PATRICK WINSTON: We've now almost completed our journey.

This will be it for talking about several kinds of learning-- the venerable kind, that's the nearest neighbors and identification tree types of learning.

Still useful, still the right thing to do if there's no reason not to do the simple thing.

Then we have the biologically-inspired approaches.

Neural nets.

All kinds of problems with local maxima and overfitting and oscillation, if you get the rate constant too big.

Genetic algorithms.

Like neural nets, both are very naive in their attempt to mimic nature.

So maybe they work on a class of problems.

They surely do each have a class of problems for which they're good.

But as a general purpose first resort, I don't recommend it.

But now the theorists have come out and done some things are very remarkable.

And in the end, you have to say, wow, these are such powerful ideas.

I wonder if nature has discovered them, too?

Is there good engineering in the brain, based on good science?

Or given the nature of evolution, is it just random junk that is the best ways for doing anything?

Who knows?

But today, we're going to talk about an idea that I'll bet is in there somewhere, because it's easy to implement, and it's extremely powerful in what it does, and it's the essential item in anybody's repertoire of learning mechanisms.

It's also a mechanism which, if you understand only by formula, you will never be able to work the problems on the quiz, that's for sure.

Because on the surface, it looks like it'd be very complicated to simulate this approach.

But once you understand how it works and look at a little bit of the math and let it sing songs to you, it turns out to be extremely easy.

So it's about letting multiple methods work in your behalf.

So far, we've been talking about using just one method to do something.

And what we're going to do now is we're looking to see if a crowd can be smarter than the individuals in the crowd.

But before we get too far down that abstract path, let me just say that the whole works has to do with classification, and binary classification.

Am I holding a piece of chalk in my hand, or a hand grenade?

Is that a cup of coffee or tea?

Those are binary classification problems.

And so we're going to be talking today strictly about binary classification.

We're not going to be talking about finding the right letter in the alphabet that's written on the page.

That's a 26-way choice.

We're talking about binary choices.

So we assume that there's a set of classifiers that we can draw on.

Here's one-- h.

And it produces either a minus 1 or a plus 1.

So that's how the classification is done.

If it's coffee, plus 1.

If it's tea, minus 1.

Is this chalk, plus one.

If it's a hand grenade, minus 1.

So that's how the classification works.

Now, too bad for us, normally the world doesn't give us very good classifiers.

So if we look at the error rate of this classifier or any other classifier, that error rate will range from 0 to 1 in terms of the fraction of the cases got wrong on a sample set.

So you'd like your error rate to be way down here.

You're dead if it's over there.

But what about in the middle?

What if it's, say, right there.

Just a little bit better than flipping a coin.

If it's just a little bit better than flipping a coin, that's a weak classifier.

And the question is, can you make a classifier that's way over here, like there, a strong classifier, by combining several of these weak classifiers, and letting them vote?

So how would you do that?

You might say, well, let us make a big classifier capital H, that works on some sample x, and has its output produces something that depends on the sum of the outputs of the individual classifiers.

So we have H1 working on x.

We have H2 working on x.

And we have H3 also working on x.

Let's say three of them, just to start us off.

And now let's add those guys up, and take the sign of the output.

So if two out of the three of those guys agree, then we'll get an either plus 1 or minus 1.

If all three agree, we'll get plus 1 or minus 1.

Because we're just taking the sign.

We're just taking the sign of the sum of these guys.

So this means that one guy can be wrong, as long as the other two guys are right.

But I think it's easier to see how this all works if you think of some space of samples, you say, well, let's let that area here be where H1 is wrong, and this area over here is where H2 is wrong.

And then this area over here is where H3 is wrong.

So if the situation is like that, then this formula always gives you the right answers on the samples.

I'm going to stop saying that right now, because I want to be kind of a background thing on the samples set.

We're talking about wrapping this stuff over the sample set.

Later on, we'll ask, OK, given that you trained this thing on a sample set, how well does it do on some new examples?

Because we want to ask ourselves about overfitting questions.

But for now, we just want to look and see if we believe that this arrangement, where each of these H's is producing plus 1 or minus 1, we're adding them up and taking the sign, is that going to give us a better result than the tests individually?

And if they look like this when draped over a sample set, then it's clear that we're going to get the right answer every time, because there's no area here where any two of those tests are giving us the wrong answer.

So the two that are getting the right answer, in this little circle here for H1, these other two are getting the right answer.

So they'll outvote it, and you'll get the right answer every time.

But it doesn't have to be that simple.

It could look like this.

There could be a situation where this is H1, wrong answer.

This is H2, wrong answer.

And this is H3, wrong answer.

And now the situation gets a little bit more murky, because we have to ask ourselves whether that area where three out of the three get it wrong is sufficiently big so as to be worse than 1 of the individual tests.

So if you look at that Venn diagram, and stare at it long enough, and try some things, you can say, well, there is no case where this will give a worse answer.

Or, you might end up with the conclusion that there are cases where we can arrange those circles such that the voting scheme will give an answer that's worst than an individual test, but I'm not going to tell you the answer, because I think we'll make that a quiz question.

Good idea?

OK.

So we'll make that a quiz question.

So that looks like a good idea.

And we can construct a little algorithm that will help us pick the particular weak classifiers to plug in here.

We've got a whole bag of classifiers.

We've got H1, we've got H2, we've got H55.

We've got a lot of them we can choose from.

So what we're going to do is we're going to use the data, undisturbed, to produce H1.

We're just going to try all the tests on the data and see which one gives us the smallest error rate.

And that's the good guy, so we're going to use that.

Then we're going to use the data with an exaggeration of H1 errors.

In other words-- this is a critical idea.

What we're going to do is we're going to run this algorithm again, but instead of just looking at the number of samples that are got wrong, what we're going to do is we're going to look at a distorted set of samples, where the ones we're not doing well on has exaggerated effect on the result.

So we're going to weight them or multiply them, or do something so that we're going to pay more attention to the

samples on which H1 produces an error, and that's going to give us H2.

And then we're going to do it one more time, because we've got three things to go with here in this particular little exploratory scheme.

And this time, we're going to have an exaggeration of those samples-- which samples are we going to exaggerate now?

We might as well look for the ones where H1 gives us a different answer from H2, because we want to be on the good guy's side.

So we can say we're going to exaggerate those samples four which H1 gives us a different result from H2.

And that's going to give us H3.

All right.

So we can think of this whole works here as part one of a multi-part idea.

So let's see.

I don't know, what might be step two?

Well, this is a good idea.

Then what we've got that we can easily derive from that is a little tree looked like this.

And we can say that H of x depends on H1, H2, and H3.

But now, if that that's a good idea, and that gives a better answer than any of the individual tests, maybe we can make this idea a little bit recursive, and say, well, maybe H1 is actually not an atomic test.

But maybe it's the vote of three other tests.

So you can make a tree structure that looks like this.

So this is H11, H12, H13, and then 3 here.

And then this will be H31, H32, H33.

And so that's a sort of get out the vote idea.

We're trying to get a whole bunch of individual tests into the act.

So I guess the reason this wasn't discovered until about '10 years ago was because you've got to get so many of these desks all lined up before the idea gets through that long filter of ideas.

So that's the only idea number two of quite a few.

Well, next thing we might think is, well, we keep talking about these classifiers.

What kind of classifiers are we talking about?

I've got-- oh, shoot, I've spent my last nickel.

I don't have a coin to flip.

But that's one classifier, right?

The trouble with that classifier is it's a weak classifier, because it gives me a 50/50 chance of being right.

I guess there are conditions in which a coin flip is better than a-- it is a weak classifier.

If the two outcomes are not equally probable, than a coin flip is a perfectly good weak classifier.

But what we're going to do is we're going to think in terms of a different set of classifiers.

And we're going to call them decision tree.

Now, you remember decision trees, right?

But we're not going to build decision trees.

We're going to use decision tree stumps.

So if we have a two-dimensional space that looks like this, then a decision tree stump is a single test.

It's not a complete tree that will divide up the samples into homogeneous groups.

It's just what you can do with one test.

So each possible test is a classifier.

How many tests do we get out of that?

12, right?

Yeah.

It doesn't look like 12 to me, either.

But here's how you get to 12.

One decision tree test you can stick in there would be that test right there.

And that would be a complete decision tree stump.

But, of course, you can also put in this one.

That would be another decision tree stump.

Now, for this one on the right, I could say, everything on the right is a minus.

Or, I could say, everything on the right is a plus.

It would happen to be wrong, but it's a valid test with a valid outcome.

So that's how we double the number of test that we have lines for.

And you know what?

can even have a kind of test out here that says everything is plus, or everything is wrong.

So for each dimension, the number of decision tree stumps is the number of lines I can put in times 2.

And then I've got two dimensions here, that's how I got to twelve.

So there are three lines.

I can have the pluses on either the left or the right side.

So that's six.

And then I've got two dimensions, so that gives me 12.

So that's the decision tree stump idea.

And here are the other decision tree boundaries, obviously just like that.

So that's one way can generate a batch of tests to try out with this idea of using a lot of tests to help you get the job done.

STUDENT: Couldn't you also have a decision tree on the right side?

PATRICK WINSTON: The question is, can you also have a test on the right side?

See, this is just a stand-in for saying, everything's plus or everything's minus.

So it doesn't matter where you put the line.

It can be on the right side, or the left side, or the bottom, or the top.

Or you don't have to put the line anywhere.

It's just an extra test, an additional to the ones you put between the samples.

So this whole idea of boosting, the main idea of the day.

Does it depend on using decision tree stumps?

The answer is no.

Do not be confused.

You can use boosting with any kind of classifier.

so why do I use decision tree stumps today?

Because it makes my life easy.

We can look at it, we can see what it's doing.

But we could put bunch of neural nets in there.

We could put a bunch of real decision trees in there.

We could put a bunch of nearest neighbor things in there.

The boosting idea doesn't care.

I just used these decision tree stumps because I and everybody else use them for illustration.

All right.

We're making progress.

Now, what's the error rate for any these tests and lines we drew?

Well, I guess it'll be the error rate is equal to the sum of 1 over n-- That's the total number of points, the number of samples-- summed over the cases where we are wrong.

So gee, we're going to work on combining some of these ideas.

And we've got this notion of exaggeration.

At some stage in what we're doing here, we're going to want to be able to exaggerate the effect of some errors relative to other errors.

So one thing we can do is we can assume, or we can stipulate, or we can assert that each of these samples has a weight associated with it.

That's W1, this is W2, and that's W3.

And in the beginning, there's no reason to suppose that any one of these is more or less important than any of the other.

So in the beginning, W sub i at time [? stub ?] one is equal to 1 over n.

So the error is just adding up the number of samples that were got wrong.

And that'll be the fraction of samples to that you didn't get right.

And that will be the error rate.

So what we want to do is we want to say, instead of using this as the error rate for all time, what we want to do is we want to move that over, and say that the error rate is equal to the sum over the things you got wrong in the current step, times the weights of those that were got wrong.

So in step one, everything's got the same weight, it doesn't matter.

But if we find a way to change their weights going downstream-- so as to, for example, highly exaggerate that third sample, then W3 will go up relative to W1 and W2.

The one thing we want to be sure of is there is no matter how we adjust the weights, that the sum of the weights over the whole space is equal to 1.

So in other words, we want to choose the weights so that they emphasize some of the samples, but we also want to put a constraint on the weights such that all of them added together is summing to one.

And we'll say that that enforces a distribution.

A distribution is a set of weights that sum to one.

Well, that's just a nice idea.

So we're make a little progress.

We've got this idea that we can add some plus/minus 1 classifiers together, you get a better classifier.

We got some idea about how to do that.

It occurs to us that maybe we want to get a lot of classifiers into the act somehow or another.

And maybe we want to think about using decision tree stumps so as to ground out thinking about all this stuff.

So the next step is to say, well, how actually should we combine this stuff?

And you will find, in the literature libraries, full of papers that do stuff like that.

And that was state of the art for quite a few years.

But then people began to say, well, maybe we can build up this classifier, H of x, in multiple steps and get a lot of classifiers into the act.

So maybe we can say that the classifier is the sign of H-- that's the one we picked first.

That's the classifier we picked first.

That's looking at samples.

And then we've got H2.

And then we've got H3.

And then we've got how many other classifiers we might want, or how many classifiers we might need in order to

correctly classify everything in our sample set.

So people began to think about whether there might be an algorithm that would develop a classifier that way, one step at a time.

That's why I put that step number in the exponent, because we're picking this one at first, then we're expanding it to have two, and then we're expanding it to have three, and so on.

And each of those individual classifiers are separately looking at the sample.

But of course, it would be natural to suppose that just adding things up wouldn't be enough.

And it's not.

So it isn't too hard to invent the next idea, which is to modify this thing just a little bit by doing what?

It looks almost like a scoring polynomial, doesn't it?

So what would we do to tart this up a little bit?

STUDENT: [INAUDIBLE].

PATRICK WINSTON: Come again?

Do what?

STUDENT: [INAUDIBLE].

PATRICK WINSTON: Somewhere out there someone's murmuring.

STUDENT: Add-- PATRICK WINSTON: Add weights!

STUDENT: --weights.

Yeah.

PATRICK WINSTON: Excellent.

Good idea.

So what we're going to do is we're going to have alphas associated with each of these classifiers, and we're going to determine if somebody can build that kind formula to do the job.

So maybe I ought to modify this gold star idea before I get too far downstream.

And we're not going to treat everybody in a crowd equally.

We're going to wait some of the opinions more than others.

And by the way, they're all going to make errors in different parts of the space.

So maybe it's not the wisdom of even a weighted crowd, but a crowd of experts.

Each of which is good at different parts of the space.

So anyhow, we've got this formula, and there are a few things that one can say turn out.

But first, let's write down the an algorithm for what this ought to look like.

Before I run out of space, I think I'll exploit the right hand board here, and put the overall algorithm right here.

So we're going to start out by letting of all the weights at time 1 be equal to 1 over n.

That's just saying that they're all equal in the beginning, and they're equal to 1 over n.

And n is the number of samples.

And then, when I've got that, I want to compute alpha, somehow.

Let's see.

No, I don't want to do that.

I want to I want to pick a classifier the minimizes the error rate.

And then m, i, zes, error at time t.

And that's going to be at time t.

And we're going to come back in here.

That's why we put a step index in there.

So once we've picked a classifier that produces an error rate, then we can use the error rate to determine the alpha.

So I want the alpha over here.

That'll be sort of a byproduct of picking that test.

And with all that stuff in hand, maybe that will be enough to calculate Wt plus 1.

So we're going to use that classifier that we just picked to get some revised weights, and then we're going to go around that loop until this classifier produces a perfect set of conclusions on all the sample data.

So that's going to be our overall strategy.

Maybe we've got, if we're going to number these things, that's the fourth big idea.

And this arrangement here is the fifth big idea.

Then we've got the sixth big idea.

And the sixth big idea says this.

Suppose that the weight on it ith sample at time t plus 1 is equal to the weight at time t on that same sample, divided by some normalizing factor, times e to the minus alpha at time t, times h at time t, times some function y which is a function of x, But not a function of time.

Now you say, where did this come from?

And the answer is, it did not spring from the heart of mathematician in the first 10 minutes that he looked at this problem.

In fact, when I asked [INAUDIBLE] how this worked, he said, well, he was thinking about this on the couch every Saturday for about a year, and his wife was getting pretty sore, but he finally found it and saved their marriage.

So where does stuff like this come from?

Really, it comes from knowing a lot of mathematics, and seeing a lot of situations, and knowing that something like this might be mathematically convenient.

Something like this might be mathematically convenient.

But we've got to back up a little and let it sing to us.

What's y?

We saw y last time.

The support vector machines.

That's just a function.

That's plus 1 or minus 1, depending on whether the output ought to be plus 1 or minus 1.

So if this guy is giving the correct answer, and the correct answer is plus, and then this guy will be plus 1 too, because it always gives you the correct answer.

So in that case, where this guy is giving the right answer, these will have the same sign, so that will be a plus 1 combination.

On the other hand, if that guy's giving the wrong answer, you're going to get a minus 1 out of that combination.

So it's true even if the right answer should be minus, right?

So if the right answer should be minus, and this is plus, then this will be minus 1, and the whole combination well give you minus 1 again.

In other words, the y just flips the sign if you've got the wrong answer, no matter whether the wrong answer is plus 1 or minus 1.

These alphas-- shoot, those are the same alphas that are in this formula up here, somehow.

And then that z, what's that for?

Well, if you just look at the previous weights, and its exponential function to produce these W's for the next generation, that's not going to be a distribution, because they won't sum up to 1.

So what this thing here, this z is, that's a sort of normalizer.

And that makes that whole combination of new weights add up to 1.

So it's whatever you got by adding up all those guys, and then dividing by that number.

Well, phew.

I don't know.

Now there's some it-turns-out-thats.

We're going to imagine that somebody's done the same sort of thing we did to the support vector machines.

We're going to find a way to minimize the error.

And the error we're going to minimize is the error produced by that whole thing up there in 4.

We're going to minimize the error of that entire expression as we go along.

And what we discover when we do the appropriate differentiations and stuff-- you know, that's what we do in calculus-- what we discover is that you get minimum error for the whole thing if alpha is equal to 1 minus the error rate at time t, divided by the error rate at time t.

Now let's take the logarithm of that, and multiply it by half.

And that's what [INAUDIBLE] was struggling to find.

But we haven't quite got it right.

And so let me add this in separate chunks, so we don't get confused about this.

It's a bound on that expression up there.

It's a bound on the error rate produced by that expression.

So interestingly enough, this means that the error rate can actually go up as you add terms to this formula.

all you know is that the error rate is going to be bounded by an exponentially decaying function.

So it's eventually guaranteed to converge on zero.

So it's a minimal error bound.

It turns out to be exponential.

Well, there it is.

We're done.

Would you like to see a demonstration?

Yeah, OK.

Because you look at that, and you say, well, how could anything like that possibly work?

And the answer is, surprisingly enough, here's what happens.

There's a simple little example.

So that's the first test chosen.

the greens are pluses and the reds are minuses, so it's still got an error.

Still got an error-- boom.

There, in two steps.

It now has-- we can look in the upper right hand corner-- we see its used three classifiers, and we see that one of those classifiers says that everybody belongs to a particular class, three different weights.

And the error rate has converged to 0.

So let's look at a couple of other ones.

Here is the one I use for debugging this thing.

We'll let that run.

See how fast it is?

Boom.

It converges to getting all the samples right very fast.

Here's another one.

This is one we gave on an exam a few years back.

First test.

Oh, I let it run, so it got everything instantaneously right.

Let's take that through step at a time.

There's the first one, second one.

Still got a lot of errors.

Ah, the error rate's dropping.

And then flattened, flattened, and it goes to 0.

Cool, don't you think?

But you say to me, bah, who cares about that stuff?

Let's try something more interesting.

There's one.

That was pretty fast, too.

Well, there's not too many samples here.

So we can try this.

So there's an array of pluses and minuses.

Boom.

You can see how that error rate is bounded by an exponential?

So in a bottom graph, you've got the number of classifiers involved, and that goes up to a total, eventually, of 10.

You can see how positive or negative each of the classifiers that's added is by looking at this particular tab.

And this just shows how they evolve over time.

But the progress thing here is the most interesting.

And now you say to me, well, how did the machine do that?

And it's all right here.

We use an alpha that looks like this.

And that allows us to compute the new weights.

It says we've got a preliminary calculation.

We've got to find a z that does the normalization.

And we sure better bring our calculator, because we've got, first of all, to calculate the error rate.

Then we've got to take its logarithm, divide by 2, plug it into that formula, take the exponent, and that gives us the new weight.

And that's how the program works.

And if you try that, I guarantee you will flunk the exam.

Now, I don't care about my computer.

I really don't.

It's a slave, and it can calculate these logarithm and exponentials till it turns blue, and I don't care.

Because I've got four cores or something, and who cares.

Might as well do this, than sit around just burning up heat.

But you don't want to do that.

So what you want to do is you want to know how to do this sort of thing more expeditiously.

So we're going to have to let them the math sing to us a little bit, with a view towards finding better ways of doing this sort of thing.

So let's do that.

And we're going to run out of space here before long, so let me reclaim as much of this board as I can.

So what I'm going to do is I'm going to say, well, now that we've got this formula for alpha that relates alpha t to the error, then I can plug that into this formula up here, number 6.

And what I'll get is that the weight of t plus 1 is equal to the weight at t divided by that normalizing factor, multiplied times something that depends on whether it's categorized correctly or not.

That's what that y's in their for, right?

So we've got a logarithm here, and we got a sign flipper up there in terms of that H of x and y combination.

So if the sign of that whole thing at minus alpha and that y H combination turns out to be negative, then we're going to have to flip the numerator and denominator here in this logarithm, right?

And oh, by the way, since we've got a half out here, that turns out to be the square root of that term inside the logarithm.

So when we carefully do that, what we discover is that it depends on whether it's the right thing or not.

But what it turns out to be is something like a multiplier of the square root.

Better be careful, here.

The square root of what?

STUDENT: [INAUDIBLE].

PATRICK WINSTON: Well, let's see.

But we have to be careful.

So let's suppose that this is 4 things that we get correct.

So if we get it correct, then we're going to get the same sign out of H of x and y.

We've get a minus sign out there, so we're going to flip the numerator and denominator.

So we're going to get the square root of e of t over 1 minus epsilon of t if that's correct.

If it's wrong, it'll just be the flip of that.

So it'll be the square root of 1 minus the error rate over the error rate.

Everybody with me on that?

I think that's right.

If it's wrong, I'll have to hang myself and wear a paper bag over my head like I did last year.

But let's see if we can make this go correctly this time.

So now, we've got this guy here, we've got everything plugged in all right, and we know that now this z ought to be selected so that it's equal to the sum of this guy multiplied by these things as appropriate for whether it's correct or

not.

Because we want, in the end, for all of these w's to add up to 1.

So let's see what they add up to without the z there.

So what we know is that it must be the case that if we add over the correct ones, we get the square root of the error rate over 1 minus the rate of the Wt plus 1.

Plus now we've got the sum of 1 minus the error rate over the error rate, times the sum of the Wi at time t for wrong.

So that's what we get if we added all these up without the z.

So since everything has to add up to 1, then z ought to be equal to this sum.

That looks pretty horrible, until we realize that if we add these guys up over the weights that are wrong, that is the error rate.

This is e.

So therefore, z is equal the square root of the error rate times 1 minus the error rate.

That's the contribution of this term.

Now, let's see.

What is the sum of the weights over the ones that are correct?

Well, that must be 1 minus the error rate.

Ah, so this thing gives you the same result as this one.

So z is equal to 2 times that.

And that's a good thing.

Now we are getting somewhere.

Because now, it becomes a little bit easier to write some things down.

Well, we're way past this, so let's get rid of this.

And now we can put some things together.

Let me point out what I'm putting together.

I've got an expression for z right here.

And I've got an expression for the new w's here.

So let's put those together and say that w of t plus 1 is equal to w of t.

I guess we're going to divide that by 2.

And then we've got this square root times that expression.

So if we take that correct one, and divide by that one, then the [INAUDIBLE] cancel out, and I get 1 over 1 minus the error rate.

That's it.

That's correct.

And if it's not correct, then it's Wt over 2-- and working through the math-- 1 over epsilon, if wrong.

Do we feel like we're making any progress?

No.

Because we haven't let it sing to us enough yet.

So I want to draw your attention to what happens to amateur rock climbers when they're halfway up a difficult cliff.

They're usually [INAUDIBLE], sometimes they're not.

If they're not, they're scared to death.

And every once in a while, as they're just about to fall, they find some little tiny hole to stick a fingernail in, and that keeps them from falling.

That's called a thank-god hole.

So what I'm about to introduce is the analog of those little places where you can stick your fingernail in.

It's the thank-god hole for dealing with boosting problems.

So what happens if I add all these [? Wi ?] up for the ones that the classifier where produces a correct answer on?

Well, it'll be 1 over 2, and 1 over 1 minus epsilon, times the sum of the Wt for which the answer was correct.

What's this sum?

Oh!

My goddess.

1 minus epsilon.

So what I've just discovered is that if I sum new w's over those samples for which I got a correct answer, it's equal to 1/2.

And guess what?

That means that if I sum them over wrong, it's equal to 1/2 half as well.

So that means that I take all of the weight for which I got the right answer with the previous test, and those ways will add up to something.

And to get the weights for the next generation, all I have to do is scale them so that they equal half.

This was not noticed by the people who developed this stuff.

This was noticed by Luis Ortiz, who was a 6.034 instructor a few years ago.

The sum of those weights is going to be a scaled version of what they were before.

So you take all the weights for which this new classifier-- this one you selected to give you the minimum weight on the re-weighted stuff-- you take the ones that it gives a correct answer for, and you take all of those weights, and you just scale them so they add up to 1/2.

So do you have to compute any logarithms?

No.

Do you have to compute any exponentials?

No.

Do you have to calculate z?

No.

Do you have to calculate alpha to get the new weights?

No.

All you have to do is scale them.

And that's a pretty good thank-god hole.

So that's thank-god hole number one.

Now, for thank-god hole number two, we need to go back and think about the fact that were going to give you problems in probability that involve decision tree stumps.

And there are a lot of decision tree stumps that you might have to pick from.

So we need a thank-god hole for deciding how to deal with that.

Where can I find some room?

How about right here.

Suppose you've got a space that looks like this.

I'm just makings this up at random.

So how many-- let's see.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

How many tests do I have to consider in that dimension?

11.

It's 1 plus the number of samples.

That would be horrible.

I don't know.

Do I have actually calculate this one?

How could that possibly be better than that one?

It's got one more thing wrong.

So that one makes sense.

The other one doesn't make sense.

So in the end, no test that lies between two correctly classified samples will ever be any good.

So that one's a good guy, and that one's a good guy.

And this one's a bad guy.

Bad guy, bad guy bad guy, bad guy.

Bad guy, bad guy, bad buy.

So the actual number of tests you've got is three.

And likewise, in the other dimension-- well, I haven't drawn it so well here, but would this test be a good one?

No.

That one?

No.

Actually, I'd better look over here on the right and see what I've got before I draw too many conclusions.

Let's look over this, since I don't want to think too hard about what's going on in the other dimension.

But the idea is that very few of those tests actually matter.

Now, you say to me, there's one last thing.

What about overfitting?

Because all this does is drape a solution over the samples.

And like support vector machines overfit, neural maps overfit, identification trees overfit.

Guess what?

This doesn't seem to overfit.

That's an experimental result for which the literature is confused.

It goes back to providing an explanation.

So this stuff is tried on all sorts of problems, like handwriting recognition, understanding speech, all sorts of stuff uses boosting.

And unlike other methods, for some reason as yet imperfectly understood, it doesn't seem to overfit.

But in the end, they leave no stone unturned in 6.034.

Every time we do this, we do some additional experiments.

So here's a sample that I'll leave you with.

Here's a situation in which we have a 10-dimensional space.

We've made a fake distribution, and then we put in that boxed outlier.

That was just put into the space at random, so it can be viewed as an error point.

So now what we're going to do is we're going to see what happens when we run that guy.

And sure enough, in 17 steps, it finds a solution.

But maybe it's overfit that little guy who's an error.

But one thing you can do is you can say, well, all of these classifiers are dividing this space up into chunks, and we can compute the size of the space occupied by any sample.

So one thing we can do-- alas, I'll have to get up a new demonstration.

One thing we can do, now that this guy's over here, we can switch the volume tab and watch how the volume occupied by that error point evolves as we solve the problem.

So look what happens.

This is, of course, randomly generated.

I'm counting on this working.

Never failed before.

So it originally starts out as occupying 26% of the total volume.

It ends up occupying 1.4 times 10 to the minus 3rd% of the volume.

So what tends to happen is that these decision tree stumps tend to wrap themselves so tightly around the error points, there's no room for overfitting, because nothing else will fit in that same volume.

So that's why I think that this thing tends to produce solutions which don't overfit.

So in conclusion, this is magic.

You always want to use it.

It'll work with any kind of [? speed ?] of classifiers you want.

And you should understand it very thoroughly, because of anything is useful in the subject in dimension learning, this is it.