Weka[36] InfoGainAttributeEval 源代码分析

作者：Koala++/屈伟

最近要用到特征选择，但需要的特征选择又有点不同，还是看看源码，本文后面介绍了 weka 中熵的计算，它的计算与公式中不太一样，以前给 weka 中文站人的介绍过一次，这次我专门周末把公式敲出来，方便大家看。

从 buildEvaluator 开始看：

```
int classIndex = data.classIndex();
int numInstances = data.numInstances();

if (!m_Binarize) {
    Discretize disTransform = new Discretize();
    disTransform.setUseBetterEncoding(true);
    disTransform.setInputFormat(data);
    data = Filter.useFilter(data, disTransform);
} else {
    NumericToBinary binTransform = new NumericToBinary();
    binTransform.setInputFormat(data);
    data = Filter.useFilter(data, binTransform);
}
```

看是要离散成二个值，还是多个值。

```
int numClasses = data.attribute(classIndex).numValues();

// Reserve space and initialize counters
double[][][] counts = new double[data.numAttributes()][][];
for (int k = 0; k < data.numAttributes(); k++) {
    if (k != classIndex) {
        int numValues = data.attribute(k).numValues();
        counts[k] = new double[numValues + 1][numClasses + 1];
    }
}

// Initialize counters
double[] temp = new double[numClasses + 1];
for (int k = 0; k < numInstances; k++) {
    Instance inst = data.instance(k);
    if (inst.classIsMissing()) {
        temp[numClasses] += inst.weight();
    } else {
        temp[(int) inst.classValue()] += inst.weight();
    }
}
for (int k = 0; k < counts.length; k++) {
    if (k != classIndex) {
        for (int i = 0; i < temp.length; i++) {
            counts[k][0][i] = temp[i];
        }
    }
}
```

Counts 第一维是属性个数，第二列是属性值个数+1，第三列是类别个数+1。将第属性值的第 0 个元素，设为样本权重，且属性值第 0 个元素的类别第 0 个元素为，所有类别缺失样本权重之和。

```
// Get counts
for (int k = 0; k < numInstances; k++) {
    Instance inst = data.instance(k);
    for (int i = 0; i < inst.numValues(); i++) {
        if (inst.index(i) != classIndex) {
            if (inst.isMissingSparse(i) || inst.classIsMissing()) {
                if (!inst.isMissingSparse(i)) {
                    counts[inst.index(i)][(int) inst.valueSparse(i)]
                            [numClasses] += inst.weight();
                    counts[inst.index(i)][0][numClasses] -= inst
                            .weight();
                } else if (!inst.classIsMissing()) {
                    counts[inst.index(i)][data.attribute(inst.index(i))
                            .numValues()][(int) inst.classValue()] += inst
                            .weight();
                    counts[inst.index(i)][0][(int) inst.classValue()]
                            -= inst.weight();
                } else {
                    counts[inst.index(i)][data.attribute(inst.index(i))
                            .numValues()][numClasses] += inst.weight();
                    counts[inst.index(i)][0][numClasses] -= inst
                            .weight();
                }
            } else {
                counts[inst.index(i)][(int) inst.valueSparse(i)]
                        [(int) inst.classValue()] += inst.weight();
                counts[inst.index(i)][0][(int) inst.classValue()]
                        -= inst.weight();
            }
        }
    }
}
```

核心的就是最下面的 else 的第一句，将这个属性的这个属性值的类别值的元素加上它的权重。if(m_missing_merge)就是把那些缺失值平均分到相应的元素中去，懒的细看了，下一个：

```
// Compute info gains
m InfoGains = new double[data.numAttributes()];
for (int i = 0; i < data.numAttributes(); i++) {
    if (i != classIndex) {
        m InfoGains[i] = (ContingencyTables
                .entropyOverColumns(counts[i]) - ContingencyTables
                .entropyConditionedOnRows(counts[i]));
    }
}
```

重要的有两个函数 entropyOverColumns 和 entropyConditionedOnRows：

```
public static double entropyOverColumns(double[][] matrix) {

    double returnValue = 0, sumForColumn, total = 0;

    for (int j = 0; j < matrix[0].length; j++) {
        sumForColumn = 0;
        for (int i = 0; i < matrix.length; i++) {
            sumForColumn += matrix[i][j];
        }
        returnValue = returnValue - lnFunc(sumForColumn);
        total += sumForColumn;
```

```
    }
    if (Utils.eq(total, 0)) {
        return 0;
    }
    return (returnValue + lnFunc(total)) / (total * log2);
}
```

这里要注意一下，其实从名字也反应出来了 Over Columns 是求这个属性的熵，也就是 InfoGain 前面的那一项。

```
public static double entropyConditionedOnRows(double[][] matrix) {

    double returnValue = 0, sumForRow, total = 0;

    for (int i = 0; i < matrix.length; i++) {
        sumForRow = 0;
        for (int j = 0; j < matrix[0].length; j++) {
            returnValue = returnValue + lnFunc(matrix[i][j]);
            sumForRow += matrix[i][j];
        }
        returnValue = returnValue - lnFunc(sumForRow);
        total += sumForRow;
    }
    if (Utils.eq(total, 0)) {
        return 0;
    }
    return -returnValue / (total * log2);
}
```

这一步也就是 InfoGain 公式的后面一项。

这里先以 ID3 为例讲一下如何计算信息熵，weka 中所用的计算有点点不同：

```
private double computeEntropy(Instances data) throws Exception {

    double[] classCounts = new double[data.numClasses()];
    Enumeration instEnum = data.enumerateInstances();
    while (instEnum.hasMoreElements()) {
        Instance inst = (Instance) instEnum.nextElement();
        classCounts[(int) inst.classValue()]++;
    }
    double entropy = 0;
    for (int j = 0; j < data.numClasses(); j++) {
        if (classCounts[j] > 0) {
            entropy -= classCounts[j] * Utils.log2(classCounts[j]);
        }
    }
    entropy /= (double) data.numInstances();
    return entropy + Utils.log2(data.numInstances());
}
```

classCounts 数组不必说，自然是每个类别的样本数。这里设数样本数为 N，类别数为 M，类别 i 的样本数为 C_i (C_i=classCounts[i])。中间的 for 循环用公式表示出来就是：

$$\text{entropy} = -\sum_{i=0}^{M} C_i * \log C_i$$

entropy/numInstance 用公式表示则为：

$$\text{entropy} = (-\sum_{i=0}^{M} C_i * \log C_i)/N$$

将 N 移进去：

$$\text{entropy} = (-\sum_{i=0}^{M} C_i / N * \log C_i)$$

这里令 $P(C_i)$为类别 C_i 的概率，上式等于：

$$\text{entropy} = (-\sum_{i=0}^{M} P(C_i) * \log C_i)$$

最后一步有+log2(numInstance)，即 $\log_2^N$。可以将它视为$(N/N)\ \log_2^N$。

$$\text{entropy} = \left(-\sum_{i=0}^{M} P(C_i) * \log C_i\right) + \left(\frac{N}{N}\right)\log_2^N$$

$N=C_1+\ldots+C_M$。则视为$(N/N)\ \log_2^N=(C_1/N)\ \log_2^N+\ldots+(C_M/N)\ \log_2^N=P(C_1)\ \log_2^N+\ldots+P(C_M)\ \log_2^N$。log(a)-log(b)=log(a/b)。代入后：

$$\text{entropy} = \left(-\sum_{i=0}^{M} P(C_i) * \log C_i/N\right)$$

它与一般看到的公式描述就是一致的了：

$$\text{entropy} = \left(-\sum_{i=0}^{M} P(C_i) * \log P(C_i)\right)$$

再看 entropyOverClumns 中的代码，又略有不同：
除 return 外的代码，公式可表示为：

$$\text{entropy} = -\sum_{i=0}^{M} C_i * \log C_i$$

而最后的一句又可表示为：

$$\text{entropy} = (-\sum_{i=0}^{M} C_i * \ln C_i) + N * \ln N/(N * \ln 2)$$

与刚才的推导方法一样。