Evaluating Classifiers

Charles Elkan
elkan@cs.ucsd.edu

January 18, 2011

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 subsets. 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 subsets is usually done randomly, in order to guarantee that both subsets are random samples from the same distribution. It can be reasonable to do stratified sampling, which means to ensure that each class is present in the exact same proportion in the training and test subsets.

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 for which the user can choose values. For k nearest neighbor classification, an important setting is the integer k, for example. When training a naive Bayes classifier, the settings include the degree of smoothing λ and the number of bins to use when discretizing continuous features, and possibly more. It is natural to run a training 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.

1 Measures of classification success

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.

2 Classification with a rare class

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.

3 Cross-validation

Often 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 $S_1, \ldots, S_k$
  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 intuitively 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. For a detailed discussion of additional pitfalls to avoid in connection with cross-validation, see [Forman and Scholz, 2010].

4 Systematic choice of algorithm parameters

Consider a supervised learning algorithm with one or more settable parameters. How should we choose values for these? Usually, we define a finite set of alternative values for each parameter. Then, the simplest approach is to run the algorithm with the same training data for each combination of parameter values. We measure performance each time on the same validation set [Hsu et al., 2010].

This simple approach likely overfits the validation set. The parameter settings that give highest accuracy (for any definition of accuracy) on the validation set are likely not the settings that perform best on future test data. To get a fair estimate of future accuracy, we need to test the single classifier with the chosen parameter settings on a completely independent test set.

To combine the approach above with cross-validation, one option is nested cross-validation. A drawback of nested cross-validation is its computational cost. If the inner and outer cross-validation both use ten folds, then the number of times a classifier must be trained is ?.

Trying every combination of parameter settings is called grid search. Finding the best settings is an optimization task. Grid search is the most naive possible optimization method. More efficient optimization algorithms typically use gradients (multidimensional derivatives), but these are typically not useful for selecting parameter settings for two reasons. First, the accuracy achieved by an algorithm is not a continuous function of many settings, in particular discrete settings such as k in kNN. Second, often there are alternative combinations of settings each of which is a local optimum.

For the reasons just explained, grid search is still used almost always in practice. However, better alternatives may exist. In particular, combinatorial search methods that sample alternative sets of settings, and then sample new sets based on recombining the best sets found so far, are promising. So-called genetic algorithms are search methods of this type, but not the only ones. A method called the Nelder-Mead algorithm (see Wikipedia), which is available in Matlab under the name `fminsearch` can be useful for finding algorithm settings much faster than with grid search.

5 Making optimal decisions

In any application of machine learning, there are many metrics of performance that can be measured on test data. Examples include log likelihood, mean squared error, 0/1 accuracy, area under the ROC curve, and more. The ultimate quantity to measure and optimize has to be domain-dependent and application-specific. For many tasks, this ultimate quantity is monetary benefit.

Decision theory provides methods for defining and maximizing expected benefit, where expected benefit is the product of the probability of an outcome and the benefit if that outcome occurs. In general, benefit is revenue minus expense. The assignment description below provides an example of a decision-theoretic analysis.

6 Evaluating probabilistic classifiers

- Definition of base rate.
- Meaning of predicted conditional probabilities.
- Meaning of well-calibrated.
- Overlapping distributions of conditional probabilities.
- Measuring the accuracy of conditional probabilities: mean squared error.
- Overfitting: deteriorating performance on test data combined with improving performance on training data.
- Training to maximize one objective versus measuring another: log likelihood versus mean squared error.

CSE 250B Quiz 5, February 4, 2010

For each statement below, clearly write "True" if it is mostly true, or "False" if it is mostly false. Then write one or two sentences explaining why or how the statement is true or false.

Both statements below are based on the same scenario, which is inspired by the second project assignment. You are training logistic regression classifiers without regularization. The classifier to predict $p(visit = 1|x)$ is trained on n_1 examples, while the classifier to predict $p(buy = 1|x, visit = 1)$ is trained on $n_2 < n_1$ examples. You have a small number $d \ll n_2$ of informative features.

1. [2 points] The classifier to predict $p(visit = 1|x)$ is likely to show worse overfitting than the classifier to predict $p(buy = 1|x, visit = 1)$.

2. [2 points] If you had twice as many informative features, both classifiers would be more likely to overfit.

References

[Forman and Scholz, 2010] Forman, G. and Scholz, M. (2010). Apples to apples in cross-validation studies: Pitfalls in classifier performance measurement. *ACM SIGKDD Explorations*, 12(1):49–57.

[Hsu et al., 2010] Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2010). A practical guide to support vector classification. Available at `http://www.csie.ntu.edu.tw/~cjlin/`.