".632 Bootstrap" vs. "k-fold cross validation"

Projekt no. 6

Andrea Szabóová, szaboand@fel.cvut.cz

ZS 2010/2011

Inštrukce

1. Vygenerujte trénovací data (zvolte si: spojitá/diskrétní/booleovská) a vyberte si vhodný klasifikátor.
2. Implementujte odhad chyby pomocí metod "k-fold cross validation" a ".632 Bootstrap" (Hastie et al: "The Elements of Statistical Learning" nebo `http://www-group.slac.stanford.edu/sluo/lectures/stat_lecture_files/sluo2006lec7.pdf`).
3. Odhadněte chybu pomocí obou metod pro různé velikosti datasetu a porovnejte s odhadem chyby na dostatečně velké nezávislé testovací množině.

Cross-validation

Consider what to do when the amount of data for training and testing is limited. The holdout method reserves a certain amount for testing and uses the remainder for training (and sets part of that aside for validation, if required). In practical terms, it is common to hold out one-third of the data for testing and use the remaining two-thirds for training.

Of course, you may be unlucky: the sample used for training (or testing) might not be representative. In general, you cannot tell whether a sample is representative or not. But there is one simple check that might be worthwhile: each class in the full dataset should be represented in about the right proportion in the training and testing sets. If, by bad luck, all examples with a certain class were missing from the training set, you could hardly expect a classifier learned from that data to perform well on the examples of that class — and the situation would be exacerbated by the fact that the class would necessarily be overrepresented in the test set because none of its instances made it into the training set! Instead, you should ensure that the random sampling is done in such a way as to guarantee that each class is properly represented in both training and test sets. This procedure is called *stratification*, and we might speak of *stratified holdout*. Although it is generally well worth doing, stratification provides only a primitive safeguard against uneven representation in training and test sets.

A more general way to mitigate any bias caused by the particular sample chosen for holdout is to repeat the whole process, training and testing, several times with different random samples. In each iteration a certain proportion — say two-thirds — of the data is randomly selected for training, possibly with stratification, and the remainder used for testing. The error rates on the different iterations are averaged to yield an overall error rate. This is the *repeated holdout* method of error rate estimation.

In a single holdout procedure, you might consider swapping the roles of the testing and training data — that is, train the system on the test data and test it on the training data — and average the two results, thus reducing the effect of uneven representation in training and test sets. Unfortunately, this is only really plausible with a $50:50$ split between training and test data, which is generally not ideal — it is better to use more than half the data for training even at the expense of test data. However, a simple variant forms the basis of an important statistical technique called $cross-validation$. In cross-validation, you decide on a fixed number of *folds*, or partitions of the data. Suppose we use three. Then the data is split into three approximately equal partitions and each in turn is used for testing and the remainder is used for training. That is, use two-thirds for training and one-third for testing and repeat the procedure three times so that, in the end, every instance has been used exactly once for testing. This is called $threefold\ cross-validation$, and if stratification is adopted as well — which it often is — it is $stratified\ threefold\ cross-validation$.

The standard way of predicting the error rate of a learning technique given a single, fixed sample of data is to use stratified 10-fold cross-validation. The data is divided randomly into 10 parts in which the class is represented in approximately the same proportions as in the full dataset. Each part is held out in turn and the learning scheme trained on the remaining nine-tenths; then its error rate is calculated on the holdout set. Thus the learning procedure is executed a total of 10 times on different training sets (each of which have a lot in common). Finally, the 10 error estimates are averaged to yield an overall error estimate.

Why 10? Extensive tests on numerous datasets, with different learning techniques, have shown that 10 is about the right number of folds to get the best estimate of error, and there is also some theoretical evidence that backs this up. Although these arguments are by no means conclusive, and debate continues to rage in machine learning and data mining circles about what is the best scheme for evaluation, 10-fold cross-validation has become the standard method in practical terms. Tests have also shown that the use of stratification improves results slightly. Thus the standard evaluation technique in situations where only limited data is available is stratified 10-fold cross-validation. Note that neither the stratification nor the division into 10 folds has to be exact: it is enough to divide the data into 10 approximately equal sets in which the various class values are represented in approximately the right proportion. Statistical evaluation is not an exact science. Moreover, there is nothing magic about the exact number 10: 5-fold or 20-fold cross-validation is likely to be almost as good.

A single 10-fold cross-validation might not be enough to get a reliable error estimate. Different 10-fold cross-validation experiments with the same learning method and dataset often produce different results, because of the effect of random variation in choosing the folds themselves. Stratification reduces the variation, but it certainly does not eliminate it entirely.When seeking an accurate error estimate, it is standard procedure to repeat the cross-validation process 10 times — that is, 10 times 10-fold cross-validation — and average the results. This involves invoking the learning algorithm 100 times on datasets that are all nine-tenths the size of the original. Obtaining a good measure of performance is a computation-intensive undertaking.

.632 Bootstrap

The bootstrap method is based on the statistical procedure of sampling *with replacement*. Previously, whenever a sample was taken from the dataset to form a training or test set, it was drawn without replacement. That is, the same instance, once selected, could not be selected again. It is like picking teams for football: you cannot choose the same person twice. But dataset instances are not like people. Most learning methods can use the same instance twice, and it makes a difference in the result of learning if it is present in the training set twice. (Mathematical sticklers will notice that we should not really be talking about "sets" at all if the same object can appear more than once.)

The idea of the bootstrap is to sample the dataset with replacement to form a training set. We will describe a particular variant, mysteriously (but for a reason that will soon become apparent) called the *0.632 bootstrap*. For this, a dataset of n instances is sampled n times, with replacement, to give another dataset of n instances. Because some elements in this second dataset will (almost certainly) be repeated, there must be some instances in the original dataset that have not been picked: we will use these as test instances.

What is the chance that a particular instance will not be picked for the training set? It has a *1/n* probability of being picked each time and therefore a *1 − 1/n* probability of not being picked. Multiply these probabilities together according to the number of picking opportunities, which is n, and the result is a figure of

$\left(1 - \frac{1}{n}\right)^n = e^{-1} = 0.368$

(where e is the base of natural logarithms, 2.7183, not the error rate!). This gives the chance of a particular instance not being picked at all. Thus for a reasonably large dataset, the test set will contain about 36.8% of the instances and the training set will contain about 63.2% of them (now you can see why it's called the *0.632 bootstrap*). Some instances will be repeated in the training set, bringing it up to a total size of n, the same as in the original dataset.

The figure obtained by training a learning system on the training set and calculating its error over the test set will be a pessimistic estimate of the true error rate, because the training set, although its size is n, nevertheless contains only 63% of the instances, which is not a great deal compared, for example, with the 90% used in 10-fold cross-validation. To compensate for this, we combine the test-set error rate with the resubstitution error on the instances in the training set. The resubstitution figure, as we warned earlier, gives a very optimistic estimate of the true error and should certainly not be used as an error figure on its own. But the bootstrap procedure combines it with the test error rate to give a final estimate e as follows:

$e = 0.632 \times e_{test-instances} + 0.368 \times e_{training-instances}$

Then, the whole bootstrap procedure is repeated several times, with different replacement samples for the training set, and the results averaged.

The bootstrap procedure may be the best way of estimating error for very small datasets.However, like leave-one-out cross-validation, it has disadvantages that can be illustrated by considering a special, artificial situation. In fact, the very dataset we considered previously will do: a completely random dataset with two classes. The true error rate is 50% for any prediction rule. But a scheme that memorized the training set would give a perfect resubstitution score of 100% so that $e_{training-instances} = 0$, and the 0.632 bootstrap will mix this in with a weight of 0.368 to give an overall error rate of only 31.6% $(0.632 \times 50\% + 0.368 \times 0\%)$, which is misleadingly optimistic.