

Evaluating Classifiers

Charles Elkan
elkan@cs.ucsd.edu

October 17, 2007

In a real-world application of supervised learning, we have a training set of examples with labels, and a test set of examples with unknown labels. The whole point is to make predictions for the test examples.

However, in research or experimentation we want to measure the performance achieved by a learning algorithm. To do this we use a test set consisting of examples with known labels. We train the classifier on the training set, apply it to the test set, and then measure performance by comparing the predicted labels with the true labels (which were not available to the training algorithm).

Sometimes we have a training set and a test set given already. Other times, we just have one database of labeled training examples. In this case, we have to divide the database ourselves into separate training and test sets. A common rule of thumb is to use 70% of the database for training and 30% for testing. Dividing the database into training and test sets should be done randomly, in order to guarantee that both sets are random samples from the same distribution.

It is absolutely vital to measure the performance of a classifier on an independent test set. Every training algorithm looks for patterns in the training data, i.e. correlations between the features and the class. Some of the patterns discovered may be spurious, i.e. they are valid in the training data due to randomness in how the training data was selected from the population, but they are not valid, or not as strong, in the whole population. A classifier that relies on these spurious patterns will have higher accuracy on the training examples than it will on the whole population. Only accuracy measured on an independent test set is a fair estimate of accuracy on the whole population. The phenomenon of relying on patterns that are strong only in the training data is called overfitting. In practice it is an omnipresent danger.

Most training algorithms have some settings that the user can choose between. For naive Bayes these algorithm parameters include the degree of smoothing λ and the number of bins to use when discretizing continuous features. It is natural to run the algorithm multiple times and to measure the accuracy of the classifier learned with different settings. A set of labeled examples used in this way to pick settings for an algorithm is called a validation set. If you use a validation set, it is important to have a final test set that is independent of both the training set and the validation set.

When evaluating a classifier, there are different ways of measuring its performance. For supervised learning with two possible classes, all measures of performance are based on four numbers obtained from applying the classifier to the test set. These numbers are called true positives tp , false positives fp , true negatives tn , and false negatives fn . They are counts that are entries in a 2×2 table as follows:

		predicted	
		positive	negative
truth	positive	tp	fn
	negative	fp	tn

A table like the one above is called a confusion matrix. The terminology true positive, etc., is standard, but whether columns correspond to predicted and rows to actual, or vice versa, is not standard.

The entries in a confusion matrix are counts, i.e. integers. The total of the four entries $tp + tn + fp + fn = n$, the number of test examples. Depending on the application, many different summary statistics are computed from these entries. In particular:

- accuracy $a = (tp + tn)/n$,
- precision $p = tp/(tp + fp)$, and
- recall $r = tp/(tp + fn)$.

Assuming that n is known, three of the counts in a confusion matrix can vary independently. Hence, no single number, and no pair of numbers, can characterize completely the performance of a classifier. When writing a report, it is best to give the full confusion matrix explicitly, so that readers can calculate whatever performance measurements they are most interested in.

In many domains one class of examples is much more common than the other class. For example, maybe only 1% of patients actually have a certain rare disease, and nowadays only 10% of email messages are actually not spam. The base rate accuracy is the accuracy obtained by predicting that every example has whatever label is most common in the training set. With 99% of examples in one class, it is trivial to achieve 99% accuracy and it can be very difficult to achieve any higher accuracy.

However, for many applications of supervised learning, a classifier can be very useful even if its overall accuracy is less than the base rate. Consider for example a scenario with 97% negative examples, and the following confusion matrix:

$tp = 15$	$fn = 15$
$fp = 25$	$tn = 945$

This classifier has accuracy $a = 960/1000 = 96\%$ which is less than the base rate 97%. But, it has precision $p = 15/(15 + 25) = 37.5\%$ and recall $r = 15/(15 + 15) = 50\%$. These levels of precision and recall are non-trivial and may be very useful in the application domain.

A common way that a classifier is used is to produce a list of candidate test examples for further investigation. For example, a search engine may produce a fixed number of web pages that a classifier predicts are most likely to be relevant to a query. The confusion matrix above means that if the classifier provides a list of 40 candidates, 37.5% of them are genuinely positive and 50% of genuine positives do appear on the list. In contrast, randomly choosing 40 candidates from 1000 would yield only 3% of positives on average, and 97% of actual positives would be missed.

Usually we have a fixed database of labeled examples available, and we are faced with a dilemma: we would like to use all the examples for training, but we would also like to use many examples as an independent test set. Cross-validation is a procedure for overcoming this dilemma. It is the following algorithm.

Input: Training set S , integer constant k

Procedure:

partition S into k disjoint equal-sized subsets S_1, \dots, S_k
 for $i = 1$ to $i = k$
 let $T = S \setminus S_i$
 run learning algorithm with T as training set
 test the resulting classifier on S_i obtaining tp_i, fp_i, tn_i, fn_i
 compute $tp = \sum_i tp_i, fp = \sum_i fp_i, tn = \sum_i tn_i, fn = \sum_i fn_i$

The output of cross-validation is a confusion matrix based on using each labeled example as a test example exactly once. Whenever an example is used for testing a classifier, it has not been used for training that classifier. Hence, the confusion matrix obtained by cross-validation is a fair indicator of the performance of the learning algorithm on independent test examples.

If n labeled examples are available, the largest possible number of folds is $k = n$. This special case is called leave-one-out cross-validation (LOOCV). However, the time complexity of cross-validation is k times that of running the training algorithm once, so often LOOCV is computationally infeasible. In recent research the most common choice for k is 10.

Note that cross-validation does not produce any single final classifier, and the confusion matrix it provides is not the performance of any specific single classifier. Instead, this matrix is an estimate of the average performance of a classifier learned from a training set of size $(k - 1)n/k$ where n is the size of S . The common procedure is to create a final classifier by training on all of S , and then to use the confusion matrix obtained from cross-validation as an informal estimate of the performance of this classifier. This estimate is likely to be conservative in the sense that the final classifier may have slightly better performance since it is based on a slightly larger training set.

The results of cross-validation can be misleading. For example, if each example is duplicated in the training set and we use a nearest-neighbor classifier, then LOOCV will show a zero error rate. Cross-validation with other values of k will also yield misleadingly low error estimates.