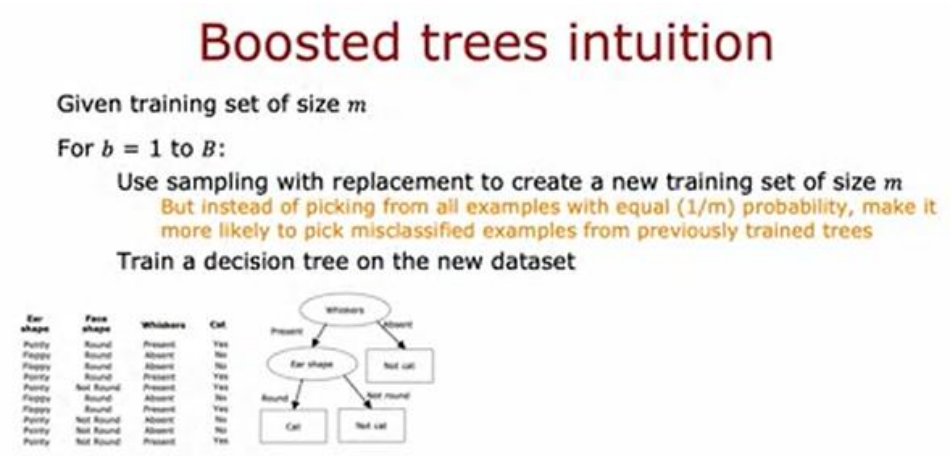
# XGBoost

Over the years, machine learning researchers have come up with a lot of different ways to build decision trees and decision tree ensembles. Today by far the most commonly used way or implementation of decision tree ensembles or decision trees there's an algorithm called XGBoost. It runs quickly, the open source implementations are easily used, has also been used very successfully to win many machine learning competitions as well as in many commercial applications.

Let's take a look at how XGBoost works. There's a modification to the back decision tree algorithm that we saw in the last video that can make it work much better. Here again, is the algorithm that we had written down previously. Given the training set to size m, you repeat B times, use sampling with replacement to create a new training set of size M and then train the decision tree on the new data set.

And so the first time through this loop, we may create a training set like that and train a decision tree like that. But here's where we're going to change the algorithm, which is every time through this loop, other than the first time, that is the second time, third time and so on. When sampling, instead of picking from all m examples of equal probability with one over m probability, let's make it more likely that we'll pick misclassified examples that the previously trained trees do poorly on.

In training and education, there's an idea called deliberate practice. For example, if you're learning to play the piano and you're trying to master a piece on the piano rather than practicing the entire say five minute piece over and over, which is quite time consuming. If you instead play the piece and then focus your attention on just the parts of the piece that you aren't yet playing that well in practice those smaller parts over and over. Then that turns out to be a more efficient way for you to learn to play the piano well.




And so this idea of boosting is similar. We're going to look at the decision trees, we've trained so far and look at what we're still not yet doing well on. And then when building the next decision tree, we're going to focus more attention on the examples that we're not yet doing well.

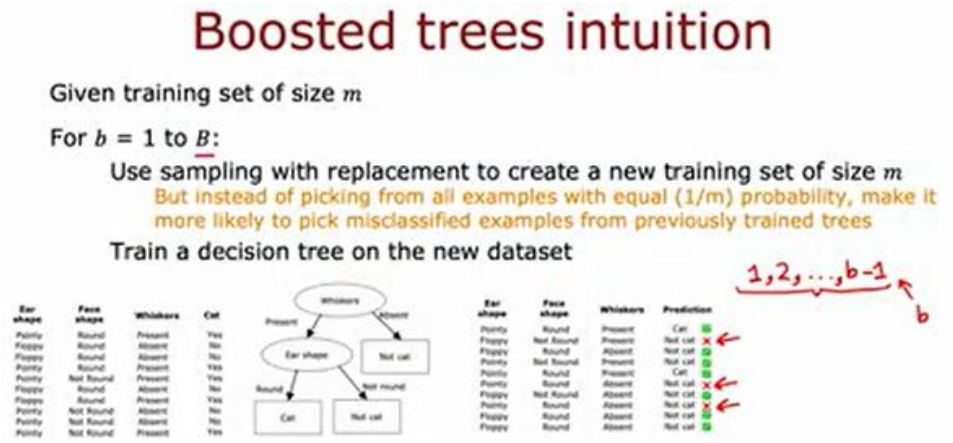
So rather than looking at all the training examples, we focus more attention on the subset of examples is not yet doing well on and get the new decision tree, the next decision tree reporting ensemble to try to do well on them. And this is the idea behind boosting and it turns out to help the learning algorithm learn to do better more quickly.

So in detail, we will look at this tree that we have just built and go back to the original training set. Notice that this is the original training set, not one generated through sampling with replacement.

And we'll go through all ten examples and look at what this learned decision tree predicts on all ten examples. So this fourth most column are their predictions and put a checkmark across next to each example, depending on whether the trees classification was correct or incorrect. So what we'll do in the second time through this loop is we will sort of use sampling with replacement to generate another training set of ten examples.

But every time we pick an example from these ten will give a higher chance of picking from one of these three examples that were still misclassifying. And so this focuses the second decision trees attention via a process like deliberate practice on the examples that the album is still not yet doing that well. And the boosting procedure will do this for a total of B times where on each iteration, you look at what the ensemble of trees for trees 1, 2 up through (b- 1), are not yet doing that well on.

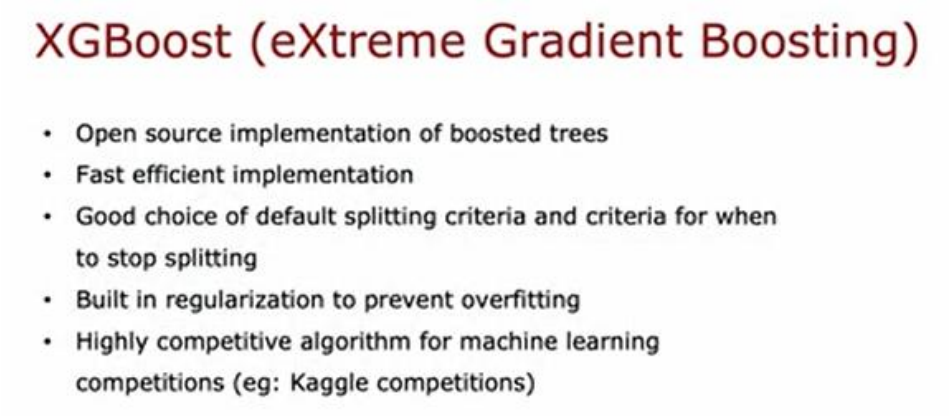
And when you're building tree number b, you will then have a higher probability of picking examples that the ensemble of the previously sample trees is still not yet doing well on.




The mathematical details of exactly how much to increase the probability of picking this versus that example are quite complex, but you don't have to worry about them in order to use boosted tree implementations. And of different ways of implementing boosting the most widely used one today is XGBoost, which stands for extreme gradient boosting, which is an open source implementation of boosted trees that is very fast and efficient.

XGBoost also has a good choice of the default splitting criteria and criteria for when to stop splitting. And one of the innovations in XGBoost is that it also has built in regularization to prevent overfitting, and in machine learning competitions such as does a widely used competition site called Kaggle. XGBoost is often a highly competitive algorithm.

In fact, XGBoost and deep learning algorithms seem to be the two types of algorithms that win a lot of these competitions. And one technical note, rather than doing sampling with replacement XGBoost actually assigns different weights to different training examples.




So it doesn't actually need to generate a lot of randomly chosen training sets and this makes it even a little bit more efficient than using a sampling with replacement procedure. But the intuition that you saw on the previous slide is still correct in terms of how XGBoost is choosing examples to focus on. The details of XGBoost are quite complex to implement, which is why many practitioners will use the open source libraries that implement XGBoost.

This is all you need to do in order to use XGBoost, you will import the XGBoost library as follows and initialize a model as an XGBoost classifier. Further model and then finally, this allows you to make predictions using this boosted decision trees algorithm. I hope that you find this algorithm useful for many applications that you may build in the future. Or alternatively, if you want to use XGBoost for regression rather than for classification, then this line here just becomes XGBRegressor and the rest of the code works similarly.