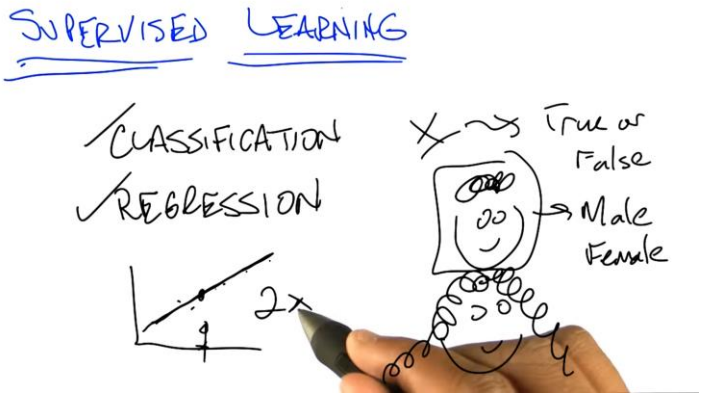
目录

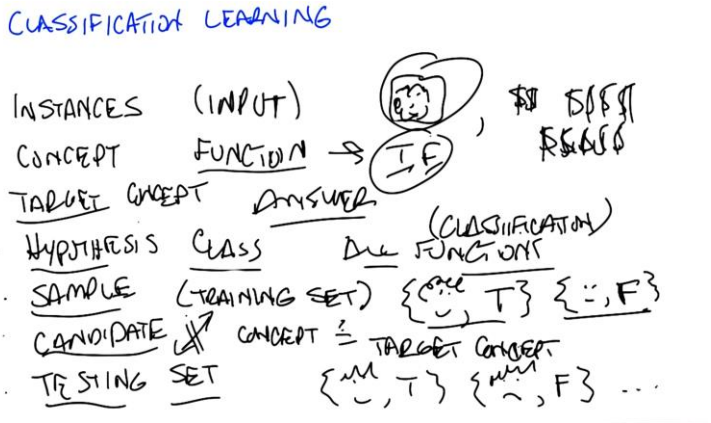
1. Decision Trees

1.1. Difference between Classification and Regression



- Two types of supervised learning that we typically think about are classification and regression.
- Classification is simply the process of taking some kind of input, let's call it x and mapping it to some discrete label, usually, for what we're talking about, something like, true or false.
- Regression is more about continuous value function. So, something like giving a bunch of points. I want to give in a new point. I want to map it to some real value. So we may pretend that these are examples of a line and so given a point here, I might say the output is right there.
- For the purposes of the sort of things that we're going to be worried about you can really think of the difference between classification and regression is the difference between mapping from some input to some small number of discrete values which might represent concepts.
- And regression is mapping from some input space to some real number. Potentially infinite number of real numbers.
- The difference between a classification task or a regression task is not about the input, it's about the output.
- If the output is continuous then it's regression and if the output is small discrete set or discrete set, then it is a classification task.

1.2. Classification Learning



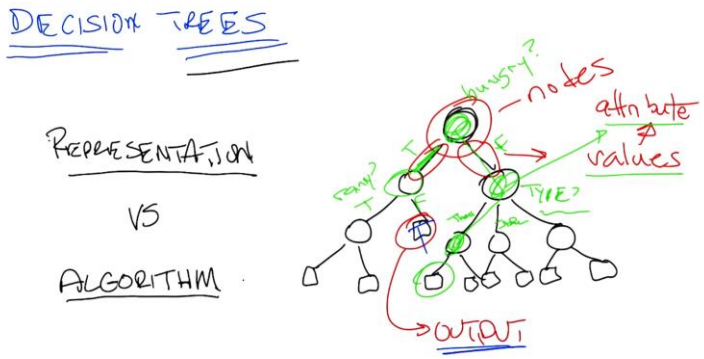
I want to define a couple of terms.

- **Instances**, or another way of thinking about them is input. Those are vectors of values, of attributes. That define whatever your input space is. So they can be pictures and all the pixels

that make up pictures like we've been doing so far before. They can be credit score examples like how much money I make. So whatever your input value is, whatever it is you're using to describe the input, whether it's pixels or its discrete values, those are your instances, the set of things that you're looking at.

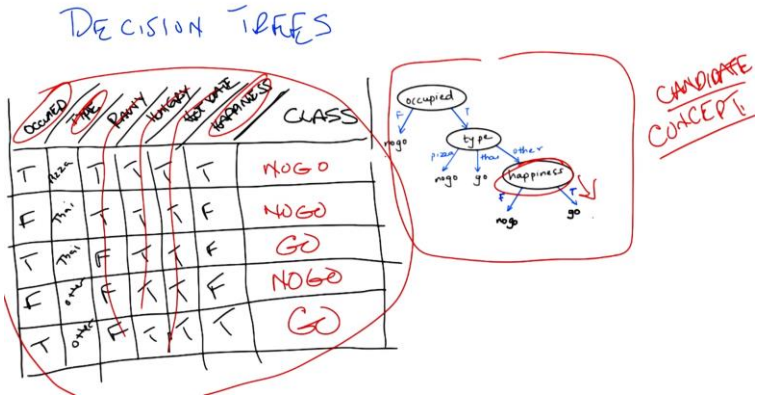
- And then what we're trying to find is some kind of function and that is the **concept** that we care about. And this function actually maps inputs to outputs, so it takes the instances, it maps them in this case to some kind of output such as true or false. This is the, the categories of things that we're worried about. The concept is the function that we care about that's going to map pictures, for example true or false. Intuitively a concept is an idea describes a set of things. OK, so we can talk about maleness or femaleness. We can talk about short and tall; we can talk about college students and grad students. And so the concept is the notion of what creditworthiness is, what a male is, what a college student is. A concept is, therefore by definition, a mapping between objects in a world and membership in a set, which makes it a function.
- So with that notion of a concept as functions or as mappings from objects to membership in a set we have a particular **target concept**. The only difference between a target concept and the general notion of concept is that the target concept is the thing we're trying to find. It's the actual answer. So, a function that determines whether something is a car or not, or male or not, is the target concept.
- **Hypothesis class** is the set of all concepts that you're willing to entertain. So it's all the functions I'm willing to think about. It could be all possible functions and that's a perfectly reasonable hypothesis class. You can think of hypothesis class as simply being all possible functions in the world. On the other hand, even so far just the classification learning, we already know that we're restricting ourselves to a subset of all possible functions in the world, because all possible functions in the world includes things like x squared, and that's regression. And we've already said, we're not doing regression. So already hypothesis classes all functions we care about and maybe it's all classification functions.
- A **training set** is a set of all of our inputs, like pictures of people, paired with a label, which is the correct output. If you get bunches and bunches of examples of input and output pairs, that's a training set. And that's what's going to be the basis for you figuring out what is the correct concept or function.
- A **candidate** is just simply a concept that you think might be the target concept.
- Given that you have a bunch of examples, and you have a particular candidate or a candidate concept, how do you know whether you are right or wrong? How do you know whether it's a good candidate or not? And that's where the **testing set** comes in. So a testing set looks just like a training set. So here our training set, we'll have pictures and whether someone turns out to be creditworthy or not. And I will take your candidate concept and I'll determine whether it does a good job or not, by looking at the testing set.
- An important point is that the training set and the testing set should not be the same. If you learn from your training set, and I test you only on your training set, then that's considered cheating in the machine learning world. Because then you haven't really shown the ability to generalize. So typically we want the testing set set to include lots of examples that you don't see in your training set. And that is proof that you're able to generalize.

1.3. Representation

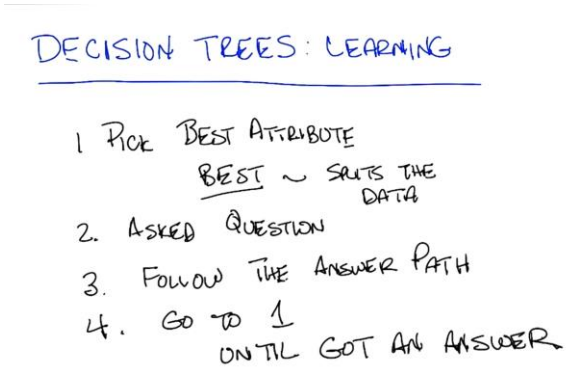


- So, that gets us to decision trees.
- So, the first thing, that, that we want to do is, we want to separate out what a decision tree is from the algorithm that you might use to build the decision tree. S
- So we have these nodes which represent attributes, and we have edges which represent value.
- So it's as if I'm making a bunch of decisions by asking a series of questions.

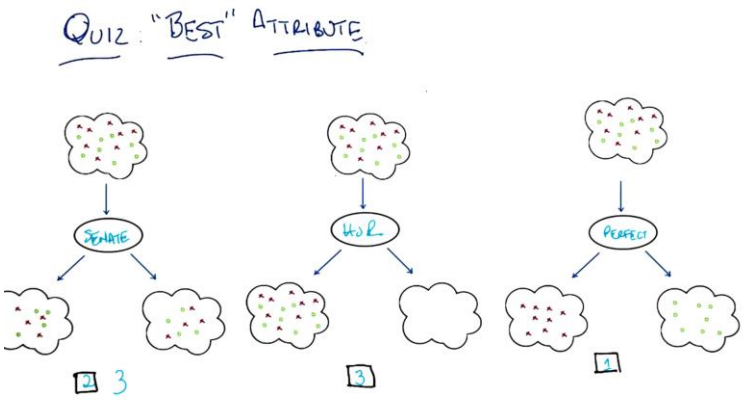
1.4. Quiz: Representation



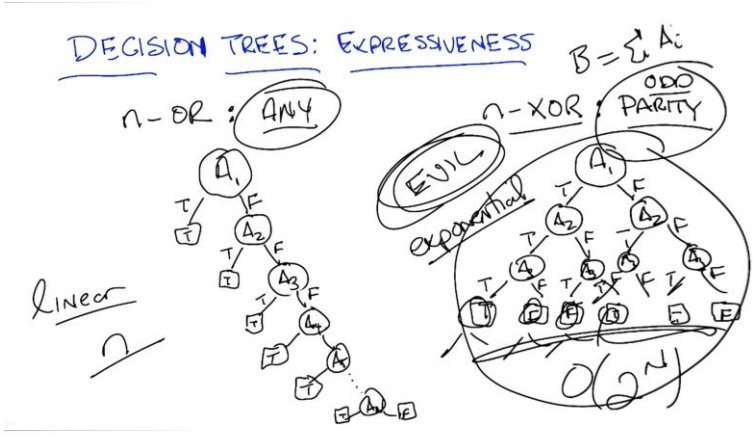
1.5. Decision Trees Learning

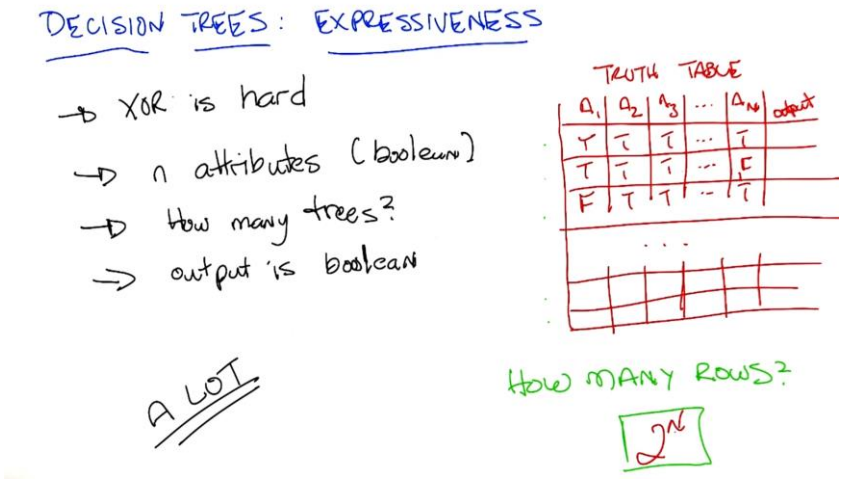


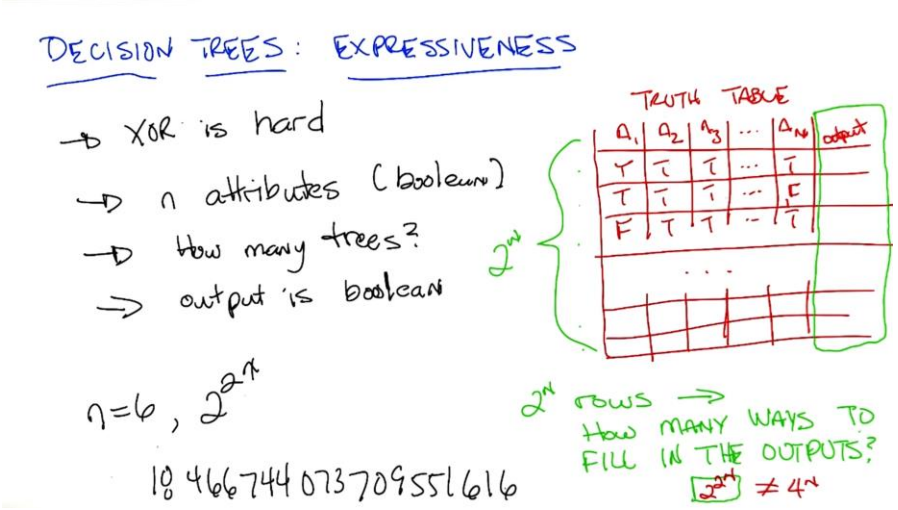
1.6. Quiz: Best Attribute



1.7. Decision Trees Expressiveness

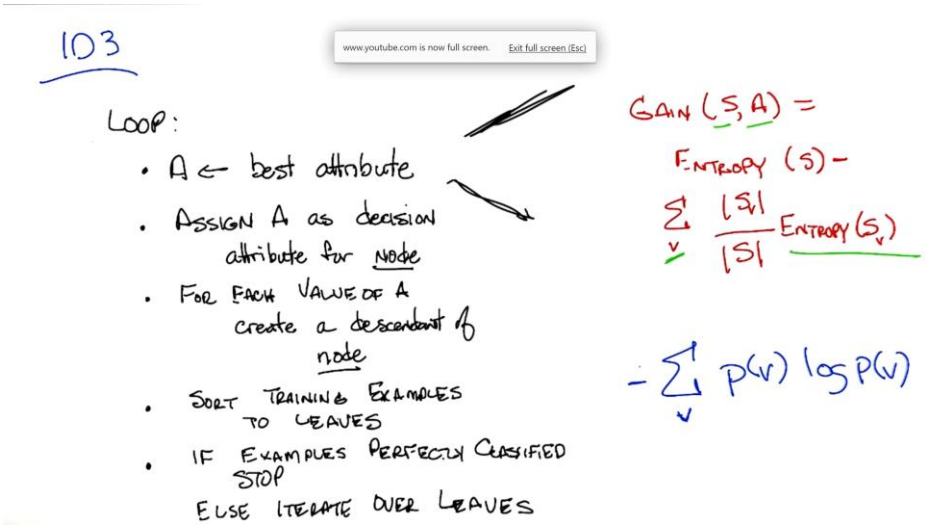






- XOR is an exponential problem and is also known as hard.
- Whereas OR, at least in terms of space that you need, it's a relatively easy one.
- This is linear.
- We have another name for exponential and that is evil. **Evil, evil, evil**. And it's evil because it's a very difficult problem. There is no clever way to pick the right attributes in order to give you an answer. You have to look at every single thing. That's what make this kind of problem difficult. So, just as a general point, Michael, I want to make, is that we hope that in our machine learning problems we're looking at things that are more like any than we are looking at things that are more like parity because otherwise, we're going to need to ask a lot of questions in order to answer the parity questions.
- So 2 to the n grows very fast. We already called that evil.
- 2 to the 2 to the n is a double exponential and it's super evil. It grows very, very, very, very, very fast.
- This means we have to have some clever way to search among them. And that gets us back to our notion of an algorithm with actually going to very smartly go through and pick out which decision tree. Because if we aren't very smart about it and we start eliminating whole decision trees along the way. Then we're going to have to look it to billions upon, billions upon, billions upon, billions upon, billions of possible decision choice.

1.8. ID3



- Here's the ID3 algorithm.
  - You're simply going to keep looping forever until you've solved the problem.
  - At each step, you're going to pick the best attribute, and we're going to define what we mean by best. There are a couple of different ways we might define best.
  - And then, given the best attribute that splits the data the way that we want, it does all those things that we talked about, assign that as a decision attribute for node.
  - And then for each value that the attribute A can take on, create a descendent of node.
  - Sort the training examples to those leaves based upon exactly what values they take on
  - If you've perfectly classified your training set, then you stop. Otherwise, you iterate over each of those leaves, picking the best attribute in turn for the training examples that were sorted into that leaf, and you keep doing that, building up the tree until you're done.
- So that's the ID3 algorithm.
- The key bit that we have to expand upon in this case, is exactly what it means to have a best attribute.
- There are lots of possibilities that you can come up with. The one that is most common s what's called **information gain**
- Information gain is simply a mathematical way to capture the amount of information that one gains by picking particular attribute. But what it really talks about is the reduction in the randomness, over the labels that you have with set of data, based upon the knowing the value of particular attribute.
- The formula's simply this:

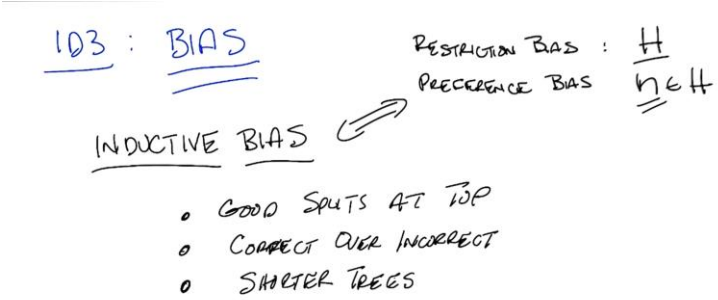
$$Gain(S, A) = Entropy(S) - \sum_v \frac{|S_v|}{|S|} Entropy(S_v)$$

  - The information gain over S and A where S is the collection of training examples that you're looking at and A as a particular attribute is simply defined as the entropy, with respect to the labels, of the set of training examples you have (S) minus the expected or average entropy that you would have over each set of examples that you have with a particular value.
  - Entropy is exactly a measure of randomness. So if I have a coin, let's say a two-headed coin. It can be heads or tails, and I don't know anything about the coin except that it's probably fair. If I were to flip the coin, what's the probability that it would end up heads or tails? M: A

half. C: It's a half, exactly, if it's a fair coin it's a half. Which means that I have no basis, going into flipping the coin, to guess either way whether it's heads or it's tails. And so that has a lot of entropy. In fact it has exactly what's called one bit of entropy. On the other hand, let's imagine that I have a coin that has heads on both sides. Then, before I even flip the coin, I already know what the outcome's going to be. It's going to come up heads. So what's the probability of it coming up with heads? M: It's one. C: So that actually has no information, no randomness, no entropy whatsoever. And has zero bits of entropy.

- The formula for entropy, using the same notation that we're using for information gain is simply the sum, over all the possible values that you might see, of the probability of you seeing that value, times the log of the probability of you seeing that value, times minus one. Further details in randomized optimization lecture where entropy's going to matter a lot.
- So the goal is to maximize over the entropy gain. And that's the best attribute.

1.9. ID3 Bias

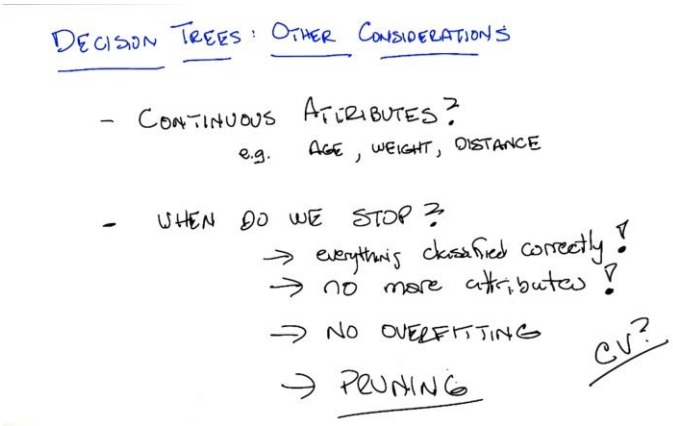


- There are two kinds of biases we worry about when we think about algorithms that are searching through space.
- One is what's called a **restriction bias**, which is nothing more than the hypothesis set that you actually care about. So in this case, with the decision trees, the hypothesis set is all possible decision trees. That means we're not considering, y equals 2x plus non-boolean functions of a certain type. We're only considering decision trees, and all that they can represent. And nothing else. Instead of looking at the uncountably infinite number of functions that are out there, we're only going to consider those that can be represented by a decision tree over all the cases we've given so far discrete variable.
- The other is called **preference bias**, which tells us what source of hypotheses from this hypothesis set we prefer, and that is really at the heart of inductive bias.
- Which decision trees would ID3 prefer?
    - It's going to be more likely to produce a tree that has basically good splits near the top than a tree that has bad splits at the top. Even if the two trees can represent the same function. Given two decision trees that are both correct. They both represent the function that we might care about. It would prefer the one that had the better split near the top.
    - It prefers ones that model the data better to ones that model the data worse. It prefers correct ones to incorrect ones. Given a tree that has very good splits at the top but produces

the wrong answer. It will not take that one over one that doesn't have as good splits at the top, but does give you the correct answer.

- Those are really the two main things that are the inductive bias for ID3. Although, when you put those two together, in particular when you look at the first one, there's sort of a third one that comes out as well, which is ID3 algorithm tends to prefer shorter trees to longer trees. Now, that preference for shorter trees actually comes naturally from the fact that you're doing good splits at the top. Because you're going to take trees that actually separate the data well by labels, you're going to tend to come to the answer faster than you would if you didn't do that.

1.10. Decision Trees Other Considerations

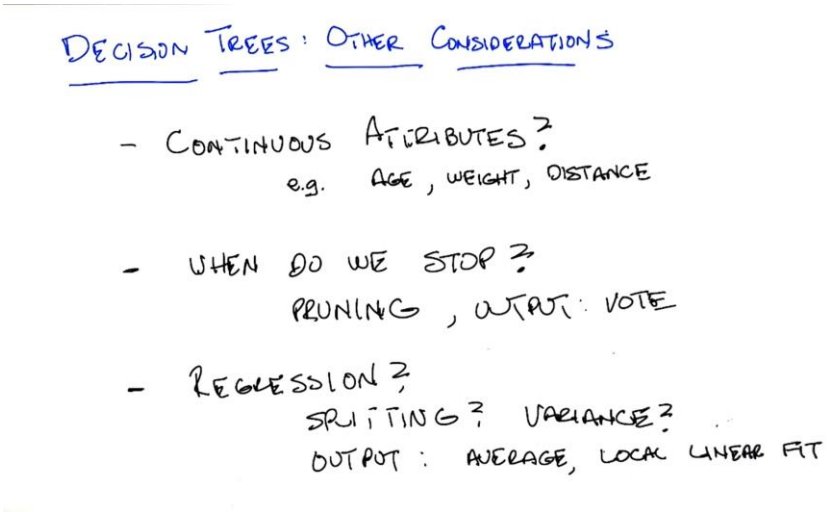


- When do we really stop?
- So the answer in the algorithm is when everything is classified correctly. But what if we have noise in our data? Then our algorithm goes into an infinite loop. So we could just say when we've run out of attributes. Although that doesn't help us in the case where we have continuous attributes and we might ask an infinite number of questions. So we probably need a slightly better criteria.
- If it's possible for us to have a little bit of noise in the data, an error here or there, then we want to have some way to deal to handle that possibility
- When you get really good at classifying your training data, but it doesn't help you to generalize, we have a name for that: **overfitting**.
- Now the way you overfit in a decision tree is basically by having a tree that's too big, it's too complicated. Violates Occam's Razor.
- You could take out a validation set. You build a decision tree, and you test it on the validation set and you pick whichever one has the lowest error in the validation set, that's one way to avoid it. There is another way you can do it that's more efficient.
- You do the same idea validation, except that you hold out a set and every time you decide whether to expand the tree or not, you check to see how this would do so far in the validation set. And if the error is low enough, then you stop expanding the tree. That's one way of doing it.
- You could do pruning. You could go ahead and do the tree as if you didn't have to worry about over-fitting, and once you have the full tree built, you could then do a kind of, you could do pruning. You could go to the leaves of the tree and say, well, what if I collapse these leaves back up into the tree? How does that create error on my validation set? And if the error is too big,

then you don't do it. And if it's very small, then you go ahead and do it. And that should help you with overfitting.

- There's a whole bunch of different ways you might prune. But pruning, itself, is one way of dealing with overfitting, and giving you a smaller tree. And it's a very simple addition to the standard ID3 algorithm.

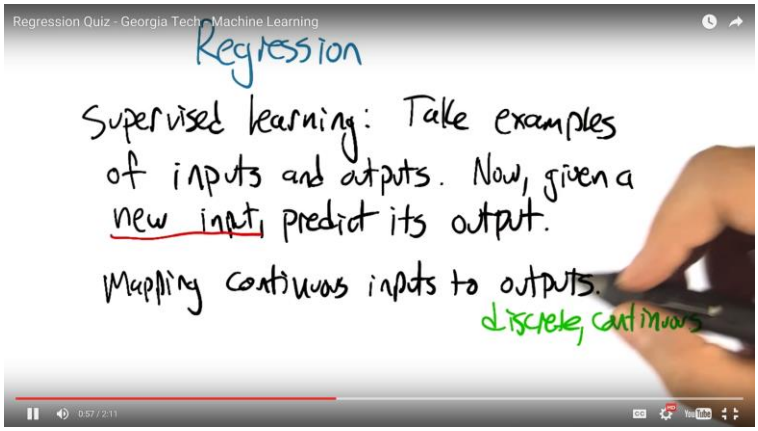
1.11. Decision Trees Other Considerations Regression



- Another consideration is the problem of regression.
- So far we've only been doing classification where the outputs are discrete, but what if we were trying to solve something that looked more like x squared or two x plus 17 or some other continuous function. In other words, a regression problem. How would we have to adapt decision trees to do that?
- What you really have now is a question about splitting. What's the splitting criteria?
- There's also an issue of what you do in the leaves. You could do some sort of more standard fitting algorithm. So report the average or do some kind of a linear fit. There’s any number of things you can do.
- Errors, how you would report an output? If you don't have a clear answer where everything is labeled true or everything is labeled false, how do you pick? So something like an average would work there. If we're trying to get as many right answers as we can, then you probably want to do a vote in the leaves.

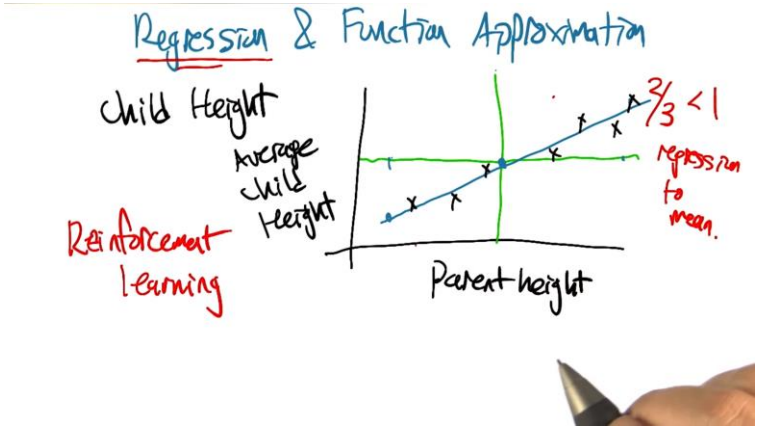
2. Regression and Classification

2.1. What is Regression?

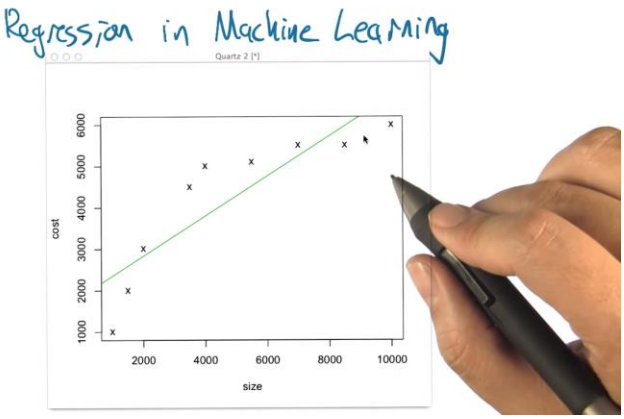


- In supervised learning we can take examples of inputs and outputs and based on that we are going to be able to take a new input and predict the corresponding output for that input

2.2. Regression and Function Approximation

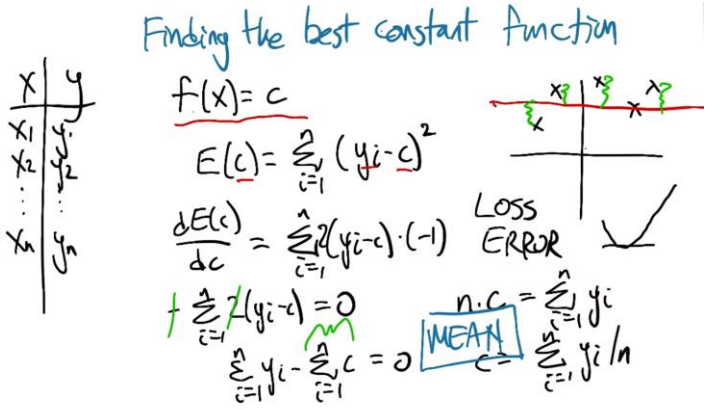


- I'm going to draw a graph and on this axis will be the parent height. And on this axis will be the average child height.
- If we plot these against each other, let's me put the mean up here. Let's say that this is mean height for the population. And now say that you know pair, we sort of imagine that parents of average height will have children of average height. But parents that are really tall, like that hypothetical person from before, will have children that are taller than average but not as tall as themselves. And similarly people that are very let's say of smaller stature will have children that are also you know short. And, but not quite as short again closer to the mean.
- It turns out that you have this, this very nice linear relationship between these quantities, and, there's an important aspect to this. Which is that the slope of this line is less than one, it's two thirds.
    - If the slope of this line was one it would mean that everybody's children would height of their parents.
    - If this slope is less than one, like it is, it turns out to be in real populations.
- And that's the fact that this is less than one is what makes it regression to the mean.
- Regression now refers to not this idea of collapsing back towards the mean, but the idea of using functional form to approximate a bunch of data points.



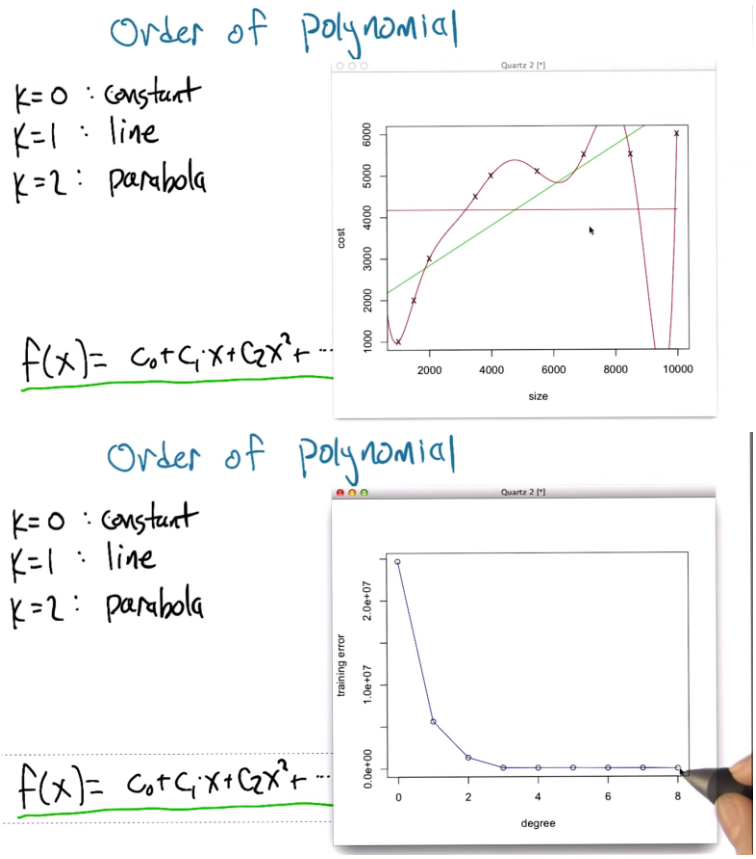
- One of the things that's very helpful about regression is that in many ways it's very simple to visualize, it's very simple to think about what some of the issues are and all the various topics in machine learning that are really important to understand and sometimes are difficult concepts really do come up in a fairly easy to understand way.
- This graph that I put up here, is supposed to tell us a little bit about housing prices.
- Let's imagine that we're off to buy a house and what we notice is that there's lots of different houses on the market, and there are lots of different, sizes.
- The square footage of the house can vary. In this case the houses that I visited can be between about 1,000 to 10,000 square feet. As you get bigger houses, the prices tend to go up too. The price that the house cost tends to rise with the size of the house.
- What I've done here is I've plotted as a little x say a set of nine houses that I've observed.
- Start off over here with a house that's a 1,000 square feet and cost a $1,000 and we end up with a house that is 10,000 square feet and cost about $6,000.
- Now we want to answer a question about what happens If we find a house on the market and it's about $5,000, what do you think a fair price for that would be?
- Apparently about $5,000. But there was no corresponding point for that, so you had to interpolate based on the points that were there you had to kind of imagine what might, happening at the 5,000 square foot mark
- What we're going to do in this case is actually try to find a function that fits this.
- What would be the best linear function that captures the relationship between the size and the cost? The green plot line turns out to be the one that minimizes the squared error, the squared deviation, between these x points and the corresponding position on green line.
    - So it finds a way of balancing all those different errors against each other and that's the best line we've got.
    - Now in this particular case, this line predicts something more like $4,000. It doesn't really look like a very good fit. But it does at least capture the fact that there is increasing cost with, with increase in size.

2.3. Find the Best Fit Quiz



- Define best fit: the one that has the least squared error. The error is the sum of the distances between these x points and the green line that we fit. We can use calculus to do this
- Imagine that what we're trying to do is that we've got a bunch of data points, and we're trying to find the best constant function. The best function that has the form, the value of the function for any given X is always the same constant, C. $f(x) = c$
- The error is going to be the sum over all of the data points. The square difference between that constant we chose and what the actual y value is. $E(c) = \sum_{i=1}^{n} (y_i - c)^2$
- There are many different error functions and there are lots of different ones that could work (i.e. absolute error, squared error, various kinds of squashed errors)
  - It turns out that this one is particularly well behaved because this error function is smooth as a function of the constant c, we can use calculus to actually find the minimum error value.
- Using the chain rule, if you want to find how this error function outputs change as a function of input c, we can take the derivative of this sum. $\frac{dE(c)}{dc} = \sum_{i=1}^{n} 2(y_i - c) * (-1)$
- Now this gives us a nice, smooth function saying what the error is as a function of c. If we want to find the minimum, we set it equal to zero $-\sum_{i=1}^{n} 2(y_i - c) = 0$
- Now we just need to solve this equation for c. $\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} c) = 0 \rightarrow n * c = \sum_{i=1}^{n} y_i \rightarrow c = \frac{\sum_{i=1}^{n} y_i}{n}$=mean
- This results in the **mean**. The best constant is the average of all your y's, the mean.
- So in the case, we just have to average the y's together and that gets us the thing that minimizes the squared error. So squared error is this really nice thing because it tends to bring things like mean back into the picture. It's really very convenient. And, it generalizes to higher order functions.

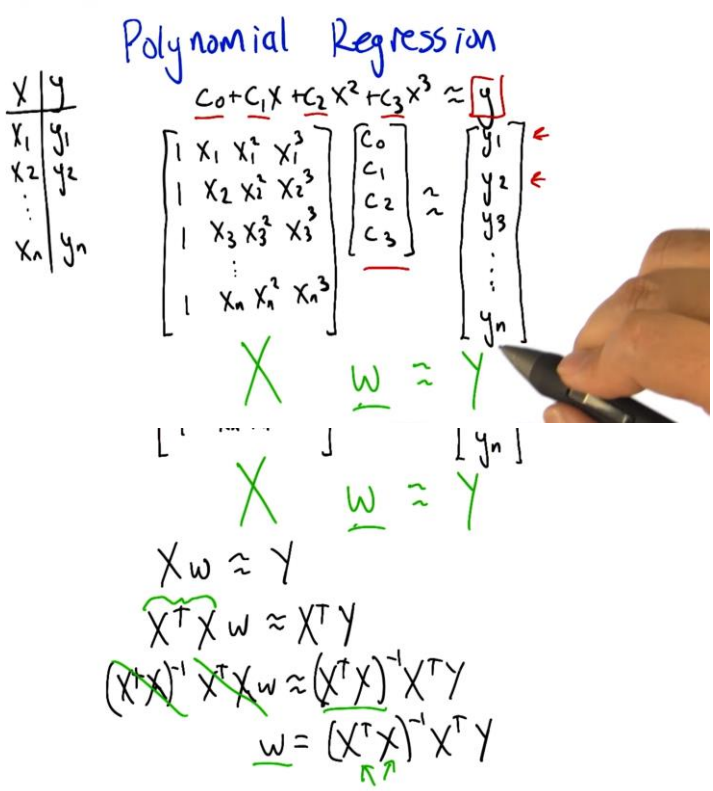
2.4. Order of Polynomial



- The ideas that we're going to try to find a way of predicting the value for various points along the way on this curve.
- One thing we could do is find the best line. But we also talked now about finding the best constant. Turns out these all belong to a family of functions that we could fit. Which are functions of this form.
- We've got x is our input and what we're going to do is we're going to take some constant and add that to some scaled version of x times some scaled version of x squared plus some scaled version of x cubed, all the way up to some order k.
- And we've talked about k equals zero, the constant function. And k equals one, the line. But there's also k equals two, parabola.
- It's going up and it's kind of flattening out and maybe we could imagine that it starts coming down again? At least, over the course of these points, it doesn't come down again but at least it sort of flattened out.
- We've got the best line now, the best constant function which is just the average. We have the best line with some slope to it. That's the green one. We have now the best parabola, kind of gets tucked in with all those other points.
- If the only thing we care about is minimizing the sum of squared error, the parabola has less squared error.
- There's more degrees of freedom so at the worst we could have just fit the parabola as a line. Right, we can always just set any of these coefficients to zero.
- So if the best fit to this really was a line then the parabola that we see here wouldn't have any curve to it.

- We have gone from order zero to order one to order two. How about order nine? Order six is in fact the best we can do here. The highest order that works is order eight. And look what it did. It hit every single point dead on in the center. It used all the degrees of freedom it had to reduce the error to essentially zero.
    - One could argue that this is a really good idea. Though, if you look at what happens around 9000, there's some craziness. To try to get this particular parabola to hit that particular point, it sent the curve soaring down with an up again.
- To show that as we have more degrees of freedom we're fitting the error better, see what the amount of error for the best fit for each of these orders of k.
- Alright and so, so what you see when we actually plot the, the squared error, this function that we're trying to minimize. As we go from order zero to order one, order two, order three, order four, order five, all the way to eight. By eight, there is no error left because it nailed every single point. So you know it's kind of a good, but it doesn't feel quite right like the curves that we're looking there looked a little bit crazy.

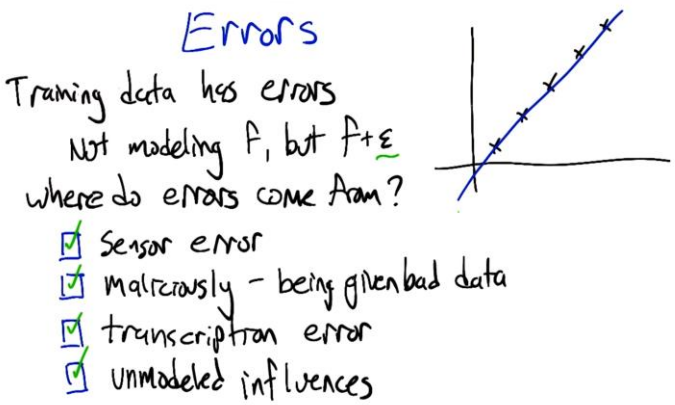
2.5. Polynomial Regression



- We can arrange all these constraints, all these equations into matrix form. If you're familiar with linear algebra.
- So if we arrange all these x values into a matrix, and then we have these other guys, we'll call this w, like the coefficients. Obviously w stands for coefficient. And we want that to equal this vector of y's. $Xw \approx Y$
- Basically just need to solve this equation for the w's. We can't exactly solve it because it's not going to exactly equal, but we can solve it in a least squares sense.

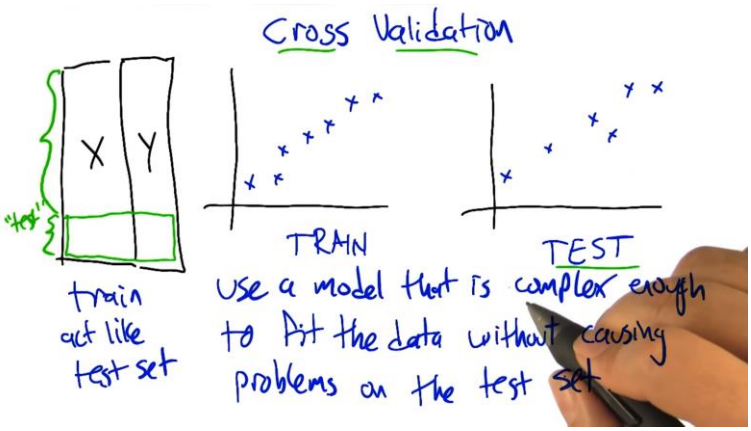
- Let's just step through the steps for how we're going to solve for w.
  - Premultiply both sides by the transpose of X $X^T Xw \approx X^T Y$
    - $X^T X$ is going to have a nice inverse - So now we can premultiply by that inverse
  - Now, conveniently since this has a nice inverse, the inverses cancel each other:
    $(X^T X)^{-1} X^T Xw \approx (X^T X)^{-1} X^T Y \rightarrow \quad w \approx (X^T X)^{-1} X^T Y$
  - w is the weights we're looking for, gives us exactly the coefficients that we need

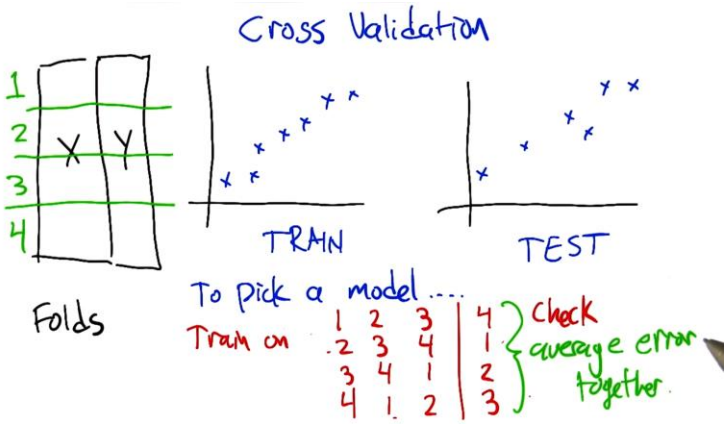
2.6. Errors Quiz



- Part of the reason we can't just solve these kinds of problems by solving a system of linear equations and just being done with it has to do with these squares is because of the presence of errors.
- The training data that we are given has errors in it.
- And it's not that we're actually modeling a function, but the thing that we're seeing is the function plus some error term on each piece of data.

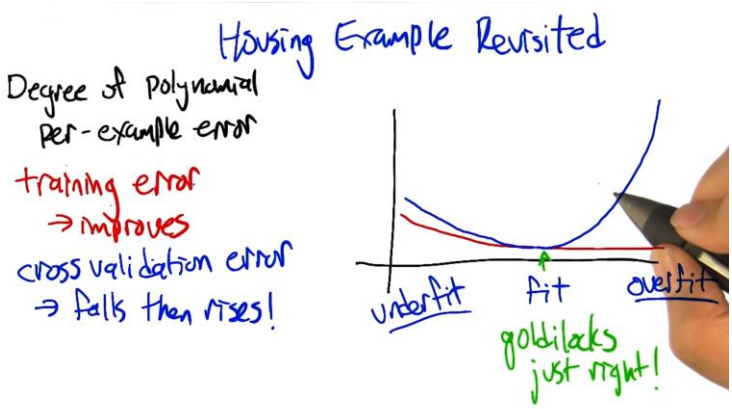
2.7. Cross Validation





- Imagine that we've got our data, this is our training set. Ultimately what we're trying to do is find a way of predicting values and then testing them.
- We do some kind of regression and we might want to fit this to a line. And the line is good, it kind of captures what's going on and if we apply this to the testing set, maybe it's going to do a pretty good job.
- But, if we are feeling kind of obsessive compulsive about it we might say well in this particular case we didn't actually track all the ups and downs of the data.
- We might want to use a higher order polynomial to fit this better. If we can fit this with a higher order polynomial maybe it'll hit all these points much better.
- So now we have this other shape and it's making weird predictions in certain places.
- What we really would like to do is we'd like to use a model that's complex enough to actually model the structure that's in the data that we're training on, but not so complex that it's matching so directly that it doesn't really work well on the test set.
- Unfortunately we don't really have the test set to play with because it’s too much teaching to the test.
- We need to actually learn the true structure that is going to need to be generalized.
- We could take some of the training data and pretend it’s a test set and that wouldn't be cheating because it’s not really the test set. We hold out some of it ,as a kind of make pretend test set, a test test set, a trial test set, what we're going to call **cross validation** set. It's going to be a stand in for the actual test data.
- This cross validation set is going to be really helpful in figuring out what to do.
- We're going to take our training data, and we're going to split it into what are called **folds**. We're going to train on the first three folds, and use the fourth one to see how we did.
- Train on the second third and fourth fold and check on the first one.
- And we're going to we're going to try all these different combinations leaving out each fold as a kind of a fake test set.
- And then average these errors. The goodness of fit. Average them all together, to see how well we've done.
- And, the model class, the degree of the polynomial in this case that does the best job, the lowest error, is the one that we're going to go with.

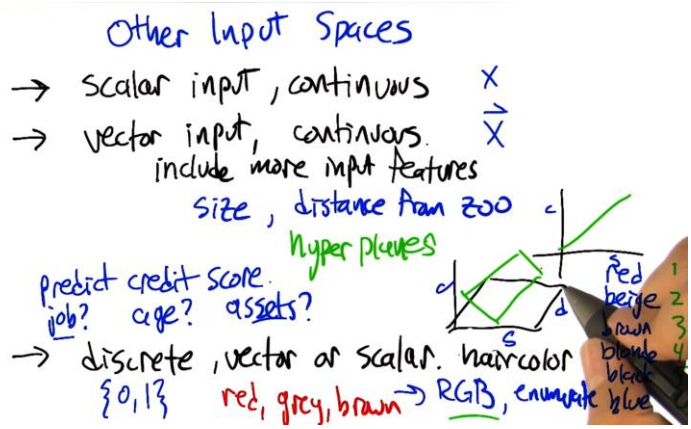
2.8. Housing Example Revisited



- What happens with the training error:
  - As we increase the degree of the polynomial from constant to linear to quadratic and all the way up to when this case order six, the errors always falling.
  - As you go up, you have more ability to fit the data, closer and closer and closer because, each of these models is nested inside the other.
  - We can always go back. If the zero fits best and I give you six degrees of freedom, you can still fit the zero.
- Now let's use this idea of cross validation. Train on the rest of the data, predict on the chunk, repeat that for all the different chunks and average together.
- So I actually did that. And this is what I got with the cross validation error.
- We have this red plot that is constantly falling and the blue plot which is the cross validation error starts out a little bit higher than the, the red plot that's got higher error because we're actually training to minimize error. We're actually trying to minimize error on the training set.
- The parts we aren't looking at, you're more likely to have some error with. That makes sense if you'd have a little bit more error on the data you haven't seen.
- Let's look at what happens as we start to increase the degree, we've got the ability to fit this data better and better and in fact, down at three and four, they're actually pretty close in terms of their ability to fit these examples.
- And then what's really interesting is what happens is now we start to give it the ability to fit the data closer and closer. And by the time we get up to order six polynomial, even though the error on the training set is really low, the error on this cross validation error is really high. And this is beautiful this inverted u, is exactly what you tend to see in these kinds of cases. That the error decreases as you have more power and then it starts to increase as you use too much of that power.
- As it gets more and more and more power it tends to overfit the training data at the expense of future generalization.
- If you don't give enough degrees of freedom, you don't give a model class that's powerful enough and you underfit the data. You won't be able to model what's actually going on and there'll be a lot of error.
- But if you give yourself too much you can overfit the data. You can actually start to model the error and it generalizes very poorly to unseen examples.

- And somewhere in between is kind of the goldilocks zone. Where we're not underfitting and we're not overfitting. We're fitting just right. And that's the point that we really want to find. We want to find the model that fits the data without overfitting, and not underfitting.

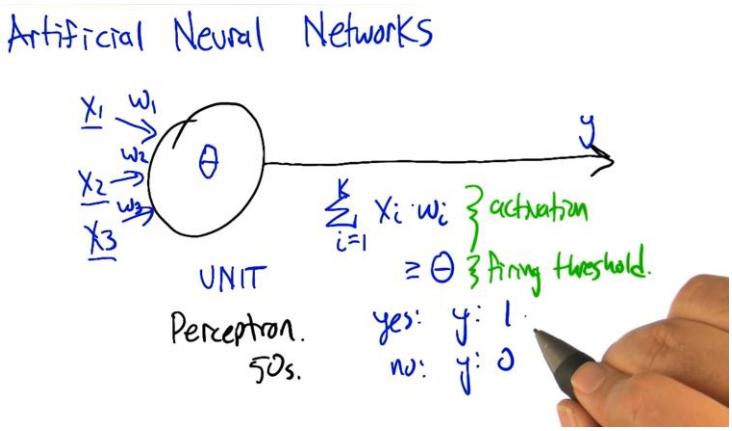
2.9. Other Input Spaces



- Up to this point I've been talking about regression in the context of a scalar input and continuous input. So basically this x variable. But the truth of the matter is we could actually have vector inputs as well.
- If you look at the housing example, like we said earlier, there are a bunch of features that we weren't keeping track off. So we could have added some of those.
- If you think about lines, we can just generalize to planes and hyperplanes.
- This notion of polynomial function generalizes very nicely.

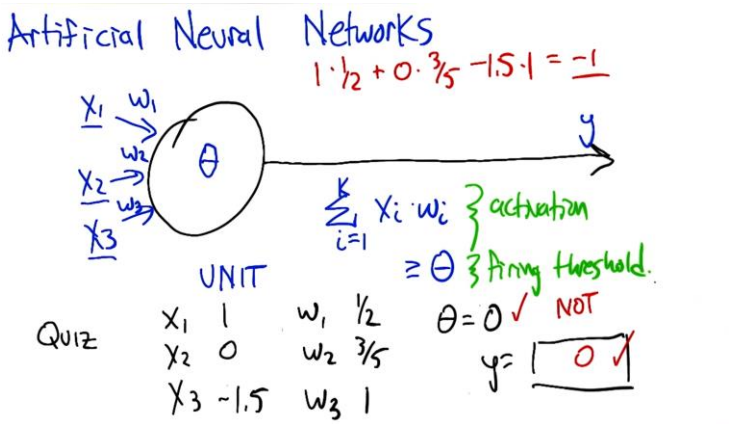
3. Neural Networks

3.1. Neural Networks



- In the field of artificial neural networks we have kind of a cartoonish version of the neuron and networks of neurons and we actually put them together to compute various things.
- One of the nice things about the way that they're set up is that they can be tuned or changed so that they fire under different conditions and therefore compute different things.
- They can be trained through a learning process.
- We're going to have inputs, think of them as firing rates or the strength of inputs. X_1, X_2, and X_3 in this case.
- Those are multiplied by weights, w_1, w_2, w_3 correspondingly.
- The weights turn up the gain or the sensitivity of the neuron, to each of the inputs respectively.
- Then we're going to sum them up. We're going to sum over all the inputs.
- The strength of the input times the weight, and that's going to be the activation.
- Then determine if that is greater than or equal to the firing threshold. If it is then we're going to say the output is one and if it's not, we're going to say the output is zero.
- This is a neural net unit called a **Perceptron**.

3.2. Artificial Neural Networks Quiz

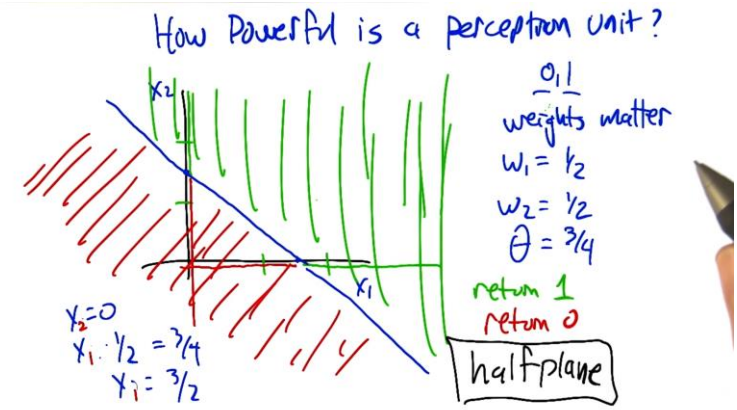


- Here the input is 1, 0, -1.5. For the three different, inputs in this case.
- The corresponding weights, are ½, ⅗, and 1.
- The threshold, let's say is 0, it should fire if the weighted sum is greater than or equal to 0
    - Multiply x_1 times w_1 so that gives us a 1/2
    - Multiply 0 times 3/5 which would get a 0

- Multiply -1.5 times 1 which will give us - 3/2.
- -1.5 plus a 1/2, so it should be negative one.

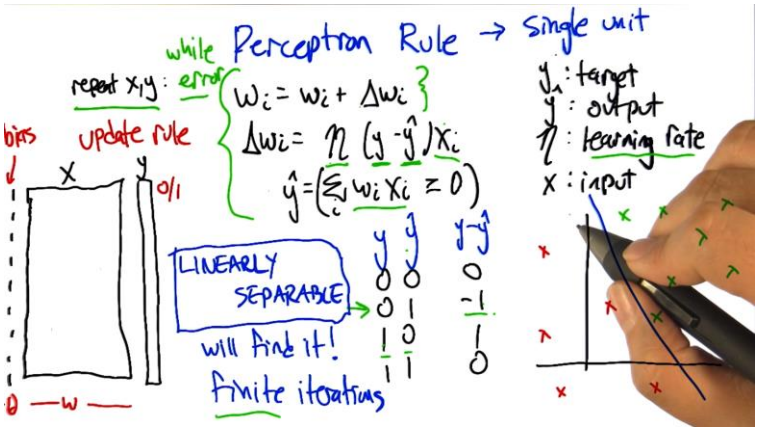
- That's the activation. If the activation is above our threshold theta, which in this case is 0, and it is not
- So the output should be 0.

3.3. How Powerful is a Perceptron Unit



- Notice that the relationships are all linear here.
- So solving this linear inequality gets us a picture like this.
- So this perceptron computes a kind of half plane
- The half of the plane that's above this line is getting us the 1 answers and below that line is giving us a zero answers. Perceptrons are always going to compute lines.
- Perceptron is a linear function, and it computes hyperplanes.

3.4. Perceptron Training

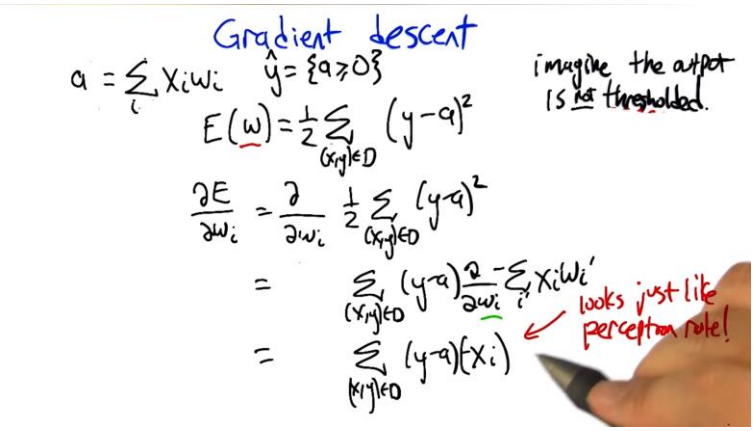


- We'd really like a system that, given examples, finds weights that map the inputs to the outputs.
- We're going to actually look at two different rules that have been developed for doing exactly that, to figuring out what the weights ought to be from training examples.
- One is called the **Perceptron Rule**, and the other is called **gradient descent** or the Delta Rule.
  - The perception rule is going to make use of the threshold outputs
  - Gradient descent is going to use unthresholded values.
- We've got a training set, which is a bunch of examples of x. These are vectors.
- We have y's, which are zeros and ones. These are the output that we want to hit.

- What we want to do is set the weights so that we capture this same data set. And we're going to do that by, modifying the weights over time.
- So one of the things that we're going to do here is we're going to give a learning rate for the weights w, and not give a learning rule for Theta, but we do need to learn the theta.
    - There's a very convenient trick for actually learning them by just treating it as another kind of weight.
    - So if you think about the way that the thresholding function works., we're taking a linear combination of the W's and X's, then we're comparing it to theta.
    - But if you think about just subtracting theta from both sides, then, in some sense theta just becomes another one of the weights, and we're just comparing to zero.
    - So what I did here was take the actual data, the x's, and I added what is sometimes called a bias unit.
    - So basically the input is one always to that. And the weight corresponding to it is going to correspond to negative theta ultimately.
    - This just simplifies things so that the threshold can be treated the same as the weights.
- So from now on, we don't have to worry about the threshold. It just gets folded into the weights, and all our comparisons are going to be just to zero instead of theta.
- We're going to iterate over this training set, grabbing an x, which includes the bias piece, and the y, where y is our target and X is our input.
- We're going to change weight i, the weight corresponding to the ith unit, by the amount that we're changing the weight by.
- We need to define that what that weight change is. We're going to take the target, the thing that we want the output to be, and compare it to what the network with the current weight actually spits out.
- So we compute this, this y hat. This approximate output y. By again summing up the inputs according to the weights and comparing it to zero.
- That gets us a zero one value.
- So we're now comparing that to what the actual value is.
    - If they are both zeros then this y minus y hat is zero and it means the output should have been zero and the output of our current network really was zero, so that's good.
    - If they are both ones, it means the output was supposed to be one and our network outputted one, and the difference between them is going to be zero.
    - But in this other case, y minus y hat, if the output was supposed to be zero, but we said one, our network says one, then we get a negative one.
    - If the output was supposed to be one and we said zero, then we get a positive one.
- Those are the four cases for what's happening here.
- We're going to take that value multiply it by the current input to that unit i, scale it down by the sort of thing that is going to be called the learning rate and use that as the the weight update change.
- If the output is already correct either both on or both off. Then there's going to be no change to the weights.
- If our output is wrong, i.e. we are giving a one when we should have been giving a zero. That means the total here is too large. And so we need to make it smaller.
    - Whichever input xi's correspond to, very large values, we're going to move those weights very far in a negative direction.

- We're taking this negative one times that value times his little learning rate.
- The other case is if the output was supposed to one but we're outputting a zero, that means our total is too small.
  - And what this rule says is increase the weights essentially to try to make the sum bigger.
- Now, we don't want to overdo it, and that's what this learning rate is about.
- Learning rate basically says we'll figure out the direction that we want to move things and just take a little step in that direction.
- We'll keep repeating over all of the input output pairs. We'll have a chance to get into really building things up, but we're going to do it a little bit at a time so we don't overshoot.
- If there is such a half plane that separates the positive from the negative examples, then we say that the data set is linearly separable. That there is a way of separating the positives and negatives with a line.
- What's cool about the perception rule, is that if we have data that is linearly separable. The Perceptron Rule will find it. It only needs a finite number of iterations to find it. It will actually find a line, and it will stop after finding a set of weights that do the trick.
- That happens if the data set is in fact linearly separable and that's pretty cool.
- If the data is linearly separable, then the algorithm works, so the algorithm simply needs to only be run when the data is linearly separable. It's generally not that easy tell actually, when your data is linearly separable especially, here we have it in two dimensions, if it's in 50 dimensions, know whether or not there is a setting of those perimeters that makes it linearly separable, not so clear.

3.5. Gradient Descent



- We are going to need a learning algorithm that is more robust to non-linear separability
- Gradient descent is going to give us an algorithm for doing exactly that. S
- What we did before was:
  - Did a summation over all the different input features of the activation on that input feature times the weight, w, for that input feature.
  - Sum all those up and get an activation. $a = \sum_i \quad x_i w_i$
  - And then we have our estimated output as whether or not that activation is greater than or equal to zero.

$$\hat{y} = \{a \geq 0\}$$

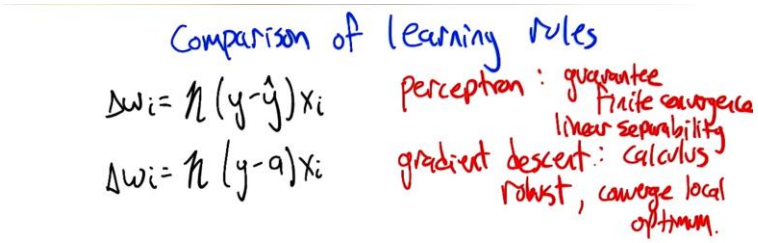
- Let's imagine that the output is not thresholded when we're doing the training, and what we're going to do instead is try to figure out the weight so that the non thresholded value is as close to the target as we can.
- This actually kind of brings us back to the regression story.
- We can define an error metric on the weight vector w. $E(w)$
- And the form of that's going to be one half times the sum over all the data in the dataset, of what the target was supposed to be for that particular example. Minus what the activation actually was, the activation being the dot product between the weights and the input, and we're going to square that. We're going to square that error and we want to try to now minimize that.

  $E(w) = \frac{1}{2}\sum_{(x,y)\in D} \quad (y-a)^2$

  - Why ½ of that? It turns out that in terms of minimizing the error this is just a constant and it doesn't matter.
  - So why do we stick in a half there? Let's get back to that.
- Just like in the regression case we're going to fall back to calculus, calculus is going to tell us how we can push around these weights, to try to push this error down. We want to know how changing the weight changes the error, and we want to push the weight in the direction that causes the error to go down.
  - Take the partial derivative of this error metric with respect to each of the individual weights, so that we'll know for each weight which way we should push it a little bit to move in the direction of the gradient. That's the partial derivative with respect to weight wi, of exactly this error measure. $\frac{2E}{2w_i} = \frac{2}{2w_i} * \frac{1}{2}\sum_{(x,y)\in D} \quad (y-a)^2$
  - So to take this partial derivative we just use the chain rule as we always do. Take the power, move it to the front, keep this thing, and then take the derivative of this thing.
  - This answers the question of why we put a half in there. Because down the line, it's going to be really convenient that two and the half canceled out. It's just going to mean that our partial derivative is going to look simpler, even though our error measure looked a little bit more complicated.
  - What we're left with is the sum over all these data points of what was inside this quantity here times the derivative of that, and here I expanded the a to be, the definition of the a.

    $= \sum_{(x,y)\in D} \quad (y-a) \quad * \frac{2}{2w_i} - \sum_i \quad x_i W_i'$
  - Now, we need to take the partial derivative with respect to weight w_i of this sum that involves a bunch of the w's in it.
  - So, when don't match the w_i, that derivative is going to be zero because changing the weight won't have any impact on it. The only place where changing this weight has any impact is at x_i. So that's what we end up carrying down. This summation disappears. And all that's left is just the one term that matches the weight that we care about. $\sum_{(x,y)\in D} \quad (y-a) \quad (-x_i)$
- So this is what we're left with, the derivative of the error with respect to any weight w_i is exactly the sum of the difference between the activation and the target output times the activation on that input unit
- That looks almost exactly like the rule that we use with the perceptrons before.
- This is now just a derivative, but let's actually write down what our weight update is going to be because we're going to take a little step in the direction of this derivative and it's going to involve a learning rate.

3.6. Comparison of Learning Rules



Here's our update rules what they end up being.

- The gradient descent rule says what we want to do is move the weights in the negative direction of the gradient.
  - If we negate the expression that we had before and take a little step in that direction we get exactly this expression.
  - Multiply the input on that weight times the target minus the activation.

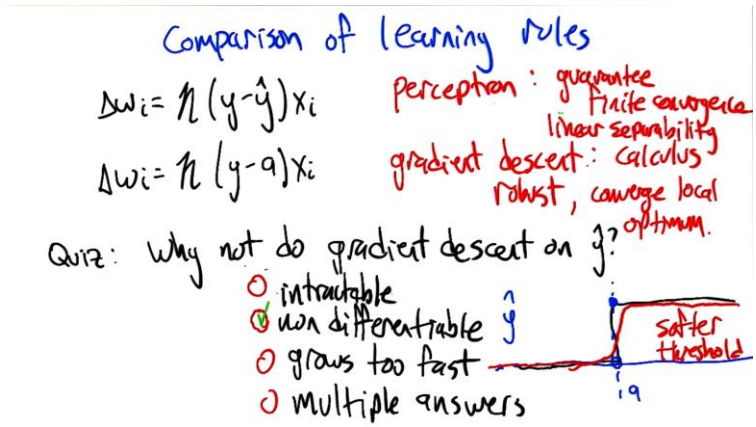
$$\triangle w_i = \eta(y - a)x_i$$

- In the perceptron case take that same activation, thresholding it, determine whether it's positive or negative, putting in a zero or a one, and putting that in here, that's what y hat (ŷ) is.

$$\triangle w_i = \eta(y - \hat{y})x_i$$

- So really it's the same thing except in one case we have done the thresholding and in the other case we have not done the thresholding.
- But we end up with two different algorithms with two different behaviors.
  - The perceptron has this nice guarantee. A finite convergence, which is a really good thing, but that's only in the case where we have linear separability.
  - Whereas the gradient descent rule is good because it's more robust to data sets that are not linearly separable, but it's only going to converge in the limit. To a local optimum.

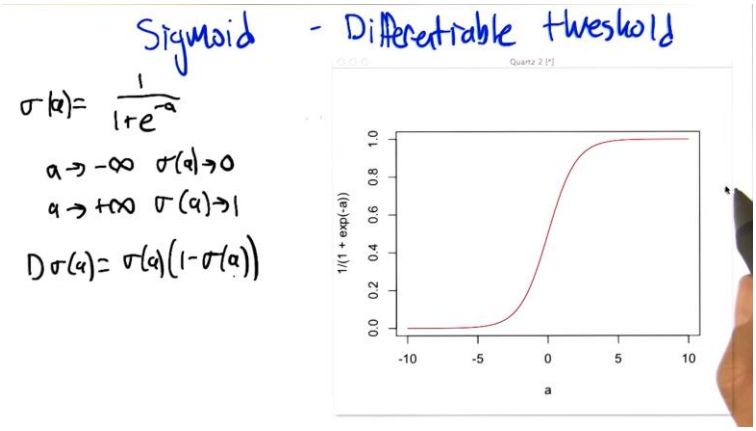
3.7. Comparison of Learning Rules Quiz



- Once we see these two things next to each other, it raises the question, why don't we just use a gradient descent type on an error metric that's defined in terms of $\hat{y}$instead of the activation$a$?
  - $\hat{y}$is the thing, that we really want to match the output.
- We don't really want the activation to match the output. There's no need for that.

- So why don't we do gradient descent on $\hat{y}$?
- There could be many reasons but the main reason is it's not differentiable. It's a discontinuous function. There's no way to take the derivative at the point where it's discontinuous.
- This activation thing. The change from activation to $\hat{y}$ has this big step function jump in it, right at zero.
- So once the activation goes positive, actually at zero. It jumps up to one. And before that, it's not. So the derivative is basically zero, and then not differentiable, and then zero again.
- So really, the zero's not giving us any direction to push, in terms of how to fix the weights. And the undefined part doesn't really give us any information either. So this algorithm doesn't really work, if you try to take the derivative through this discontinuous function.
- What if we made this, more differentiable? What is it that makes this so un-differentiable? It's this really pointy spot.
- You could imagine a function that was like this, but then, instead of the point spot, it smoothed out a bit. Kind of a softer version of a threshold, which isn't exactly a threshold. But at least it's differentiable.
- So that would force the algorithm to put its money where its mouth is.
- If that really is the reason that the problem is non-differentiable, fine, we'll make it differentiable.

3.8. Sigmoid



- Let's take a look at a function called the **sigmoid**.
- We're going to define the sigmoid using the letter sigma (σ) and it's going to be applied to the activation just like we were doing before, but instead of thresholding it at zero, what it's instead going to do is compute this function of$a$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- It ought to be clear that as the activation gets less and less, we'd want it to go to zero, and in fact it does.
  - As a goes to negative infinity, the negative a goes to infinity, e to the infinity is something really, really big. So it's 1 over something really big, which is almost zero.

    $$a \to -\infty \; then \; \sigma(a) \to 0$$
  - The sigmoid function goes to zero as the activation goes to negative infinity, just like threshold

- As the activation gets really really large, we're talking about e to the minus something really large, which is like e to the negative infinity which is almost zero, so one over one plus zero is essentially one.

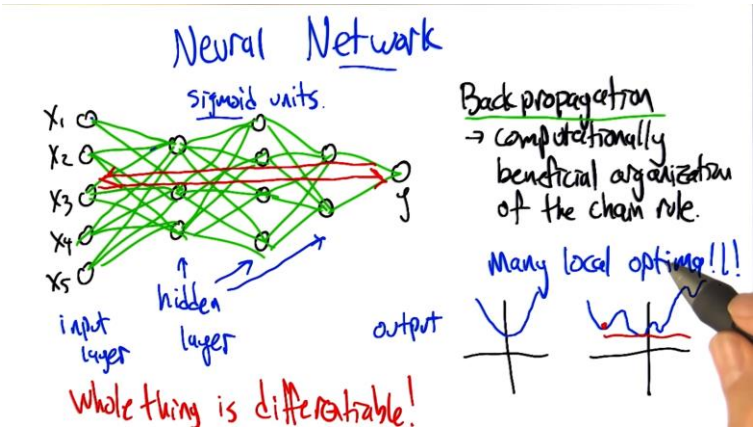
$$a \to \infty \; then \; \sigma(a) \to 1$$

- Reference slide, minus five and below it's essentially at zero, and then it makes this kind of gradual, sigmoid s-shaped curve, then it comes back up to the top and it's basically at one by the time it get to five.
- Instead of just an abrupt of transition to zero, we have this gradual transition between negative five and five. Now it's differentiable and now we can take derivatives
- Not only is this function differentiable, but the derivative itself has a very beautiful form. It turns out if you take the derivative of this sigma function, it can be written as the function itself times one minus the function itself.

$$D\sigma(a) = \sigma(a)(1 - \sigma(a))$$

- This is just really elegant and simple. If you have the sigma function in your code, there's nothing special that you need for the derivative. You could just compute it this way.

3.9. Neural Network Sketch

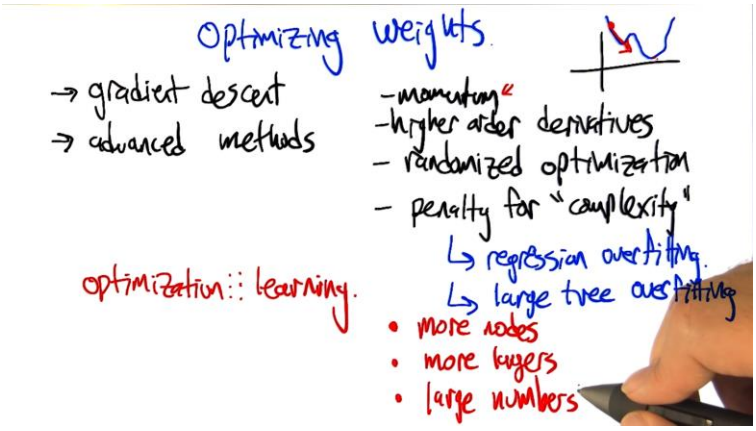


- Now the idea is that using exactly these kind of sigmoid units, we can construct a chain of relationships between the input layer, which are the different components of x, with the output y.
- The way this is going to happen is there's other layers of units in between and each one is computing the weighted sum, sigmoided, of the layer before it.
- These other layers of units are often referred to as **hidden layers** because you can see the inputs and the outputs. This other stuff is less constrained, or indirectly constrained.
- Each of these units takes the weights, multiplied by the things coming into it, put it through the sigmoid and that's your activation, that's your output.
- What's cool about this is, in the case where all these are sigmoid units this mapping from input to output is differentiable in terms of the weights, and by saying the whole thing is differentiable, what I'm saying is that we can figure out for any given weight in the network how moving it up or down a little bit is going to change the mapping from inputs to outputs.
- So we can move all those weights in the direction of producing something more like the output that we want. Even though there's all these sort of crazy non linearities in between.
- And so, this leads to an idea called **backpropagation**, which at its heart is a computationally beneficial organization of the chain rule.

- Compute the derivatives with respect to all the different weights in the network, all in one convenient way, that has this lovely interpretation of having information flowing from the inputs to the outputs.
- Then error information flowing back from the outputs towards the inputs
- That tells you how to compute all the derivatives
- Then, therefore how to make all the weight updates to make the network produce something more like what you wanted it to produce.

- This is where learning is actually taking place, this backpropagation is referring to the fact that the errors are flowing backwards. Sometimes it is even called error backpropagation.
- If you replace the sigmoid units with some other function and that function is also differentiable, then we can still do this basic kind of trick that says we can compute derivatives, and therefore we can move weights around to try to get the network to produce what we want it to produce.
- It's really just analogous to a perceptron, because we're not really doing the hard thresholding, we don't have guarantees of convergence in finite time.
- In fact, the error function can have many local optima, and what we mean by that is this idea that we're trying to set the weight so that the error is low, but you can get to these situations where none of the weights can really change without making the error worse. And you'd like to think we're done. We've made the error as low as we can make it, but in fact it could actually just be stuck in a local optima, that there's a much better way of setting the weights It's just we have to change more than just one weight at a time to get there.
- So if we think about the sigmoid and the error function that we picked, the error function was sum of squared errors, so that looks like a parabola in some high dimensional space, but once we start combining them with others like this over and over again then we have an error space where there may be lots of places that look low but only look low if you're standing there but globally would not be the lowest point.
- So you can get these situations in just the one unit version where the error function as you said is this nice little parabola and you can move down the gradient and when you get down to the bottom you're done.
- But when we start throwing these networks of units together, we can get an error surface that looks just in its cartoon form looks crazy like the slide, and you could easily get yourself stuck at a point like this where you're not at the global minimum. You're at some local optimum.

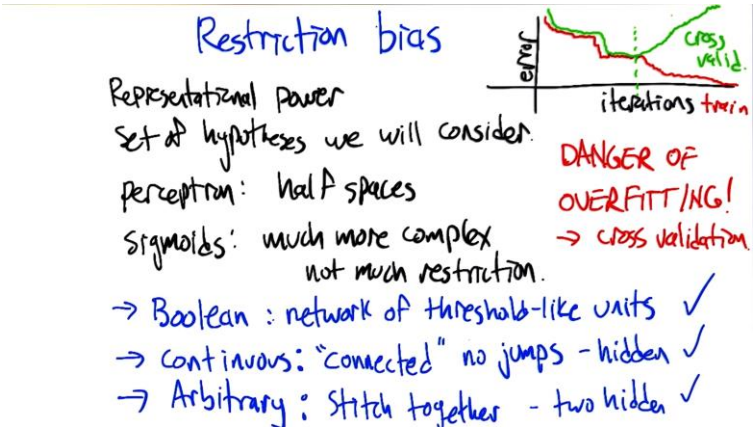
3.10. Optimizing Weights



- One of the things that goes wrong, when you try to actually run gradient descent on a complex network with a lot of data is that you can get stuck in these local minima. I'm trying to find a set of weights for the neural network that tries to minimize error on the training set.
- So, gradient descent is one way to do it, and it can get stuck, but there's other kinds of advanced optimization methods that become very appropriate here. In fact, there's a lot of people in machine learning who think of optimization and learning as kind of being the same thing.
- What you're really trying to do in any kind of learning problem is solve this high order, very difficult optimization problem to figure out what the learned representation needs to be.
- There's things like using momentum terms in the gradient
    - As we're doing gradient descent we don't want to get stick on this ball here, we want to kind of pass all the way through it to get to this ball, so maybe we need to just continue in the direction we've been going.
    - So, instead of thinking of it as a kind of physical analogy. Instead of just going to the bottom of this hill and getting stuck, it can kind of bounce out and pop over and come to, what might be a lower, minima, later.
- There's a lot of work in using higher order derivatives to better optimize things instead of just thinking about the way that individual weights change the error function to look at combinations of weights.
    - Hamiltonions and what not.
- There's various ideas for randomized optimization that can be applied to make things more robust.
- And sometimes it's worth thinking we don't really want to just minimize the error on the training set, we may actually want to have some kind of penalty for using a structure that's too complex.
    - Something similar also happened in the decision tree section. We had an issue with decision trees where if we let the tree grow too much to explain every little quirk in the data, you'd overfit.
    - We came up with a lot of ways of dealing with that, like pruning. Not going too far deeply into the tree. You can either do that by filling out the tree and then backing up so you only have a little bit of small error or by stopping once you've reached some sort of threshold as you grow the tree out.
    - That's really the same as giving some kind of penalty for complexity.
    - Complexity in the tree setting has to do with the size of the tree, in regression it had to do with the order of the polynomial.
- For neural network, there's two things you can do with networks
    - Add more and more nodes
    - Add more and more layers.
- The more nodes that we put into a network:
    - The more complicated the mapping becomes from input to output
    - The more local minima we get
    - The more we have the ability to actually model the noise, which brings up exactly the same overfitting issues.
- Another thing that can make a complex network is the numbers, the weights, are very large.
    - So same number of weights, same number of nodes, same number of layers, but larger numbers often leads to more complex networks and the possibility of overfitting.

- Sometimes we want to penalize a network not just by giving it fewer nodes or layers but also by keeping the numbers in a reasonable range.

3.11. Restriction Bias

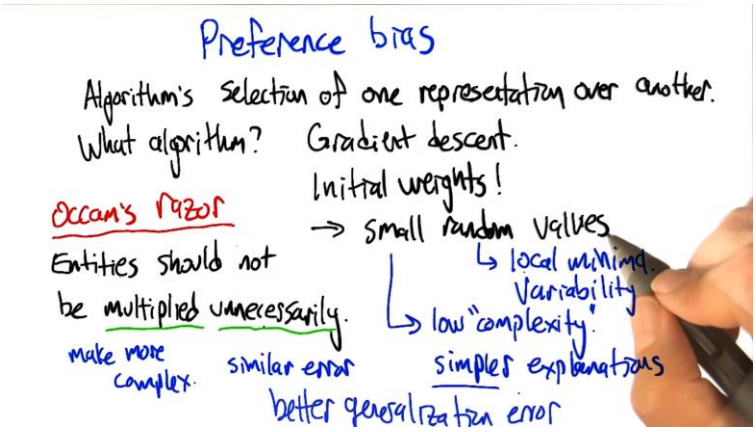


- What are neural networks more or less appropriate for, what is the restriction bias and the inductive bias of this class of classifiers, and regression algorithms?
- Restriction bias tells you something about the representational power of whatever data structure it is that you're using and it tells you the set of hypotheses that you're willing to consider.
  - If there's a great deal of restriction, then there's lots of different kinds of models that we're not even considering.
- We started out with a simple perceptron unit, and that we decided was linear. So we were only considering planes.
- Then we move to networks, so that we could do things like XOR, and that allowed us to do more.
- Then we started sticking Sigmoids and other arbitrary functions into nodes so that we could represent more and more. If you let weights get big and we have lots of layers and lots of nodes they can be really complex.
- Therefore we are actually not doing much of a restriction at all.
- Boolean functions we can represent if we give ourselves a complex enough network with enough units, we can basically map all the different sub-components of any Boolean expression to threshold like units and basically build a circuit that can compute whatever Boolean function we want.
- What about continuous functions? A continuous function is one where, as the input changes the output changes somewhat smoothly, there's no jumps in the function. There's no discontinuities.
  - If we've got a continuous function that we're trying to model with a neural network, as long as it's connected, it has no discontinuous jumps to any place in the space, we can do this with just a single hidden layer, as long as we have enough hidden units, as long as there's enough units in that layer.
  - One way to think about that is, if we have enough hidden units, each hidden unit can worry about one little patch of the function that it needs to model.

    - The patches get set at the hidden layer and at the output layer they get stitched together. And if you just have that one layer you can make any function as long as it's continuous.
- If it's an arbitrary function, we can still represent that in our neural network.
    - Any mapping from inputs to outputs we can represent, even if it's discontinuous, just by adding one more hidden layer, so two total hidden layers.
    - And that gives us the ability to not just stitch these patches at their seams, but also to have big jumps between the patches.
- So in fact, neural networks are not very restrictive in terms of their bias as long as you have a sufficiently complex network structure, so maybe multiple hidden layers and multiple units.
- This is worrisome because it means that we're almost certainly going to overfit. We're going to have arbitrarily complicated neural networks and we can represent anything we want to. Including all of the noise that's represented in our training set.
- It is the case though, that when we train neural networks, we typically give them some bounded number of hidden units and we give them some bounded number of layers.
    - It's not like any fixed network can actually capture any arbitrary function, any fixed network can only capture whatever it can capture, which is a smaller set.
    - So going to neural nets in general doesn't have much restriction, but any given network architecture actually does have a bit more restriction.
- Another thing we can do with overfitting is what we've done the other times we've had to deal with overfitting.
    - Use ideas like, cross validation to decide:
        - How many hidden layers to use.
        - How many nodes to put in each layer.
        - When to stop training because the weights have gotten too large.
- It's probably worth pointing out that this is a different property from the other classes of supervised learning algorithms we've looked at so far.
    - In a decision tree, you build up the decision tree and you may have overfit.
    - In regression, you solve the regression problem, and again that may have overfit.
- What's interesting about neural network training is it's this iterative process that you started out running, and as it's running, it's actually errors going down and down.
- So, in this standard kind of graph, we get the error on the training set dropping as we increase iterations.
    - It's doing a better and better job of modeling the training data.
- But, in classic style, if you look at the error in some kind of held-out test set, or maybe in a cross validation set, you see the error starting out kind of high and maybe dropping along with this, and at some point it actually turns around and goes the other way.
- So here, even though we're not changing the network structure itself, we're just continuing to improve our fit, we actually get this pattern that we've seen before, that the cross validation error can turn around and at this low point, you might want to just stop training your network there.
- The more you train it, possibly the worse you'll do.
- It's reflecting this idea that the complexity of the network is not just in the nodes and the layers, but also in the magnitude of the weights.
- Typically what happens at this turnaround point is that some weights are actually getting larger and larger and larger.

- Just wanted to highlight that difference between neural net function approximation of what we see in some of the other algorithms

3.12. Preference Bias

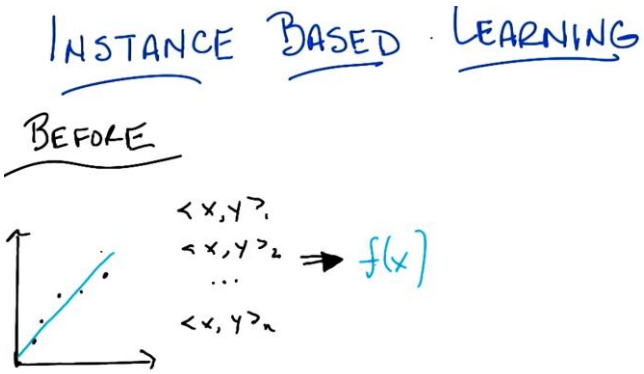


- Preference bias tells you something about the algorithm that you are using to learn that tells you, given two representations, why I would prefer one over the other.
- Think back what we talked about with decision trees, we preferred trees where nodes near the top had high information gain, we preferred correct trees, we preferred trees that were shorter to ones that were longer unnecessarily and so on and so forth.
- We haven't actually chosen an algorithm. We talked about how derivatives work, how backpropagation works, but we haven't talked about how we start? You tell me how to update the weights but, how do I start out with the weights?
- You can't run this algorithm without initializing the weights to something. We did talk about how you update the weights but they don't just start undefined and you can't just update something that's undefined. We have to set the initial weights to something.
- Pretty typical thing for people to do, is small, random, values
    - Random values because we have no particular reason to pick one set of values over another. So you start somewhere in the space. Probably helps us to avoid local minimum. There's also the issue, if we run the algorithm multiple times if we get stuck, we like it not to get stuck exactly there again if you run it again. So it gives some variability, which is a helpful thing in avoiding local minimal.
    - Small values because if the weights get really big that can sometimes lead to overfitting, because it let's you represent arbitrarily complex functions.
- If we start out with small random values, that means we are starting out with low complexity. So that means we prefer simpler explanations to more complex explanations, correct answers to incorrect answers, and so on and so forth.
- So neural networks implement a kind of bias that says we prefer correct over incorrect but all things being equal, the simpler explanation is preferred.
- This reminiscent of the principal that is known as **Occam's razor**, which is often stated as entities should not be multiplied unnecessarily.
    - It's necessary if you're getting better explanatory power, you're fitting your data better.
    - Unnecessarily would mean we're not doing any better at fitting the data. If we're not doing any better at fitting the data, then we should not multiply entities.

- Multiply here means make more complex. So don't make something more complex unless you're getting better error, or, if two things have similar error, choose the simpler one, use the one that's less complex.
- That has been shown to, if you mathematize this and you use it in the context of supervised learning, that we're going to get better generalization error with simpler hypotheses.

4. Instance-Based Learning

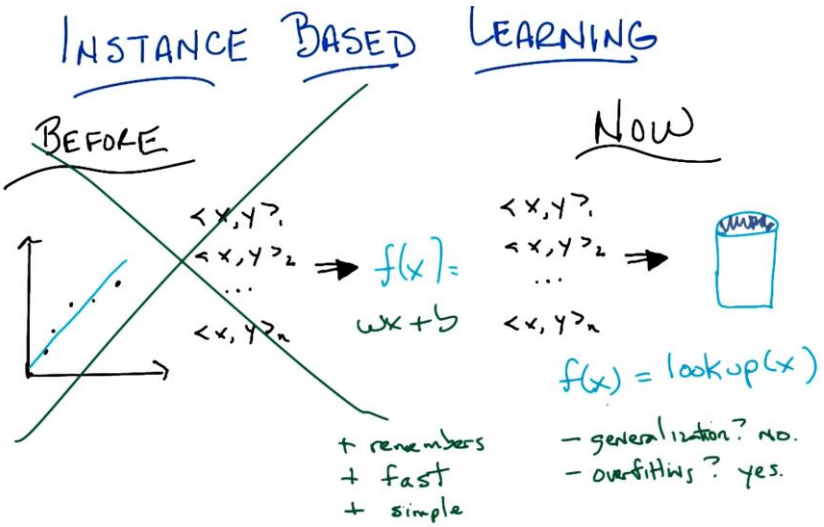
4.1. Instance Based Learning Before



Before:

- We were given a bunch of training data,
- And we would then learn some function.
- For example, if we have a bunch of points in a plane, we might learn a line to represent them
- And what was happening here is we take all this data, we come up with some function that represents the data, and then we would throw the data away effectively
- And then in the future when we get some data point, let's call it x, we would pass it through this function whatever it is, and that would be how we would determine answers going forward.
- So. I want to propose an alternative and the alternative is basically going to not do this.

4.2. Instance Based Learning Now



What I'm proposing

- Take all the training data we had and put it in a database.
- Next time we get some new x to look at just look it up in the database.
- Done.
- None of this fancy shmancy learning. None of this producing an appropriate function like wx+b. None of that fancy stuff anymore. We just stick into the database. We'll look it up when we're done. Period.

Advantages:

- It doesn't forget, so it actually is very reliable.
- It's very dependable, if you put in an x, y pair you ask for the x you're going to get that y back instead of some kind of crack potty, smooth version of it.
- Another thing is that there's none of this wasted time doing learning. It just takes the data and it very rapidly just puts it in the database. So it's fast.
- It's simple.

Issues:

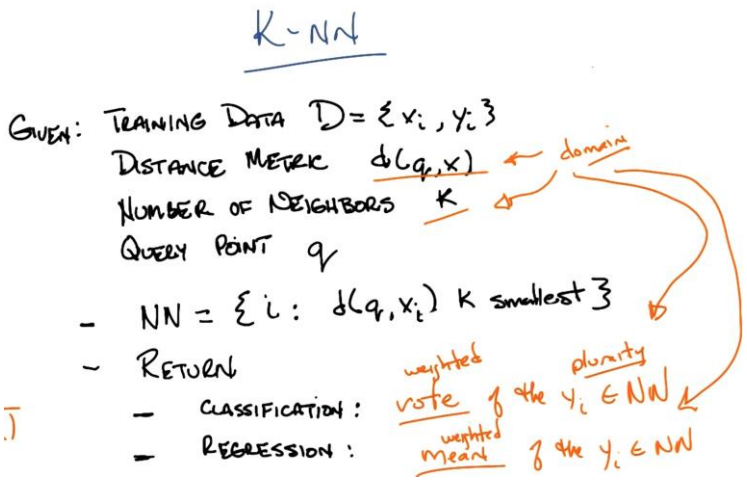
- In particular the way that you wrote $f(x) = lookup(x)$. If I give you one of the other points in between it will return no such point found. Which means it's really quite conservative. It's not willing to go out on a limb and say well I haven't seen this before but it ought to be around here, instead it just says, I don't know.
- So the down side of remembering is, no generalization.
- A similar issue is that it reminds me of the issues that we saw with regard to overfitting. It bottles the noise exactly, it bottles exactly what it was given. So it's going to be very sensitive to noise. It can overfit in a couple of ways:
  - By believing the data too much that is literally believing all of it
  - What do you do if the same x shows up multiple times but each with a different y.

4.3. Cost of the House

- Here's some data, it's a graph and you see here's a y axis and here's an x axis, and each of these points, represents a house, on this map, which I'm, I'm cleverly, using in the background.
- You'll notice, that each of the dots is colored. I'm going to say that red represents, really expensive houses, blue represents, moderately expensive houses, and green represents, really inexpensive houses.
- Using the technique that we talked about, use ML and the data to determine the cost of the black dots
- Using neighbors you can figure this out, except the house in the middle, this one doesn't really have any very close neighbors on the map.
- So, this whole nearest neighbor thing doesn't quite work, in this case when you got a bunch of neighbors that are saying different things.
- We need to look at a bigger context - look at more of my neighbors than just the closest one.

- Also, we've been talking about distance sort of implicitly. But this notion of distance is actually quite important.
  - Maybe distance is straight-line distance, as the crow flies, driving distance, etc.
- We have to be very careful what we mean by distance
- So, we went from just picking our nearest neighbor to picking our nearest neighbors. What's a good value you think we should stick to with neighbors? We could do as many nearest neighbors as is appropriate. Or maybe we should just make it a free parameter and call it K. K nearest neighbors and we'll pick our K numbers. So we have an algorithm, **k nearest neighbors**, which takes K nearest neighbors as a way of deciding how you're going to label some query point here.
- And we've identified two parameters to the algorithm so far.
  - K - the number of neighbors we're going to use.
  - And some notion of distance.
- We're using distance here in a kind of in an overloaded sense, because this is something on a map. But really distance is a standard for similarity.
- In other ways, things like the number of veterans you have, whether you're on one side of the highway or the other, the school district you're in, things like that, are other things you might add as features or dimensions when you talk about similarity or distance.
- We have a general algorithm now and I think it does a pretty good job of addressing the points you brought up.
- We no longer have to worry about overfitting as much, at least it seems that way to me. And we have a way of being a little bit more robust about not having an exact data point in the database.

4.4. K-NN



- Here is pseudocode for our K-NN algorithm. And I'm sort of writing it as like, a function:
  - So, you're going to be given some training data D $D = \{x_i, y_i\}$
  - You're given some kind of distance metric or similarity function. This is important because this represents the domain knowledge
    $d(q,x)$
  - You get some number of neighbors that you care about,k, which also represents domain knowledge. Tells you something about how many neighbors you think you should have. k

- And then are given some particular new query point and I want to output some kind of answer, some label, some value.

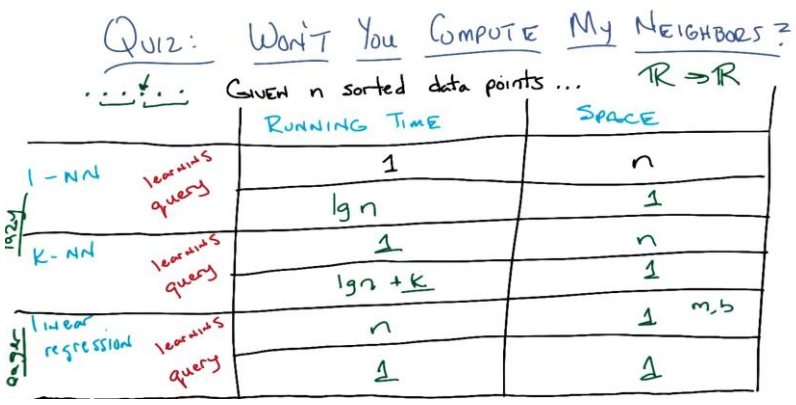
  q
- So the K-NN algorithm is remarkably simple given these things. You simply find a set of nearest neighbors such that they are the K closest to your query point.

  $NN = \{i: d(q, x_i)\ k\ smallest\}$

What to return. One is where we're doing classification and one is where we're doing regression.

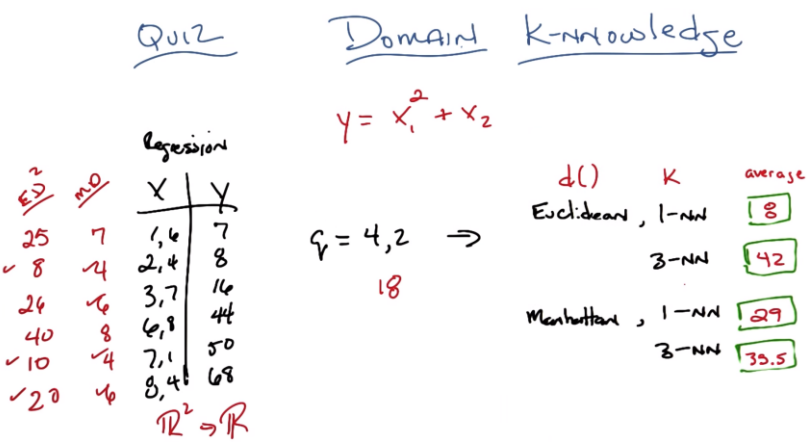
- Classification:
  - A reasonable thing to do there would be. Did we get Y's associated with the things in NN, and if so, simply vote, whichever y_i is most frequent among the closest points wins. Essentially find a vote of the y_i's, that are apart of the neighborhood set and take the plurality, whichever one occurs the most (the mode)
  - If there are ties among the output, then you're just going to have to pick one, and there's lots of ways you might do that. You may take the tie that is most commonly represented in the data period. Or I'll just randomly pick each time, or any number of ways you might imagine doing that.
- Regression
  - In the regression case our y_i's are numbers. If you have to return one, a standard thing to do would be to take the average, or the mean.
  - Simply take the mean of the y_i's, and you don't have to worry about a tie.
- If you have more than k that are closest because you have a bunch of ties, in terms of the distance, just take all of them. Get the smallest number greater than or equal to k.
- It's a very simple algorithm, but some of that's because a lot of decisions are being left up to the designer.
  - The distance metric, The number k, How you're going to break ties, exactly how you choose to implement voting or implement the mean or the average operation that shows how to do here.
  - You could put a bunch of different things here in the hands of the designer and you could end up with completely different answer.
- One thing that you might do, as an example, is rather than doing a simple vote by counting, you could do a vote that is weighted by how far away you are. So we could have a weighted vote. That might help us with ties.

4.5. Quiz: Won't You Compute My Neighbors



- They say that KNN are lazy learners, versus something like linear regression, which is an eager learner.
- Linear regression is eager. It wants to learn right away and it does.
- Nearest neighbor algorithms are lazy. They want to put off learning until they absolutely have to and so we refer to this class as lazy and this class as eager.
- If we never query it then the lazy learner definitely comes out ahead.

4.6. Quiz: Domain K NNowledge



- There's several lessons here. And one lesson I don't want you to take away.
- So here's the lesson: I actually had a real function here.
  - There was no noise.
  - It was fairly well represented.
  - The proper answer was 18 and basically none of these are right.
  - But the first thing I want you to notice is you get completely different answers, depending upon exactly whether you do one versus three, whether you do Euclidean versus Manhattan.
  - And that's because these things make assumptions about your domain that might not be particularly relevant.
  - This suggests that maybe this thing doesn't do very well, that KNN doesn't do very well because none of these are close to 18. That seems a little sad.
- However, KNN tends to work really, really well, especially given it's simplicity
- It just doesn't in this particular case, and there's really a reason for that.
  - It has to do with this sort of fundamental assumptions in bias of KNN this example happened to violate some of that bias.
  - So I think it's worth to take a moment to think about what the preference bias is for KNN and to see if that can lead us to understanding why we didn't get anything close to 18 here.

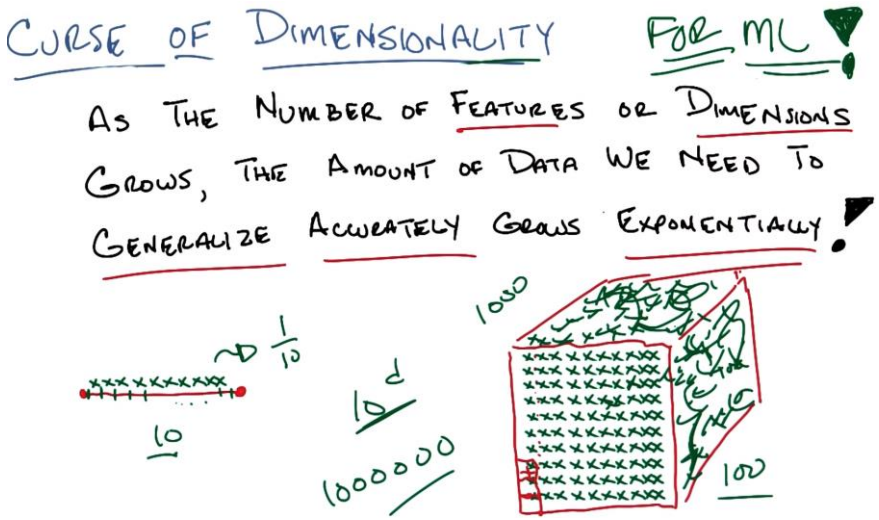
4.7. KNN Bias

- Preference bias is our notion of why we would prefer one hypothesis over another, all other things being equal. And what that really means is, it's the thing that encompasses our belief about what makes a good hypothesis.
    - So in some of the previous examples that we used it was things like shorter trees, smoother functions, simpler functions, those sorts of things were the ways that we expressed our preferences over various hypothesis.
    - KNN is no exception. It also has preference by its built in as does every algorithm of any note.
- There are three that are indicative of this bias, and they're all somewhat related to one another.
- The first one is a notion of **locality**.
    - There's this idea that near points are similar to one another.
    - The whole thing we are using to generalize from one thing to another is this notion of nearness.
    - Exactly how this notion of nearness works out is embedded in whatever distance function we happen to be given. There's further bias that might come out, based upon exactly the way we implement distances.
    - So, in the example we just did, euclidian distance is making a different assumption about what nearness or similarity is, compared to Manhattan distance, for example.
    - So, locality however it's expressed in the distance function that is similarity, is built in to KNN that we believe that near points are similar. Kind of by definition.
- That leads actually to the second preference bias which is this notion of **smoothness**.
    - By choosing to average and by choosing to look at points that are similar to one another, we are expecting functions to behave, smoothly.
    - In the 2D case it's kind of easy to see, you have these points and you're basically saying, these two points should somehow be related to one another more than this point and this point.
    - And that sort of assumes kind of smoothly changing behavior as you move from one neighborhood to another.
- There is another assumption which is a bit more subtle

- For at least the distance functions we've looked at before, the Euclidian distance and the Manhattan distance, they all kind of looked at each of the dimensions and subtracted them and squared them or didn't or took their absolute value and added them all together.
- What that means is we were treating, at least in those cases, that all the features mattered, and they mattered equally.
- Trying to think about what that might mean, its definitely the case that when you look for similar examples in the database you want to care more about x_1 because a little bit of a difference in x_1 gets squared out. It can lead to a very large difference in the corresponding y value.
- Whereas in the x_2's, it's not quite as crucial. If you're off a little bit more, then you're off a little bit more, it's just a linear relationship. It does seems like that first dimension needs to be a lot more important, I guess, when you're doing the matching. Then the second one.

- The notion of relevance turns out to be very important and highlights a weakness of KNN.
- This brings a theorem or fundamental result of machine learning that is particularly relevant to KNN, but it's actually relevant everywhere.

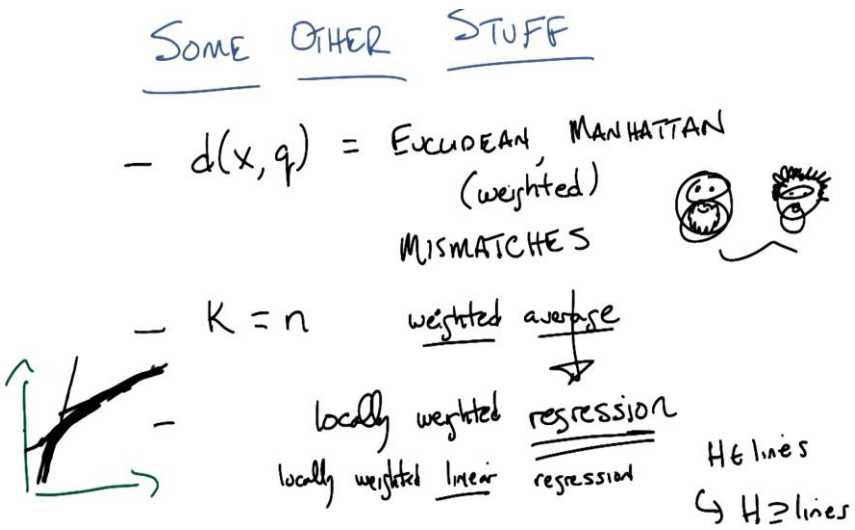
4.8. Curse of Dimensionality



- As the number of features, or equivalently dimensions, grows, that is as we add more and more features, the amount of data that we need to generalize accurately also grows exponentially.
- This is a problem of course because Exponentially means bad in computer science land because when things are exponential they're effectively untenable. You just can't win. It means that we need more and more data as we add features and dimensions.
- Now as a machine learning person this is a real problem because what you want to do, or what your instinct tells you to do is, we've got this problem, we've got a bunch of data, we're not sure what's important. So why don't we just keep adding more and more features.
- We've got all these sensors and we'll just add this little bit and this little bit, and we'll keep track of GPS location and we'll see the time of the day and we'll just keep adding stuff and then we'll figure out which ones are important.
- But the **curse of dimensionality** says that every time you add another one of these features, you add another dimension to your input space, you're going to need exponentially more data as you add those features, in order to be able to generalize accurately.
- This is a very serious problem and it captures some of the difficulties in KNN.

- If you have a distance function or a similarity function, that assumes that everything is relevant, or equally relevant, or important, and some of them aren't, you're going to have to see a lot of data before you can figure that out, sort of before it washes itself away.
- Let's look at this little line segment. Say I've got ten little points that I could put down on this line segment and I want them all to represent some part of this line.
- Let's pretend I did the right thing here and I have them kind of uniformly distributed across the line segment. So that means each one of these points is sort of owning, an equal size sub segment of this segment, each owning 10%
- Let's say I move from a line segment now to a two dimensional space. So a little square segment and I've taken my little ten x's, and I put them down here before. How can I make it so that each of the x's I put down represents the same amount of distance as the x's in the line segment? Fill up the square with x's.
- That'll be 100 x's. So each one now holds a hundredth of the space, and I went from needing ten points to 100 points in order to cover the same amount of space.
- What happens if I now move into three dimensions? In this case we're talking about data points in a nearest neighbor method and boy that does seem like a big growth from ten to a 100 to 1000. the growth is exponential.
- If we went into four dimensions then we would need need 10,000 points. And in six dimensions, we would need 1,000,000 points. And so on and so forth. So something like. Ten to the D, where D is the number of dimensions.
- This isn't just an issue for KNN, this is true in general. Don't think about this now as nearest neighbors in the sense of KNN. But think of it as points that are representing or covering the space. And if you want to represent the same sort of hyper-volume of space as you add dimensions, you're going to have to get exponentially more points in order to do that. And coverage is necessary to do learning.
- Really problematic because it's very natural to just keep throwing dimensions into a machine learning problem. Like it's having trouble learning. Let me give it a few more dimensions to give it hints. But really what you're doing is just giving it a larger and larger volume to fill.
- And it can't fill it unless you give it more and more data. So you're better off giving more data than you are giving more dimensions.

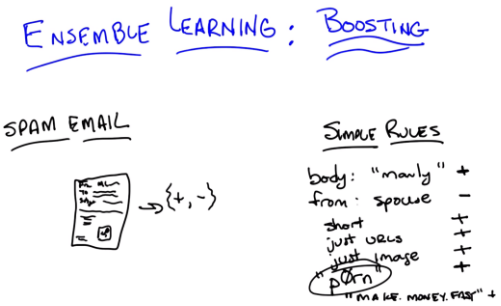
4.9. Some Other Stuff



- The other stuff that comes up in KNN mainly comes up in these assumptions we make about parameters to the algorithm.
- The one we talked about probably the most is our distance measure
- We looked at Euclidean and we looked at Manhattan. We even looked at weighted versions of those.
- Your choice of distance function really matters. If you pick the wrong kind of distance function, you're just going to get very poor behavior.
- It's probably worth pointing out that this notion of weighted distance is one way to deal with the curse of dimensionality. You can weigh different dimensions differently.
- Also. instead of just taking a weighted average, what about using a distance matrix to pick up some of the points, and then do a different regression on that substantive point. Replace this whole notion of average with a more regression-y thing.
    - This actually has a name, it's actually called locally weighted regression and actually works pretty well and in place of sort of averaging function, you can do just about anything you want to. You could throw in a decision tree, you could throw in a neural network, you could throw in lines do linear regression. You can do, almost anything that you can imagine doing.

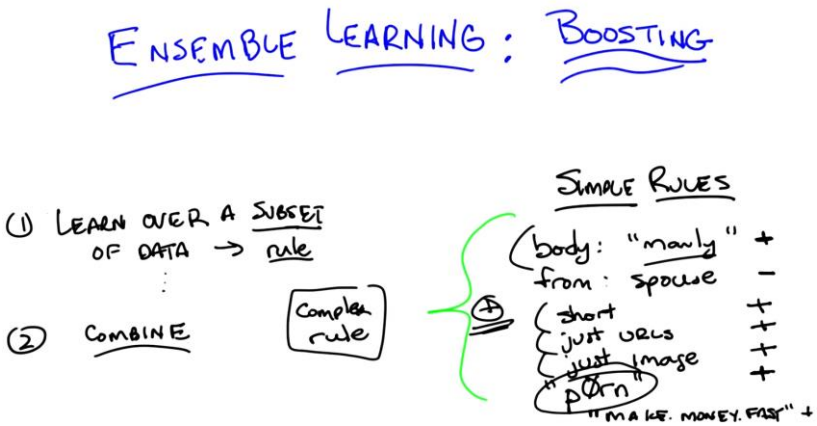
5. Ensemble Learning and Boosting

5.1. Ensemble Learning Boosting



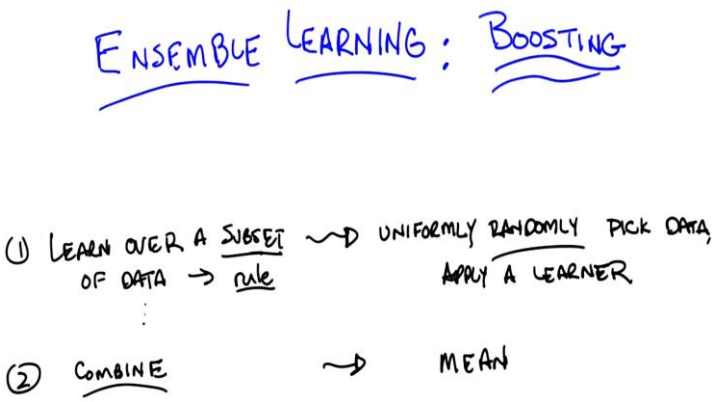
- I want to start this out by going through a little exercise with you: spam email.
- Normally we think of this as a classification task, where we're going to take some email and we're going to decide if it's spam or not. Given what we've talked about so far, we would be thinking about using a decision tree, neural networks, or KNN.
- I want to propose an alternative which is going to get us to ensemble learn: I don't want you to try to think of some complicated rule that you might come up with that would capture spam email. Instead, I want you to come up with some simple rules that are indicative of spam email.
- You're going to get some email message and you want some computer program to figure out automatically for you if something is a piece of spam or it isn't. And I want you to help write a set of simple rules that'll help indicate that something is spam.
  - If it comes from my spouse, it's probably not spam.
  - The length of the message. some spam is very, very short, like just the URL.
  - It's just an image.
  - Lots of misspelled words, a blacklist of words.
- There are tons of these rules that can be thought of. We could come up with a bunch of them.
- Something they all kind of have in common, all of them are sort of right, they're useful but no one of them is going to be very good at telling us whether a message has spam on its own. Any one is evidence but it's not enough to decide whether something is spam or not.
- In decision trees, there's a similar problem going on there. We can think of each of the nodes in a decision tree as being a very simple rule and the decision tree tells us how to combine them. So, we need to figure out how to do that here and that is the fundamental notion of ensemble learning.
- You could also something similar with other ML methods, like neural networks, where each of these rules becomes a feature and we're just trying to learn ways to combine them all together. The difference here in this case is that typically with the new network we've already built the network itself and the nodes and we're trying to learn the weights, whereas in something like a decision tree you're building up rules as you go along. And typically with ensemble learning you're building up a bunch of rules and combining them together until you got something that's good enough.

5.2. Ensemble Learning Simple Rules



- First you take a bunch of simple rules, all of which kind of make sense and that you can see as sort of helping, but on their own, individually, don't give you a good answer.
- Then you magically combine them in some way to create a more complex rule, that in fact, works really well.
- Ensemble learning algorithms have a sort of basic form to them that can be described in just one or two lines.
- The basic form of an ensemble learning algorithm:
  - Learn over a subset of the data, and that generates some kind of a rule.
    - Then you learn over another subset of the data and that generates a different rule. Rinse repeat for a third. fourth rule, and yet a fifth rule, and so on and so forth.
  - Eventually you take all of those rules and you combine them into one of these complex rules.
- In the email case I might look at a small subset of email that I know is already spam and discover that the word manly shows up in all of them and therefore pick that up as a rule. That's going to be good at that subset of mail, but not necessarily be good at the other subset of mail.
- I can do the same thing and discover that a lot of the spam mails are in fact short or a lot of them are just images or just URLs and so on and so forth.
- That's how I learn these rules -- by looking at different subsets. Which is why you end up with rules that are very good at a small subset of the data, but aren't necessarily good at a large subset of the data.
- And then after you've collected these rules, you combine them in some way
- Think of this as any other classification learning problem that you would have where you're trying to come up with some way to distinguish between the positives and the negatives.
- We look at only a subset for each rule because if we look at all of the data then it's going to be hard to come up with these simple rules. This will be discussed more later when we talk about overfitting
- This is Ensemble Learning: You learn over a subset of the data over and over again, picking up new rules, and then you combine them and you're done.

5.3. Ensemble Learning Algorithm

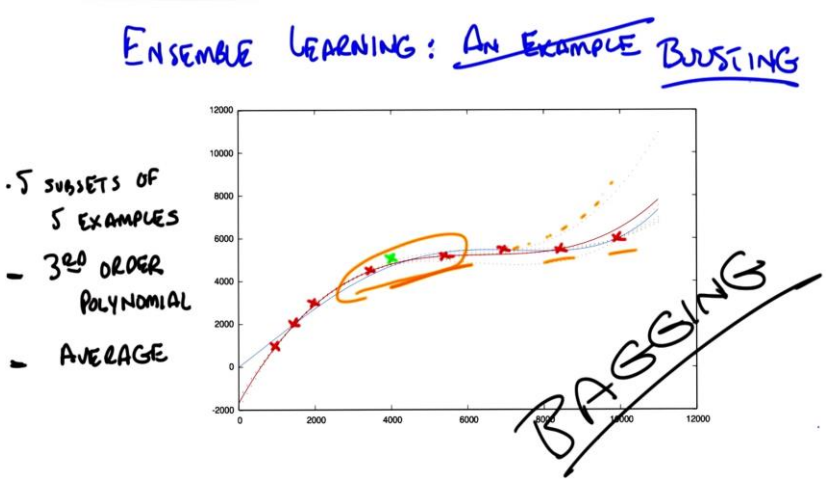


- Here's kind of the dumbest (or simplest) thing you can imagine doing, and it turns out to work out pretty well.
  - We're going to pick subsets uniformly, just to be specific about it.
  - Uniformly randomly choose among some of the data, and say that's the data I'm going to look at, and then apply some learning algorithm to it.
- Okay, so just: pick a subset of the data, apply a learner to it, get some hypothesis out, and get some rule out, and now I'm going to combine them.
- Since we're being simple, try doing something simple for combining. So, a very simple way of combining, would be to average them. We'll simply take the mean. Each one of them learned over a random subset of the data. You have no reason to believe that one's better than the other.
- There's a couple of reasons this 'idea' could go wrong i.e. bad random subset.

5.4. Quiz: Ensemble Learning Outputs

- You've got N data points. The learner that you're going to use over your subsets is a 0-order polynomial. The way you're going to combine the output of the learners is by averaging them.
- Your subsets are going to be constructed in the following way: You uniformly randomly picked them and you ended up with N disjoint subsets, and each one has a single point in it that happens to be one of the data points.
- If you look on the left of the slide you've got a graph of some data points, each being a subset
- When you do your ensemble learning, you learn all these different rules and then you combine them. What is the output going to be? What does the ensemble output? I want a description and if the answer's a number, that's a perfectly fine description. But I'll give you a hint, it's a short description.
- So, the ensemble outputs the average or mean. A constant. Which happens to be the mean of the data. We've seen this before when we are outputting very simple hypotheses. This is what happens if you do an unweighted average with k-NN where k equals n.
- Therefore, we should probably do something a little smarter than this then.

5.5. Ensemble Learning An Example

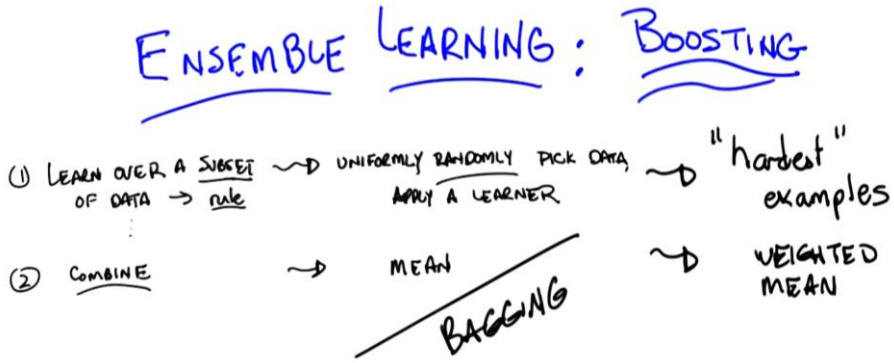


- Looking at the same housing data that we've looked at a couple of times before. You'll notice that I marked one of them as green
- I'm going to take the housing data you've got, I'm going to do some ensemble learning on it. And I'm going to hold out the green data point. So of the nine data points, you're only going to learn on 8 of them. And I'm going to add that green data point as my test example and see how it does. This is a cross-validation. Or you could just say, I just put my training set and my test set on the same slide.
- The first thing I'm going to do is pick a random subset of these points. I'm going to pick five points randomly and I'm going to do that five times, so I'm going to have five subsets of five examples. I'm going to choose these randomly, and I'm going to choose them with replacement, so we're not going to end up in the situation we ended up in just a couple of minutes ago where we never got to see the same data point twice.
- So five subsets of five examples, and then I'm going to learn a 3rd-order polynomial. And I'm going to take those 3rd-order polynomials, I'm just going to learn on that subset, and I'm going to combine them by averaging.
- Here's what you get: I'm showing you a plot over those same points, with the five different 3rd-order polynomials. As you can see they're kind of similar. But some of them sort of veer off a little bit because they're looking at different data points. One of them is actually very hard to see. It veers off because just, purely randomly, it never got to see the two final points. But they all seem to be pretty much in agreement between points three and four. There's a lot of consistency there.
- So the question now becomes: how good is the average of these compared to something we might have learned over the entire data set?
- What you're looking at now is the red line: the average of all of five of those 3rd-order polynomials.
- The blue line is the 4th-order polynomial that we learned when we did this with simple regression
- You actually see they're pretty close.
- It turns out that on this data set (and I did this many, many, many times just to see what would happen with many different random subsets):
    - Typically the blue line always does better on the training set (the red points) than the red line does.

- The red line almost always does better on the green point on the test set or the validation set.

- It learns an average of 3rd-degree polynomials, 3rd-order polynomials, which is itself a third order polynomial. It does better by doing this than just learning a 3rd-order polynomial directly.
- The danger is often overfitting, overfitting is like the scary possibility. Mixing the data up in this way and focusing on different subsets of it somehow manages to find the important structure as opposed to getting misled by any of the individual data points.
- It's basically the same kind of argument you make for cross-validation. In practice, this particular technique of ensemble learning does quite well in getting rid of overfitting.
- This particular version, where you take a random subset and you combine by the mean, is called **bagging**.
- You'll notice it's not what I said we were going to talk about during today's discussion. I said we were going to talk about boosting. So we're talking about bagging but we're going to talk about boosting. The reason I wanted to talk about bagging is because it's really the simplest thing you can think of and it actually works remarkably well. But there are a couple of things that are wrong with it, or a couple of things you might imagine you might do better that might address some of the issues and we're going to see all of those when we talk about boosting right now.

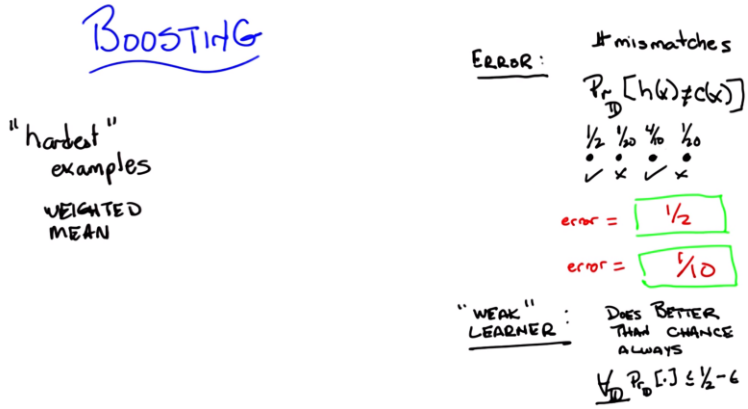
5.6. Ensemble Boosting



- Let's go back and look at our two questions we were trying to answer.
  - We've answered the first one -- learn over a subset of data and define a rule -- by choosing that subset uniformly randomly and applying some learning algorithm.
  - We answered the second question -- how do you combine all of those rules of thumbs -- by saying, you simply average them. And that gave us bagging.
- There's an alternative to the first question (we'll leave open the second one for a moment) that's going to get us to what we're supposed to be talking about today, which is boosting.
- Rather than choosing uniformly randomly over the data, we should try to take advantage of what we are learning as we go along, and instead of focusing just kind of randomly, we should pick the examples that we are not good at.
- We should pick a subset based upon whether the examples in that subset are hard. Certainly when I'm trying to learn a new skill, I'll spend most of my energy on the stuff that I'm kind of on the edge of being able to do, not the stuff that I've already mastered.
- So that answers the first question. We're going to look at the hardest examples (I'm going to define for you exactly what hard means). I'm going to have to introduce at least one technical definition, but I want to make certain you have that.

- And the second one, the combining, well that's a difficult and sort of complicated thing, but at a high level, I can explain it pretty easily by saying we are going to still stick with the mean.
    - We are going to do a weighted mean because the basic idea is to avoid the sort of situations that we came across when we looked at the data before, where taking an average over a bunch of points that are spread out just gives you an average or a constant that doesn't give you a lot of information about the space.
    - So we're going to weight it by something, and it's going to turn out the way we choose to weight it will be very important.

5.7. Quiz: Ensemble Boosting



- The whole goal of what we're going to add for boosting here is we're going to expand on this notion of hardest examples and weighted mean.
- How have we been defining error so far? Usually we take the squared difference between the correct labels and what's produced by our classifier or regression algorithm. That is how we've been using error when we're thinking about regression error.
- How about a notion of accuracy- how good we are at classifying examples? Sticking with classification for a moment, that would be the same as squared error, except that it's doesn't really need to be squared. That is to say, if the outputs are zeroes and ones, the squared error is just whether or not there's a mismatch. So it could just be the total number of wrong answers.
- What we've been doing so far is counting mismatches. We might define an error rate or an error percentage as the total number of mismatches over the total number of examples. But implicit in that is the idea that every single example is equally important. That's not always the case.
- Learning only happens if your training set has the same distribution as your future testing set. And if it doesn't, then all bets are off and it's very difficult to talk about induction or learning. That notion of distribution is implicit in everything that we've been doing so far, and we haven't really been taking it into account when we've been talking about error.
- So here's another definition of error.
    - The subscript D stands for distribution. We don't know how new examples are being drawn, but however they're being drawn, they're being drawn from some distribution D
    - H is our old friend, the hypothesis. That's the specific hypothesis that our learner has output. That's what we think is the true concept
    - C is whatever the true underlying concept is.
    - I'm going to define error as the probability, given the underlying distribution, that I will disagree with the true concept on some particular instance x.

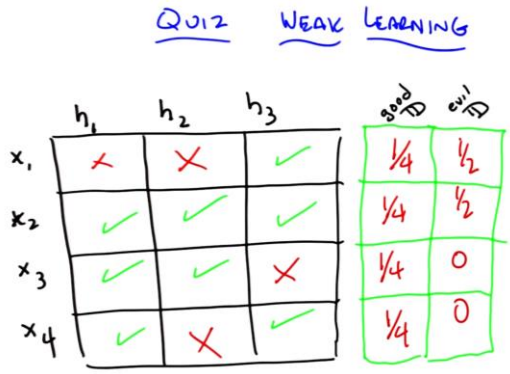
- Let me give you a specific example. I'm going to draw four possible values of x. And when I say I'm going to draw four possible values of x, I mean I'm just going to put four dots on the the screen.
- Then I'm going to tell you that this particular learner outputs a hypothesis. a potential function, that ends up getting the first one and the third one right, but gets the second and the fourth one wrong.
- Half of them are right and half of them are wrong. So, the number of mismatches, is, 2 out of 4 or a half. However, it assumes that you're likely to see all four of the examples equally often. However, that may not in fact be the case.
- Here's another example of error for you. What if each of the points is likely to be seen in different proportions. So you're going to see the first one half the time. You're going to see the second one 1/20th of the time. You're also going to see the fourth one 1/ 20th of the time and the third one, 4/10ths of the time, what is the error rate now?
- Take into consideration those probabilities, the number of mismatches is half, but the actual number of errors, the expected number of errors is 90% correct, 10% error.
- What's important to see here is that even though you may get many examples wrong, in some sense some examples are more important than others because some are very rare. And if you think of error, or the sort of mistakes that you're making, not as the number of distinct mistakes you can make, but rather the amount of time you will be wrong, or the amount of time you'll make a mistake, then you can begin to see that it's important to think about the underlying distribution of examples that you see.
- The notion of error turns out to be very important for boosting because, in the end, boosting is going to use this trick of distributions in order to define what hardest is. Since we are going to have learning algorithms that do a pretty good job of learning on a bunch of examples, we're going to pass along to them a distribution over the examples, which is another way of saying, which examples are important to learn versus which examples are not as important to learn.
- That's where the *hardest* notion is going to come in. So, every time we see a bunch of examples, we're going to try to make the harder ones more important to get right than the ones that we already know how to solve.
- **Weak learner** - a learning algorithm, which is what we mean by a learner here, that is actually fairly straightforward, that means no matter what the distribution is over your data, will do better than chance when it tries to learn labels on that data.
    - No matter what the distribution over the data is, you're always going to have an error rate that's less than 1/2.
    - What that means, sort of as a formalism, is written down here below:

$$\forall_D Pr_D[.] \leq \frac{1}{2} - \varepsilon$$

        - For all D, that is to say no matter what the distribution is, your learning algorithm will have an expected error (the probability that it will disagree with the true actual concept if you draw a single sample) that is less than or equal to ε (epsilon).
        - **Epsilon** - ε - is a term that you end up seeing a lot in mathematical proofs, particularly ones involving machine learning.
        - Epsilon just means a really, really small number somewhere between a little bigger than 0 and certainly much smaller than 1.

- Here what this means technically is that you're bounded away from 1/2. Another way of thinking about that is you always get some information from the learner. The learner's always able to learn something. Chance would be the case where your probability is 1/2 and you actually learn nothing at all which kind of ties us back into the notion of information gain way back when with decision trees.

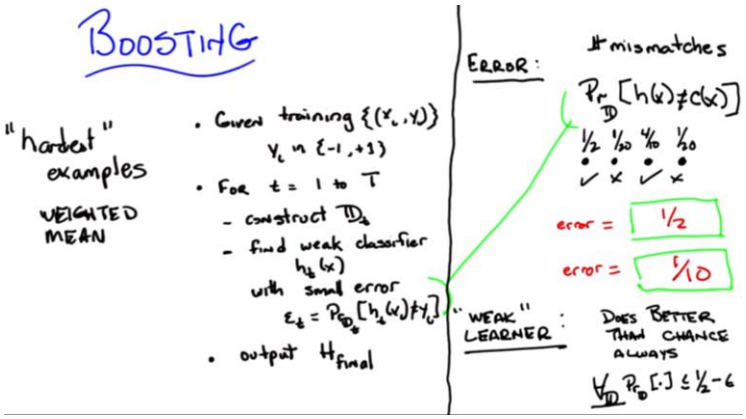
5.8. Quiz: Weak Learning



- Here the entire hypothesis space consists only of these three hypotheses and the entire instance space consists entirely of only four examples
- I have an x in a square if that particular hypothesis does not get the correct label for that particular instance and I have a green check mark if that particular hypothesis does in fact get the right label for that example.
- I want you to come up with the distribution over the 4 different examples, such that a learning algorithm that has to choose between one of those hypotheses will in fact be able to find one that does better than chance (by having an expected error greater than 1/2).
- Then if you can do that, I want you to see if you can find a distribution which might not exist, such that if you have that distribution over the four examples, a learning algorithm that only looked at H1, H2 and H3 would not be able to return one of them that has an expected error greater than 1/2.
- You always need to have some distribution over your examples to really know what your expected error is.
- All zeros means no such distribution. So if you put in all zeros you're saying no such distribution exists. But otherwise it should add up to one down each of the columns.
- Good distribution, put equal weight on X1, X2, X3, and X4 then H1 gets three out of four correct, that's 3/4. That's better than 1/2. Then fill that in the good boxes, quarters all the way down.
- Evil distribution, looking at X1 if you put all the weight on H1 it does very badly, 100% error, H2 is 100% error. But H3 is 0% error. So putting all the weight on X1 is no good. And if you look X2, X3, and X4, they all have the property that there's always a hypothesis that gets them right. So I started to think, well, maybe there isn't an evil distribution. And then I kind of lucked into putting 1/2 on both the first and the second one because I figured that that ought to work, but then I realized that's an evil distribution because if you choose H1, H2, or H3, they all have exactly 50% error on that distribution.
- So what this tells us is, since there is a distribution for which none of these hypotheses will be better than chance, there is **no weak learner** for this hypothesis space, on this instance set.

- If we had more hypotheses and more examples and we had the X's and the Y's in the right places, then there'd be lots of ways to get weak learners for all the distributions because you'd have more choices to choose from. What made this one particularly hard is that you only had three hypotheses and not all of them were particularly good.
- Technically if you have a whole lot of hypotheses that are bad at everything, you're going to have a very hard time with a weak learner. And if you have a whole bunch of hypotheses that are good at almost everything, then it's pretty easy to have a weak learner.
- The lesson you should take away from this is: a weak learner is actually a pretty strong condition. You're going to have to have a lot of hypotheses that are going to have to do well on lots of different examples, or otherwise, you're not always going to be able to find one that does well no matter what the distribution is. So it's actually a fairly strong, and important condition.

5.9. Boosting In Code



- We're given a training set made up of a bunch of x_i , y_i pairs. x is your input and y is your output. For reasons that'll be clear in a moment, all of your labels are either -1 or 1, where -1 means not in the class and 1 means you're in the class. So this is a binary classification task.

  Given training $\{(x_i, y_i)\}$; y_i in $\{-1,1\}$
- Then loop at every time step, let's call it t, from the first time step 1 to some big time in the future, just call it T and not worry about where it comes from right now.

  For $t = 1$ to T
  - Construct a distribution, call that D_t, this is your distribution over your examples at some particular time t.

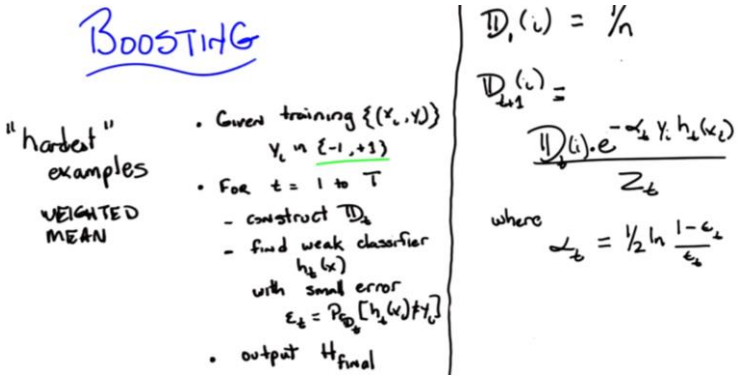
    Construct D_t
  - Given that distribution, find a weak classifier. Your weak learner should output some hypothesis. Let's call that h_t, the hypothesis that gets produced at that time step, and that hypothesis should have some small error. Let's call that error ε_t (epsilon sub t), because it's a small number, such that it does well on the training set, given the particular distribution, which is to say, that the probability of it being wrong (disagreeing with the training label) is small with respect to the underlying distribution. To be clear here, it doesn't have to be tiny. It could actually be almost ½, but not bigger than 1/2.

    find weak classifier $h_t(x)$ with small error $\varepsilon_t = Pr_{D_t}[h_t(x_i \neq y_i]$
- You're going to do that and you'll do that for a whole bunch of time steps, constantly finding hypotheses at each time step h_t with small error, constantly making new distributions, and then eventually, you're going to output some final hypothesis. I haven't told you yet how you're going to to get the final hypothesis, but that's the high level bit. Look at your training data, construct

distributions, find a weak classifier with low error, keep doing that you have a bunch of them, and then combine them somehow into some final hypothesis.

- Where do we get this distribution and where do we get this final hypothesis? Next slide...

5.10. The Most Important Parts



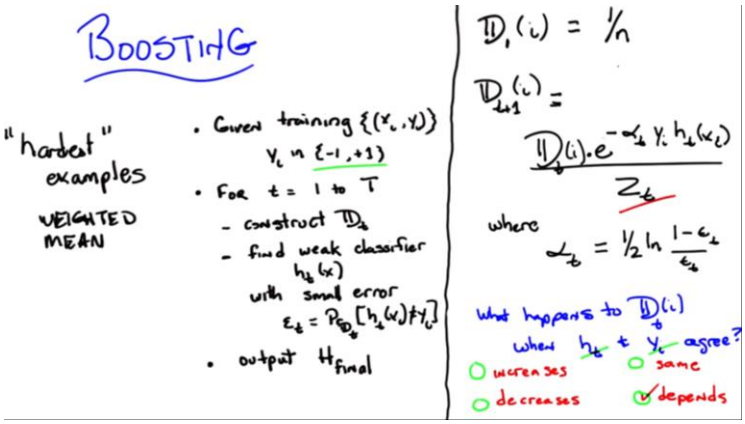
Let's start with the distribution.

- Let's start with the base case, and that is the distribution at the beginning of time, D_1.
  - This distribution is going to be over each of the examples and those examples are indexed over i. I'm simply going to set that distribution to be uniform. We'll call it n examples. I'm going to say that for every single example, they happen 1/n times, that is, a uniform distribution. We have no reason to believe, for the purposes of this algorithm, that any given example is better, more important, harder than any other example. I know nothing. See if you can learn over all of the examples.

    $$D_1(i) = \frac{1}{n}$$

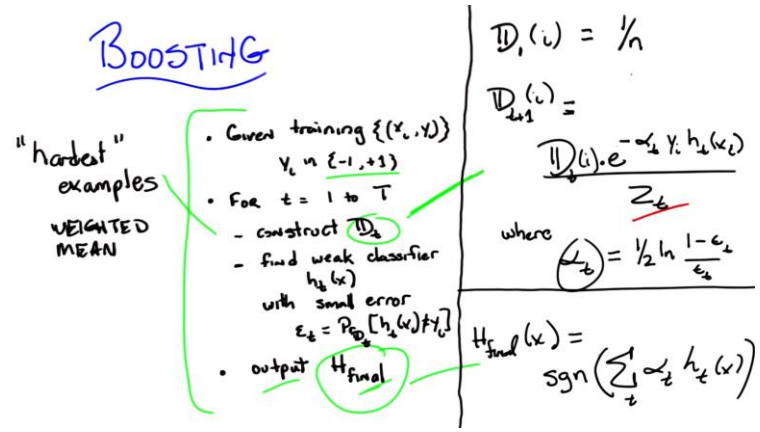
- So I start out with a uniform distribution because that's what you usually do when you don't know anything. At every time step t, I'm going to construct a new distribution, D_{t+1} to be the equation on the right of the slide
  - We know that D is our distribution and it's some number, where, over all the examples, it adds up to 1. It's a stand-in for how important a particular example is -- how often we expect to see it. And that's the trick that we're using with distributions.
  - I'm going to take the old distribution for a particular example and I'm going to either make it bigger or smaller, based upon how well the current hypothesis does on that particular example. The trick:
    - We know that h_t always returns a value of either -1 or +1 because that's how we define our training set. So, h_t is going to return -1 or +1 for a particular x_i.
    - y_i, which is the label with respect to that example, is also always going to be +1 or -1
    - $\alpha_t$is a constant (will get into later, just think of it as a number right now). It matters eventually. But right now, that number is always positive. It's a learning rate kind of thing
  - $y_i * h_i$ is going to be 1 if they agree, and -1 if they disagree. So, that means when they agree, that whole product will be positive, which means you'll be raising e to some negative number. When they disagree, that product will be negative, which means you'll be raising e to some positive number.

5.11. Quiz: When D agrees



- If y_i and h_t agree, that means they're both negative or they're both positive, so when we multiply them together, we get 1. 1 times whatever is a positive number is going to be positive. We're negating that, so it's negative. e to the negative power is something between zero and one, so that's going to scale it down. So, it looks like it could depend on the normalization.
- The z_t term is, in fact, whatever normalization constant you need at time t in order to make it all work out to be a distribution (sum to 1).
- Then it's not going to change. But if some of them are correct and some of them are incorrect, the ones that are correct are going to decrease. And the ones that are incorrect are going to increase.
- So the answer is that it depends. It depends on what else is going on. Decrease is what typically happens though.
- A similar question is what happens when they disagree and at least one other example agrees? Then that should increase.
- It's going to put more weight on the ones that it's getting wrong. The ones that it's getting wrong must be the ones that are harder. Or at least that's the underlying idea. And the ones that it's getting right, it puts less weight on those. The loop goes around again and it tries to make a new classifier.
- The ones that it's getting wrong are getting more and more weight, but we are guaranteed, or at least we've assumed, that we have a weak learner that will always do better than chance on any distribution, it means that you'll always be able to output some learner that can get some of the ones that you were getting wrong, right.

5.12. Final Hypothesis

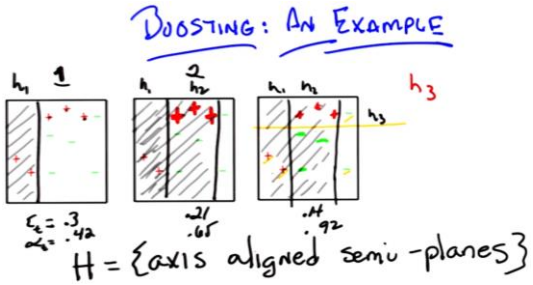


- That ties together what constructing D does for you, and connects it to the hardest examples.
- That gets us to a nice little trick where we can talk about how we actually output our final example.
- So, the way you construct your final example is basically by doing a weighted average, and the weight is going to be based upon this α_t.
- So the final hypothesis is just the sgn function of the weighted sum of all of the rules of thumb, all of the weak classifiers that you've been picking up over all of these time steps, where they're weighted by the α_t's.

  $$H_{final}(x) = sgn(\sum_t \quad \alpha_t h_t(x))$$

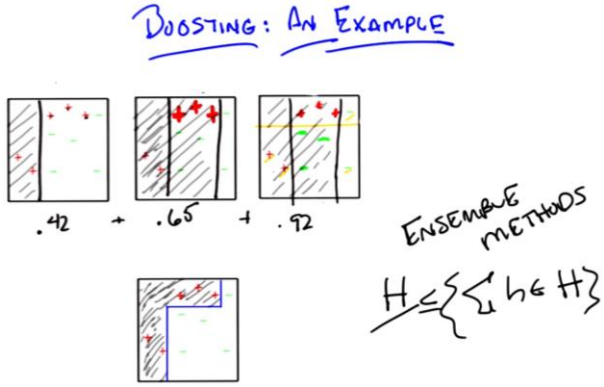
- where $\alpha_t = \frac{1}{2} ln \frac{1-\varepsilon_t}{\varepsilon_t}$ That is to say, α_t is a measure of how well you're doing with respect to the underlying error.
  - So, you get more weight if you do well than if you do less well, where you get less weight.
  - Use natural logs because you're using exponentials and that's always a cute thing to do
- So what does this look like to you? Well, it's a weighted average based on how well you're doing or how well each of the individual hypotheses are doing and then you pass it through a thresholding function where
  - If it's below zero, you say negative
  - If it's above zero, you say positive
  - If it's zero, you just throw up your hands and return zero.
- In other words, you return the sign of the number. So you are throwing away information there, (we into this information in the next lesson, it's going to turn out that this little bit of information is actually pretty important).
- This is boosting. There's really nothing else to it. You have a very simple algorithm which can be written down in a couple of lines. The hardest parts are constructing the distribution and then simply bringing everything together

5.13. Three Little Boxes



- I've got three little boxes on the screen. Now, they're the same boxes. I've drawn them up here beforehand because I'm going to solve this problem in only three steps.
- So just pay attention to the first box for now. You have a bunch of data points; red pluses (+) and green minuses (-) with the appropriate labels and they all live in this part of the plane. We want to figure out how to be able to correctly classify these examples.
- We have to specify what our hypothesis space is. So the hypothesis space is the set of axis-aligned semi-planes. For the purpose of this example this means I'm going to draw a line, either horizontal or vertical, and say that everything on one side of that line is positive and everything on the other side of that line is negative.

- They’re axis aligned because it's only horizontal and vertical, and they're semi-planes because the positive part of it is only in part of the plane.
- I'm going to just walk through what boosting would/might end up doing with this particular example given that you have a learner that always chooses between axis-aligned semi-planes.
- Step 1. All of the examples look the same because we have no particular reason to say any are more important than the other or any are easier or harder than the other. And that's just the algorithm we had before. We run through and we ask our learner to return some hypothesis that does well in classifying the examples. It turns out that though there are many, and in fact, there are an infinite number of possible hypotheses you could return, one that works really well is one that looks like a vertical line that separates the first two data points from the rest.
- What I'm saying here is that everything to the left of this line is going to be positive and everything to the right is going to be negative. It gets correct (correctly labeled positive) the two plusses to the left. Iit gets correct all of the minuses as well. But it gets wrong the three plusses on the right side.
- So I'm just going to ask you to trust me here but it turns out that the specific error here is 0.3 and if you stick that into α you end up with 4.2.
- Step 2. The ones that it got right should get less weight and the ones that it got wrong should get more weight
- The this learner output by putting a line to the right of the three plusses, because he's gotta get those right in saying that everything to the left is in fact, positive. It gets the three that you were doing poorly on right, in step 1 and it picks up still the other two which it was getting right. And it gets wrong these three minuses which aren't worth so much.
- So the error of this step by the way, turns out to be 0.21 and the α at this time step turns out to be 0.65.
- Step 3: The ones that got it wrong should get pushed up in weight, the green minuses in the middle patch, they should become more prominent. The three plusses on the right should become less prominent than they were but it still might be more prominent than they were in the beginning (actually won’t because the α bigger so it will have actually a bigger effect on bringing it down), but it’ll still be more prominent than the other ones that haven't been bumped, which will get smaller again, they're really going to disappear.
- Step 3: separate the 3 plusses on the right

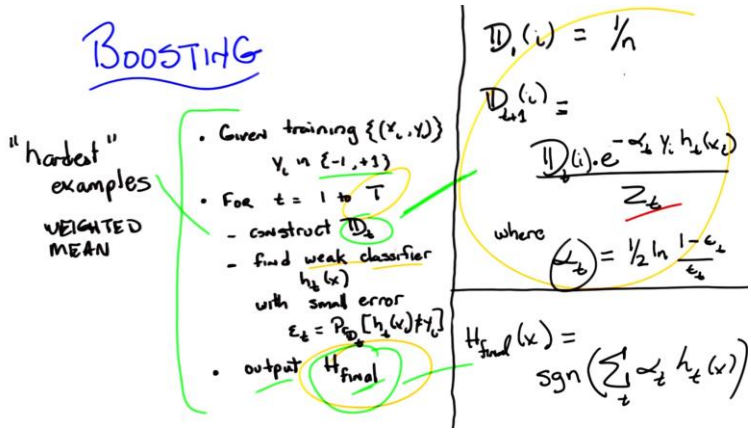


- If you look at these three hypotheses and their weights, you end up with something kind of interesting. If you look at this third hypothesis that's chosen here, it turns out to have a very low error (that is decreasing) of 0.14, and it has a much higher α of 0.92. Now if you look at these weights and you add them up, you end up with a cute little combination. So, let me draw that

for you. If you take each of the three hypotheses that we produced and you weight them accordingly, you end up with the bottom figure.

- Even though we were only using axis-aligned semi-planes for all the weak learners, at the end of the day, it actually kind of bent the line around and captured the positive and negative examples perfectly.
- If you try to look at just some particular hypothesis class H, because you're doing weighted averages over hypotheses drawn from that hypothesis class, this hypothesis class is at least as complicated as this hypothesis class and often is more complicated. So you're able to be more expressive, even though you're using simple hypotheses, because you're combining them in some way.

5.14. Good Answers



This slide is more of a conversation, left most of it as is...

- Boosting basically says, if I have some examples that I haven't been able to classify well, I'm going to re-rate all my examples so that the ones I don't do well on become increasingly important. That's what boosting does.
- That's what this whole bit of D is all about. It's all about re-weighting based on difficulty and hardness.
- We know that we have the notion of a weak learner. That no matter what happens for whatever distribution, we're always going to be able to find some hypothesis that does well.
- So, if I'm trying to understand why boosting in the end, why the final hypothesis that I get at the end, is going to do well, I can try to get a feeling for that by asking, well, under what circumstances would it not do well? So, if it doesn't do well, then that means there has to be a bunch of examples that it's getting wrong.
- So how many things could it not get right? How many things could it misclassify? How many things could it get incorrect? That that number has to be small. There cannot be a lot of examples that it gets wrong.
- So, here's my reasoning, let's imagine I had a number of examples at the end of this whole process. I've done it T times. I've gone through this many times and I have some number of examples that I'm getting wrong. If I were getting those examples wrong, then I was getting them wrong in the last time step, right? And, since I have a distribution and I re-normalize, and it has to be the case that at least half of the time, I am correct, the number of things I'm getting wrong has to be getting smaller over time. Because let's imagine that I was at a stage where I had a whole bunch of them wrong. Well, then I would naturally renormalize them with a distribution so that all of those things are important. But if they were all important, the ones that I was getting wrong, the next time I run a learner, I am going to have to get at least half of them right, more than half of them are right.
- Why can't it just be the case that the previous ones which were getting right start to get more wrong as we shift our energy towards the errors?
- What goes on is that you get this exponentially aggressive weighting over examples. And you're driving down the number of things you get wrong sort of exponentially quickly, over time. That's why boosting works so well and works so fast.
- I don't get why that's causing us to get fewer things wrong over time. In your example that you worked through, that had the error in the alphas and the errors kept going down and the alphas kept going up. Is that necessarily the case?

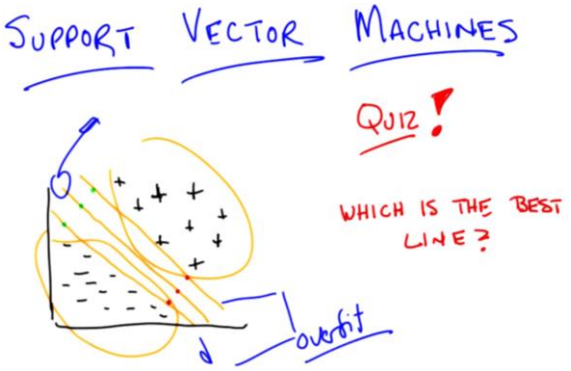
- Well, what would be the circumstances under which it isn't the case? How would you ever go back and forth between examples? Over time, every new hypothesis gets a vote based upon how well it does on the last difficult distribution. So even if the ones that you were getting right, you start to get wrong, if you get them increasingly wrong, that error's going to go down and you're going to get less of a vote.
- I feel like the error should be going up, because we're asking it harder and harder questions as we go.
- Even though we're asking it harder and harder questions, it's forced to be able to do well on those hard questions. It's forced to, because it's a weak learner. That's why having a weak learner is such a powerful thing.
- But why couldn't we on iteration 17 have something where the weak learner works right at the edge of its abilities and it just comes back with something that's 1/2 minus epsilon.
- That's fine. But it has to always be able to do that. If it's 1/2 minus epsilon, the things it's getting wrong will have to go back down again.
- Why would the error go down each iteration?
- Well, it doesn't have to, but it shouldn't be getting bigger.
- Why shouldn't it be getting bigger?
- So, imagine the case that you're getting. You are working at the edge of your abilities. You get half of them right roughly and half of them wrong. The ones you got wrong would become more important, so the next time around you're going to get those right versus the other ones. So you could cycle back and forth I suppose, in the worst case, but then you're just going to be sitting around, always having a little bit more information. So your error will not get worse, you'll just have different ones that are able to do well on different parts of the space. Right? Because you're always forced to do better than chance. So.
- Yeah but that's not the same as saying that we're forced to get better and better each iteration. I don't see that, that property just falling out.
- Well, I don't see it falling out either, but then I haven't read the proof in like seven, eight, nine years.
- So we generate a new distribution. What is the previous classification error on this distribution? I mean, if it were the case that we always return the best classifier then I could imagine trying to use that but…
- Well we, well we don't, we don't require that.
- Yeah, I mean, it's just finding one that's epsilon minus, or 1/2 minus epsilon.
- Right, so let's, let's see if we can take the simple case, we got three examples, right, and you're bouncing back and forth and you want to construct something so that you always do well on two of them. And then poorly on one, kind of a thing, and that you keep bouncing back and forth. So let's imagine that you have one-third, one-third, one-third, and your first thing gets the first two right and the last one wrong. So you have an error of a third. And you make that last one more likely and the other two less likely. Suitably normalized, right?
- Yep.
- So now, your next one, you want to somehow bounce back and have it decide that it can miss, so let’s say you missed the third one. So you, you get the third one right. You get the second one right but you get the first one wrong. What's going to happen? Well, three is going to go down. You'll have less than a third error because you had to get one of the ones you were getting right wrong, you had to get the one you were getting wrong right. So your error is going to be, at least

in the example I just gave, less than a third. So, if your error is less than a third, then the weighting goes up more. And so, the one that you just got wrong doesn't go back to where it was before. It becomes even more important than it was when you had a uniform distribution. So the next time around, you have to get that one right, but it's not enough to break 1/2. So you're going to have to get something else right as well, and the one in the middle that you were getting right isn't enough. So you'll have to get number three right as well.

- Interesting.
- Right? And so, it's really hard to cycle back and forth between different examples, because you're exponentially weighting how important they are. Which means, you're always going to have to pick up something along the way. Because the ones that you coincidentally got right two times in a row become so unimportant that it doesn't help you to get those right, whereas the ones that you've gotten wrong, in the past, you've got to, on these cycles, pick up some of them in order to get you over 1/2.
- Mmm
- And so, it is very difficult for you to cycle back and forth.
- Interesting.
- And that kind of makes sense, right? If you think about it in kind of an information gain sense, because what's going on there is you're basically saying you must pick up information all the time.
- Hm. You are kind of non-linearly using that information in some way. So that kind of works. It makes some sense to me, but I think that in the end what has to happen is there must be just a few examples in a kind of weighted sense that you're getting wrong. And so if I'm right, that as you move through each of these cycles, you're weighting in such a way that you have to be picking up things you've gotten wrong in the past. So in other words, it's not enough to say, only the things that are hard in the last set are the ones that I have to do better. You must also be picking up some of the things that you've gotten wrong earlier more than you were getting them right because there's just not enough information in the one's that you're getting right all the time, because by the time you get that far along, the weight on them is near zero and they don't matter.
- Interesting.
- And then if you say, well, Charles, I could cycle back by always getting those wrong, yes, but then if you're getting those wrong, they're going to pull up and you're going to have to start getting those right too. And so, over time, you've gotta not just pick out things that do better than a half but things that do well on a lot of the data. Because there's no way for all of the possible distributions for you to do better than chance otherwise.

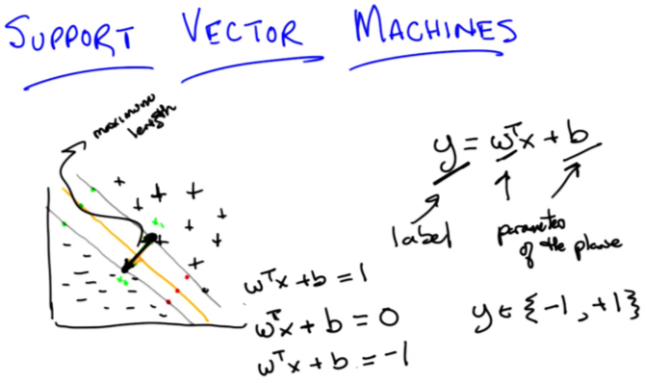
6. Kernel Methods and SVMs

6.1. Quiz: The Best Line



- All lines fit the sample data but only the middle line is the best.
- Since this is just a sample from the population, we don't know what's going to happen really close to the lines closer to pluses/minuses once you add test data.
- Otherwise you're believing the training data that you've got too much. You've decided that all of these boundaries are fine because of the specific set of training data that we've got. And while you want to believe the data, because that's all you've got, you don't want to believe the data too much.
- That's overfitting. So, the problem with those two lines, or one way to think about the problem with those two lines, is that these lines are more likely to overfit because they're believing the data too much.
- This line, or the line it's intended to represent anyway, is the line that is consistent with the data while committing least to it.
- Overfitting up to this point, we were generally talking about overfitting as being something where there's a great deal of model complexity in some sense.
- And it doesn't seem like those lines that are closer to the pluses or closer to the minuses are inherently more complex, they're still lines.
- It's a more literal interpretation of the words over and fit, you have decided to fit the data, and you believe it too much, and what you really want to do is commit. The least that you can commit to the data while still being consistent with it, finding the line of least commitment in the linear separable set of data, is the basis behind support vector machines.

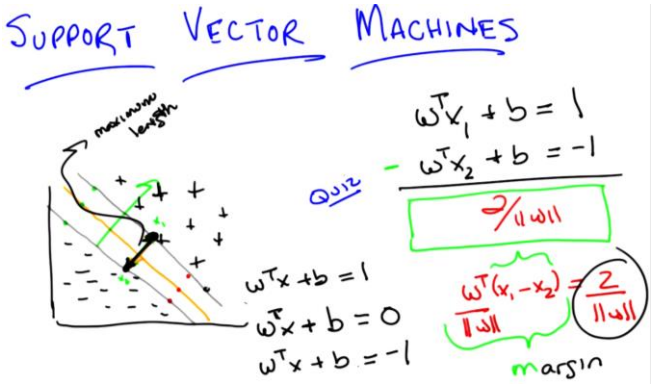
6.2. Support Vector Machine



- I'm going to try to encapsulate, what we just talked about, the line of least commitment, by drawing another line.
- If we think of this top gray line here as sort of the line that gets as close to the plus points as possible, without crossing over them and mis-classifying them, and we think of the bottom gray line as the one that gets as close as possible to the minus signs without crossing over them and misclassifying them, and then the middle line is sort of in the happy medium.
- What you really want is that, somehow, this distance between these lines is as big as possible. You want to have a line that leaves as much space as possible from the boundaries.
- Here even though we're going to be drawing with lines, we really want to deal with the general case, where we're talking about hyperplanes. Generally, when we write about hyperplanes, we describe them as some output, let's just call it $y = w^T x + b$. Here, because of what we're trying to do with classification:
  - The output y is going to be some value that indicates whether you're in the positive class or you're in the negative class.
  - w represents the parameters for our plane along with b, which is what moves it out of the origin.
- y here is going to be our classification label. Whenever we're talking about using a linear separator what we've been talking about is that you are taking some new point, projecting it onto the line, and then looking at the value that comes out from projecting it.
- In this case we want positive values to mean yes, you are part of the class, and negative values to mean that you aren't a part of the class.
- This is now what our linear classifiers actually look like, even in multiple dimensions with hyperplanes.
- Let's take that and and push it to the next level. Let's figure out exactly what we would expect the output of our hyperplane to be in this example that I've drawn on the screen here. So, we know we want to find this orange line in the middle, which has the property that it is your decision value. It tells you whether you are in the positive class or negative class, but also, it has the property of being as far away from the data as possible while still being consistent with it.
- If you're on the decision boundary for this particular line that's where it's kind of not sure if it's positive or negative, so that should be zero. $0 = w^T x + b$
- One question we can ask ourselves then, if we look at these other lines is, what's the equation for the other gray lines that are right at our positive or negative examples?
- Just like we did with boosting, let's say that our labels are always going to be from the set {-1, +1}. We know that our labels are -1 and +1, so we're going to take advantage of that fact by saying that the line that brushes up against the positive example should output +1 on the very first point that it encounters. That way the things that are kind of past the line are going to be +1 and the things before the line in kind of that demilitarized zone are going to be between zero and +1.
- The top gray line would be $1 = w^T x + b$.
- The bottom gray line would be $-1 = w^T x + b$
- This helps us in a very a simple way. We know that we want the boundary condition line, the one that is actually our decision boundary, to be as far as possible from both the positive and negative examples, so that would mean then that the distance between the two gray lines, which are parallel to that line, needs to also be maximized.

- We want this vector here to have the maximum length that we can have. Okay, so, how are we going to figure out how long that particular line is? So, here is a simple idea. Well, the lines are parallel to one another. We can pick points on that line to define that particular distance there. So, just because it's really easy to do the math, I'm going to chose a point here and a point here. Those points have the property, that, if I draw the line between them, you get a line that is perpendicular to the two gray lines. And I don't know what their respective x values are, so I'm just going to call, them x1 and x2, and that is going to define the two points that I have. The vector that is defined by their difference is in fact going to have the length that tells you how far apart those two lines are, which, in turn, because of the way that we've constructed them, tells you how far apart you're boundary decision line is from the data and we want that to be maximized because then we made the least commitment to the data.
- The equation for our positive line is $1 = w^T x_1 + b$. And all I've done there is substitute, some point -- I don't have to know what is, it's going to turn out -- that puts me in some particular place on that line. And similarly, I can do the same thing for my negative line and get $-1 = w^T x_2 + b$.
- Now, we want the distance between these two hyperplanes (or lines in this example) to be maximized. In order to figure out what that means, we need to know exactly what that line is. So it's the difference between the two.
- So, we can just use our favorite trick when we're doing systems of linear equations and just subtract the two lines. We basically have two equations and two unknowns. And, we simply subtract them from one another so that we can get a single equation that happens to represent the distance between them. So, if I subtract the two from one another, what do I get? (quiz next)

6.3. Quiz: Distance Between Planes

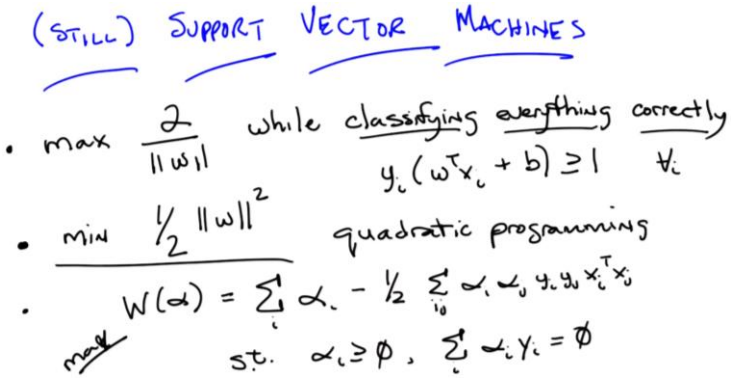


- I've got these two equations of two different hyperplanes, though they're parallel to one another because they have the same parameters. That is to say, I have two equations and two unknowns. I want to subtract them from one another and what we want you to do is we want you to solve for the line that is described by their difference.
- The output that I want you to figure out here is exactly what the distances between those two planes
- The distance between the two lines I feel that's just the norm of x1 minus x2. But the difference between these equations is going to be:

  $$w^T(x_1 - x_2) = 2$$
- The distance between x1 and x2, I still feel like it's just that norm of x1 minus x2. But I want you to tell it to me in terms of W because the only things we have to play with here are W and b. That's what defines our line and I want to find the right line, so I'd like to know something about the relationship between W and the distance between x1 and x2.
- If I divide it by W, that would be helpful because then x1 minus x2 would be . But you can't do that because W is a vector, and you can't really divide by a vector. We want to move w over from one side to the other. We could start doing all kinds of tricks with inverses, and with the inverse of a vector. There's all kinds of things that you could do, but actually the easiest thing way of doing it is getting rid of W on one side. And the easiest way to do that is to divide both sides by the length of W. So, rather than dividing both sides by W, we divide them by the length of W.
- So W divided by the length of W, is a normalized version of W. So it's like something that points in the same direction as W, but sits on the unit sphere. In fact it's a hypersphere. So we do that and that effectively is like giving you a value 1 because, like you said, it's a unit sphere. And so now we're actually talking about the difference between the vector x1 and the vector x2 projected onto the unit sphere and that's equal to $\frac{2}{||w||}$
- We've taken x1 minus x2 and projected it onto W. So it's like the length of x1 minus x2, but in the W direction.
- W actually represents a vector that's perpendicular to the line. And since we chose x1 and x2, their distance or the difference between them would in fact be perpendicular to the line. What we've just done is projected the difference between those two vectors onto something that is also perpendicular to the line. And so what that ends up giving us is, in fact, its length.

- The thing on left, not just where the braces are, that actually turns out to be the distance between the two hyperplanes, let's let's give that a letter and call it m. We're saying that equals 2 over the norm of W ($m = \frac{2}{||w||}$).
- So, if we want to maximize that the only thing that we have to play with is W and that is made larger and larger as W gets smaller and smaller, in other words, pushing it toward the origin.
- So set Ws to all zeroes, and we should be golden. Except if we push all the Ws to zero, we might not be able to correctly classify our points but what this does tell us is that we have a way of thinking about the distance of this vector and where the decision boundary ought to be.
- We want to find the parameters of the hyperplane such that we maximize this distance over here represented by this equation while still being consistent with the data, which makes sense because that's actually what we said in the first place.
- By the way, this thing has a name and it's the reason why I chose m -- it's called **the margin**, and what all of this exercise tells you is that your goal is to find a decision boundary that maximizes the margin, subject to the constraint that you actually want to correctly classify everything, and that is represented by that term.
- Now somehow, it feels like having gone through all this we ought to be able to use it for something and turn it into some other problem we might be able to solve so that we can actually find the best line. And it turns out we can do that.
- So, it turns out that this whole notion of thinking about finding the optimal decision boundary is the same as finding a line that maximizes the margin.
- And we can take what we've just learned, where we've decided the goal is to maximize and turn it into a problem where we can solve this directly.

6.4. Still Support Vector Machines



- We want to do somehow is maximize a particular equation, that is $\frac{2}{||w||}$. As a reminder, W are the parameters of our hyperplane. So somehow, we want to maximize that equation, subject to the constraints that we still classify everything correctly.
- Okay, so we want to maximize while classifying everything correctly.While classifying everything correctly is not a very mathematically satisfying expression, it turns out we can turn that into a mathematically satisfying expression. And let me show you how to do that. So here's a simple equation. While classifying everything correctly turns out to be the same as:

  $y_i(w^T x_i + b) \geq 1 \; \forall i$
- Well, what we really want is that the linear classifier, $w^T x_i + b$, is greater than or equal to 1 for the positive examples and less than or equal to -1 for the negative examples. But you cleverly multiply it by the label on the left-hand side, which does exactly that. If y_i is 1, it leaves it untouched. And if y_i is negative, it makes it less than or equal to minus 1. That's very clever.
- It turns out that trying to solve this particular problem, maximizing $\frac{2}{||w||}$, while satisfying that constraint, is a little painful to do. But we can solve an equivalent problem, which turns out to be much easier to do, and that is this problem. That is, rather than trying to maximize $\frac{2}{||w||}$, we can instead try to minimize $\frac{1}{2}||w||^2$
- The point that maximizes one will minimize the other because we took the reciprocal. As long as we're talking about positive things. And since these are lengths, they'll be positive. Taking the reciprocal changes the direction of what the answer is. And the squaring makes it monotone. It doesn't change the ordering of things.
- This is easier because when you have an optimization problem of this form, something like minimizing a W squared, subject to a bunch of constraints, that is called a quadratic programming problem. And people know how to solve quadratic programming problems in relatively straightforward ways.
- Now, what else is nice about that is a couple of things. One is, it turns out that these always have a solution, and in fact, always have a unique solution. Now, I am not going to tell you how to solve quadratic programming problems because I don't know how to do it other than to call it up in MATLAB. But there's a whole set of classes out there, where they teach you how to do quadratic programming. The important thing is that we have defined a specific optimization problem and that there are known techniques that come from linear algebra that tell us how to solve them. And we can just plug and play and go.

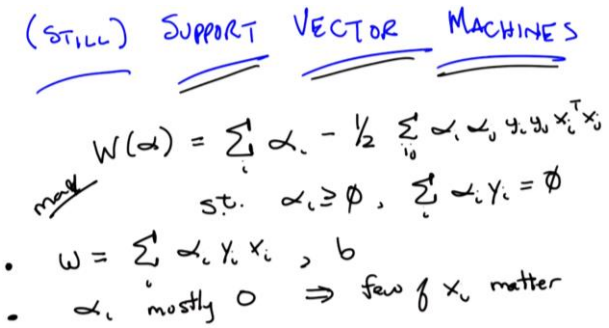
- In particular, it turns out that we can transform this particular quadratic programming problem into a different quadratic programming problem, actually, truthfully, into the normal form for a quadratic programming problem, that has the following form.

  max $\quad w(\alpha) = \sum_i \quad \alpha_i - \frac{1}{2}\sum_{ij} \quad \alpha_i \alpha_j y_i y_j x_i^T x_j$

  $s.t. \quad \alpha_i \geq 0, \sum_i \quad \alpha_i y_i = 0$

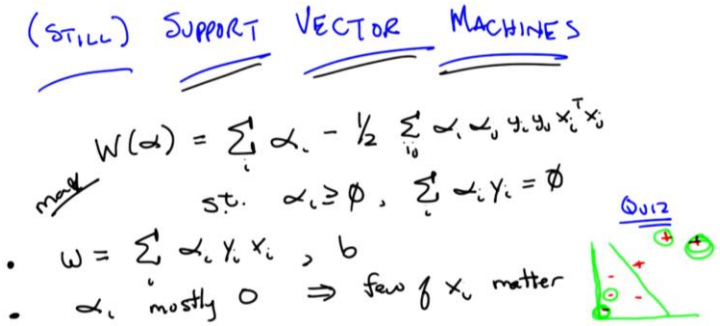
- Here's what this equation tells you:
  - We have basically started out by trying to maximize the margin (that's the same thing as trying to maximize $\frac{2}{||w||}$) subject to a particular set of constraints, which are how we codify that we want to classify every data point correctly in the training set.
  - We've argued that that's equivalent to minimizing $\frac{1}{2}||w||^2$, subject to the same constraints.
  - And then notice that you can convert that into a quadratic programming problem, which we know how to solve.
  - And it turns out that quadratic programming problem has a very particular form.
  - Rather than try to minimize $\frac{1}{2}||w||^2$, we can try to maximize another function that has a different set of parameters, which I'll call alpha - α.
  - The equation has the following form: It's the sum over all of the data points i, indexed by i, of this new set of parameters alpha, minus ½ times, for every pair of examples, the product of their alphas, their labels, and their values, subject to a different set of constraints, namely that all of the alphas are non-negative, and that the sum of the product of the alphas, and the labels that go along with them, are equal to zero.
- Instead of explaining how we got from $\frac{1}{2}||w||^2$to this quadratic equation, go read a quadratic programming book.
- What I really need you to believe, though, mainly because I'm asserting it, is that these are equivalent. So if you buy up to the point that we are trying to maximize the margin, then you just have to take a leap of faith here that, if we instead maximize this other equation, it turns out that we are solving the same problem. And that we know how to do it using quadratic programming. Or other people know how to do it and they've written code for us.
- All right, so trust me on this. **This is what it is that we want to solve**. What's really interesting is what this equation actually tells us about what we're trying to do.

6.5. Still More Support Vector Machines



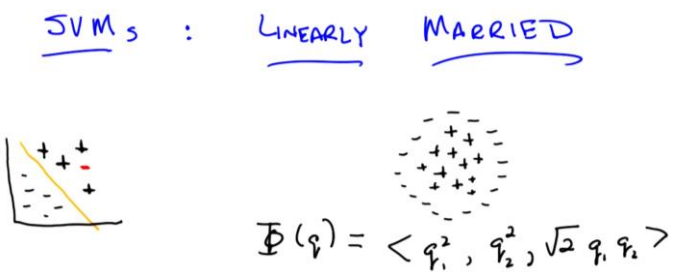
- We've done a little bit of moving stuff around, and kept the same equation of before. Remember, our goal is to use quadratic programming to maximize this equation.
- It turns out that once you find the alphas that maximize this equation, you can actually recover the w, which was the whole point of this exercise in the first place.
- Once you know w it's easy to recover b. You just find the value of x, you stick it into W, you know it's equal to +1, and then poof, you can find out b.
- So you can recover w directly from this and you can recover b from it in sort of an obvious way. But there are some other properties that are a little bit more interesting for you.
- I want you to pay attention to two things. One I am just going to have to tell you, and the other I want you to think about.
- Here's the one that I'm going to tell you. It turns out that each of those alphas are mostly zero, usually. So if w is the sum of the data points times their labels times alpha, and if the alpha is zero, then the corresponding data point isn't really going to come into play in the definition of W at all. So a bunch of the data just don't really factor into W.
- So basically, some of the vectors matter for finding the solution to this, and some do not. It turns out, each of those points are vectors. But you can find all of the support that you need for finding the optimal w in just using a few of those vectors. The non-zero alphas. So you basically built a machine that only needs a few support vectors.
- So the data points for which the corresponding alpha is non-zero are the support vectors, those are the ones that provide all the support for W. So knowing that W is the sum over a lot of these different data points, and their labels, and the corresponding alphas, and that most of those are zeroes, that implies, that only a few of the X's matter.

6.6. Quiz: Optimal Separator



- Thinking about everything that we've said, and thinking about what it means to build an optimal decision boundary maximizing a margin I want you to point to one of the positive examples that almost certainly is going to have a zero alpha, and one of the minus examples that almost certainly is going to have a alpha that are not a part of the support vectors. That is, in some sense, doesn't matter. Okay, go. Answer
- So I guess it really does make some intuitive sense that the line is really, really nailed down by the points close to it. And the points that are far away from it, really don't have any influence. So I would put zero alphas on the lower left hand minus and then one of the upper pluses.
- The points that are far away from the decision boundary, and can't be used to define the contours of that decision boundary, it doesn't matter whether they're plus or minus.
- It's like KNN except that you already done the work of figuring out which points actually matter. So you don't have to keep all of them. You can throw away some of them. So it doesn't just take the nearest ones, it actually does this complicated quadratic program to figure out which ones are actually going to contribute.
- It's just another way of thinking about instance-based learning, except that rather than being completely lazy, you put some energy into figuring out which points you could actually stand to throw away.
- Basically the alphas say pay attention to this data point or not. But if you look carefully at this equation, the only place where the x's come into play with one another is here. Generally speaking, given a couple of vectors, xi transpose xj is the dot product, which is the projection of one of those onto the other.
- This dot product represents the length of the projection. If the x's are, well if they are orthogonal to each other than it's going to be zero. But if they kind of point in the same direction, it's going to be a large value, and if they point in opposite directions it's going to be a negative value. So it indicates how much they're pointing in the same direction. So it could be a measure of their similarity.
- This is the kind of a notion of similarity. So if you look at this equation, what it basically says. Find all pairs of points. Figure out which ones matter for, for defining your decision boundary. And then think about how they relate to one another in terms of their output labels. With respect to how similar they are to one another.

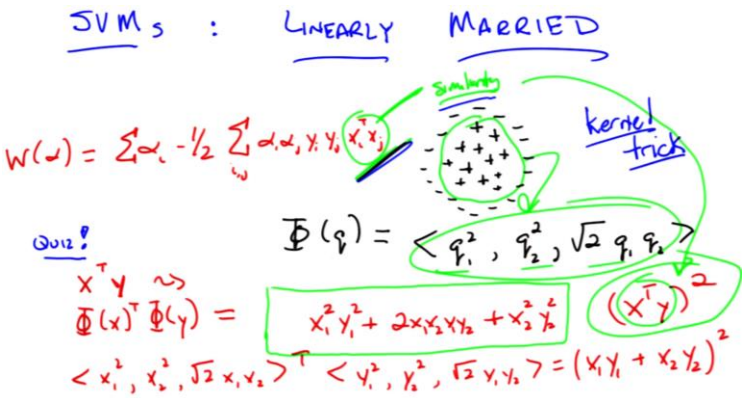
6.7. Linearly Married



- Given a linearly separable graph as seen above, what happens if I add one more point, the minus point in red?
- In some sense the margin is now negative, because there's going to be no way of slicing this up so that all the negatives are on one side and all the positives on the other. It's not linearly separable. This can be solved using SVM (was used as HW assignment)
- The other graph in the slide looks like there's a ring around the whole thing. You can draw lines all day long, and it's just not going to slice things up. So now we have to come up with some clever way of managing to make this work so here's the little trick we're going to do: I am going to change the data points without changing the data points.
- So that means it has two different components, Q1 and Q2. And I am going to produce from those two components, Q1 and Q2, a triple. So I am going to put that point into three dimensions now. And the dimensions are going to look like this.

  $\Phi(q) =< q_1^2, q_2^2, \sqrt{2}\, q_1 q_2 >$
- So Q, is a two dimensional point. So it's got, Q1 and Q2 are its two components. Take the first component, make a new vector where the first component of that is squared, take the second component, make a new vector where that value is the second component squared. Then square root times the product of those two as the third dimension.
- Let me point out something for you. One is I haven't actually added any new information, in the sense that I'm still only using Q1 and Q2. Yeah, I threw a constant in there, and I'm multiplying by one another, but at the end of the day, I haven't really done much. It's not like I've thrown in some boolean variable that gives you some extra information. All I've done is taken Q1 and Q2 and multiplied them together, or against one another.
- I did this because it's going to turn out to provide a cute little trick. And in order to see that cute little trick we need to return to our quadratic programming problem. You'll recall that I asked you to talk about Xi and Xj , and what it looks like in this equation. And what I think we agreed to is that we can think about Xi transpose Xj as capturing some notion of similarity. So, it turns out then, if we sort of buy that idea of similarity, that really what matters most in solving this quadratic problem, and ultimately solving our optimization problem, is being able to constantly do these transpose operations.
- So, let's ask the question that, if I have this new function, phi (Φ), what would happen if I took two data points, and I did the transpose or the dot product between them.

6.8. Quiz: What is the Output

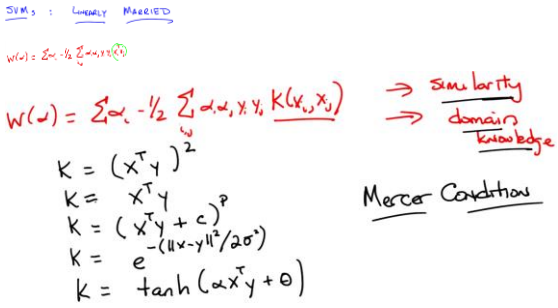


- Let's imagine we have two points. I'm going to call them X and Y, just so I can confuse you with notation. And they are both two dimensional points. So they're in a plane. And they have components X1 and X2 and components Y1 and Y2.
- Rather than computing the X transpose Y, their dot product, I want to compute the dot product of those two points, but passed through this function phi.
- So, x is really x1 x2 and y is really y1 y2 and phi x is now this crazy triple x so I, so I wrote $x1^2$ $x2^2$, root 2 x1 x2. That's the vector that we get for phi x. $x_1^2, x_2^2, \sqrt{2}\, x_1 x_2$
- And then the y vector gets transformed to the same thing, except for with y's, y1 squared, y2 squared, root 2 y1 y2.
  $y_1^2, y_2^2, \sqrt{2}\, y_1 y_2$
- So, then, the, the dot product is just the products of the corresponding components summed up. So x1 squared y1 squared plus X2 squared y2 squared plus 2x1x2y1y2 $x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2$
- We can factor this. X1 Y1 Plus X2 Y2. Whole thing's squared. $(x_1 y_1 + x_2 y_2)^2$
- And what's an even simpler way of writing that? It's x transpose y on the inside, and then we square it on the outside. $(x^T y)^2$
- There was method to the madness when I created the phi function?
- This is basically a particular form of the equation for a circle Which means that we've gone from thinking about the linear relationship between Xi and Xj or your data points and we've now turned it into something that looks a lot more like a circle.
- If you believe me in the beginning where we notice that Xi transpose Xj is really about similarity. It's really about why it is you would say two points are close to one another or far apart from one another. By coming into this transformation over here, where we basically represented the equation for a circle, we have now replaced our notion of similarity from being this very simple projection to being the notion of similarity is whether you fall in or out of a circle. So more sort of about the distance as opposed to what direction you're pointing.
- Both of them are fine because both of them represent some notion of similarity, some notion of distance in some space. In the original case, we're talking about two points that are lined up together. And over here together with this particular equation we represented whether they're inside the radius of a circle or outside the radius of a circle. Now this particular example assumes that the circle is centered at the origin and so on and so forth, but the idea I want you to see is that we could transform all of our data so that we separated points from within one circle to points that are outside of the circle. And then, if we do that, projecting from two dimensions

here into three dimensions, we've basically taken all of the pluses and moved them up and all of the minuses and moved them back, and now we can separate them with the hyperplane.

- Without knowing which ones are the pluses and which ones are the minuses because they are the ones that were closer to the origin, so they get raised up less. I can basically take my data, and I transform it into a higher dimensional space, where suddenly I'm now able to separate it linearly.
- That's very cute, but I chose this particular form for a reason. Can you guess why? There are lots of different ways we could have fit the circle pattern. I chose this particular form because not only does it fit the circle pattern, but it doesn't require that I do this particular transformation. Rather than taking all of my data points and projecting them up into three dimensions directly, I can instead still simply compute the dot product and now I take that answer and I square it.
- In this formulation of the quadratic program that you have there in terms of capital W if you write code to do that, each time in the code you want to compute Xi transpose times Xj, if you just squared it right before you actually used it, it would be as if you projected it into this third dimension and found a plane.
- This is known as the **kernel trick**.
- So, again if we really push on this notion of similarity. What we're really saying is we care about maximizing some function that depends highly upon how different data points are alike, or how they are different. And simply by writing it this particular way, all we're saying is, you know what, we think the inner product is how we should define similarity. But instead, we could use a different function altogether, phi or more nicely represented as x transpose y squared, and say, that's our notion of similarity. And we can substitute it accordingly.
- We never used phi. We're able to avoid all of that by coming up with a clever representation of similarity. That just so happened to represent something, or could represent something in a higher dimensional space.
- You can't just use anything, but in practice it turns out you can use almost anything. Also it turns out for any function that you use, there is some transformation into some dimensional space, higher dimensional space, that is equivalent. Now, it may turn out that you need an infinite number of dimensions to represent it. But there is some way of transform, transforming your points into higher dimensional space that happens to represent this kernel, or whatever kernel you choose to use.
- The kernel is the function itself. In fact, let me clean up this screen a little bit. And, and see if we can make this a little bit more precise and easier to understand. (next slide)

6.9. Kernel



- Let's look at this xi transpose xj (circled in green). I've just replaced it with a function, which I'm going to call a kernel. Which takes xi and xj as parameters, and will return some number.
  $K(x_i, x_j)$
- As we talked about before, we think of the xi transpose xj, as some notion of similarity, and so this kernel function is our representation, still, of similarity. Another way of thinking about that, by the way, is that this is the mechanism by which we inject domain knowledge into the support vector machine learning algorithm.
- You can create these kernels and these kernels have arbitrary relationships to one another. So, what you're really doing is, projecting into some higher dimensional space, where, in that higher dimensional space, your points are in fact, linearly separable.
- Also, because you're using this kernel function to represent your domain knowledge, you don't actually have to do the computation of transforming the points into this higher dimensional space. I mean, in fact if you think about it, with the last kernel that we used, computationally, there was actually no more work to be done. Before we were doing x transpose y, and now we're still doing x transpose y, except we're then squaring it. So that's just a constant bit more work. And the other kernel we talked about was just X transpose Y by itself. That's a kernel too.
- We can actually write a general form of both of these. And as a very typical kernel, it's the polynomial kernel where you have x transpose y plus some constant, let's call it c, raised to some power p. And as you can see, both of those earlier kernels are, in fact, just a special case of this. That should look familiar, where we were doing polynomial regression. Now, rather than doing polynomial regression the way we were thinking about it before, we use a polynomial kernel and that will allow us to represent polynomial functions.
- There're lots of other kernels you can come up with, here's just a couple. I will just sort of leave em up to you, to think about. And there's tons of them.Here's one that I happen to like, a sort of radial basis kernel. So if x and y are really close to each other, then it's e to the minus zero over something which is like e to the zero, which is like one. So there's similarities like one if they're on top of each other. If they're very far apart, then it's like their distance is something very big divided by something e to the minus something very big is very close to zero. So it does have that kind of property like the sigmoid where it transitions between zero and one but it's not exactly the same shape as that.
- Actually if you wanted to get something that looked like a sigmoid, here's one. Where alpha's different from the other alphas, but I couldn't think of a different Greek letter. And this function gives you something that looks a lot more like a sigmoid.
- There's been a lot of research over the years on what makes a good kernel function. The most important thing here is that it really captures your domain knowledge. It really captures your

notion of similarity. You might notice that since it's just an arbitrary function that returns a number, it means that X and Y or the different data points you have, don't have to actually be points in a numerical space. They could be discrete variables. They could describe whether you're male or female. As long as you have some notion of similarity to play around with, that you can define, that returns a number, then it doesn't matter. It will always work.

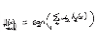
- So can you do things like strings or graphs or images. How are two strings similar? Maybe they're, they're similar if their edit distance is small. The number of transformations that you have to give in order to transform one string to another. If there are few of those, then they're very similar. If there are a lot of those then they're very dissimilar.
- While it's not clear whether there are any bad kernel functions, it is the case that in order for all the math to go through, there is a specific technical requirement of a kernel function. It has a name. And it's the **Mercer Condition**. The Mercer condition is a very technical thing we'll talk about this again, a little bit in the homework assignment. But for your intuition in the meantime, it basically means it acts like a distance, or it acts like a similarity. It's not an arbitrary thing that doesn't relate the various points together. Being positive is something definite in in this context means it's a well behaved distance function.

6.10. Back to Boosting

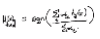
- It appears that boosting does not always over-fit, it doesn't seem to over-fit in the ways that we would normally expect it to over-fit. And in particular we'd see a, you know, an error line on training And what we expect to see is a testing line that would, you know, hue pretty closely and then start to get bad.

- But what actually happens is that instead, this little bit at the end where you get over fitting seems to instead. Just keep doing well. In fact, getting better and better and better.

- And I promised you an explanation for why that was. You don't have this problem with overfitting at least not in the typical way as you keep applying it over and over again like you do with something like neural networks. It really boils down to noticing that we've been ignoring some information.
- What we normally keep track of is error. So error on say a training set is just the probability that you're going to come up with an incorrect answer or come up with an answer that disagrees with your training set, and that's a very natural thing to think about and it makes a lot of sense.
- But there's also something else that is actually captured inside of boosting and captured by a lot of learning algorithms we haven't been taking advantage of, and that's the notion of confidence. So confidence is not just whether you got it right or wrong. It's how strongly you believe in a particular answer that you've given.
- A lot of the algorithms we talked indirectly have something like that. In a nearest neighbor method, if you are doing five nearest neighbor and all five of the neighbors agree, that seems different than the case with 3 vote one way and 2 vote the other.
- If you think of that in terms of regression then you could say something like the variance between them is sort of a stand in for confidence. Low variance means everyone agrees, high variance means there's some major disagreement.
- So what does that mean in the boosting case? As you recall, the final output of the boosted classifier is given by a very simple formula.



  - h of x is equal to the sign of the sum over all of the weak hypotheses that you've gotten of alpha times h.
  - So the weighted average of all of the hypotheses. If it's positive you produce a plus one. And if it's it negative you produce a minus and if it's exactly zero you don't know what to do so you just produce zero. Just throw up your hands.
- So I'm going to make a tiny change to this formula, just for the purpose of explanation, that doesn't change the fundamental answer. And I'm just going to take exactly this equation as it is. And I'm going to divide it, by the weights that we use.



- The alpha is always set to be the natural log of something. These alphas are applied to hypotheses whereas the alphas in the SVM settings were being applied to data points. Unfortunately in machine learning people invent things separately and reuse notation. Alpha's an easy Greek character to draw, so people use it all the time.
- But here, remember, alpha's the measure of how good a particular weak hypothesis was, and since it has to do better than chance, it works out that it will always be greater than zero.

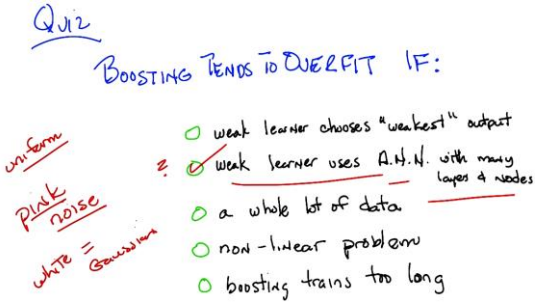
- This normalization factor, this denominator doesn't, it's just a constant with respect to x, the input. So it won't actually change the answer. So it really is the same answer as we had before, just a different way of writing it. What it ends up doing is it normalizes the output. So it turns out that this value is always going to be between minus one and plus one. Otherwise it doesn't change anything about what we've been doing for boosting.
- This makes it easier for me to draw what I want to draw next. So, we know that the output of this little bit inside the sign function is always going to be between minus one and plus one. Let's imagine that I take some particular data point x and I pass it through this function, I'm going to get some value between minus one and plus one. And let's just say for the sake of the argument, it ends up here. 

- It's a positive example and it's near plus one. This would be something that the algorithm is getting correct. It's not just getting it correct, but it is very confident in its correctness because it gave it a very high value. By contrast there could have been another positive that ends up around here. 
- So it gets it correct but it doesn't have a lot of confidence so to speak in its correct answer because it's very near to zero. So that's the difference between error and confidence.
- So now imagine there's lots of little points like this. 
- And if you're doing well, you would expect that very often you're going to be correct. And so you end up shoving all the positives over here to the right, and all the negatives over here to the left. And it would be really nice if you were sort of confident in all of them.
- So now I want you to imagine that we've been going through these training examples, and we've gotten very, very good training error. In fact, let's imagine that we have negative training error. Let's imagine that we have no training error at all. So we label everything correctly. So then the picture would look just a little bit different We're going to have all the pluses on one side, and all the minuses on the other. 
- But we keep on training, we keep adding more and more weak learners into the mix.
- What ends up happening in practice is, you have to do some kind of distribution on the hard examples. And the hard examples are going to be the one that are very near the boundary. So as you add more and more of these weak learners what seems to happen in practice is that these pluses that are near the boundary and these minuses that are near the boundary just start moving farther and farther away from the boundary. So, this minus starts drifting until it's all the way over here, this minus starts drifting until it's all the way over here. And the same happens for the pluses. 
- As you keep going, what ends up happening is that your error stays the same. It doesn't change at all, however your confidence keeps going up and up.

- Which has the effect, if you'll look at this little drawing over here of moving the pluses all around over here, so they're all in a bunch, and the minuses are on the other side.
- There's a big gap between the leftmost plus and the rightmost minus, in the context of this lecture reminds us of a margin. Basically what ends up happening is that as you add more and more weak learners here the boosting algorithm ends up becoming more and more confident in its answers which it's getting correct. And therefore effectively ends up creating a bigger and bigger margin. 

- Large margins tend to minimize overfitting. So, counter intuitively, as we create more and more of these hypotheses, which you would think would make something more and more complicated, it turns out that you end up with something smoother, less likely to overfit and ultimately, less complicated. So the reason boosting tends to do well and tends to avoid overfitting even as you add more and more learners is that you're increasing the margin.
- So, there you go, do you think, then, that boosting never overfits?

6.11. Quiz: Boosting Tends to Overfit

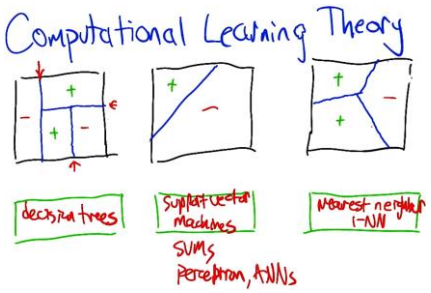


- So we just tried to argue that boosting has this annoying habit of not always overfitting, but of course something can always over fit. Because otherwise we just do boosting and we're done, then neither of us would have jobs.
- So here are five possibilities.
- All right, Michael. What's the answer?
- The last one, boosting tends to overfit if boosting trains too long. You just told me a story about that not being true. So I'm going to eliminate that one from consideration.
- Boosting tends to overfit if it's a nonlinear problem. I don't see why the problem being linear or nonlinear has anything to do with overfitting.
- A whole lot of data is the opposite of what tends to cause overfitting. If there's lots of data then you'd think that it would actually do a pretty reasonable job if there's a lot to fit. There's a lot going on there. It's unlikely to overfit. In fact if a whole lot of data included all of the data, and you actually could get zero training error over it, then you know you have zero generalization error because it'll work on the testing data as well, because it's in there.
- Weak learner uses artificial neural network with many layers and nodes. So I'm guessing that you wanted me to think about that being something that, on its own, is prone to overfitting, because it's got a lot of parameters. If we fit a neural net, and then we fit another neural net, and we fit another neural net. And we're combining all the outputs together in the correct, weighted way. It's not obvious to me that that should be a good thing to do. I'm not sure it would overfit, but it seem like it sure could.
- Let me look at the first one. Weak learner chooses the weakest output. Well, I mean boosting is supposed to work as long as we have a weak learner. And it doesn't matter if it chooses the weakest or the strongest. All that matters is it does significantly better than a half.
- So the only one of these choices that is likely to be true is the second one.
- So let me give you an example of when that would be correct. So let's imagine I have a big powerful new network that could represent any arbitrary function. Okay, it’s got lots of layers and lots of nodes. So, boosting calls it, and it perfectly fits the training data, but of course overfits. So then it returns, and it's got no error, which means all of the examples will have equal weight. And when you go through the loop again, you will just call the same learner, which will use the same neural network, and will return the same neural network. So every time you call the learner, you'll get zero training error, but you will just get the same neural network over and over and over again. And a weighted sum of the same function is just that function.
- So if it overfit, boosting will overfit. And not only will it overfit, but it'll just, it'll be stuck in a horrible loop of error.

- So that's why this is the sort of situation where you can imagine boosting providing a lower fit. If the underlying learners all overfit and you can never get them to stop overfitting, then there's really not much you can do.

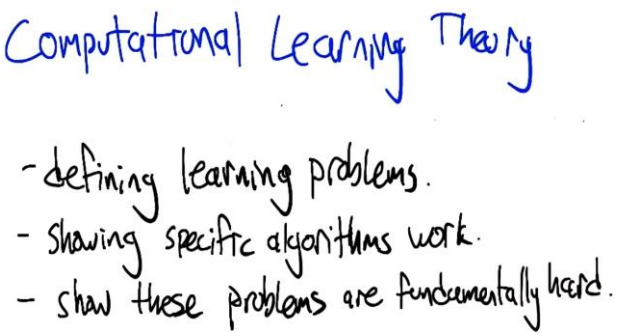
7. Computational Learning Theory

7.1. Quiz: Computational Learning Theory



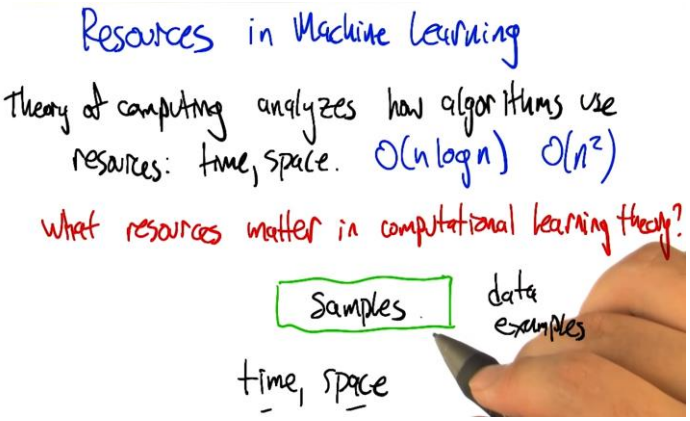
- I have this feeling that it's necessary to always make sure you know what problem you're solving, before you start proposing algorithms for solving it. And we haven't really nailed down what exactly that problem is. And that makes things hard. It makes it hard to know, for example, whether or not one algorithm's better than another.
- So one of the things we talked about so far is algorithms for doing for learning classifiers. So, if you can imagine that each of these three boxes is the output of a classifier in a two dimensional space. So it's a two dimensional input space. We've run a leaner, and when it spits out a classifier, this is how it classifies the the regions of the space. So I use blue lines to separate the square, the two dimensional space, into regions, and then I labeled each region with a minus or a plus, so you can tell what the classifier was actually doing.
- So I would choose for the second one support vector machines, and the reason why is because there seems to be a single line that seems to be in the middle between where the plus and minus are, so it looks like it's trying to maximize the margin. Though, those aren't training points, these are labeling the regions. The support vector machine is going to find a linear separator. And so the output, the classifier that comes out of it is going to separate any kind of region into pluses and minuses, with some lines separating them out. That's also true of perceptrons, it's also true of certain kinds of you know, simple neural nets. So all of those seem like they would be reasonable answers for the middle one.
- I'm going to say the third one is a nearest neighbors method. One nearest neighbor. So we, I don't think we got a chance to draw one of these diagrams when we were talking about nearest neighbors. But this is called a Voronoi diagram. Because what's happening is it's breaking up the plane into the regions that are closest to some particular points. So you give it a set of points, it breaks up the plane into the things closest to that. And in this particular case our we probably had three points. And one was labeled plus, and another was labeled plus. One was labeled minus, and so it break the space up into these sections that way.
- I'm going to say the first one is decision trees. Because decision trees split things you know, if you're talking about in a two dimensional space it's going to split at the top node in one dimension. And here it looks like that's this split. Then maybe, looks like on the left side that's a minus. On the right side there's another split. On the left side of that it's a plus. On the right side there's another split into pluses and minuses. You get these sort of interesting nested rectangles. They also can be very pretty.

7.2. Learning Theory



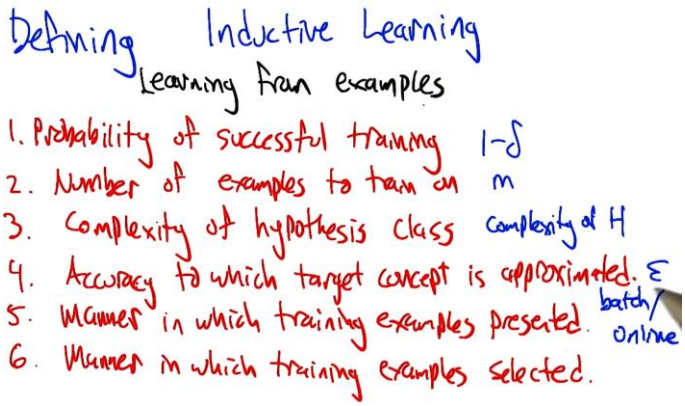
- Computational learning theory really gets us a formal way of addressing three really important questions.
    - One is, what's a learning problem? Let's, let's define very carefully what it is that we want a learning algorithm to do.
    - If we can do that, we can actually show that specific algorithms either work or don't work, with regard to the definition of the problem. And maybe we can even come up with algorithms that solve those problems better. So that's kind of on the upper-bound side.
    - And then on the lower-bound side we can also show, for example, in some cases that some problems are just fundamentally hard. So you, you define a particular learning problem and you discover. Wait, the algorithms that I'm thinking of don't seem to work. You might actually be able to show that no algorithms, say, no algorithms in some particular class are ever going to be able to solve them because, that problem is not solvable by problems in that class. So those problems are fundamentally hard.
- So, answering these kinds of questions require that you be fairly careful about defining things and using mathematical reasoning to determine what's going on. So we're going to focus mostly on that, talk about some algorithms that are not necessarily practical. You wouldn't necessarily want to use them, but they do help illuminate what the fundamental learning questions are and why certain algorithms are effective and ineffective.
- We just justified this in the same way that a computing theoretician might try to justify theory. In fact, the, the kinds of analyses and tools that are used for analyzing learning questions are also the same kinds of tools and mechanisms that are used in the context of analyzing algorithms in computing.

7.3. Quiz: Resources in Machine Learning



- Often in theory of computation, we analyze algorithms in terms of their use of resources, usually time and space.
- When we talk about an algorithm running in say n log n time, we're saying something about the time that it takes.
- If you say that it uses n^2 space, we're talking about the amount of memory that it takes up as the function of the growth of the inputs.
- We're trying to select among algorithms. We want algorithms that use their resources well. What do you think would some reasonable resources to try to manage in a learning algorithm?
- Two of them are time and space. After all, at the end of the day, it's still an algorithm. We need to be able to analyze algorithms in terms of time and space.
- The only thing that matters in machine learning or the most important thing in machine learning is data. So I would think that another resource that we care about is the data and in particular the set of training samples that we have. We want to know, can we learn well with a small amount of samples. In particular if, our learning algorithm works great in terms of time and space, but in order to run it you actually have to give Examples of every possible input, then that's not going to be a very useful algorithm. So, the fewer samples that it can use, the more that it's generalizing effectively, and the better it is at learning.

7.4. Defining Inductive Learning

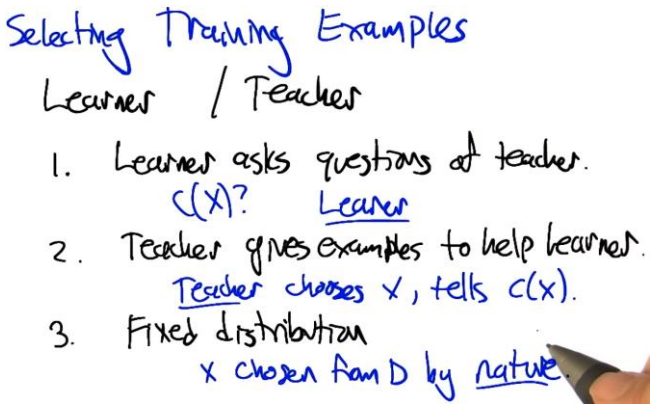


- Inductive learning is learning from examples. It's the kind of learning problem that we've been looking at the whole time, but we haven't been very precise about all the various quantities that we want to be able to talk about. The number of properties that we actually need to be thinking

about when we go and define what an inductive learning problem is and measure what an inductive learning algorithm does.

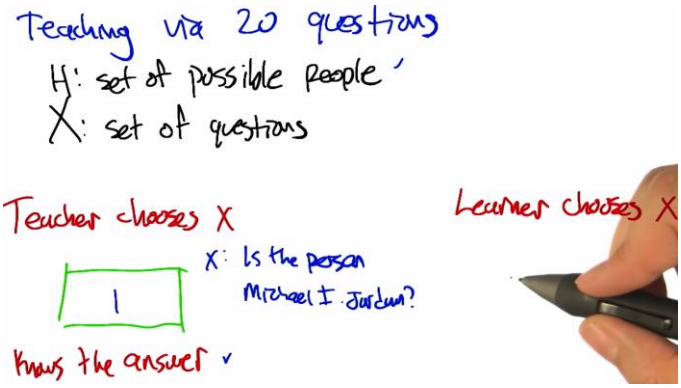
- What's the probability that the training is actually is going work. You know, whatever it is that it's trying to produce, it may be that in some cases, because the data is really noisy or just got a bad set of data, it might not actually work. So, we generally talk about a quantity like 1 minus delta as the probability of success. Delta here obviously is a probability of failure and this is just 1 minus that.
- There's also issues like the number of examples to train on.
- There's also something that we really haven't talked about yet, but you could imagine that the complexity of the hypothesis class might matter. The complexity of the class could be like the sum of the complexities of all the hypotheses in the class. A hypothesis class is complex if it has very complex hypotheses, then you can say, well, if you have a hypothesis class that can't represent much, then it will be hard for you to learn anything complicated. The downside to having a hypothesis class that is very, very complex is it would be much easier to overfit. So getting something that actually works well might be challenging. You might need a lot more data to nail down what you're really talking about. So it's a bit of a double edged sword.
- It may be easy to learn if you don't have to learn very well. So the accuracy to which the target concept is approximated, often written as epsilon, is another thing that's going to be important in understanding the complexity of a learning algorithm.
- So those, those are kind of the main complexity related quantities that I wanted to talk about. There's also some choices as to how the learning problem is actually framed. There's the manner in which training examples are presented. And there's the manner in which training examples are selected for presentation. And we're going to talk about both of these. When I talk about the manner in which training examples are presented, there's two that I think are really important to look at
  - Batch- that's mostly what we've looked at so far, that there's a training set that's fixed and handed over to the algorithm in a big bolus, right. A big group. A big batch.
  - Or it could also be presented online, so one at a time. So we say to the training algorithm, or the learning algorithm, here's an example. And then it has to predict what the label is. And then the algorithm can say, oh here's what the right label is. Let's try again. Here's another example. And it can go back and forth like that.
- Let's talk about the manner in which training examples are selected in the next slide.

7.5. Selecting Training Examples



- It matters how we select training examples when when a learner is needing to learn. Let's at least articulate some various ways that training examples could be selected. And then for each one we might end up with a different answer as to how much training data is going to be needed, if the training examples are selected in that particular way.
- So, you keep using the word learner, and when there's a learner there's often a teacher. So, I'm trying to think about this in the different ways that learners and teachers can interact. So, I'm going to try to think about my, my experience as a professor. So, there are a couple. Here's one, one is: sometimes the learner asks questions of the teacher. In this case, the learner would be selecting the training examples. The learner would say, here's an input x, would you please tell me c(x)?
- Then there's another case. So the learner asks questions of the teacher. But sometimes the teacher just goes ahead and gives x, c(x) pairs. And let's say it actually is trying to help the learner.
- It could be the learner asking to try to get data. It could be the teacher asking to try to lead the learner to something good. Neither sound like things that we've been looking at so far, that somehow the x's and the c(x)'s, the training examples, just come out of nowhere. They come from some underlying distribution. It's just nature. It's just coming at us from some process that we don't know what that process is.
- So in some sense there's three individuals that could be asking for examples. There's the teacher, there's the learner, and there's the world around them.

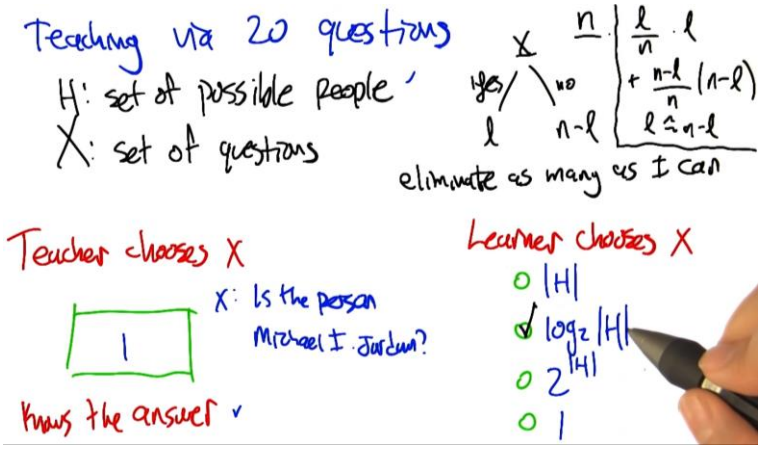
7.6. Quiz: Teaching Via 20 Questions



- Think about two different cases. One where the teacher is going to try to select which questions to ask. So the teacher's going to say to the student, here's the question you should ask me next. To try to figure out which person it is, as quickly as possible. And then we'll, next we'll look at what happens if the learner's the one asking those questions. So in some sense, it doesn't seem like it should make a difference, because, in either case, the learner is going to ask the question and the teacher is going to answer truthfully. Just in one case, the learner has to come up with the questions, and in the other case the teacher has to come up with the questions.
- The teacher actually knows the answer. How many questions are necessary for a smart learner to figure out the right person, assuming that the teacher is the one who gets to feed the learner good questions?
- You could've done this in one question, a sufficiently helpful teacher. You can, you can do this in one.

- That seems like cheating but it’s a very different problem, when the, the question has to come from the learner's side.

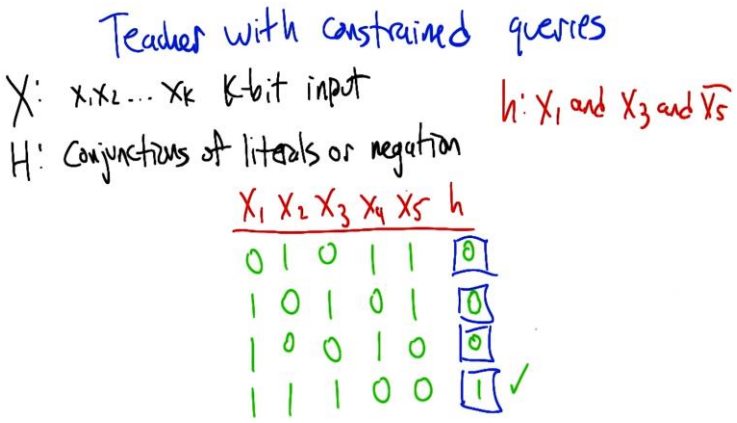
7.7. Quiz: The Learner



- If the learner is choosing which questions to ask the learner its in a distinctly more difficult situation because it doesn't actually know what the right answer is.
- There's a couple a ways to think about it, but let me just kind of go top down. So, here's what I'm thinking. I think the same principle applies, where you sort of want to ask the question that gives you the maximum amount of information.
- So, when we ask a question, x, it's going to either have a yes answer. Or a no answer. And let's say that at any given point in time, we have n possible people that we're trying to reason about. And then after we choose x, after we choose the question, if the answer is yes then we're going to have, I don't know, some other number of people.
- So I always want to ask the question that's going to give me maximum information. I'm the learner. I don't know, everything like the teacher does, but what I do know are all of my hypotheses. Alright. So I know how each hypothesis responds to each of the questions. So, in general, it seems to me that I should try to eliminate as many hypotheses as I can, okay? That makes sense?
- As a learner, I don't know the answer, so finding questions that whittle it down to one if the answer is yes aren't very helpful because the answer might be no. Let's see so under the assumption the target hypothesis was chosen from H say, uniformly at random. What's the expected number of hypotheses we can eliminate with a question that has this form? Well, it's binary. So if you assume that the hypothesis space covers everything. Then the best you can do is eliminate about half of them, on average. Since all possibilities are there, the smallest the, the best you can hope for is about a half.
- Okay, so you want to pick questions that roughly split things in half. And if you can do that every single time, then every single time, you will split the set in half. So that'll take you a logarithm amount of time.
- So the number of times you can divide a number in half before you get down to one is exactly the log base two. So if we start with size of h hypotheses, it will take us like log base two to whittle it down to a single question. This is a lot larger. I mean this is a nice small number, but it's a lot larger than one, right? You know, one, one is really great. This is you know, exponentially, no, this is much, much worse than one. But but it's still really good. You can, you

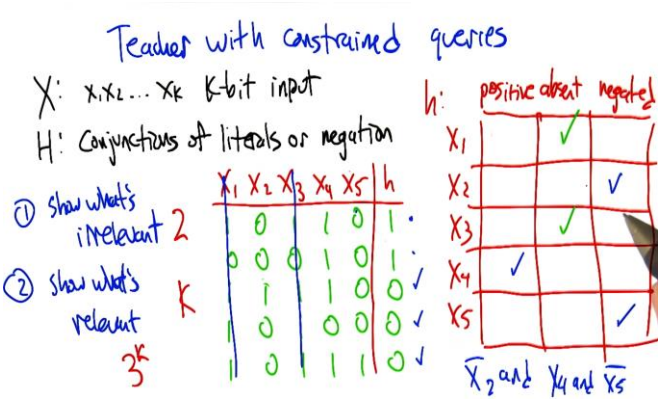
know, it's not considering all the hypothesis. It's very cleverly whittling it down so that it only has to look at the log of that.

7.8. Teacher With Constrained Queries



- When I gave the example of a teacher asking questions to to lead the learner to a particular hypothesis, I allowed the teacher to ask any possible question in the universe of possible questions. Right, so the teacher could construct a question, and in this particular case, that, that specifically had a yes answer for the target hypothesis. And a no answer everywhere else. And that just simply doesn't happen in any realistic learning questions. So we need to think a little bit more about the question of what happens when the teacher is constrained with respect to the kinds of queries that it can suggest. So I'm going to, I'm going to set up classic computational learning theory hypothesis class, and we'll go through some examples about what happens when the teacher's constrained in this way. So let's imagine that our input space is basically k-bit inputs, so x1 through xk, and the hypothesis class and literals or their negation.
- Here's a concrete example. Here's a hypothesis that is in this set. X1 and x3 and not x5. We're going to connect together some of these variables, not necessarily all of them. And some of them can be negated. And, it's all the connectors have to be ands.
- So let's say our input is this. X1 is 0. x2 is 1. x3 is 0. x4 is 1. x5 is 1. X2 and x4 don't matter. I just have to look at x1, x3, and x5. The bottom output is the only 1.

7.9. Quiz: Reconstructing Hypothesis

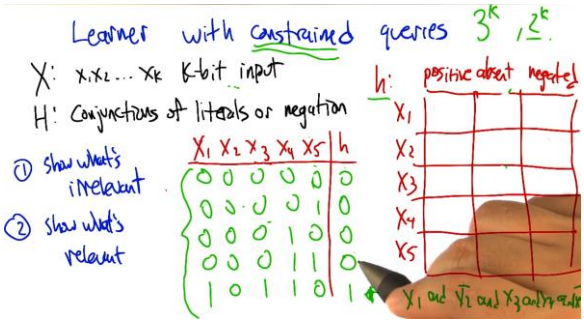


- All right, so here, here is, here's a table of examples with the input pattern and the actual output pattern of the hypothesis we're looking for. But I'm not going to tell you the hypothesis, you have to figure it out. So the way you're going to do that is, by figuring out for each variable, does it appear in the conjunction in its positive form, in its negative form or it's not there at all? For example if you want to say the hypothesis is X1 and not X5, you would write something like X1 and not X5, and the other ones are all absent. And just to be clear Charles, I, I've picked this particular set of examples so that, you particularly, Charles Isbell PhD, would have the best chance of figuring out the hypothesis with the smallest number of examples.
- I appreciate that.
- Go. Answers
- So, what, how do you start with this now, Charles?
- Okay, so the first thing I'm going to do is, I'm going to look at the first two examples. And the reason I'm doing that is because I know they both generate true. And so I'm going to look for variables that are inconsistent. So if I look at X1, for example, it's a one in the first case, and a zero in the second case. So it can't be the case that it's required in order for this to be true. And the same would be true for x3. So I'm going to say that neither x1 nor x3 matter. By contrast, x2, x4, and x5 all have the same values in both of those cases.
- So we don't know much about them quite yet.
- No
- But let's so let's, that seems very well reasoned. So we know that x1 can't be part of this formula and x3 can't be a part of this formula. So, let's just sort of imagine that they're not there cause they don't really give us any information any more.
- Beautiful.
- Alright, So what's left?
- So what's left is now to make certain that x2, x4 and x5 are necessary, and particularly necessary with the, the values that they have. So I guess all I can really do is see if there's anything else to eliminate. If I were just looking at the first two, I would think that the answer was not X2, X4, not X5.
- Alright. so, hang on, not X2, X4, not X5.
- Right. So, that's what I currently think it is based upon what I just saw.
- And that would be, that's consistent with the first two examples.
- Right. And so, now I want to make certain is consistent with the next three examples. This is the easiest way for me to think about this, anyway. So, let's see. Not X2, X4, so that should be false,

which it is. They're all false, so let's see not X2. But x4, up, that should be false, which it is. And then I do that same thing. But wait, why isn't that the answer? That can't be the answer.

- It is the answer, you got it.
- Huh I got it right.
- So the thing to notice is that in, in these first two examples we have X2 is false X4 is true and X5 is false and that's enough to make the conjunction true but making- flipping any one of those bits is enough to make it false so what I showed in the in the remaining examples is that just by turning this X2 into an X1 leaving everything else the same we lose it. Similarly if we flip the X4 to zero and leave everything else the same, we lose it. Similarly if we flip x5 to one, and leave everything else the same, we lose it. So that means that each of these is necessary to make the conjunction. They're all actually in there.
- That's just what I was thinking. So, in other words, you gave me some positive examples to eliminate things that were necessary, and then you gave me negative examples to validate that each of the variables that I saw so far were necessary because getting rid of any one of them, gave me the wrong answer.
- Exactly. Let's, let's even write down those two steps. So the first thing was show what's irrelevant. And how many questions How many queries might we have needed to show that?
- Well, one per variable.
- Well actually we only need two because what I did is I, I used, all the relevant ones I kept the same and all the irrelevant ones I flipped from one to the other. I just have to show you that it's still, the output is still one even though they have two different values.
- Oh, no, no. When I said all of them, you know, k of them was because I didn't know that, what if all of them were irrelevant
- Then it would still be two. because then I could just show you the all zeroes, and the all ones.
- You're right. You're right. You just need, oh that's right. That's exactly right.
- Alright. And then I have to show you that the, that each, the remaining variables is relevant by flipping it and showing you that the answer is zero. And how many questions did I need to do for that?
- Three.
- Yeah, three in this case cause there were three variables that were used in the formula. What's the most it could be?
- Well k, cause all of them could be relevant.
- Yeah, so it's you know, it's kind of interesting that, that in fact the total number of hypothesis here is three to the K. Because you know, you can see it right off this table that for each of the variables, it's either positive, absent or negated. But the number of questions that a smart teacher had to ask was more like, K plus two.
- Huh.
- Which is pretty powerful.
- Right, so. The smart teacher can help me do this in linear term. So what if I were, I, I didn't have the teacher who could give me these examples and I always had to ask?
- That's a good question, let's let's do that.

7.10. Learner With Constrained Queries

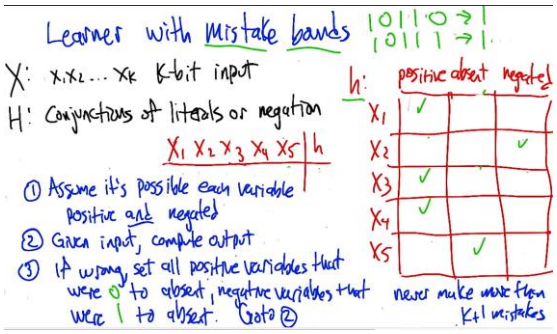


- Alright, so, you asked unfortunately what happens when the, the learner is now a part of this. Now the learner doesn't have that advantage that the teacher had of knowing what the actual answer was and therefore being able to show specifically what's irrelevant and show what's relevant. So, what could the learner do to try to learn about this? So again, remember that there are 3 to the k possible hypotheses, and if it could use the 20 questions trick, it could do this in log base 2 of 3 to the k, which is the same as K times log base 2 of 3. Which is you know, worse than what we had. It's this is, this is larger than in K. So, but can we actually do that?
- I'm going to say yes.
- I don't think we can, so can you help me figure out how that would go?
- Oh, I was just going to assert it, then hope you would tell me. so, how would we do that? Well, we, we, the trick we did before is, we, we tried to find a specific question we could ask, such that we would eliminate half the hypotheses.
- Indeed. But it's not clear how you could even ask such a question. Yeah, so, so just to do this as a thought exercise, I have a hypothesis in mind.
- Okay.
- And you can ask me anything you want, and I will tell you true or false. But you're going to have a very painful time finding it.
- Yeah, but that's just because I'm human. Okay, so I need to find a question where, of all the hypoth, so I have all the possible 3 to the K hypotheses. I want to try to come up with something that's going to eliminate a third of them which is just going to be hard for me to do because I could write the program to do this.
- I'm not sure you could. I think, at the moment, there's well, because I didn't choose my hypothesis at random. I chose a specific hypothesis. Though I guess I could have chosen at random from a subset, and you would have still had a hard time finding it. But let's, just as an exercise. Throw out, give me a, give me a x1 to x5, and I'll tell you what the output is.
- Okay, 00001. Or actually, you know what? All zeros.
- Okay, all zeroes, the output is zero.
- Oh, that's what I should, that's not what I should have done. I should have. No, no.
- That's okay, I won't count that one.
- [LAUGH] Can I just give you like, maybe 3 to the k of them and you'll not count any of them until I get it right?
- Well, that's the problem, right? Well, not 3 to the k, but if you, if you, you know, make 2 to the k guesses, do, you'll be okay. But you'll also have looked at all possible inputs. So that's not really that interesting. But in particular, the example that I'm thinking of, you're going to have to guess almost this many just to get a positive example. So almost everything that you throw in is giving almost no information. Because saying no doesn't really tell you very much.

- Yeah that's what I was thinking. Well, what I was thinking was I need to find one where the answer is yes.
- Exactly, and I made it so that it's going to take you exponential time just to find one. Once you've found that one, then you're, then you're home free but it's going to take you, you know, you essentially have to enumerate all possibilities before you find one.
- Okay, 0 0 0 0 1, okay? 0 0 0
- There's only one pattern that gives a one.
- Right. Exactly. And you're going to, because every single one of them is relevant. And I'm going to have to look.
- Two of them are negated. This is the only pattern that gives you a one. Now once you have found that and you know that that's the only one, now it's easy. You can just read off the equation. So, what's the equation?
- XN not two and X3 and X4 and not X5.
- And that is the, that's the equation and you are not, you're not, there's no as a learner you are not going to be able to find that, right? Because it's just a needle in a haystack until you hit it.
- Yeah, so it's, it's going to take me exponential time, but it, but remember we're not worried about time. We're worried about sample complexity. So remember the cheat that we have here. The cheat that we have here is that I know all the hypotheses and what they say.
- It doesn't help you.
- Yeah it does, because the hypothesis, cause every hypoth, well no, that's not true. I'm thinking the wrong thing. I'm sorry. I'm cheating, you're right. I'm cheating. I'm, I'm acting as if we have the example you had before.
- So this constrained-ness is really, it's very frustrating, right? Because the question that you really want to be able to ask, you can't really ask, right? You want to be able to ask a question that, that takes the hypothesis class and split it in half and. Well maybe you can, maybe you can nearly do that. But it's still going to be, oh no sorry, that would make it linear. I'm sorry, let me say that again. You'd like to be able to ask a question that, that splits this hypothesis class in half, but unfortunately almost all of your questions give very little information. Just knocks out a couple of the possible hypotheses, and so it ends up being 2 to the k kind of time, not time but samples before you can get a handle on what the hypothesis is. So, it is harder for the learner too.
- Right, so when the learner does it you have no reason to believe one hypothesis over the other. You've got all of them. And so in order to figure it out, no it kind of has to be that way because otherwise it is still linear. So, this is bothering me, because if what you said is true, then why does 20 questions work? Why do i ever get log, log 2.
- Right. So we'd like to be able to ask questions. So I, so here, let's play this game now. You think of a, a formula. And I'm going to.
- Oh, wait, you. I know the, the answer is, is that the 20 questions is still the optimal thing to do, given that you know nothing. So that, that log base 2 is kind of an expected answer, but sometimes you'll do much worse, and sometimes you'll do better.
- No, in this particular case, if I could ask you more general questions. I can do this in, in with the, you know, linear in K. So the questions that I'd like to ask you are things like, is X1 in the formula, yes or no? [LAUGH] Is X1 positive in the formula? Is X1 negative in the formula? I can just fill in these boxes by asking the right questions.
- Right.

- But, but those questions are not in our constrained set. And it's the constrained set that matters here. And our constrained set is, in this particular example just really harsh.
- So, and there's no way to approximate that, right? So I can't say, okay, so the first question I want to ask is x1 positive, negative, or absent? So, if I looked at all the hyp, if I looked at all the hypotheses I could do that by asking, now it's very hard to do, because there's no direct way to ask that question. The only way to ask that question is, I have to try. Well, I have to try all possible exponential cases to know.
- Yeah, 'because we're constrained to only ask queries that are data points, right? So give me the label for this data point. And that's not really the same as is the hypothesis you're thinking of having this particular property.
- But as soon as I get a one, I know something.
- Soon as you get a one, you're in a much happier place. So, in fact, if we didn't, if we had conjunctions of literals without negations
- Mm-hm.
- We'd be in a much better situation, because then you could, your first question can be, you know, one one one one one one. You know the answer has to be one, or the formula's empty. So then you're, you're basically off and running, but the fact that there can be negation in there means that most queries really give you useless information.
- So, so Michael, okay, so you've depressed me. You've basically said this is really hard to do, to learn because I think that we've convinced ourselves, at least you've convinced me that until I get a one, until I, I, I get a positive result, I can't really know anything. And eventually I will get one if I can just do an exponential number of samples, but then my sample complexity is exponential, and I'm sad. So what you're basically saying is, I'm sad sample complexity makes me a bad person, and there's really nothing I can do to learn anything or get anything good out of my learning process.
- That seems like a very sad way of saying it.

7.11. Learner With Mistake Bounds

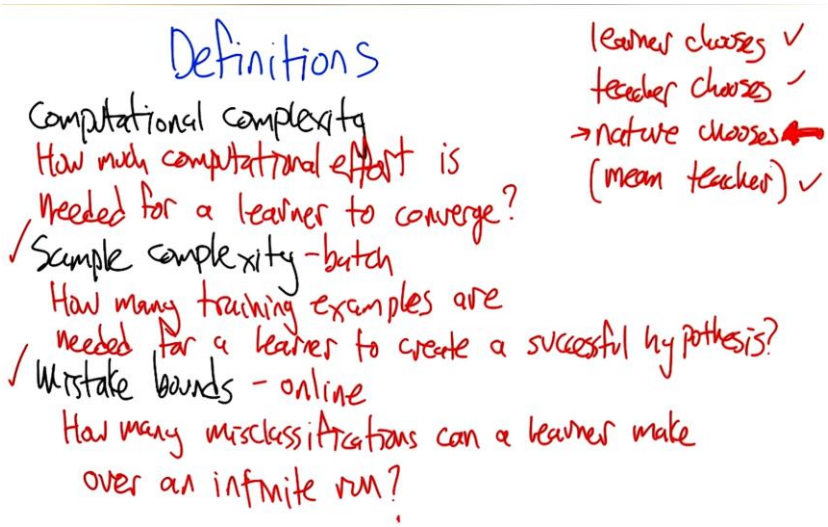


- So maybe, maybe this will make you feel better Charles. So there's we can actually, you know, when all else fails, change the problem. So let's say that instead of trying to learn the way we were describing it before, we're going to change the rules. We're going to say we're going to use a learning formalism that's sometimes referred to as mistake bands. So here's how the things work in mistake bands, the learner is sitting around and input arrives. And then the learner gets to guess an answer for that input. So the learner the, maybe the learner chose the input or maybe it came from a, a helpful teacher or maybe a malicious teacher, turns out it's not going to matter. But the input's going to show up. The learner's going to guess the answer, so it doesn't have to now guess the hypothesis and get that right. It just has to get the. The output correct for this input. If the learners wrong, then we're going to charge it a point and tell it, that it was wrong and then it goes up to one and we repeat this and so, this is going to run forever and what we're going to do is bound the total number of mistakes made, into infinity. Right? So, were going to keep playing this game forever. And we want to say it'll never make tot total number of mistakes will never be larger than a certain amount.
- So this is giving the learner some new powers, right? So the learner now is guessing answers. And all we have to guarantee is that if is guess is wrong, it better learn a heck of a lot from that. Hmm.
- Otherwise if it even if it doesn't know much if it guesses right that's fine. It doesn't have to learn.
- Okay. I see. So, so in the case before so long as I had been guessing false I would have been okay even if I didn't know what the hypothesis was. And then the moment I got a one, I got a true, I could actually learn something. Then I should learn something and try to do better guesses from that point on. Okay.
- Outstanding, yes, exactly so. So lets turn that into an algorithm. So here's an algorithm that's really simple and actually can learn very effectively for these mistake bound problems. So it, it works like this. It starts off in this weird state where it imagines that in the formula that every variable is present. In, in both it's positive and negated form. Right, which is kind of weird. And so what that, that would mean is the formula is x1 and not x1. X2 and not x2. X3 and not x3. X4 and not x4. X5 and not x5. So any input that it gets, it;s always going to produce the same answer, right? What, what is that answer.
- False. False, right, so it's going to keep saying false, for a long time. Until at some point, it actually could be right. Each time it's right, it's actually not getting charged any mistakes for that. It's just that at some point, it's going to get an input that the correct answer is true. It's going to say false, and it's going to have made a mistake. So let's say that this here, here's that example. X1 is true. X3 and X4 are true, and the other two are false, and the learner said false, but the answer was actually true. So if the answer to this one is true, what do we know?

- We know that one zero. Wait we know that 0100 1, which is the opposite of what you're saying could not be a part of the formula. So we could remove that from our formula.
- okay. That's one way to think of it. Couldn't quite think of it that way.
- But am I right? Yeah, if this is what you meant. so...
- [LAUGH]
- Uh...the first variable, the x1 not cannot be in the formula. Cause if it was, there's no way that this would've been able to produce true.
- Right.
- So we can erase x1 not. We can erase x2... in a positive form, because of the second bit. We can, the third bit says that X3, if it's in there, it can't be negated. We don't know if it's in there, but we know it can't be negated. X4 Is the same. And x5, if it's in there.
- Right you get rid of the positive. Yeah, that's what I meant. That's why I wrote down the opposite of everything that was up there.
- Yeah, and that is what we're left with. Okay, it's not exactly what the algorithm says, but it, it produced the right answer in this case. Alright, so now, now what is it going to do? Now it's going to continue to say no, unless it sees. This particular bit pattern, right, this is the only thing that will ever predict true on and it will always be right,when it does that because we know that is the correct answer for that. But let's so, so it's going to guess no everywhere else and so let's say it gets something wrong again and let's say it gets one zero, one one, one wrong. So in this particular case, it's going to guess no, and we say, I'm sorry, the answer is yes. All right, so now what does it know from that?
- Well, I'm just reading your algorithm now. And I'm just going to do what number three says.
- All right. That's a good idea. It says if we're wrong, which we are in this case, set all the positive variables that were zero to absent. All the positive values that were zero, there are none of those and said all the negative variables that were one, to absent, alright. So then that was x5, x5 is there in its negated form, but it's actually a one in the input, so we're going to turn that away to absent. Alright.
- Mm-hm.
- So and that's the same thing that you did when we were looking at the problem before, you said if you have two answers, where the, two inputs where that output is both true, any bits that are different in those two patterns, must be not part of the formula.
- Right.
- Alright, so now we've definitely, X5 is not in the formula and that's actually correct there's, there's at no point in the future where we have to revisit that. In this other cases we are not quite so sure. It could be that they're, in there or not in there. And, so each time that we get one wrong, we're going to move something from negated to absent. And when we do that, that thing is always going to be correct. So at most we can move K things from negated or positive to absent.
- Oh! So if I. . Think about, oh, so even if we may have to see in fact, every, even if we may see an exponential number or examples. We will never make more than k plus one mistakes.
- Perfect.
- And so that's exactly what I wanted to do before, right? So, if, if I'm a teacher, if I'm a good teacher in this case, then I can basically, and I knew that you started out assuming that everything was false.

- That you know, all variables were there in both their positive or negative form, I can just give you one example that is true. And that would let you eliminate half of the formula right away, and then I could just keep giving you examples that are true but only with one variable difference each time. And then eventually you would learn it. So, then the total number of samples you would need would also be k+1 if I know how you're starting out as a learner. Does that make sense?
- If we charge things by mistakes.
- No. But even if we don't charge things by mistake. If I'm a teacher who's trying to give you the best examples and I know that as the learner you're starting out with a formula that is x one and not x one, x two and not x two. Like you said before. Then I could just give you the first example that is true. That'll eliminate half of those literals. And then only give you true examples from that point on, where only change one of the variables that are left, and you'll know that you can make them absent to get rid of it, and so, just as you can only make k plus 1 mistakes, I could give you exactly the right k plus 1 examples. If I know how you're starting out as a learner. If I don't know that, then I have to do the k plus 2 that you showed before.

7.12. Definitions

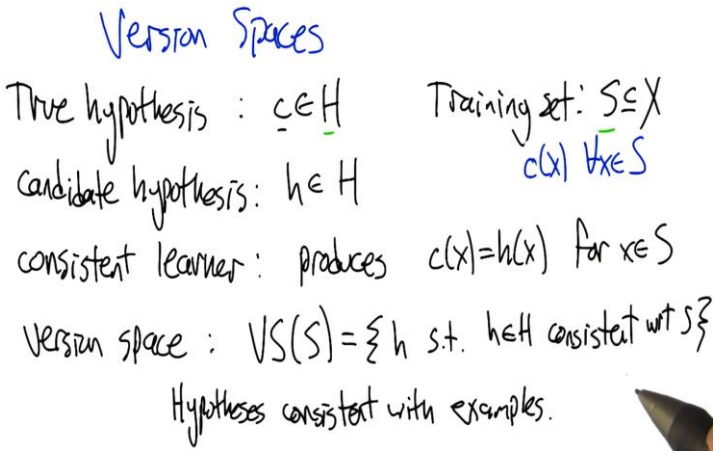


- Alright, so, remember, Charles, we were talking about three different kinds of ways of choosing the inputs to the learner. Okay? So what were they again? We just looked at two of them.
- So the learner chooses examples. The teacher, hopefully a nice one, chooses examples, and then there was the case with the examples are given to us by nature. And I guess there was a fourth one, which is. That a mean teacher gives it to us but I, you know I don't think that's all that interesting. I tend to think of nature as a mean teacher.
- Well not only that but in the, at least in the mistake bound setting that we just looked at again the, the learner was robust against any choice of where the inputs were coming from. So mean teacher it would've it would've done just as well.
- That's a fine point.
- It doesn't matter when it makes its mistakes, it's only going to make a fixed number of them no matter what.
- Okay.
- So, we've kind of dealt with three of these but we haven't really talked about the, the nature chooses case yet. And that's in some ways the most interesting and relevant and in other ways

the most complicated because you have to really take into consideration this space of possible distributions. I think now though we're in a pretty good situation in terms of being able to define some important terms and we're going to use those terms to get a handle on this question of what happens when nature chooses.

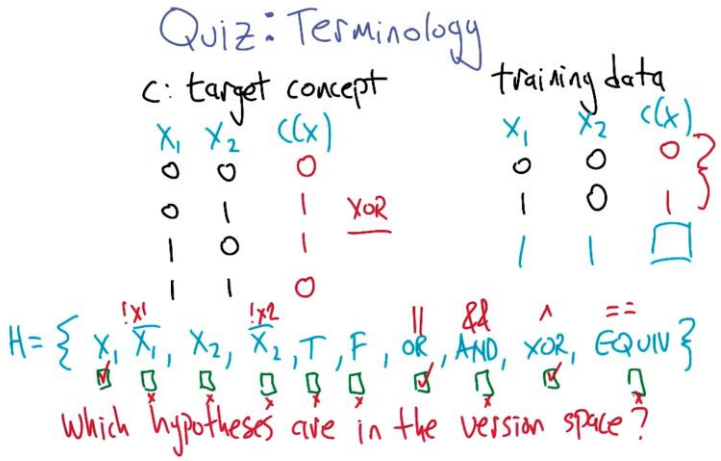
- Okay, that sounds reasonable.
- So **computational complexity**, we talked about. The, the, how much computational effort is going to be needed for a learner to convert to the answer.
- **Sample complexity** in the case of a batch, that is to say, we have a training set. Is how large does that training set need to be for the learner to be able to create successful hypotheses. And that's in the batch setting.
- In the online setting, we have this notion of a **mistake bound**. How many misclassifications can the learner make over an infinite run?
- Mind if I ask you a question? You said something I thought pretty interesting. How, for computational complexity, you said how much computational effort is needed for a learner to converge To the right answer. Is that the, is that a requirement when you talk about computational complexity? Or is just that you need to know how much computational effort is needed for a learner to converge to something?
- Well, so if it's converging to something and we don't care what it's converging to, then it's really easy to have an algorithm with low computational complexity, right? It's just like return garbage. It runs in constant time.
- Mm hm.
- So, yeah, it's in the context of actually solving the problem, that computational complexity is most interesting.
- Well, I was going to say, what if a set of hypothesis that I'm willing to entertain, doesn't include the true concept.
- good. Right. So in some sense maybe in that case what we're trying to find is the best hypothesis in the hypothesis class. But we can still ask about computational complexity in that case. So, I was saying successful hypothesis here. By that I meant, you know, whatever it is that the problem demands that we return and if it's a hypothesis class that's very limited we might just return the best thing in that class.
- Okay.
- But the important thing here is that we're not going to talk about computational [LAUGH] complexity, for the most part. We're going to focus on sample complexity for the time being. And that's really the relevant concept when we're talking about the idea of nature choosing.
- I believe you

7.13. Version Spaces



- All right. We're going to do a couple more definitions. This notion of a **version space** turns out to be really important in understanding how to analyze these algorithms. So, imagine we've got some hypothesis, space H, capital H. And a true hypothesis that we're trying to learn, C of H. This is also sometimes called a concept. And we're trying to learn from a training set S, which is a subset of the possible inputs. And what our training set consists of is those examples along with the true class for all of those X's. So, at any given time a learner might have some candidate hypothesis, little H and big H, and a learner who produces a candidate hypothesis little H, such that C of X is equal to H of X for all X of the training set, is called a **consistent learner.** Right, so what would be another way of describing hat a consistent learner is?
- A consistent learner can actually learn the hypothesis, or the true concept.
- Well it produces, right, I mean of all the possible things it could return, it returns the one that matches the data that it's seen so far.
- Right, so it's consistent with the data. That makes sense.
- Yeah. And the version space is essentially the space of all the hypotheses that are that are consistent with the data. So we'll say the version space for a given set of data S is going to be the set of hypotheses, that are in the hypothesis set, such that they're consistent with respect to the samples that they're given.
- Okay that makes sense.

7.14. Quiz: Terminology

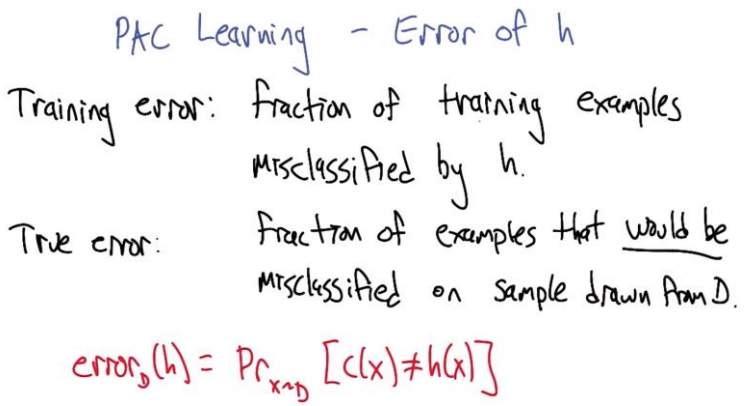


- Alright, here's the quiz. So just to practice this concept of what a version space is let's go through an example. So here we go. Here's the actual target concept C and, for all possible, it's, mapping from two input bits to an output bit. And the two inputs, X1 and X2 can take on all four possible different values and the outputs are zero One, one, zero, which is to say the XOR function, okay so that's what we're trying to learn.
- Okay.
- But the training data that we have only has a couple examples in it. It says well, one training example says if the inputs are zero and zero. The output is zero. And if the inputs are one and zero, the output is one. And, ultimately, we're going to ask questions like, well, what happens if the input is one and one? What's the output? Now, since we know that it's XOR, we know that the answer is zero but that's kind of using information unfairly. So here's what we're going to do. Here's a set of, a hypothesis set. These are set of functions. That says, well the output could be just copy X1, negate X1, copy X2, negate X2, ignore the inputs and return true, ignore the inputs and return false, take the OR of the inputs, take the AND of the inputs, take the XOR of the inputs and then return whether or not the inputs are equal to each other. And, what I'd like you to do is, some of these are in the version space and some of them are not for this training set that I marked here. So, can you check off which ones are in the version space?
- I think I can.
- Awesome. Let's do it. Answer
- All right Charles, what do you think?
- Okay, so being in the version space just means that you're consistent with the, data that you see. Right?
- Good. Mm-hm.
- Okay, so we should be able to very quickly go through this. So X1, just copy X1 over. Well, if we look at the training data, in the first case, X1 is zero and the output is zero. Second case, X1 is one and the output is one. So that is, in fact, consistent with just always copying X1.
- So that is in the version space, right?
- Right. And without even having to think about it, since x1 is in the version space, doing the opposite of x1 can't possibly be in the version space.
- Agreed.
- So let's skip that. Looking at x2, we can see that in the first case, you go x2 0, c of x is 0. Yeah, so yeah that's looking good. That's consistent so far. But then on the next row you get 0 and then

you get the, opposite of 0. You get 1, the compliment of 0. So, that definitely is inconsistent with just copying over X2, so you can't have X2 and by very similar reasoning you can't have the opposite of X2.

- Agreed.
- Okay, so, true is clearly not consistent because we got a 0 once and a 1 once.
- Mm-hm.
- And by the same argument false is not consistent because we got a zero once and a one once. Now or, let's see or.
- But in case it's not clear, I probably could have been clearer about this, but here, zeroes and falses are the same thing and ones and trues are the same thing.
- Right. Just like in C.
- [LAUGH]
- Okay. So I just assume everything you do is written in C, Michael, so, it just works that way. In fact there's a C right up there at the top of the, at the top of the slide.
- Yeah so I feel like I should do this.
- Now I understand what's going on.
- Right, now it's in C.
- Much better. Okay, so, or. Or means if either one of them is true, then you say yes and, huh! That's actually consistent with or.
- Yep.
- Hm. But it is not consistent with and, because one and zero would be zero, not one. Second case, XOR, well, I already know it's consistent with XOR, because I happen to know that that's the target concept.
- Yeah.
- And an equiv would be not consistent. Though interestingly, not equivalent would be consistent.
- Yes, because not equivalent is XOR.
- Oh, yeah, that's right.
- All right, so that's, yeah, that's it, X1, OR, AND, XOR.
- Excellent. Cool.

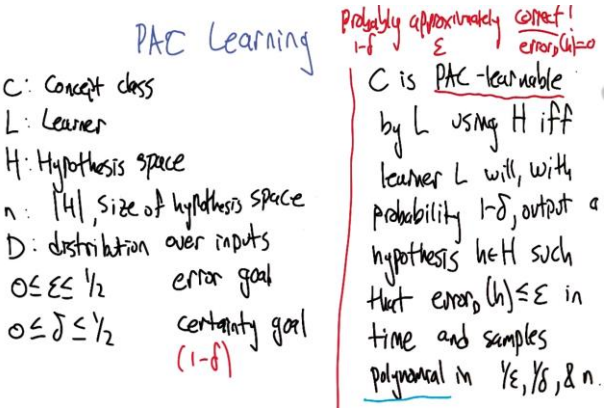
7.15. Error of h



- Alright, the next topic we're going to get into is to nail down a concept called PAC learning. So to do that, we're going to delve into what the error of a hypothesis is. And there's two kinds of errors. There's the training error and the true error. So the **training error** is on the training set. What is the fraction of those examples that are classifieds by some given hypotheses H.

- Okay.
- Now H could be different from the target concept. The target concept ought to have a training error of zero. But some other hypothesis h might have some error. Okay?
- Sure, that makes sense.
- Okay, so the **true error** is actually the fractions of examples that would be misclassified on a sample drawn from D in essentially the infinite limit. So what is the, essentially the probability that a sample drawn from D Would be mis-classified by some hypothesis, h.
- Okay.
- And we can actually write that mathematically. Error with respect to some distribution D of some hypothesis h, is the probability that if we draw the input X from that distribution that you're going to get a mismatch between the true label, according to the concept, and the hypothesis that we're currently evaluating.
- Right, so this sort of captures the notion that it's okay for you to misclassify examples you will never ever ever see.
- Yes, that's right. So we're only being penalized proportional to the probability of getting that thing wrong. So, for examples that we never see, that's fine. For examples that we see really really rarely. We get only a tiny little bit of contribution to the overall error.
- Okay, I can follow that.

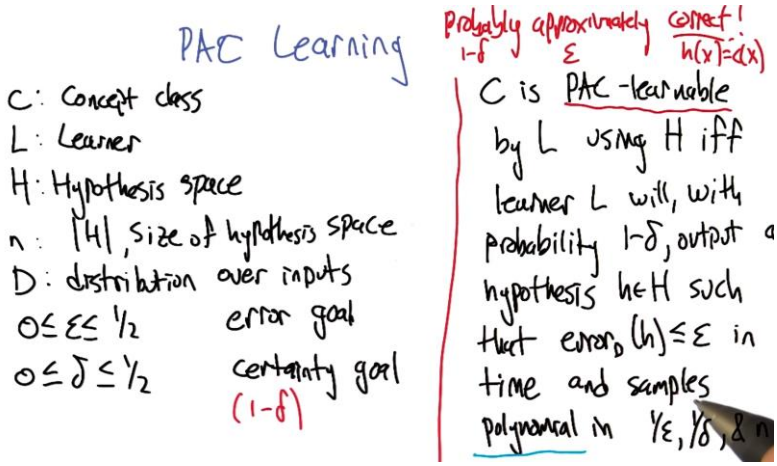
7.16. PAC Learning



- Alright, with just a couple more definitions we'll be able to nail down what PAC Learning is. So let's consider a, concept class C. That is to say that the set from which the concept that we're trying to learn comes from. L is our learner. H is the hypothesis space, so it's the set of, of mappings that the learner is going to consider. N is going to be the size of that space. So kind of the space of the hypotheses that are being considered. D is that distribution of inputs like we looked at before. Then we got Greek letters epsilon and delta, where epsilon is our error goal. Which is to say we would like the error in the hypothesis that we produce to be no bigger than epsilon. It could be smaller, that'd be great. It could be zero, that'd be awesome. But it can't be bigger than epsilon. But, we could get really unlucky and actually not meet our error goal. And so delta is what allows us to set a, a kind of uncertainty goal or certainty goal, which is to say, that with probability one minus delta, the algorithm has to work.
- And by work, you mean has to. Be able to produce a true error, less than or equal to epsilon.
- Exactly.
- So why can't we always force epsilon to be zero, and, you know, delta to be zero?
- Right, to be absolutely sure that we get zero error. So the, part of it is because we are sampling training examples from this distribution D. and it's always possible that we, for example, unless well, as long as there's probability mass on more than one example, there's always a probability that we draw a finite sample that only ever gives up one example over and over again.
- That's fair.
- So we just have to be really careful about that. So, so if we're that unlucky it's very unlikely to happen but when that happens our error is going to be very high. But that's okay because it won't happen very often.
- Okay sure,sure. So basically you can't force it to be zero under all circumstances either epsilon or delta you can't force them to be zero under all circumstances. So you have to allow somewhere to.
- Exactly, it gives us the wiggle room we need. Thats actually where this name **PAC** comes from. It stands for **probably approximately correct**.
- Oh I see.
- So we would like to be correct, but then we are just going to keep adding weasel words until we can actually achieve it. We would like to be correct, but we cant be exactly correct, so lets be approximately correct. But we can't be approximately correct all the time, so let's only probably be approximately correct.
- So using your words here you could have called that one minus delta epsilon correct.

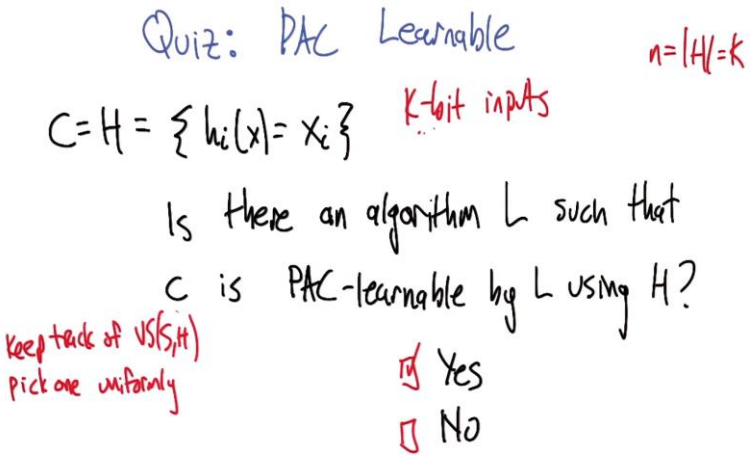
- Yes, right. That's exactly right. That's what the Greek letters are playing, and correct is the, this you know, error Sub D of H equals 0.
- Yeah, so 1 9 is delta upsilon error sub D of H equals 0 doesn't roll off the tongue quite as well as probably approximately correct.
- I would agree with that.
- Okay fair enough.

7.17. PAC Learning Two



- Alright, now we can actually dive in and give a definition for PAC-learnable. So, a concept class, C, is **PAC-learnable** by some learning algorithm, L, using its own representation of hypothesis, H, if and only if that learner will, with high probability, at least from its set of hypothesis. That has error less than or equal to epsilon, so it's very accurate. And it needs to be the case that the time that it takes to do this. And the number of samples that it needs draws from this distribution D is relatively small. In fact, bounded by a polynomial in one over epsilon, one over delta and the size of the hypothesis space n.
- Okay, so in other words you're saying something is PAC-learnable if you can learn to get low error, at least with some high confidence you can be fairly confident that you will have a low error in time that's sort of polynomial in all the parameters.
- Exactly, and you can see here that this, this epsilon and delta actually giving us a lot of wiggle room, if you really want to have perfect error. Or perfect certainty. Then these things go to infinity. So, you just, you need, you need to look at all the possible data.
- Mm.
- So, yeah this is really going to only give us partial guarantees.
- Okay. Sure. Okay, I think I understand that.

7.18. Quiz: PAC Learnable

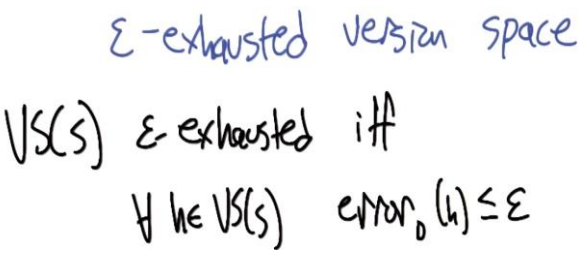


- Let's just do a little quiz. So, here is our concept class and our hypothesis class, which are the same in this case, which is the set of functions H sub I, that return, for an input X, it returns the Ith bit of that input.
- Mm-hm.
- All right. So if we're talking about K bit inputs then there's going to be a hypothesis in this set for each of those K bits, and that hypothesis returns the corresponding bit. And so, we're given a training set with examples like that. And then we need to produce outputs that are consistent with that. And so what I'd like you to try to figure out is, tell me, do you think that there is an algorithm L, such that C is PAC learnable by L using H? So if you can give me a learning algorithm that would do a good job, and would, with high probability, be able to figure this out, then you should go for it.
- Okay. Answer
- Okay, so what'd you think? Was that enough information to kind of chew on it a little bit?
- Maybe. [LAUGH] So, let's see, I have, I guess, k hypotheses I have to chose from. I know that basically you always return the output of one. I don't get to chose what those examples are, they're drawn from some distribution.
- Right.
- And I want to know whether I'm going to need to see, another of examples that are, polynomial, in the error that I'm interested in, with the certainty that I'm interested in. And I gotta be able to come up with an algorithim, a learning algorithm that will do that.
- Exactly. So what do you think?
- I want to say the answer is yes.
- So how would you argue that though? Do you have a particular algorithm in mind?
- Well I actually did, I was going to keep the all the hypothesis that are consistent with the data that I've seen.
- Okay, alright and what is that called?
- The version space.
- Right. Okay.
- So keep track of that and then, whenever I stop getting samples, I have to pick one of those. So I'm just going to, pick one of them uniformly.
- Okay. Well that seems good so far. You could pick one uniformly. So what makes you think that that's actually going to. Be able to return the right answer, or a, a close enough answer with not

that much data. How do we actually bound the number of samples that we need to make it so that when we pick uniformly we actually get a good answer?

- Well, it's a little hard, I mean my, my intuition is that, given what we don't know what the concept is only the class that it comes from If I do anything other than choose uniformly, then I can be very unlucky. I basically... uniformly basically means in this case I don't know anything else to do. So there's sort of no better algorithm than the one that we just came up with, which is find all the hypotheses that are consistent... And you have no reason to choose one over the other because you don't have anymore data. So you should just close your eyes and pick one. And if you do anything else, then in the absence of some specific domain knowledge that tells you to do otherwise, you know, you can just basically end up being very unlucky.
- So, I, I agree with you and this is a good algorithm. But what we lack at the moment is an argument as to why the number of samples that it needs, isn't exponential. Right? Cause there's an exponentially. Large, you know, 2 to the K different possible inputs. If we had to see all of them to be able to guess right, that would be too many.
- Mm hm.
- So, we really want it to be, you know, a polynomial in, in K. Not an exponential in K. So I'm going to say that we don't have enough background yet to be able to answer this definitively. The, yes is the right answer. But we're going to need to dive in a little bit more deeply, and develop some more concepts to be able to argue why that's the right answer.

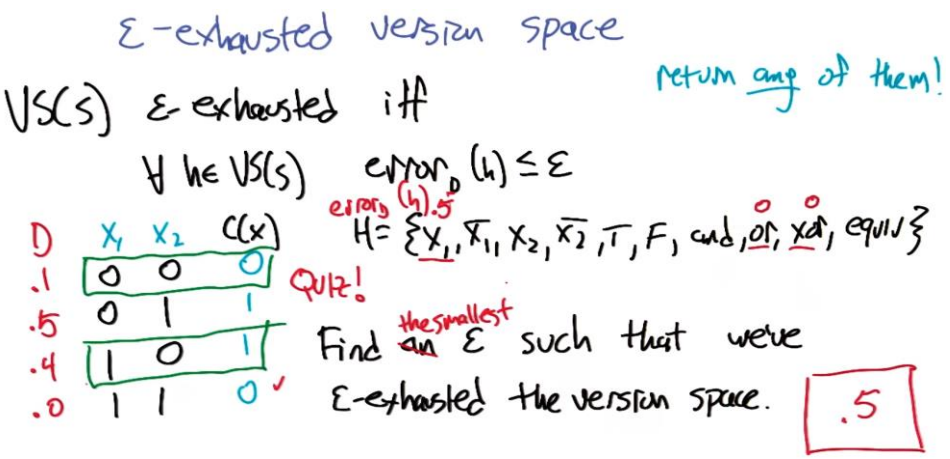
7.19. Epsilon Exhausted



- So this is going to be a key concept for being able to develop and answer two questions like that and it's the notion of epsilon exhaustion. Which sounds kind of tiring.
- I know I'm epsilon exhausted right now.
- Yea, me too. So what we mean by this is, well here's the definition. So, a version space. Of, version space that is derived from a particular sample, it's considered epsilon exhausted if and only if for all the hypotheses that are in that version space they have low error. So if we can do this then your algorithm going to work. Your algorithm says at that point choose any of the hypotheses in your hypothesis set. You are going to be fine, you are going to have low error.
- Sure.
- If you don't do this, your algorithm is going to be in trouble because you are uniformly at random choose one of the hypothesis and it has a chance of being wrong. It could be a fairly high chance if there is, say there is only two hypothesis left in the version space, one has high error and one has low error. We really have to make sure that the only things left in the version space have low error
- Okay, that makes sense, that makes sense.
- Alright, so we're going to have to develop a little bit of theory to figure out when that occurs but this is a really key concept.

- Okay, so, I guess if was trying to put this into English just to make certain I understand it, what you're saying is, something is epsilon exhausted, a version space is epsilon exhausted exactly in the case when everything that you might possibly choose has an error less than epsilon.
- Sure.
- Period. And so if there's anything in there that has error greater than epsilon, then it's not epsilon exhausted.
- Right. It's still epsilon energized.
- Right. That makes a lot of sense, epsilon energized. That's a pretty good name for a band. OK.

7.20. Quiz: Epsilon Exhausted

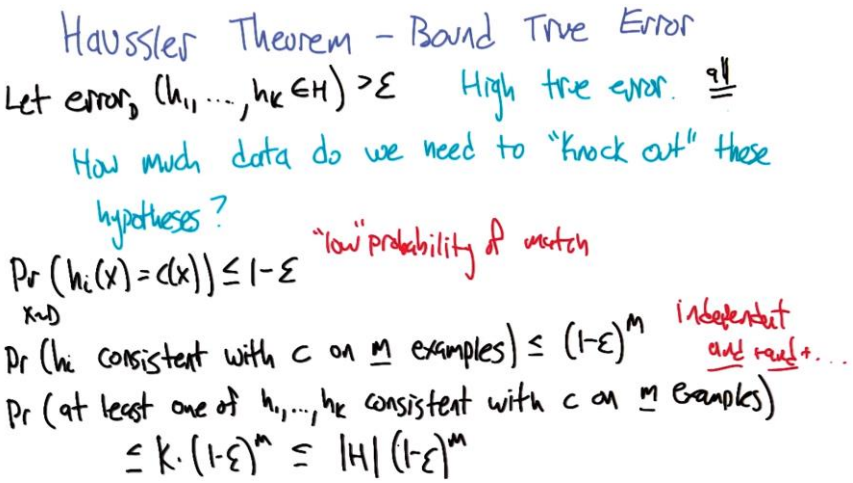


- Alright, and just to make sure this epsilon exhaustion conpcet is clear, let's actually use it in a quiz. So, here's our example from before. We've got our target concept, which is X or, we've got our training data which is these, with the things that are in the green boxes here, 0 0 output 0 1 0 output 1. And this time, I'll actually write down a distribution D that gives the. Probability of drawing each of these different input examples. All right? So now what we'd like to do is find an epsilon such that that this training set that we've gotten has epsilon exhausted the version space.
- Okay. I got it. I think I got that.
- You think you, you think you can work that one through?
- Yeah.
- Anything else that we'd have to tell you? Remember again that These are the hypothesis in the hypothesis set.
- huh. Yeah. Yeah. I got it.
- Alright. Let's, let's, let's see what you got.
- Okay. Answer
- Okay, so one.
- See, now that was mean. So right, one is an epsilon, no because epsilon has to be less than or equal to half, we already said that, but you're right. In a sense setting epsilon to one, is always a valid answer to the question, find an epsilon set that we've epsilon exhausted the version space. Because all it's saying is, that there is nothing left in the set that has an error greater than one. And since the error is defined to be the probability, it can't be greater than one. So that was kind of you know, kind of rude.
- All right, I, I thought I just answered the question. But I guess you want me to give you

- Yeah but you, but you prob-, what you probably should have pointed out is that I left out the word smallest.
- Oh! Yes, yes. Okay. Well, so, I don't know the answer to that but, I think I could walk through it very quickly.
- Okay.
- Okay, so you saying the ones that are in green are the training examples that we see, right?
- Right.
- So we should be able to use that to figure out what the version space actually is.
- Right, which we did in a previous question.
- Right, although I don't remember [LAUGH] what the answer was.
- I'll remind you.
- I'll remind you,It's okay. So it was x1
- Mh-hm.
- Right, because the x1 matches, it was, or and x or.
- Mh-hm.
- I think that was it.
- Yeah, I think that's right. Okay, so, then what we can do is given that those are the three things that we've done. We could actually compute, what the error is according to this distribution for each of those three.
- Yes, exactly so.
- So let's, let's start with X1. So which one, x1, so all three of those are going to get the first one and the third one correct, right?
- All of them are going to get the first one and the third one correct. Yes, by design.
- By design.
- Right, to be in the version space.
- So now we can ask which ones will get the second one wrong? The fourth one doesn't matter because it has zero probability of showing up.
- That's right. So, it doesn't matter if you get this one right or wrong, it's not going to contribute to this true error measure.
- Okay. So let's look at x one. So x one will in fact get the second one wrong, because the output Is not the same as the value for x one.
- Good. And so what;s the probablity that x one, this hypothesis x one, is going to give a wrong answer on a randomly drawn input?
- Well, half the time it will get the second answer, and so the error is, in fact, one half.
- Yes. Exaclty. Good. All right. Let's move on to the or.
- Okay. So, we can do an easy one actually. We can do xor. Since we know xor is the right answer, we know It will has a probablity of being wrong of zero.
- Oh, good point.
- Okay. And so for or we can do the same thing. So is, we know it's going to get the first and the third ones right. So now we can ask whether it's going to get the second one, right. And zero or one is in fact true. Or one. So in fact it also has an error of zero.
- Okay.
- Which is kind of interesting. So and so, even though the function is xor, if we can get to the point where we have or or xor left, we actually will get zero true error.
- That's right.

- But in the meantime, because x1 has still survived the two examples that we have. Epsilon is therefore 0.5.
- Right, in particular, we're saying that, this is, if, if epsilon were smaller than 0.5, then it wouldn't be epsilon exhausted because you'd have a hypothesis that has error that's too high.
- Right.
- So this is the smallest epsilon that we can use. And in fact, we let you through if if it was anything value that I was really hoping you'd be able to reason out.
- Okay, well that all made sense.
- Good, nice work.
- Thanks.

7.21. Haussler Theorem

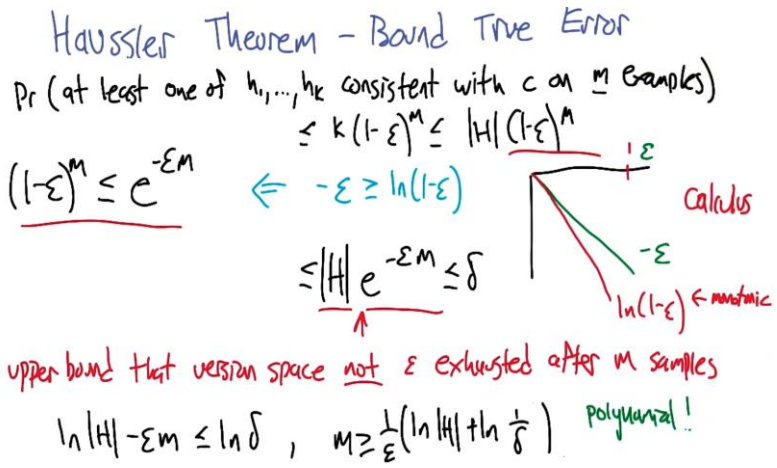


- Alright, so now we're ready to dive in and actually work out some math that turned out not to be so bad, but it was, it ends up being okay specifically because we were very carefully about setting up all the definitions to lead us to this moment in time. It is our destiny, Charles.
- Oh good. I love destiny. Is this destiny's theorem?
- No, actually turns out it's **Haussler's Theorem**.
- Is Haussler like German for destiny? M:It might be, but I don't think it is. So, Hausser is the name of a person in this particular case. And what he worked out is a way of bounding the true error as a function of the number of training examples that are drawn.
- Oh. Nice.
- So, let's consider. From the hypothesis set that we have, all the hypotheses can be categorized as to whether or not they have high true error or low true error.
- Sure.
- Right so, let's let H1 through HK be the ones that have a priority of high true error. What we'd like to do is make sure that as we're drawing data sets that we have knocked all these guys out. We've gotten enough examples that actually allow us to verify that they have high error. So they have high error on the, on the training set. So they have high training error. Alright, so how many, how much data do we need to establish that these guys are actually bad? Alright, so let's take a look at the probability that if we draw an X, an input from this distribution D. That, for any of these hypotheses. Hi, and this set of bad hypotheses. That it will match the true concept, right? So that H sub I of X is equal to C of X. And we know that that's less than or equal to one

minus epsilon. It's unlikely that they match. Because it's likely, or relatively likely, that they mismatch. Right? That's what this exactly means. This, this error being greater than epsilon.

- Oh, I see, so if I have an error of greater than epsilon, that means that the probability that I'm wrong is greater than epsilon, which means the probability that I'm right is one minus epsilon, less than one minus epsilon. Okay, that makes sense.
- Yeah, so it's sort of a relatively low probability of match.
- Well, if epsilon is high.
- Low relative to one minus epsilon.
- Right, okay.
- So this is, this is a fact about the hypothesis in, in the abstract. For any given sample set we've got a set of m examples, and what we'd like to know is, since we're trying to knock it out, what's the probability that even after we've drawn m examples that this hypothesis, h of i, remains consistent with c. Right, even though the data doesn't really match all that well, we've drawn M examples, and it still looks like it matches. It's still in the version space. So the probability that that happens if it were the case that everything was independent is going to be one minus epsilon raised to the M power. Right. Because it's less than one minus epsilon to be wrong once Well, which is to say, that your consistent ones. To continue to be consistent, we keep having to have this probability come true. So, it's going to be, we're going to just keep multiplying it in again and again and again. So one minus epsilon raised the the m power.
- Right. So that makes sense because you're basically saying it's Consistent with the first example and the second example and the third example and dot dot dot nth example. So, and with independent variables is just multiplication so it's one minus epsilon times, one minus epsilon times, one minus epsilon n times. Okay, I see that.
- Great. Alright, so can we use that to figure out what's the probability that at least one of these h1 through hk's is consistent with c on m examples. We have to knock them all out. That's...that's really what the goal is. To knock out all the ones that have high true error We failed at that. If one of them still slips through, one of them still looks consistent and remains in the version space. So what's the probability that at least one of these remains consistent. That this still has happened.
- I think I know that. So just like you did add before and did multiplication... Another way of writing at least one of is to say or. So h1 or h2 or h3 or h4... or hk is consistent. And just like and is multiplication, or is addition. That's true.
- And there are k different ones of these, so I have to say this one minus epsilon to the m plus one minus epsilon to the m plus one minus epsilon to the mk times So that would be one minus epsilon to the m times k.
- Great. So that is a bound on the probability that at least one of these bad hypothesis is going to remain in the version space even after m examples. But how many bad examples, how many bad hypothesis are there? What's an upper bound on that?
- Well, there might be one or there might be two. There's actually k of them, but we know that k itself is bound by the total number of hypotheses.
- Yeah, that's right. So it has to be, you know, the number of bad hypotheses. We, we assume if c is equal to h anyway that there's at least one that is not bad, but, you know, almost h of them can be bad. So that, that should give us a bound. Alright, so now we're going to to work on this expression a little bit more to put it in a more convenient form.
- Okay.

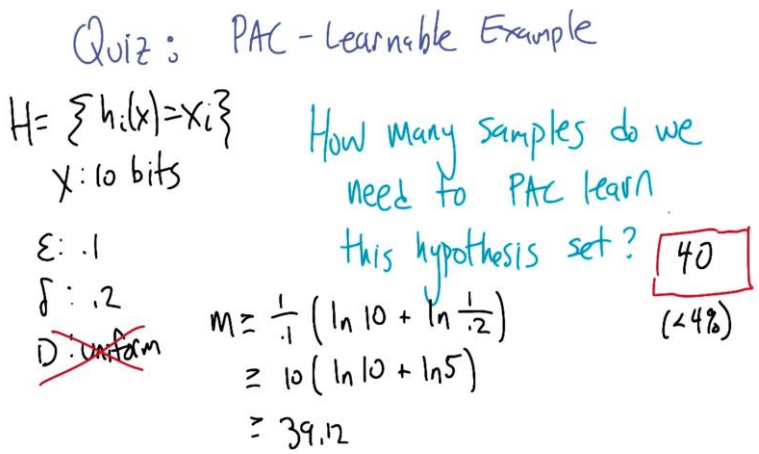
7.22. Haussler Theorem Two



- All right, so it turns out there is going to be an useful step here. Which is, we are going to take advantage of the fact that minus epsilon is greater than or equal to the natural log of one minus epsilon; which maybe it's not so obvious but if you plot it you could see that it's true. If this is the epsilon axis then minus epsilon looks like a straight line going down like that.
- Sure it's got slope minus one.
- Yep, and the log of one minus epsilon looks like this, it starts off, they'll they're totally lined up at zero, epsilon zero.
- Sure because one minus zero is one then absolute log of one is zero.
- Exactly, and then what happens is it starts to fall away from it, the slope is actually, I mean you could, you can test this by taking the derivative of it but the slope is if it's you know it's monotonically I'm changing [LAUGH] so that it falls away from the line and always stays below it,okay, so if we believe that, which, you know I'm going to just say calculus.
- Well, you can kind of see that, right? Because when epsilon is one, that would be the natural log of zero and the only way you can raise e to a power and get zero, is by having effectively, negative infinity.
- Yeah, so right, by then, it's definitely below and but, it stay below all along, I mean, because, just that is not enough, because.
- Well, it has to stay below all along because natural log is a monotonic function. Alright, so it can't like, get bigger and then get smaller again, so, yeah, okay I buy that.
- Good, alright, so if that's the case,if we accept this line, then, it's also going to be the case, that one minus epsilon to the m, is than or equal e to the minus epsilon m, so why is that? So, if you multiply both sides by m, and then take each of the both sides, you get exactly this expression.
- Sure. Alright? So now that we've gotten that, we can use it here in our derivation and rewrite that as, the size of the hypothesis space times e to the minus epsilon m. Alright that gives us another upper bound on the quantities that we had before and this is much more convenient to work with. The epsilon which had been kind of trapped in the parentheses with the y minus now comes up to the exponent where we can work with it better.
- Sure.
- Alright, so what this is is an upper bound that the version space is not epsilon exhausted after m samples.
- Mm-hm.
- And that is what we would like delta to, we would like delta to be a bound on that.

- Mm-hm.
- Right, so if delta is the,is the failure probability, essentially. So the failure probability ought to be bigger than or equal to this expression here, alright?
- Okay.
- So now, the last thing we need to do, is we can just re-write this in terms of M.
- Mm. So if we do that, let's see what happens. All right, when we're done rewriting that what we find is the sample size M needs to be at least as large as one over epsilon times the quantity the log of the size of the hypothesis space plus the log of one over delta.
- Okay and that is polynomial in one over epsilon, one over delta, and the size of the hypothesis base.
- Indeed!
- Nice.
- So that's pretty cool.
- That is pretty cool.
- So, right, it tells us if you know the size of your hypothesis base, and you know what your epsilon and delta targets are, you know, sample a bunch, and then you'll be okay.
- That's pretty good!

7.23. Quiz: PAC Learnable Example



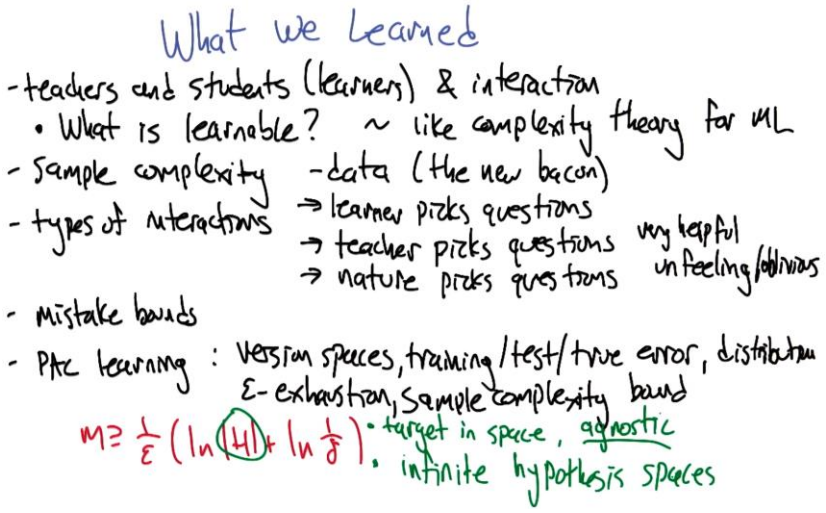
- This puts us in a good position to be able to go back to that question we looked at before, so lets look at it a little bit differently this time. If our hypothesis space is the set of functions that take 10 bit inputs and there is a hypothesis corresponding to returning each of those 10 bits, separate hypothesis, one returns the first bit, one returns the second bit. And so on, and now we have a target of Epsilon equals .1, so we would like to, return a hypothesis whose error is less than or equal to .1, and we want to be pretty sure of it, the error, or failure probability needs to be less than or equal to point that our distribution of our inputs is uniform.

  $$m \geq \frac{1}{\varepsilon}(ln|H| + ln\frac{1}{\delta})$$

- So given this setup, how many samples do we need to pack learn this hypothesis set.
- Okay.
- And remember the algorithm that we're going to use is we're going to draw sample of the size that we want. Then we are going to be confident that we've epsilon exhausted that, that, version space. And so anything they left in the version space should have low error. And that procedure should fail with probability, no more than .2.
- Right. So it's just exceeds probability .8.
- Yeah.
- Or. Better.
- Okay, think we can work that through?
- I think we can.
- Alright let's do it.
- Okay. Cool. Answer
- All right Charles, what do you think?
- I don't know but I know how to do it.
- All right.
- I'm just going to substitute any equation we had before.
- Yeah, that's what I was thinking, uh-huh.
- Alright, so M is greater than or equal to, 1 over epsilon times the natural log of the size of the hypothesis space.
- Mm, which is what, that is not one of our variables here.
- ten.
- Yeah. Right, so it's not 2 to the 10. Even though the input space is 2 to the 10. The number of hypotheses. There's one hypothesis corresponding to each of the bit positions. So, good?

- Right. Plus, the natural log of 1 over delta. So that would be greater than or equal to ten times the natural log of ten.
- [LAUGH]
- Plus the natural log of, five. So, let's see. The natural log of ten is something like three point something, the natural log of five is something like, two point something. We add those up, multiply by ten you're going to end up with 39.12.
- Good, so, we need, you know, 40 samples?
- Yeah. That sounds about right.
- That actually doesn't sound too bad. Well, you know, it's not learning a very hard problem, but it's, you know, a pretty big input space. So let's see. What, how big is the input space? It's like two to the ten, which is.
- 1,024.
- 1,024. So how much of 1024 is 40? It's, it's, you know, less than 4%. Hm, that's not bad.
- Before we leave this quiz, let me point out one more thing: that this bound is actually agnostic to the distribution from which samples came, so this idea that it's from a uniform distribution is actually not being directly used here. So so this is pretty cool. It actually doesn't matter, we only need 40 samples no matter what the distribution is. It's not like some distributions are harder or easier ,because we are measuring the true error on the same distribution that we used to, to create the training set. So if it's a really hard distribution and some tough examples never appear, then we're unlikely to see them in the training set, but they're not going to contribute very much to the true error.
- Well that makes sense. So the distri, oh right. So in some sense, I mean, I guess the equation doesn't show this, but in some sense, the distribution is, cancels out between the training and the true error
- Yeah, that's one way to think about it.
- Well, I like that. So 40 is pretty good to get 10% error. If we wanted to get say, only 1% error, Then we would go from 40 to 400.
- Mm.
- Right?
- That's a good point, yeah.
- And it's, it's, it's one decimal point even. And so, that would be about 40% of the data.
- Yeah, that's true. Yeah, if we want to go a little bit beyond that we may need all the data multiple times.
- Mm-hm.
- Yeah, but this example doesn't look so bad. So let's just move on before we think about it too hard.
- Okay. That seems fair, I like that.

7.24. What Have We Learned



- So actually that was all we were going to talk about in this lesson about computational learning theory. So let's just recap where we went so far.
- Okay. So what? You want me to do it?
- Yeah, that's been our technique all along.
- Fine. So here you're the teacher and I'm the student. I get that. Which is actually one of the things that we talked about. Aha.
- We talked about what it would mean to be a learner versus being a teacher. And how teachers and learners interact to make learning happen faster or not.
- Okay.
- But that was actually in a larger context which I thought was kind of cool which was this sort of notion of trying to understand what is actually learnable. Right and I think the comparison that made sense to me was that we were trying to do the equivalent to what we do in computer science with complexity theory and algorithms. While here we were bouncing up from a specific algorithm like decision trees or. KNN and asking a question about how fundamentally hard is the problem of learning.
- Good.
- And you know, like that. And we focused on a particular measure of difficulty, which I guess drove everything else, so we talked about which was sampled before it was excepted. Okay, how many examples, how many samples do we need in order to learn some concept?
- Good yeah, that's a really powerful idea because it's a different resource than what's normally studied in computer science, things like time and space. This is now how much data do we need?
- Right, which makes sense because we do machine learning and machine learning people, what we care about is data.
- I, I saw a t-shirt recently that says data is the new bacon.
- Mm. So you're saying data is delicious?
- Yeah, I think we like data a lot.
- I love data. Okay, so that ties us back into a discussion about teachers and students because what we talked about was how If the relationship between the teacher and the student was one way versus another way, we might get different answers about sample complexity. So in

particular, we talked about what would happen in a world where the learner had to ask all the questions.

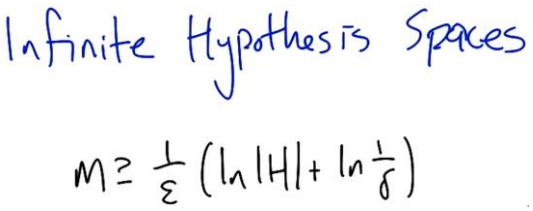
- And that's powerful because the learner knows what the learner doesn't know but the learner doesn't know what the learner needs to know. So that is somewhat powerful. But it may be useful for the teacher to be more involved.
- Right, so that's the other thing, where the teacher, gets to actually pick the questions.
- Great.
- And then the third sort of case was where. The teacher didn't really pick the questions or the teacher didn't have an intent to pick the questions, but the teacher was, in fact, nature, so like a fixed distribution.
- Yeah, good.
- Right. And some of those are, you know, easier to deal with than others, like the teacher, since the teacher knows the answer, can ask exactly the right set of questions and get you there very quickly versus, say, when the teacher is just nature. And, you know, you get it according to whatever distribution there happens to be.
- Sort of oblivious, maybe, is a better word.
- I think unfeeling.
- Nature just doesn't care about me.
- I think nature cares about you just as much as nature cares about everyone else.
- Yeah, that's exactly what I was afraid of.
- Yeah. Okay so let's see, what else did we cover? So we talked about mistake bounds as a different way of measuring things.
- Hm. You know how many mistakes do you make as opposed to how many samples do you need. That was kind of neat.
- Yeah.
- I know there's some tie-in there. And then the bit that I like a lot is that we started talking about version spaces and PAC learn ability. And what really worked for me with that was this distinction between training error which we talked about a lot. Test error which is how we've been thinking about all of the assignments we've been doing, and true error. And true error in particular got connected back to, to this notion of nature.
- Right, the distribution d.
- Right. And then you introduce the notion of epsilon exhaustion of version spaces, and it gave us an actual sample complexity bound For the case of distributions in nature.
- And the sample complexity bound is pretty cool, because it depends polynomially on the size of the hypothesis space, and the target error bound and the, the failure probability.
- Hm. So actually that reminds me, I had two questions about this one.
- Uh-huh?
- So, the first question was, that equation, m greater than or equal to one over epsilon times the quantity, natural log size of hypothesis space plus natural log of one over delta, close quantity.
- Assumed that our target concept was in our hypothesis space, didn't it?
- Yes, that's true.
- So, whatever happens if it isn't?
- Then we have a learning scenario that's referred to in the literature as agnostic, that the learner doesn't have to have A hypothesis that is in the target space. And, instead, needs to find the one

that fits nearly the best of all the ones in there. So it doesn't have to actually match the true concept. It has to, it has to get close to the best in its own collection.

- Okay, well, so, do we get the bounds?
- It's very similar bound. I think, I think maybe there's an extra epsilon, there's an extra squared on the epsilon.
- Hm. Okay, okay.
- And I think there's maybe slightly different constants in here. So it's, it's a very similar form. It's still polynomial. It is worse though because it, the learner has kind of less strength to depend on.
- Okay, that's fair. Okay, so then my second question was, I just realize staring at this now since you wrote it in red, that the bound depends upon the size of the hypothesis space.
- Indeed.
- So what happens if we have an infinite hypothesis space?
- Well, according to this bound, The technical term is your hosed.
- Oh, is that what the h stand for?
- Yes.
- Hm, so n would be greater than one over epsilon times the natural log of infinite which I'm pretty sure is infinite.
- Yeah, even with the, even once you multiply it by one over epsilon. So yeah, you know, this is a really important issue, and I think it really deserves its own lessons. So let's, let's put this off to lesson eight. You're right that the infinite hypothesis spaces come up all the time. They're really important. They almost everything we've talked about so far in the class, like actually learning algorithms, deal with infinite hypothesis bases, we would really like our bounds to deal with them as well.
- Yeah, I would like that.
- So, I know, anything else to talk here, or should we say, without further ado, let's move on to the next lesson?
- I think we should say, without further ado, let's move on to the next lesson.
- Alright then, without further ado, let's move on to the next lesson.
- Okay, well then I will see you next time, Michael. Sure Dan, thanks, thanks for listening
- Oh well, you know, I, I enjoy doing it so much.
- [LAUGH]
- Bye.

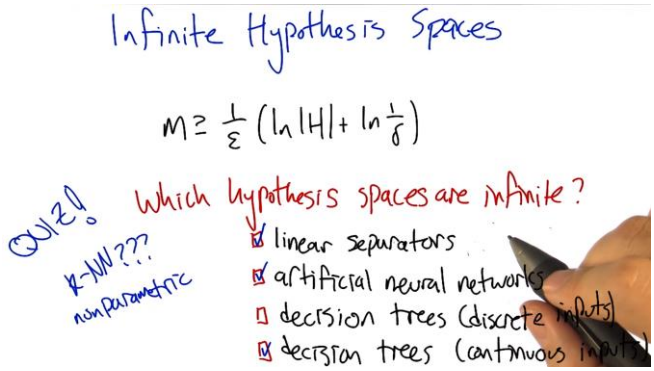
8. VC Dimensions

8.1. Infinite Hypothesis Spaces



- Howdy, Charles.
- Howdy, Howdy.
- Sure, why not. How's it going?
- How do? It's, doing quite well. Thank you for asking.
- Sure.
- How's it going on your end of the world?
- I'm feeling okay, let's say. Though I'm a little concerned because last time we, we were talking and you said you had a question. And I promised that I would get into it. And it's complicated, but it's really interesting. So, let's remind ourselves what the question was. So, we're talking about bounding the number of samples that we need to learn. A classifier or a concept in some given hypothesis base, h. and we ended up driving a formula that looks like this. So, the formula tells us that we're okay, as long as the number of samples is at least as large as 1 over the air parameter, epsilon ; times the quantity log of the number of hypotheses. Plus the log of 1 over the failure parameter. And, and so if the We want to make sure that we succeed with very low failure probabilities. Delta's very small and that means we need more samples and if we want to make sure that this error is really small, that also makes this quantity big, which means we need more samples. Right, so do you remember this?
- I do remember this.
- Alright and what was your concern?
- Well my concern was that the number of depended on the size of the hypothesis space. And I was wondering what happen if you have a really, really large hypothesis space. Like for example, one of infinite size or infinite cardinality I suppose is the right term.

8.2. Quiz: Which Hypothesis Spaces Are Infinite



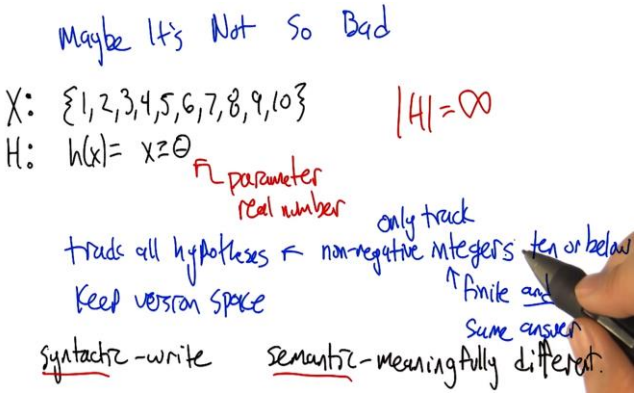
- Okay, here's a quiz! So, just to get at this issue of why it's so important that we consider infinite hypothesis spaces, let's look at some hypothesis spaces that have come up in prior lectures. So,

for each one on this list check it off if it's infinite, and otherwise don't check it off. Does that make sense?

- Makes sense to me.
- Alright, let's do it.
- Go. Answer
- Alright, what do you got?
- Who me?
- Yeah. So, for each one, tell me whether or not it's infinite.
- Okay. Linear separators they seeming like half planes, or lines, or, you know, depending upon dimensionality. There are, of course, an infinite number of these things. There are an infinite number of lines.
- So, that, we'll check that one off.
- So, y equals mx plus b. And, I can put any real number for m. Any real number for b. Infinite number of those, in fact there's not just an infinite number of those there is of those infinite numbers. We should talk about that one day. Artificial neutral networks are exactly the same thing, they have weights those weights are real numbers, so even if there were only one weight there is an infinite number Of real numbers to chose from, so that's also the limit. Decision trees, with discrete inputs. I have two answers for you here Michael.
- Oh!
- Answer one is, of course, it's finite, a, a, assuming there is only a finite number of features. The other answer is it could be infinite if I'm allowed to re-use features over and over again even if they're useless to reuse. But that is sort of insane and silly, and no one will ever do that, so I think the, that right answer is to leave it unchecked.
- Okay.
- And then finally decision tree with continuous inputs. Well, that's the same. We had a long conversation about this when we talked about decision trees. We can keep asking questions about them so if there's a sort of an infinite number of questions you can ask. I can say, well, is this feature greater than .1. And then ask is it greater than .11. Then is it greater than .111, then .1111, then 1111111 and so on and so forth. So, that is also infinite.
- So basically everything we've talked about, or nearly everything we've talked about actually doesn't fit the analysis that we talked about last time.
- What about k and n?
- Yeah so k and n is a little bit of a mess. I think you and I maybe don't completely agree on this one. So I think of k and n, the classifier that comes out of a k and n is defined by the set of data points that are the, the neighbors.
- Mm-hm.
- And. There's an infinite number of ways of laying out those points. So there's an infinite number of different tan N base classifiers that you could have. But you have a counter argument to that.
- Right, which is that if you assume the training set is fixed. And that's just part of the parameters of the hypothesis base then. It is an infinite. There's, in fact, only one. It all just depends upon Q. And it always gives you the same answer, no matter what. So I think the hypothesis space, you could argue, is finite. It all depends upon what it is you're taking as part of the hypothesis. And what it is you aren't taking as part of the hypothesis space.

- Right. Sort of, whether the data is built in or not. It strikes me that these other methods are also similar in that, if you bake in the data, there's just the one answer. But yeah it's a, k and n is weird. Right? Because it's sometimes called non-parametric.
- Right.
- Which sounds like it should mean that it has no parameters but what it actually means that it has an infinite number of parameters.
- Right. By the way, I don't think that's true about baking things in. So, for example, if I give you a set of data. There's still an infinite number of neural-networks that are consistent with that data. There's a whole bunch of decision trees that are consistent with that data. So you don't always get the same answer every time. Certainly with neural-networks you don't because you're starting at a random place.
- But it, but if you, right. If you're bake in the algorithm and the data and I think in k and n that is exactly what you're doing. But I agree that we can agree to not agree.
- Agreed.

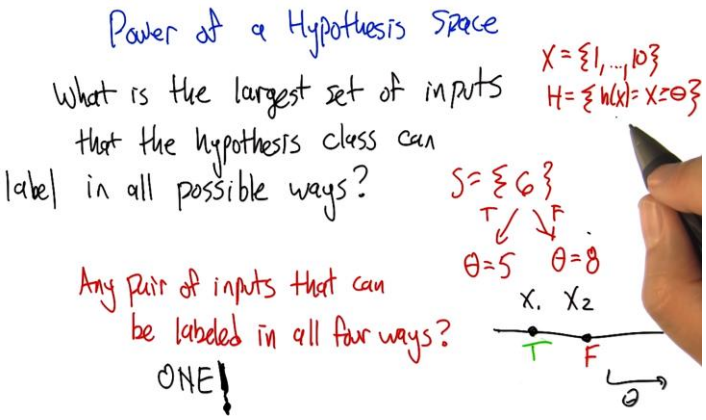
8.3. Maybe It Is Not So Bad



- So here's an example to explain why maybe the situation's not so bad after all. So let's look at a particular example. We've got our input space consisting of say, the first 10 integers. And our hypothesis space is, you take an input, and then you just return whether it's greater than or equal to some theta. So that's a parameter. And now, how big is the hypothesis space?
- What type is data?
- Let's say theta's a real number.
- Oh, so it's infinite. Infinite!
- Indeed it is. Now, on the other hand, what would you do to try to learn this? Can you use the algorithm that we talked about before to learn in this particular space? So, I guess what I'm asking is, is there a way you can sort of sneakily apply the ideas from before, now the ideas from before were that you actually keep track of all the hypotheses. And to keep the version space, and once you've seen enough examples that are randomly drawn, you would be able to know that you've epsilon-exhausted the version space, and then, ultimately, any hypothesis that's left is going to be okay. So, what could we possibly do to track all of these hypotheses? It's problematic, because there's an infinite number of them.
- Okay. I see where you're going with this. So when I asked you what type it was, you said it was a real number, but it would have been easier if it, theta weren't a real number, but were in fact, you know, a positive integer say, or a non-negative integer.
- That's true, though there's still an infinite number of those.

- True, but it doesn't matter because the size of X is, it's so finite. So any value of theta greater than ten for example It doesn't matter. It doesn't matter because it will always give you the same answer.
- Alright. So if we, what if we only track the non-negative integers 10 or below. This would be, what, it's finite. And it gives us the same answer, as if we had actually tracked the, the infinite hypothesis space. So there's kind of, well, I dunno, you had a, you had a good way of saying it before, do you want to say it again? What is the difference between kind of this hypothesis space that we're working with, and the hypothesis space as we defined it?
- So there's a notion of syntactic hypothesis space which is all the things you could possibly write, and then there's the semantic hypothesis space which are the actual different functions that you are practically represented.
- Yeah, I like that, that, that you can make a distinction between semantically, say, finite hypothesis base and actually, it specified syntactically infinitely. And you also have the example of of a decision tree. With discrete inputs as also being kind of like this. That we, you know, we, generally think about only ones that split on a attribute once, but syntactically you could keep splitting on it. It just doesn't give you a semantically different tree. So, this is kind of at the heart of what we're going to be able to do to talk about how we can learn and if in an hypothesis space, more complicated ones than this example here. But at the same time, without having to track an infinite number of hypothesis, because there's just not that many that are meaningfully different.
- I like that.

8.4. Power of a Hypothesis Space

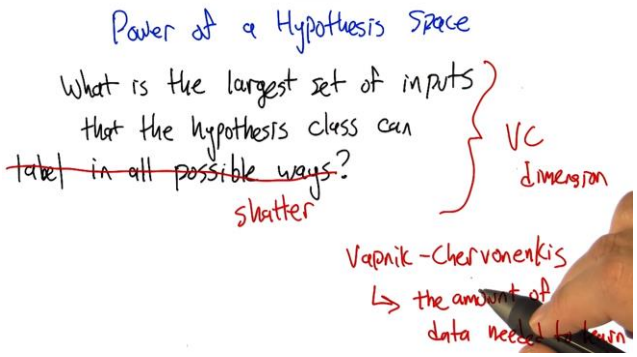


- So, this is how we're going to be able to measure the power of a hypothesis space. This is a really clever definition. I did not come up with this and it goes like this. For a given input and hypothesis space, we're going to ask what is the largest set of inputs that the hypothesis class can label in all possible ways? So, in this example that we were looking at, it's actually really simple because it turns out the answer is one. so, here, here's an example. So being able to do this with one is not such a big deal. If S is a set of points, a set of inputs, in this case just six, it's one of the inputs from the set. Then are there hypotheses in the hypothesis class that can label this in all possible ways? Well, there's only two possible ways. It can label it as true and it can label it as false. So, here if we set theta to I don't know, five, it'll label it as true. If we have a different hypothesis that say sets theta to eight, then we can label it as false. There is a set of

inputs of size one that we can label in all possible ways. But is there any pair of inputs that we can label in all four ways?

- I'm going to say no.
- And why is that?
- Well because you're writing it, you're writing it in sets but I sort of think of these as points on a number line, and theta as a separating line. And there's just kind of no way to label anything to the left of the line as negative, ever. Because you're requiring that x is greater than equal to theta to be positive, so you can never label anything to the left of that line as negative. So all I have to do, right, is make x1 negative and x2 positive, and there's nothing you can do. Is that right?
- Indeed it is. So, in particular, pick any two points x1, x2 on the line just like you said, if as we roll theta, if we just kind of consider sets of theta as moving from left to right, it starts off where x1 and x2 are both going to be labeled as true. Then as we move theta to the right, x1 is going to eventually start to be labeled false, so that okay, that's now two of the combinations we've seen. We're going to keep moving theta to the right, and now x2 is labeled as false. So we've seen three of the combinations, but which combination didn't we see?
- true, false.
- And there's just no way to make that happen. Just like you said.
- So would you say this is a weak hypothesis space?
- It definitely seems to be pretty weak, even though it's infinite. In fact, did it depend on x being finite?
- No, actually, it didn't. You're right.
- Yeah, so all, so this really applies in the, in this very general setting. We can take this definition, bring it to bear on an input hypothesis pair like this, and it gives us a sense of how expressive or how powerful the hypothesis space is. And in this case, not very expressive.

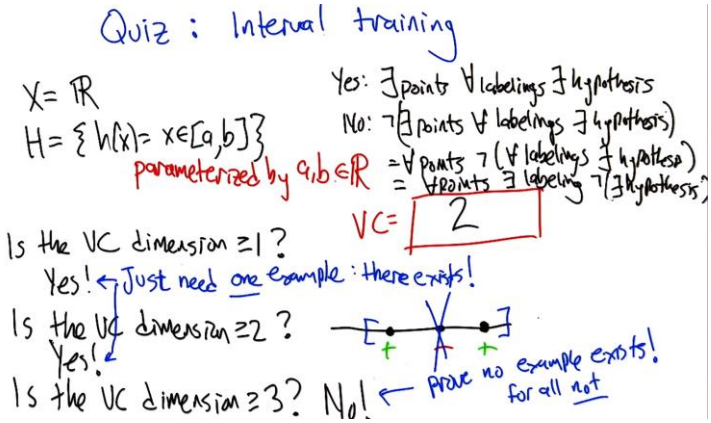
8.5. What Does VC Stand For



- So this is a concept that we're going to be able to apply in lots of different settings when we have infinite hypothesis classes. And this is really the fundamental way that it's used except usually, there's kind of a more of a technical sounding definition. This notion of labeling in all possible ways is usually termed shattering. So this quantity that we're talking about here, this, this size of the largest set of inputs that the hypothesis space can shatter, is called the VC dimension.
- What does VC stand for?

- VC stands for Vapnik - Chervonenkis which is a pair of actual people. So that, you know, really smart insightful guys that put together this notion of a definition and what they did is they can relate the VC dimension of a class to the amount of data that you need to be able to learn effectively in that class. S, as long as this dimensionality is finite. Even if the hypothesis class is infinite. We are going to be able to say things about how much data we need to learn. So, that's, that's really cool. It really connects things up beautifully. So, I think what would be a really useful exercise now is to look at various kinds of hypothesis classes. And for us to measure the VC dimension.
- Okay, sounds like fun.

8.6. Quiz: Internal Training



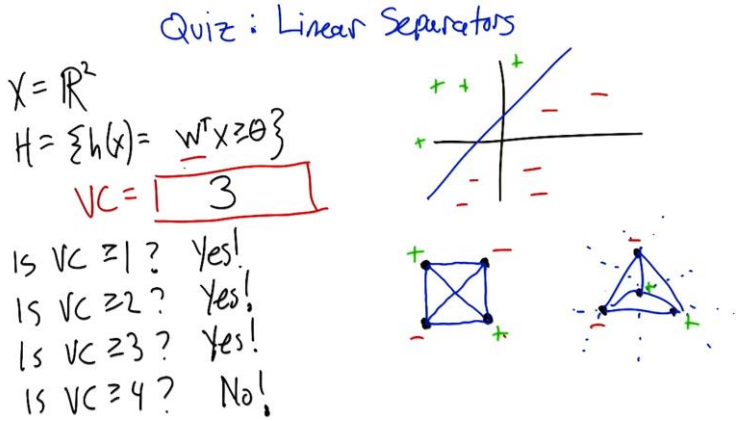
- So let's look at a concrete example, where the hypothesis space is the set of intervals. So the inputs that we are trying to learn about are just single numbers on the real line. And the hypothesis space is this set of functions that return true for all the things that are between a and b, and this is parameterized by a and b. So how many different hypotheses are there in our class here?
- At least 2.
- Sure. How about how many are there in the class?
- There's an infinite number of them.
- That's right. So, so this is one of these situations where it's going to be really helpful to apply the notion of VC dimension if we think we'd like to be able to learn from a finite set of data. Which, you know, generally we like that. So how do we figure out what the VC dimension is? We want to know, what is the largest set of inputs that we can label in all possible ways, using hypotheses from H. Alright, so, I want you to figure that out. Figure out the size of the largest set that we can shatter, that we can label in all possible ways using these hypotheses. And then just, you know, write it as an integer in this box.
- Cool. Answer
- OK, so how do we figure this out?
- Cleverly, so I, when I, when I see things like this, I just like to be methodical, so why don't we just be methodical so, I'm going to ask the question whether the vc dimension is at least one, because it's pretty easy to think about and maybe I'll get a feel for how to get the right answer that way. OK, so is the vs dimension at least one? Well, the answer is pretty clearly yes, so if you just put a dot on the number line somewhere. You could label it positive just by picking any a less than or equal to that point and any b greater than or equal to that point. So, if, if I were like

drawing parentheses or something to indicate the interval, I could just put parenthesis around the point and that will give me a plus or brackets, that would be fine. Okay, so that's that's pretty easy. And if I wanted it to be negative, I could just put both of the brackets on either side of the point, it doesn't matter, let's say to the left. Alright, that make sense Michael?

- That's exactly what I was thinking about, yeah. Though I would've put the brackets on the right.
- Yeah, you would. okay, so then we could see...do the same argument for, see if the VC dimension is greater than or equal to two. So if I put two points on the line, so there are only,there're four possibilities I gotta get. Plus plus, minus plus, plus minus, and minus minus. Okay, so we gotta get plus plus, plus minus, minus plus and minus minus. So, the, the first and the last one are really easy. Actually they're all easy but you can definitely do this. So, if you want to get plus plus, you just need to put brackets so that they surround the two points, that's good. If you want to get plus minus you put the left bracket in the same place and you put the right bracket just to the right of the point, yeah, and you do the same thing for minus plus and then for minus minus you put the brackets on either side of both of the points and so, since you like it to the right I'm going to put em to the left.
- [LAUGH] Good.
- And there you go, that was, that was pretty easy I think. Okay so next we need to figure out whether the VC dimensions at least three. So we need three dots on a line, three, distinct dots on a line. And we've got eight possibilities but Michael I don't want you to write down those, those eight possibilities because I think I see an easy way to answer the question right away.
- Excellent
- So, this is a lot like the last example we did with, with the theta. Except now.
- Yeah.
- We only have two parameters. And the problem with had with the theta was that as we moved the theta over, from left to right, we lost the ability to, to, to have a, a, a positive followed by a, a negative. So I think there's a similar thing here. So, if you label those three points this way. Plus, minus, and plus. I don't, I don't think you can do that, and that's because in order to get point one and point three in the interval, you're going to have to put the brackets on both sides of them. So you're going to have to put a, a left bracket to the left of the first point and a right bracket to the right of the third point. And that's the only way to make those two plus. But then you're always going to capture the one in the middle. So you can't actually shatter three points, with this hypothesis class.
- Now, you have to argue though, that there isn't some other way you could arrange the three points. I don't know like, I don't know, stacking them on top of each other or something.
- You mean vertically on top of one each other?
- Yeah.
- Well then they wouldn't be in R, they'd be in R2.
- Well no, just like right on top of each other.
- Well then they're all the same point.
- And you can't label them. Again, you have the same problem that you can't label one of them negative and the other ones positive if they're all on top of each other.
- Right.
- So, so there isn't, there just isn't any way to set up these three points so that you're able to assign them all possible labels.
- Right.

- So, good, so that gives us two as our answer here. So, by the way, I think that you said something I think that's really important. In order to prove the lower bound, in order to prove one and two, all we had to do was come up with an example where we could shatter, right?
- Yes, that's exactly right.
- Right, so so that's good and that's that's really nice because otherwise we're in a heap of trouble [LAUGH] if we have to show that you can shatter every single thing. We just have to show that you can shatter one thing. So, it exists. So that whole VC dimension is really a...there exists some set of points you can shatter, not you can shatter everything.
- That's right, and what would be an example of points that you couldn't shatter yeah, a pair of points that you couldn't shatter?
- Well, the ones on top of one another.
- Yeah, exactly, because you wouldn't be able to assign them different labels.
- Right.
- So that would be a really bad choice, and here all we need is a good choice.
- Right. So, if you make good choices you can shatter things, which sounds more violent than I intended. Okay but, in the third case of the VC dimension, it wasn't enough to show an example that you couldn't shatter, because, then you could do the same thing as you point out, with a VC dimension of two. Instead you have to prove that no example exists. So, there does not exist or a for all not word or something.
- For all, not.
- [LAUGH] Exactly. So, that, that's a, that's an interesting set of set of requirements there, right? So, proving a lower bound seems easier than proving an upper bound.
- Though it's interesting because in this case, in cases one and two, you had to show that all the different combinations were covered, whereas in this last case we just had to give one combination that couldn't possibly be covered.
- Yeah, but it couldn't possibly be covered no matter what we did. No matter what the input arrangement was.
- Right.
- Yeah.
- Whereas in the first case, I had to show all possibilities. I mean, you know, all possible labelings but only for one example of orderings or one collection of points. So just messily doing some bad predicate calculus to, nail down what you're saying. That when we say that the answer is yes, we're saying that there exists a set of points of a certain size, since that for all labelings, no matter how we, we want to label it. There is some hypothesis that works for that labeling. But to say no, we have to do the legation of that which is not exist for all exist. Which, by standard logic rules says that, that means for all points, no matter how you arrange the points, it's not the case that for all labels. There exists hypothesis which again DeMorgan's Law its not against DeMorgan's Law to to apply this idea that says that's the same as for all arrangements of points there's some labeling where there's no hypothesis that's going to work and that's exactly how you made your argument.
- Huh, except I didn't use DeMorgan's Law and upside down a's and backwards z's. Oh you did, oh you did.

8.7. Quiz: Linear Separators



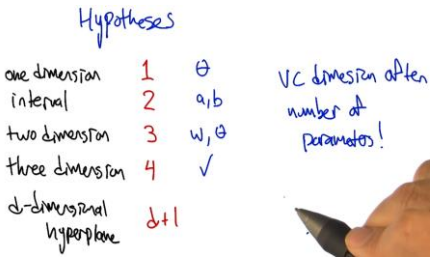
- Alright, let's do another quiz. That previous example that we looked at of intervals, was nice and pedagogical, and reasonable to think about, but we actually hadn't really talked about any learning algorithms that used intervals. On the other hand, linear separators are a very big deal in machine learning. So, it's, it's very worthwhile, and it turns out to be not too bad to work out what the vc dimension is for linear separators. So, let's say that we're in two dimensional space, and so our hypotheses have the form that you've got a parameter, a weight parameter, w. And were going to just take that weight parameter, take the dot product with whatever the input is, and see whether its greater than or equal to some value, theta. And if it is, then we say that's a positive example, and if not it's a negative example, and geometrically that just means that we've, we end up specifying a line, and everything on one side of the line is going to be positive, and everything on the other side of the line's going to be negative.
- Got it. That makes sense. So what's the vc dimension? Oh, they're going to have to tell us. I like that.
- Alright. Answer
- Alright so we're back in again, and we're going to attack it I the way that we, that you attacked the previous ones, where we're going to ask, kind of systematically is the VC dimension, greater than or equal to 1, 2, 3, 4 by, by giving examples until we just can't anymore [LAUGH]. So good, so is the VC greater than or equal to one?
- Yes.
- Yes. So, what would that mean? All we need to do is provide a point, I don't know, call it the origin. And.
- Basically, we get to just pretend that it's like a single point on a line with a VC dimension of one and it, the same argument that we had before, applies.
- That's a good way to say it. Just you know, just think about the x asis, axis itself, and we can label something, well actually it's simpler in a sense because, we can keep the line steady and we can just flip which side is, you know, by negating all the weights we can flip which side is positive and which side is negative, and that gives us the 2 labelings of that point.
- Right, and because similar argument for VC of 2.
- So, if the 2 points were on a line, then to do the 4 different combinations, we could.
- So right, by putting that line to the left, we can label both of them positive. That's easy, or we could label both of them negative by flipping the weights. Now we've to do the other 2 cases where they've different labels. So, I'm going to recommend putting a blue line between them.
- It's a thin blue line.

- [LAUGH] Yes, and you know, the one on the right is positive the one on the left is negative, or we can flip the weights and then flip the signs.
- Yes, and 3 is where we got into trouble last time, so let's let me start off by giving ourselves a clean slate. So, this ran us into trouble in the case of the intervals because we couldn't do that case an it looks like we're kind of hosed again, right?
- Yeah, we're. We're actually completely hosed again, if we do this.
- [LAUGH]
- So, I'm going to say that the problem is not with the hypothesis space. The problem is with the hand that is drawing points on the screen. So that's you, so here's the.
- My hand is really depressed.
- Well, I'm going to make your hand happier. So, I think it's right that you can't separate this. It's, and, and the reason you can't separate it's because we've sort of nothing to do here, just like we'd before. But, we are not restricted to the number line. So I'm going to recommend cheating, and moving that point in the middle off the number line. So make a triangle, stick it up in the middle somewhere.
- Alright, and that gives us the ability to handle this case now, because we can just send our slicey line this way. Put everything below it as positive and everything above it as negative.
- Right, now of course we still, by doing that we might have messed up the other labeling, so we should check to make certain that we haven't we haven't screwed anything up. So, we can, we can make the top minus and the bottom, plus that's true and we can just by flipping the weights we can make it the top plus and the bottom minus right? So that's good. And the question is that can we do anything else.
- Yeah, I think it's pretty clear. We could definitely label them all positive or all negative just by putting a vertical line somewhere off to the left.
- Yeah, and I think it's actually easier than this because if you just think about vertical lines, then we really are back in the one dimensional case.
- Right. And, and we handled the other 7 cases in the one dimmensional case really easily. It was just this, this extra case that we didn't know how to do and now we do, we just use that 3rd dimmension. [LAUGH] Or the 2nd dimmension, even better.
- Fair enough. Okay, so the answer's yes. I feel good about that. Okay, so, that's good. So we got, we got 1, 2, and 3 out of the way, so we know it's more powerful. We know that it's better. This's, this's kind of nice. So now the question is 4. So, thinking about it, I think that the answer is no.
- [LAUGH] That would be nice, wouldn't it? But, no, we need a, we need a slightly better argument and I think, I think we can do that, what we need. Again, what would be helpful is if we had an example, where we could say, okay, here's a labeling that no matter how you lay out the points, you're going to fail.
- So, in order for that to work, we need to try to use all the power of the 2 dimensions so we don't fall in a trap. Right if, like we almost did with VC3 by making them collinear. So, why don't you place 4 points in the plane and make a kind of like a diamond shape, or a square, which is like a diamond.
- It's a diamond shape if you yeah, tilt your head a little bit.
- Okay, so I'm going to tilt my head to look at it. So, here's my argument. Now, I don't know if this is quite right Michael, so, so help me out with it. The reason I don't think you can do 4 is because we've only got lines to work with. Okay so, if you connect all the points [CROSSTALK]

- Hm-mm.
- All, all pairs of points the way they, all ways they can be connected. So, you know, draw the square on the outside and then draw the 2
- Hm-mm.
- Cross ones in the middle. Does that make sense?
- Yeah, it makes sense, but I'm not sure where we're going with this.
- Okay, so I'm not either so [LAUGH] so, so, so work with me. So that's kind of all the boundaries that you can imagine drawing. And the problem that I see here is that because of the way that the, the 2 lines that the x and the interior of the square's set up. There's kind of no way to label the ones on the other side of those lines differently without crossing them. So that made no sense what I said, right? So, try putting the, a plus in the upper left and bottom right. And minus for the other 2. So, if you look at the, the 2 1's that are connected by the line with the plus, and the 2 right? There's no way to put a single line that will allow you to separate out the pluses from the minuses here.
- Yeah, yeah. Exactly. So, in particular anything that puts, these 2 pluses on the same side is either going to put one minus or the other minus on that same side.
- Right.
- It has a very XOR kind of feeling to it, to me.
- Yeah does, it, it,it does and in fact it has an x right there in the middle.
- [LAUGH] It does, no but it, that is true, but I meant it in a slightly different way, which is if you think about these 4 points as actually being you know, zero, zero. 1-1, 0-1 and 1-0.
- Mm-hm.
- Then, the labeling here is exactly XOR. And XOR is one of these things that you can't capture with a linear separator. So I think, I think you got it.
- Oh, it makes sense. And I think the important thing here, is that oh I like the XOR argument. The important thing here is that, no matter where I move those four points, I can take the one closer or one further away. And I could, they're no longer squares, but whatever I want to be, they're always, you're always going to have a structure where you can draw those kind of crosses between the 4 points. Or, you're going to end up collapsing the points on top of one another or making 3 of them co-linear or all four of them co-linear and so that makes it even harder to do any kind of separation. Cause now we're back.
- Right. You fail on all the, but there, there's one case that I'm not sure that you quite described yet. Like that.
- Right. Well, I think that, that works out to be the same thing, right? If you draw the connecting lines together they're all going to cross at the middle point.
- There's no crossings.
- They all cross at that point. They all meet at that point.
- They don't cross at that point.
- Well, so those are line segments, but those are just line segments they represent lines that go on forever. Good point.
- Yeah so, but the way, the way that I would see this one is, again to just give an example of a labeling that just can't be separated would be this one. Like if you capture the outside points, assigning all the outside points one label, you can't assign the input, the inside points a different label. It's inside the convex hall, it's going to have the same label as the other ones.

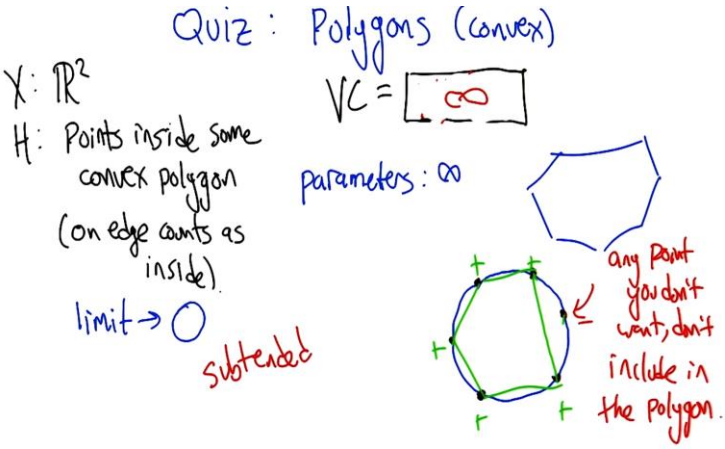
- Exactly. So, the, and I, and well so in my head the, the main issue is as lines, when lines cross there's really nothing you can do with a single line. Never cross the streams.
- [LAUGH] Yeah. It still doesn't feel like quite the same. I mean maybe we're belaboring this point. Here in this square, if you actually let this, this corner point pushed into the middle, then we can, I think we can linearly separate them.
- Sure.
- So, I feel like these are 2 different cases, but regardless the point is, that what we, what we argue is no matter how you lay out the points, there's always going to be a labeling that can't be achieved in the hypothesis class.
- Yeah, the whole crossing of the lines thing, really is about being able to get all 4 points. It's not saying that any pair of points. Works out okay. So, what you'd end up doing is taking one of those points and dragging them into the middle, and then the lines all meet like in, in what you've drawn. And you end up with the basically the, the same argument. I think it's the same thing. But, I do agree with one thing, Michael. Which is that we are belaboring this point.
- Because the good news or the, the exciting news is no, we really argued that the VC dimension of linear separators is not greater than or equal to four. So therefore, it's 3. Because 3 works and 4 doesn't.
- And 3 is my favorite number. So, I have a question for you Michael, I noticed that we keep getting in all the examples we have done so far, we keep getting one more VC dimension, so does this kind of argument work if I went from planes to, or lets see, 2D space or 3D space or dimension still three or does it keep getting bigger?

8.8. The Ring



- Alright, so let me try to, to write that down in a, in a way that let's us summarize it. So I think what you're trying to say is when we did that one dimensional case, it had, the b c dimension was one. When we did the interval, it was two. When we did two dimensional, linear separator, it was three. And you're wondering whether in, three dimensions, it would be four.
- Yes.
- So that's, yeah, a really good insight. Let me, be a little bit more precise here. That the hypothesis spaces in each of these cases here, they, they were defined this parameter theta. In the interval case it was defined by a and b. In the two dimensional case it was defined by w and theta, and w was in two dimensions so this was actually, a vector of size two. So, yeah, each time we went up, it, to do a different example, we actually added another parameter. And, it looks like the bc dimension is the number of parameters.
- Hm.
- So in a sense it's it's the dimensionality of the space in which the hypotheses live. So it really, it really fits very nicely. That doesn't exactly answer your question. It is the case that for a three dimensional problem there's going to be four dimensions. And so it turns out you are right. That for any d dimensional hyperplane concept class or hypothesis class, the vc dimension is going to end up being d plus 1.
- Oh, I see, and that's because the number of parameters that you need to represent a d dimensional hyperplane is in fact, d plus 1.
- That's right. Yeah, d, the weights for each of the dimensions plus the theta, you know, the greater than or equal to thing.

8.9. Quiz: Polygons



- So, Michael, I know we said that was the last quiz but I think we should do one more quiz. So the quiz is going to be on convex polygons. And X is going to be an R squared. And the hypothesis is going to be points inside some convex polygon. And, and inside means the same thing as we

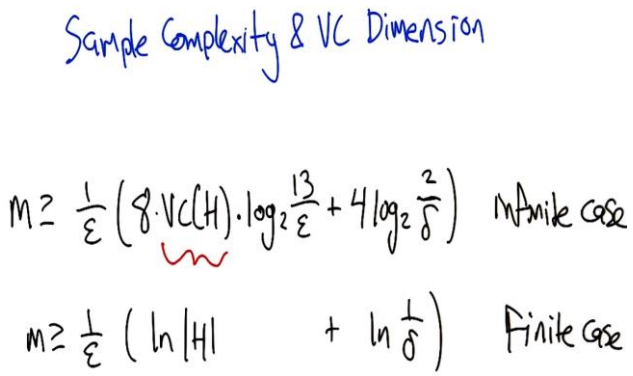
meant with circles. So, if you're on the polygon or on the perimeter of the polygon, then you're inside the polygon for the purpose of this discussion. So, here's my question to you Michael. What is the VC dimension of convex polygons?

- Well, if I had to.
- Ask someone else, you would say it was a quiz and you'd let them do it.
- Is this a quiz? Oh, it's a quiz for me.
- Well, I dunno, do you want to let the students get a, get a try?
- Well, yeah, and then we can answer it by simply going to the quiz if we actually go to the quiz
- [LAUGH] Okay, so let's go to a quiz. Go. Answer
- If I had to guess, which you are kind of making me do, I would say, well, for one thing, the number of parameters is infinite. Right? Because if it's some convex polygon, and we're not putting any bound on the number of sides on that polygon, then to specify it, you have to give what the points are for each of the vertices and the, you know, as the number of sides grows, the number of parameters grows. So it's, it's unbounded. So it could be that the vc dimensions is going to end up being unbounded but they do seem you know at the limit they turn into circles and circles ended up being a vc dimensions of three so maybe, you know, maybe it's three.
- Maybe. So, so actually you, you, you've really sort of stumbled on the right answer there, or maybe not so stumbled, on, on to the answer there. So, in the limit, convex polygons become circles. Right? So draw a circle for me, okay, now, lets sort of try to do this smartly, so put a point on the edge of the circle, yeah I like how you placed that, so pretty clearly you could come up with a convex polygon that puts that either in or outside of it right? Because you know, there is only one point, thats pretty easy.
- Yeah, and the circle is kind of irrelevant.
- Yeah the circle is kind of irrelevant, but its going to be part of my little trick. So put another point on the circle somewhere. And in the same way we've been doing it before with lines, you know, you can put both of the inside a convex polygon or outside, you know, you can do all the labels. I think that's pretty easy to see. Now try three. So, the first thing I want you to notice Michael, is that if I look at those three points and I connected them together, what do I get?
- Oh a triangle!
- I get a triangle which is by the way, it starts with a C.
- [LAUGH] A sheep that has the number of vertices equal to your favorite number.
- That's right. But it's also a kind of geometric shape, it starts with an A.
- It starts with a
- It starts with AC?
- Appaplectic.
- No it starts with a C. AC, Accenuated. No it starts with the letter C.
- Oh. Convex.
- Yes. It's actually convex polygon. Try putting a fourth point on there. And in fact put the fifth point. And a sixth point. Now, here's my question. We've put all of these points on this circle, right? Now let's just say it's a unicircle because it's easy to think about it. So we put all these points on the circle. Do you think we could shatter this with a convex polygon?
- To shatter it? Right, to give it all possible labellings. Well, let me draw the polygon. So each one being in or out.
- Well, the thing is, the way you've drawn this polygon, all of them are in. So, if you used this polygon, what would you be labeling those six points?

- All positive.
- All positive. What if I didn't want you to label one of the points positive? Pick one of the points. Any point will do. So if I don't want that to be in the polygon, what do I have to do?
- Just push the, the corresponding vertex a little bit inside.
- Right. And the easiest way to do that would be not to have a vertex there at all but simply not to connect that point.
- Oh. It's kind of like a, a rubber band art or string art if we just kind of pop that one out. [NOISE]
- Right. So, any point you'd, of those six points you don't want to be labeled positive. Just don't connect in as a part of your polygon.
- I see. So, for any given pattern or subset, which is what we need to be able to show, that, you know, when we're shattering, we need to show that no matter what the subset is, there's going to be some. Hypothesis that labels it appropriately. You're saying, well just, you know, label the points as plus and minus, and connect up the pluses. It's going to leave the minuses outside because they're going to be on the edge of the circle. And the pluses are all going to be in the polygon because they touch the edges of it.
- Yeah, because they are in fact the vertices. And in this case you just think of the fact that if there's only two positive points a line is a very, very simple convex polygon and if there's only one point, then a point is a very simple convex polygon.
- So the VC dimension is six!
- No! So what happens if I had seven points? Could I do it?
- So the VC dimension is seven!
- What if I had eight points? Could I do it? It's the same trick. We can make it eight.
- So, can we make it nine?
- No.
- Yes.
- Yes.
- So, at what point can we stop?
- When we run out of tape for the recording.
- Exactly. So that means that the number of points that we can capture this way is in fact unbounded. Which means the VC dimension is infinite.
- Nice example.
- Now, I do want to point out that there's a, a teensy tiny little point here that, that we sort of skipped over, but I can explain in five seconds, which is we made polygons. We didn't actually argue that they were convex, but they are convex, because they're all inside the unit circle, and by construction, every, any polygon whose vertices are on the unit circle will be convex. So it's just that's why we needed a circle, that's why we were being clever with it, but there you go. So we have a polygon that we can always draw with those the right thing and because its always subtended by it's circle it will be convex. So we have actually found a vc dimension that's infinite [CROSSTALK]. Or a hypothesis class that has a vc dimension. [CROSSTALK] It has to be infinite, yeah that's what I said. We have actually found the hypothesis class whose vc dimension is infinite and we came up with a proof where y would be that case, and nicely, I think very nicely connects with the observation you made earlier. That, somehow, it connects with the number parameters. I think it's kind of cool. I mean, you, you, end up with a circle, not having a very good VC dimension, a very high VC dimension, but convex polygons, which somehow seem not to be as cool as circles, are in fact, in fact have infinite VC dimension. Okay so there you go so

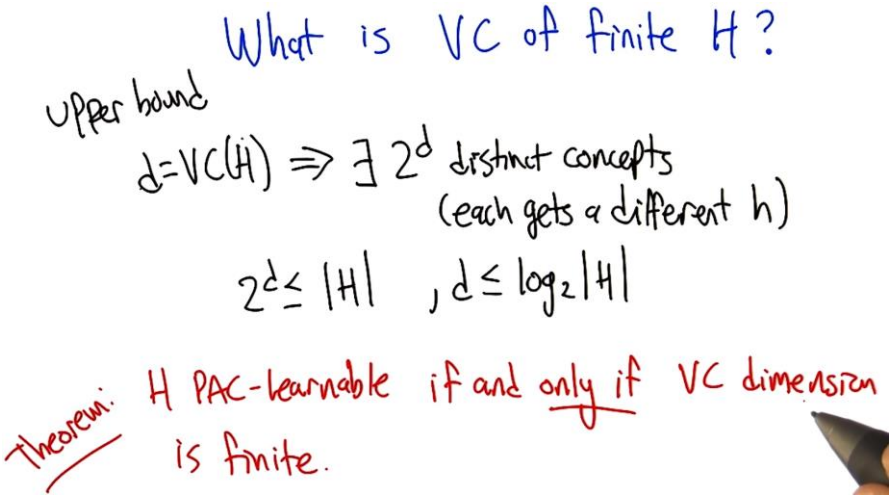
we've done some practice of VC dimensions. So you've given me all this VC dimension stuff, I agree that it's cool Michael, but what does it have to do with, what we started out this conversation with? How does that answer my question about the natural log of an infinite hypothesis space?

8.10. Sample Complexity



- That is exactly the right question to ask. It's fun to spend all day finding the VC dimension in various hypothesis classes. But that is not why we are here. The reason we're, why we're here is to use that insight about VC dimension to connect it up with sample complexity. And so here is the equation that you get. When you connect these things up. It turns out that if you have a sample set the, the size of your trading data, is at least as big as this lovely expression here. Then that will be sufficient to get epsilon error, or less, with probability 1 minus delta. And so, the form of this looks a lot like how things looked in the finite case. But, in fact it's a little bit weirder. So 1 over epsilon times quantity eight times the VC dimension of H. So that's where this quantity is coming into play. So the VC dimension gets bigger, we're going to need more data. Times the log base 2 of 13 over epsilon. Sure. Plus 4 times the log base 2 of 2 over delta. So, again, this log of, of something like 1 over delta to the inverse of delta, was in the other bound, as well, that's capturing how certain we need to be that, that things are going to work. And again, as, as delta gets small, the failure probability gets small. This quantity gets bigger. And the num, and the size of sample needs to be bigger. But, but this is the cool thing. That the VC invention is coming in here in this nice, fairly linear way.
- So it sort of plays the same role as the natural log of the size of the hypothesis space.
- Yes, that's exactly right! And in fact, things, things actually map out pretty similarly in the finite case and the infinite case. That there's an additive term having to do with the failure probability. There's a, you know, one over epsilon in the front of it and then this quantity here, having to do with the hypothesis space, is either the size of the hypothesis space or the dimension of it, depending. Well the size here is logged and the VC dimension is not, so that's a little bit of a difference.
- Mm.
- But but there's a good reason for that as it turns out.
- There is?
- Yes, indeed. So why we, why don't we take a moment and look to see what is the VC dimension of a finite hypothesis class? The VC dimension concept doesn't require that it's continuous. It's just that when it's continuous, the VC dimension is required. So that maybe that's a useful exercise. Let's do that.

8.11. VC of Finite H



- So we can actually work out what the VC dimension of a finite H is and, in fact, it's easier to just think about it in terms of an upper bound. So, let's, let's imagine that the VC dimension of H is some number, D. And the thing to realize from that, is that, that implies that there has to be at least two to the d distinct concepts. Why is that? Is because each of the two to the d different labelings is going to be captured by a distinct hypothesis in the class, because if we can't use the same hypothesis to get two different labelings. So that means that the, that two to the d is going to be less than or equal to the number of hypotheses. It could be that there's more, but there can't be any fewer, otherwise we wouldn't be able to get things shattered. So, just you know, simple manipulation here, gives us that d is less than or equal to the log base 2 of h, so there is this logarithmic relationship, between the size of a finite hypothesis class. And the VC dimension of it, and again, that's what we were seeing in the other direction as well, that the, that the log of the hypo, size of the hypothesis space was kind of playing the role of the VC dimension in, in the bound. Okay, that makes sense. And, and from that, it's easy to see how 13 got in there.
- Yes. It should be pretty much obvious to even the most casual observer of 13.
- Yes, I think that's right. So I don't think there's any reason for us to explain it.
- Yeah, I think one would have to really go back and look at the, at the proof to get the details of why the, it has the form that it has, but, or at least the details of the form. The, the, the, overall structure of the form, I think we understand. It's just that the details come out of the proof and we're not going to go through the proof.
- And I think that's probably best for everyone.
- So what, what we're seeing at the moment is that a finite hypothesis class or a finite VC dimension, give us finite bounds, and therefore make things PAC-learnable. What's kind of amazing though is that there's a general theorem that says, in general, if H is PAC-learnable if and only if the VC dimension is finite. So that means that, we know that anything that has finite VC dimension is learnable from the previous bound. But we're saying that it's actually the other way as well, that if something is learnable it has finite VC dimension. Or to say it another way, if it has infinite VC dimension, you can't learn it. VC dimension captures, in one quantity, the notion of PAC-learnability, which is, which is really beautiful.
- Yeah, I agree. That V and that C guy, they're pretty smart.

9. Bayesian Learning

9.1. Intro

Assertion: Learn the **BEST** Hypothesis Given Data and some domain knowledge

Suggestion to be more precise: Learn the **MOST PROBABLE** Hypothesis given data and domain knowledge

Pr(h | D) - probability of h given D
try to find:

$$\boldsymbol{argmax_{h \in H} Pr(h \mid D)}$$

9.2. Bayes Rule

$$Pr(h \mid D) = \frac{Pr(D \mid h) * Pr(h)}{Pr(D)}$$

- What's nice about Bayes Rule is it gives us a handle to talk about what we're trying to do when we say we're trying to find the most probable Hypothesis given data and domain knowledge

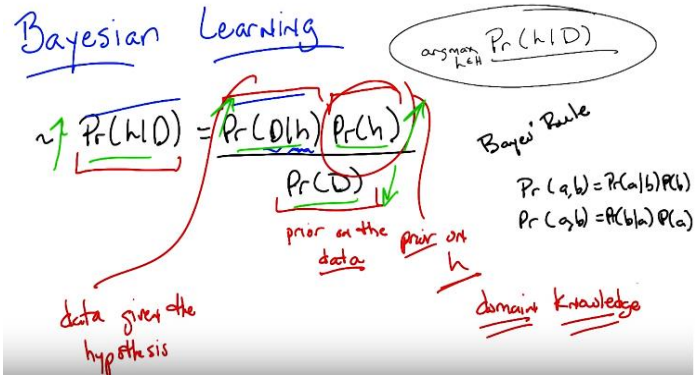
D = {(x_i , d_i)}

D → Our training data
x_i → a single input
d_i → the label on the input

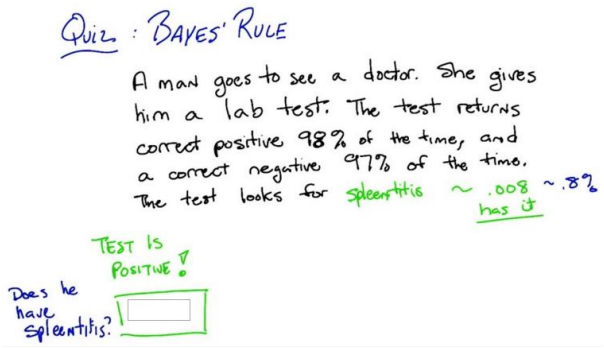
$Pr(D \mid h)$→ data given the hypothesis
$Pr(h)$→ prior on h → Domain Knowledge

- encapsulates our prior belief that one hypothesis is likely or unlikely compared to other hypotheses

$Pr(D)$→ prior on the data



9.3. Bayes Rule Quiz



$$Pr(s \mid t) = Pr(t \mid s) * Pr(s) / Pr(t)$$
$$Pr(\underline{s} \mid t) = Pr(t \mid \underline{s}) * Pr(\underline{s}) / Pr(t)$$

$$Pr(t \mid s) = .98$$
$$Pr(s) = .008$$
$$Pr(t \mid \underline{s}) = 1 - .97 = .03$$
$$Pr(\underline{s}) = 1 - .008 = .992$$

$$Pr(s \mid t) = .98 * .008 = .00784$$
$$Pr(\underline{s} \mid t) = .03 * .992 = .02976$$

Therefore more likely than not **NO**. 21% chance yes, 79% chance no.

9.4. Bayesian Learning

For each $h \in H$ calculate: $Pr(h \mid D) = \frac{Pr(D \mid h) * Pr(h)}{Pr(D)}$

Output: $\boldsymbol{argmax_{h \in H} Pr(h \mid D)}$

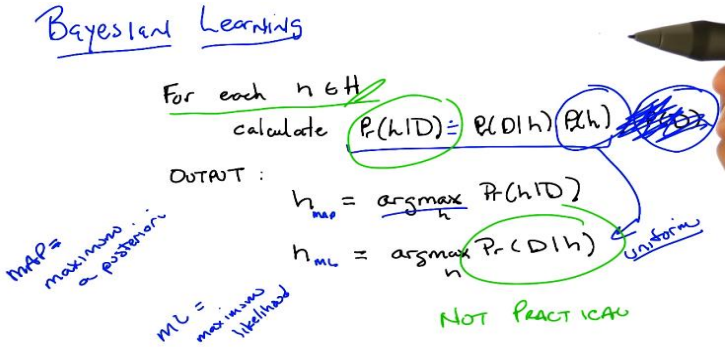
Since all we're ever calculating is argmax though, we never have to calculate the Pr(D). Which is good because many times is it unknown.

MAP (maximum a posteriori) → $Pr(h \mid D) = P(D \mid h) * Pr(h) \rightarrow h_{map} = argmax_h Pr(h \mid D)$
ML (maximum likelihood) → $Pr(h \mid D) = P(D \mid h) \rightarrow h_{ML} = argmax_h Pr(D \mid h)$

- We can assume the all hypotheses, Pr(h), are equally likely.
- Therefore we can drop Pr(h) to get ML above since they are uniform

Therefore we have simplified this considerably, given we don't have a strong prior hypothesis - Pr(h)

However this is NOT PRACTICAL since it requires calculating for ALL $h \in H$



9.5. Bayesian Learning in Action

1) Given $\{< x_i , d_i >\}$as examples of c(noise-free)
2) $c \in H$
3) Uniform prior

Bayes Rule: $Pr(h \mid D) = \frac{Pr(D \mid h) * Pr(h)}{Pr(D)}$

$$Pr(h) = \frac{1}{|H|}$$

$$Pr(D \mid h) = \begin{matrix} 1 \; if \; d_i = h(x_i) \; \forall x_i, d_i \in D \\ 0 \; otherwise \end{matrix}$$

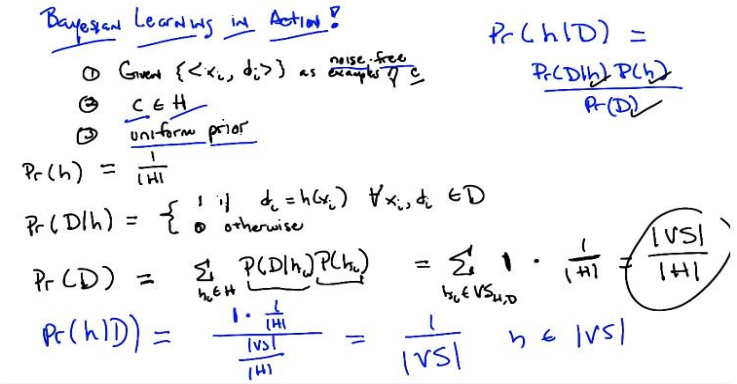
$$Pr(D) = \sum_{h_i \in H} P(D|h_i)P(h_i) = \sum_{h_i \in VS_{H,D}} * \frac{1}{|H|} = \frac{|VS|}{|H|}$$

- VS → Version Space

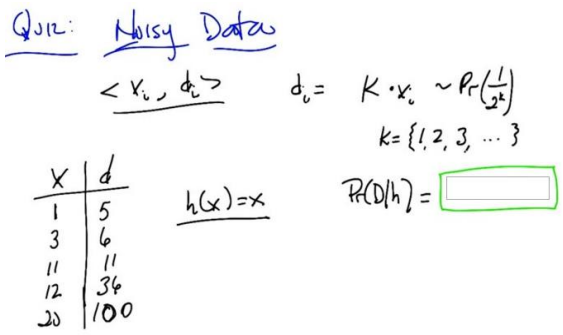
Substitute through:

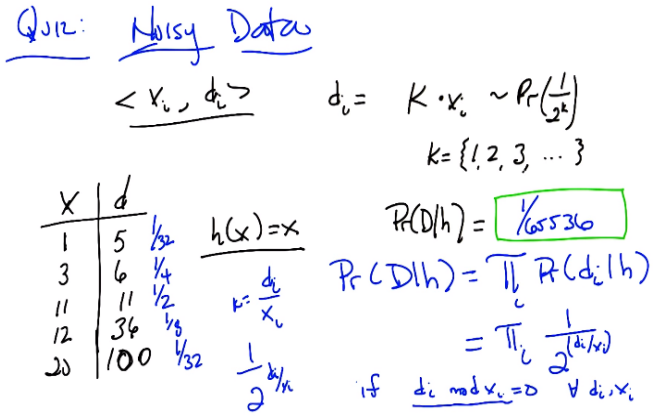
$$Pr(h|D) = \frac{1 * \frac{1}{|H|}}{\frac{|VS|}{|H|}} = \frac{1}{|VS|}$$

NOTE: this is only true for hypotheses that are still in the version space ($h \in VS$), 0 otherwise.
NOTE: This also only works when you know there is no noise in your sample space, that your concept is in your hypothesis space ($c \in H$), and that you have a uniform prior.



9.6. Noisy Data Quiz





9.7. Return to Bayesian Learning

1) Given $\{< x_i , d_i >\}$
2) $d_i = f(x_i) + \varepsilon_i$
3) $\varepsilon_i \sim N(O, \sigma^2)$

$\varepsilon_i \rightarrow$ error

What is h_{ML}?

Recall: $h_{ML} = argmax_h P(h|D) = argmax_h P(D|h)$
Therefore: $h_{ML} = argmax\ \pi_i P(d_i|h)$ → note π_i means product of all i

So, if we have some particular labels, some particular value d_i that is at variance with that. What's the probability of us seeing something that far away from the true underlying F? Well, it's completely determined by, the *noise model*. And the noise is a ***Gaussian***. Note the below Gaussian Function:

Using the $h_{ML} = argmax\ \pi_i P(d_i|h)$formula above and apply Gaussian function above:

$$h_{ML} = argmax\ \pi_i P(d_i|h) = argmax\ \pi_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2}(d_i - h(x_i))^2/\sigma^2}$$

Now attempt to simplify this in a way that only focuses on values we care about, *i*. For example we can remove:

$$\frac{1}{\sqrt{2\pi\sigma^2}}$$

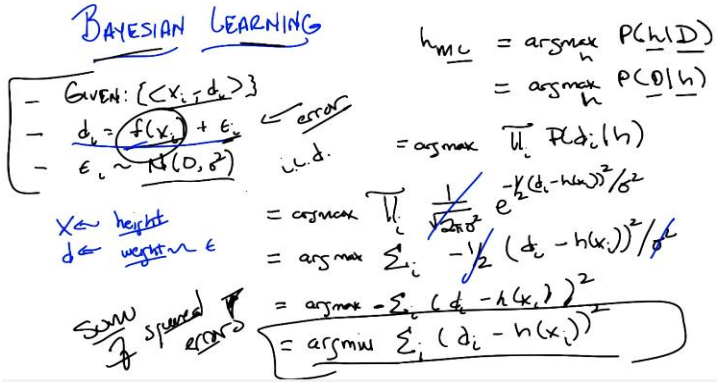
We can also use natural log to isolate only on the exponent for *e*. Take the natural log and note that a natural log of a product is actually the sum of the logs and the natural log of *e* just returns the exponent only. So we are left with:

$$argmax \sum_i \quad -\frac{1}{2}(d_i - h(x_i))^2/\sigma^2$$

Simplify more now because the ½ and the σ^2 have nothing to do with *i*

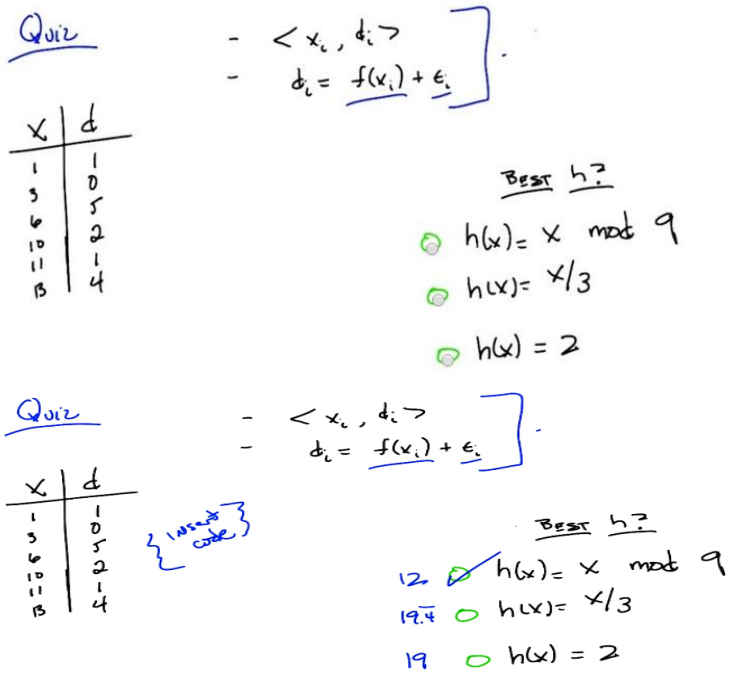
$$argmax -\sum_i \quad (d_i - h(x_i))^2 \text{ OR: } \qquad argmin \sum_i \quad (d_i - h(x_i))^2$$

This is actually the sum of squared errors!



9.8. Best Hypothesis Quiz

Note use squared error like we resolved to above.



9.9. Minimum Description Length

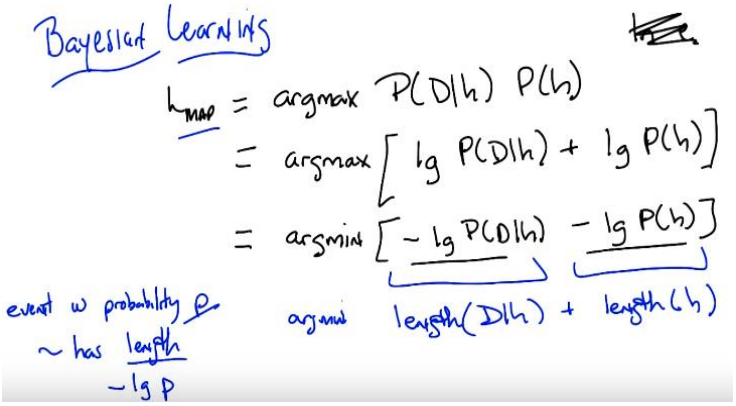
$h_{map} = argmax_h Pr(D|h)P(h)$

$argmax_h\ [lg\ P(D|h) + lgP(h)]$→ the log of a product is the sum of the logs

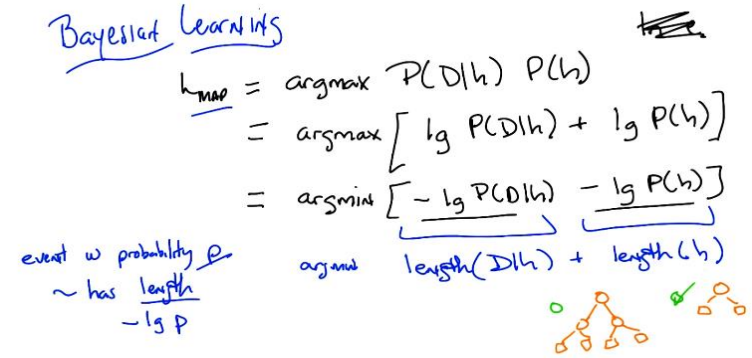
$argmin_h\ [-lg\ P(D|h) - lgP(h)]$→ converted to min

We know from information theory, based the notion of entropy, that the optimal code for some event with probability *P* has length $-lg\ P$. This is saying that, in order to find the maximum a posteriori hypothesis, we want to somehow minimize two terms that can be described as lengths. Because the event is what has the length, it's the length of the data, given the hypothesis, and the length of the hypothesis.

$argmin\ length(D|h) + length(h)$



9.10. Which Tree Quiz



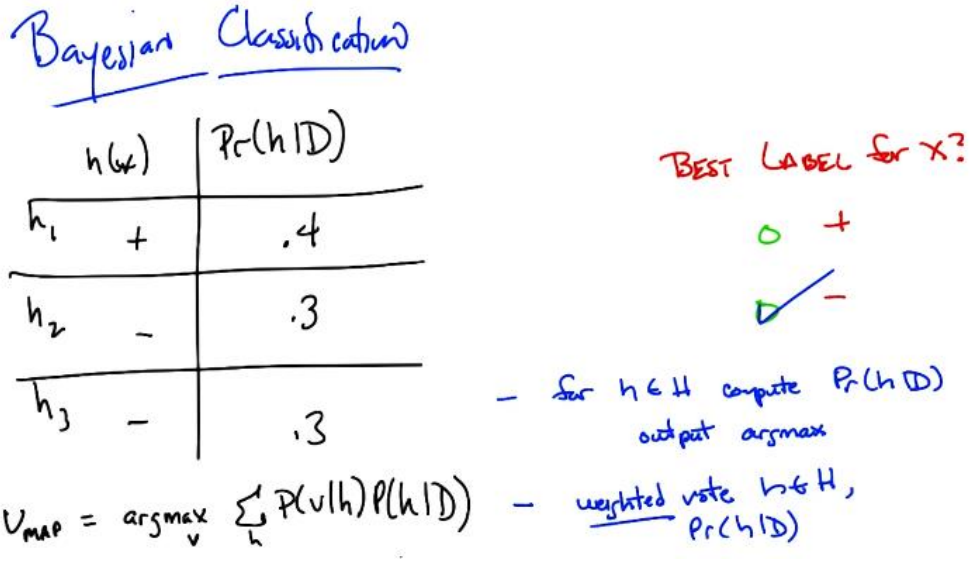
$argmin$ $length(D|h)$ + $length(h)$

"misclassification error" "size of h"

The "best hypothesis", the hypothesis with the maximum a posteriori probability, is the one that minimizes error and the size of your hypothesis. You want the most simple hypothesis that minimizes your error. That is pretty much occam's razor. What is important here in reality is that these are often traded off for one another. If I give a more complicated or bigger hypothesis, I can typically drive down my error. Or I can have a little bit of error for a smaller hypothesis. But this is the sort of fundamental tradeoff here. You want to find The simplest hypothesis that still explains your data, that is, minimizes your error.

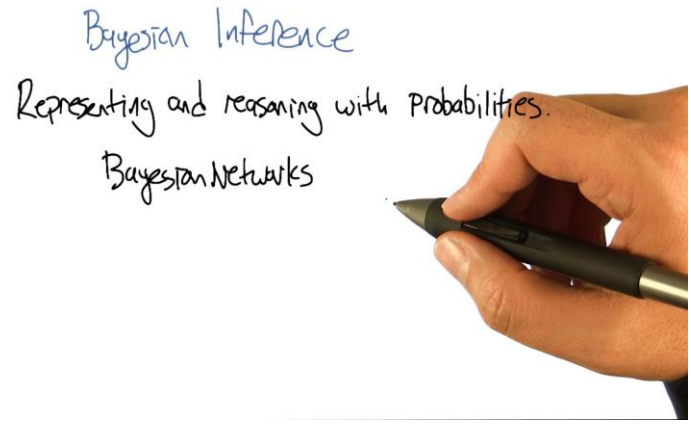
This actually has a name: Minimum Description Length. There have been many algorithms over the years that have tried to do this directly by simply trading off some notion of error, and some notion of size. And finding the tradeoff between them that actually works.

9.11. Bayesian Classification Quiz

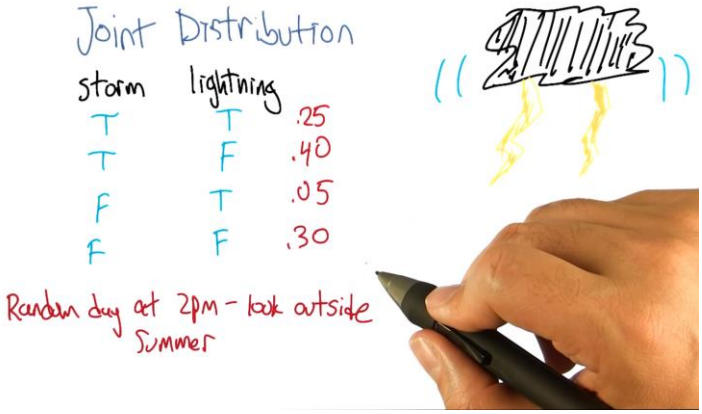


10. Bayesian Inference

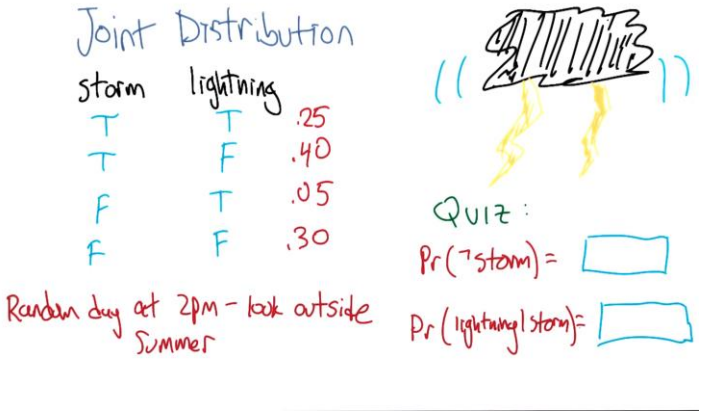
10.1. Intro



10.2. Joint Distribution



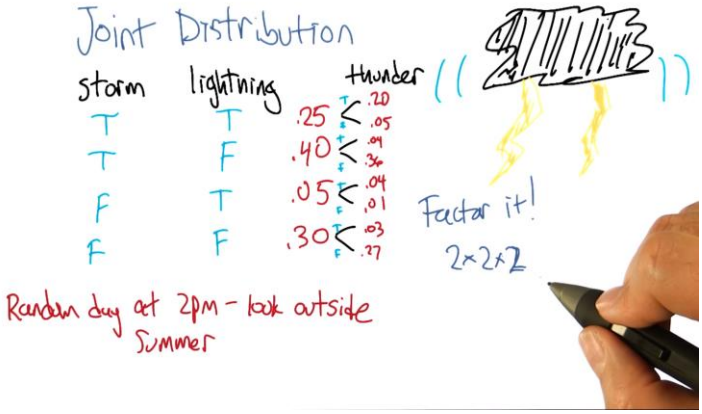
10.3. Joint Distribution Quiz



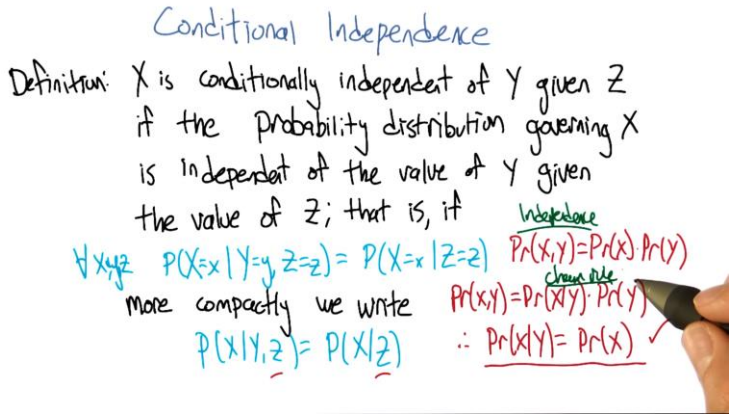
What is the probability:

- There is no storm - .35
- There is lightning given there is a storm - 0.25/0.65 = 0.3846

10.4. Adding Attributes



10.5. Conditional Independence



Here we define Conditional Independence as:

> X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given the value of Z; that is if:
>
> $$\forall x, y, z\ P(X = x|Y = y|Z = z) = P(X = x|Z = z)$$
>
> More compactly we write:
>
> $$P(X|Y, Z) = P(X|Z)$$

- The probabilities associated with the values in this variable X Is independent of the value of y given the value of z.
- So if I tell you what z is, then you can figure out what the probability of x is without having to look at y.

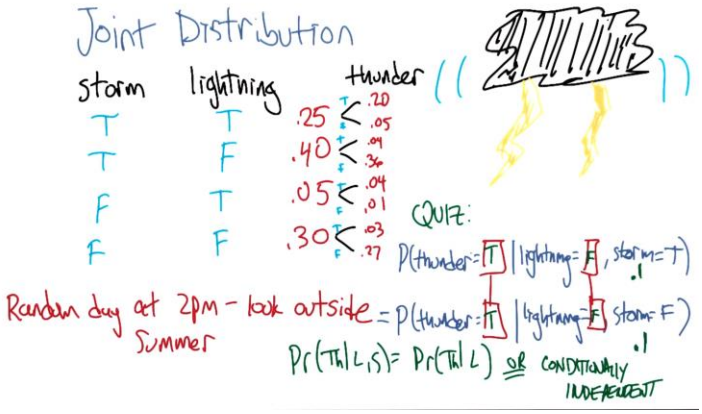
This is similar to the earlier notion we had of Independence and the Chain Rule

> Independence:
>
> $$Pr(x, y) = Pr(x) * Pr(y)$$
>
> Chain Rule
>
> $$Pr(x, y) = Pr(x|y) * Pr(y)$$
> $$\therefore\ Pr(x|y) = Pr(x)$$

- What independence means, right, is that the joint distribution between two variables is equal to the product of their marginals.

- Comes from basic probability theory and so if you think about what that means from the chainable point of view it's like saying the probability of x given y is equal to the probability of x.
- So, it looks just like the equation for conditional independence.

10.6. Conditional Quiz

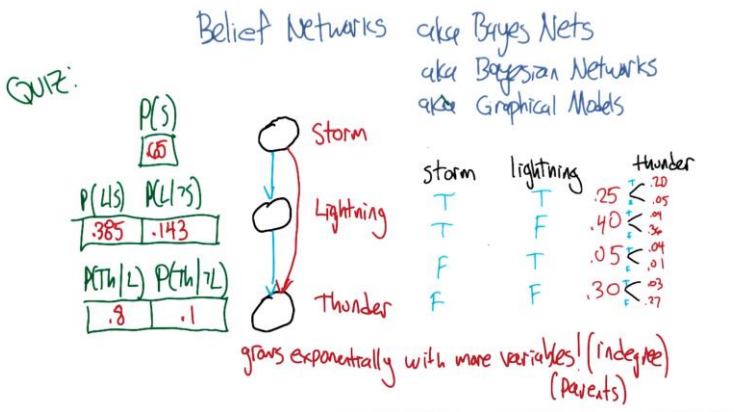


Find values for Thunder and Lightning that satisfies:

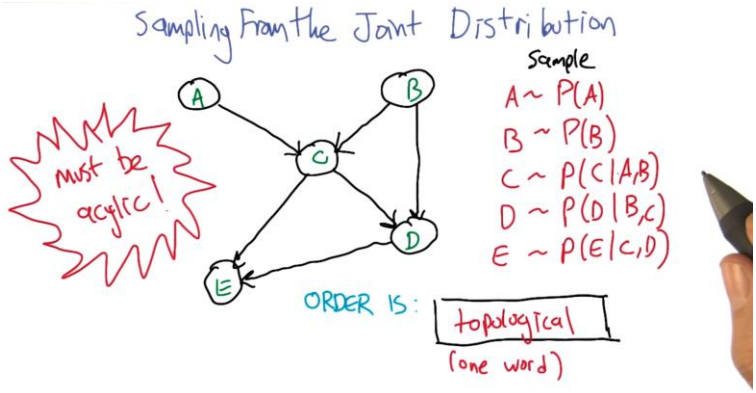
$$P(thunder = ? \,|\, lightning = ?, storm = T) = P(thunder = ? \,|\, lightning = ?, storm = F)$$

P(Th=T|L=T,S=T)=.8
P(Th=F|L=T,S=T)=.2
P(Th=T|L=F,S=T)=.1
P(Th=F|L=F,S=T)=.9
P(Th=T|L=T,S=F)=.8
P(Th=F|L=T,S=F)=.2
P(Th=T|L=F,S=F)=.1
P(Th=F|L=F,S=F)=.9
So any answer works!

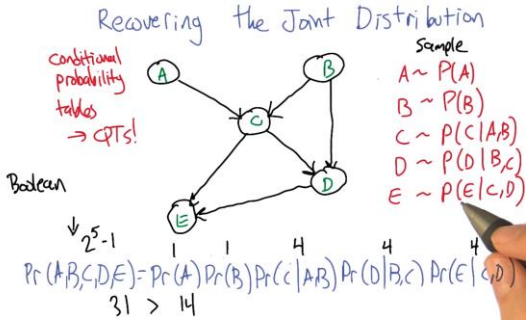
10.7. Belief Networks Quiz



10.8. Sampling from the Joint Distribution

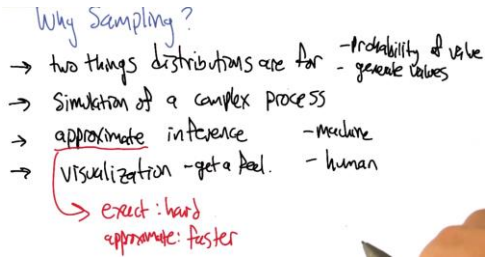


10.9. Recovering the Joint Distribution



- We can go from, from the values in these conditional probabilities tables in each of the nodes, to computing the probability of any combination, any joint combination of variables that we want.
- So, it turns out it's really, really simple. We can just go and use these same ideas and say the joint probability for some assignment to the variables, is equal to just the product of all the individual values.
- Here you have the exponential growth that only depends upon the number of parents you have.
- If you have no parents, then it is constant, if you have parents, then is grows exponentially with the number of parents.
- So the fewer number of parents, the more compact the distribution ends up being.

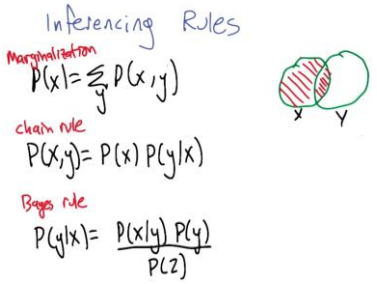
10.10. Sampling



- So why do you have a distribution?
- A distribution is so that given some value, you can, you can tell me what's the probability of me seeing that value which is kind of what it looks like when you have the probability function, but also if you have a nice distribution you can generate values according to that distribution.

    - A little bit circular in the sense that this answer doesn't tell 'why' it was useful to generate them other than it's one of the things you can do
- If a distribution represents kind of a process, it would be nice if I could duplicate that process, right? So, I would have to be able to generate values in the right way, consistent with the distribution in order to generate that process. Consistent with whatever the underlying bias is.
- So the basic idea is that you would like to do some inference, you'd like to figure out what might be true of the world in different situations.
    - Instead of doing some complex probability calculation, you're just going to imagine a bunch of possible worlds and see how often is it the case that whatever it is you want to figure out is true.
    - That's a lot of what people are doing when we're making judgments in the world. We're just really, really good at this kind of sampling from past realities that are relevant, and we can make judgments based on that
- Another thing is this notion of visualization. Which may be in a broader way than it sounds, not necessarily to actually see what the distribution is like, but to kind of get a feel for it.
- In the real world, there are perhaps hundreds and hundreds of variables with complicated relationships and conditional independencies that aren't necessary intuitive just by looking at the graph.
    - So picking one conditional probability table and looking at it isn't going to tell you much.
- But by sampling I get real examples that are concrete that, as a human being, I can understand without having to really glock all the 25 different conditional probability tables.
- This notion of approximate inference. Now generally we don't like approximations when we can do things exactly.
- However exact is hard and time-consuming (NP) while approximations are faster

10.11. Inferencing Rules

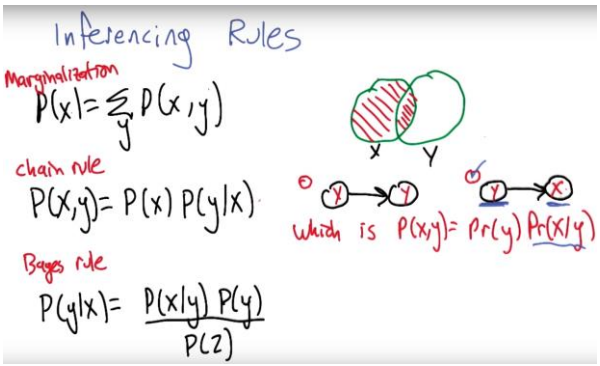


Here's just kind of a little cheat sheet

- **Marginalization** is this idea that we can represent the probability of a vaue by summing over some other variable and looking at the joint probabilities of those.
    - And if, if you've trouble remembering this one, this is how I like to think about it, if we're trying to figure out the probability of x, then one thing we can do is break it up into the cases where x and not y. Plus, places where x and y. So, the probability of x is it can be broken down into the probability of x when y is false plus the probability of x when y is true.
    - So it's really simple in that sense, but it actually turns out to be a useful thing to be able to do. To marginalize out.
- **Chain rule**, we've used this a bunch of times.
    - The probability of x and y can be written as the probability of x times the probability of y given x.
    - And that's important that we've the given X. If you drop that then it implies that they are completely independent of one another. In the case where the variables are independent, you can just look at their product. In the general case you actually have to look at the second one given the first one.
    - The order on the left doesn't matter, so, you've the probability of X times the probability of Y, but you could have written the probability of Y times the probability of, X given Y.
- **Baye's rule**, this time written correctly where the denominator has to be the probability of x
    - This has been discussed numerous times

These three together turn out to be our, you know, "three musketeers" in working out the probability of various kinds of events

10.12. Inferencing Rules Quiz



So, a person who's adept at manipulating Bayes Nets would know that this chain rule idea, this probability of X and Y can be written either as a probability of X times the probability of Y given X. Or

as the probability of Y times the probability of X given Y, actually correspond to two different networks. So which of these two networks corresponds to the fact that the probability of x and y, the joint probability of X and can be written as the probability of Y times the probability of X given Y

- So we know that from the last discussion we had about how you would recover the joint, that what you're saying on the right of this equation probability y times probability x given y means that the probability of y, the variable y doesn't depend on anything.
- So, between those two graphs the one on the right is the one where you're saying that you don't need to know the value of any other variable in order to determine the probability of y
- So it has to be the one on the right, and just to make sure if you look at the second product the probability of x given y says that while you determine the probability of x given the value of y and there is an arrow from y to x so, the second one is in fact correct.
- This is actually just one way you could read this network is to say what is this node x with an arrow coming into it? That is the probability of x. But, the, the things pointing into it are what's exactly being given. What it's being conditioned on.
- This is why when you look at a network, it's very hard not to think of them as dependencies. Even though they're not dependencies, they're conditional independencies.
    - Well the arrows are a form of dependence but it's not a causal dependence necessarily, it's again just the way the probabilities are being decomposed

10.13. Inference by Hand Quiz

Let's put some of these rules into play by actually doing some inference by hand. Ultimately, we're going to derive some algorithms that can do this so you don't have to think about it so hard. But understanding those algorithms, it's helpful to have gone through an exercise where you actually use these ideas.

- So here's a setup:
    1) Let's imagine that we've got two boxes. One has 4 balls in it and one has 5 balls in it. And we're going to choose one of those boxes uniformly at random. Either the box that we choose is equal to box 1, or the box that we choose is equal to box 2.
    2) After that, we're going to draw at random, uniformly at random, from what's inside the box, one of the balls, and let's say it turns out to be green.
    3) We reach into that same box a second time, and the question is, what's the probability that that second ball will be blue, given that the first one we drew was green?
- Now to create a Bayes Net. If you think about it as a process, which now means I'm thinking about this as things causing the other:
    1) The first thing that you did in the process is you picked the box. The first variable in the net is going to be the box variable.

        $$P(Box = 1) = 1/2$$

    2) Now pick a ball. The color of the ball you pick, depends upon the box. So the second variable here is what color ball you get when you do the first draw from the box. And we can represent this as a conditional probability table. So for box 1, it's three quarters green, one quarter yellow or orange, zero for blue. And for box 2, it's two fifths, zero, and three fifths. And so that captures what happens on the first draw.

        Ball 1

Box	Green	Yellow	Blue

1	3/4	1/4	0
2	2/5	0	3/5

3) For the second draw, it depends upon what you drew the first time. Because you said we were drawing without replacement. So it definitely depends upon what you, what you drew the first time. But also, it still depends upon the box.
    a) We've got tables for a box, we've got tables for ball 1, and we need to know what ball 2 is going to be. Well, the value that ball 2 takes definitely depends upon whatever value ball 1 takes. But it also depends upon which box you're in. So you need an arrow from there as well.
    b) So there's a lot of probabilities that we have to write down. But let's just write down a piece of that table.
    c) Let's say that the value of ball 2 depends on which box. And it depends on what ball 1 is. But let's just look at the piece of that table where ball 1 is green because that's what we're ultimately going to need here.

Ball 2, given Ball 1 was green without replacement

Box	Green	Yellow	Blue
1	2/3	1/3	0
2	1/4	0	3/4

QUESTION: So now that we have written it as a Bayes net, the question is, what's the probability that the second draw is blue, given that the first draw had been green.

$P(2 = blue|1 = green)$

To get this value then we need to add the probabilities from each box:

$P(2 = blue|1 = green, Box = 1)P(Box = 1|1 = green)$ + $P(2 = blue|1 = green, Box = 2)P(Box = 2|1 = green)$

- This follows just algebraically from two of the rules that we just talked about, it's the combination of the marginalization rule and the chain rule.
    - Marginalization rule lets us introduce the box variable. But the way that we wrote it before, it was, you have to 'and' it in but we can actually apply the chain rule to split that into a conditional probability.
- So the below values can be retrieved from our boxes above:

$P(2 = blue|1 = green, Box = 1)$ = 0

$P(2 = blue|1 = green, Box = 2)$ = ¾

- For the other two Probabilities we need more work:

$P(Box = 1|1 = green)$ = $P(1 = green|Box = 1)P(Box = 1)/P(1 = green)$ = $\frac{3}{4} * \frac{1}{2}/1 = \frac{3}{8}$

$P(Box = 2|1 = green)$ = $P(1 = green|Box = 2)P(Box = 2)/P(1 = green)$ = $\frac{2}{5} * \frac{1}{2}/1 = \frac{1}{5}$

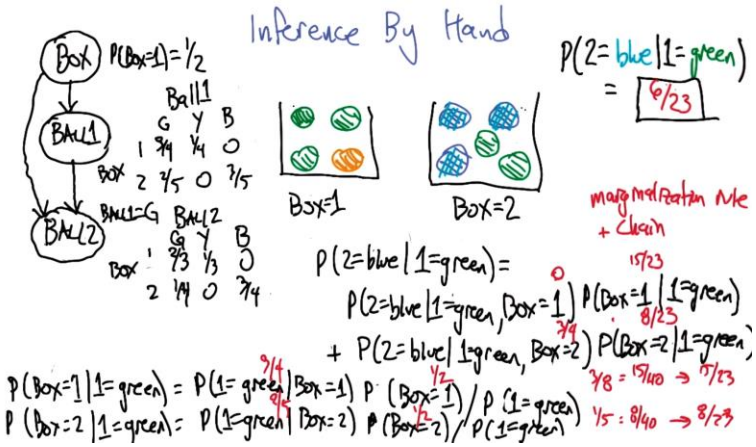
    - NOTE: here we disregard $P(1 = green)$. Since it requires more calculation, which we eventually don't need since both equations are dividing by the same value and they would not affect end probability
    - Now we need to normalize the values (⅜ and ⅕)

⅜ = 15/40 → normalize → 15/23

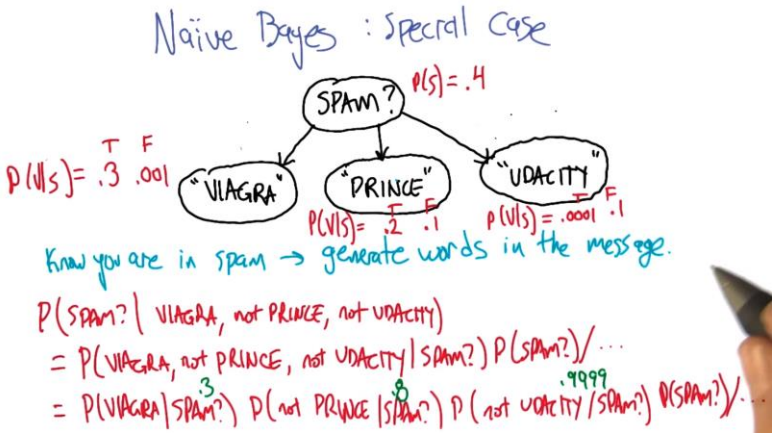
⅕ = 8/23 → normalize → 8/23 *Normalized values needed to add to 1

- So here we were able to compute the normalized results above without ever needing to know the value of P(1=green)
- So back to the original equation:

$$P(2 = blue|1 = green) = 0 * \frac{15}{23} + \frac{3}{4} * \frac{8}{23} = \frac{6}{23}$$



10.14. Naive Bayes



- This is a Bayesian network structure that is kind of capturing features of email messages as they come in. So, we should be able to answer questions like what's the probability that a given message is spam, given that the message has Viagra in it but not prince or udacity. So, how would we work this out?

$$P(SPAM?|VIAGRA, not\ PRINCE, not\ UDACITY)$$

- Apply Bayes Rule:

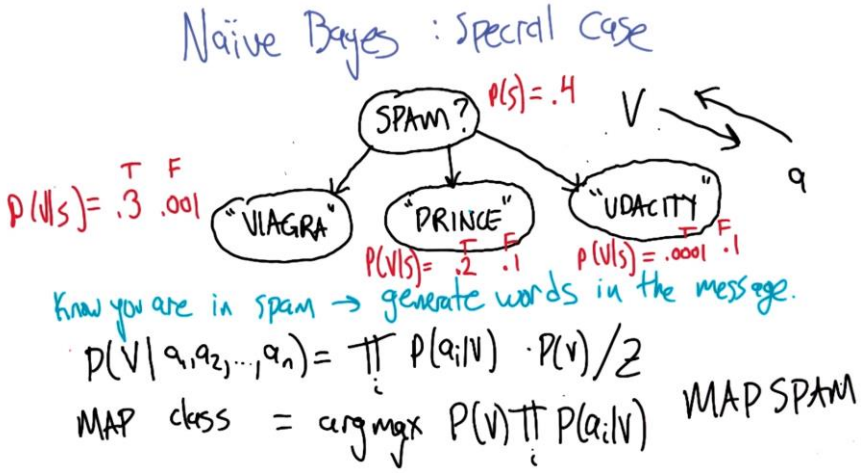
$$P(VIAGRA, not\ Prince, not\ PRINCE, not\ UDACITY|SPAM?)P(SPAM?)/\ldots$$

- Now we have applied Bayes rule, we have flipped things around, why is this giving us an advantage? For this kind of network structure it actually has a huge advantage because we can break this first quantity up.
- So this is a really convenient structure because it really just decomposes into all these separate helpful quantities.

- In particular, we can actually derive this by applying the chain rule. But what we end up with is that this joint probability over these three variables decomposes into a product of three independent joint probabilities.

  $P(VIAGRA|SPAM?)P(not\ PRINCE|SPAM?)P(not\ UDACITY|SPAM?)P(SPAM?)/..$

  .3 * (1-.2) * (1-.0001)

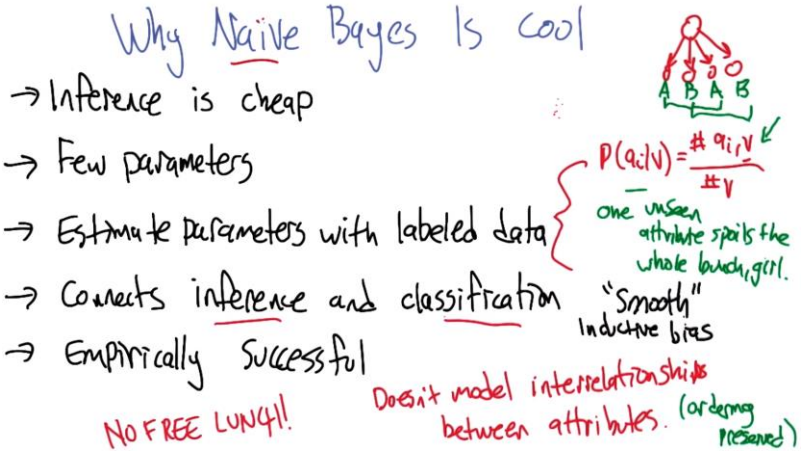


- We can do probability calculations in this setting, and that's actually going to give us answers to classification problems. And we're going to connect this back to machine learning. But but first let's write a general form of this formula.

  $P(V|a_1, a_2, a_n) = \pi_i P(\frac{a_i}{V}) * P(V)/Z$

- So essentially if you think of this top node as being the class, this is what was playing the role of V here, and these are all a bunch of attributes, then, even if we have a way of generating attribute values from classes, what this lets us do is to go the other way. That we observe the attribute values and we can infer the class.
- So the maximum posterior class if you're just trying to find what's the most likely class given the data that you've seen, you can just take an arg max over all the different possible values of that root node of the prob, its probability times the product of all the attribute values given that class.

10.15. Why Naive Bayes is Cool

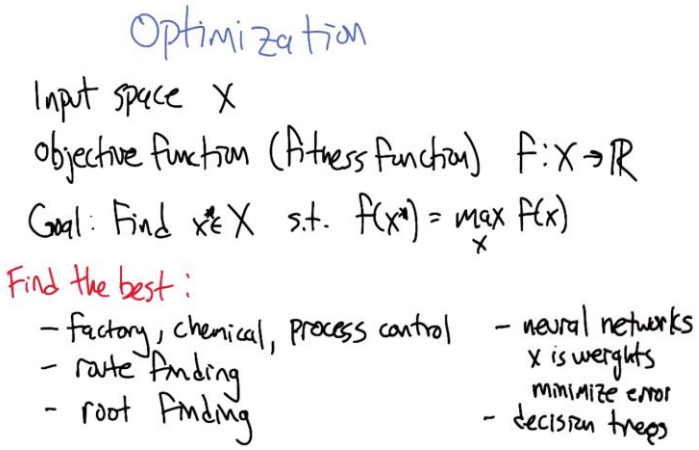


- Naive Bayes, where you have a network that has a label producing or conditionally producing a bunch of attribute values, is just a really cool and powerful idea.
- One of the issues is that, even though inference in general is a very difficult problem (NP hard), to work out what these probabilities are, when you have a naive Bayes structure, it's cheap using the formula that we had on the previous slide.
- The number of parameters that you need to write down, again even if you have a very large number of variables, it's not exponential in the number of variables, it's just linear.
- There's two probabilities for each of the attributes and one probability for the class. We can actually estimate these probabilities.
- So far, we've only been talking about Bayes Nets not in a learning setting, but in a setting where we just write down what all the numbers are. We can actually very easily estimate these parameters. How would we do that? Well the easy way to do it, is you count. When you're trying to estimate the probability of a particular attribute value given a class, it's really just in your labeled data. How often do you have an example that has an attribute value in that class, and then divide by the number of times you had that class at all, and that gives you the conditional probability.
- In the case of infinite data this is actually going to give you exactly the right number.
- It also connects this notion of inference that we've been talking about with classification.
- It actually allows us to do all kinds of interesting things like instead of only generating what the labels are, we can actually generate what attributes are. We can do inference in any of these directions.
- It turns out it's wildly successful empirically. Google uses a tremendous amount of Naive Bayes classification in what they do. If you have enough data you can estimate these values really well, and Naive Bayes is just remarkably good.
- However, there's no free lunch
    - The network says is that all of the attributes are conditionally independent given that you know the label, that just can't be true.
    - We talked about this before where we were using evasion inference to, to derive the sum of squared errors that it makes a very strong assumption about where your errors come from and an even stronger assumption about where your errors don't come from.
    - So you're not modeling any of the interrelationships, between, the different attributes and that just doesn't seem right.

- However, even though the probabilities are hard to believe are accurate, the classification tends to be right
- Another problem boils down to the equation in the slide. It's really nice and neat that you can compute the probabilities of seeing an attribute, given a value by just doing counting. But, we don't have an infinite amount of data. What if I'm unlucky enough that for some particular attribute value, I have never seen it paired with that label, V? The numerator would be 0, thus making the whole product 0.
  - In fact that's not what people often do. People will often, what they call ‘smooth’ the probabilities, by essentially initializing the count, so that nothing is zero, everything has a tiny little non-zero value in it. And there's smarter and less smart ways of doing that. However the zeroing out problem is a real thing and you have to be a little bit careful.

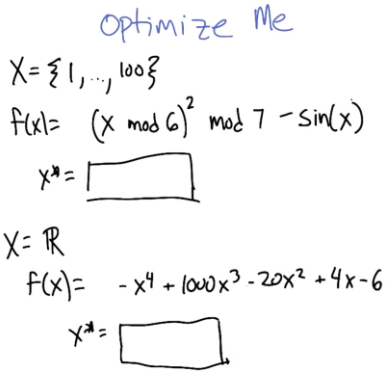
11. Randomized Optimization

11.1. Optimization



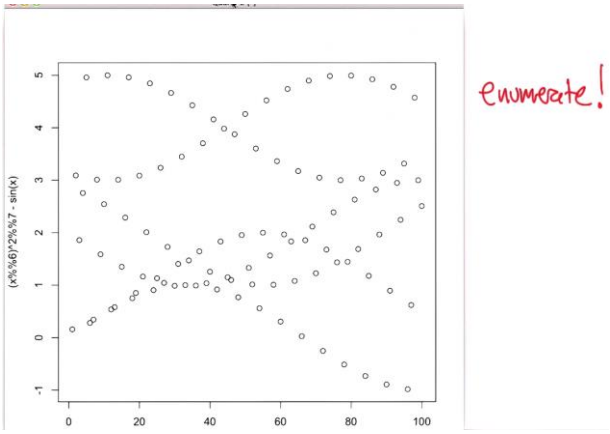
Minimizing error is a kind of optimization. Everything we did in the first third of the class is optimization.

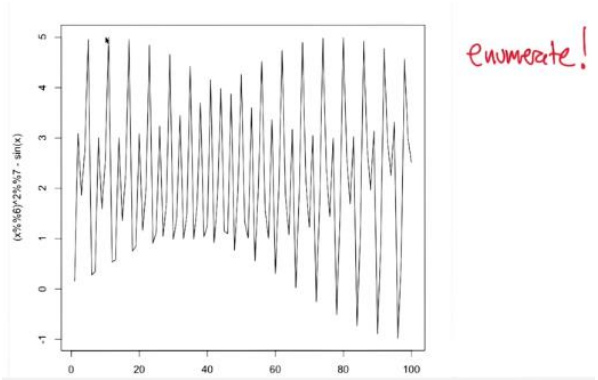
11.2. Optimize Me Quiz



- Here's two different problems, there's two different input spaces, two different functions, and I want you to find the optimal x* or something really close to it, if you can't get it exactly right.

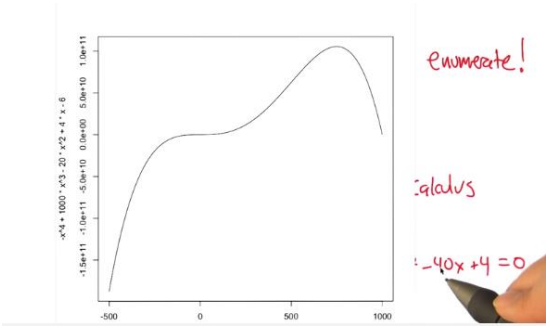
Number 1: x* = 11. Plotted:





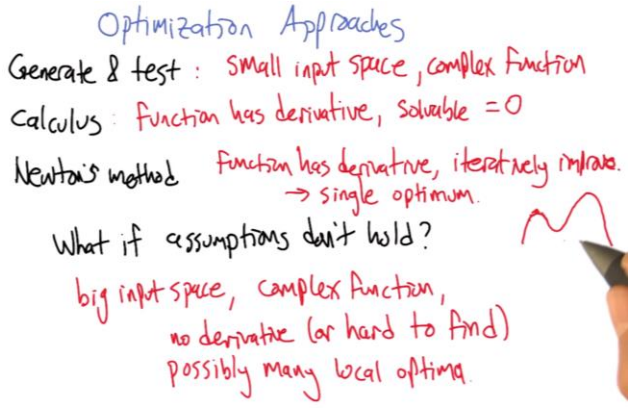
Same function as above but with dots connected.

Number 2: Use calculus because the function is a polynomial. However, the derivative is a cubic with 3 solutions. Take the derivative and set equal to 0. Time-consuming though. Answer is about 750 reached by visually analyzing the graph below.



It could also be analyzed using Newton's method: Guess a position and use the derivative at that point (does not involve solving the cubic) and do gradient ascend. The process converges at 750

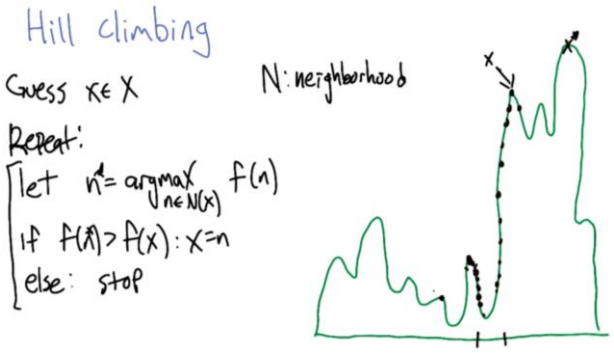
11.3. Optimization Approaches



- So in terms of optimization approaches, we actually just looked at a couple of different ideas.
- We just looked at generate and test, this sort of idea that you can just run through all the different values in the input space and see which one gives the maximum.
  - Good idea when you have just a small set of things you need to generate and test.
  - Requires small input space.

- Helpful if it's a complex function because there really isn't any other choice if the function has kind of crazy behavior, like in the mod example earlier
- For things like calculus, just solving for what the optimum is analytically.
  - Good idea when you have a function where you can do that (has to have a derivative)
  - How are you going to write down a function that doesn't have a derivative?
    - If what we're trying to optimize is crazy because we defined it to be crazy.
    - We can define it with ifs and things like that, to make it only piecewise continuous.
    - Or it might be the thing we're optimizing is some process in the real world that we just don't have a functional representation of.
    - All we can do is evaluate the fitness value at different inputs or if the inputs are discrete
    - So it might not actually give us any feedback if the derivative is not moving in a smooth continuous space.
- Newton's method can be helpful even outside of that case, where we have a derivative, and we have time to iteratively improve
  - Just keep querying the the function, creeping up on what the optimum turns out to be.
  - Wants you to have just a single optimum, because even Newton's method can get stuck if it's in a situation where you have multiple curves, like in slide above, because Newton's method is just going to hone in on the local peak.
  - So that would be bad if you have lots of local maxima in this case, or optima in this case.
  - So you can list this as possibly many local optima. The function has a kind of a peak. That things around the peak are less than the peak but that peak itself may be less than some other peak somewhere else in the space.
- What if these assumptions don't hold?
  - i.e. big input space, Complex function, No derivative, possible many local optima, or difficult to find derivative.
- So, as you might expect that our answer to this hard question is going to be randomized optimization.

11.4. Hill Climbing

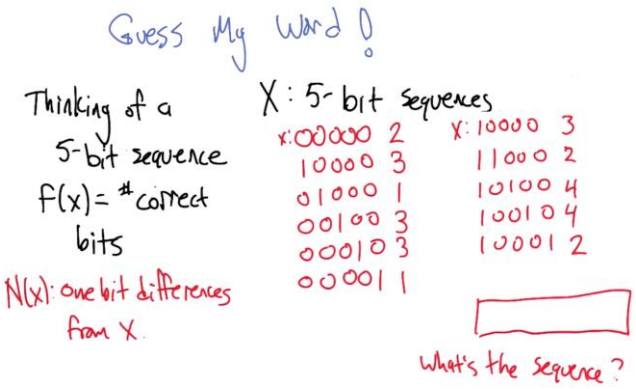


- So here's an algorithm for hill climbing.
- So if this is the function that we're trying to find the maximum of, then one thing we could do is imagine just guessing some x, I don't know, say, Which has some particular f of x value, and then to say, okay, well, let's move around in a neighborhood around that point, and see where we can go that would actually improve the function value.
- So here the neighborhood might be a little to the left, a little to the right on the x axis, and what we find in one direction it's going down and the other direction it's going up.

- So what hill climbing says is find the neighbor that has the largest function value.
- This is steepest ascent hill climbing.
- And if that neighbor is above where we are now, has a higher function value, then move to that point.
- Otherwise, we stop because we are at a local optimum.
- What this is going to do is to iterate moving up this curve, always in a better direction until it hits this peak here.
- Then, it's going to look on both sides of it and everything in the neighborhood is worse.
- So, it just stops there and this is the x that it returns.
- Not a bad answer, it's not the best answer, but, it's a good answer.

Other points in the slide were descriptions about if you chose a different value of x and how you would always just get the local maxima.

11.5. Guess My Word Quiz



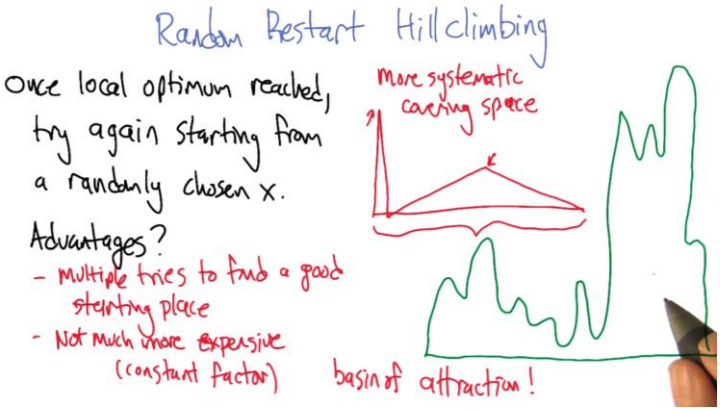
Define a neighbor function N(x) so that I can think about this as one bit differences from where you are. Let's start with all 5 zeros. The score for that is 2. The fitness function F(x) (the score) is the number of correct bits.
All possible neighbors: There are 5 of them. (writing the next 5 numbers and their scores on the side).
x is selected from the previous set and the next 4 are selected. From the numbers and their scores a human could find the answer after the first series…

10110

This is a friendly function, only a global optima.

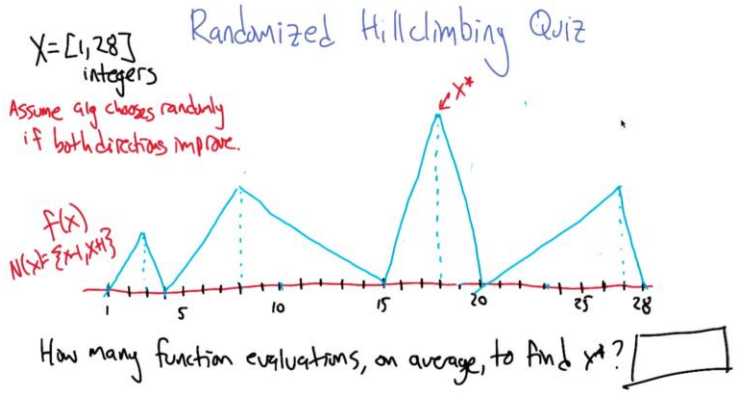
The advantage here is that we really understood the structure of our space.

11.6. Random Restart Hill Climbing



- So Random Restart Hillclimbing is going to give us a way to deal with the fact that hillclimbing can get stuck, and the place where it gets stuck might not actually be the best place to be. And so we can go back to that function that we looked at before.
- So what randomized hillclimbing's going to do is once a local optimum is reached, we're just going to start the whole thing again from some other randomly chosen x. It's sort of what you do if you were trying to solve a problem and you got stuck.
- This method takes away the luck factor of happening to pick a good starting place.
- So you get multiple tries to find a good starting place. That there could be various places where you start that don't do so well but as long as there's places where you do well then you might luck into starting one of those places and climb up to the tippy top and win.
- Another advantage is that it's actually not much more expensive. So, whatever the cost is, of climbing up a hill, all you've done is multiply it by a constant factor, which is how many times you are willing to do a random restart.
- If there is only one optimum and there is no local optimum, then we'll just keep getting the same answer. So, could be that we might keep track of that and notice that you seen in a space where these random restarts aren't getting us any new information. So, you might as well stop now.
- Also, maybe you just keep starting too close to the same place you were starting before. It may be random, but you can get unlucky random selections. So, maybe you should make certain your next random point is far away from where you started, so that you cover the space. You want to cover the space as best you can. So we could try to be more systematic.
- So here (drawn in slide above in red) might be an example of a function where that would be really relevant. So imagine here's our input space and most of the random points that we choose are all going to lead us up to the top of this hill. But a very small percentage are actually going to lead us to the top of the real hill, the top that we really want: The optimum. We might not want to give up after a relatively small number of tries because there could be a small “basin of attraction” and if it is small enough it could take lots of tries to hit it. In fact it could be a needle in a haystack in which case there is only a small place you can start that reaches that optimum.
- So the assumption here is that you can make local improvements and that local improvements add up to global improvements.

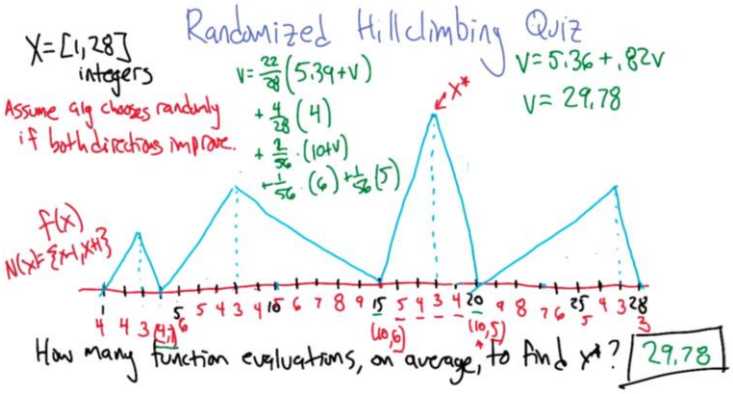
11.7. Randomized Hill Climbing Quiz



1-4:3	4/28
5-14:8	10/28
15-20:18	6/28
21-28: 27	8/28

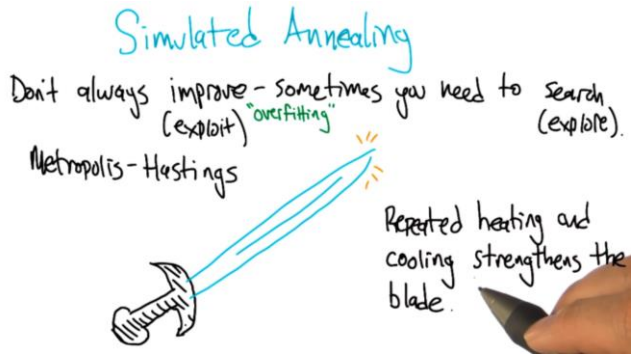
….

Long-winded question



Bad example overall since there are only 28 total numbers, and this returns more than that.

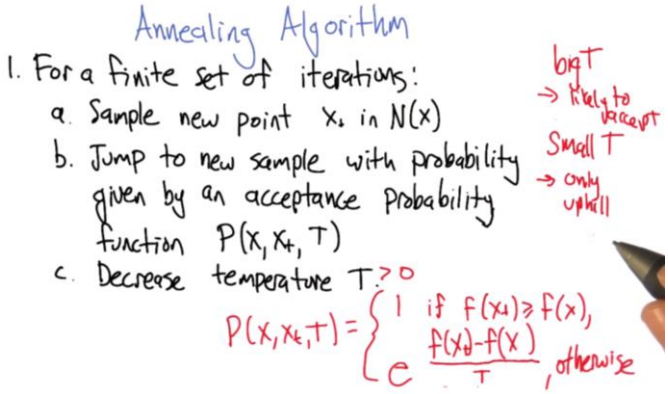
11.8. Simulated Annealing



- Instead of just waiting until you hit a local optimum to decide that you're going to randomly restart, there's always a chance of taking a step in a downward direction while you're trying to do your hill climbing.
- So the basic idea is that we don't always improve. Sometimes we actually need to search. We need to take the point where we are now and wander away from it, with the hope of finding something even better.
- Think of this as being related to the notion exploring and exploiting.
    - Hill climbing is always trying to exploit. It's always trying to climb its way up the hill as quickly as it can, which can lead it to getting stuck.
    - Exploring is the idea of visiting more of the space with the hope that you can actually climb even further.
- Trade these off really carefully. If you're just exploring, it means you're just randomly wandering around the space. You're not using any of the local information to improve yourself.
- But if you only exploit then you can get stuck in these local optima. So you need to do a bit of both.
- If you exploit all the time that's kind of like overfitting. That's like believing your data too much and not taking any chances at all. The fundamental problem of overfitting is believing your data too much. And dealing with just the coincidences of what you happen to see. And that's being very myopic. And exploiting, in this case, only taking the direction which you go is like believing the data point where you happen to be. I predict, from this one example, that I should be headed in this direction, and I don't care about anything else. And I'm not going to worry about anything else. Well, that's kind of like believing too much, which is sort of what overfitting is.
- Meanwhile search on the other hand, the exploring, is like believing nothing and taking advantage of nothing. Obviously you need to take advantage of the local information. You need to believe it at least a little bit at least some of time time or otherwise it's as if you've learned nothing.
- Alright, well I definitely agree with you that there's a trade-off there and that there's a resemblance to overfitting.
- The simulated annealing algorithm is related to an algorithm called **Metropolis-Hastings**.

11.9. Annealing Algorithm

Simulated annealing is a probabilistic method for finding the global minimum of a cost function that may possess several local minima. Annealing involves heating and cooling a material to alter its physical properties due to the changes in its internal structure. As the metal cools its new structure becomes fixed, consequently causing the metal to retain its newly obtained properties. In simulated annealing we keep a temperature variable to simulate this heating process. We initially set it high and then allow it to slowly 'cool' as the algorithm runs. While this temperature variable is high the algorithm will be allowed, with more frequency, to accept solutions that are worse than our current solution. This gives the algorithm the ability to jump out of any local optimums it finds itself in early on in execution. As the temperature is reduced so is the chance of accepting worse solutions, therefore allowing the algorithm to gradually focus in on a area of the search space in which hopefully, a close to optimum solution can be found. This gradual 'cooling' process is what makes the simulated annealing algorithm remarkably effective at finding a close to optimum solution when dealing with large problems which contain numerous local optimums.



The Annealing Algorithm is remarkably simple and remarkably effective.

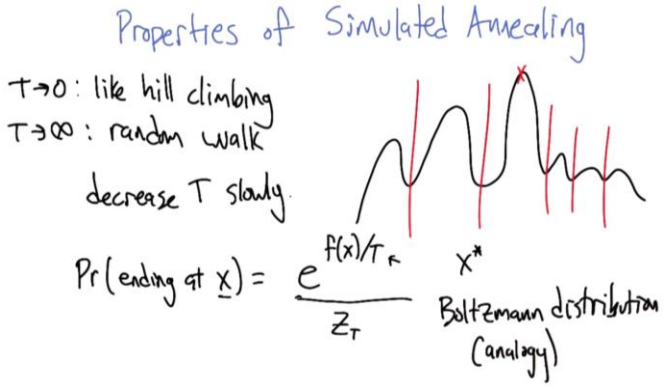
- Repeat for some finite set of iterations.
- We're going to be at some point x, and we're going to sample a new point x_t, from the neighborhood of x.
- We're going to move our x to that x_t probabilistically. In particular, we've got this probability function $P(x, x_t, T)$, which is going to help us decide whether or not to actually make the move.

$$P(x, x_t, T) = 1 \; if \; f(x_t) \geq f(x), e^{\frac{f(x_t)-f(x)}{T}} \; otherwise$$

- So, the probability that if we're currently at x, and we're thinking about moving to x_t and the current temperature is T, then what's going to happen?
  - If the fitness of the new point is bigger than or equal to the old point, we're just going to make the move. It's a little different from the hill climbing the way we described it, where we said let's visit all the points in the neighborhood. This is kind of a useful thing to be able to do when the neighborhood's really large, just choose anything in the neighborhood, and if it's an improvement, you know, go for it.
  - If it's not an improvement then what we're going to do is:
    - Look at the fitness difference between the point that we're evaluating, and the point where we are now.
    - Divide that by the temperature

    - Take e to that
    - Interpret that as a probability. And we either make the move or not according with that probability.
  - If the point that we currently visited x is infinitesimally close to neighbor point x_t, then that means that difference is going to be very close to 0. $e^0$is 1, so we make the move if it's infinitesimally smaller than where we are now.
  - If it's a big step down that means that number's really negative. And a negative divided by some positive number T. That will be a really big negative number, and e to a really big negative number is 1 over e to a really big number. So that makes it very close to 0. If a giant step down, we probably won't take it. We are smoothly going in directions that kind of look bad as a function of how bad they look and exaggerating that difference through an exponential function. Let's say there is a moderate step down that is not so huge that when dividing by T is negative infinity, but is something like -5. What happens? If T is very big, close to infinity, it does not matter what the difference is. When the temperature is really high we are willing to take downward steps. If T is infinity, even if the neighbor is much worse off, basically you will jump to the next neighboring point. If T is small, as T approaches zero, any difference between $f(x_t)$ and f(x) gets magnified by the small T and will go uphill.
  - There's lots of randomness happening.

11.10. Properties of Simulated Annealing

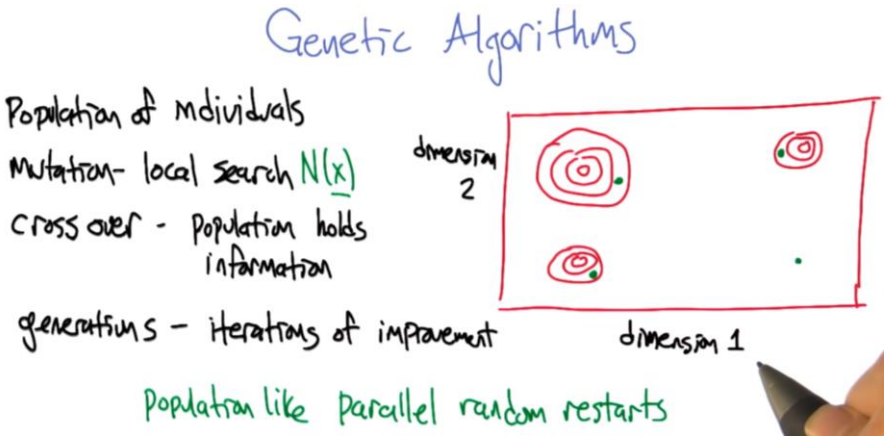


- As the temperature goes to zero, it's acting more and more like hill climbing, only taking steps that improve the fitness.
- As T goes to infinity, it's like a random walk around the neighborhood.
- The algorithm decreases T, but do we need to decrease T quickly or slowly. What's the right way to do it?
- In practice, we want to decrease the temperature slowly, because it gives the system a chance to explore at the current temperature before we start to cool it out.
- We're not going to be able to go through the argument for exactly why this is true, but there is a remarkable fact about simulated annealing that is worth mentioning. That the probability of ending at any given point x in the space is actually $\frac{e^{\frac{f(x)}{t}}}{Z_t}$, e to the fitness of x divided by the temperature and then normalized, because this is a probability distribution over the input space.

- This is pretty remarkable. This is saying that it's most likely to be in the places that have high fitness because those are where it's going to have the highest probability of being.
- You can see now what the relationship with the temperature is here too, that as we bring the temperature down, this is going to act more like a max. It's going to put all the probability mass on the actual x^*, the optimum. And as the temperature is higher, it's going to smooth things out and it'll be randomly all over the place.

- That's why it's really important that we eventually get down to a very low temperature. But if we get there too quickly, then it can be stuck because it is spending time ending up at points that are not the optimum, if they're say, close to the optimum.
- This distribution has a name it's called the **Boltzmann distribution**.

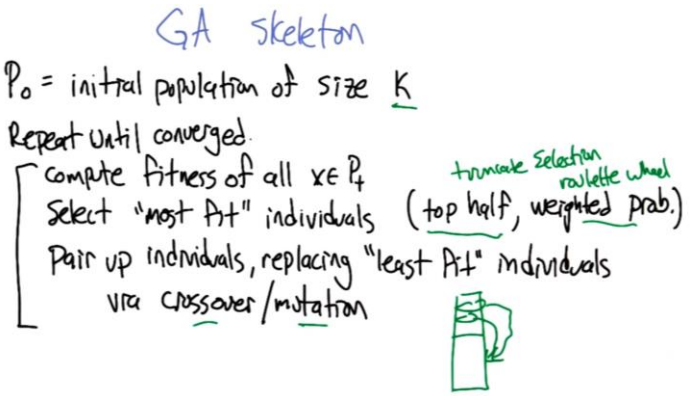
11.11. Genetic Algorithms



- Let's imagine we've got a two dimensional space. Think of this as being one of those contour maps.
- Imagine that x now comes in these two different dimensions. What we're trying to do is find the peak, which happens to be there. We actually check what the values are at these green points.
- So what we find is that, from this initial point, this green point here. If we increase on dimension 2, we get a better value. Also, if we increase on dimension 1, we get a better value. Maybe what we ought to do is take elements of these 2 solutions, these 2 inputs, and combine them together. Move out on dimension 1 and dimension 2 and maybe that will actually give us a good score as well. In this particular case, it puts us right in the basin of attraction of the local maxima.
- This turns out to be useful in many spaces, especially spaces that can be specified combinatorially like this. Where there are separate dimensions that contribute in various ways to the overall fitness value.
- What we're going to do build an analogy with Biological evolution. In particular, instead of thinking about these input points, these little green dots, we're going to think of each input point as an individual, and a group of them taken together as a population.
- The the idea of local search where you make little changes to an input, we're going to now call that mutation.
- These are all concepts that we were already using when we were doing these other randomized optimization algorithms. One thing that's different though, is the notion of **crossover**.
- Crossover takes different points and gives you a way of combining their attributes together with the hope of creating something even better. So, that is where it starts to actually kind of deviate

from the standard notion of local search or randomized optimization. And gets us into something that feels a little more like evolution. What we were calling iteration before in the context of genetic algorithms, we can call it a generation.

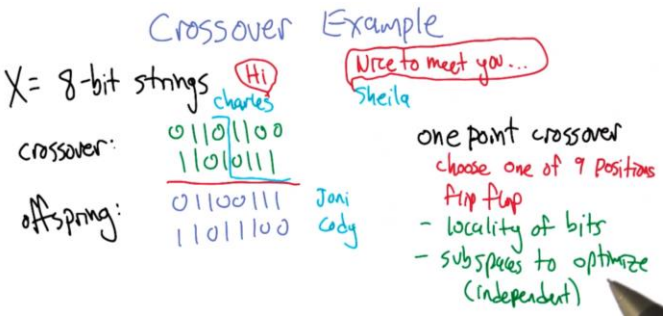
11.12. GA Skeleton



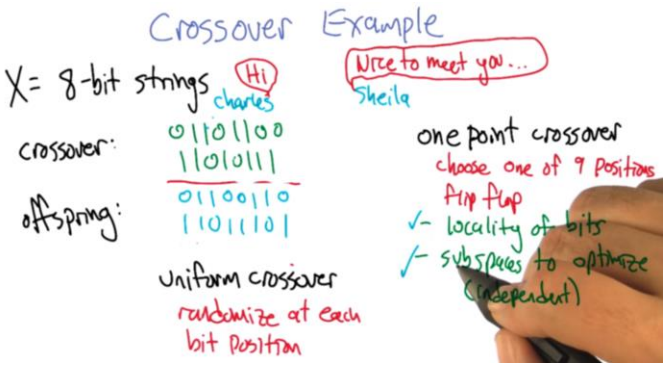
Here's a skeleton of an algorithm that implements a GA.

- Start off with some initial population and usually the population size is fixed to some constant, we'll call it K. Generate a bunch of random individuals to get things started.
- Then repeat until essentially things converge.
  - Compute the fitness of all of the individuals in the population.
  - Select the most fit individuals, whatever the fitness function tells you is fit. Apply the fitness function f to all the individuals and select the ones with the highest scores, say, top half of the population in terms of their scores, and we declare them to be the most fit and everybody else to be the least fit.
    - There's other ways you can do it as well i.e. truncation selection and there's also an idea called roulette wheel selection where what you do is you actually select individuals at random but you give the higher scoring individuals a higher probability of actually being selected. So we don't just strictly choose the best ones, we choose weighted by who's the best.
    - Similar idea to exploration and in fact you can use Boltzmann distribution type ideas similar to the annealing type idea for doing this selection where you have a temperature parameter. If you set the temperature parameter to zero then you get something like just choosing the top half. And if you set the temperature to be something like infinity then you're just going to randomly choose samples from the population irrespective of the fitness of those individuals.
  - Pair up those most fit individuals. This is like a dating service now. And let them produce offspring using crossover and maybe a little bit of mutation too. So instead of just taking the combination of the two parent individuals, we take their combination and then we make little local changes to it, to mutate them. And we let the value of that new offspring replace one of the least fit individuals in the population.
- We can define crossover in many different ways.

11.13. Crossover Example



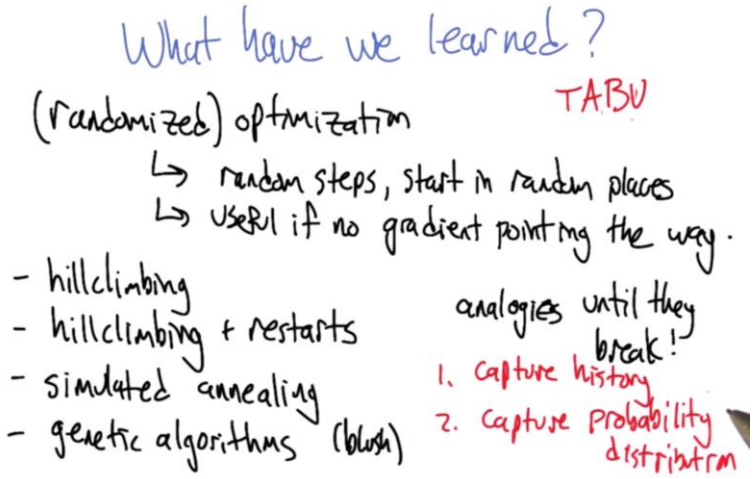
- The crossover operation is always going to depend critically on how you represent the input space.
- Let's say concretely that our input space is eight bit strings.
- Here's two parents, 01101100 and 11010111 and now we're going to use them to create a new individual, a new offspring.
- Now we've put these two bit sequences together and we've lined up so that the bits correspond in each of the different positions and want to generate a new individual that uses elements of the two individuals that we have.
- There are many ways to do this, if we really push the genetic notion as far as we can then each of those things represent some biological representation, like chromosomes. What happens in genetics is you mix and match your chromosomes and alleles together. Let's say one child is the first four bits of the first string and the last four bits of the last string.
- Pick a random number along the sequence at the halfway point and now mix and match and create two offspring.
  - One uses the first half of Charles and the second half of Sheila
  - The other one is the other way around.
- This particular way of combining where you randomly choose a position and then flip flop is called **one point crossover**.
- 2 kinds of assumptions built there. Could be it's a bias of some sort.
  - One assumption is that locality of the bits matter. By picking halfway through you are saying the first four bits are somehow related and the last four bits are somehow related because otherwise this wouldn't make any sense.
  - Second point, which is just a first point, is that it assumes that there are subparts of the space that can be independently optimized that you can then put together. In particular they're independent part of the subspace, there's these two dimensions and each dimension kind of matters independently and the total reward or the total fitness is some kind of linear combination of them.
  - If those two things aren't true than really doing crossover like this won't help you at all. You're just kind of randomly mixing things together.
- It's kind of an assumption about the way space works, like the example we did when we were doing bit guessing, you can be heading in a good direction, that there are pieces that are right and if we reuse those pieces we can get even righter.



- If it is the case that the sequence of the ordering of bits matters, we have this locality property.
- Let's focus on ideas where we still have this *subspace to optimize* property. But we don't really have a *locality of bits* property, the ordering doesn't matter anymore. So keeping them clumped together like that is not a useful thing.
- With the one point crossover it really matters that you know the two bits that are next to each other are very likely to stay connected, that is it's unlikely that the split will happen exactly between them and so will tend to travel as a group.
- But, if we don't think it's important that the bits next to each other need to travel together (that It should be equally likely for any of the bits to remain together) we need to crossover a lot more than just that one time. We might need to cross over every time.
- We could generate individuals by just scrambling at each bit position. Same values stay the same, different values are assigned to each offspring separately:

  01101100
  11010111
  01100110
  11011101
- So now, we've got two individuals, and every bit from these individuals comes from one of the parents and so that means that if there is sub pieces that are correct that may be preserved in the offspring but no longer does it matter what the ordering is. We get exactly the same distribution over offspring, no matter how we order the bits.
- So this idea is sometimes called **uniform crossover**. And essentially, we are just randomizing at each bit position.
- This kind of crossover happens biologically at the level of genes so we imagine that we get our genes from our parents but for each different gene, like the gene for eyes and the gene for hair color, are not particularly linked to each other, they're uniformly chosen at each position.

11.14. What Have We Learned



- There's lots of tweaky things that you need to do to get this to work very effectively.
- You have some choice about how to represent the input, and you have some choice about how you can do your selection, and your fit to finding your fitness function. But, at a generic level, this is a useful thing. Some people call genetic algorithms the second best solution to any given problem. So, it's a good thing to have in your toolbox.

What have we learned?

- Randomized optimization, we take random steps where we start off in random places and it's a way to overcome when you can't take a natural gradient step.
- Hillclimbing. And we had 2 flavors of that including Random Restart Hillclimbing
- We did simulated annealing.
- We did genetic algorithms.
- We talked a little bit about how this all connects back up with learning, because in many cases, we're searching some parameter space to find a good classifier, a good regression function. This notion of finding something that's good, finding a way to be optimal is pervasive throughout machine learning.
- AI researchers like analogies, they actually like pushing analogies until they break. Every single thing in this lecture is an analogy to something.

2 observations I want to make.

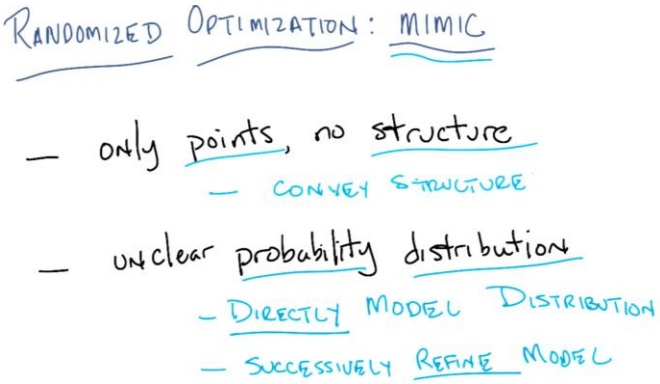
- One is, I'm looking at hill climbing, that makes sense; hill climbing restarts makes sense; simulated annealing makes sense, but, they don't really remember a lot.
    - So you do all this hill climbing and you go 8 billion steps, and then what happens? You end up with the point.
    - You do simulated annealing. You do all this fancy stuff with slowly changing your temperature at every step, the only thing you remember is, where you are and maybe where you last were.
    - With genetic algorithms, it's a little more complicated that because you keep a population, but really you're just keeping track of where you are, not where you've been.
    - So in some sense, the only difference between the 1 millionth iteration, and the 1st iteration is that you might be at a better point. And it just feels like, if you're going to go through all this trouble of going through what is some complicated space that hopefully has some

structure, there should be some way to communicate information about that structure as you go along.

- The second thing is, what I really liked about simulating annealing is that it came out at the end with a really nice result, which is this Boltzmann distribution, that there's some probability distribution that we can understand, that is actually trying to model.
- So, here are my questions then. It's the long way of asking a real simple question. Is there something out there, something we can say more about? Not just keeping track of points, but keeping track of structure and information. And is there some way that we can take advantage of the fact that, all of these things are somehow tracking probability distributions just by their very nature of being randomized.
- These are all kind of amnesic. They kind of just wander around, and forget everything about what they learned. They don't really learn about the optimization space itself. And use that information to be more effective.
- There are some other algorithms that these are kind of the simplest algorithms, but you can re-combine these ideas you know, sort of crossover style, to get other more powerful algorithms.
    - There's one that's called **taboo search**, that specifically tries to remember where you've been, and you're supposed to avoid it, they become taboo regions.
    - And then there's other methods that have been popular for a while that are gaining in popularity, where they explicitly model the probability distribution over where good solutions might be.
    - So they might be worth actually talking a little more about that.

This leads into MIMIC, efforts to resolve these issues, discussed by Dr Isbell

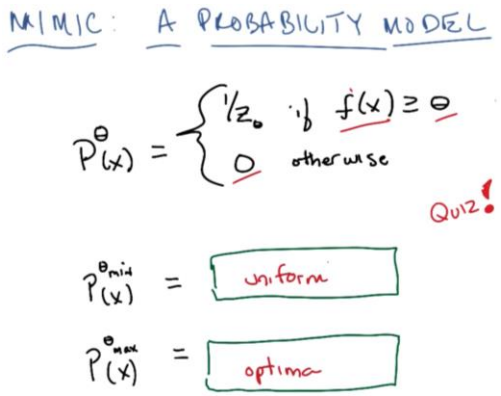
11.15. MIMIC



- We have all these cool little randomized optimization algorithms. Most of them seem to share this property that the only thing that really happens over time is that you start out with some point and you end up with some point, supposedly the optimal point. And the only difference between the first point and the last point (or any point in between) is hat point might have been closer to the optimum by some measure. And very little structure was actually being kept around or communicated, only the point was being communicated.
- You could argue this isn't quite true with genetic algorithms, but really you move from a single point to just a few points.

- The other problem that I had is that we had all of this sort of probability theory that was underneath what we were doing, all this randomization. But somehow it wasn't at all clear in most of the cases, exactly what probability distribution we were dealing with.
- What I decided to do is to go out there in the world and see if I could find a class of algorithms that took care of these two points for us. It turns out that I wrote a paper about this, almost 20 years ago. I will point that a lot of other work has been done since this that refines on these ideas. I just want to go over the high level bit here because I really think it kind of gets at this idea.
- In particular, the paper that I'm talking about introduced an algorithm called **MIMIC** which has a very simple structure to it. The basic idea was to directly model a probability distribution. And given that you have this probability distribution that you're directly modeling the goal is to do this search through space, just like we did with all the rest of these algorithms, and to successfully refine the estimate of that distribution.
- If you can directly model this distribution and refine it over time, then that will in fact convey structure, what we're learning about the search space while we're doing the search. And not just simply the structure of the search space, but the structure of the parts of the space that represent good points or points that are more optimal than others.
- This simple MIMIC algorithm captures these basic ideas, it's fairly simple and easy to understand, while still getting some of the underlying issues.

11.16. A Probability Model Quiz



- Here's a probability distribution over x, parameterized by Θ, and Θ is going to turn out to stand for threshold.
  - Probability distribution = $\frac{1}{z_\Theta}$ for all values of x such that the fitness function is greater than or equal to Θ.
  - And it's 0 otherwise.

$$P^{\Theta}(x) = \frac{1}{z_\Theta} \; if \; f(x) \geq \Theta \; ; \; 0 \; otherwise$$

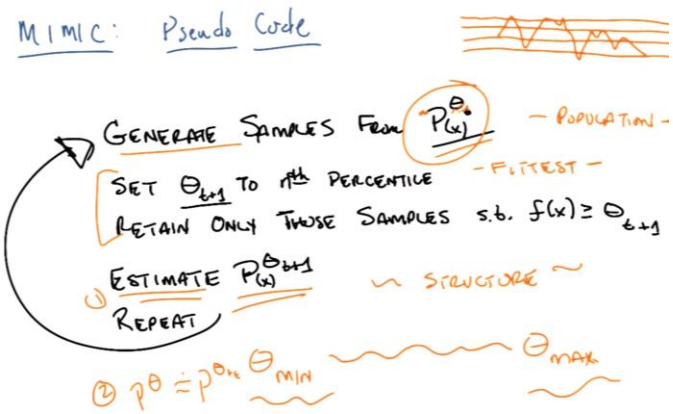
- z_Θ here is a kind of normalization factor that accounts for the probability of landing or choosing an x in that space. In the space of high-scoring individuals above threshold.
- Another way of saying this is that this probability is uniform over all values of x whose fitness are above some threshold.
  - Kind of like a mountain range and if you put a slice through it then everything that's kind of above that slice, if we are going to sample uniformly from that collection.
  - And everything below it, we will simply ignore (doesn't get sampled from)

- Θ is some threshold and so let's imagine the fitness function has to have some minimum value and some maximum value. Let's call those Θ_{min} and Θ_{max} respectively.
- I want you to describe in one or two words:
    - $P^{\Theta \quad mkn}(x)$, that is the probability distribution where the threshold is its minimum value
    - $P^{\Theta \quad max}(x)$, that is the probability distribution where the threshold is its maximum value.
- Let's assume that Θ_{min} is the lowest value that the fitness function can take on and Θ_{max} is the largest value of the fitness function.

Answer:

- So Θ_{max} is going to be the largest value that f can return, which is actually the optimum of the function. So that probability distribution assigns a probability of 1 if there's a unique optimum, it'll decide a probability of one to that single point.
    - What if there are multiple optima? Then it's uniform over all of them.
    - So, it is the distribution that generates only optimal points. So, it's a distribution over optima.
- For Θ_{min} if it's the minimum that the function can achieve then it ought to be the case that everything in this space of x's is part of that. So it should assign uniform probability to all points in the input space.
- So it is in fact, simply, the uniform distribution.
- Now, this is going to be pretty cool, because in this, in this slide right here we basically have the MIMIC algorithm.
    - We are basically going to try to estimate this particular distribution; $P^{\Theta}(x)$
    - We're going to start out with $P^{\Theta \quad mkn}(x)$, which is the uniform distribution.
        - something that's really easy to sample from because we're just sampling uniformly from the input space
    - We're going to sample uniformly from all of the points, and then somehow use those points to do better and better estimates until we get from uniform to a distribution of only the optimal points.
        - something that'd be really useful to sample from because, if we could sample from the set of optima, it's really easy to find an optimum

11.17. Pseudo Code



Here's the basic idea (in pseudocode)

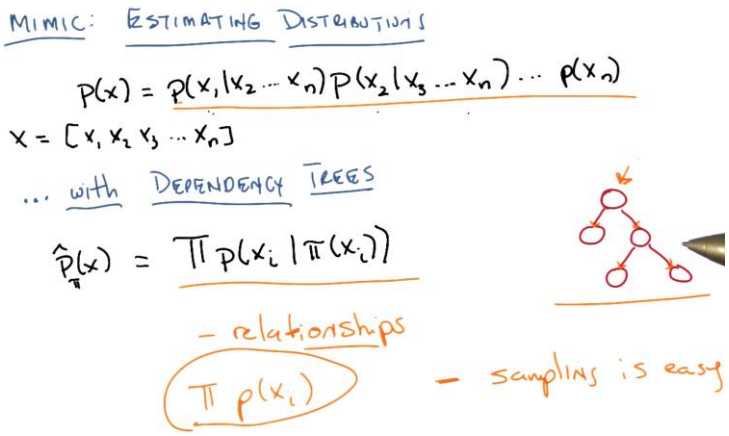
- We are in the middle of this process, we have some Θ at some time step t, and we're simply going to generate samples that are consistent with that distribution, so, we're going to treat our probability distribution as not just something that gives us a probability but something from which we can sample.
- Generate samples according to P^{Θ_t} . Generate a bunch of those samples.
- Set $\Theta_{t+1}$to n^{th} Percentile
    - Now that we have these samples, we're going to come up with a new Θ_{t+1}. And that Θ_{t+1} is going to be the best samples that we just generated. the n^{th} Percentile. Let's say the top half.
    - It's a lot like genetic algorithms except instead of having this population that moves from one bit to another, we generate samples that's like our population. We pick the most fit of that population by retaining only those that are the best ones.
- Estimate $P^{\Theta_{t+1}}(x)$
    - Now that we've got this new population, rather than using that population again, we estimate a new distribution that's consistent with those.
- Repeat - then we just lather, rinse, repeat until we come to some conversions.
- Feels a lot like the genetic algorithm except one of the things that we cared about, a structure. So this structure that we're maintaining, remember this complaint that I had that we weren't somehow keeping track of structure in a nice way, is all going to be hidden inside how we represent these probability distributions. That is going to be the structure that moves from time step to time step rather than just the population of points moving from time step to time step.

- $P^{\Theta_t}{}_{(x)}$ is a probability distribution that is uniform over all points that have a fitness value that is greater or equal than Θ. If we generate samples from that distribution we are generating all the points whose fitness is at least as good as Θ and we take from those the set of points that are much higher than Θ and use that to estimate a new distribution. We keep doing that and, since we are constantly taking the best of those, theta will get higher and higher and move to Θ_{min} over time to Θ_{max}. And when we get to Θ_{max} we have converged and we are done.
- Updating the theta and specifically will be tracking the n^{th} percentile (i.e. 50th percentile, 25th percentile). If we take the median (50th percentile) then we are sampling, taking the top half of what's left, we are refitting a distribution to that set of individuals that were drawn and repeating in the sense that now the fitness of those individuals the median should be higher because we are fitting from a more fit collection.
- If there are two things that are true; The first is that we can do the estimate. Given a finite set of data, can we estimate a probability distribution? The second thing that needs to be true is that the probability distribution for

$P^{\Theta} = P^{\Theta+e}$

This means that generating samples from $P^{\Theta_t}{}_{(x)}$ should give samples from $P^{\Theta_{t+1}}{}_{(x)}$
So that means that I have to not only estimate the distribution, I have to do a good enough job to estimate this distribution such that when I generate samples from it I would also generate samples from the next distribution I am looking for.
If those things are true you will be able to move from Θ_{min} and eventually get to Θ_{max}.

11.18. Estimating Distributions



- I've written here the chain rule version of a joint probability distribution. Note, we're dropping the Θ's here for the purpose of just describing this generic distribution

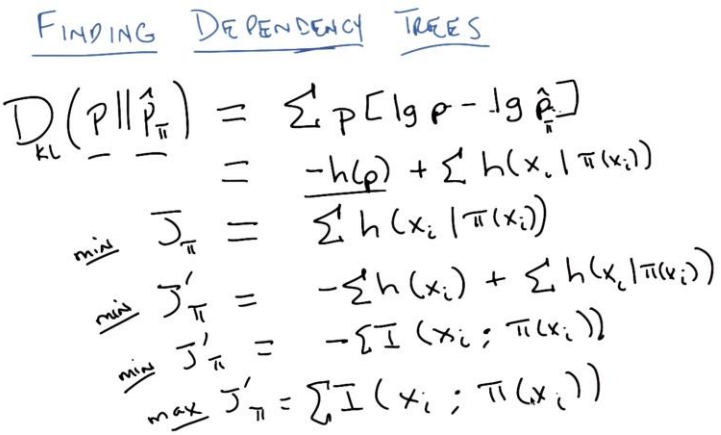
  $$P(x) = P(x_1|x_2 \ldots x_n)P(x_2|x_3 \ldots x_n) \ldots P(x_n)$$
- Just to be clear, what these subscripts mean here, every x that we have is made up of a set of features. Let's just say there are N of these features. And so really, X is a vector of features. There's feature 1 through feature N.

  $$X = [x_1 x_2 \ldots x_n]$$
- What I really want to know for my P^{Θ} is I want to say probability of me seeing all of the features of some particular example is just the joint distribution over all of those features.
- We could just estimate this but that's going to be hard. The first distribution is conditioned on a lot of things, so it's an exponential sized conditional probability table and we need to have an enormous amount of data in order to estimate it well, this is sort of the fundamental problem.
- We have addressed this in the inference lecture where we can try to estimate this incredibly painful joint distribution by making some assumptions about conditional independence. In particular one kind of assumption is that we only care about what's called **Dependency Trees**.
- The Dependency Tree is a special case of a Bayesian network where the network itself is a tree. Every node, every variable in the tree, has exactly one parent. See the red tree in the slide.
  - This should be a directed graph though it's obvious by looking at it that this is the root of the tree.
- So all we have here now is a tree. Every node has one parent, that means that every node, every random variable, depends on exactly one other random variable.
- We can rewrite this joint distribution now as a product over each of the features depending only upon its parent.

  $\hat{p}_{\pi}(x) = \prod \quad p(x_i|\pi(x_i))$ * The first capital $\prod$ means product, the second lower case π means parent.
- When I compare this dependency representation of a distributionion versus the full joint, the main positive is that you're only ever conditioned on at most one parent because the root of the tree doesn't actually have a parent. So just assume that in this case π returns itself or something and so it's the unconditional distribution. And that could happen at any time.
- No one is ever conditioned on more than one other feature and therefore the conditional probability tables stay very, very small.
- If these are binary features then it's two numbers, your probability when your parent is true and your probability when your parent is false.

- Representing the entire thing, the amount of the data that you need is going to turn out that you only need to keep track of quadratic number of features.
- Remember, one of the steps in the algorithm is you have to estimate the distribution, so we're going to have to figure out, of all the trees that I might have, which is the best tree.
- Dependency trees have this nice feature that they actually let you represent relationships. The point is that you're actually able to represent relationships between variables, which in this case are sort of features in our space. But you don't have to worry about too many of them, and the only question here then is which of all the possible relationships you could have where you only are related to one other thing, do you want to have.
- In some sense a dependency tree, since you can depend on at most only one other thing, is the simplest set of relationships you can keep track of. The next simplest would be not having any parents at all, in which case you would be estimating simply $\prod \quad p(x_i)$
    - That's even simpler. But it doesn't, doesn't allow any of the inter-relationships or any of the co-variants essentially information to be captured.
- We could have started with something like $\prod \quad p(x_i)$ except here you're forced to ignore all relationships because you're treating everything as independent. And we don't believe that things are independent or at least we think there a possibility there's some dependence. And so by allowing at most that you're connected to one other parent that's sort of the least committed to the idea you could still be while still allowing you to capture these relationships.
- I want to be strict here that the MIMIC algorithm or just this whole notion of representing probability distributions does not depend upon dependency trees. We're going to use dependency trees here because you have to make some decision about how to represent the probability distributions, and this is kind of the easiest thing to do that still allows you to capture relationships.
- At the very least this will allow us to capture the same kind of relationships that we get from crossover in genetic algorithms. That was a bit of the inspiration for this, that crossover is representing structure, in this case structure that's measured by locality, and this is kind of the general form of that.
- If there is some locality, then however we're going to learn the dependency tree from the samples is going to be able to capture that. Therefore, it will kind of wrap its head around the same kind of information that crossover is exploiting. In fact it can do better than that because it doesn't actually require locality the way that crossover does.
- One other thing that's worth mentioning here about this distribution and why it's nice is it captures relationships. It's very easy to sample from. Given a dependency tree where each one of these features, each one of these nodes, represents a feature, it is very simple, very easy to generate samples consistent with it. You just start at the root, generate a sample, unconditional sample according to whatever the distribution is and then you go through the parents and you do the same thing. So it's exactly a topological sort and in trees topological sorting is very easy and this is in fact, linear in the number of features.
- It is exactly an instance of what we talked about when we did Bayesian inference The thing that is new is how do we figure out dependency tree from the sample.

11.19. Finding Dependency Trees



- I'm going to step through how you find dependency trees and hopefully it's fairly straightforward, at least if you understand information theory. I want to stress that although we're going to spend some time doing this, this is just one example of ways that you could represent a probability distribution. It's just going to turn out that this one is particularly easy and powerful
- The first thing we have to remember, Michael, is that we have some true probability distribution, P, that we're trying to do, P^{Θ} in this case. But the general question of how you represent a dependency tree doesn't depend on Θ or anything else, there's just some underlying distribution we care about, let's call that P.
- We're going to estimate P with another distribution which I'm going to represent as $\hat{P}$, for approximation, that's going to depend upon that parent function that we defined before
  $P||\hat{P}_\pi$
- So, somehow we want to not just find a dependency tree that represents the underlying distribution, but we want to find the best one. Best one here sort of means closest one, or most similar, or the one that would generate the points in the best possible way. It turns out, for those who remember information theory, that there's actually a particular measure for that and it's called the **KL Divergence** (**Kullback Leibler Divergence**)
- Assuming noncontinuous variables, KL Divergence basically has this form:
  $D_{KL}(P||\hat{P}_\pi) = \sum \quad p[lg\ p - lg\ \widehat{p_\pi}\ ]$
  - Basically it's a measure of the divergence, D, between the underlying distribution, P, that we care about and some other candidate distribution, $\hat{P}$, that we're trying to get as close as possible.
- If $\hat{P}$ and $P$are the same distribution, the Kullback–Leibler divergence is equal to zero.
- As they differ, or as they diverge, this number gets larger and larger. Now it turns out that these numbers are unitless, they don't obey the triangle inequality and this is not a distance. This is truly a divergence. But if we can get this number to be minimized, then we know we have found a distribution $\hat{P}$, that is as close as we can get to P
  - Pushkar will have a whole unit to remind everybody about information theory where this comes from. You can pause here if need be to go back and listen to Pushkar's lecture
  - Basically this is the right way to define similarity between probability distributions, you just have to kind of take that on faith.

- What we really want to do is we want to minimize the Kullback–Leibler divergence, that is minimize the difference between the distribution that we're going to estimate with a dependency tree and the true underlying distribution, and just by doing some algebra you end up getting down to what looks like a fairly simple function.

$$D_{KL}(P||\hat{P}_\pi) = \sum \quad p[lg\ p - lg\ \widehat{p_\pi}\ ] = \ -h(p) + \sum \quad h(x_i|\pi(x_i))$$

  - If you were paying close attention to the algebra in Pushkar's lecture, you will realize that $p\ log\ p$ is just entropy. So or it's minus the entropy, so you can rewrite this as simply minus the entropy of the underlying distribution, plus the sum of the conditional entropies, for each of the x_i's, given its parent. Which has some sort of intuitive niceness to it.
  - This is what you end up with just by doing the substitution. $p\ log\ p$ gives you minus entropy of $p\ -p\ log\ \widehat{p}$, which gives you the conditional entropy according to the function, the parent function π.
- In the end all we care about is finding the best π. So, $-h(p)$ doesn't matter at all and so we end up with a kind of cost function that we would like to minimize, called J here, which depends upon π, which is just the sum of all the conditional entropies.

  $min\ J_\pi = \sum \quad h(x_i|\pi(x_i))$
- So basically the best tree that we can find will be the one that minimizes all of the entropy for each of the features, given its parents, because we want to choose parents that are going to give us a lot of information about the values of the corresponding features.
- In order for $\sum \quad h(x_i|\pi(x_i)$ to minimized you would have to have picked a parent that tells you a lot about yourself, because entropy is information, entropy is randomness, and if I pick a really good parent then knowing something about the parent tells me something about me and my entropy will be low.
- So if I can find the set of parents for each of my features such that I get the most out of knowing the value of those parents then I will have the lowest sum of conditional entropies.
- Now, this is very nice and you would think we'd be done except it's not entirely clear how you would go about computing $\sum \quad h(x_i|\pi(x_i)$ (actually we do but it's excruciatingly painful).
- It turns out that there's a cute little trick that you can do to make it less excruciatingly painful to compute this. First define a slightly different version of this function.

  $min\ J'_\pi = \ -\sum \quad h(x_i) + \sum \quad h(x_i|\pi(x_i))$
  - We want to minimize this particular cost function, J, which we get directly from the Kullback–Leibler divergence. So all I've done is define a new function J', where I've added this term - $-\sum \quad h(x_i)$
    - Just minus the sum of all of the unconditional entropies of each of the features.
  - Now I'm able to do this because nothing in $-\sum \quad h(x_i)$ depends upon π and so doesn't actually change the proper π.
- Minimizing $\sum \quad h(x_i|\pi(x_i))$versus minimizing $-\sum \quad h(x_i) + \sum \quad h(x_i|\pi(x_i))$ should give the same π, it's sort of like adding a constant if you've got a max. It doesn't change which element gives you the max.
- By adding this term we've actually come up with something kind of cute. This expression should look kind of familiar from Information Theory. It's the negative of mutual information.

  $min\ J'_\pi = \ -\Sigma\ I(x_i; \pi(x_i))$

- So, minimizing this expression, is the same thing as maximizing mutual information.

$$max\ J'_\pi = \Sigma\ I(x_i; \pi(x_i))$$

- This is going to induce a very simple algorithm for figuring out how to find a dependency tree. And the trick here is to realize that, conditional entropies are directional (conditional entropies are the $h(x_i|\pi(x_i)$from above).
  - x_i depends upon $\pi(x_i)$ → meaning it's directional
    - If you were to do $h(\pi(x_i)|x_i)$I would be getting a completely different number.
  - Mutual information on the other hand is bi-directional. It's going to turn out to be easier to think about.
- But before we do that let's just make certain that this makes sense:
  - We wanted to minimize the Kullback–Leibler divergence

$$D_{KL}(P||\hat{P}_\pi) = \sum \quad p[lg\ p - lg\ \widehat{p_\pi}\,]$$

  - We work it all out and it turns out that we really want to minimize the cost function, another way of rewriting this of conditional entropies.

$$min\ J_\pi = \sum \quad h(x_i|\pi(x_i))$$

  - We threw this little trick in, which just allows us to turn those conditional interviews into mutual information.

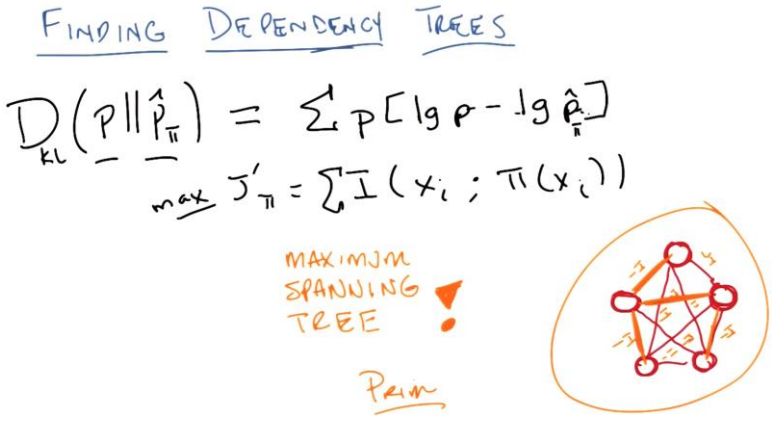
$$min\ J'_\pi = -\sum \quad h(x_i) + \sum \quad h(x_i|\pi(x_i)) = -\Sigma\ I(x_i; \pi(x_i))$$

  - And what this basically says is that to find the best π- the best parents, the best dependency tree - means you should maximize the sum of the mutual information between every feature and its parent.

$$max\ J'_\pi = \Sigma\ I(x_i; \pi(x_i))$$

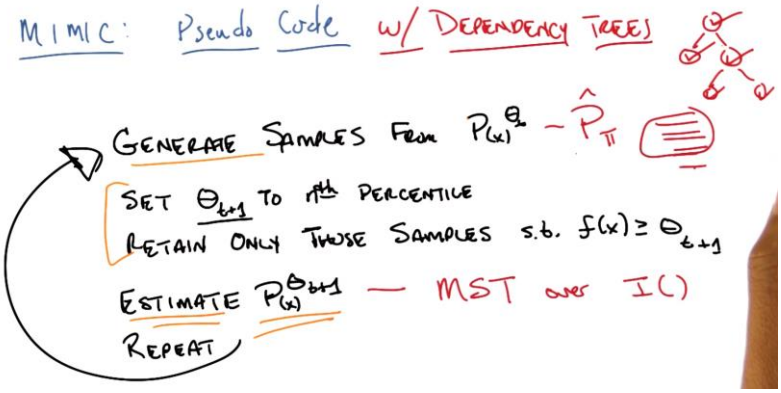
    - That should make sense since you want be associated with the parent that gives the most information about you.
- Just to be clear the Kullback–Leibler divergence is a summation over all possible variables in the distribution, and so all we've done here is carry that all the way through. This new cost function that we're trying to minimize is a sum over the negative mutual information between every variable, every feature, and its parents. And that's the same thing as trying to maximize the sum of the mutual informations between every feature and its parent.
- Basically the best dependency tree is the one that captures dependencies the best. Alright, so now we need to figure out how to do that optimization. And it turns out it's gloriously easy.

11.20. Finding Dependency Trees Three



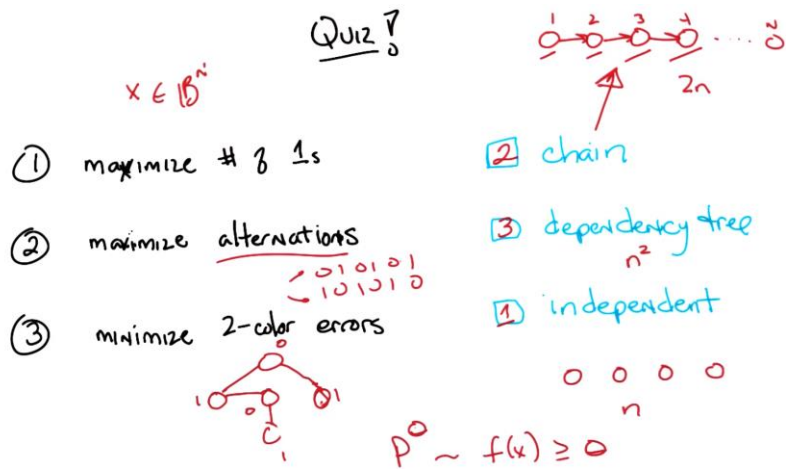
- We're trying to maximize this cost function $max\ J'_\pi$ which is basically the sum of the mutual information between every single feature and its parents. And I want to point out that that actually induces a graph. In fact a fully connected graph.
- In the slide is drawn a bunch of nodes as part of a graph and all the edges between them. Each of these nodes is going to represent a feature, an x_i, and what's going to go on the edges is going to be the mutual information between them.
- So, I now have a fully connected graph, which has the mutual information between each pair of nodes, all in squared edges, and I want to find a subgraph of this graph, in particular a tree, that maximizes these sums. 
- So each time we pick an edge it's actually saying that what we're going to care about the relationship between this pair of variables. And we're only allowing edges, because that involves only 2 nodes, so we're not getting sets of 3 or 5 or however many nodes larger than that.
- So we want to be considering a subset of edges that form a tree that has the highest total information content. That's a **maximum spanning tree**.
- It turns out that finding the tree consistent with this graph, such that $\Sigma\ I(x_i; \pi(x_i))$ is true is in fact the same thing as finding the maximum spanning tree, which we all vaguely remember from our algorithms class.
- So, that's really neat, we've turned the problem of finding the best distribution, and in particular the best dependency tree - independent of the true underlying distribution I want to point out - into a problem of a well-known, well understood computer science problem: finding the maximum spanning tree. If we find the maximum spanning tree, then we have found the best dependency tree.
- And what you'll do when you find the maximum spanning tree is you'll end up with some tree, and you're done.
- I want to point out there's 2 different algorithms that you could use to find the maximum spanning tree:
- Prim and Kruskal. Those are the two that they teach you in algorithms class, but there are others.
- It turns out, for this particular case, you want to use Prim because Prim is the proper algorithm to use whenever you have a densely connected graph, it just happens to be faster.
    - Since this is a fully connected graph it is about as dense a connected graph as you can get.
- So, Prim's algorithm runs in time quadratic, or polynomial anyway in the number of edges, so it's actually fairly efficient as things go
- So use Prim's algorithm, you find the Maximum Spanning Tree, and you are done. This was a bit of a diversion that we had to do, let's go back to the original algorithm and just point out what we would be doing here

11.21. Back to Pseudo Code



- Back to the pseudo code that we have for MIMIC and see if we can plug in some of the stuff that we we just talked about just as a way of refreshing.
- The goal is to generate samples from some $P^{\theta_t}(x)$. So in this case we're just going to estimate that by finding the best dependency tree we can, that is the best π function, $\widehat{P_\pi}$.
- So we're going to be starting with some dependency tree And we're going to generate samples from that, and we know how to generate samples from that given a dependency tree like say the tree in the slide.
    - We simply start at a node, generate according to its unconditional distribution and then generate samples from each according to its conditional distribution given its parents.
- In order to compute the mutual information, that is in order to compute the entropies, we have to know the probabilities. So, we generate all of these samples, and that tells us, for every single feature x_i, how often that was taking on a value of one or taking on a value for zero (if we assume everything's binary). So that means we have unconditional probability distributions. Or at least estimates of unconditional probability distributions, for every single one of our features
- At the same time, because we have these samples, we know for every pair of features the probability that one took on a value, given a value for the other, already have the conditional probability table for each of those as well.
- Just the act of generating these samples and building that mutual information graph gives us all the conditional and unconditional probability tables that we need.
- Given that, we can generate samples in time linear and the number of features, and we're done.
- Now we know how to generate samples from some tree that we have and we generate a bunch of these samples, find the best ones, retaining only those samples, and now we know how to estimate the next one by simply doing the maximum spanning tree over all the mutual information.
- And then we just repeat and we keep doing it.
- And there you go. A particular version of MIMIC with dependency trees. Again, just to really drive this point home, you don't have to use dependency trees. You can use unconditional probability distributions, which are also very easy to sample from and very easy to estimate. You could come up with more complicated things, if you wanted to, like try to find the best Bayesian network. You can do all of these other kinds of things and that'll work just fine.
- Dependency Trees though are very powerful, because they do allow you to capture these relationships, that is to say they give you a probability distribution that has structure that we were looking for, while not having to pay an exponential cost for doing the estimation

11.22. Probability Distribution Quiz



- This quiz is less about the details of mimic itself than it is about the probability distribution.
- As discussed, MIMIC doesn't care about your probability distribution, you should just pick the best one.
- We're going to assume in all of these cases, that our input value is x binary strings of length N.
- The first problem we want to maximize the number of 1s that appear in that sample, you're fitness function is maximize the number of 1's
- In the second problem, we want you to maximize the number of alternations between bits.
- The third problem is you want to minimize two color errors in a graph. This one is a little bit harder to describe, so see the graph in the slide. The two color problem is given a graph with some edges and you want to assign a color to each node such that every node has a different color than its neighbor. We assume there's only two colors here.
- Now here are the three distributions. The first distribution is a chain. So a chain would be in kind of Bayesian network speak, a chain would be a graph, where basically every feature depends on its previous neighbor. So, in a four-bit string, I'm saying that the first bit depends on nothing, the second bit depends on the value of the first bit, the third bit depends on the value of the second bit and the fourth bit depends on the value of the third bit. In fact, it's not just a chain. It's a specific chain, it's the same chain as the ordering of the bits.
- The second one is what we've been talking all about along. It's a dependency tree. Unlike in the case with the chain above where I am giving you a specific chain ordered by bits, I don't know which is the dependency tree is or you have to find it but there is some dependency tree I want you to represent.
- And the third one is the easiest to think about and that's where everything is independent of everything else. It would be a bunch of nodes with no edges between them, directed or otherwise. So the joint probability across all your features is just a product of the unconditional probabilities. So it's the simplest probability distribution you can have.
- Each of these is representable by a dependency tree. So in the independent case, since we know that it's independent, or at least we're imagining it's independent, we just have to estimate one probability per node, which is like N. In the chain case, we have a conditional probability per node. So it's, like $2N$ parameters. And in the dependency tree case, we're estimating N^2 parameters and then pulling the tree out of that.

- A chain is a dependency tree, independents are a dependency tree, and a dependency tree is surprisingly enough a dependency tree. But these numbers and parameters do matter because although a dependency tree can represent an independent set of variables, the way you're going to discover that the are independent is that you're going to need a lot of data to estimate those in square parameters in order to realize that the conditional probability tables effectively don't mean anything. So, it will be very easy for a dependency tree to overfit in the case where all the variables or all the features are independent.
- I'm asking you to figure out which distribution will represent the optima, the structure between the optimal values.

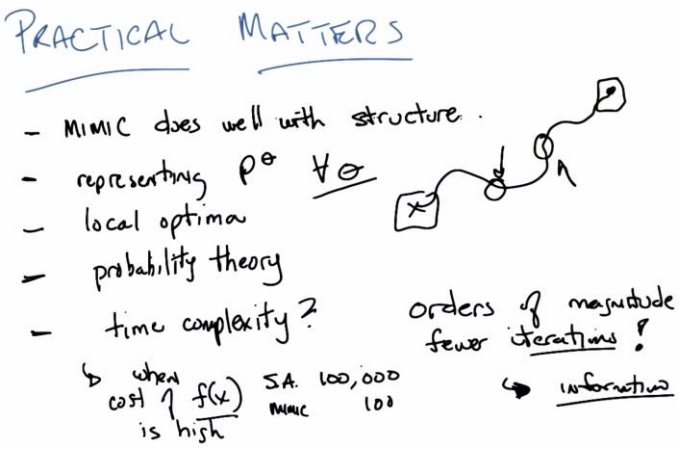
Answer

- So problem one, where you're maximizing the number of 1s, to represent the optima here, or the influence of any given bit on the fitness value, they're all independent. They all just contribute whatever they contribute. And so I don't see any reason to capture other dependencies. Local information is enough.
- For number 3, think about it this way: Remember, the probability distribution that we care about, P^{Θ}, is basically uniform for all x's such that the fitness value is greater than or equal to Θ.
  $P^{\Theta} \approx f(x) \geq \Theta$
- Imagine we start out with Θ equal to 0, which is the lowest value that you could get, then the probability of bit 1 being a 1 is ½ because every single value is okay. So in order to get a uniform distribution across N bits, I can sample each bit independently, uniformly, and that will give me an overall uniform distribution.
- Now, at the very end, when I have all 1s as the maximum value, that's also easy to represent. Because the probability of bit 1 being 1 for the optimum value is 1, the probability of bit 2 is 1, etc. And they all can be independently represented. So we can definitely represent the minimum Θvalue and the maximum Θvalue, these extremes, the question here is, can this distribution represent values of Θin between?
- So let's say we had 4 bits (in the slide) and imagine $\Theta$was 2. Notice that when $\Theta$is 2, it's not just the number of points that will give you exactly 2, it's the ones that will give you at least 2 (2, 3, and 4)
  - So how many different ways are there to get 4? Well, there's only one way to get a value of 4, and that is all 1s.
  - How many different ways can you do 3? Well, there's 4 ways to get 3. You basically have to choose the one that you give as 0, right? And each one of those bits, each one of these values will be a 1, three quarters of the time.
  - And how many different ways can you get 2? 6, so you can actually write all of those out and count the number of times that each one is a 1.
- And those are all your samples, and you just simply estimate it. You'll end up with a uniform distribution which will be consistent with the examples that I just sampled from, but will probably not exactly capture P^{Θ} because it will sometimes be able to generate with probability greater than 0, say, a value of all 0's. So this is just an approximation, even in the simple case.
- We know that the extreme values can be represented by an independent distribution, but it's not clear that all the values in between can be represented by such a simple distribution. But that's always going to be the case. What you want is a distribution that will definitely capture

the optimum and gives you a good chance of generating samples that get you closer to the optimum along the way.

- So even though in the case before where $\Theta$is 2, we might still generate examples where you get a fitness of 0 or 1, you have a high probability if you generate enough samples of actually getting values of Θ greater than 2, 3, or 4, even.
- For problem 2, maximize the number of alternations, we pointed out that it's really important to, to know who your, what your neighbor is, because you want to be a different value than your neighbor. And the chain gives you exactly that information without anything extra. So I would put the 2 in the first box.
- There's only one box left, so I'd put the 3 in the middle box.
- The interesting thing to look at here is that it is surprisingly appropriate. The coloring problem, it is specifically saying my value depends on several of my neighbors, not just one.
- However, you could capture a lot of the necessary information by finding a good dependency tree.
- It turns out that in practice MIMIC does very well on problems like this. Even where you have really complicated graph structures. It tends to do much better than a lot of the other randomized optimization problems. And that's because what we've got here in this graph is structure. And what mimic is trying to represent, is in fact, structure.
- In fact, in all of these cases, well, except for the, the first case, there's not a single answer often. There are possibly many answers, but the answers all have in common their relationship to some underlying structure. So in the maximizing alternations case, you have 010101 or 101010. These are two completely different values. In fact, as you pointed out when I asked you this earlier, they're complimentary. But, each one has a very simple structure, which is, every bit is different from its neighbor to the i, actually both of its neighbors. Every bit is different from its neighbors. And that doesn't matter what their values are. Given the value of one of them, it actually completely determines the value of everything else.
- So MIMIC doesn't get lost trying to bounce back and forth between this possible maximum and this possible maximum. Instead, it represents both of them at the same time. Because the only thing that matters is the relationships.

11.23. Practical Matters



- MIMIC does well with structure. When the optimal values that you care about depend only on the structure, as opposed to specific values, MIMIC tends to do pretty well. By contrast

Randomized hill climbing, genetic algorithms, these other things that we've looked at before can sometimes get confused by two different values that are both optima but look very different from one another. Where it's the structure that matters and not the actual values.

- The chain example before we had be where you had alternating values as one of those cases where it's easy for randomized algorithms that only look for point values to get confused because they're basically being drawn in multiple directions at once.
- The quiz also brought up another point and that is it's an issue of representing everything, that is it's not enough just to be able to represent a probability distribution of the optima, you really want to be able to represent everything in between as you move through probability space toward your answer. You don't just want to represent at the end and at the beginning, which is pretty easy because uniform distribution, but can you represent a point in between? If you can't are you going to end up getting stuck, and actually turns out that MIMIC can get stuck in local optimal, though it typically does not, in part because you get randomized restarts for free.
- But the problem of local optima is still a problem of local optima.
- Now, when I say something like you get randomized restarts for free, I'm actually cheating a little bit and hiding something which is a little bit more important, which is what you really get for free is probability theory. So there's a hundred, literally, hundreds of years of work on how to think about representing probability distributions and what you can do with them and there are terms like 'importance sampling' and 'rejection sampling' and all these kinds of tools that we have for representing probability distributions that you can actually inherit with something like MIMIC for dealing with these painful cases where you might not be able to represent distributions.
- But the single most important thing about Mimic, or what to get out of here, is that representing structure does matter, but you pay a price. That price basically boils down to *time*. So the question we might ask ourselves is what is the sort of practical time complexity of MIMIC? And, it really boils down to something very simple.
    - I have run this algorithm on many, many examples and I've compared it to simulated annealing, compared it to genetic algorithms, compared it to randomized hill climbing. It works pretty well for the sorts of examples I've come up with.
    - MIMIC tends to run orders of magnitude fewer iterations. And I'm not exaggerating here: I mean that if I run Simulated Annealing it might take 100,000 iterations, but for MIMIC, it might take only 100. And this is consistently true.
    - However, it turns out that that's not good enough. It turns out that the fact that MIMIC can do something in three, four, five, six, seven orders of magnitude fewer iterations isn't an argument for always using it since different algorithms can take different times for a single iteration. MIMIC is slower per iteration.
        - Simulated annealing just does this little tiny step, it computes a bunch of neighbors and then does a probability comparison, and then takes a step.
        - MIMIC is drawing from a distribution, computing which things are say above the median performance, and then it's re-estimating a new distribution. And then that's the end of the iteration. Depending on how many samples it takes to do that, it could take a very long time, and in particular, it's going to always be a lot more samples than what simulated annealing is doing.
    - So, when would MIMIC still be worth using where we know that we can get to the answer but simulated annealing will take orders of magnitude more iterations and MIMIC will take

fewer iterations? When would it still be worth it to take the one with fewer iterations even though each iteration is expensive? What is MIMIC giving you for all of that work that it's doing in building dependency trees and running Prim's algorithm and finding maximum spanning trees. STRUCTURE!

- You get structure. Another way of thinking about structure in this case is you're getting information. You get a lot more information because you get structure, you get a lot more information per iteration as well. So, that's the price you're paying, you're getting more information every single time you do an iteration, at the cost of building this maximum spanning trees and everything it is you're doing in estimating your probability distribution.

- So, why would it be worth it to do that? What's the other source of expense?
  - There's all this computation within an iteration but what it's trying to do is to find inputs that have high scores and so you do have to compute the scores for all those inputs. So the fitness calculation is very important.
  - ***So MIMIC tends to work very well when the cost of evaluating your fitness function is high***. So, it's really important that I only have to take 100 iterations if every single time I look at a fitness function and try to compute it for some particular x, I pay some huge cost in time.

- How many function evaluations there are in an iteration? You get one, basically for every sample you generate.
- What about when comparing samples? Would depend upon how many samples you feel like you need to generate, and of course you can be very clever because, remember, Θ will generate a bunch of samples for $\Theta + 1$. So, if you keep track of the values you've seen before, you don't have to recompute them. So it's actually pretty hard to know exactly what that's going to be.
  - Let's imagine that at every iteration, you generate 100 samples.
  - So, at most, you're going to have to evaluate $f(x)$ 100 times.
  - So, MIMIC is still a win over something else if the number of iterations that something else takes is more than 100 times fewer.
- Any real fitness functions that might actually be expensive to compute?
  - Yes, a lot of stuff that is important is like that. If you're trying to design a rocket ship or something like that the fitness evaluation is doing a detailed simulation of how it performs. And that could be a very costly thing.
  - Also, MIMIC has been used for things like antenna design, designing exactly where you would put a rocket in order to minimize fuel cost sending it to the moon, these sorts of things where the cost really isn't evaluating some particular configuration where you have to run a simulation or you have to compute a huge number of values of equations and so on and so forth to figure out the answer.
  - Another case where it comes up a lot is where your fitness function involves human beings. Where you generate something, and you ask a human, how does this look, or does this do what you want it to do, because humans, as it turns out, are really slow.
- So, you end up with cases where fitness functions are expensive, something like this becomes a big win.
- You are looking at all of the different algorithms, at different models or at everything that we have been doing, both for unsupervised learning and earlier for supervisor learning. A lot of times, your trade-off is not just in terms of whether you are going to overfit or not, it's whether

it's worth the price you need to pay in terms of either space or time complexity to go with one model versus the other.

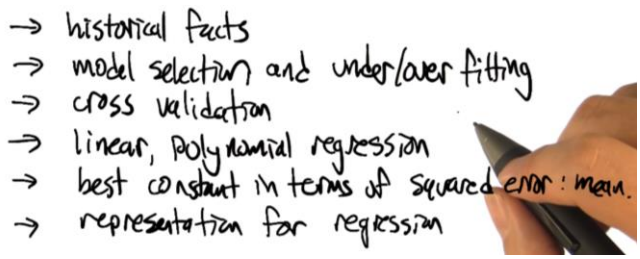
- We've been talking about it in terms of sample complexity, but there's still time complexity and space complexity to worry about.

12. Midterm Summaries - All "What We've Learned" Slides

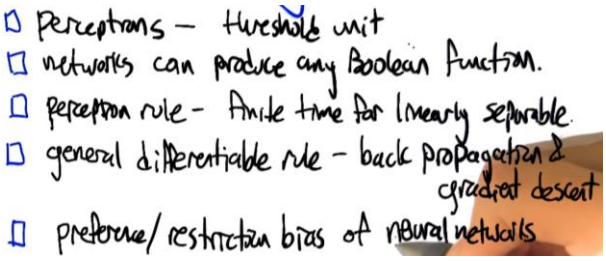
12.1. Decision Trees

- REPRESENTATION
- ID3: A TOP DOWN LEARNING ALGORITHM
- EXPRESSIVENESS OF DTs
- BIAS OF ID3
- "BEST" ATTRIBUTES (GAIN (S, A))
- DEALING WITH OVERFITTING

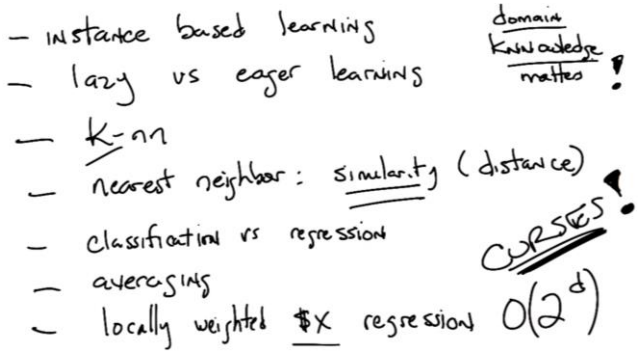
12.2. Regression and Classification



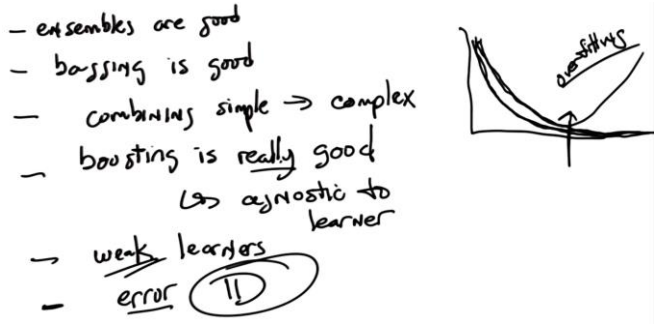
12.3. Neural Networks



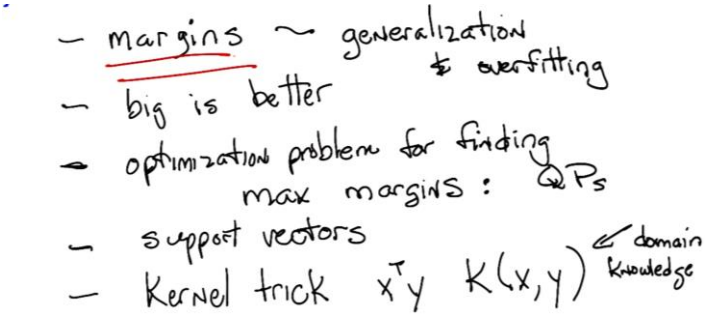
12.4. Instance-Based Learning



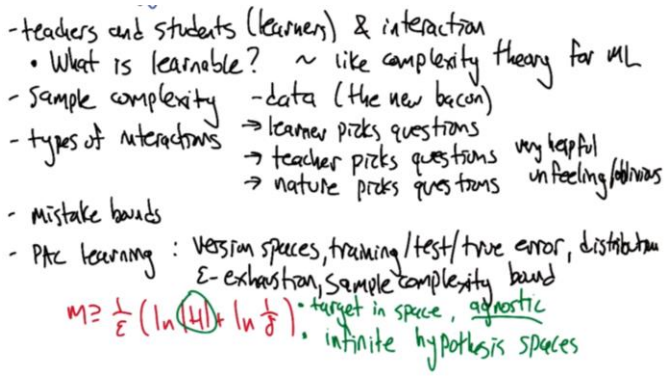
12.5. Ensemble B&B - Boosting



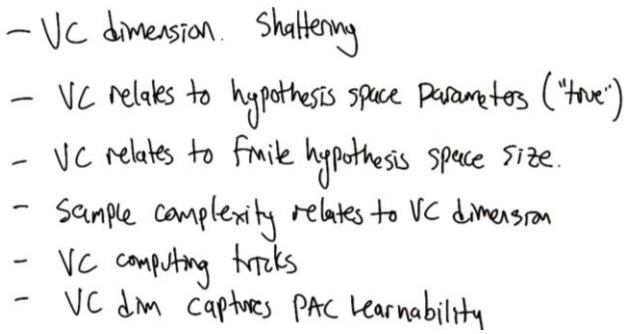
12.6. Kernel Methods and SVM's



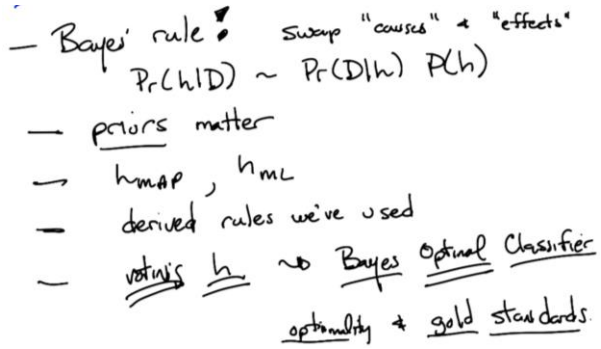
12.7. Computational Learning Theory



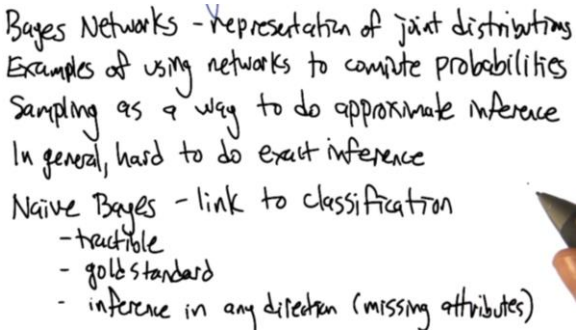
12.8. VC Dimensions



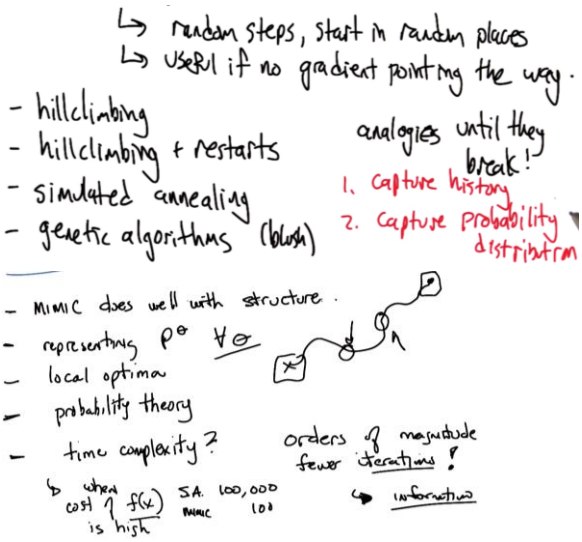
12.9. Bayesian Learning



12.10. Bayesian Inference

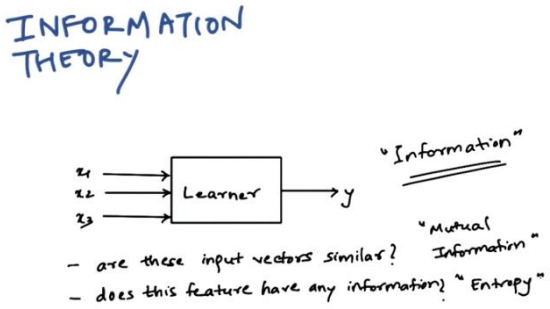


12.11. Randomized Optimization



12.12. Information Theory

First we'll try to understand where Information Theory is used in motion learning. So consider this to be any machine learning algorithm.



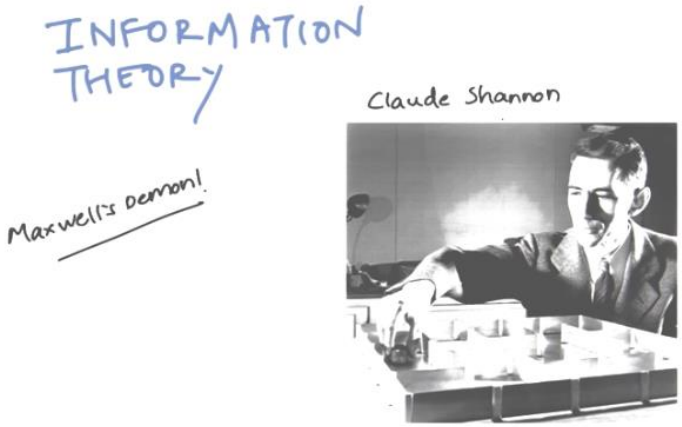
For example, let this learner be, or a decision learner, we have several inputs, x1, x2, x3, and one output. For simplification, let's assume that this is a regression problem. That's why we have one output.
We want to ask interesting questions like how is x1 related to y, x2 related to y, x3 related to y.

Why do you want to ask such questions?
If you remember, from our IDT algorithm, the first step is to find out which input best splits our output. So we need to find out which of these, x1, x2, or x3 gives you the most information about y. So we have to first understand what the word information in information theory means.

In general every input vector and output vector, in the machine learning context, can be considered as a probability density function. So, information theory is a mathematical framework which allows us to compare these density functions, so that we can ask interesting questions like are these input vectors similar?

If they're not similar, then how different they are? And so on. We call this measure as mutual information.

Or we could ask if this feature has any information at all. So we'll call this measure entropy. So we are going to find out what these terms mean, how they're related to information learning in general, and we'll briefly look at the history of this field.



Information theory has also a background in phys.

12.13. Sending a Message

Let's assume that you want to send a message from Atlanta to San Francisco. And to make it easier, let's assume that we want to send a simple message which consists of n coin flips, or the output of ten coin flips.
Let us construct two messages out of coin flips.

Now, I have two coins, but you see these coins are different because this one has a heads and a tails, it has two different sides, And this one (pointing to the unfair coin) has both the sides which are very similar looking. So it's a biased coin so every time I flip its going have the same state. While when I flip this (pointing to the fair coin) it might either end up here, 50% of the time or end up here 50% of the time.

So we'll construct two messages after flipping both of them and recording what their state is. So here it is, I did ten coin flips with the fair coin I got a few heads, a few tails, in this particular sequence. The unfair coin, I'm calling every state as a head state and I basically saw ten heads.

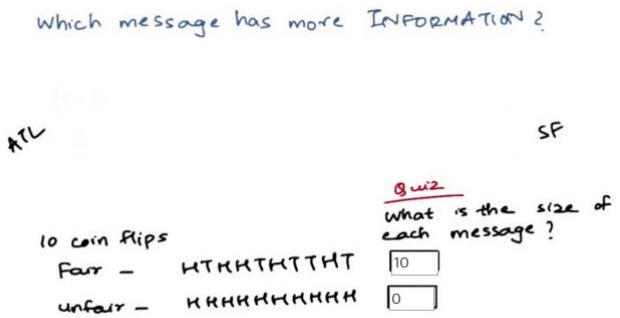
Fair - HTHHTHTTHT
Unfair HHHHHHHHHH

If you also observe the fair coin I have like five heads and five tails. I got five heads and five tails, so it is a fair coin.

So, if I had to transmit this sequence (pointing to the Fair coin sequence), how many bits of message will I require?

Let's assume that I can represent this sequence using ten binary digits. A zero representing heads, one representing tails and I can write down this sequence as zeros and ones using ten bits. I can also write down the same sequence with the, of the unfair coin using those ten binary digits. So I'll get something like zero, one, zero, zero, one, zero, one.

And here, everything will be zeros. Let's assume I have to transmit these two particular sequences from Atlanta to San Francisco. What will be the size of each message in case of the fair coin and the unfair coin? What do you think?



In the unfair coin case, will already know the outcome of that test.

- if the output of this coin is predictable, you don't need to communicate anything. If the sequence is predictable or it has less uncertainty, then it has less information.
- But if the output is random, you need the communicate the result of each and every flip. So more information has to be translated.

Shannon described this measure as entropy. He said, if you had to predict the next symbol in a sequence, what is the minimum number of yes or no questions you would expect to ask.

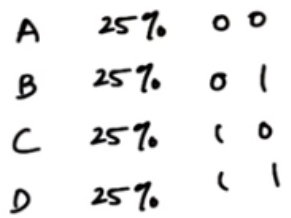
In the first example, you have to ask a yes or no question for every coin flip. So you have to ask at least one question for every flip.

In the unfair coin, you don't have to ask any questions. So the information in the second case is zero, while the information in the first case is one. Let's consider another example to understand this better.

12.14. Sending a New Message

Let's consider that we want to transmit a message which is made up of four words, A, B, C and D. And let's assume that all the four letters are used equally in the language. The frequency of each letter occurring in the language is equal.

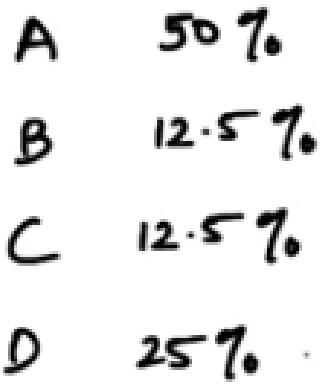
So, you can be present A, B, C and D in binary, with two bits itch, each, like so.



Which means if we have a sequence such as zero, one, zero, zero, one, one, the six bits spell out the word BAD, bad.
So basically we'd require two bits or symbol.

Other way to look at this sequence is that you need to ask two questions to this sequence to at least recognize one symbol. So, two bits per symbol also means that you have to ask two yes or no questions per symbol.

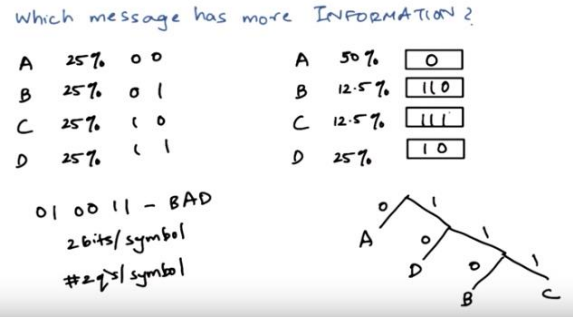
Now let's consider the second message to be made up of the same symbols but in this case A occurs more frequently than B, C or D.



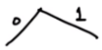
D of course more frequently than B and C.

And, let's assume that these are the probabilities by which we can see A, B, C, or D.

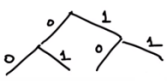
Now, we can do the same thing again, we can use the same binary representation to represent A, B, C, and D. So again, we'll end up with two bits per symbol. But can we do any better?
Well, A occurs more frequently than the others so can we somehow use this to our benefit and use a different bit representation to get slightly less than two bits per symbol? Think about it. Can you think of a new representation that might be better?



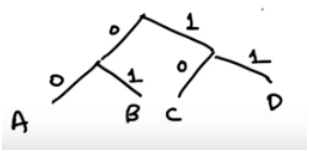
Okay, so the way you can think about this question is by looking at the first message and why it makes sense to have two bits per symbol.
So you can represent this bit pattern in a tree.

So when a new bit comes in, it can be either 0 or a 1. 

If it's a 0, the next symbol can be another 0 or it can be 1. 

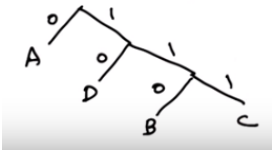
The same case here 

so if there are two 0s, it is an A, if it is a 0 followed by 1, it is a B, if it is a 1 followed by 0, it is a C, if it is a 1 followed by 1, it is a D.



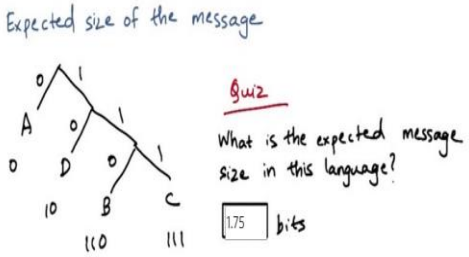
So that is why you need to ask two questions to reach either of these symbols. What happens in this new language? So in the this new language, A occurs 50% of the time.

We can directly ask if it is A or not A.
So let's represent that has 0 or 1, and let A be 0.
So now we got our A as just a 0.
Now if we go on the 1 branch we can again ask, if it was a 0 or a 1.
Now observe that D occurs twice as frequently as B or C.
So, in this case we can represent D here using 1, 0 and then B or C can occur on this branch but both cannot occur at the same place, so we need to differentiate between them using another symbol.
So let's do that using 0 and 1 again. So this can be B and this can be C.
So B is basically 1, 1, 0 and C will be 1, 1, 1.



Now have we actually saved any bits per symbol? Have we saved the number of questions we asked?
Yes because A occurs more frequently and I needed to ask only one question.
But what is the exact number of questions we are to ask for symbols?

12.15. Expected Size of the Message



To work this out we will need to know the frequency of A, B, C, and D.

A 50%
B 12.5%
C 12.5%
D 25%

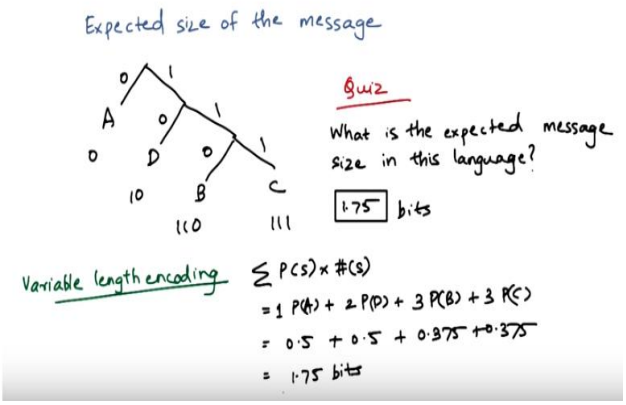
We already know that.

We will also need to know how many bits each of those symbols require. We also know that.

We will then calculate the expected number of bits to transmit each symbol and then add them up.

So for any symbol, the expected number of bits is given by the probability of seeing that symbol, and the size required to transmit that symbol. And we add them up for all the symbols in the language.

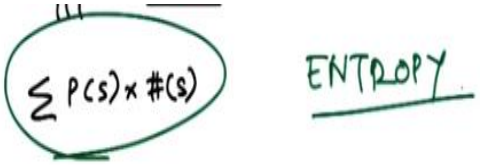
This is going to give us 1.75 bits on an average.
Since we had to ask less questions in this language, than the previous language, this language has less information. This is also called as variable length encoding.



This should give you some idea into figuring out why some symbols in Morse code are smaller than others.

In the English alphabet, the letters e and t occur most frequently. That's why, in the Morse code, e is generated by a dot and t is generated by a dash. Since e and t occur more frequently, they have the smallest message size.

This measure, which calculates the number of bits per symbol, is also called entropy.



And it is mathematically given as this formula.

To make it more legible, we need to find out how to denote the size of s more properly. The size of s is also given by the log of 1 upon the probability of that symbol.

$$= \sum P(s) \log \frac{1}{P(s)}$$

$$= -\sum P(s) \log P(s)$$

So the formula of entropy is given as this.

12.16. Information Between Two Variables

Now we know what is the information in one random variable. Now assume that I told you to predict if you're going to hear thunder or not. Well, that's very difficult. But what if I tell you if it is raining or not.

Your guess regarding the thunder is going to be significantly better.

So there is some information in this variable (rain) that tells you something about this variable (Thunder).

We can measure that in two different ways. The first one is called as joint entropy. Joint entropy is the randomness contained in two variables together as given by H(x,y). It is given by this particular formula, where P(x,y) is the joint probability of X and Y.

$$H(X,Y) = -\sum P(x,y) \log P(x,y)$$

The other measure is called conditional entropy.
Conditional entropy is a measure of the randomness of one variable given the other variable. And it is generated by H of Y given X (H(y|x)).

$$H(Y|X) = -\sum P(x,y) \log P(Y|X)$$

To understand these two concepts, you have to imagine what happens when X and Y are independent.

If X and Y are independent, then the conditional probability of Y given X is just the conditional probability of Y. It's quite obvious, right. If two variables are independent of each other, Y variable doesn't get any information from X at all. The joint entropy between X and Y, if X and Y are independent, is the sum of information of both X and Y. That is why the entropies have been added here.

$$H(Y|X) = -\sum P(x,y) \log P(Y|X)$$

If $X \perp Y$, $H(Y|X) = H(Y)$

$$H(X,Y) = H(X) + H(Y)$$

12.17. Mutual Information

Although conditional entropy can tell us when two variables are completely independent, it is not an adequate measure of dependence.
Now consider the conditional entropy of y given the variable x. This conditional entropy may be small if x tells us a great deal about y or that x of y is very small to begin with.

So we need another measure of dependence to measure the relationship between x and y and we call that mutual information. It is denoted by the symbol I.
It is given as

$$I(x,y) = H(y) - H(x|y)$$

So mutual information is a measure of the reduction of randomness of a variable given knowledge of some other variable.

If you like to understand the derivations for these particular identities, I'll refer you to Charles's notes on, on this topic. But we'll jump directly into an example and try to calculate these values and understand what it means to have a high value of mutual information or low value of mutual information. So let's do that as a quiz.

12.18. Two Independent Coins

Quiz

2 independent coins

$P(A) = P(B) = 0.5$

$P(A, B) = 0.25$

$P(A|B) = P(A) = 0.5$

$H(A) = 1$

$H(B) = 1$

$H(A, B) = 2$

$H(A/B) = 1$

$I(A, B) =$

$$H(A) = -\sum P(A) \log P(A)$$
$$= -0.5 \log 0.5 - 0.5 \log 0.5$$
$$= 1$$

$$H(A, B) = -\sum P(A, B) \log P(A, B)$$
$$= -4 (0.25 \log 0.25)$$
$$= 2$$

$$H(A|B) = -\sum P(A, B) \log P(A|B)$$
$$= -4 (0.25 \log 0.5)$$
$$= 1$$

$$I(A, B) = H(A) - H(A|B)$$

Let's just simply try and substitute our values in the formulas that we know of.
Since A and B are independent events, the triangle probably is given by the product of A, product of probability of A and B, so that gives us 0.25.

Probability of A given B. Since A and B are independent of each other. Probability of A given B is just probability of A, which is 0.5.

So then the entropy of A is given by this formula.

$$H(A) = -\sum P(A) \log P(A)$$

If we expand on this we get, we get the entropy of A as 1.

Similarly the entropy of B is also 1.

What is the joint entropy? The joint entropy is given as this formula. $H(A,B) = -\sum P(A,B) \log P(A,B)$

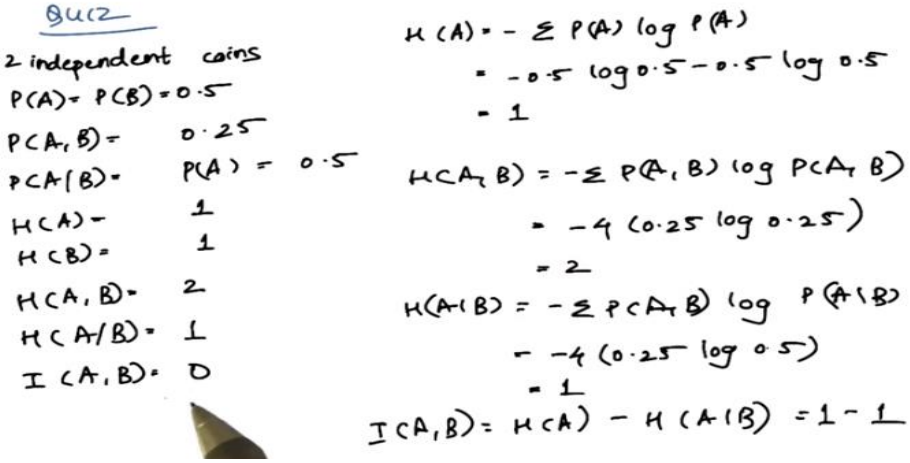
So if we substitute the values we get the joint entropy of A and B as 2.
What is the condition entropy of A given B? It is given as this formula.

$$H(A|B) = -\sum P(A,B) \log P(A|B)$$

If you substitute the values, we get the conditional entropy as 1.

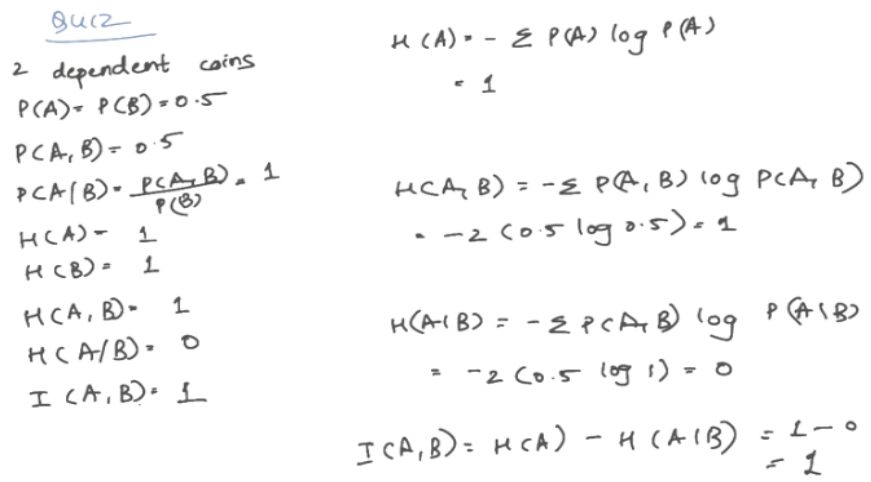
Mutual information between A and B is given by this formula. $I(A,B) = H(A) - H(A|B)$

If we substitute the values of the variables that we have already calculated, entropy of A and entropy of A given B, we get 1 minus 1, which is zero.
So, since the two coins are independent, there is no mutual information between them.



12.19. Two Dependent Coins

Let's do another quiz, where the two coins are dependent on each other. So let's assume a case where you flip two coins, A and B and there's some gravitational force or some kind of weird force acting between them. So, whatever the A flips as, if the A flips as heads, B also turns out to be head. And if A flips as tails, B also comes out to be tails. So complete information is transferred from A and B, and so they're completely dependent on each other. So, find out similarly, what is the joint probability between A, B, conditional probability, their entropies and the conditional entropies and their mutual information. Go.



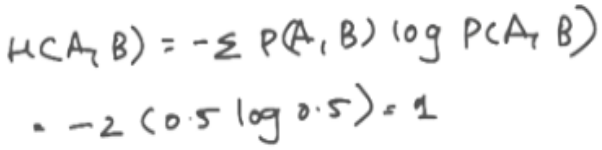
So, let's start with the joint probability.

Since A and B are both dependent on each other, there are only two possibilities.
Both can be heads or both can be tail, tails. So the joint probability is also 0.5.

What is the, what is the conditional probability? Conditional probability is given as probability of A comma B upon probability of B. So, conditional probability is 1.

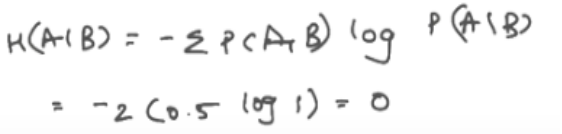
What is the entropy of A? It is similar to the last example, because we are still using fair coins.
So, the entropy of A is 1.
Entropy of B is also 1.

What is the joint entropy between A and B? Let's use this formula, and see if our answer changes.



The joint entropy comes out to be 1, which is different than our last example.

What is the conditional entropy?
The conditional entropy is given by this formula.



Let's substitute the values to find out what we get. The conditional entropy comes out to be 0.

What is a mutual information between A and B?

$$I(A,B) = H(A) - H(A|B) = 1 - 0 = 1$$

So the, so the mutual information in this case is 1, while in the previous case, it was 0.

So since these coins are dependent on each other, the random variable A gives us some information about the random variable B. This tells you, how mutual information works.

12.20. Kullback-Leibler Divergence

To conclude our discussion of information theory, we will also discuss something called Kullback-Leibler divergence. It is also famously called the KL divergence.
It is useful to realize that mutual information is also a particular case of KL divergence. So KL divergence actually measures the difference between any two distributions. It is used as a distance measure. For this particular lesson, it is sufficient to understand how KL divergence is used to measure the distance between two distributions.

The KL divergence is given by this particular formula.

$$D(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)}$$

It is always non-negative and zero only when P is equal to Q. When P is equal to Q the log of 1 is zero, and that's why the distance is zero. Otherwise, it is always some non-negative quantity. So it serves as a distance measure. But it is not completely a distance measure because it doesn't follow the triangle law.

But then you should ask yourself why you need to know KL divergence, or where it is used?

Usually in supervised learning you are always trying to model data to a particular distribution. So in that case one of the distributions can be of unknown distribution. And we can denote that as P(x). And then can sample our data set to find out Q(x).

While doing that, we can use KL divergence as a substitute to the least square formula that we used for fitting. So it's just a different way of trying to fit your data to your existing model. And we'll come back to KL divergence in some of our problem sets.

12.21. Summary

So, let's summarize what we have learned now.
We, we first understood what is information and we found out that information can be measured in some way and we measured it in terms of entropy.

Then we started to understand how we can measure the information between two way variables. And there we defined terms as, terms like joint entropy, conditional entropy and mutual information.

And then finally we introduced a term called a KL divergence, which is very famously used as a distance measured between two distributions.

This was just a primer to information theory and it forms as a base to what is required for you to go through this machine learning course. If you want to learn more about information theory, follow the links in, in the Comments sections.