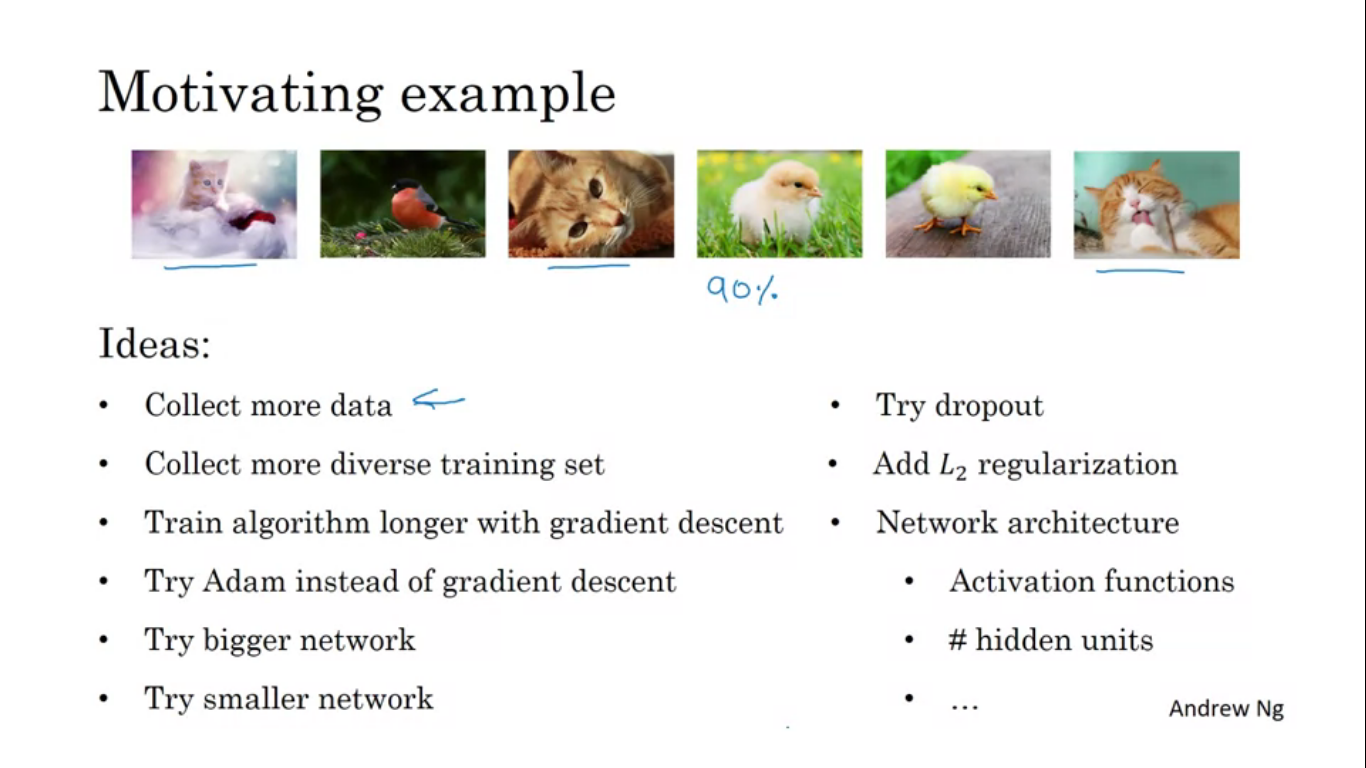
ML Strategy



When trying to improve a deep learning system, we often have a lot of ideas we could try. If we choose poorly we will waste both our time and resources.

Orthogonalization in the sense of tv refers to that the tv designers have designed the knobs so that each knob does only on kind of tuning. This makes it easier to tune the tv in comparison to the fact that with touching one knob we can change various tunings.

So in a car if we have separate controls then it is orthogonal but if there is a mixture of controls it becomes much more difficult to control the car.

Orthogonal means at 90 degrees.

Four things to keep in mind/ chain of assumptions in ML

First, you are doing good on the training set. For some applications this might mean doing comparably human level performance.

Next we hope that this also fits well on the dev set.

Then we hope it is doing well on the test set

Finally we hope it is doing good in the real world.

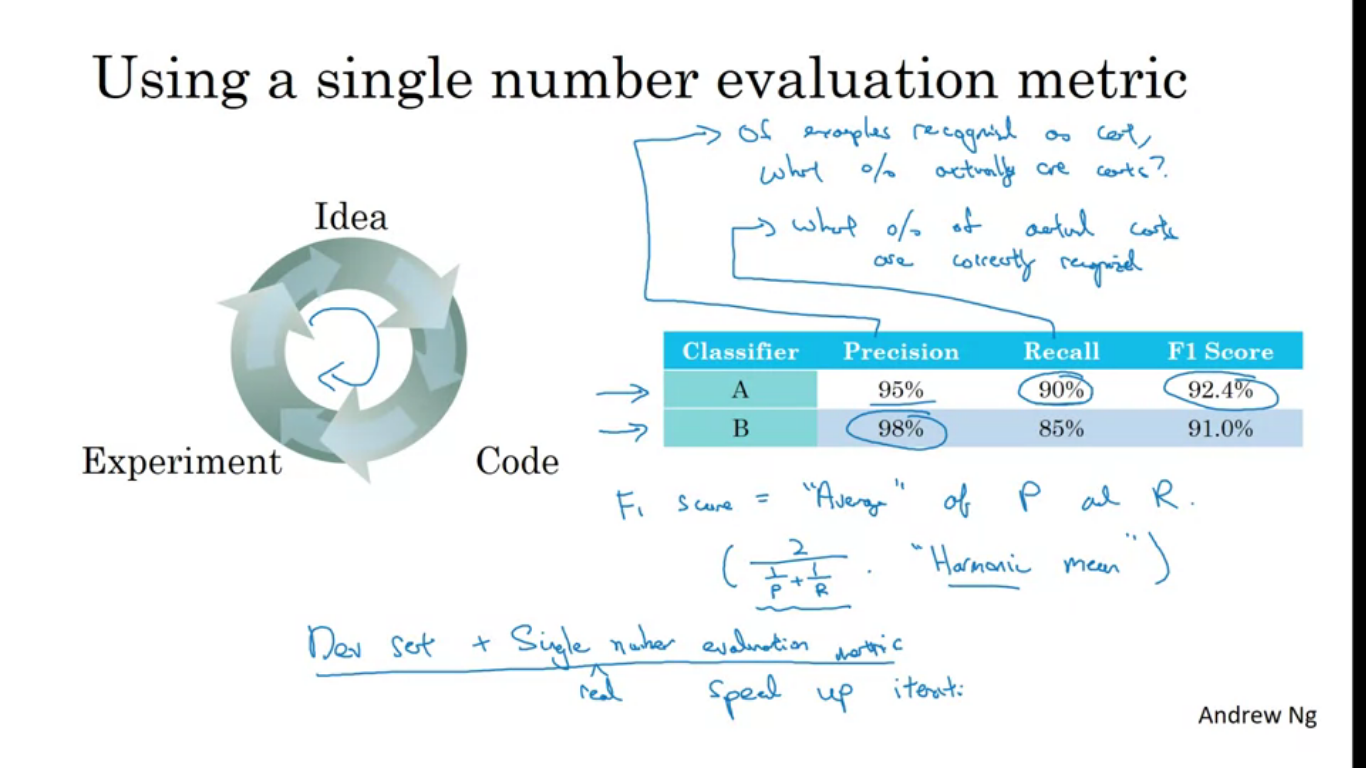
So if our model is not fitting well on training well on training set, we want one knob, or some specific knobs we can use to make sure we can tune our algorithm to make it fit well on the training set. Sp he knobs for training set are - bigger network, or better training algorithm

If we find that the algorithm is not fitting dev set well, then there is separate set of knobs including regularization, getting a bigger training set.

If we find we do well on dev set and not on test set, then we need a dev set, as our model has over-trained the dev set, and we need a bigger dev set.

Lastly if it does all well but does not do well on real life, we might want to go back and change either the dev set or the cost function.

**Single Number Evaluation metric**

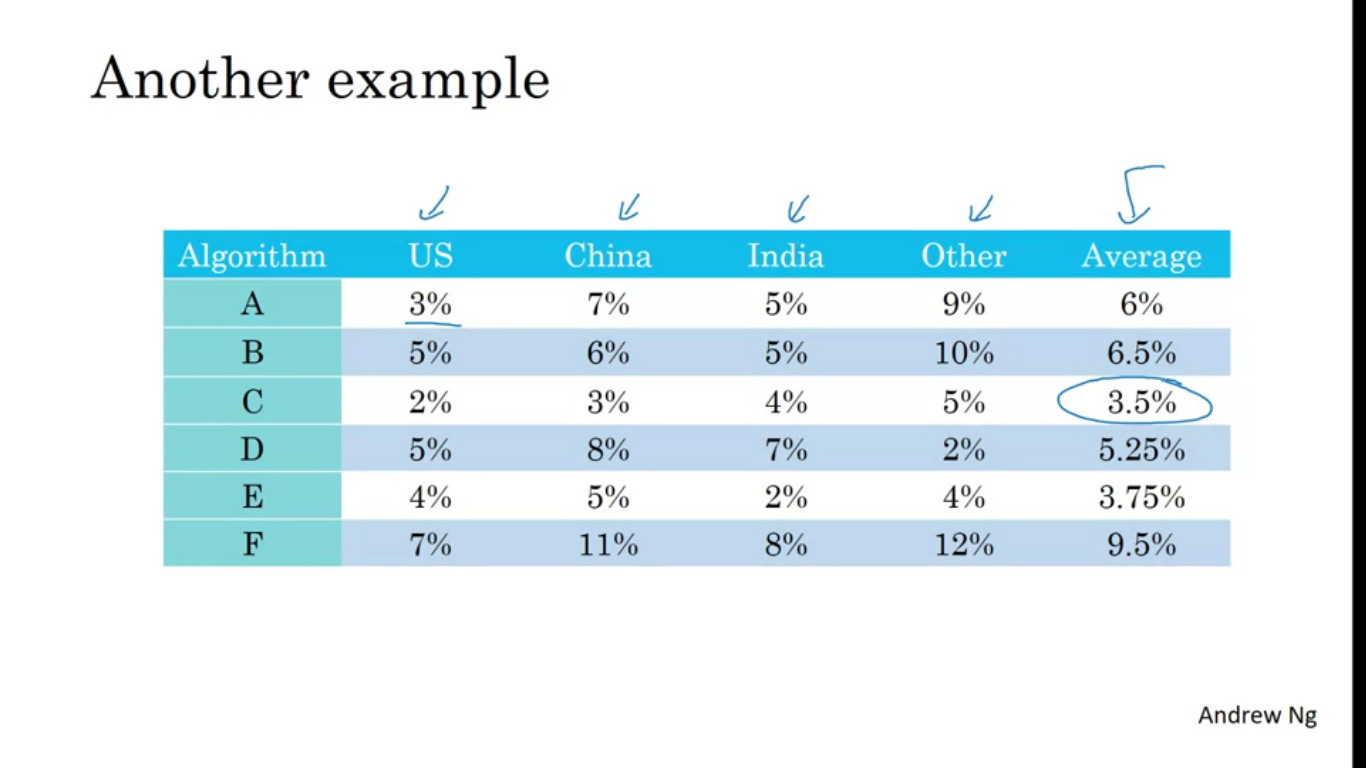


When you are trying different options of building your machine learning systems you’ll find that your progress will be real fast if you have a single real number evaluation metric, that lets you quickly tell if the new thing you tried is working better or worse than your last idea.

One reasonable way to evaluate the performance of your classifier is to look at its precision and recall. Precision means of the example that your classifier recognizes as cats what % actually are cats. Recall is of all the images which really are cats what % were correctly recognized by your classifier. So it’s reasonable to try to evaluate the classifier in terms of its precision and recall.

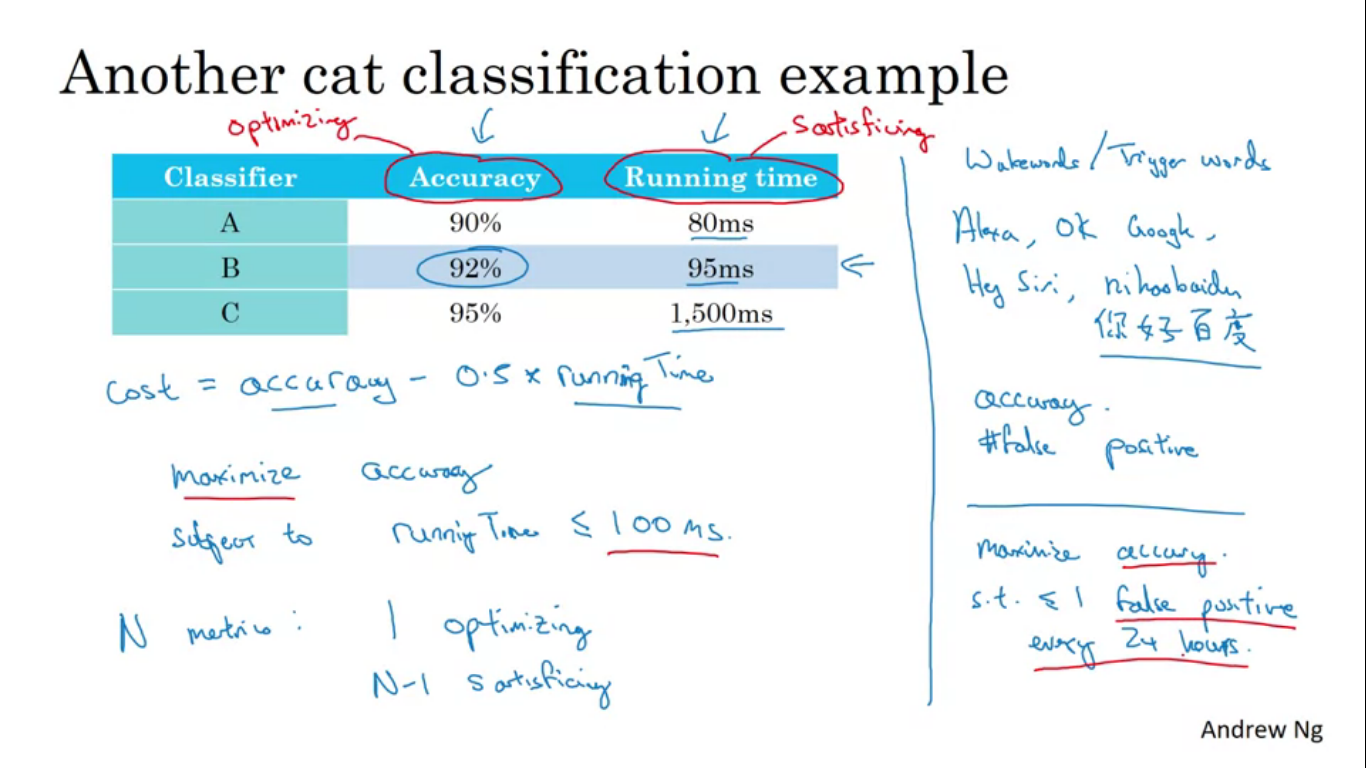
If a classifier A does better on recall and classifier B does better on precision, then you are not sure which classifier is better. With two evaluation metrics it is difficult to know which is better. So rather than using only precision and recall, we define a new evaluation metric called as F1-Score. F1-Score is the average of precision P and recall R. Since classifier A has a better F1 score so we can select classifier A over B.

Having a well defined dev set by which you are measuring precision and recall, and a single row number evaluation metric helps us find out whether classifier A or B which is better. It speeds up the iterative process of improving your ML algorithm.



Let’s say you are building cat app in four major geographies US, China, India and Other. In addition to comparing multiple algorithm performances, we also find out average, which serves as a single evaluation metric. We see algorithm C has the lowest average error.

**Satisficing and optimizing metrics**



It’s always not easy to combine all the things we care into a single row number evaluation metric. In those cases it is useful to setup satisificing and optimizing metrics.

Seeing the example one thing we can do is combining the accuracy and running time to find out overall evaluation metric. So combining using a formula:

Cost = accuracy -0.5\*running time

But it seems a little artificial combining the two like this. So what else we can do is to maximise accuracy but subject to that the running time has to be less than or equal to 100ms.

So accuracy is an optimizing matrix, but the running time is a satisficing metric, meaning it just has t be less than 100 ms and else we don’t care.

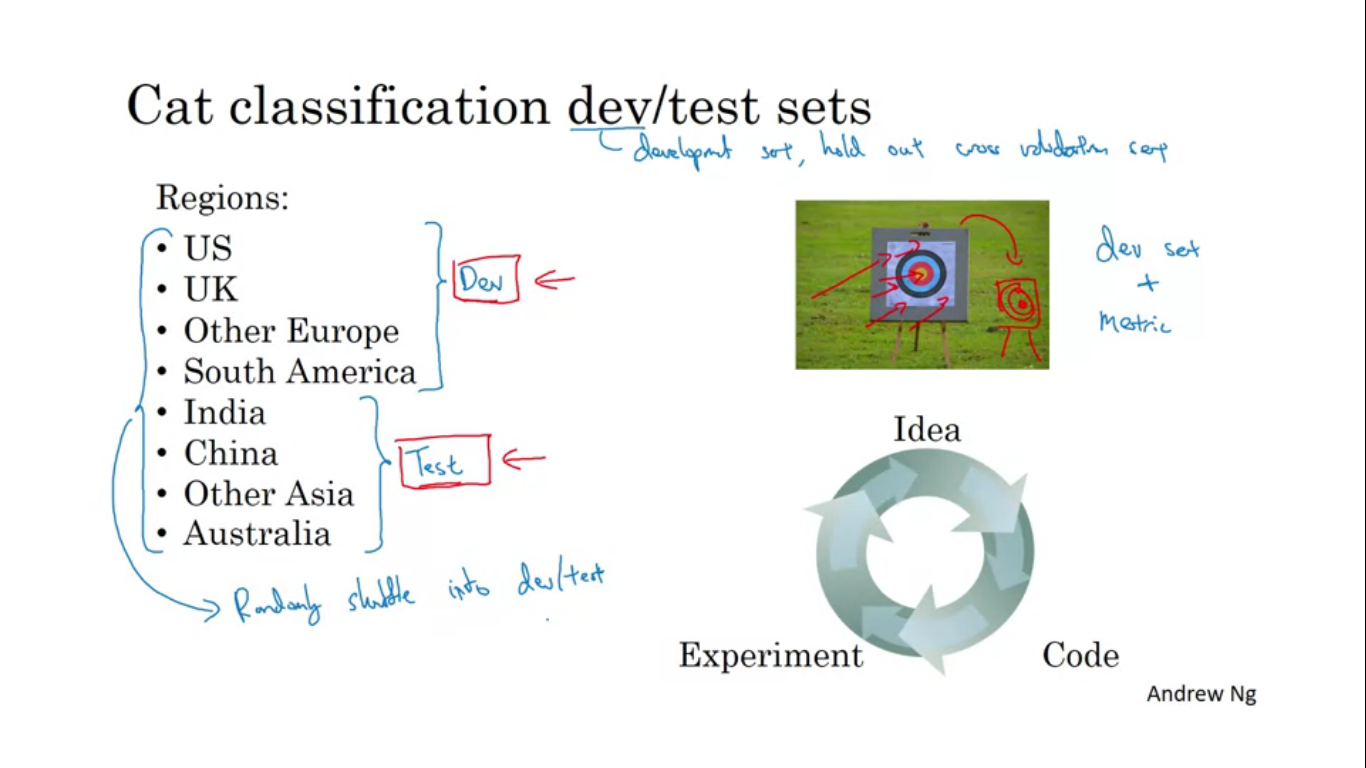
N metrics that we care about it’s sometimes reasonable to pick one of them to be optimizing and N-1 to be satisficing.

Let’s say we are building a system to detect wake words/trigger words. We might care about the accuracy of trigger word detection system and also false positives, so when no one actually said the trigger word how often does it randomly wake up. Maximize accuracy subject to you have one FP every 24 hours.

**Train/Dev/Test set distributions**

The way we distribute train/dev/test set really speeds/slows down the complete process.

Dev set/development set/hold out cross validation set



Let’s say we are creating the cat classifier and the countries are as mentioned, to divide them, we could pick first four of these regions and put in dev set and other foes into test set. This is a bad idea since it comes out from different distributions.

Once we set up the dev set and define the metric, the team can iterate very quickly and use the dev set and metric to evaluate classifiers and try to pick the best one. So here the data from the regions in the test set might be very different from those in dev set.

So the best thing to do would be to randomly shuffle the data into the dev and the test set, so that both have data from all the 8 regions.

Given an input X about a loan application can you predict Y whether or not they will replay back the loan? This helps you decide whether or not to approve the loan. The dev set came from medium income zip codes, but the team decided to test them on low income zip codes. The distribution data from medium and low income zip codes are different.

**Choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.**

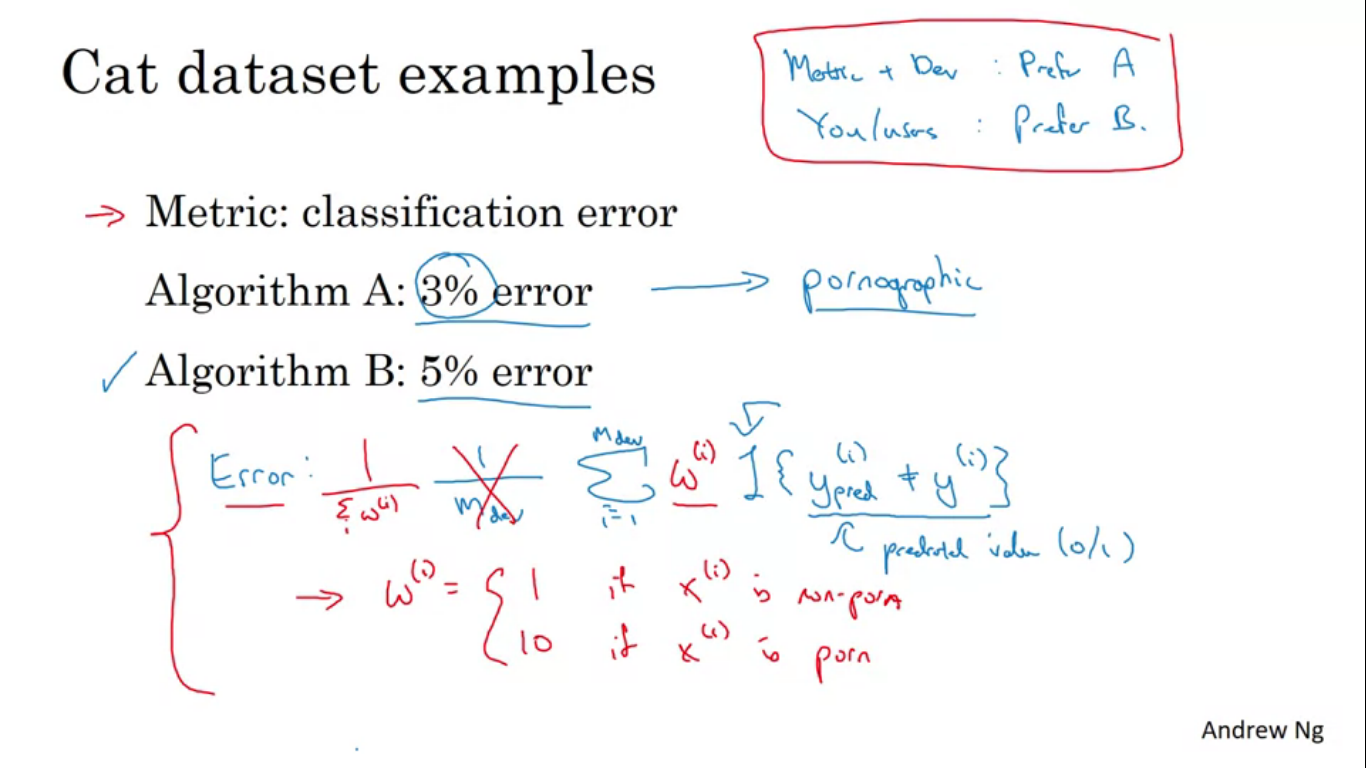
**The dev set and test set should come from the same distribution.**

**Size of dev and test set**

Size of test set to be big enough to give high confidence in the overall performance of the system. The old ML rule of 70/30 split or 60/20/20 split no longer applies and dev and test set sizes have reduced.

When to change the dev/test set and metrics

If somehow it happens that you realise that halfway through the training you realise that you put your target in the wrong place.

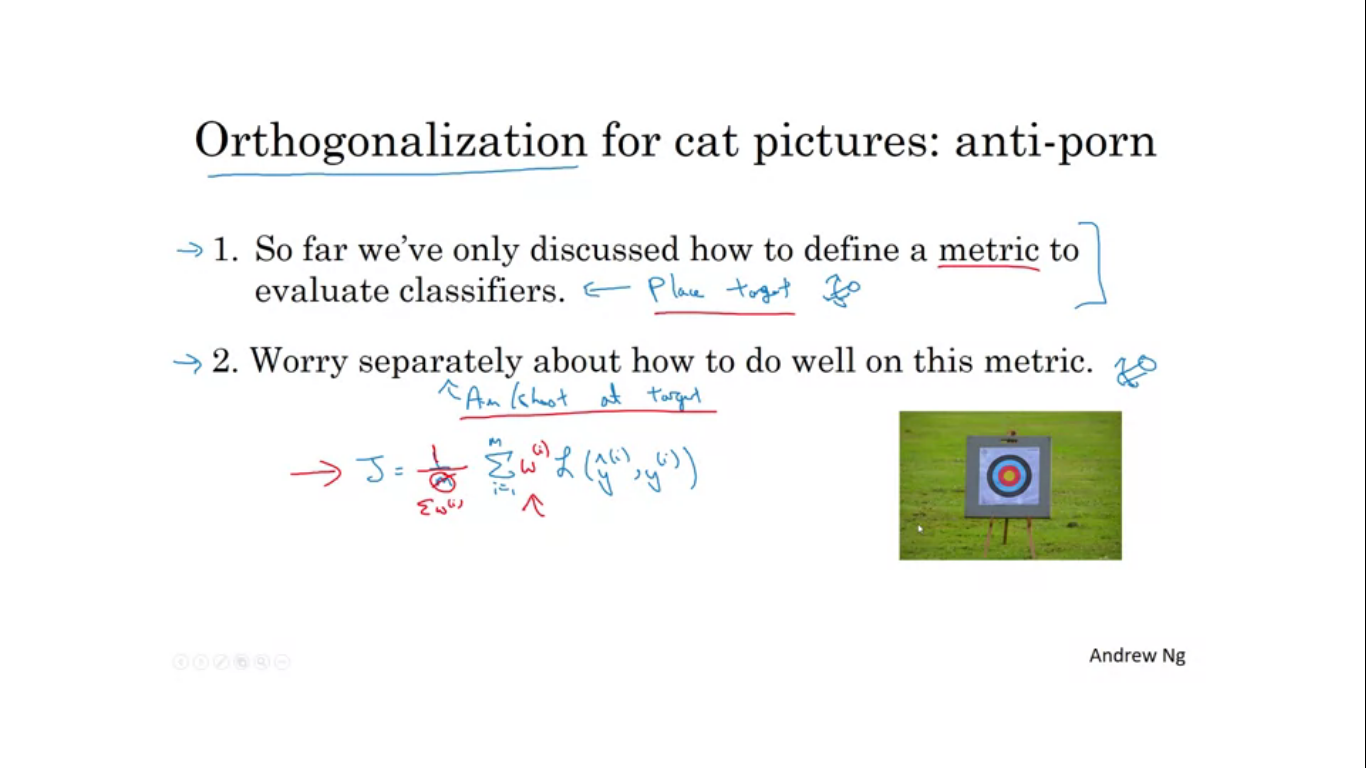


Metric is the classification error cat classifier example, Algorithm A and Algorithm B have 3 % and 5% error. It seems Algorithm A has lesser error so better but it shows porn, whereas algorithm B has greater error but is porn free. So from company’s point of view Algorithm B is a much better algorithm.

So here evaluation metric + dev set prefers algorithm A but the user prefer algorithm B her our rank metric is not giving us proper result, it is mis-predicting Algorithm A as better algorithm then it is the sign that you should change your evaluation metric or perhaps your dev/test set.

The formula just counts up the number of misclassified examples. Problem with this metric is that it treats porn and non porn equally. One way to change the evaluation metric is to add a weight term, as shown. So this way we are giving a much larger weight to porn examples so that error increases.

**Orthogonalisation for cat pictures**



So far we have only discussed how to define a metric to evaluate classifiers

Worry separately about how to actually do well on this metric.

**Comparing ML systems to human level performance**



Advancement in ML, have made machines to actually become competitive t human level performance. Workflow is much efficient when you try to do something that humans can also do.

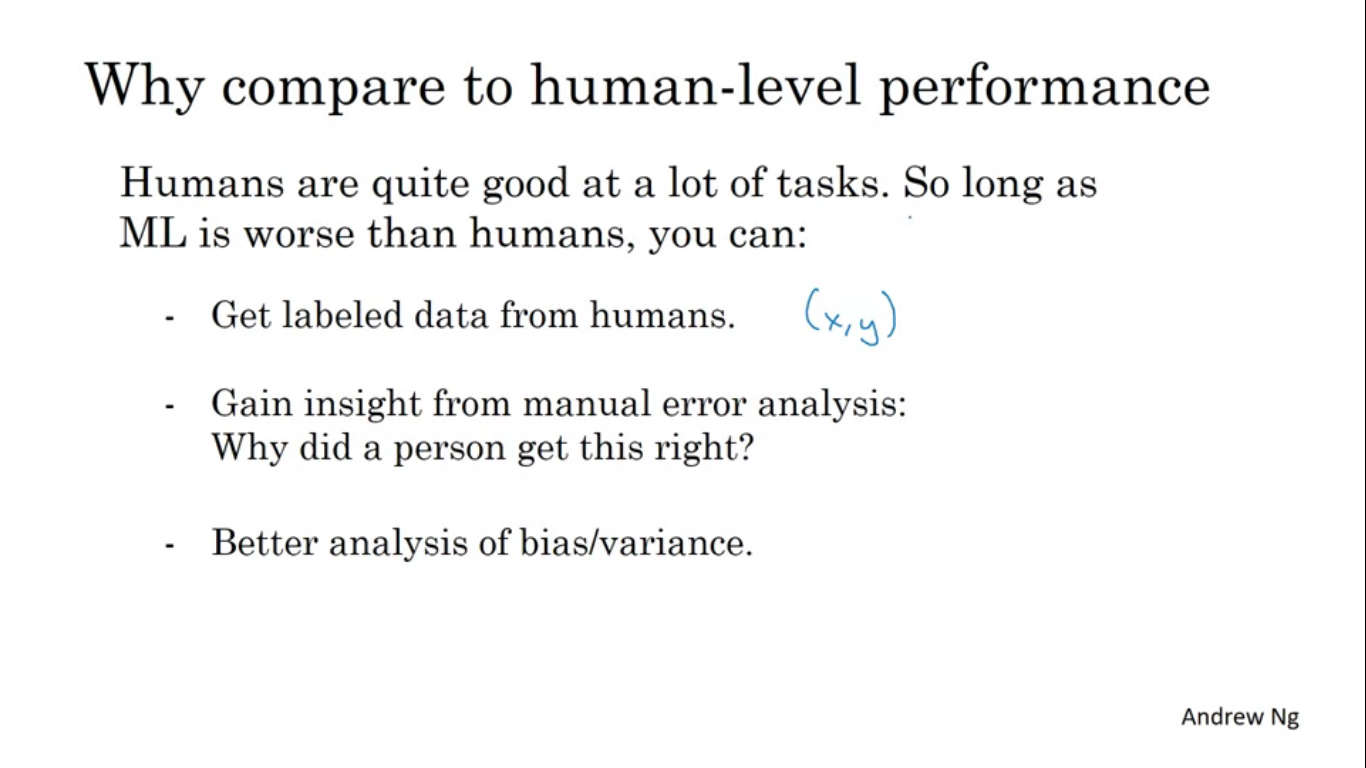
When training a algorithm, it initially increases up to human level accuracy and then the slope of the performance increase actually slows down and it never actually crosses a theoritical limit called as baye’s optimal error. Best possible error, so if we say audio some audio are so noisy that no one can tell its transcript, or some cat images are so blurry that no one can tell about it. Baye’s error is very best theoretical function for mapping from X->Y. Progress is quite fast until you surpass human level performance.

**Why progress slows down after human level performance?**

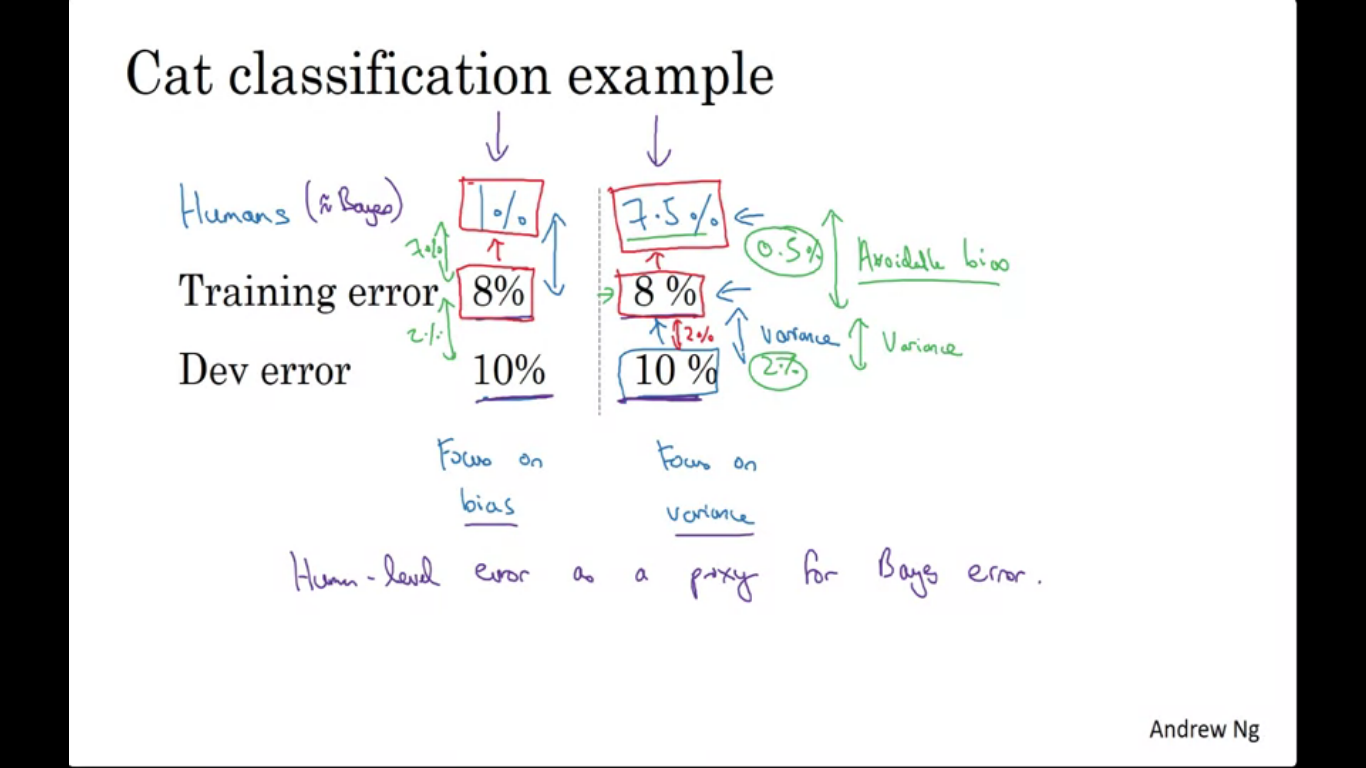
Human level performance for many tasks is not that far from baye’s optimal error.

So long the performance is worse than human level there are certain tools we can use which are harder to use once we surpassed human level performance

**Why compare to human level performance?**



**Avoidable Bias**



For cat classification, human error is 1%, the learning algorithm achieves 8% training error and 10% dev set error then maybe we wanted to do better on the training set. Huge gap between human performance and training shows the algorithm no fitting the training set well so we have to focus on reducing bias. So train a bugger NN or run GD on training set longer.

In another application lets have the same value of training and dev error and human error is 7.5%. it is doing fine on the training set and we would want to reduce variance.

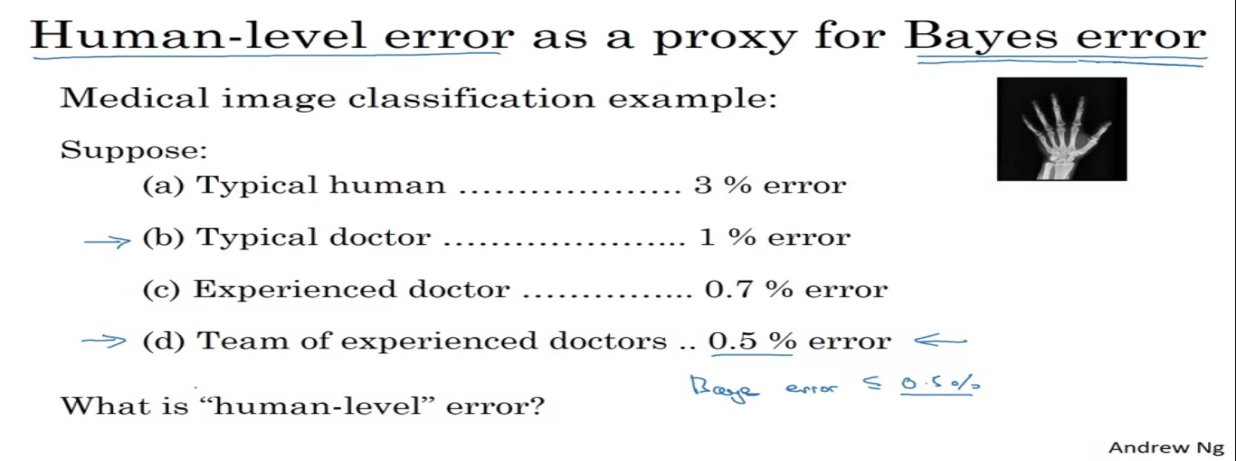
Human level error as a estimate for baye’s error. So depending on baye’s error value with the same training and dev error we decided to focus on whether bias /variance reduction tactics.

Difference between Bayes error and training error is avoidable bias.

You can’t do better than baye’s error unless you are overfitting.

Difference between training error and dev error is variance problem.

Having estimate of human level performance gives us an estimate of bye’s error allowing us to more quickly to make decision to reduce bias/variance.



Surpassing human level performance

Team of humans = 0.5% error

One human = 1% error

Training error = 0.6% error

Dev error = 0.8% error

0.5% is the estimate of baye’s error, so avoidable bias is 0.1% and variance as 0.2%.

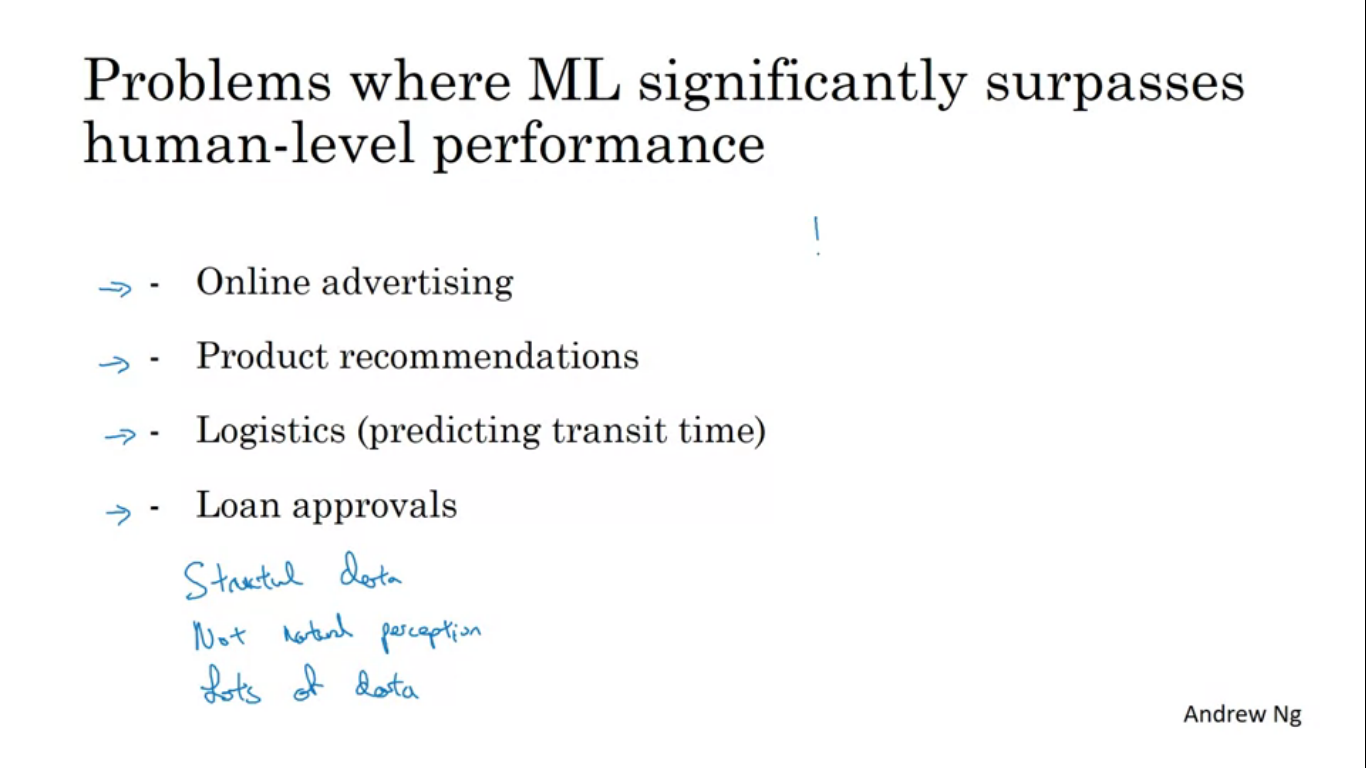
Team of humans = 0.5% error

One human = 1% error

Training error = 0.3% error

Dev error = 0.4% error

So does this mean we have over fitted by 0.2% or the baye’s error is actually 0.1% or 0.2% or 0.3%? we don’t have enough information to tell if we should reduce bias or variance. Moreover if your accuracy is better then it is harder to rely on human intuition.



All four examples are learning from structured data, these are not natural perception problem like computer vision or NLP.

**Two fundamental assumptions of supervised learning**

* You can fit the training set pretty well. You can achieve low avoidable bias
* Training set performance generalizes pretty well to the dev set or test set.

In view of orthogonalization, there are certain knobs to fix avoidable bias issues (bigger net, training longer) and certain other set of issues to deal variance (regularization, get more data).

