Weka[31] crossValidation 源代码分析

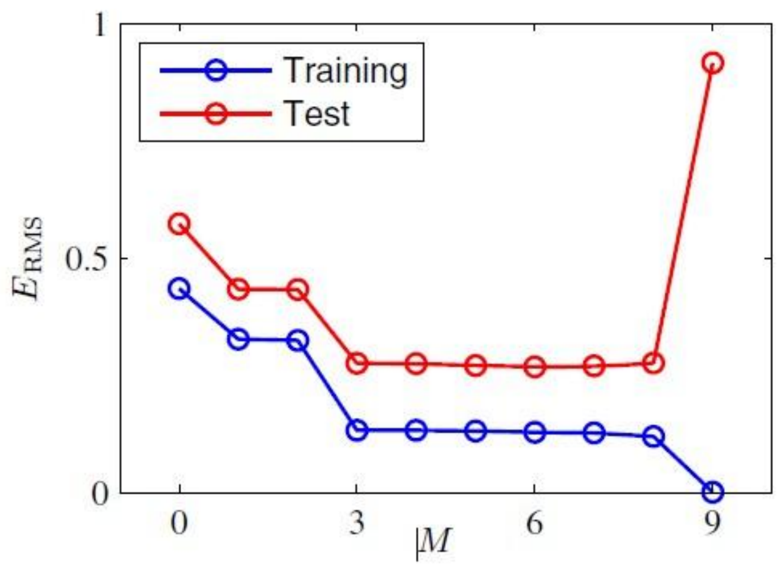
作者：Koala++/屈伟

Weka 学习论坛里的人也帮我下载过一些论文，而群主也希望我建设群论坛，而我一开始看到的就是 cross validation，感觉介绍的有点简单了，这里我只是再深一点而已。我还是 Ng 的介绍为主。

在看到学习算法的时候，我们都是以最小化经验误差为目标，比如有方法：梯度下降，牛顿法，lagrange 乘子法，坐标上升法，都是我 blog 里提到过的方法了。如果我们用下面的步骤来得到模型：

1. 在数据集上训练多个模型。
2. 选择训练误差最小的模型。

下面是说明这个问题的一幅图（它的原意倒和这没什么关系），直接去看 Pattern recognition and machine learning 第一章。



这幅图大概意思就是模型复杂到一定程序（可以表示学习到概念之后），再使用更复杂模型，那么它的训练误差会下降，而测试误差会提高。这幅图其实还有一个比较深的意义，那就是你可以通过它来选择合适的模型，不要一看测试误差高，就加样本。

然后有一个代替它的方法是 hold-out cross validation 或是被称为 simple cross validation。

1. 把数据集分为两部分，一部分为训练集（比如 70%的数据），和测试数据集（比如 30%）。测试数据集也被称为 hold-out cross validation set。
2. 在训练集上训练多个模型。
3. 每一个模型在测试数据集上得到一个分类误差值，选择分类误差最小的模型。

通常测试集的大小在数据集的 1/4 到 1/3 之间。一般 30％是一个 t 典型的选择。这样做的原因是：如果只是以最小化经验误差为目标，那么最终选择的就是过拟合的模型。

但用这种方法也有一个缺点就是它浪费了 30％的数据，就算我们最后选择出我们认为合理的模型，再用全部的数据进行训练，只不能保证这个模型是最好的。如果训练样本充分，那倒没什么，如果训练样本不足，而模型与模型训练所需的样本也是不一样的(不太清楚如何表述，可以看一下 learning theory，就是求的最小需要样本)。再说明白一点，在样本不充

足的情况下，或不知道是不是充足的情况下，模型 A 在 70%的数据上比模型 B 好，不能说在 100%的数据集上，模型 A 一定比模型 B 好。

接下来的方法是 k-fold cross validation，这个算法就不写伪代码了，马上看真代码，一个典型的选择是 k=10，它与上面的方法相比，它只留下了 1/k 的数据，但是我们也需要训练 k 次，比以前多了 k 次(其实也不完全是这样，就算是划分训练测试集，也不可能只测一次)。

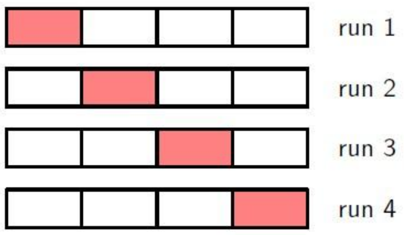


Figure 1.18 The technique of S-fold cross-validation, illustrated here for the case of $S = 4$, involves taking the available data and partitioning it into S groups (in the simplest case these are of equal size). Then $S - 1$ of the groups are used to train a set of models that are then evaluated on the remaining group. This procedure is then repeated for all S possible choices for the held-out group, indicated here by the red blocks, and the performance scores from the S runs are then averaged.

还有就是 Pattern recognition and machine learning 提到的：A further problem with techniques such as cross-validation that use separate data to assess performance is that we might have multiple complexity parameters for a single model (for instance, there might be several regularization parameters). Exploring combinations of setting for such parameters could, in the worst case, require a number of training runs that is exponential in the number of parameters.

在数据实在太少的情况下，就要用一种特殊的 cross validation 方法了，是 leave-one-out cross validation，就是每次训练只留下一个样本不训练，用它来测试。

在听 Ng 的课程时，他的学生提了一个很有意思的问题，在 Learning Theory（这个还是比较重要的，被 Ng 视为是否入门的标准之一）中，不是可以从样本数求出来一个模型学习所需要的样本数（在一定条件下），为什么还要用 model selection 这一类方法呢？

下面翻译一下(大意)Ng 的回答：

It turns out that when you're proving learning theory bounds, very often the bounds will be extremely loose because you're sort of proving the worse case upper bound that holds true even for very bad – what is it – so the bounds that I proved just now; right? That holds true for absolutely any probability distribution over training examples; right? So just assume the training examples we've drawn, iid from some distribution script d, and the bounds I proved hold true for absolutely any probability distribution over script d. And chances are whatever real life distribution you get over, you know, houses and their prices or whatever, is probably not as bad as the very worse one you could've gotten; okay? And so it turns out that if you actually plug in the constants of learning theory bounds, you often get extremely large numbers.

你在证明学习理论的边界时，通常边界都是异常 loose，因为你在证的都是比较糟糕的上界，也就是在很坏的时候都成立的边界，这些就是训练样本无论服从何从概率分布都成立。现实中的数据，也许不会和最坏的情况一样。当你将常量加入学习理论的边界时（像算法时间空间分析时，忽略所有常量），你通常都会得到一个非常大的值。

Take logistic regression – logistic regression you have ten parameters and 0.01 error, and with 95 percent probability. How many training examples do I need? If you actually plug in actual constants into the text for learning theory bounds, you often get extremely pessimistic estimates with the number of examples you need. You end up with some ridiculously large numbers. You would need 10,000 training examples to fit ten parameters. So a good way to think of these learning theory bounds is – and this is why, also, when I write papers on learning theory bounds, I quite often use big-O notation to just absolutely just ignore the constant factors because the bounds seem to be very loose.

以 logistic regression 为例－你有 10 个参数和在 95％的概率下错误率小于 0.01。我需要多少样本，如果你将常量代入边界，你会得到一个非常悲观的估计，你会得到一个大的可笑的值，比如 10,000 个训练样本来拟合 10 个参数。所以一个好的方式来理解这些学习边界是，忽略常量，因为边界非常 loose。

There are some attempts to use these bounds to give guidelines as to what model to choose, and so on. But I personally tend to use the bounds – again, intuition about – for example, what are the number of training examples you need grows linearly in the number of parameters or what are your grows x dimension in number of parameters; whether it goes quadratic – parameters? So it's quite often the shape of the bounds. The fact that the number of training examples – the fact that some complexity is linear in the VC dimension, that's sort of a useful intuition you can get from these theories. But the actual magnitude of the bound will tend to be much looser than will hold true for a particular problem you are working on.

有一些关于用这些边界来指导选择哪一种模型的方法，但我个人倾向于用这些边界——再强调一下，直观的——比如，你所需要的样本数是与参数的个数是线性关系，还是二次关系。所以通常都是这些关系给你了一个直观的认识。而它得到出的边界比真实的一些你正在处理的特殊问题要 loose 的多。

代码在 classifiers.Evaluation 类中：

```java
/**
 * Performs a (stratified if class is nominal) cross-validation
 * for a classifier on a set of instances. Now performs
 * a deep copy of the classifier before each call to
 * buildClassifier() (just in case the classifier is not
 * initialized properly).
 **/
public void crossValidateModel(Classifier classifier, Instances data,
       int numFolds, Random random) throws Exception {

    // Make a copy of the data we can reorder
    data = new Instances(data);
    data.randomize(random);
    if (data.classAttribute().isNominal()) {
        data.stratify(numFolds);
    }
    // Do the folds
    for (int i = 0; i < numFolds; i++) {
        Instances train = data.trainCV(numFolds, i, random);
        setPriors(train);
        Classifier copiedClassifier = Classifier.makeCopy(classifier);
        copiedClassifier.buildClassifier(train);
        Instances test = data.testCV(numFolds, i);
        evaluateModel(copiedClassifier, test);
    }
    m NumFolds = numFolds;
}
```

Randomize 很简单：

```java
public void randomize(Random random) {

    for (int j = numInstances() - 1; j > 0; j--)
        swap(j, random.nextInt(j + 1));
}
```

Randomize 注意它是从后向前打乱的，这样写的确简单点。

```
public void stratify(int numFolds) {

    if (numFolds <= 0) {
        throw new IllegalArgumentException(
                "Number of folds must be greater than 1");
    }
    if (m_ClassIndex < 0) {
        throw new UnassignedClassException(
                "Class index is negative (not set)!");
    }
    if (classAttribute().isNominal()) {

        // sort by class
        int index = 1;
        while (index < numInstances()) {
            Instance instance1 = instance(index - 1);
            for (int j = index; j < numInstances(); j++) {
                Instance instance2 = instance(j);
                if ((instance1.classValue() == instance2.classValue())
                        || (instance1.classIsMissing() && instance2
                                .classIsMissing())) {
                    swap(index, j);
                    index++;
                }
            }
            index++;
        }
        stratStep(numFolds);
    }
}
```

如果类别是离散值，两层循环的作用是把类别相同的样本连到一起，stratify 的本意是分层，也就是在循环完之后样本的类别是 class0,class0,…,class0,class1,…class1,…,classN,…, classN。stratStep 的代码如下：

```
protected void stratStep(int numFolds) {

    FastVector newVec = new FastVector(m_Instances.capacity());
    int start = 0, j;

    // create stratified batch
    while (newVec.size() < numInstances()) {
        j = start;
        while (j < numInstances()) {
            newVec.addElement(instance(j));
            j = j + numFolds;
        }
        start++;
    }
    m_Instances = newVec;
}
```

第 numFolds 选择一个样本，这样的目的是可以让类别尽可能分开一点，不要一个训练集或是测试集（当然更可能是它）类别很不平均。

在 crossValidationModel 的代码中，用 trainCV 函数得到训练集，学习一个分类器，用 testCV 得到测试集，分类器在 test 测试集上测试，evaluateModel：

```
public double[] evaluateModel(Classifier classifier, Instances data)
        throws Exception {
```

```
    double predictions[] = new double[data.numInstances()];

    for (int i = 0; i < data.numInstances(); i++) {
        predictions[i] = evaluateModelOnce((Classifier) classifier, data
                .instance(i));
    }
    return predictions;
}
```

在这里基本上就可以认为看完了，而 evaluateModelOnce 调用的函数 updateStatsForClassifier 可以在需要的时候再看吧，与 crossValidation 已经有点距离了。