When to change dev:test sets and metrics

(SPEECH)
You've

(DESCRIPTION)
Text, Setting up your goal. When to change dev, test sets and metrics.

(SPEECH)
seen how set to have a dev set and evaluation metric is like placing a target somewhere for your team to aim at.

But sometimes partway through a project you might realize you put your target in the wrong place.

In that case you should move your target.

Let's take a look at an example.

Let's

(DESCRIPTION)
New slide, Cat dataset examples. Metric, classification error. Algorithm A, 3 percent error. Algorithm B, 5 percent error.

(SPEECH)
say you build a cat classifier to try to find lots of pictures of cats to show to your cat loving users and the metric that you decided to use is classification error.

So algorithms A and B have, respectively, 3 percent error and 5 percent error, so it seems like Algorithm A is doing better.

But let's say you try out these algorithms, you look at these algorithms and Algorithm A, for some reason, is letting through a lot of the pornographic images.

So if you shift Algorithm A the users would see more cat images because you'll see 3 percent error and identify cats, but it also shows the users some pornographic images which is totally unacceptable both for your company, as well as for your users.

In contrast, Algorithm B has 5 percent error so this classifies fewer images but it doesn't have pornographic images.

So from your company's point of view, as well as from a user acceptance point of view, Algorithm B is actually a much better algorithm because it's not letting through any pornographic images.

So, what has happened in this example is that Algorithm A is doing better on evaluation metric.

It's getting 3 percent error but it is actually a worse algorithm.

In this case, the evaluation metric plus the dev set prefers Algorithm A because they're saying, look, Algorithm A has lower error which is the metric you're using but you and your users prefer Algorithm B because it's not letting through pornographic images.

So when this happens, when your evaluation metric is no longer correctly rank ordering preferences between algorithms, in this case is mispredicting that Algorithm A is a better algorithm, then that's a sign that you should change your evaluation metric or perhaps your development set or test set.

In this case the misclassification error metric that you're using can be written as follows: this one over m, a number of examples in your development set, of sum from i equals 1 to mdev, number of examples in this development set of indicator of whether or not the prediction of example i in your development set is not equal to the actual label i, where they use this notation to denote their predictive value.

Right. So these are zero.

And this indicates a function notation, counts up the number of examples on which this thing inside it's true.

So this formula just counts up the number of misclassified examples.

The problem with this evaluation metric is that they treat pornographic and non-pornographic images equally but you really want your classifier to not mislabel pornographic images, like maybe you recognize a pornographic image in cat image and therefore show it to unsuspecting user, therefore very unhappy with unexpectedly seeing porn.

One way to change this evaluation metric would be if you add the weight term here, we call this w(i) where w(i) is going to be equal to 1 if x(i) is non-porn and maybe 10 or maybe even large number like a 100 if x(i) is porn.

So this way you're giving a much larger weight to examples that are pornographic so that the error term goes up much more if the algorithm makes a mistake on classifying a pornographic image as a cat image.

In this example you giving 10 times bigger weights to classify pornographic images correctly.

If you want this normalization constant, technically this becomes sum over i of w(i), so then this error would still be between zero and one.

The details of this weighting aren't important and actually to implement this weighting, you need to actually go through your dev and test sets, so label the pornographic images in your dev and test sets so you can implement this weighting function.

But the high level of take away is, if you find that evaluation metric is not giving the correct rank order preference for what is actually better algorithm, then there's a time to think about defining a new evaluation metric.

And this is just one possible way that you could define an evaluation metric.

The goal of the evaluation metric is accurately tell you, given two classifiers, which one is better for your application.

For the purpose of this video, don't worry too much about the details of how we define a new error metric, the point is that if you're not satisfied with your old error metric then don't keep coasting with an error metric you're unsatisfied with, instead try to define a new one that you think better captures your preferences in terms of what's actually a better algorithm.

(DESCRIPTION)
New slide, Orthogonalization for cat pictures, anti-porn. One, So far we've only discussed how to define a metric to evaluate classifiers. Two, Worry separately about how to do well on this metric.

(SPEECH)
One thing you might notice is that so far we've only talked about how to define a metric to evaluate classifiers.

That is, we've defined an evaluation metric that helps us better rank order classifiers when they are performing at varying levels in terms of streaming of porn.

And this is actually an example of an orthogonalization where I think you should take a machine learning problem and break it into distinct steps.

One step is to figure out how to define a metric that captures what you want to do, and I would worry separately about how to actually do well on this

(DESCRIPTION)
An inset photo of an archery target is displayed.

(SPEECH)
metric.

So think of the machine learning task as two distinct steps.

To use the target analogy, the first step is to place the target.

So define where you want to aim and then as a completely separate step, this is one you can tune which is how do you place the target as a completely separate problem.

Think of it as a separate step to tune in terms of how to do well at this algorithm, how to aim accurately or how to shoot at the target.

Defining the metric is step one and you do something else for step two.

In terms of shooting at the target, maybe your learning algorithm is optimizing some cost function that looks like this, where you are minimizing some of losses on your training set.

One thing you could do is to also modify this in order to incorporate these weights and maybe end up changing this normalization constant as well.

So it just 1 over a sum of w(i).

Again, the details of how you define J aren't important, but the point was with the philosophy of orthogonalization think of placing the target as one step and aiming and shooting at a target as a distinct step which you do separately.

In other words I encourage you to think of, defining the metric as one step and only after you define a metric, figure out how to do well on that metric which might be changing the cost function J that your neural network is optimizing.

Before going on, let's look at just one more example.

(DESCRIPTION)
New slide, Another example. Algorithm A, 3 percent error. Algorithm B, 5 percent error. Under the label dev, test, three photos of cats are displayed.

(SPEECH)
Let's say that your two cat classifiers A and B have, respectively, 3 percent error and 5 percent error as evaluated on your dev set.

Or maybe even on your test set which are images downloaded off the internet, so high quality well framed images.

But maybe when you deploy your algorithm product, you find that algorithm B actually looks like it's performing better, even though it's doing better on your dev set.

(DESCRIPTION)
A second trio of cat photos is displayed to the right of the originals, with the label, User images.

(SPEECH)
And you find that you've been training off very nice high quality images downloaded off the Internet but when you deploy those on the mobile app, users are uploading all sorts of pictures, they're much less framed, you haven't only covered the cat, the cats have funny facial expressions, maybe images are much blurrier, and when you test out your algorithms you find that Algorithm B is actually doing better.

So this would be another example of your metric and dev test sets falling down.

The problem is that you're evaluating on the dev and test sets a very nice, high resolution, well-framed images but what your users really care about is you have them doing well on images they are uploading, which are maybe less professional shots and blurrier and less well framed.

So the guideline is, if doing well on your metric and your current dev sets or dev and test sets' distribution, if that does not correspond to doing well on the application you actually care about, then change your metric and your dev test set.

In other words, if we discover that your dev test set has these very high quality images but evaluating on this dev test set is not predictive of how well your app actually performs, because your app needs to deal with lower quality images, then that's a good time to change your dev test set so that your data better reflects the type of data you actually need to do well on.

But the overall guideline is if your current metric and data you are evaluating on doesn't correspond to doing well on what you actually care about, then change your metrics and/or your dev/test set to better capture what you need your algorithm to actually do well on.

Having an evaluation metric and the dev set allows you to much more quickly make decisions about is Algorithm A or Algorithm B better.

It really speeds up how quickly you and your team can iterate.

So my recommendation is, even if you can't define the perfect evaluation metric and dev set, just set something up quickly and use that to drive the speed of your team iterating.

And if later down the line you find out that it wasn't a good one, you have better idea, change it at that time, it's perfectly okay.

But what I recommend against for the most teams is to run for too long without any evaluation metric and dev set up because that can slow down the efficiency of what your team can iterate and improve your algorithm.

So that says on when to change your evaluation metric and/or dev and test sets.

I hope that these guidelines help you set up your whole team to have a well-defined target that you can iterate efficiently towards improving performance.