Weka[30] VotedPerceptron 源代码分析

作者：Koala++/屈伟

代码中列出来的参考论文是 Y. Freund and R. E. Schapire (1998). Large margin classification using the perceptron algorithm。我下载的是 preliminary version，一共是 19 页，如果感觉长了，可以看一下 Ng 的 Lecture notes 的 The perception and large margin classifiers。

这篇论文的二个主要贡献之一：giving a bound on the online learning error of the perceptron algorithm, when it is run as an online algorithm that performs an update each time it gets an example wrong. Note that the bound below on the number of errors does not have an explicit dependence on the number of examples m in the sequences, or on the dimension n of the inputs.这个定理的证明我翻译了，可以在 weka 中文站找到，SVM1。它的第二个贡献是使用了核方法，不过这个类没有实现。

直接从 buildClassifier 开始：

```
if (insts.checkForStringAttributes()) {
    throw new UnsupportedAttributeTypeException(
           "Cannot handle string attributes!");
}
if (insts.numClasses() > 2) {
    throw new Exception("Can only handle two-class datasets!");
}
if (insts.classAttribute().isNumeric()) {
    throw new UnsupportedClassTypeException(
           "Can't handle a numeric class!");
}
```

不能处理字符串属性，只能处理二分类问题，不能处理连续类别属性。

```
/** Make space to store perceptrons */
m Additions = new int[m MaxK + 1];
m IsAddition = new boolean[m MaxK + 1];
m Weights = new int[m MaxK + 1];

/** Compute perceptrons */
m K = 0;
out: for (int it = 0; it < m NumIterations; it++) {
    for (int i = 0; i < m Train.numInstances(); i++) {
        Instance inst = m Train.instance(i);
        if (!inst.classIsMissing()) {
            int prediction = makePrediction(m K, inst);
            int classValue = (int) inst.classValue();
            if (prediction == classValue) {
                m Weights[m K]++;
            } else {
                m IsAddition[m K] = (classValue == 1);
                m Additions[m K] = i;
                m K++;
                m Weights[m K]++;
            }
            if (m K == m MaxK) {
                break out;
            }
        }
    }
```

```
}
```

m_IsAddition 是记录论文中的 vi 是原数据集中的哪个样本，m_Additions 表示的是符号，这里 g(h(x))所得到的是{-1,1}而不是{0,1}。用 makePrediction 得到当前样本的预测值，如果和样本真实的类别值相同，不做处理，如果不同，那么记录下这个样本的类别值(也就决定了它的符号)，和这个样本的下标，而与原论文有点不同的是，这里没有 v_k+1=v_k + y_i x，而是直接记录了。最后如果达到了最大迭代值，跳出。

```
private int makePrediction(int k, Instance inst) throws Exception {

    double result = 0;
    for (int i = 0; i < k; i++) {
        if (m IsAddition[i]) {
            result += innerProduct(m Train.instance(m Additions[i]),
                inst);
        } else {
            result -= innerProduct(m Train.instance(m Additions[i]),
                inst);
        }
    }
    if (result < 0) {
        return 0;
    } else {
        return 1;
    }
}
```

这里看起来很怪异，是因为它并没有更新权值，而是记录下了所有的更新，可以把 alpha 视为一个常量，它在所有需要更新的样本（也可以把它视为所有的更新），算出内积。

```
private double innerProduct(Instance i1, Instance i2) throws Exception
{

    // we can do a fast dot product
    double result = 0;
    int n1 = i1.numValues();
    int n2 = i2.numValues();
    int classIndex = m Train.classIndex();
    for (int p1 = 0, p2 = 0; p1 < n1 && p2 < n2;) {
        int ind1 = i1.index(p1);
        int ind2 = i2.index(p2);
        if (ind1 == ind2) {
            if (ind1 != classIndex) {
                result += i1.valueSparse(p1) * i2.valueSparse(p2);
            }
            p1++;
            p2++;
        } else if (ind1 > ind2) {
            p2++;
        } else {
            p1++;
        }
    }
    result += 1.0;

    if (m Exponent != 1) {
        return Math.pow(result, m Exponent);
    } else {
        return result;
    }
```

```
}
```

内积的计算，没什么好看的，只是因为数据可能用的是稀疏表示方法，所以写的有点怪。

```
// Get probabilities
double output = 0, sumSoFar = 0;
if (m_K > 0) {
    for (int i = 0; i <= m_K; i++) {
        if (sumSoFar < 0) {
            output -= m_Weights[i];
        } else {
            output += m_Weights[i];
        }
        if (m_IsAddition[i]) {
            sumSoFar += innerProduct(m_Train.instance(m_Additions[i]),
                inst);
        } else {
            sumSoFar -= innerProduct(m_Train.instance(m_Additions[i]),
                inst);
        }
    }
}
double[] result = new double[2];
result[1] = 1 / (1 + Math.exp(-output));
result[0] = 1 - result[1];
```

这里 m_Weights 的值是由 buildClassifier 中得到的，它的意思就是在当前这个权重下，分对了多少个，当它错误的时候，又有新的权重。而下面的内积计算与 makePrediction 的是一样的。

个人认为这代码写成这样，似乎主要是为了构造出来一个算法来证明它的边界。而它后面将它转换成了批学习，证明了它的边界。