Weka[20] IB1 源代码分析

作者：Koala++/屈伟

首先先解释一下算法名字，很多人很奇怪为什么叫 IB1,IBK，IB Instance-Based 的缩写，但按 Jiawei Han 书上所写，KNN 其实是 Instance-based learning(也被称为 Lazing learning)中一种，他还讲解了基于案例的推理(Case-based reasoning)。算法其实是 KNN，但是作者论文的名字是 Instance-based Learning Algorithms。

我先介绍一下 IB1，IB1 就是只找一个邻居。我们还是先看 buildClassifier。

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

    if (instances.classAttribute().isNumeric()) {
        throw new Exception("IB1: Class is numeric!");
    }
    if (instances.checkForStringAttributes()) {
        throw new UnsupportedAttributeTypeException(
               "IB1: Cannot handle string attributes!");
    }
    // Throw away training instances with missing class
    m_Train = new Instances(instances, 0, instances.numInstances());
    m_Train.deleteWithMissingClass();

    m_MinArray = new double[m_Train.numAttributes()];
    m_MaxArray = new double[m_Train.numAttributes()];
    for (int i = 0; i < m_Train.numAttributes(); i++) {
        m_MinArray[i] = m_MaxArray[i] = Double.NaN;
    }

    Enumeration enu = m_Train.enumerateInstances();
    while (enu.hasMoreElements()) {
        updateMinMax((Instance) enu.nextElement());
    }
}
```

是的，KNN 也有 buildClassifier，听起来蛮奇怪的。第二个 if，IB1 不能对字符串属性进行学习，因为这种属性不好定义距离，比如 a 和 ab 是 0.5 还是 1 呢？然类别缺失的样本抛弃。m_MinArray 和 m_MaxArray 分别保存每一个属性的最小值和最大值。最下面是对样本进行循环，找出最大值，最小值，updataMinMax 代码如下：

```
private void updateMinMax(Instance instance) {

    for (int j = 0; j < m_Train.numAttributes(); j++) {
        if ((m_Train.attribute(j).isNumeric()) &&
            (!instance.isMissing(j))) {
            if (Double.isNaN(m_MinArray[j])) {
                m_MinArray[j] = instance.value(j);
                m_MaxArray[j] = instance.value(j);
            } else {
                if (instance.value(j) < m_MinArray[j]) {
                    m_MinArray[j] = instance.value(j);
                } else {
                    if (instance.value(j) > m_MaxArray[j]) {
                        m_MaxArray[j] = instance.value(j);
                    }
                }
```

```
            }
        }
    }
}
```

Double.isNaN(m_MinArray[j])判断是不是 m_MinArray 和 m_MaxArray 已经赋值过了，else，如果可以更新 min 和 max 更新。

再看一下 classifyInstance 函数：

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

    if (m_Train.numInstances() == 0) {
        throw new Exception("No training instances!");
    }

    double distance, minDistance = Double.MAX_VALUE, classValue = 0;
    updateMinMax(instance);
    Enumeration enu = m_Train.enumerateInstances();
    while (enu.hasMoreElements()) {
        Instance trainInstance = (Instance) enu.nextElement();
        if (!trainInstance.classIsMissing()) {
            distance = distance(instance, trainInstance);
            if (distance < minDistance) {
                minDistance = distance;
                classValue = trainInstance.classValue();
            }
        }
    }

    return classValue;
}
```

因为要进化归范化，所以对这个分类的样本再次调用 updateMinMax。然后对训练样本进行循环，用 distance 计算与每一个样本的距离，如果比前面的距离小，则记录，最后返回与测试样本距离最小的样本的类别值。

```
private double distance(Instance first, Instance second) {
    double diff, distance = 0;

    for (int i = 0; i < m_Train.numAttributes(); i++) {
        if (i == m_Train.classIndex()) {
            continue;
        }
        if (m_Train.attribute(i).isNominal()) {
                // If attribute is nominal
            if (first.isMissing(i) || second.isMissing(i)
                    || ((int) first.value(i) != (int) second.value(i))) {
                distance += 1;
            }
        } else {
            // If attribute is numeric
            if (first.isMissing(i) || second.isMissing(i)) {
                if (first.isMissing(i) && second.isMissing(i)) {
                    diff = 1;
                } else {
                    if (second.isMissing(i)) {
                        diff = norm(first.value(i), i);
                    } else {
                        diff = norm(second.value(i), i);
                    }
```

```
                    if (diff < 0.5) {
                        diff = 1.0 - diff;
                    }
                }
            } else {
                diff = norm(first.value(i), i) - norm(second.value(i), i);
            }
            distance += diff * diff;
        }
    }

    return distance;
}
```

和 Jiawei Han 书里面说的一样，对离散属性来说，两个样本任一在对应属性上为缺失值，距离为 1，不相等相然还是为 1。对于连续属性，如果两个都是缺失值，距离为 1，其中之一在对应属性上为缺失值，把另一个不为缺失值的属性值规范化，距离为 1-diff，意思就是设到可能的最远(当然那个缺失值比 m_MinArray，m_MaxArray 还小还大，这就不对了)。如果两个都有值，就把距离相加，最后平方。

为了完整性，将 norm 列出来：

```
private double norm(double x, int i) {

    if (Double.isNaN(m MinArray[i])
            || Utils.eq(m MaxArray[i], m MinArray[i])) {
        return 0;
    } else {
        return (x - m MinArray[i]) / (m MaxArray[i] - m MinArray[i]);
    }
}
```