Evaluating Vision Systems

Adrian F. Clark <alien@essex.ac.uk>
CSEE, University of Essex

Where the discipline stands

With our current knowledge, we cannot predict how well a vision system will work — unlike, say, designing turbine blades for a jet engine where materials science, computational fluid dynamics *etc* give the designers confidence that the turbine blades will work before having to build and test one

We can find out how well a vision system works only by building and testing it

This shows the discipline has some way to go

Ground truth

To be able to test a vision system, we need images or videos for which the results are known — we call these *ground truth*

Ground truth is normally collected by having experts annotate the data in some way

Even experts can get it wrong sometimes, so we usually involve more than one expert and use those cases where they agree — though this can result in difficult cases being omitted

Sometimes experts aren't always needed, as with the citizen scientists in the Galaxy Zoo projects

Our vision systems label images

For the problems we'll consider in this course, the vision system will read an image and assign a label to it, such as *cancerous* or *benign* in cancer diagnosis

A more general problem is where there are several different regions in an image, each of which has to be labelled — this is more difficult in practice though the principles described here still apply

What a vision system can produce

We give (say) an image containing a cancerous lesion to our vision system and it is labelled as *cancerous* (it has found the right answer): a **TRUE POSITIVE**

We give it (say) an image with no cancerous lesion but it is incorrectly labelled as *cancerous*: a **FALSE POSITIVE**

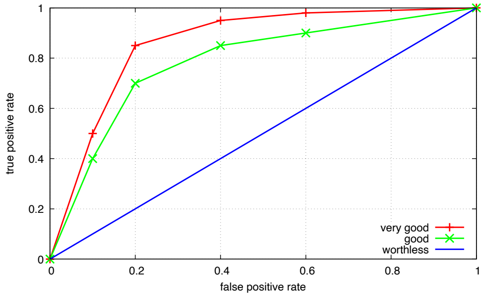
We give it an image of (say) a cancerous lesion but the vision system rejects it: a **FALSE NEGATIVE**

We give the vision system an image of (say) a coffee cup and it is rejected: a **TRUE NEGATIVE**

There is very little training on negative outcomes in computer vision; there should be more. We'll see a good example of where it is done when we look at the Viola-Jones algorithm for detecting faces.

Receiver Operating Characteristic (ROC) curves

A line on a ROC curve is produced by changing the value of a tuning parameter such as a threshold



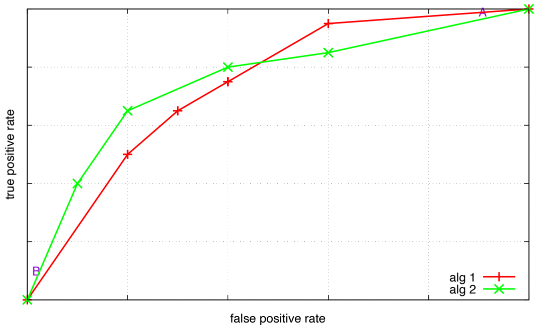
ROC curves

Interpreting a ROC curve

There is *always* a trade-off between a high true positive rate and a high false positive rate

Closer to the top-left corner means the algorithm performs better

Crossing ROC curves



Crossing ROC curves

We might ask ourselves whether *alg1* or *alg2* is better

Where to operate an algorithm

You can decide only in the context of the problem the vision system is being used for

For *cancer detection* (say), we want the TP rate to be as high as possible so we choose *alg1* and operate it at point **A** on the graph, accepting that it will generate more FPs

For *secure access by face recognition* (say), we want the FP rate to be as low as possible so we choose *alg2* and operate it at point **B** on the graph, accepting that people may have to try several times

You'll often see the *area under the curve* used to choose which algorithm is better — but as we have just seen, this makes no sense operationally; and as we shall see in a moment, no sense for comparison either

Comparing performance

If you see two lines on a ROC curve or two accuracy figures in a table, you might reasonably ask which is better

To help us get a grip on this, imagine taking a fair coin and tossing it a number of times. Let us say that we obtain:

- 3 heads from 10 tosses;
- 30 heads from 100 tosses;
- 300 heads from 1,000 tosses.

Let us now consider which of these is the most surprising.

The binomial distribution

A coin toss obeys a *binomial* distribution

$$P(t) = \binom{N}{t} p^t (1-p)^t$$

with mean Np and variance $Np(1-p)$. As the coin is fair, $p = \frac{1}{2}$ and so the three cases work out as:

1. The expected number of heads (*ie* the mean) is $Np = 5$. The sd is $\sqrt{Np(1-p)} = \sqrt{10 \times \frac{1}{2} \times \frac{1}{2}} = 1.58$. Hence, 3 heads is $(5-3)/1.58 \approx 1.3$ sds from the mean.
2. 30 heads is $(50-30)/5 = 4$ sds from the mean.
3. 300 heads is $(500-300)/15.8 \approx 13$ sds from the meam.

Counting successes and failures

Let's take two algorithms and see if they succeed or fail on the same test input:

- if both succeed, we increase N_{ss} by 1
- if both fail, we increase N_{ff} by 1
- if the first algorithm being tested succeeds and the second fails, we increase N_{sf} by 1
- if the first algorithm fails and the second succeeds, we increase N_{fs} by 1

McNemar's test

Having accumulated N_{ss} *etc*, we plug them into the formula

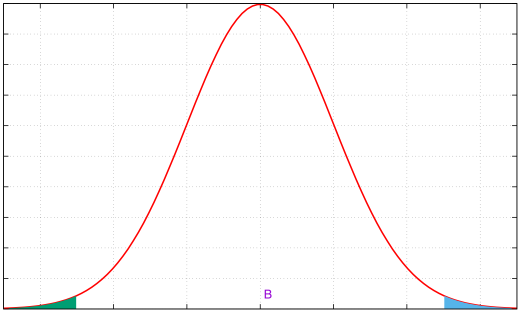
$$Z^2 = \frac{(|N_{sf} - N_{fs}|)^2}{N_{sf} + N_{fs}}$$

Note that this doesn't make use of N_{ss} or N_{ff} because they don't provide any useful information for the comparison

Because the procedure forces a binary choice to be made regarding which one is better in each test, Z follows a *binomial* distribution, just like tossing a coin

The binomial distribution is symmetric and bell-shaped

Tails of the binomial distribution



Tails of the binomial distribution

When $z = 1.96$, there is a one-in-twenty chance that the results are **not** statistically significant but are instead due to some quirk of the data

Critical values of Z

So when $Z \geq 1.96$, we can be fairly happy that the two algorithms being compared have different performances

This is a *vastly* better way of comparing performances than using ROC curves. . . it is used in industry, or was when I worked there, but is rarely used in academic papers — though I don't understand why as it provides irrefutable evidence that a new technique is an improvement on an existing one

Is one algorithm *better* than another?

If you want to decide whether one algorithm is better than another, the critical value of Z is 1.646 rather than 1.96 — this is because we're performing a *one-tailed* test rather than a *two-tailed* one; these values are given in Table 6.2 of the notes

McNemar's test is used in `fact compare` in some of the laboratories

Evaluating performance with FACT

You may already have used FACT to produce tables of TP *etc* and several quantities calculated from those measures, such as *specificity* and *recall*

You can get their definitions by adding `--detail=2` to the `fact` command line

FACT also outputs a class confusion matrix; running it with `--detail=2` also explains how it is interpreted

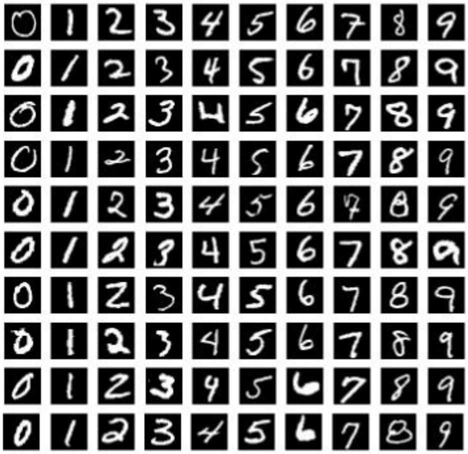
MNIST

The MNIST dataset of handwritten digits has become a kind of standard for evaluating vision systems:

- 60,000 training images
- 10,000 test images

though with small differences in the number in each class

People have recently started using a drop-in replacement for MNIST called *FASHION-MNIST*, which is a more difficult problem



Examples from the MNIST dataset

Using MNIST for evaluating algorithms

Section 6.5 of the lecture notes give an example from some of my research of testing a single machine learning technique, a *Support Vector Machine*, on the MNIST dataset

You will see from the FACT-like tables that the accuracy achieved by the SVM with my technique's tuning parameter $T = 1.0$ is better than that with $T = 0.4$, and using McNemar's test confirms that the performance differences are significant

Do study the class confusion matrices too: you'll see that '3' and '5' are often confused, as are '4' and '9' — these are not surprising — but did you expect '2' and '7' to be confused?

This kind of study gives insight into how human-developed algorithms work. . . though not into the shortcomings of solutions produced by machine learning, a problem that will be mentioned again later in the module