0;
int lastInstance = data.numInstances();

// missing values get sorted to the end of the instances
data.sort(attr);
while (lastInstance > 0
       && data.instance(lastInstance - 1).isMissing(attr)) {
   lastInstance--;
   missingValueCounts[(int)
        data.instance(lastInstance).classValue()]++;
}
int i = 0;
int cl = 0; // index of next bucket to create
int it;
while (i < lastInstance) { // start a new bucket
   for (int j = 0; j < counts.length; j++)
      counts[j] = 0;
   do { // fill it until it has enough of the majority class
      it = (int) data.instance(i++).classValue();
      counts[it]++;
   } while (counts[it] < m_minBucketSize && i < lastInstance);

   // while class remains the same, keep on filling
   while (i < lastInstance
          && (int) data.instance(i).classValue() == it) {
      counts[it]++;
      i++;
   }
   while (i < lastInstance && // keep on while attr value is the same
          (data.instance(i - 1).value(attr) == data.instance(i)
                .value(attr))) {
      counts[(int) data.instance(i++).classValue()]++;
   }
   for (int j = 0; j < counts.length; j++) {
      if (counts[j] > counts[it]) {
         it = j;
      }
   }
   if (cl > 0) { // can we coalesce with previous class?
      if (counts[classifications[cl - 1]] == counts[it]) {
         it = classifications[cl - 1];
      }
```

```
            if (it == classifications[cl - 1]) {
                cl--; // yes!
            }
        }
        correct += counts[it];
        classifications[cl] = it;
        if (i < lastInstance) {
            breakpoints[cl] = (data.instance(i - 1).value(attr) + data
                    .instance(i).value(attr)) / 2;
        }
        cl++;
    }
    if (cl == 0) {
        throw new Exception("Only missing values in the training data!");
    }
    OneRRule r = new OneRRule(data, attr, cl); // new rule with cl branches
    r.m_correct = correct;
    for (int v = 0; v < cl; v++) {
        r.m_classifications[v] = classifications[v];
        if (v < cl - 1) {
            r.m_breakpoints[v] = breakpoints[v];
        }
    }

    return r;
}
```

下面的 if 是判断是否两个段的类别值相同，如果相同就可以合并(coalesce)。第一个 if 看起来比较怪，它其实是想判断是不是这一段里有多个最大值，而其中一个就是上次的最大值，并且没有被认为是最大值。如果是，那么就用上次的最大值来代替。

下来的 correct 和 classification 没什么好讲的，下来一个 if 为什么是两个样本值该属性值加起来除 2，是因为 i 已经加过了，这时做的是这一段结束值与下一段的开始值以中间为界分开。

最后一个 for 就是复制一下，不讲了。

最后一个函数 classifyInstance，如果是 m_ZeroR 分类器，说明只有一个类别属性。下一个 if，如果是缺失值，那么就是 m_rule 的 m_missingValueClass，当然也可能有学习时没有缺失值，分类时有的情况，那么返回 0。如果是离散值，直接返回在属性 m_attr 值的上的类别值，如果是连续值，看它在哪个段上，返回该段上的类别值。

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

    // default model?
    if (m_ZeroR != null) {
        return m_ZeroR.classifyInstance(inst);
    }
```

```
    int v = 0;
    if (inst.isMissing(m_rule.m_attr)) {
        if (m_rule.m_missingValueClass != -1) {
            return m_rule.m_missingValueClass;
        } else {
            return 0; // missing values occur in test but not training set
        }
    }
    if (m_rule.m_attr.isNominal()) {
        v = (int) inst.value(m_rule.m_attr);
    } else {
        while (v < m_rule.m_breakpoints.length
                && inst.value(m_rule.m_attr) >= m_rule.m_breakpoints[v])
{
            v++;
        }
    }
    return m_rule.m_classifications[v];
}
```