Evaluating Classifiers

Suppose you have collected a dataset of already-classified instances and you have built a classifier. Maybe your classifier is a probabilistic one using the joint probability distribution; or maybe it's a naïve Bayes classifier; or maybe it uses kNN; or maybe it works in some other way. How will you know how good your classifier is?

1 Accuracy

The simplest form of evaluation is in terms of *classification accuracy*: the proportion of instances whose class the classifier can correctly predict. But how can we find this out?

We take a dataset that contains instances whose class we already know. We ask the classifier to classify each of these instances in turn. Then we compare its prediction with the actual class of the instance. We take the proportion of correct classifications as an estimate of the accuracy of the classifier.

In fact, we can draw up what's known as a *confusion matrix*, e.g.:

		predicted class	
		spam	*ham*
actual	*spam*	580	120
class	*ham*	10	290

In this example, the classifier's accuracy is 0.87: it classified 870 of 1000 emails correctly.

It is tempting to draw up this matrix and calculate the accuracy using the same dataset that you used to build the classifier. But this is not a proper way to evaluate your classifier!

What we're trying to find out is how well the classifier *generalises*, i.e. how well it is likely to perform on *unseen* instances that will get presented to it in future when it is being used for real.

Its accuracy on examples it has already *seen* (the ones from which probabilities were calculated in a naïve Bayes classifier, or the ones stored in the memory of a kNN classifier) is likely to be a hopelessly optimistic estimate of this future performance. The accuracy of a $1NN$ classifier, for example, when tested on the very same examples that are already in its memory, can be 100%! Of course, it is highly unlikely that the $1NN$ classifier, if it were then deployed for real, would classify without error.

We must build the classifier using one dataset, called the *training set*. And, we must evaluate it on a different dataset, called the *test set*, a set of independent instances that played no part in building the classifier. What do we require of these two sets?

- Both sets must contain already-classified instances.
- Both sets must be *representative* samples, where proportions in the sample are good reflections of proportions in the full population.
- Ideally, both datasets must be 'large'.
  - Generally, the larger the *training set*, the better the classifier.
  - Generally, the larger the *test set*, the more reliable the estimated accuracy will be. E.g. we will be more confident if the estimate is based on 1000 test instances than on 100 test instances.

Rarely, however, will we be able to collect *two* large, independent and representative datasets of already-classified instances.

Assuming we have collected one large dataset of already-classified instances, we will look at several ways of forming training and test sets from this single dataset.

1.1 The holdout method

The simplest method is to take your original dataset and partition it into two, randomly selecting instances for a training set (usually 2/3 of the original dataset) and a test set (1/3 of the dataset). You build the classifier using the training set and then evaluate it on the 'held-out' test set.

This has the advantage of being simple. But it makes poor use of the available data and it raises questions about the representativeness of each dataset (e.g. you may just get lucky with all the 'easy' instances in the test set).

1.2 The repeated holdout method

The holdout method can be made more reliable by repeating it several times, with randomly selected training and test sets each time. The accuracies obtained on each iteration are averaged to give an overall accuracy.

The more iterations that are used, the less effect 'lucky' or 'unlucky' sets will have on the result. However, the different test sets may overlap and this may not be ideal.

1.3 Cross-validation

In k-fold cross-validation, the original dataset is first partitioned into k subsets of equal size, $P_1, \ldots, P_k$. Each subset is used in turn as the test set, with the remaining subsets being the training set. In other words, first $P_2, \ldots, P_k$ form the training set and P_1 is the test set; second $P_1, P_3, \ldots, P_k$ form the training set and P_2 is the test set; and so on; finally, on the kth fold, $P_1, \ldots, P_{k-1}$ form the training set and P_k is the test set. The accuracies from each of the 'folds' are averaged to given an overall accuracy. A typical value for k is 10.

This avoids the problem of overlapping test sets and makes very effective use of the available data.

If time is available, you could even repeat the method multiple times, with different partitions each time.

1.4 Leave-one-out cross-validation

Leave-one-out cross-validation (LOOCV) is a special case of k-fold cross-validation in which $k = n$, where n is the size of the original dataset. Hence, the test sets are all of size 1. In other words, first instances $x_2, \ldots, x_n$ form the training set and x_1 is the only test instance; second $x_1, x_3, \ldots, x_n$ form the training set and x_2 is the only test instance; and so on; finally, on the nth fold, $x_1, \ldots, x_{n-1}$ form the training set and x_n is the only test instance.

This makes the best use of the available data and avoid the problems of random selections. It is, however, time-consuming.

1.5 Parameter tuning

Most classifiers have parameters, and their values need to be chosen. The obvious example is the value of k for a kNN classifier. Its value can have a large effect on accuracy. Less obvious, and probably less significant, are the values you use to avoid zero probabilities in a naïve Bayes classifier.

You might just guess these values. Then again, you might want to try out different possibilities and select the best of them. A common but strictly incorrect approach is to build classifiers with different settings, test each of them on the test set, and then report the accuracy found by the best of them. However, the test set should not be used *in any way* to create the classifier. The proper approach requires splitting the original dataset into three: training set, validation set and test set. You select the best of the classifiers based on accuracy on the validation set. Then the accuracy that you report to the world is that which you obtain on the test set.

1.6 And then...

Once evaluation of the classifier is complete, if you decide that you want this classifier to go live (e.g. if its accuracy looks good enough), then you can use the *whole* dataset to build your final classifier.

You might subsequently find that your classifier doesn't perform as accurately in practice as it did on your test set(s). Why might this happen? It is possible that your classifier has found patterns in the training data that are not representative of the population as a whole. In this case, we say that the classifier *overfits* the training set. It happens when the training set contains noise (i.e. erroneous examples) or when the training set is too small to be representative of the population. There are a number of techniques for guarding against overfitting, but we won't discuss them in this module.

2 Demos

In the lecture, we will look at a demo in which we classify the drinking dataset using naïve Bayes and a few versions of kNN

Then we'll look at a demo using a spam database.

Exercises

1. (Past exam question)

   (a) Explain what is meant in A.I. by the term *classification*.

   (b) In a factory, the quality control department must classify the products into two classes: *tainted* or *clean*. Each object has two attributes: the *size* has values *light* or *heavy*; the *colour* has values *black* or *grey*. You collect a dataset of 16 instances (below). Give the *joint probability distribution*.

size	colour	class
light	*grey*	*tainted*
light	*grey*	*tainted*
light	*grey*	*tainted*
light	*black*	*tainted*
light	*black*	*tainted*
light	*black*	*tainted*
light	*black*	*tainted*
heavy	*black*	*tainted*

size	colour	class
heavy	*grey*	*clean*
heavy	*grey*	*clean*
heavy	*grey*	*clean*
heavy	*grey*	*clean*
heavy	*grey*	*clean*
light	*black*	*clean*
heavy	*black*	*clean*
heavy	*black*	*clean*

   (c) Using your joint probability distribution:

      i. Compute $P(\textit{colour} = \textit{grey})$.
      ii. Compute $P(\textit{colour} = \textit{black}, \textit{class} = \textit{clean})$.
      iii. Compute $P(\textit{colour} = \textit{black} | \textit{class} = \textit{tainted})$.
      iv. Determine whether $P(\textit{colour} = \textit{black})$ is independent of $P(\textit{size} = \textit{heavy})$. Show your working.
      v. Classify the following new instance:

$$\{\textit{size} = \textit{light}, \textit{colour} = \textit{black}\}$$

      Show your working.

   (d) For the same dataset, use the *naïve Bayes classifier* to classify the new instance from part (iii)e. Show your working.

   (e) Look at your answers to questions 1(c)v and 1d. If you obtained the *same* classification, **explain in general** when the two classifiers will agree on a given instance. If you obtained *different* classifications, **explain** why this can happen.

   (f) In a competitor's factory, product attributes are: *size*, which has values 0–10; and *colour*, which has values *white*, *grey* or *black*. You collect the following dataset of 5 instances, shown here with a unique identifier for ease of reference:

id	size	colour	class
1	2	*grey*	*clean*
2	4	*grey*	*clean*
3	2	*white*	*tainted*
4	10	*black*	*tainted*

   A kNN classifier is constructed in which: global distance is just the sum of the local distances; range-normalised absolute difference is used to compute the local distances between *size*s; and a local distance function based on the following ordering is used to compute the local distances between *colour*s:

$$\textit{white} < \textit{grey} < \textit{black}.$$

   What classification would this kNN classifier give to the following new instance

$$\{\textit{size} = 8, \textit{colour} = \textit{white}\}$$

      i. using 1NN?
      ii. using 3NN where the class is predicted by majority-voting? and
      iii. using 3NN where the class is predicted using inverse distance-weighted voting?

   Show your working.

2. (Challenge exercise)

   Suppose I run a competition in which I invite people to submit classifiers. They can submit any classifier they wish, not just the ones we have covered so far. I inform entrants that I have a dataset containing 100 instances, exactly 50 of which are of class A, and 50 are of class B. I inform them that I will be using LOOCV to train and test their classifiers using this dataset. The classifier that obtains highest overall accuracy will win a year's supply of *Pants Pizza Parlour* meal vouchers.

   What classifier would you enter in order to guarantee to be a winner? Explain your answer.