# Weka[14] Adaboost 源代码分析

作者：Koala++/屈伟

需要注意的是 AdaBoostM1 继承自 RandomizableIteratedSingleClassifierEnhancer，天哪，这么长的名字，他们也起的出来。

这个要看一下构造函数：

```
public AdaBoostM1() {
    m_Classifier = new weka.classifiers.trees.DecisionStump();
}
```

这里的 m_Classifier 是继承自曾祖父 SingleClassifierEnhancer 的，DecisionStump 就是只有一个结点的决策树，stump 是树桩的意思，蛮形象的。

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

    super.buildClassifier(data);

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

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

    // only class? -> build ZeroR model
    if (data.numAttributes() == 1) {
        System.err
            .println("Cannot build model (only class attribute
                        present in data!), "
                    + "using ZeroR model instead!");
        m_ZeroR = new weka.classifiers.rules.ZeroR();
        m_ZeroR.buildClassifier(data);
        return;
    } else {
        m_ZeroR = null;
    }

    m_NumClasses = data.numClasses();
    if ((!m_UseResampling)
            && (m_Classifier instanceof WeightedInstancesHandler)) {
        buildClassifierWithWeights(data);
    } else {
        buildClassifierUsingResampling(data);
    }
}
```

这里 buildClassifier 就是复制 m_NumIterations 个分类器，具体的代码如下：

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

    if (m_Classifier == null) {
        throw new Exception("A base classifier has not been specified!");
    }
    m_Classifiers = Classifier.makeCopies(m_Classifier,
        m_NumIterations);
}
```

如果就只有 1 个属性，那么表示只有类别属性，用 m_ZeroR 来分类就可以了，ZeroR 分类是返回类别数最多的那个类别。再向下，判断是否使用重新采样，分类器是否可以考虑带权样本。先看一下 buildClassifierWithWeights(删除了前面的部分代码)：

```
protected void buildClassifierWithWeights(Instances data) throws
Exception {
    // Do boostrap iterations
    for (m_NumIterationsPerformed = 0; m_NumIterationsPerformed <
        m_Classifiers.length; m_NumIterationsPerformed++) {
        // Select instances to train the classifier on
        if (m_WeightThreshold < 100) {
            trainData = selectWeightQuantile(training,
                    (double) m_WeightThreshold / 100);
        } else {
            trainData = new Instances(training, 0, numInstances);
        }

        // Build the classifier
        if (m_Classifiers[m_NumIterationsPerformed] instanceof
            Randomizable)
            ((Randomizable) m_Classifiers[m_NumIterationsPerformed])
                    .setSeed(randomInstance.nextInt());
        m_Classifiers[m_NumIterationsPerformed].
            buildClassifier(trainData);

        // Evaluate the classifier
        evaluation = new Evaluation(data);

    evaluation.evaluateModel(m_Classifiers[m_NumIterationsPerformed],
                training);
        epsilon = evaluation.errorRate();

        // Stop if error too small or error too big and ignore this model
        if (Utils.grOrEq(epsilon, 0.5) || Utils.eq(epsilon, 0)) {
            if (m_NumIterationsPerformed == 0) {
                m_NumIterationsPerformed = 1; // If we're the first we have
```

```
                                            // to to use it
            }
            break;
        }
        // Determine the weight to assign to this model
        m_Betas[m_NumIterationsPerformed] = Math.log((1 - epsilon)
                / epsilon);
        reweight = (1 - epsilon) / epsilon;

        // Update instance weights
        setWeights(training, reweight);
    }
}
```

直接看到 for 循环那里，要循环 m_Classifiers.length，也就是 m_NumIterations 次。如果 m_WeightThreshold 不小于 100，就取全部数据，如果小于 100，调用 selectWeightQuantile：

```
protected Instances selectWeightQuantile(Instances data, double quantile)
{

    int numInstances = data.numInstances();
    Instances trainData = new Instances(data, numInstances);
    double[] weights = new double[numInstances];

    double sumOfWeights = 0;
    for (int i = 0; i < numInstances; i++) {
        weights[i] = data.instance(i).weight();
        sumOfWeights += weights[i];
    }
    double weightMassToSelect = sumOfWeights * quantile;
    int[] sortedIndices = Utils.sort(weights);

    // Select the instances
    sumOfWeights = 0;
    for (int i = numInstances - 1; i >= 0; i--) {
        Instance instance = (Instance) data.instance(sortedIndices[i])
                .copy();
        trainData.add(instance);
        sumOfWeights += weights[sortedIndices[i]];
        if ((sumOfWeights > weightMassToSelect)
                && (i > 0)
                && (weights[sortedIndices[i]] !=
                    weights[sortedIndices[i - 1]])) {
            break;
        }
    }
```

```
    return trainData;
}
```

将每个样本的权重放到 weights 数组中，sumOfWeights 是样本总权重。而相应要选择的权重之和为 weightMassToSelect，sortedIndices 是以权重升序排序的，然后从权重高的样本开始选，然后判断是否达要了要选择的总权重，这里还要注意要把最后一批权重相等的全部选中。

回到 buildClassifierWithWeights 里如果是 Randomizable 的分类器就给它一个随机种子，然后用训练数据学习一个分类器 m_Classifiers[m_NumIterationsPerformed]。再得到它的错误率，如果错误率已经大于 0.5，说明可能比猜还要差，而等于 0 表示已经完美了，不需要再迭代了，而如果只迭代了 0 次，我们只能选择它，还是要设为 1 的。

这里 ln( (1-epsilon) / epsilon )有的地方写的是(1/2) * ln( (1-epsilon) / epsilon )，比如 wiki，但两者实质上是没有区别的，我后面会讲是怎么回事。下面 rewright 是(1-epsilon) / epsilon，在论文和 wiki 里可能是 exp(ln( (1-epsilon) / epsilon ) ) ，两者是相等的，最后调用的函数是 setWeight：

```
protected void setWeights(Instances training, double reweight)
        throws Exception {

    double oldSumOfWeights, newSumOfWeights;

    oldSumOfWeights = training.sumOfWeights();
    Enumeration enu = training.enumerateInstances();
    while (enu.hasMoreElements()) {
        Instance instance = (Instance) enu.nextElement();
        if (!Utils.eq(m_Classifiers[m_NumIterationsPerformed]
                .classifyInstance(instance), instance.classValue()))
            instance.setWeight(instance.weight() * reweight);
    }

    // Renormalize weights
    newSumOfWeights = training.sumOfWeights();
    enu = training.enumerateInstances();
    while (enu.hasMoreElements()) {
        Instance instance = (Instance) enu.nextElement();
        instance.setWeight(instance.weight() * oldSumOfWeights
                / newSumOfWeights);
    }
}
```

这里先得到更新权重前的总权重，再将分类错误的样本赋以新的权重，然后将其归一化。

```
public double[] distributionForInstance(Instance instance) throws
Exception {

    // default model?
    if (m_ZeroR != null) {
```

```
        return m_ZeroR.distributionForInstance(instance);
    }

    if (m_NumIterationsPerformed == 0) {
        throw new Exception("No model built");
    }
    double[] sums = new double[instance.numClasses()];

    if (m_NumIterationsPerformed == 1) {
        return m_Classifiers[0].distributionForInstance(instance);
    } else {
        for (int i = 0; i < m_NumIterationsPerformed; i++) {
            sums[(int) m_Classifiers[i].classifyInstance(instance)]
                  += m_Betas[i];
        }
        return Utils.logs2probs(sums);
    }
}
```

如果是 m_ZeroR，没什么说好的，返回最多的类别，下面是就 m_NumIterationsPerformed 个分类器分类的结果结合起来，每个分类器决定的类别相应的 sums 元素加 m_Betas[i]。然后再转换为概率。Logs2probs 的注释写到 convert an array containing the natural logarithms of probabilities stored in a vector back into probabilities. The probabilities are assumed to sum to one.

buildClassifierUsingResampling 函数大同小异，只把不同的地方列出来：

```
do {
    sample = trainData.resampleWithWeights(randomInstance, weights);

    // Build and evaluate classifier
    m_Classifiers[m_NumIterationsPerformed].buildClassifier(sample);
    evaluation = new Evaluation(data);
    evaluation.evaluateModel(
           m_Classifiers[m_NumIterationsPerformed], training);
    epsilon = evaluation.errorRate();
    resamplingIterations++;
} while (Utils.eq(epsilon, 0)
       && (resamplingIterations < MAX_NUM_RESAMPLING_ITERATIONS));
```

这里是如果 resample 可以让错误率为 0，就停止或是达到最大重新采样的迭代次数。这里的 resampleWithWeights 的注释是：Creates a new dataset of the same size using random sampling with replacement according to the given weight vector. The weights of the instances in the new dataset are set to one. The length of the weight vector has to be the same as the number of instances in the dataset, and all weights have to be positive. 根据给定的权重向量放回抽样，在新的数据集中权重被置为 1，权重向量的长度是与样本的个数是相同的，所有的权重都是正的。