Weka[26] Voting 源代码分析

作者：Koala++/屈伟

Voting 这个类和 Bagging 差不多，不过我感觉 Voting 更好用一点，Bagging 继承关系看起来可能有很强的扩展性，但对于我，这一点没什么意义。Voting 继承自 RandomizableMultipleClassifiersCombiner，而它又继承自 MultipleClassifiersCombiner，它继承自 Classifier。

下面看一下 Voting 的成员变量：

```
/** combination rule: Average of Probabilities */
public static final int AVERAGE_RULE = 1;
/** combination rule: Product of Probabilities (only nominal classes) */
public static final int PRODUCT_RULE = 2;
/** combination rule: Majority Voting (only nominal classes) */
public static final int MAJORITY_VOTING_RULE = 3;
/** combination rule: Minimum Probability */
public static final int MIN_RULE = 4;
/** combination rule: Maximum Probability */
public static final int MAX_RULE = 5;
/** combination rule: Median Probability (only numeric class) */
public static final int MEDIAN_RULE = 6;
/** combination rules */
public static final Tag[] TAGS_RULES = {
       new Tag(AVERAGE_RULE, "AVG", "Average of Probabilities"),
       new Tag(PRODUCT_RULE, "PROD", "Product of Probabilities"),
       new Tag(MAJORITY_VOTING_RULE, "MAJ", "Majority Voting"),
       new Tag(MIN_RULE, "MIN", "Minimum Probability"),
       new Tag(MAX_RULE, "MAX", "Maximum Probability"),
       new Tag(MEDIAN_RULE, "MED", "Median") };

/** Combination Rule variable */
protected int m_CombinationRule = AVERAGE_RULE;
```

比起 Bagging 是丰富了不少，另外，源代码上列出来参考文献我没找到，事实上也没必要，直接看代码就可以了。

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

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

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

    m_Random = new Random(getSeed());

    for (int i = 0; i < m_Classifiers.length; i++) {
        getClassifier(i).buildClassifier(newData);
    }
}
```

很简单，就是训练多个分类器。

下面是 distributionForInstance 的代码：

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

```
    double[] result = new double[instance.numClasses()];

    switch (m_CombinationRule) {
    case AVERAGE_RULE:
        result = distributionForInstanceAverage(instance);
        break;
    case PRODUCT_RULE:
        result = distributionForInstanceProduct(instance);
        break;
    case MAJORITY_VOTING_RULE:
        result = distributionForInstanceMajorityVoting(instance);
        break;
    case MIN_RULE:
        result = distributionForInstanceMin(instance);
        break;
    case MAX_RULE:
        result = distributionForInstanceMax(instance);
        break;
    case MEDIAN_RULE:
        result[0] = classifyInstance(instance);
        break;
    default:
        throw new IllegalStateException("Unknown combination rule '"
               + m_CombinationRule + "'!");
    }

    if (!instance.classAttribute().isNumeric()
      && (Utils.sum(result) > 0))
        Utils.normalize(result);

    return result;
}
```

从第一个开始看，distributionForInstanceAverage：

```
protected double[] distributionForInstanceAverage(Instance instance)
        throws Exception {
    double[] probs = getClassifier(0).
        distributionForInstance(instance);
    for (int i = 1; i < m_Classifiers.length; i++) {
        double[] dist = getClassifier(i).
            distributionForInstance(instance);
        for (int j = 0; j < dist.length; j++) {
            probs[j] += dist[j];
        }
    }
    for (int j = 0; j < probs.length; j++) {
        probs[j] /= (double) m_Classifiers.length;
    }
    return probs;
}
```

它将所有分类器算出来的分布累加后平均。下面是 distributionForInstanceProduct：

```
protected double[] distributionForInstanceProduct(Instance instance)
throws Exception {
    double[] probs = getClassifier(0).
        distributionForInstance(instance);
    for (int i = 1; i < m_Classifiers.length; i++) {
        double[] dist = getClassifier(i).
            distributionForInstance(instance);
```

```
        for (int j = 0; j < dist.length; j++) {
            probs[j] *= dist[j];
        }
    }

    return probs;
}
```

和上面的 Average 区别不大，只是将加换为了乘，没平均(当然也不可能去平均了)。接下来 distributionForInstanceMajorityVoting。

```
protected double[] distributionForInstanceMajorityVoting(Instance
instance) throws Exception {

    double[] probs = new double[instance.classAttribute().numValues()];
    double[] votes = new double[probs.length];

    for (int i = 0; i < m Classifiers.length; i++) {
        probs = getClassifier(i).distributionForInstance(instance);
        int maxIndex = 0;
        for (int j = 0; j < probs.length; j++) {
            if (probs[j] > probs[maxIndex])
                maxIndex = j;
        }

        // Consider the cases when multiple classes happen to have the same
        // probability
        for (int j = 0; j < probs.length; j++) {
            if (probs[j] == probs[maxIndex])
                votes[j]++;
        }
    }

    int tmpMajorityIndex = 0;
    for (int k = 1; k < votes.length; k++) {
        if (votes[k] > votes[tmpMajorityIndex])
            tmpMajorityIndex = k;
    }

    // Consider the cases when multiple classes receive the same amount
    // of votes
    Vector<Integer> majorityIndexes = new Vector<Integer>();
    for (int k = 0; k < votes.length; k++) {
        if (votes[k] == votes[tmpMajorityIndex])
            majorityIndexes.add(k);
    }
    // Resolve the ties according to a uniform random distribution
    int majorityIndex = majorityIndexes.get(m Random
            .nextInt(majorityIndexes.size()));

    // set probs to 0
    for (int k = 0; k < probs.length; k++)
        probs[k] = 0;
    probs[majorityIndex] = 1; // the class that have been voted the most
                              // receives 1

    return probs;
}
```

在第一个 for 循环的时候就用 votes 数组记录下投票的结果，在下面一个 for 循环中，用

tmpMajorityIndex 来记录投票得票最多的票数是多少，为了解决在多个类别值所得票数一样这种情况，先用 marjorityIndexes 来记录得票最多的几个类别值索引号，再随机选择其中之一，将他设为 1，其余的都设为 0，这里用随机是必要的，不然类别 1 和类别 2 总是相同，如果不随机，将总是选择类别 1。

```
protected double[] distributionForInstanceMax(Instance instance)
        throws Exception {

    double[] max = getClassifier(0).distributionForInstance(instance);
    for (int i = 1; i < m_Classifiers.length; i++) {
        double[] dist = getClassifier(i).
            distributionForInstance(instance);
        for (int j = 0; j < dist.length; j++) {
            if (max[j] < dist[j])
                max[j] = dist[j];
        }
    }

    return max;
}
protected double[] distributionForInstanceMin(Instance instance)
        throws Exception {

    double[] min = getClassifier(0).distributionForInstance(instance);
    for (int i = 1; i < m_Classifiers.length; i++) {
        double[] dist = getClassifier(i).
            distributionForInstance(instance);
        for (int j = 0; j < dist.length; j++) {
            if (dist[j] < min[j])
                min[j] = dist[j];
        }
    }

    return min;
}
```

Max 和 Min 一起看，max 返回的值是每个类别值可能的最大值，min 刚好相反。

再接下来，classifyInstance：

```
public double classifyInstance(Instance instance) throws Exception {
    double result;
    double[] dist;
    int index;

    switch (m_CombinationRule) {
    case AVERAGE_RULE:
    case PRODUCT_RULE:
    case MAJORITY_VOTING_RULE:
    case MIN_RULE:
    case MAX_RULE:
        dist = distributionForInstance(instance);
        if (instance.classAttribute().isNominal()) {
            index = Utils.maxIndex(dist);
            if (dist[index] == 0)
                result = Instance.missingValue();
            else
                result = index;
        } else if (instance.classAttribute().isNumeric()) {
            result = dist[0];
```

```
        } else {
            result = Instance.missingValue();
        }
        break;
    case MEDIAN_RULE:
        result = classifyInstanceMedian(instance);
        break;
    default:
        throw new IllegalStateException("Unknown combination rule '"
                + m_CombinationRule + "'!");
    }
    return result;
}
```

除了 MEDIAN_RULE，上面的方法都一样，如果是离散型的，就赋以最可能的类别值。如果是连续型的，那就是 dist[0]了，下面看一下 classifyInstanceMedian：

```
protected double classifyInstanceMedian(Instance instance)
throws Exception {
    double[] results = new double[m_Classifiers.length];
    double result;

    for (int i = 0; i < results.length; i++)
        results[i] = m_Classifiers[i].classifyInstance(instance);

    if (results.length == 0)
        result = 0;
    else if (results.length == 1)
        result = results[0];
    else
        result = Utils.kthSmallestValue(results, results.length / 2);

    return result;
}
```

用中位数的方法。

但是无论用哪一种，都不能说它比另一种要好，你也可以对这个类进行扩展，我自己用加权投票的算法的时候，自己写了几次，看王义的代码，发现他继承这个类在写，才发现它的。