CS7641 ML Lectures Transcription

October 6, 2023

CONTENTS

CHAPTER

1

SUPERVISED LEARNING

1.1 Decision Trees

1.1.1 Difference between Classification and Regression

Okay, hi Michael. Hey Charles, how's it going. It's going pretty well, how's it going in your end of the world? Very nice, what are we want to talk about today? Well today we are going to talk about supervised learning. But, in particular what we're going to talk about are two kinds of supervised learning, and one particular way to do supervised learning. Okay, so the two types of supervised learning that we typically think about are classification. And regression. And we're going to spend most of the time today talking about classification and more time next time talking about regression. So the difference between classification and regression is fairly simple for the purposes of this discussion. Classification is simply the process of taking some kind of input, let's call it x. And I'm going to define these terms in a couple of minutes. And mapping it to some discrete label. Usually, for what we're talking about, something like, true or false. So, what's a good example of that? Imagine that I have a nice little picture of Michael. It looks just like me! It looks exactly like you. So I have a nice little picture here and I want to know whether this is a male. Or a female. So given an input like this I will map it to male or female. So what do you think, Michael? Do you think this is a male or a female? So you're, you're classifying me as male or female based on the picture of me and I would think you know, based on how I look I'm clearly male. Yes. In fact, manly male. So, this would be a classification from pictures to male. The alternative would be something like a picture to female, and I'm just going to take a completely stereotypical image of either a female or. I think it's actually, that's actually me when I let my hair go long. Right, so, so which points out that this can be pretty hard. But this is where we're going to spend most of our time talking about it first as a classification task. So taking some kind of input, in this case pictures, and mapping it to some discrete number of labels, true or false, male or female, car versus cougar, anything that, that you might imagine thinking of. Car versus cougar? Yes. That, I guess that's an important thing if you're driving. You don't want to run into any cougars or probably other cars either. Well you know, you're sitting down and you're trying to decide whether you should ride this thing that you see or not. And if its a cougar maybe you don't want to and if it's a car maybe you do. Excellent. Don't drive a cougar. Don't drive a cougar. That's the first lesson in machine learning. Excellent. Okay, so that's classification. We'll return to regression in a little bit later during this conversation. But, just as a preview, 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. Okay, so that's regression but we'll talk about that in a moment. Right now, what I want to talk about is classification. Would an example of regression also be, for example, mapping the pictures of me to the length of my hair? Like a number that represents the length of my hair? Absolutely, for the purposes of, 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. Cool, let's do a, let's do a quiz. Make sure we get this. Okay, I like that.

1.1.2 Classification or Regression Question

So we've talked about uh,classification and regression and gave a couple of examples of each. So now I want you to take a moment to make certain that you understand the difference because it's a crucial difference for supervised learning. Here's a very short little quiz. You have three questions here and we divided the world up into some input to some learning algorithm, whatever that learning algorithm is. The output that we're expecting and then a box for you to tell us whether its classification or regression. So, the first question, the input is credit history, whatever that means, the number of loans that you have, how much money you make, how many times you've defaulted, how many times you've been late, the sort of things that make up credit history, and the output of the learning algorithm is rather you should lend money or not. So you're a bank, and you're trying to determine whether given a credit history, I should lend this person money, that's question one. Is that classification, or is that regression? Question two you take as input a picture like the examples that we've given before. And the output of your learning system is going to be whether this person is of high school age, college age, or grad student age. The third question is very similar. The input is again a picture. And the output is, I guess, of the actual age of the person, 17, 24, 23 and a half, whatever. So take a moment and try to decide whether these are classification tasks or regression tasks.

1.1.3 Classification or Regression Solution

And so now let's give the explanation for the quiz. Alright, so, let's see what happened here. So, what you're saying is in some cases the inputs are discrete or continuous or complicated. In other cases the outputs could be discrete or continuous or complicated. But I think what you were saying is, that what on, on, what matters to determine if something is classification or regression is whether the output is from a discrete small set or, or whether it's some continuous quantity. Is that right? Right, that's exactly right. 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. So, with that in mind, what do you think is the answer to the first one? So, lend money. If it was something like predicting a credit score, that seems like a more continuous quantity. But this is just a yes no, so that's a discrete class, so I'm going to go with classification. That is, correct. It is classification and the short answer is, because it's a binary task. True, false. Yes, no. Lend money or don't lend money. Got it. So it's a simple classification test. Okay, with that in mind, what about number two? Alright, so number two. It's trying to judge something about where they fall on a scale, high school, college, or grad student. But all of, the system is being asked to do is put them into one of those three categories, and these categories are like classes, so it's classification. That is also exactly right. Classification. We moved from binary to trinary in this case, but the important thing is that it's discrete. So it doesn't matter if it's high school, college grad, professor, elementary school, any number of other ways we might decide where your status of matriculation is is a small discrete set. So, with that in mind, what about number three? Alright, so the input is the same in this case. And the output is kind of the same except there's, well there's certainly more categories because there's more possible ages than just those three. But when you gave the example you did explicitly say that ages can be fractional like, you know, 22.3. So that definitely makes me think that it's continuous, so it should be regression. Right, I think that is exactly the right thing, you have a continuous output. Now, I do want to point something out. That while the right answer is regression, a lot of people might have decided that the answer was classification. So, what's an argument? If I told you in fact the answer was classification, what would be your argument for why that would be? Well, I guess. Can you think of one? Yeah, I guess. I mean, you know, if you think about ages as being discrete. You just say, you know, you can one or two or three or, you know, whatever up to 130, say. But there, but there's just that, that set. There isn't really, you know, usually we don't talk about fractional ages. So, so it seems like you could think of it as, as, as a set of classes. Right. So let's imagine. So, how old are people? Let's imagine we only cared about years, so you're either one or two or three or four or five. Or maybe you can be one and a half, and two and a half, and three and a half. But, whatever, it's, it's not all possible real number values. And we know that people don't live beyond, say, 250. Well, in that case, you've got a very large discrete set but it's still discrete. Doesn't matter whether there's two things in your set, three things in your set, or in this case 250 things in your set, it's still discrete. So, whether it's regression, or classification, depends upon

exactly how you define your output and these things become important. I'm going to argue that in practice, if you were going to set up this problem, the easiest way to do it would be to think about it as a real number and you would predict something like 23.7. And so it's going to end up being a regression task and we can might, maybe think about that a little bit more as we move along. So either answer would be acceptable depending upon what your explanation of exactly what the output was. Was. You buy that? That makes sense. Excellent. Okay. Alright, let's move beyond the quiz and start thinking about exactly what it means to define a classification problem. And then later what it means to define a regression problem.

1.1.4 Classification Learning One

Okay so, classification learning. Before we get into the details of that, I want to define a couple of terms. Because we're going to be throwing these terms out all throughout the lessons. And I want to make certain that we're all on the same page and we mean the same thing. But we're returning to this again and again and again. So, the first term I want to introduce is the notion of instances. So, instances, or another way of thinking about them is input. Those are vectors of valu, 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, or how much money Michael makes. [LAUGH] How much money I wish I made, so on and so fourth. 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. So you have instances, that's the set of inputs that you have. 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. And for most of the conversation that we're going to be having, we're going to be thinking about binary classification, just true and false. But the truth is, this would work whether there were three outputs, as we did with high school, college, or grad school, or whether there were 250 as we were contemplating for ages. But the main thing here is the concept, is the function that we care about that's going to map pictures, for example, to true or false. So okay, I get, I get the use of the word," instance", right?" Instance" is just, like, a single thing that's out there. But I have an intuitive notion of what a concept is. How does that relate to this more formal notion? Like, can we connect this to, kind of, the intuitive notion of what a concept is? I guess so. So a concept, I don't know. How would you want to put that? So a concept is something That, I mean were talking about is a notion of a function, so what it means formally is that you have some input, like a picture, and it immediately inputs maps anything in that input space to some particular defined output space, like true or false, or male or female, or credit, credit worthy or not. 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. Okay, I think I see that. So, so essentially if you want to think of tallness, the concept of tallness, one way to define it is to say, Well in general if you give me something I can tell you rather or not its [UNKNOWN] so it's going to map those somethings to am I tall or not. True or false. Right. Exactly and so really when you think about any concept and we talk about this in generally [UNKNOWN] is effectively a way of saying is effectively a set of things. That are apart of that concept. So, you can have a concept of a car and if I gave you "cars" you would say these things are in it and if I gave you "non-cars" you would say they are not. And so a concept is, therefore by definition, a mapping between objects in a world and membership in a set, which makes it a function. Okay, that makes sense. Okay.

1.1.5 Classification Learning Two

So with that, with that notion of a concept as functions or as mappings from objects to membership in a set we have a particular target concept. And 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. All right? So, a function that determines whether something is a car or not, or male or not, is the target concept. Now this is actually important, right, because we could say that. We have this notion in our head of things that are cars or things that are males, or thing, or people who are credit worthy but unless we actually have it written down somewhere we don't know whether it's right or wrong. So there is some target concept we're trying to get of all the concepts that might map pictures or people to true and false. Okay, so if you trying to teach me what tallness is so you have this kind of concept in mind of these, these things are tall and these things are not tall. So you're going to have to somehow convey that to me. So how are you

going to teach me? So that's exactly the, well that's what comes up with the rest of these things that we're defining here. Okay. So let me tell you what the next four things are then you can tell me whether that answers your question. Got it. How about that? OK, so we've got a notion of instances, input, we've got a notion of concepts. Which take input and maps into some kind of output. And we've got some target concepts, some specific function from particular idea that we're trying to find, we're trying to represent. But out of what? So that's where the hypothesis comes in. And in fact I think it's better to say hypothesis class. So that's the set of all concepts that you're willing to entertain. So it's all the functions I'm willing to think about. So why wouldn't it just be all possible functions? It could be all possible functions and that's a perfectly reasonable hypothesis class. The problem with that is that if it is all possible functions it may be very, very hard for you to figure out which function is the right one given finite data. So when we actually go over decision trees next I think it will be kind of clear why you need to pick a specific hypothesis class. Okay. So let's return to that in a little bit but it's an excellent question. So, conceptually in the back of your head until we, we come to specific examples, 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 regeression. And we've already said, we're not doing regression. We're doing classification. So already hypothesis classes all functions we care about and maybe it's all classification functions. So we've already picked a subset. So we got all these incidences, got all these concepts, we want to find a, a particular concept and we've got this set of functions we're willing to look at. So how are we going to determine what the right answer is. So if you try to answer Micheal's question that we do that in machine learning is with a sample or another name for which I prefer is a training set.

1.1.6 Classification Learning Three

So what's a training set? Well a training set is a set of all of our inputs, like pictures of people, paired with a label, which is the correct output. So in this case, yes, this person is credit worthy. [LAUGH] Versus another example. You can tell I'm credit worthy based on my curly hair. Purely on the hair. Versus someone who has no curly hair and therefore is obviously not credit worthy. And 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. I see. So instead of just telling me what tall means, you're going to give me lots of examples of, this is tall, this is not tall, this is tall, this is tall, this is tall, this is not tall. And that's the way that you're explaining what the target concept is. Right. So if you want to think about this in the real world, it's as if we're walking down the street and I'm pointing out cars to you, and non-cars to you, rather than trying to give you a dictionary that defines exactly what a car is. And that is fundamentally inductive learning as we talked about before. Lots and lots of examples, lots of labels. Now I have to generalize beyond that. So, last few things that we we talk about, last two terms I want to introduce are candidate, and testing set. So what's a candidate? Well a candidate is just simply the, a concept that you think might be the target concept. So, for example, I might have, right now, you already did this where you said, oh, okay I see, clearly I'm credit worthy because I have curly hair. So, you've effectively asserted a particular function that looks at, looks for curly hair, and says, if there's curly hair there, the person's credit worthy. Which is certainly how I think about it. And people who are not curly hair, or do not have curly hair are, in fact, not credit worthy. So, that's your target concept. And so, then, the question is, 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 credit worthy 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. So in this case, because you decided curly hair matters, I have drawn, I have given you two examples from a training set, both of which have curly hair, but only one of which is deemed credit worthy. Which means your target concept is probably not right. So to do that test I, guess you can go through all the pictures in the testing set, apply the candidate concept to see whether it says true or false, and then compare that to what the testing set actually says that answer is. Right, and that'll give you an error. So, by the way, the true target, the true target concept is whether you smile or not. Oh. That does make somebody credit worthy. It does in my world. Or at least I, wish it did in my world. Okay. So, by the way 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 training set to include lots of examples that you don't, the testing set, I'm sorry, to include lots of examples that you don't see in your training set. And

that is proof that you're able to generalize. I see. So that, and that makes intuitive sense, right? So, like, if, if you're a teacher and you're telling me, you give me a bunch of fact and then you test me exactly that bunch of facts, it doesn't, I don't have to have understood them. I just can regurgitate them back. If you really want to see if I got the right concept, you have to see whether or not I can apply that concept in new examples. Yes, which is exactly why our final exams are written the way that they are written. Because you can argue that I've learned something by doing memorization, but the truth is you haven't. You've just memorized. Here you have to do generalization. As you remember from our last discussion, generalization is the whole point of machine learning.

1.1.7 Example 1 Dating

All right, so we've defined our terms, we, we know, what it takes to do, at least supervised learning. So now I want to do a specific algorithm and a specific representation, that allows us to solve the problem of going from instances to, actual concepts. So what we're going to talk about next are decision trees. And I think the best way to introduce decision trees is through an example. So, here's the very simple example I want you to think about for a while. You're on a date with someone. And you come to a restaurant. And you're going to try to decide whether to enter the restaurant or not. So, your, input, your instances are going to be features about the restaurant. And we'll talk a little bit about what those features might be. And the output is whether you should enter or not. Okay, so it's a very simple, straightforward problem but there are a lot of details here that we have to figure out. It's a classification problem. It's clearly a classification problem because the output is yes, we should enter or no, we should move on to the next restaurant. So in fact, it's not just a classification problem, it's those binary classification problems that I said that we'd almost exclusively be thinking about for the next few minutes. Okay. So, you understand the problem set up? Yes, though I'm not sure exactly what the pieces of the input are. Right, so thats actually the right next question to ask. We have to actually be specific now about a representation. Before I was drawing squiggly little lines and you could imagine what they were, but now since we're really going to go through an example, we need to be clear about what is it mean to be standing in front of the restaurant. So, let's try to figure out how we would represent that, how we would define that. We're talking about, you're standing in front of a restaurant or eatery because I can't see the reliably small restaurant. And we're going to try to figure out whether we're going to go in or not. But, what do we have to describe our eatery? What do we have? What are our attributes? What are the instances actually made of? So, what in, or another way of putting it is, what are the features that we need to pay attention to that are going to help us to determine whether we should yes, enter into the restaurant. Or no, move on to the next restaurant. So, any ideas Michael? Sure. I guess there's like the type of restaurant. Okay, Oh, is it tall or short, and is it a good credit risk? [LAUGH] Oh wait, no, no, no wait, I know. Like the Italian versus French, versus, you know, Vietnamese. So let's call that the type. So it could be Italian, it could be French, it could be Thai, it could be American, there are American restaurants, right? Sure. Greek, it can be, Armenian. It can any kind of restaurant you want to. Okay, good. So that's something that probably matters because maybe you don't like Italian food or maybe you're really in the mood for Thai. Sounds perfect. Okay anything else? Sure. How about whether it smells good? You know, I like cleanliness. Let's let's, or you know what, let's, let's be nice to our eateries and let's say atmosphere. Mm. Right because if there's, you know, no atmosphere, then it is going to be really hard to breathe. That's exactly right. So is it fancy? Is it a hole-in-the-wall, which I'm going to spell HIW. Is it a hole-in-the-wall, umm, those sorts of things. The, you know? Casual, I guess, is another category. Casual. And so on, and so forth. You could imagine lots of things like that, but these things might matter to you and your date. Okay, so, we know the type of the restaurant that we have, we know whether it's fancy, whether it's casual, whether it's a hole in the wall. Some of the best food I've ever had are in you know, well-known hole in the walls. Those sorts of things. Anything else you can think of? Sure, Sometimes, I might use something like looking inside and seeing whether there's people in there and whether they look they're having a good time. Right. So that's an important thing. So let's just say If it's occupied. Now why might that matter in reality? Well it matters because if it's completely full and you may have to wait for a very long time, you might not want to go in. On the other hand. If you're looking at a restaurant you've never heard of, and there's only two people in it, and it's Friday at 7 p.m. Maybe that says something about something. Maybe you want it to be quiet. You know, those sorts of things might matter. Okay, so, we've got type, we've got atmosphere, we've got occupied. Anything else you can think of? And I have been out of the dating market for a while, but I guess it could imagine, I could imagine how hard I am trying to work to impress my date. perfect. So do you have a hot date or not? Or, this is someone who you really, really, really want to impress and so, maybe it matters then, it's even more important whether it's a fancy restaurant or a hole in the wall, or whether it's French or whether it's an American restaurant. That make

sense? I think that makes sense. Notice, by the way, that the first two sets that we have have multiple possible categories here. So it could be Italian, French, Thai, American, so on and so forth. Atmosphere is something that can have many, many possible values, but the last two things that we talked about were all binary. Either it's occupied or it's not. Yes or no or, you have a hot date or you don't. And I think we could go on like this for a long time but, let's try to move on to maybe a couple of other features and then try to actually figure out how we may actually solve this.

1.1.8 Representation

Alright, so Michael. Last set of features that that's come up with three or four, three or four more features and then move on. Sure. So come up with a couple. Alright, so I could, sometimes I'll look at the menu that's posted outside, and I'll see if the, you know? How pricey it is. Okay, so cost. Right, so cost can be represented as discrete input. By the way, it could also be represented as an actual number. Right? We could say, look it's cheap, it's moderately expensive, it's really expensive or you could have a number which is the average cost of an entry. And it doesn't really matter for, for what we're talking about now but just some way of representing cost. Okay. Just give me one or two more features but I want to give me some features that don't have anything to do with the restraunt itself but might still be important. Hmm. because, by the way, hot date probably doesn't have anything to do with the restaurant itself. So, even though we've been talking the features of the restaurant. We've actually been picking up, at least, one feature that doesn't have anything to do directly with the restaurant itself. Not to. So, whether I'm hungry? I like that. Here's another one. What's the weather like. Is it raining outside? Which is a different sense of atmosphere. Right. because if it's raining outside, maybe it's not your, your favorite choice but you don't want to walk anymore. Okay, so we have a ton of features here. We've gone through a few of them. Notice that some of the specifically have to do with the restaurant and some of them have to do with things that are external to the restaurant itself but you can imagine that they're all important. Or possibly important to whether you should enter into the restaurant or not. Agreed? And there's a bunch of features you could imagine coming up with that probably have nothing to do with whether you should enter into the restaurant or not. Like, how many cars are currently parked across the country. Probably doesn't have an impact on whether you're going to enter into a specific restaurant or not. Okay. So, we have a whole bunch of features and right now we're sticking with features we think that might be relevant. And we're going to use those to make some kind of decision. 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. So the first thing we want to think about is the fact that a decision tree has a specific representation. Then only after we understand that representation and go through an example, we'll start thinking about an algorithm for finding or building a decision tree. Okay, are you with me? Yeah. Excellent. Okay, so a decision tree is a very simple thing. Well, you might be, might be surprised to know it's a tree, the first part of it. And it looks kind of like this. So, what I've drawn for you is example. Sample generic, decision tree. And what you'll see is three parts to it. The first thing you'll see is a circle. These are called nodes, and these are in fact, decision nodes. So, what you do here, is you pick a particular attribute. [NOISE] And you ask a question about it. The answer to that question, which is its value for what the edges represent in your tree. Okay. So we have these nodes which represent attributes, and we have edges which represent value. So let's be specific about what that means. So here's a particular attribute we might pick for the top node here. Let's call it hungry. That's one of the features that Michael came up with. Am I hungry or not? And there's only two possible answers for that. yes, I'm hungry, true, or false, I am not hungry. And each of these nodes represent some attribute. And the edges represent the answers for specific values. So it's as if I'm making a bunch of decisions by asking a series of questions. Am I hungry? And if the answer is yes, I am hungry, then I go and I ask a different question. Like is it rainy outside? And maybe it is rainy and maybe it's not rainy, and let's say if it isn't rainy then I want to make a decision, and so these square boxes here are the actual output. Okay so it's hungry, so you're hungry, yes, and it's not raining, so what do you do? So, let's just say you go in. True, I go in so, when it's, I'm hungry and it's not raining, I go in. And that, that true is answering a different question. It's not in the nodes I guess. So, in the leaves, the t and f means something different. That's right. It's the out, that's exactly right. The, the leaves, the little boxes, the leaves of your decision tree is your answer. What's on the on the edges are the possible values that your attribute can take on. So, in fact, let's try to, let's make that clear by picking a different by picking another possible attribute. You could imagine that if I am not hungry, what's going to matter a lot now is say, the type of restaurant, right. Which we said there were many, many types of restaurants. So, you know thai, I forget [CROSSTALK] Italian. Italian, and you know, something. French fries. And French Fries. So if I'm not hungry, then what matters a lot more is the type of restaurant, and so I'll move down this path instead

and start asking other questions. But ultimately, no matter how I, what this decision tree allows me to do is to ask a series of questions and depending upon those answers, move down the tree, until eventually I have some particular ouput answer, yes I go in the resteraunt, or no I do not Ok this is still seeming a little abstract to me, can we, can we maybe work through a very concrete example. Yeah, I think that makes a lot of sense. Let's do a quiz. Ha, okay, let's do a quiz.

1.1.9 Representation Quiz Question

Okay, so we've now seen an abstract example of decision trees. So let's make certain that we understand it with a concrete example. So, to your right is a specific decision tree that looks at some of the features that we talked about. Whether you are occupied or not, whether the restaurant is occupied or not what type of restaurant it is. Your choices are pizza, Thai, or other. And whether the people inside look happy or not. The possible outputs are again binary either you don't go or you do go, into the restaurant. On your left is a table which has six of the features that we've talked about. Whether the restaurant is occupied or not, the type of restaurant, whether it's raining outside or not, whether your hungry or not. Whether you're on a hot and whether the people inside look happy, and some values for each of those. And what we would like for you to do is to tell us what the output of this decision tree would be in each case. Alright, so the decision tree here is a, it's a classifier, but we had another name. Oh, it's a concept. Yes. Alright, and each row of this is a different time that we're stopping at a restaurant, and the, the little values there summarize what is true about this particular situation. And, and you're saying we need to then trace through this decision tree and figure out what class is, okay yeah, that's what you said. Okay, I'm ready. Alright, perfect. Okay, that's the quiz, go.

1.1.10 Representation Quiz Solution

You've got your answers. Let me tell you what we think the answers are. Now the nice thing about a decision tree sort of conceptually and intuitively, is that it really is asking a series of questions. So, we can simply look at these rows over here and the values that our features have and we can just follow down the path of the tree. So, in the first case. We have true. We have true for occupied, which means we want to go down the right side of the tree. And check on the type. So in the first case, the type is pizza. And so we go down the first branch and that means. We do not go down the tree. So, the output is no go. So, okay, so now, I got a different answer. So, I looked at this and I said happiness is true. And, the bottom box says happiness true, you go. Right. So, you got the wrong, you got what I'm going to tell you is the wrong answer by going from the bottom of the tree up. The way decision trees work is you always start at the root of the tree. That is the very top of the tree. And you ask the questions in tho, in that order. It just seems like it would be faster to start at the bottom. Yeah but then you would never look at anything else in the tree. Good point. All right so. If you start at the bottom, you can't go up. Alright. So, okay, so let me see if I get this straight. So I'll, I'll do the, the second instance. The second instance, you say that we need to start at the top of the tree where it says occupied. And so now I look at the instance and the instance says that it's false for occupied, so we go down that left branch and we hit no go. Oh wait but now I haven't look at any of the other nodes. You don't have to look at any of the other nodes because it turns out that if it's not occupied you just don't go into a restaurant. So you're the type of person who doesn't like to be the only person in a restaurant. Got it, all right, so that's a no go. So that's a no-go. That's an important point, Michael. Actually, you might also notice that this whole tree, even if you look at every single feature in the tree, only has three of the attributes. It only looks at occupied. Type. And happiness. I see. So hot date is sort of irrelevant which is good, because in this case it's not really changing from instance to instance anyway. True. And neither is hungry you might notice. Oh, I am kind of hungry. Although, I'm always hungry. Although raining does in fact change a little bit here and there. But it apparently it doesn't matter. I see. Because you always take an umbrella. Got it. Okay, so let's quickly do the other three and see if we we come to the same conclusion. Alright. Well all the instances that have occupied false we know those are no go, right away. Oh, good point. So we can do it kind of out of order. And the other ones are both occupied. One is tie and one is other. For the tie one we go. The other one, oh I see, for the other one we have to look at whether there's happiness or not, and in this instance happiness is true. So we get on the right branch and we go. And we go. Exactly it. So we notice hot date doesn't matter, hungry doesn't matter and rainy doesn't matter. And the only thing that matters are whether you're occupied, what type of restaurant you're at and whether you're happy or not. Or whether the, the patrons in the restaurants are restaurant is, are happy or not. But, here's the other thing about this. It's not just about the features. Let's tie it back in to the other things, that we mentioned in the beginning. This, in our case, this table actually

represents our testing set. It's the thing that we're going to look at to determine whether we were right or wrong. These are the examples that we're going to do to see whether we generalize or not. And this particular tree here is our candidate concept. So there's lots and lots and lots of other trees that we might have used. We might have used a tree that also took, asked questions about whether it was rainy or not or asked questions about whether you were on a hot date or not. But we didn't. We picked a specific tree that had only these three features and only asked in this particular way. So what we're going to talk about next. Is how we might decide whether to choose this tree over any of the other possible number of trees that we might have chosen instead.

1.1.11 Example 2 - 20 Questions

Alright, so we understand at this point how to use these decision trees, but this is a class about machine learning, right? So we need to figure out how to make those decision trees happen as a result of processing a set of training data. So, it's not clear to me how we're going to make that leap. Let's see if we can figure it out together. So, I'm going to play a game with you, Michael, and if we do the game well. Then I think we'll have an idea of what a good algorithm might be for actually building a decision tree. So we're going to play 20 questions. You're familiar with 20 questions? Right, that's the game where I'm allowed to ask yes/no questions to try to guess something that you're thinking of, and if I take more than 20 of them, then I lose. Right, okay. So, here's what I'm going to do. I'm going to think of a thing, and, you ask me questions and I'm going to answer them and we'll see if you can guess what thing I'm thinking about. Okay, I've got something in my head. The first question, the typical first question is it animal, vegetable or mineral but that doesn't seem like a yes/no question, so is it, is it an animal, or like a, a living creature. The answer is yes. Alright, is it a person? Is it a person? The answer is yes. Is it a famous person? Is it a famous person? That's a deep philosophical question, but I'm going to say yes. Is it a famous person that we both know in like directly. Oh, who we both personally know directly, I do not believe so. Yeah. So, the answer is no. Is it a living person? Living person, no. So it is a dead, famous person. Is the person famous for, say being in the music industry. The answer yes. Did this person live during the 20th century? The answer is yes. Is the genre of music that the person was associated with, say hip hop or rap? No. Is the person a singer? Singer? Yes. Male or female, is the person female? Person female? No. That's ten questions, Michael. Yes, the, the clock is ticking down, but I feel like I have narrowed it down quite a bit at this point. Did the person die since you've become a professor? So, say in the last How long have you been a professor? Too forever it sounds like feels like. Let's see, recent death. And I'm going to say the answer is yes. Is the person's name Michael? The answer is yes. Alright, alright, I think I'm on to it. Is it Michael Jackson? No. Woo hoo! Of course, it's Michael Jackson. Alright, Thriller. Yes, Michael Jackson is the answer. Alright. So that was, that was very fun. And I'm very glad that I was able to solve it. [LAUGH] But it's not clear to me how, this is going to give us an algorithm for building decision trees. Okay, so let's think about that for a second. So you asked a bunch of questions, and you asked them in a particular order. Right? So, here's a question I have for you. Why was the first question you asked me whether it was an animal or not? well, it seemed like I needed some way of narrowing down the space, and so I wanted to get as much information out of that question as I could to try to make progress towards figuring who it actually was. Right. So, animal is the first question and it was because it narrowed things down. So, your goal in asking questions was to narrow the possibilities, right? Sure, right because I only have 20 questions and then I'm, you know I'm out of it, so if I asked questions like You know, if I had said–started with, Is it someone named Michael, that would have been really bizarre. Right, and if the answer was no, it's not clear that it would tell you anything at all. So, actually, that's an interesting point. You started with animal; you could have started with Michael. And if I had said yes, that would have told you something. But if I had said No, it wouldn't have told you hardly anything at all. Right? So animal is a better attribute, or feature, to ask about than Michael as a first question. Do you agree with that? Yeah, because it could have been like a stapler or something like that and then I, the Michael question would've been pretty silly. Exactly. So what about persons? So first you asked about animal. Then you asked about person. Why person? Right, because, because again it seemed like of the space of possibilities that I could think of person would help kind of again narrow things down. That I'd if it was the answer was yes, I would be able to focus there. If the answer was no I could focus some place else. Exactly. And so I think in fact we could, we have a general statement here. That each one of these questions. Person, famous, do we know this person, personally, so on and so forth. All make sense because they further narrow down the possibilities. And that bit about further is important. Because it implies that the usefulness of a question depends upon the answers that you got in the previous questions. So even though Michael is not a particularly good first question to ask, it's a perfectly reasonable twelfth question to ask or however far down it is, given that you already know this person's, this is an animal, a

person, a famous person we don't know personally who's not living, who's into music, etc., etc., etc. Okay, that make sense? Yeah. Okay. So I think then, we have the hints of an algorithm. So let's try to write down that algorithm and, and see if it matches with our intuition.

1.1.12 Decision Trees Learning

Okay, so, inspired by 20 questions let's try to write down exactly what it is that you did in going through your 20 questions to get your answer to discover Michael Jackson was the person I was thinking about. So, what is the very first thing you did? I tried to imagine a bunch of possible answers that you could be, could have in mind. And, I tried to think of a question that would help separate them into two roughly equal chunks. Okay, so what's the way of putting that in the language that we've been using for supervised learning and classification so far? Oh I see. So if I had known in advance, here's here's 200 things that you might ask me about. And what their, what their attribute values are. What would be a question that would split that set in half, right? So, instead of just imagining a bunch of things, if I actually had a list, which I do in the case of a training set. Alright, so the first thing you did is you picked the best attribute that you could think of. And, by the way, you said something very particular here. You actually defined what best is. You said, that best is the same thing as splitting things roughly in half. So let's revisit that in a moment. Okay, so the first thing you did is you picked the best atribute. You asked a question about it and then, depending upon the answer, you went and picked another attribute right? Does that seem fair Yeah. Okay. So, we think about decision trees, a way of talking about that is that you follow the path of the answer, and then lather, rinse and repeat. [LAUGH]. You went back and pick the best attribute, asked the question, followed the answer path, so on, and so on, and you kept doing that until what? Until, I narrowed the space of possibilities to, in this case, just one item. Right, so until you got to the answer. All right. And that is an algorithm. So you pick the best attribute, and you actually define what best attribute was. You want to pick one that would somehow eliminate at least half of the things which you might worry about and keep the other half in play. You asked a specific question. You followed the path to that and then you went back and you picked another attribute and so on and so forth until you got an answer that you wanted to. That's an algorithm and that's an algorithm that we might use to actually build a decision tree. And the only difference between what you did and what you would do with learning a decision a tree is that, since you don't know in advance what the answer actually is going to be, because you don't know what specific object I might be thinking of, you have to actually follow all possible paths at each time and think of all possible best next attributes until you completely can answer any question. Does that make sense? I see. So, right, so instead of just playing the game interactively, I kind of imagine all possible ways it could go and build that whole flow chart, that whole tree. Right, so, let's see if we can do that with some pictures and I actually want to decide rather I really believe your definition of best. Okay? Sure. Alright, so, let's do that.

1.1.13 Best Attribute Quiz Question

Alright, so let's take a moment to have a quiz where we really delve deeply into what it means to have a best attribute. So something Michael and I have been throwing around that term, let's see if we can define it a little bit more crisply. So, what you have in front of you are three different attributes that we might apply in our decision tree for sorting out our instances. So, at the top of the screen what you have is you have a cloud of instances. They are labelled either with red x or a green o, and that represents the label so that means that they are part of our training set, so this is what we're using to build and to learn our Decision Tree. So, in the first case you have the set of instances being sorted into two piles. There are some xs and some os on the left and some xs and some os on the right. And the second case you have that same set of data being sorted so that all of it goes to the left and none of it goes to the right. And in the third case you have that same set of data that's sorted so that a bunch of the xs end up on the left and a bunch of the os end up on the right. What I want you to do is to rank each one, where one is the best and three is the least best attribute to choose. Go.

1.1.14 Best Attribute Quiz Solution

Okay Michael. So, are you ready to give me a ranking Yes, yes, I think I am. Okay, well, if you give me a ranking then. So, did you say one was the best. One is the best and three is the least best. Alright, so I am really excited about the third cloud structure. The third attribute to be split on. Because, what it does. Is it takes all our x's and o's That need to have different labels and it puts them into two different buckets. One

where they all get to be red x's and the other where they all get to be green o's. So I would say the far right one is ranked number one. I would agree with you and in fact I would say that infact we're done. Yeah, it's perfect. It is perfect, agreed. One is perfect. Or 3 is perfect in this case, because I gave it a one. Alright, so then, I think the worst one is also easy to pick, because if you look at the middle attribute, the attribute that's shown in the middle, we take all the data, and we put it all on the left. So we really have just kicked the can down the road a little bit. There's nothing. That this attribute splitting has done. So, I would call that the worst possible thing you could do. Which is to basically to do nothing. No, I think that's right. and, in fact, it really is doing nothing. Right. Okay. So what about the first attribute? So, I'm going to put that at, you know, if the first one is too hot and the middle one is too cold, I would say this one is, wait, no, no. [LAUGH] I was going down the Goldilocks path. So, so this one is sort of in between. In that it splits things so you have smaller sets of data to deal with, but it hasn't really improved our ability to tell the reds and the greens apart. So in fact, I'd almost want to put this as three also but I'll put it as two. Okay. I think an argument could be making it three. An argument could be made for making it two. Your point is actually pretty good, right? We have eight red things and eight green things up here. And the kind of distribution between them, sort of half red and, half red x's, half green o's, we have the same distribution after we go through this attribute here. So it does some splitting, but it's still, well you still end up with half red, half green, half x, half o. So, that's not a lot of help, but it's certainly better than doing absolutely nothing. Well is it though? I mean, it seems it could also be the case. That what we've done is that we're now splitting on that has provided no valid information, and therefore can only contribute to overfitting. That's that's a good point. That's a good point. So, do you want to change your answer to three? I don't know, what did you want the answer to be? It seems to me that the first one and the second one are just plain bad. They are just plain bad. The question is whether one is, more bad than the other. I, I don't know. I don't know how to judge. Well I'll tell you, I would accept either two or a three as an answer here. I think you can make an argument either way. And I think you actually made both arguments. That's very nice of you. So. Thank you. You're very welcome. So this is perfect. This is the House of Representatives and this is the Senate. [LAUGH] What do you mean they're? Oh. So, there you go. Okay. [LAUGH] Not exactly sure what you mean, but it seems somehow denigrating to our political system. It is not at all denigrating. It is, it is, I would call it incisive political satire.

1.1.15 Decision Trees Expressiveness AND

Okay. So Michael, for the last 15 minutes or so we've been talking about decision trees, sort of in the abstract without saying too much about the kinds of functions they can actually represent. So, for the next few minutes or so I want to talk a little bit about not just decision trees in the abstract, but exactly how expressive they can be. Is that okay? Yeah, I think that would be really helpful. I think so too. So in fact, I want to look at a set of functions, and in particular I'm interested in looking at Boolean functions. Boolean. So what's your favorite simple Boolean function? Implication. What's your other favorite simple boolean function? Well I like Nor. Right. So what I just heard you say is you like And, so let's do that one. Oh, that's great. That is my favorite. All right. So, in fact, let's do very simple And. So, how does And work, right? So, you've got two attributes. Let's say A and B. And, this expression, A and B, is true when? When A and B are true. When they are both true at the same time. Right, exactly. So, how would you build a decision tree, given that there are only two attribute, A and B, to represent that function? Okay so I'd have a note that's A and B. And if that's true. Nope, you're not allowed to do that. Oh. Every node can be, have at most one attribute. All right well let's let's put A in an attribute. Okay, so here's a little node. Let's call it A. Okay. Now what? And well I mean, for it to be a node it needs to have the little two branchy things. True and false. Okay. All right, so how about true on the left and false on the right? Sure, as long as you label them. So, all right. So then A, if A is true, okay, I don't know. But oh, but if A is false, then we know that A and B must be false. Doesn't matter what B is. So we can just put a leaf under the F. That's correct. All right, I like that. Okay. What about when A is true? What, is that an F? That is an F, and that is a minus sign for false. Oh, a minus sign. I get it, okay. I thought you were just not done drawing the F yet. good. All right. So, oh yeah. On the true side then, I don't think we know. So I think we need to split on B also. Okay. So put a little B under there. All right, done. All right, and then true-false split on B. All right and so now we've got these two cases. So if B is false, then again, it doesn't matter what A was but A turns out to be true. But it's still the, the, it should be a minus sign underneath that. Okay. So it's not A and B is not true. But if A is true and B is true then A and B is true. So there should be a plus sign on the left. That's exactly right. Woo. So clearly decision trees proof by, we just drew it here, can represent the Boolean function And. Sure. [CROSSTALK]. You said something int, you said something interesting, Michael. You said it doesn't matter what A is if B is false. So what would happen if I switch, A and B around. That's the same. So if B, okay, in

the beginning, we say, if B is false, it's false. If B is true, we check A. If A is false, then it's false. But if A is true, then it's true. So it actually still represents exactly the same function, A and B. Oh, because A and B is collaborative. No, commutative. Yes? No? Hello? It's one of those things. It's commutative. As opposed to associative. As opposed to what? Associative. Well I mean it, it's that too. But I mean, it's the reason that you can just switch those two things and it didn't make a difference is because they play the same role in the function. That's true in, in terms of representation of the decision trees. You know, it doesn't really matter which attribute you pick or the order in which you do them. You might get a better tree or a worse tree or a longer tree or a shorter tree. But for something simple like, two valued And, it really just doesn't matter. Okay. kind of neat, huh? Yeah.

1.1.16 Decision Trees Expressiveness OR

Okay, so we can do A and B. So, let's pick another function. What's your other favorite, Boolean function of two variables? Well, if we're doing two attributes, you know, or is the other really nice one. I like or. So, let's see. Do you think, now that we've shown that we can do A and B, could we do A or B? I feel like we could do A or B. because there's that nice De Morgen's Law relationship between them but but let's just, how about this, can you erase the tags at the bottom? Like that? Oh, yeah but that's not going to work, is it? So, so if B and A are both true, so the left branch, then the, the, the leaf should be plus? Yes. And if B is true and A is false then the tag should be plus. Yes. And if B is false, I want to say false but we don't know because A could be true and A or B is true if either of the two of them are true. Oh okay, so maybe the mistake here is just trying to use exactly the same thing. So let's just erase it. All right. And start from scratch. So what would you do starting from scratch? I'd split on B again. Okay. True False or False True. True False. True False. All right, and now if B is true. We know that A or B is true, so we can put a plus under the left side. Right. But if B is false. Mm-hm. Then we need to split on A. Okay. And if A is true, then we're golden, we get the Or is true. And if A is false, then it is false. Very good, and that represents A or B. That represents the function, the logical function A or B, yeah. Right. What happens if I swap A and B around, does it still work? I mean, my, my, since they're collaborative or commutative. Since they're commutative I want to say it shouldn't make a difference, but let's just double check that. So if A is true then the output's true. If A is false and B is true then the output's true. And if A and B are both false, then the answer's false, yeah, totally. Excellent. And, you know, if I hadn't erased it you would see that the two trees actually look very, very much alike. They're sort of mirror each, of each other. If you just swapped around. Yeah the, yeah exactly, they're mirrors of one another.

1.1.17 Decision Trees Expressiveness XOR

[LAUGH] Okay, let's pick one more Boolean function to do. What function am I thinking of? XOR is always good. Let's do XOR. So what does XOR mean? Remind me. Okay. XOR is exclusive or, which means it's true if A is true or B is true, but not if they're both true, so it has elements of both or and and in it. Right. And I think of XOR usually when I'm, when I'm, like, playing with light switches in my house. If I have a, a light that's activated by two different light switches, it's usually an XOR function. If one is up, the light's on, if the other one's up, the lights on, but if they're both up, the light goes back off. Right. The other thing when I think of XOR is when people say or most of the time when people say or, when they're talking, they mean xor. Really? Yeah. So a lot of times when people say or in English, they really mean xor. They say, well do you want to go to the movies or do you want to go swimming? Do want to eat chicken or do you want to eat fish? And really, they're saying either or. I see, you want one or the other, but you can't have chicken and fish or go swimming and the movies. No, those things are not possible to do together. Got it. Okay. All right, so how would you do XOR? We got, we still have our- Well, I would start with the- Or, because it's a lot like or. So, what would you want to do? Okay, so to do XOR, we can split on A. Okay. And there's a true branch and a false branch. And what happened with and and or at this point is, there is at least one branch where we actually knew the answer, at this point, but I don't think that's true here. That's right. So, so if A is true, the output might be true or false. It depends on B. And if A is false, the output might be False or True. It depends on B. So I- This is exactly right. So I guess in both cases we still have to split on B. Okay. All right. So, now it should be relatively easy in the sense that there's a separate leaf for all possible inputs. And so we can just write the XOR function on the bottom. So if A and B are both true, then the output is false because it's exclusive. If A is true and B is false, then it should be true, because A is true. If A is false and B is true, then it should be true because B is true. And if A and B are both false then it should be false. That's right. And by the way, if you were to think about it for a little while, it would probably occur to you that's this tree is just a another representation of the full truth table. Very

nice. And in fact, if we wanted to do or again, we could say, well I was very smart last time, but I could actually write or like this. And then just fill out the values at the bottom. If A is true and B is true, then yes. If A is true and B is false, then yes. If A is false and B is true, then yes. If A is false and B is false, then no. And this is a perfectly legitimate a decision tree to represent or because it basically is just another simple representation of the truth table, but as we pointed out when did last time, it's more of a decision tree than we actually need. I can see that. Because in particular, we don't actually need this. All right. So this little difference here between writing out the entire truth table on the left, as we did for XOR, and not having to write out the entire decision tree on the right for o,r actually isn't going to turn out to matter when we try to do things either more complicated or with more attributes that we did for the simple and, or, and XOR. Would you like to see? Yeah, that sounds really cool. Beautiful.

1.1.18 Decision Tree Expressiveness

So, we saw before when we looked at AND and OR versus XOR that in the case of AND and OR we only needed two nodes but in the case of XOR we needed three. The difference between two and three is not that big, but it actually does turn out to be big if you start thinking about having more than simply two attributes. So, let's look at generalized versions of OR and generalized versions of XOR and see if we can see how the size of the decision tree grows differently. So in the case of an n version of OR. That is we have n attributes as opposed to just two. We might call that the any function. That is a function were any of the variables are true then the output is true. We can see that the decision tree for that has a very particular and kind of interesting form. Any ideas Michael about what that decision tree looks like? So, well. So going off of the way you described OR in the two case. We can start with that. And you. You pick Pick one of the variables. And if its true then yeah. Any of them is true since the leaf is true. What happens if its false? Well, then we have to check what everything that's left. [LAUGH] So then we move on to one of the other attributes like A2. mm hm. And again, if it's true, it's true and if it's false then we don't know. Look at A3. Good idea. This could take some time. Yes, oh that was actually an interesting point. Let's say if there were only three, we would be done. Right? Right. But wait, what if there were five? Then we need one more node. What if there were n? Then we need n minus 4 more nodes. Right, so what you end up with in this case is a nice little structure around the decision tree. And how many nodes do we need? Looks like one for each attribute, so that would be n. n nodes, exactly right. So we have a term for this sort of thing, the size of the decision tree is, in fact, linear. In n. And that's for any. Now what about an n version of XOR? Mm. So XOR is, if one is true but the other one's not true then it's true. And if they're both true. Yeah I don't. It's not clear headed. Generalize that. So why not hmm. So, while the usual general version of this we like to think of as parity. All parity is a way of counting, so there's usual two forms of parity that we worry about. Either even parity or odd parity. So let's pick one, it doesn't matter. Let's say. Odd. I like that, we'll do odd parity. And all that works out to be in this case is, if the number of attributes that are true is an odd number, then the output of the function is true, otherwise it's false. Got it? Got it. Okay, so, how would we make that decision tree work? Ooh. Well, we got to split on something and there all the same, so let's split on A1 again. Okay. So what do we do if A1 is true, versus being false. We don't know much if A1 is true. We have to look at everybody else. Right. So let's look at A2. What if A2 is true versus false? Well if A1 and A2 are true then, then the output is going to be whatever the parity of all the remaining variables are. So you still have to do that. Uh-huh, yup. And I'm already running out of room, so let's pretend there's only three variables. What's the output? [LAUGH] All right, so the far left. Is there's three trues which is odd so the output is true. Yep. The next leaf over, only two trues. A1 is true, A2 is true, but A3 is false, so that's two trues which is is even so the answer's false. Um-huh. Is this going to, is this pattern continuing? Now we've got. No, so then it's false again because we've got two trues and a false to get to the next leaf. Mm-hm. And we've got one true to get to the next leaf so that's true. Oh, that looks like XOR. It looks just like XOR. In fact, each one of these sub trees is kind of a version of XOR isn't it? Now what we have is, we have to do the same thing on the right. So we gotta redo A2, and we're going to be in the same situation before. And we're going to start drawing on top of each other because. [LAUGH] there's just not enough room. Hm, so, what's the answer to the one on the very right. Where all of them is false. All of them are false. So that's, an even number of trues. Zero is even. So that's false. Okay, so in the case where only A3 is true, it's true and we just keep going on and on and on again. Now imagine what would happen, in fact let me ask you Michael, what would happen if we had four attributes instead of three. Then we would be really tired of this game. Yes, and I am already tired of this game so the question is, can you. We get a whole another, a whole other level of this tree. Yep. We have the, it just goes on and on and on and nobody wants to think about it anymore. [LAUGH] So, how many nodes do you think there are? Well, for three there was one, two, three, four, five, six, seven. Which seems suspiciously like one less than the power of two.

Mm-hm. And that is exactly right. You need more or less 2 to the n nodes. Or. 2 to the n, maybe, minus 1. Yeah. So let's just say big O of 2 to the n. Everyone watching this is a computer scientist to they know what they're doing. Okay so, you need an exponential therefore, as opposed to linear number of nodes. Gad. Yeah, so you very, very quickly run out of room here. You very, very quickly have a really, really big tree because it's growing exponentially. So, 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, the parity questions. And we can't be particularly clever about how we do it. Though, if we were kind of clever and added another attribute, which is like, the sum of all the other attribute values, that would make it not so bad again. So maybe it's just a, it's just a kind of, bad way of writing the problem down. Well, you know, what they say about AI is that the hardest problem is coming up with a good representation. So what you just did is, you came up with a better representation, where you created some new pair, new variable. Let's call it B, which is just the sum of all of the As, where we pretend that I don't know, true is one and false is zero. This is actually a really good idea. It's also called cheating. [LAUGH] Because you [LAUGH] got to solve the problem by picking the best representation in the first place. But you know what? It's a good point, that in order for a machine running to work, you either need an easy problem or you need to find a clever way of cheating. So, let's come back and think about that throughout all the rest of the lessons. What's the best way to cheat?

1.1.19 Decision Tree Expressiveness Quiz Question

All right, so what that last little exercise showed is that XOR, in XOR parody, is hard. It's exponential. But that kind of provides us a little bit of a hint, right? We know that XOR is hard and we know that OR is easy. At least in terms of the number of nodes you need, right? But, we don't know, going in, what a particular function is. So we never know whether the decision tree that we're going to have to build is going to be an easy one. That is something linear, say. Or a hard one, something that's exponential. So this brings me to a key question that I want to ask, which is, exactly how expressive is a decision tree. And this is what I really mean by this. Not just what kind of functions it kind of represent. But, if we're going to be searching over all possible decision trees in order to find the right one, how many decision trees do we have to worry about to look at? So, let's go back and look at, take the XOR case again and just speak more generally. Let's imagine that we once again, we have n attributes. Here's my question to you, Michael. How many decision trees are there? And look, I'm going to make it easy for you, Michael. They're not just attributes, they're Boolean attributes. Thanks. Okay? And they're not just Boolean attributes, but the output is also Boolean. Got it? Sure. But how many trees? So it's, I'm going to go with a lot. Okay. A lot. Define a lot. Define a lot. So, alright, well, there's n choices for which node to split on first. Yeah. And then, for each of those, there's n minus 1 to split on next. So I feel like that could be an n factorial kind of thing. Maybe. I like that. And then, even after we've done all that, then we have an exponential number of leaves. And for each of those leaves, we could fill in either true or false. So it's going to be exponential in that too. Hm, so let me see if I got that right. So you said we have to pick each attribute at every level. And so you see something that you think is probably going to be, you know? Some kind of commutatorial question here. So, let's say n factorial, and that's going to just build the nodes. That's just the nodes. Well, once you have the nodes, you still have to figure out the answers. And so, this is exponential because factorial is exponential. And this is also exponential. Huh. So let's see if we can write that down. So let me propose a way to think about this, Michael. You're exactly right the way you're thinking. So, let's see if we can be a little bit more concrete about it. So, we have Boolean inputs and we have Boolean outputs, so this is just like AND, it's just like OR, it's just like XOR, so, whenever we're dealing with Boolean functions, we can write down a truth table. So let's think about what the truth table looks like in this case. [SOUND] Alright, so, let's look at the truth table. So what a truth table will give me is, for, the way a truth table normally works is you write out, each of the attributes. So, attribute one, attribute two, attribute three, and dot dot dot. And there's n of those, okay? We did this a little earlier. When we did our decision tree. When we tried to figure out whether I was on a hot date or not. And then you have some kind of output or answer. So, each of these attributes could take on true or false. So one kind of input that we may get would be say all trues. Right? But we also might get all trues, except for one false at the end. Or maybe the first one's false and all the rest of them are true, and so on, and so forth. And each one of those possibilites is another row in our table. And that can

just go on for we don't know how long. So we have any number of rows here and my question to you is how many rows? Go.

1.1.20 Decision Tree Expressiveness Quiz Solution

So, we're back. What's the answer Micheal? How many rows do we have? So if it was just one variable we're splitting on, then it need to be true or false, so, that's two rows. If it was two variables, then there's four combinations and three, would be eight, combinations. So, generalizing the end, it ought to be 2 to the n. That's exactly right, there are 2 to the n different rows. And, that's what always happen when we're dealing with n, you know, n attributes, n boolean attributes. There's always 2 to the n possibility. Okay, so I get just halfway there and I get to your point about, combinatorial choices, among the attributes. But ,that's only the number of rows that we have. There's another question ,we need to ask which is, exactly how big is the truth table itself?

1.1.21 Decision Tree Expressiveness Quiz 2 Question

Alright, so here's the question for you. We know we have 2DN, different possible instances we might see. That is two to the end, different ways we might assign different values to the attributes. But, that still doesn't tell us how many decision trees we may have, or how many different functions we might have. So, if we have 2DN rows, here's my question. How many different ways might we fill out. This column over here of outputs? Remember, an output can be either true or false. Go.

1.1.22 Decision Tree Expressiveness Quiz 2 Solution

Okay, Michael, what's your answer? Alright. So. Again, a lot feels like a good answer, it's already written down on the left. But it's also wait, wait, may be we can quantify this. So if it were. Maybe one way to think about this is if each of the, each of those empty boxes there, is either true or false. It's kind of like a bit. And we're asking how many different bit patterns can we make? And in general, it's two to the number of positions, but here the number of positions is 2 to the n. So it ought to be 2, to the 2 to the n. Which is that the same as 4 to the n? No. Okay. But you're right. It's 2 to the 2 to the n. So it's a double exponential and it's not the same thing as 4 to the nth. It's actually 2 to the 2 to the nth. Now how big of a number do you think that is Michael? I'm going to go again to my go to place and just say a lot. It is, in fact, a lot and I'm going, I actually, I'm going to look over here, and I'm going to tell you. That for even a small value of n, this gets to be a really big number. So for, for one, it's 2 to the 2 to the 1, which is 4. That's not a big number. That's true. What about two? For two, it's 2 to the 2 to the 2. So 2 to the 2 is 4, so it's 2 to the 4, which is 16. What about three? Alright, so that's two to the 8th, which is 256? Mm-hm. So that's growing pretty fast, don't you think? Sure, but those aren't big numbers yet. What if I told you, [LAUGH] that for n equals 6, 2 to the 2 to the n was, I'm going to start writing it, okay? 18466744073709551616. Holy monkeys. Yes, that is in fact the technical term for this number, it's a holy monkey. It is a very, very big number. 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. So what's the point of this exercise, Michael? It's, it's to point that the space of decision trees, the hypothesis space that we've chosen, is very expressive because there's lots of different functions that you can represent. But that also 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, billion upon, billion of possible decision choice.

1.1.23 ID3

So now, we have an intuition of best, and how we want to split. We've, we've looked over, Michael's proposed, the high-level algorithm for how we would build a decision tree. And I think we have enough information now that we can actually do, a real specific algorithm. So, let's write that down. And the particular algorithm that Michael proposed is a kind of generic version of something that's called ID3. So let me write down what that algorithm is, and we can talk about it. Okay, so 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, we might define best in a moment. 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, and 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. And the key bit that we have to expand upon in this case, is exactly what it means to have a best attribute. All right so, what is exactly, what exactly is it that we mean by best attribute? So, there are lots of possibilities, that you can come up with. The one that is most common, and the one I want you to think about the most, is what's called information gain. So information gain is simply a mathematical way to capture the amount of information that i want to gain by picking particular attribute, funnily enough. 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. So the formula's simply this. 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, sort of, the expected or average entropy that you would have over each set of examples that you have with a particular value. Does that make sense to you, Michael? So what we're doing, we're picking an attribute and that attribute could have a bunch of different values, like true or false, or short, medium, tall? Right. And. And that's represented by v. Okay, each of those is a different v. And then we're saying okay, for over those leaves, we're going to do this entropy thing again. Mm-hm. And we right. So what, and what is entropy? entropy. So, we'll talk about entropy later on in the class in some detail and define it exactly and mathematically. And some of you probably already know what, what entropy is, but for those of you who don't, it's 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 versus tails? A half. 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? It's. One. One. So that actually has no information, no randomness, no entropy whatsoever. And has zero bits of entropy. So, when I look at this set of examples that I have, and the set of labels I have, I can count the number that are coming up, let's say, red x's. Versus the ones that are coming up green o's. And if those are evenly split, then the entropy of them is maximal, because if I were to close my eyes and reach for an instance, I have no way of knowing beforehand whether I'm more likely to get an x or I'm more likely to get an o. On the other hand, if I have all the x's in together, then I already know before I even reach in that I'm going to end up with an x. So as I have more of one label than the other the amount of entropy goes down. That is, I have more information going in. Does that make sense, Michael? I think so. So, is there, can, maybe we can say what the formula is for this, or, or? Sure. What is the formula for it? You should remember. It's, if we have, well, I'm not sure what the notation ought to be with these S's but it has something to do with Log P log, no wait, it's P(log)P. Mm-hm. So the actual 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. And I don't want to get into the details here. We're going to go into a lot more details about this later when we get further on in the class with randomize optimization, where entropy's going to matter a lot. But for now, I just, you have, I want you to have the intuition that this is a measure of information. This is the measure of randomness in some variable that you haven't seen. It's the likelihood of you already knowing what you're going to get if you close your eyes and pick one of the training examples, versus you not knowing what you're going to get. If you close your eyes and you picked one of the training examples. Okay? Alright. So, well, so, okay, so then in the practice, trees that you had given us before, it was the case that we worked, we wanted to prefer splits that I guess, made things less random, right? So if things were all mixed together, the reds and the greens, after the split if it was all reds on one side and all greens on the other. Then each of those two sides would have very, what? They would have very low entropy, even though when we started out before the split we had high entropy. Right, that's exactly right. So if you, if you remember the, the, three examples before. One of them, it was the case that all of the samples went down the left side of the tree. So the amount of entropy that we had, didn't change at all. So there was no gain in using that attribute. In another case, we split the data in half. But in each case, we had half of the x's and half of the o's together, on both sides of the split. Which means that the total amount of entropy actually didn't change at all. Even though we split the data. And in the final case, the best one, we still split

the data in half, but since all of the x's ended up on one side and all of the o's ended up on the other side, we had to entropy or no randomness left whatsoever. And that gave us the maximum amount of information gain. So is that how we're choosing the best attribute? The one with the maximum gain? Exactly. So the goal is to maximize over the entropy gain. And that's the best attribute.

1.1.24 ID3 Bias

So, we've got a whole bunch of trees we have to look at, Michael. And were going to have to come up with some clever way to look through them. And this get's us back, something that we've talked about before, which is the notion of bias. And in particular, the notion of inductive bias. Now, just as a quick refresher, I'm want to remind you that there is two kind of biases we worrying about when we think about algorithms that are searching through space. One is what's called a restriction bias. The other is called preference bias. So a restriction bias 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. Okay? That means we're not considering, y equals 2x plus 3. We're not considering quadratic equations. We're not considering non-boolean functions of a certain type. We're only considering decision trees, and all that they can represent. And nothing else. Okay? So that's already a restriction bias and it's important. Because, instead of looking at the infinite number uncountably infinite number of functions that are out there, that we might consider. We're only going to consider those that can be represented by a decision tree over in, you know, all the cases we've given so far discreet variable. But a preference bias is something that's just as important. And it tells us what source of hypotheses from this hypothesis set we prefer, and that is really at the heart of inductive bias. So Michael, given that, what would you say is the inductive bias of the ID3 algorithm? That is, given a whole bunch of decision trees, which decision trees would ID3 prefer, over others? So, it definitely tries, since it's, since it's making it's decisions top down. 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. Good point. So good splits near the top. Alright. And you said something very important there Michael. Given two decision trees that are both correct. They both represent the, the function that we might care about. It would prefer the one that had the better split near the top. Okay, so any other preferences? Any other inductive bias on the, on the ID3 algorithm. It prefers ones that model the data better to ones that model the data worse. Right. So this is one that people often forget: it prefers correct ones to incorrect ones. So, 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. So that's really, 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. So, if you go back to the example where we went before, where one of the attributes doesn't split the data at all, that is not something that ID3 would go for, and it would in fact create a longer and unnecessarily longer tree. So it tends to prefer shorter trees over longer trees. So long as they're correct and they give you good splits near the top of the tree.

1.1.25 Decision Trees Continuous Attributes

Alright. So, we've actually done pretty well. So through all of this, we finally figured out what decision trees actually are. We know what they represent. We know how expressive they are. We have an algorithm that let's us build the decision trees in an effective way. We've done just about everything there is to do with decision trees, but there is still a couple of open questions that I want to think about. So, here's a couple of them and I want you to, to think about and then we'll discuss them. So, so far all of our examples that we've used. All the the things we've been thinking about for good pedagogical reasons. We had not only discreet outputs but we also had discreet inputs. So one question we might ask ourselves, is what happens if we have, continuous attributes? So Michael, let me ask you this. Let's say we had some continuous attributes. We weren't just asking whether someone's an animal or whether they're human or whether it's raining outside or we really cared about age or weight or distance or anything else that might have a continuous attribute. How are we going to make that work in a decision tree? Well, I guess the literal way to do it would be for something like age to have a branching factor that's equal to the number of possible ages. Okay, so that's one, one possibility. So we stick in age and then we have one. 1.0, we have one for 1.1, we have one for 1.11, we have one for 1.111 Ahh, I see. Alright. Well, at the very least, okay. Heheheh. What

if, what if we only included ages that were in the training set? Presumably there's at least a finite number of those. Oh, we could do that. We could just do that, except what are we going to do then when we come up with something in the future that wasn't in the training session. Oh, right. Can we look at the testing set? No were not allowed to look at the testing set. That is cheating, and not the kind of good cheating that we do when we pic good representation. Okay, fair enough. Well we could, we could Ranges. What about ranges? Isn't that the way we cover more than just individual values? Give me an example. Say ages you know, in the 20s. Okay, so, huh. How would we represent that with a decision tree? Let's say in the 20s. Age. How we do that. You could do like age, element sign, bracket. 20. 20 comma 21, or, or 29 or 30 right per end. Yeah it's too much. Why don't I just say age Is between less is, let's see, greater than or equal to, 20 and, less than 30. And just draw a big oval for that. Alright? So that's a range, so that's all numbers between, 20 and 30 inclusive of 20 but not 30 Right. Yeah. And what's good about that is that's a question. You are either in your 20s or you are not. So, the output of that is actually true or false. So, I guess the good news there is that now we know how to evaluate attributes like that because we have a formula from ID3 that tells you what to do. But seems like there's an awful lot of different ones to check. Right, and in fact if it's truly a continuous variable, there are in principal an infinite number of them checked. But we can do now the sort of cheating you wanted to do before. We can just look at the training set, and we could try to pick questions that cover the sorts of data in the training set. So, for example, if all of the values are in the 20s, then there is no point of even asking the question. You will start just split instead upon values that were, say less than 25 or greater than 25, and you could imagine all kinds of ways where you might do that. You might look at all of the values that show up in the training set, and say well, I am going to do a binary search. So, I am just going to create an attribute for Less than half of whatever is in the training set or greater than half of whatever the range is in the training set. Does that make sense? Yeah, that's clever. Right. Thank you. I just made that up on the spot. Okay, so you do those sorts of things and that's how you would deal with continuous attributes. That brings me to a next question, I'm going to actually do this as a quiz because I want an answer from our audience.

1.1.26 Decision Trees Other Considerations Quiz Question

So, here's the next question I want to ask you, simple true or false question. Does it make sense to repeat an attribute along any given path in the tree? So, if you we pick some attribute like A, should we ever ask a question about A again? Now, I mean something very specific about, by that. I mean, down a particular path of the tree, not just anywhere else in the tree. So, in particular, I mean this. So, I ask a question about A, then I ask a question about B, and then I ask a question about A again. That's the question I'm asking. Not whether A might appear more than once in the tree. So, for example, you might have been the case where A shows up more than once in the tree, but not along the same path. So, in the second case over here, A shows up more than once, but they really don't really have anything to do with one another because once you've answered B, you will only ever ask the question about A once. So, my question to you is, does it make sense to repeat A more than once along a particular path in the tree? Yes or no? Answer the question.

1.1.27 Decision Trees Other Considerations Quiz Solution

Okay, Michael, what's the answer? So, alright. Does it make sense to repeat, an attribute along a path in the tree? So, it seems like it could be no, [SOUND] in that, you know, if we're looking at attributes like, you know, is a true, then later we would ask again is a true because we would already have known the answer to that. Right, and by the way, information gain will pick that for you automatically. It doesn't have to be a special thing in the algorithm, if, if, you consider, an attribute that you've already split on, then you're not going to gain any information, so it's going to be the worst thing, to split on. Exactly. Alright, but it, Okay, doing good. But it seems like maybe you're trying to lead us on because ,this we're in the continuous attributes portion of our show. Okay, well what's the answer there? Is the answer not also false? Well we wouldn't want to ask the same question, about the same attribute. So, we wouldn't have age, between 20 an 30, and then later ask again, age ,between 20 and 30. But ,maybe we want to ask, you know, given that we are less than 20, we're, are we teenagers or not, so we might have a different range, on age later in the tree. So, that's exactly right, Michael. So, the answer is no, it does not make sense ,to repeat an attribute along a path of the tree, for discrete, value trees. However, for continuous [UNKNOWN] attributes, it does make sense. Because, what you're actually doing, is asking a different question. So, one way to think about this, is that the question is age in the 20's or not. Is actually ,a discreet valued attribute that you've just created, for the purposes of the decision tree. So, asking that question doesn't make sense but asking a different question, about age, does in fact make sense. So ,once you know, that you are not in the 20's you

might ask well am I less than, 20 years old? Maybe a teenager or am I greater than 40. How old am I, 44? Greater than 44, in which case, I'm old. So if it's, if you ask, [LAUGH] the tree that you drew isn't quite right though, right? Yeah, it is. because, if you go down the false branch, it means you are less than 20. No, I can be greater than 30. Oh, good one. Yes, I'm very clever, or at least I accidentally got it right. One of the two, it's the same thing. Okay, so there you go.

1.1.28 Decision Trees Other Considerations

So, we've answered the thing about continuous attributes. Now, here's another thing. When do we really stop? When we get all the answers right. When all the training examples are in the right category. Class. Right, so the the answer in the algorithm is when everything is classified correctly. That's a pretty good answer, Michael. But what if we have noise in our data? What if it's the case that we have two examples of the same object, the same instance, but they have two different labels? Then this will never be the case. Oh. So, then our algorithm goes into an infinite loop. Which seems like a bad idea. So we could have, we could, we could just say, or we've run out of attributes. [LAUGH] [LAUGH] Or we've run out of attributes. That's one way of doing it. In fact, that, that was, that's going to have to happen at some point, right? That's probably a slightly better answer. 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. Don't you think? Hm. So, what got us down this path, was thinking about what happens if we have noise. Why would we be worried about having noise anyway? Noise anyway. Well, I guess the training data might have gotten corrupted a little bit or maybe somebody copied something down wrong. Right, so since that's always a possibility, does it really make sense to trust the data completely, and go all the way to the point where we perfectly classify the training data? But Charles, if we can't trust our data, what can we trust? Well, we can trust our data, but we want to verify. [LAUGH] The whole point is generalization. And 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, right? I guess so. So, what will we do? [LAUGH] I mean, we actually have a name for this, right? When you get really, really, really good at classifying your training data, but it doesn't help you to generalize, we have a name for that. Right. That sounds like overfitting. Exactly. We have to worry about overfitting. So you can overfit with the decision tree too? Yeah. What, you don't believe that? No, no, no. I was, I was being naive, I was being, I know that you can overfit a decision tree. [LAUGH] I was just. [LAUGH] Yeah but your [SOUND] is the [SOUND] that you use when you're, when you're like, I don't believe what you just said Charles, but I'm going to go along with it anyway, because I have to get off the phone soon. [LAUGH] Fair enough. I'll try to, I'll try to have a different personality then. [LAUGH] Okay, step one, have a different personality with maximal information gain. Okay, so we don't want to, we don't want to overfit. So we need to come up with some way of overfitting. Now the way you overfit in a decision tree is basically by having a tree that's too big, it's too complicated. All right. Violates Occam's Razor. So, what's a kind of, let's say, modification to something like ID3 to our decision tree algorithm that will help us to avoid overfitting? Well last time we talked about overfitting, we said cross-validation was a good way of dealing with it, which, it allowed us to choose from among the different, say degrees of the polynomial. Right. So maybe we could do something like that? I don't know. Try all the different trees and, see which one has the lowest cross validation error? Maybe there's too many trees. Maybe, but that's a perfectly reasonable thing to do, right? You take out a validation set. You build a decision tree, and you test it on the, on the validation set and you pick whichever one has the lowest error in the validation sect, that's one way to avoid it. And then you have, don't have to worry about this question about stopping, you just grow the tree on the training set minus the validation set until it does well on that. And you check it against the crossvalid, you check it against the validation set, and you pick the best one. That's one way of doing it, and that would work perfectly fine. There is another way you can do it that's more efficient. Which is, you do the same idea validation, except that you hold out a set and as you, everytime 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. So is there, is there a problem in terms of, I mean if we're expanding the tree depth for search wise, we could be at, you know, we could be looking at one tiny little split on one side of the tree before we even look at any, anything on the other side of the tree. That's a fine point. So how would you fix that? Maybe expand breadth first? Yeah, that would probably do it. Anything else you could think of? Well, so, you could do pruning, right? 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. So, that whole class of ways of doing it, is called pruning. And 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.

1.1.29 Decision Trees Other Considerations Regression

So another consideration we might want to think about with decision trees but you're not going to go into a lot of detail but I think might be worth at least mentioning is the problem of regression. So, so far we've only been doing classification ,where the outputs are discreet, 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? Any ideas Michael? So these are now continuous outputs, not just continuous inputs. Right, maybe the outputs are all continuous, maybe the outputs are discrete, maybe they're a mix of both. Well it certainly seems like out rule of using, information gain is going to run into trouble because it's not really clear how you measure information on these continuous values. So, I guess you could measure error some other way. Well we're not, it's not, it's not error right it's tryin to measure how mixed up things are? Oh so ,maybe something like variance? Cause in a continuous space you could talk about you know, if there's a big spread of, in the values that, that would be measured by the variance. Oh good. So what you really have now is a question about splitting. What's the splitting criteria? Maybe [CROSSTALK] I guess there's also an issue of, of what you do in the leaves. Right. So, what might you do in the leaves? I guess you could do some sort of more standard kind of fitting algorithm. So, like, report the average or, or do some kind of a linear fit. [SOUND] Is any number of things you can do. By the way ,that's worth pointing out on the, on the output that if we do pruning like we did before, we have errors, we did actually say when we talked about that how you would report an output. Right? If you don't have a clear answer where everything is labeled true or everything is labled false, how do you pick? So something like an average would work there. I don't know, I mean, it seems like it depends on what we're trying to measure with the tree. If the tree is, we're trying to get as many right answers as we can, then you probably want to do like a vote in the leaves. Right, which ,at least, if the only answer is true or false, that would look more like an average I guess. Right, so you pick, you do a vote. So we do a vote, so we do pruning. We do have to deal with this issue of the output. Somehow ,and something like a vote mixing. And here, when you have a regression, then I guess average is a lot like voting. Yeah, in a continuous phase. Yeah. So either way we're doing a kind of voting. I like that.

1.1.30 Decision Trees Wrap up

Hi Michael, so that covers Decision Trees Excellent. So, since you are the one who is listening, you get to tell me what we have learned today? Well, we learned about the Decision Tree representation, we learned the top down algorithm for inducing a Decision Tree. And we call that ID3 All right. So we got a representation, we got a top down learning algorithm ID3. We learned about the, the expressiveness and the bias. Right. So those are 2 separate things, we learned about the sort of expressiveness. And we learned about the bias of ID3. And we gave one, so is this specific to ID3? We, we looked at one specific way of deciding on splits, which was to do this maximum information game. Right. So we talked about in general,um, what are good attributes or what are best attributes. So, information gain is one way of doing it. As one example. And, by the way, this notion of best attribute is something we'll end up returning to sometimes explicitly, and sometimes implicitly, throughout the entire course. And lastly, I feel like we talked about the problems with over-fitting and how in the Decision Tree context, you can prune back the tree to avoid over-fitting. Over-fitting is an issue. Over-fitting is always an issue and we came up with a couple of strategies for dealing with over-fitting in the context of Decision Trees. Okay! So we've learned everything there is to know about Decision Trees, there's nothing else to know. [LAUGH] Somehow I find that hard to believe. Yeah, there's a lot there and the students will get a chance to learn even more as they do the assignments. Cool. Excellent. All right Michael, thank you. Sure, look forward to the next chat.

1.1.31 What is Regression Question

Hey Charles, how you doing? I'm doing just fine Michael, how are you doing? I'm doing pretty well, thanks. I'm happy to get a chance to tell you about something today. Excellent, and what is it you're going to tell me about? We're going to talk about regression. Like, progression? [LAUGH]. No, regression. So, let

me tell you about regression. So we are, in this section of the class, talking about supervised learning. And Supervised learning, 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, right. So this, this covers all of this, this the things we are talking about in the context of supervised learning, right. Right. Now, what makes regression special subtopic. We are going to be talking about mapping continuous inputs to outputs. As opposed to, what was the other thing that we were mapping, what other kinds of outputs did we think about? Well, we had discrete outputs and continuous outputs. Right, and so this is going to be the focus on continuous. So regression seems like sort of an odd word. It doesn't really kind of fit for this. So often I think about regression as. So this is, this is me being all sad and sort of reverting back to a childhood state. And that's, you know, that's in the psychological sense, that's what regression refers to. But it turns out that, that's not what it means in this setting. But the story by which those things became linked, I think, is kind of interesting. So let me tell you about that. Okay. So, this is a picture of you Charles. [LAUGH] Okay. I'll accept that. You can tell it's you because he's really tall. And you're, you're a fairly tall man. I know you don't think of yourself that way, but you think of everyone else as being short which is really the same thing. Fair enough. Alright, so let's say that this is Charles. Let's say that this is someone of average height. Just someone at random. Mm-hmm. So now, let's pretend, Charles, that you have children. I do have children. All right. So let's, that's okay but we can just pretend, and we want to ask the question what would you expect the average height of your children to be? Would you expect it to be sort of, you know, sort of Charles' height? Or average height or may be somewhere between. So let's let's actually ask this as a quiz.

1.1.32 What is Regression Solution

Okay, Charles, so what do you, what do you think about this? Wait, how old are my children? Let's say what their adult height is going to be. I would expect them to be a little bit smaller than me. A little bit smaller than you? Mm-hmm. So, so their, the average height, their average height of your children, you would you say it would be like an average height person? Or like your height or sort of in between? In between. In between. All right. So it turns out that if you actually do this, you measure people's heights and you measure the heights of their children, that that is in fact what you tend to see. That very, very tall people, like you tend to have taller than average children. But the height is between. It actually regresses to the mean. And here we really do mean regresses in the sense of going back to this kind of more primitive state that, if you think about average height people, as being like your ancestors, then, you know, you as a, as a very tall person tend to have kids that that tend regress back toward that average value that sort of, more older, more ancient value. So does that that make some sense to you? That makes some sense. But one comment and one question. Comment, that is awesome because I've always actually wondered what regresses to the mean actually means. The second, what prevents us from all being the same height then? Yes, so what, what seems to be happening is that there's a kind of a noisy process and some people turn out to be taller, but then the, then the next generation there's a little bit of a history effect in people stay taller, but it tends to drift back towards the mean. So it's, so it's, it's sort of like a random walk, to some extent. Oh, that actually kind of makes sense.

1.1.33 Regression and Function Approximation

Alright, so what does this have to do with function approximation or regression. So how does this notion of regression of falling back toward the mean have to do with this idea of approximating functions, mapping inputs to outputs, it seems kind of odd. So it turns out that the relationship is, here's the, here's the connection between them. 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. So if we plot these against each other, let's let 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. And 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. Right. If the slope of this line was one, what would that mean Charles? That would mean that everybody's children had the, would, the same height of their parents. Right, right, and so that's exactly right, and so but if this slope is less than one, like it is, it turns out to be in, in real populations. Then what's happening is the children are little shorter than the

parents. Children of taller parents are shorter than they are. And the children of short parents are taller than they are. And that's the fact that this is less than one is what makes it regression to the mean. Now this, this was worked out in I believe in the late 1800s and it was just such a beautiful way of connecting all these different quantities together. To kind of think of them as being related in this functional way. That people said, oh this is really great. I'm going to use this idea of regression. And what they started to mean actually was this not this idea of regression to the mean. But this idea of finding a, a mathematical relationship based on a bunch of measurements of points. So this term ended up getting misused. But that's the term that we have now. So regression now refers to not this idea of collapsing back towards, towards the mean, but, the idea of using functional form to approximate a bunch of data points. Isn't that weird. That's pretty cool. There's another example of this sort of idea where where a reasonable word, like, like regression which we're referring to some physical thing in the, in the world due to experiments like psych experiments at this point became this mathematical concept where the name doesn't really fit anymore, like there isn't really anything regressing in what we're doing. Mm-hm. There's another, really important example that we're going to get to the later in the course. Do you, do you know what I'm thinking of Charles? No. So reinforcement learning is my field of study. And often your field of study. Often. Often. And it turns out that reinforcement learning doesn't mean what the words mean anymore. That this was a concept that the psychologist used to explain what they were observing. And then some mathematicians, well let's call them computer scientists, took the word themselves, started to use it and used it wrong, but now it stuck. [LAUGH] And regression is another example like that. They, the word is sort of being used wrong, but it stuck and that's what we're going to use. This explains why every time I have a conversation with a psychologist about reinforcement learning, we talk past each other. Yes, they get very confused. I tried it to tell them upfront that's not really what I mean, but I'm going to use the words anyway, but it still confuses them. Hmm.

1.1.34 Linear Regression

Alright, so, 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. So what I'd like to do now is to step through an example Of doing some regression and to point out what some of the pitfalls are and how they're generally handled in the machine learning context. So, this graph that I put up here, is, we just made these numbers up, but it's supposed to tell us a, a little bit about housing prices. So 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, right. So ,the square footage of the house can vary. And in this case the houses that I visited can be between, about 1,000 to 10,000 square feet. And of course, as you get bigger houses, you tend to get more, the, the prices tend to go up, too. Alright, so the price that the house cost is, tends to rise with the size of the house. So, 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? I don't know what year this happened in. And we end up with a house that is 10,000 square feet and cost about $6,000. Again, I don't. This is not true in Providence Rhode Island, I'll tell you that. Are you sure? Yeah, I'm pretty sure. Yeah, it's really not true in Atlanta Georgia. So Alright... So, so, so imagine that this is the relationship we observe. But now we want to answer a question like, Well, 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? So what do you, what do you think, Charles? Looking at this, what do you think a fair price for a 5,000 square foot house would be? Apparently about $5,000. About, $5,000. Right. So, how did you do that? I looked at the graph, I went over to 5,000 square feet at the x-axis and I went up. Until I found ,where one of the x's was on the y axis and I said, oh, that's about 5,000 square feet. Well, but there was no corresponding point for that, so you had to interpolate or something ,uh, based on the points that were there you had to kind of imagine what might, might be happening at the 5,000 square foot mark, right? That's true, although this one was a little easy because at 4,000 and 6,000 square feet, they were almost exactly the same. Mm, and so that, to you, made it feel like there was probably ,um, that's probably the level where things in this range would be. Yeah. Okay. Alright, that seems kind of reasonable. So sure, though what we're going to do in this case is actually try to find a, a function that fits this. Mm-hm. Alright ,so what we can do is actually say, well what if there is a linear relationship. What would be the best linear function that captures the relationship between the size and the cost. So ,what I have here is, it turns out of all the possible linear functions, this is 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, it's interesting right, because if you put your idea of 5,000 square feet. Look what this line predicts. It's something more like $4,000, right. Do you see that? I do. That is doesn't seem right to me. It doesn't, yeah, 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. That's true.

1.1.35 Find the Best Fit Question

Alright. So it's worth asking, how do we find this best line? So again there's an infinite number of lines. How do we find one that fits these points the best. And again we're defining best fit. As the one that has the least squared error, where the error is going to be some of the distances between these x points and the green line that we, that we fit. I'm not even sure that this really is the best fit in this case. I just kind of hand drew it. So, okay. So is this something that we, that we would want to solve by hill climbing which is to say, we kind of pick the. The slope and the intercept of the line, and we just kind of try different values of this until it gets better and better and then we can't get any better, and we stop. Can we do this using calculus? Can we use random search, where we just like, pick a random M, pick a random B, and see if we're happy with it? Or is this the sort of thing where we probably would just go and ask a physicist because it involves, like, continuous quantities, and we're discrete people?

1.1.36 Find the Best Fit Solution

And that's the correct answer. All right. So let's actually go through that exercise and derive how we do that. because it's not so bad in two dimensions and it generalizes to higher dimensions as well. Okay. So it turns out that we can use calculus to do this, I am not going to step through the two-variable example for reasons that I am embarrassed to say. But I am going to show you a different example. So 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, right? So the best function that has the form, the value of the function for any given X is always the same constant, C. So if our data looks like this, we got a bunch of X's and a bunch of Y's, then what we're going to do, we're going to say for any given value of C, any given constant, we can have an error. What's the error going to be? The error is going to be the sum over all of the data points. Speaker 1: The square difference between that constant we chose and what the actual yi value is. So that's why. Michael. These differences here. Yes, Charles. Can I ask you a question? Sure. Why are we doing sum of squares? There is many different error functions and sometimes called a, a relative concept called the loss function. There is lots of difference once that could work, you can do the absolute error, you can do the squared error, you can do various kinds of squashed errors where you know. The errors count different depending on how, how much deviation there is. It turns out that this one is particularly well behaved because of this reason that I'm explaining now that that because this error function is smooth as a function of the constant C, we can use calculus to actually find the minimum error value. But there's lots of other things that could work and they actually do find utility in various different machine learning settings. Okay. So just now using the chain rule, if you want to find how do this error function output change as a function of input c. We can take the derivative of this sum you know, bring the two over. Times this, times the derivative of the inside, which is negative one in this case. And now this gives us a nice, smooth function saying what the error is as a function of c. And if we want to find the minimum, what do we want to do to this quantity? Set it equal to zero, because that's what I remember from Calculus. That's right. So in particular if the error you know, the error function is a nice smooth thing the derivative is negative and then zero and then positive. When it hits zero that's when the thing has bottomed-out. Alright. So now we just need to solve this, this equation for c. So we have one equation and one unknown. Alright, so that gets us this. But, this quantity, it's just the constant added to itself n times. So it's n times c. We move that to the other side. We get n times c. N is the number of data points as you recall. Is the sum of the yi's. We divide two by n and what do we see? So what is it Charles? The best constant is the average of all your y's. Great, it's the mean. The mean comes back. Right, so in the case of finding the best constant here, we just have to average the y, the y's together and that catches thing that minimizes the squared air. So squared air 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, higher order of function tier, not higher functions, but more variables like, like lines. Sorry. Lines that have some, some non constant slope. By doing the same kind of process and things actually work really nicely.

1.1.37 Order of Polynomial

Alright, so now let's, let's get back to our data set that we were looking at before. So again, the ideas that we're, we're going to try to find a way of predicting the value for various points along the way on this curve. And 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. Alright. 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. Would it probably be a good choice at this particular case? Yes. It does seem like it's got, sort of, curvy downy nature, right? Mm hm. 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. So let's take a look at that. Let's take a look at the. The best parabola to fit this. Alright, so, so here we go. We've got the, 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 and look at it, it does, it does a nice job, right? Kind a gets, gets tucked in with all those other points. so, so what do you think? Is this the best way of, of capturing this. This particular set of points? Well, if the only thing we care about is minimizing the sum of squared error, my guess is that the parabola has less squared error. Yeah. It ha, 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, to zero. So if the best fit to this really was a line. The best fit to this data point was a line, then the parabola that we see here wouldn't have any curve to it. So, yeah. Our arrows going down. Hm, As we have gone from order zero to order one to order two. So can you think of any other way getting there in order to getting down even more. How about order 14 million. Interesting, while in this particular case, given the amount of data that we have, we can't go past the number of data points, yeh after that. They're really unconstrained. Okay. Then how about order nine? Order nine is a good idea. But just to give you an idea here, we're going to step up a little more. This is order four and look at, look at how lovely it can actually capture the flow here. That's, very faded. Order six and in fact the best we can do here is of the, of the, sorry. The most, the highest order that, that works is order eight. And, son of a gun, look what it did. It hit every single point dead on in the center. Boom. Boom. Boom. Boom. It used all the degrees of freedom it had to reduce the error to essentially zero. Excellent. So [LAUGH], one could argue that this is a really good idea. Though, if you look at what happens around 9000, there's some craziness. Do you see that? I do. At the Yeah, the To try to get this particular parabola to hit that particular point, it sent the curve soaring down with an up again. We also did that between 6500 and 8500, it sent [CROSSTALK]. Yes good point [UNKNOWN] [CROSSTALK] Right, right, that's right went off the top of the plot. So Yeah, that's kind of [INAUDIBLE]. But let's just, just to show that we really are, as we have more degrees of freedom we're fitting the error better. Let me show you what it looks like, 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 its kind of a good, nut it doesn't feel quite right like the curves that we're looking there looked a little bit crazy.

1.1.38 Pick the Degree Question

All right, so let's, let's do a quiz. Give you a chance to kind of think about what where are these trade-offs are actually going to be. So we're going to pick the degree for the housing data, and your choices are going to be the degree zero, one, two, three, or eight. So a constant, that's the first choice. Or a line that has some slope that, you know, sort of increases with the data, that's your second choice. Or it could be we use a degree two parabola. So sort of goes up and then levels off. Or you can it might be a little hard to see but here's a cubic that that goes up flattens out a little bit and then rises up again at the end. Or we could go with the full monty, the octic. You can see that might not be spelled correctly. That actually has enough degrees of freedom that it can hit each of these points perfectly. Like that. The authority line. [LAUGH] Good, you got that in.

1.1.39 Pick the Degree Solution

So Charles, how would we go about trying to figure this out? How would we go about trying to figure this out? Yeah, what do you think? Which one would you choose and how would you choose? so, well

that's a good question. Well just given what you, what you've given me, I'm going to ask. I think smartly guess, that probably k equals 3, is the right one, and K equals 3 And I'll tell you why. It's because zero, one and two seem to make quite a few errors. Mh-hm Three does a pretty good job but doesn't, doesn't over commit to the data. Hm. And that's the problem with eight, is that eight says, you know, the training data that I have is exactly right and I should been and moved heaven and earth in order to, to match the data. And that's probably the wrong thing, certainly if there's any noise or, or anything else going on in the data. Right. So it sort of seems like it's overkill, especially that it's doing these crazy things between the points. Whereas the cubic one, even though it clings pretty close to the points, it stays between the points, kind of between the points. Yeah. Which seems like a really smart thing. So yeah so, so that turns out to be the right answer but let's actually let's actually evaluate that more concretely.

1.1.40 Polynomial Regression

Alright. So we talked through how it works when you've got you're trying to fit your data to a constant function, to a zero order polynomial. But let's, let's at least talk through how you do this in the more general case. This is, this is what I've been doing to, to fit various curves to the data at least implicitly. So, what we're really trying to do is we've got a set of data, x and y. Set n, n examples of x's and their corresponding y's. And what we're trying to find is these coefficients, C0, C1, C2, C3. Let's say if we're trying to do cubic regression where C0 gets added to C1 times x, which gets added to C2 times x squared. Which gets added to C3 times X cubed and we're trying to get that to look a lot like y. Now we're not going to get to exactly equal y but let's pretend for a moment that we could. We have a bunch of these examples and we want it to work for all of them. So we can arrange all of the, all these constraints, all these equations into matrix form. If you're familiar with linear algebra. So the way that we can write this is here are the, here are the coefficients that we're looking for, the C's, and here are what we're going to multiply them by. We're going to take the X one and look at the zeroth power, the second power, the third power. And that equation I'll use my hands cause that's I always, I always need to use my hands when I do matrix multiplication. So you're going to across here and down there to multiply these and add. And that needs to correspond to y1. And same thing this now the second row. Multiplied by these coefficients. Need to give us our y2 and so forth. Alright. So if we arrange all these x values into a matrix, and we'll call it, you know, x. And then we have these other guys. And we'll call this w, like the coefficents. Obviously w stands for coefficent. And we want that to sort of equal This vector of y's. And we basically just need to solve this equation for the w's. Now, we can't exactly solve it because it's not going to exactly equal, but we can solve it in a least squares sense. So let me just step through the steps for doing that. Alright, so let's, so here's how we're going to solve for w. So what we're going to do is premultiply by the transpose of x. Both sides. I mean really what we wanted to do at first is if we are solving for Y, we need to multiply by the inverse of X, but this isn't really going to be necessarily well behaved. But if we pre mulitplied by the X transpose then this thing is going to have a nice inverse. So now we can pre multiply by that inverse. All right. Now, conveniently because this has a nice inverse, the inverses cancel each other. [NOISE] We get that the weights we're looking for can be derived by taking the x matrix times its own transpose, inverting that, multiplying by x transpose and then multiplying it by the y. And that gives us exactly the coefficients that we need To have done our polynomial regression. And it just, it just so happens that we have some nice properties in terms of these x transpose x. Not only is it invertible, but it does the right thing in terms of minimizing the least squares. It does it as a projection. Now, we're not going to go through the process by by which we argue that this is true. Does it have something to do with calculus? It most likely has something to do with calculus. And we'll get back to calculus later. But in this particular case we can, we're just using projections and linear algebra. And most importantly the, the whole process is just we take the, the data we arrange it into this matrix with whatever sort of powers that we care about. And then we just compute this quantity and we're good to go. Okay.

1.1.41 Errors Question

Alright, now, 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, the reason we have to do these squares is because of the presence of errors. The training data that, that we are given, has errors in it. And it's not that we're actually modelling, a function, but ,the thing that we're seeing is the function plus some, you know, some error term on each piece of data. So, I think, it's reasonable to, to think about where did these errors come from? So, I don't know, what do you think ,Charles, why, why is it we're trying to fit data, that ,has error in it, can't we just, can't we just have no errors? [LAUGH] I would like to have no errors. Certainly ,my code, has no

errors. [CROSSTALK] well, so let's see where might errors come from. So, they could come from, sensor error, right? Just ,somehow you're, you're getting inputs and you're getting outputs and that output's, being read by, some machine or by a camera or by something and you just, there's just error in the way that you read the data. Just an error in the sensors. Alright, can you think of other ways. I guess, I guess ,in this case you're imagining that the data came by actually measuring something, with the machine. So that, that makes, a lot of sense. What other ways, can we put together the data? I don't know I could think of a bunch. I mean the error, well, the errors could come, maliciously. There could be some, something out there, that is trying to give us bad data. Alright, that seems like a possibility, that, when the data set was collected, let's say that we're collecting, various, Oh, maybe if I. Oh, this happens, this happens a lot. So, so, if you're trying to collect data from other Computer Science departments and you're trying to put together, some kind of collection of, you know, how much do you spend on your. Graduate students ,say,uh, sometimes ,these departments will actually misrepresent the data and give you give you, things that are wrong. Because, they don't want to tell you the truth, because they're afraid of what you are going to do. Yeah, I've noticed that everyone does that except, for Georgia Tech and Brown University. Yeah, there are highly honest and reputable universities in my experience. Yeah, that's what I feel. well, another time that you can get data, is if somebody, is, copying stuff down. So, what about sort of the idea of a transcription error. Uh-huh. So we're just, you know, we've copied everything, but, you know, there's, there's just some of the, some of the lines that got filled in just got mistyped. Yeah, and you think ,that's different from sensor error? Well, it's, it's maybe a slightly different kind of sensor error, right? So ,sensor errors were actually saying there's something physical, that's being measured and there's just noise in that. Transcription error, is similar except it's a person. [LAUGH] Mm. Right? The, the there's a little blips in the person's head and they can do, it can be a very different kind of error. You can get, like transpositions of digits, maybe instead of ,um, just you know, noise. Okay, how bout, how bout one more? How about ,uh, there's really, just noise, in the process. So how about that, that we took in input X, but there's something else going on in the world, that we weren't measuring, and so the output ,might depend on other things besides, simply ,the input that we're looking at. So what would be an example of that? So an un-modeled influence, might be, well, if we're. [CROSSTALK] Let's look at the housing data. That's what I, that's what I was thinking exactly. So ,in the housing data ,we were just trying to relate, the size of the houses, to the price, but, there's a lot of other things like change of the houses to the price and Location, location. Location and location, right those are three really good reasons, that are not in the particular regression, that we did, that could ,actually influence the prices. So right, that and, you know, the quality of the house and who, who built it, and, you know, the colors, the colors. Even, even, even time of day, or what the interest rates were that morning, versus the, what people thought they might be the next day. Who knows? Right and so all these different things are being considered ,in that particular regression, so we're just kind of imagining ,that it's noise, that it's just having a, a ,uh, bumpy influence on the whole process. Sure. All right. So, so what I'd like you to do is select ,the ones that you think actually are important, the ones that, that, that could actually come up, when you're using machine learning and regression to solve your problems.

1.1.42 Errors Solution

All right, and if you know, if you were paying attention as we were going through this, these are all very common, and realistic things. So, you know these are all true, these are all sources of error. And this is why we really need to be careful when we fit our data. We don't want to fit the error itself, we want to just fit the underlying signal. So let's talk about how we might be able to figure that out. How can we, how can we get a handle on what the underlying function really is apart from the errors and the noise that are, that are in it.

1.1.43 Cross Validation

Alright, so let me try to get to this concept of cross validation. So, imagine that we've got our data, this is our training set. We can, again, picture geometrically in the case of regression. And, ultimately what we're trying to do is find a way of predicting values and then testing them. So, what we imagine is we do some kind of regression and we might want to fit this too a line. And, you know, 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, you know, 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. So what can we do in terms of if we, if we fit it with the line and the errors not so great. What else could we switch to Charles? We could just use the test. No, sorry. What, what I mean is if we fit, we fit this to a line and we're sort of not happy with the fact that

the line isn't fitting all of the points exactly. We might want to use ,uh, maybe a higher order polynomial. Oh, I'm sorry, totally misunderstood you. To fit this better. So if we, we can fit this with a higher order polynomial and maybe it'll hit, all these points much better. You know, so we have this kind of, kind of other shape, and now it's doing this, it's making weird predictions in certain places. So, really what we'd like to do is, and what was your suggestion? If we trained on the test set, we would do much better on the test set, wouldn't we? Yes. But that, that, that's definitely cheating. Why is cheating? Is there some, why is it cheating? Well, if we exactly fit the error, the, the test set. That's not a function at all, is it? [LAUGH] If we exactly fit the, the test set, then again that's not going to generalize to how we use it in the real world. So the goal is always to generalize. The test set is just a stand-in For ,what we don't know we're going to see in the future. Yes, very well said. Thank you. Actually that suggests something very important, right, it suggest that ,um, nothing we do, on our training set or even if we cheat and use the test set .Actually makes sense unless we believe that somehow the training set and the test set represent the future. Yes, that's a very good point, that we are assuming that this data is representative of how the system is ultimately going to be used. In fact, there's an abbreviation that statisticians like to use. That the data, we really count on the data being independent and identically distributed, Mm-hm. which is to say that all the data that we have collected, it's all really coming from the same source, so there is no, no sort of weirdness that the training set looks different from testing set looks different from the world but they are all drawn from the same distribution. So would you call that a fundamental assumption of supervised learning? I don't know that I'd call it a fundamental of supervised learning per se, but it's a fundamental assumption in a lot of the algorithms that we run, that's for sure. Fair enough. There's definitely people who have looked at, well what happens in real data if these assumptions are violated? Are there algorithms that we can apply that still do reasonable things? But the stuff that we're talking about? Yes, this is absolutely. A fundamental assumption. Alright, but here's, here's where I'm trying to get with this stuff. So what we really would like to do, is that 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 no so complex that it's, it's matching that so directly that it doesn't really work well on the test set. But unfortunately we don't really have the test set to play with because that again, is going to, it's too much teaching to the test. We need to actually learn the true structure that is going to need to be generalized. So, so how do we find out. How can we, how can we pick a model that is complex enough to model the data while making sure that it hasn't started to kind of diverege in terms of how it's going to be applied to the test set. If we don't have access to the test set, is there something that we can use in the training set that we could have it kind of act like a test set? Well, we could take some of the training data and pretend its a test set and that wouldn't be cheating because its not really the test set. Excellent. Indeed, right, so there's nothing magic about the training set all needing to be used to fit the coefficient. It could be that we hold out some of it ,as a kind of make pretend test set, a test test set, a trial test set, a what we're going to say cross validation set. And it's going to be a stand in for the actual test data. That we can actually, make use of that doesn't involve actually using the test data directly which is ultimately going to be cheating. So, this cross validation set is going to be really helpful in figuring out what to do. So. Alright, so here's how we're going to do this, this concept of cross validation. We're going to take our training data, and we're going to split it into what are called folds. I'm not actually sure why they're called folds. I don't know if that's a sheep reference. Why would it be a sheep reference? I think there's a sheep-related concept that is called a fold. Like, You know, we're going to bring you back into the fold. Oh. It's like the, it's like the group of sheep. You are just trunk full of knowledge. Alright so what we're going to do is train on the first three folds, and use the fourth one to, to see how we did. Train on the [LAUGH] second there 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, a fake test set. And then average these errors. The ,uh, the, the goodness of fit. Average them all together, to see how well we've done. And, the model class, so like 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. Alright, so if this is a little bit abstract still let me, let me ground this back out in the housing example.

1.1.44 Housing Example Revisited

Alright so here's how we're going to look at this. So as you may recall, in this housing example. If we look at different degrees of polynomials and how well they fit the data. Let's look at the training error. The per example training error. So how far off is it for each of the data points? And 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 error's always falling. As you go up, you have more ability to fit the data, closer and closer and closer, right? because, each of these models is, 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. So, that's what happens with the training

error, but now let's use this idea of cross validation to say what if we split the data up into chunks and have each chunk being predicted by the, the rest of the data? Train on the rest of the data, predict on the chunk. Repeat that for all the different chunks and average together. So, so I actually did that. And this is what I got with the cross validation error. So there's a I don't know there's a couple of interesting things to note about this plot. So that we see, 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. So, why do you think that is Charles? Well that makes sense right? because we're actually training to minimize error. We're actually trying to minimize error on the training set. So 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. Right, so, good. So, so, in the, on the, this red curve. We're actually predicting predicting all the different data points using all of those same data points. So it is using all the data to predict that data. This blue point, which is really only a little bit higher in this case, is using, in this particular case I used all but one of the examples to predict the remaining example. But it doesn't have that example when it's, when it's doing its fitting. So it's really predicting on a new example that it hasn't seen. And so of course you'd expect it to be a little bit worse. In this particular case, the averages are all pretty much the same so there's not a big difference. But now, let's, 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 better, and, in fact, down at you know say, three and four, they're actually pretty close in terms of their ability to, to, to fit these examples. And then what's great, what's really interesting is what happens is now we start to give it more, the ability to fit the data closer and closer. And by the time we get up to, to order six polynomial, even though the error on the training set is really low, the error on this, on this cross validation error, the error that you, that you're measuring by predicting the examples that you haven't seen, is really high. And this is beautiful this, this inverted u, is, 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 [LAUGH] of that power. Does that make sense to you? It does make sense, so. The, the problem is that as we give it more and more power we're able to fit the data. But as it gets more and more and more power it tends to over fit the training data at the expense of future generalization. Right. So that's exactly how we, we referred to this is this sort of idea that if you don't give yourself enough degrees of freedom, you don't give yourself a model class that's powerful enough you will 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. So what was the answer on the, housing exam? Well, so, it seems pretty clear in this, in this plot that it's somewhere, it's either three or four. It turns out, if you look at the actual numbers, three and four are really close. But three is a little bit lower. So three is actually the thing that fits it the best. And, in fact, if you look at what four does. It fits the data by more or less zeroing out the, the quartic term, right? It doesn't really use the, this power. Oh, but that's interesting. So that means it, it barely uses the, the, the extra degree of freedom you give it. But even using it a little bit, it still does worse than generalization. Just a tiny bit worse. Huh. Yup exactly so. That's actually kind of cool.

1.1.45 Other Input Spaces

Alright. Up to this point I've been talking about regression in the context of a scalar input and continuous output. Sorry. 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. So what would might, what might be an example of where we might want to use a vector input? A couple of things. One 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. Great yeah, we could include more input features and therefore combine more things to get it. But how would we do that? So let's say for example, that we have. Two input variables that we think might be relevant for figuring out housing costs. The size, which we've been looking at already, But also let's say the distance to the nearest zoo. We, we think that that's a really important thing. People like to live close to the zoo and so. But probably not too close to the zoo. [LAUGH] Possibly not too close to the zoo. But let's let's imagine that it's like size, something that actually Or actually, let, let's do it the other way, let's sort of imagine that, that, that the further away from the zoo, you are, the better it is. Just like the bigger the size is, the better it is. Mm-hm. So how do we combine these two variables into one in the context of the kinds of function classes that we've been talking about? Well, if you think about lines, we can just generalize the planes and hyper planes. Right so, in the case of, of a 1 dimensional input. That 1, 1

dimensional input gets mapped to the cost. But in the case of 2 dimensional inputs, like size and distance to the zoo. We have something that's more like a plane, combining these two things together in, in the linear fashion to actually predict what the. Cost is going to be. So right, so these, this notion of linear functions generalizes, this notion of polynomial function function generalizes too very, very nicely. All right, there is another kind of input that's important too, that, let's think about a slightly different example to help drive the idea home. So let's imagine we are trying to predict. Credit score, what are some things that we might want to use as features to do that. Do you have a job? I do, actually. [LAUGH] yes. Oh, I am sorry, I am sorry, I misunderstood. So you are asking, you are saying one [UNKNOWN] that could be important for predicting someone's credit score is just to know do they currently have a job. Right another thing might be well you, you can ask instead how much money they actually, how, how much, how many assets they have. How much money do they have? Credit cards. Great. So, so, right. So things like, what is the value of the assets that, that they own, right? So this is a continuous quantity like we've been talking about. But something like do you have a job, yes or no, is a discreet quantity. And one of the nice things about these kinds of regression approaches that we've talking about, like polynomial regression, is that we can actually feed in these discrete variables as well. Certainly if they're, if they're Boolean variables like, do you have a job or not? It, you can just think of that as being a kind of number that's just zero or one. No, I don't have a job. Yes, I have a job. What if it's something like, you know, how many houses do you own? Hmm. That's pretty easy because that's, you could just treat that as a As a quantity, a scalar type quantity. What about Are you. Type of job. Type of job, I like that. How about hair color? So, yeah, how would we do that? If we, if we're trying to feed it in to some kind of regression type algorithm, it needs to be a number or a vector of numbers, and they can be discrete. So right. So how do we encode this as some kind of a numerical value? Well, we could do something ridiculous like actually write down the RGB value which would make it kind of continuous. Interesting. That seems insane, but you could do that. Or you could just enumerate them and just assign them values one through six in this case. Right, 1, 2, 6 or they could be vectors like, is it red, yes or no? Is it beige, yes or no? Is it brown, yes or no? Have it be a vector and actually for different kinds of discreet quantities like this it can make it different, right? So in particular if we just gave the numbers. Then it's kind of signalling to the algorithm that blonde is halfway between brown and black, which doesn't really make sense. We could reorder these. Actually the RGB idea doesn't seem so bad to me. [UNKNOWN] of course, you have an interesting question of what's the real RGB value. It implies that somehow interpreting between them Make sense. That's right, that's right. It also implies an order right. It implies that the scalar order of RGB is somehow mean something that it's no different from saying red is one and beige is two. So, if we multiply it, for example, by a positive coefficient then the more RGB you have The better or the worse, right? Hmm. Interesting. Though, in fact what I had in mind here is for RGB, it's three different hair colors. I thought the g stood for green. There's, people don't have green hair, they have gray hair. But I thought the g in RGB stood for green. Yeah it does usually but I'm making a hair joke. [LAUGH] Oh oh. I am sorry. I am glad you explained that. You know Michael. No problem sir. I really, I really like the [UNKNOWN] factor idea. Yeah so I think. I think. I imaging that we are going to return to this issues when we start actually encoding problems as mission learning problems. I think your right. But I think that's I think that's said about regression for the time being. I agree.

1.1.46 Conclusion

So, Charles, what did we learn about regression? well, we learned a bunch of interesting historical facts about where the word came from, which I thought was interesting anyway. Oh good. We learned about model selection and overfitting. And underfitting. And fitting in general, cross validation. Talked about, how to do linear and polynomial regression. Yeah. That the best constant, that the best constant in terms of squared error is the mean. Mh-hm These little cocktails for that. Well we also did the same thing for well we talked about the process for how you do it in genreral for any polynomial function. I think that's everything. Well one more thing, and we talked a little bit about representation, and how to make that work in regression. Great, we talked about input representations and what some of the issues are. There. Yeah. Great, I think that's, I think that's a good amount. I think so, too.

1.1.47 Neural Networks

Hey Charles. How's it goin'? It's going pretty well Michael. How are things going with you? Good. You know I'm excited to tell you about neural networks today. You may be familiar with neural networks because you have one, in your head. I do? Well, yeah. I mean, you have a network neurons. Like, you know, you know neurons, like brain cells. Let me, let me, I'll draw you one. Okay. So this is my template

drawing, a nerve cell, a neuron. And you can, you know, you've got billions and billions of these inside your head. And they have you know, most of them have a pretty similar structure, that there's the, there's the kind of the main part of the cell called the cell body. And then there's this thing called an axon which kind of is like a wire going forward to a set of synapses which are kind of little gaps between this neuron and some other neuron. And what happens is, information spike trains Woo woo! Travel down the axon. When the cell body fires it has an electrical impulse it travels down the, the, the axon and then causes across the synapses excitation to occur on other neurons which themselves can fire. Again by sending out spike trains. And so they're very much a kind of a computational unit and they're very, very complicated. To a first approximation, as is often true with first approximations they're very simple. Sort of by definition of first approximation. So what, what, 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. And one of the nice things about the, 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. And they can be trained through a learning process. So that's what we're going to talk through if you haven't heard about this before. Okay. So we can replace this sort of detailed version of a neuron with a very abstracted way kind of notion of a neuron. And here's how it's going to work. We're going to have inputs that are kind of you know, think of them as firing rates or the strength of inputs. X1, X2, and X3 in this case. Those are multiplied by weight, w1, w2, w3 correspondingly. And so the weights kind of turn up the gain or the sensitivity of the neuron, this unit, to each of the inputs respectively. Then what we're going to do is we're going to sum them up. So 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 we're going to ask is that greater than, or equal to the firing threshold. And 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. So this is a particular kind of neural net unit called Perceptron. Which is a very sexy name because they had very sexy names in the 50s They did. When this was first developed. Alright? So this, this whole neuron concept gets boiled down to something much simpler, which is just, a linear sum followed by a threshold, thresholding operation, right? So it's worth kind of thinking, how can we, what sort of things can this, can networks of these kinds of units compute? So, let's see if we can figure some of those things out.

1.1.48 Artificial Neural Networks Question

Alright ,just to make sure that you understand. let's let's think through an example. let's imagine, that we've gotta a neuron. We got one of these perception units. And the input, to it ,is one, zero negative one point five. For the three different, inputs in this case. And the corresponding weights, are half three fifths and one... And the threshold, let's say is zero, meaning that it should fire, if the weighted sum is above zero, or equal to zero, and otherwise, it should not fire. So, what I'd like you to compute, is based, on these numbers, what the output y would be in this case

1.1.49 Artificial Neural Networks Solution

Alright Charles you want to help us kind of work through this example? Sure. So ,we multiply x1 times w1 so that gives us a half Um-huh. We multiply zero times three fifth which would get a zero. Um-huh. And we multiply minus one point five times one. Which will give us minus three halves. And so, the answers negative. Whatever it is. It is right, so it's, this was negative ahead, negative one and a half plus a half, so it should be negative one. Right. And, but that's not the output that we should actually produce, right? That's the activation. What do we do with the activation? Well we see if the activation is above our threshold fata, which in this case is zero, and it is not So the output should be zero. Good.

1.1.50 How Powerful is a Perceptron Unit

Alright. Well we'd like to try to get an understanding of how powerful one of these perceptron units are. So, what is it that they actually do? So they, they return, in this case either 0 or 1 as a function of a bunch of inputs. So let's just for simplicity of visualization, let's just imagine that we've got 2 inputs, X1 and X2. So Charles, how could we represent the region in this input space that is going to get an output of 0 versus the region that's going to get an output of 1. Order the weights. Right. So indeed, the weights matter. So let's, let's give some concrete values to these weights. And let's just say, just making these up that weight 1 is a half, weight 2 is a half, and our threshold data is three quarters. So now what we want to do is again, break up this, this space into where's it going to return 1 and where's it going to return 0. Okay, so I think

I know how to figure this out. So, there's kind of an, there's 2 sort of extreme examples, so let's take a case where X1 is 0. X1 is 0. Okay, good. So that's this Y axis, uh-huh. Alright. So if X1 is 0, what value would X2 have to be in order to break a threshold of three quarters? Well, the weight on X2 is a half. Mm-hm. So then, the value of X2 would have to be twice as much as the threshold which in this case is one and a half. Right. So we're trying to figure out where is it, if X1 is 0, where does X2 need to be so that we're exactly at the threshold. So that's going to be. Right. The X2 times the weight, which is half has to exactly equal the threshold which is three quarters. So, if we just solve that out, you get X2 equals 3 halves. So okay that's this point here. That's going to be a dividing line. So anywhere above here, what's it going to return? It will return, it will break the threshold, and so it will return a 1. These are all going to be 1s and then below this these are all going to be 0s. Right. Alright. Well now we have a very, very skinny version of the picture. [LAUGH] Well what else can we do? Well we can do the same thing that we just did except we can swap X2 and X1 because, they have the same weight. So, we could say X2 equal to 0 and figure out what the value of X1 has to be. Good, and that seems like it would be exactly the same algebra, and so we get X1 is 3 halves, gives us at the one and a half point above here are going to be 1s and below here are going to be 0s. Okay, so now we've got 2 very narrow windows, but what we notice is 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 right? So, so the half of the, the plane that's above this line, the half plane thatt's above this line is getting us the 1 answers and below that line is giving us a zero answers. So Michael can we generalize from this, so you're telling me then that because of the linear relationship drawn out by a perceptron that perceptrons are always going to compute lines. Yeah. Always going to compute, yeah these half planes right. So there's a dividing line where you're equal to the threshold and that's always going to be a linear function and then it's going to be you know, to the right of it or to the left of it, above it or below it but its always halves at that point. Okay, so perception is a linear function, and it computes hyperplanes. Yeah, which maybe in some sense it doesn't seem that interesting, but it turns out we're already in a position to compute something fascinating. So let's do a quiz.

1.1.51 How Powerful is a Perceptron Unit Quiz Question

So this example that we, you know, created just at random actually is it computes an interesting function. So let's, let's focus on just the case where our X1 is in the set zero, one and X2 is in the set zero, one. So those are the only inputs that we care about, combinations of those. What is Y computing here? What is the name of that relationship that function that's being computed? And so, just as a hint, there's a, there's a, there's a nice short one-word answer to this if you can kind of plug it through and see what it is that it's computing.

1.1.52 How Powerful is a Perceptron Unit Quiz Solution

Charles, can you figure this out? Yes, I believe I can. So, the first thing to note is that because we're sticking with just 0 and 1, and not all possible values in between, we're thinking about a binary function. And the output is also binary. Which makes me think of Boolean functions, where zero represents false and one represents true, which is a common trick in machine learning. Alright, so and let me, let me mark those on the picture here. So we're talking about the only four combinations are here. And you're saying in particular. That we're interpreting these as combinations of true and false. Right False, false true false, false true and true, true. Exactly and if you look at it the only way that you get something above the line is when both are true. And that is called and. Also take conjunction. Right, exactly so, exactly so. So this is, even though we're, you know we're setting these numerical values but it actually is, gives us a way of specifying a kind of logic key. Right. So here's a question for you Michael. Could we do or? That's a very good question. Or looks a lot like And in this space, it, it seems like it aught to be possible. So let's let's do that as a quiz. .

1.1.53 How Powerful is a Perceptron Unit OR Quiz Question

Alright, so we're going to go in the opposite direction now. And we're saying, we're going to tell you what we want y to be, we want y to be the or function. So it should be outputting a one if either x one or x two is one, and otherwise it should output a zero. And what you need to do is fill in numbers for weight one, weight two, and theta so that it has that semantics. Now, just so you know, there is no unique answer here. There's a whole bunch of answers that will work, but we're going to check to see that you've actually typed in one that, that works.

1.1.54 How Powerful is a Perceptron Unit OR Quiz Solution

Alright Charles, let's, let's figure this one out. It turns out, as I said, there's lots of different ways to make this work, but, what we're going to do is move that line that we had for conjunction. If we, what we really want to do now is figure out how to move it down ,so that now, these three points, are, in the green zone. They're going to output, one, because they're the or, and the only one in the, that's left in the zero zone in the, in the red zone is the zero, zero case. Right. So, How are we going to be able to do that? Well, since ,we want it to be case that, either, X2 or X1, being one get you above the line, then, we need a threshold and a set of weights ,that put either one of them over. You don't have to have both of them, you only need one of them. Okay. So, let's imagine a case where X1 is one and X2 is zero ,then basically, oh, there you're right, there's a whole lot of answers, so a weight of 1, for X1, would give you a one. Right? Yes, Huh And so, if we made the threshold 1, that would work. What about weight 2? Well, we do exactly the same thing. So, we set, weight 2 equal to 1. And that means, that in the case where both of them are 0, you get 0 plus 0, which gives you something less than 1. If ,one of them is 1 and the other is 0, you get 1, which gives you right at the threshold. And, if both of them, are, one then you get two, which is still greater than one. Good, alright, that seems like it worked. The other way we could do it, is we can keep the weights at in another way we can do it, is keep the weights where they were before, that just moves this line nice and smoothly down. And then, right? So before, we had a, a threshold of, one and a half. Now we need a threshold of, like, a half ,ought to do it. Yep. Or even less, as long as it's greater than zero. So, a quarter should work, as well. So, good, so, lots, lots of different ways to do that. And, cool. Can we do not? What's not of two variables? That's a good question. Let's do not of one variable, then. Okay.

1.1.55 How Powerful is a Perceptron Unit NOT Quiz Question

Maybe you should help me finish this picture here. So what we've got is X1 is our variable and so we can take on any sort of values. And I marked negative one, zero, and one here. And if we're doing not, right, then what should the output be for each of these different values of X1? So like if the, if the, if X1 is zero, then we want the output to be. one. One. And if X1 is one, we want the output to be Zero. Zero. All right, so now what we'd like you to do is say okay, what should weight one be and what should theta be so that this, you get, we get this kind of knot behavior.

1.1.56 How Powerful is a Perceptron Unit NOT Quiz Solution

Alright Charles, you were about to say, how we could do this. I think the answer is, simply, that we basically need to flip the, here's my thinking. We need to flip zero and one, which suggests that either our weight or our threshold needs to be negative. And since we, we The threshold is in above, it's going to end up being our weight being negative. So, let's say, if we have a zero, we want to turn that into something above the threshold and if it's a one, we want it to be below the threshold. So, why don't we make the weight negative one. Okay. And that, that turn a zero into a zero and it will turn a one into a minus one. Alright. And so, then the threshold just has to be zero. So that would mean that anything, I see, so anything that's negative will be greater than, zero or negative would be greater than or equal to the threshold. And anything on the other side of that. would be under the threshold. So we get this kind of dividing line at one, so were taking advantage of the fact the equation had a greater than or equal to in it. So, yeah, right, that ought to be a Not. So ,we've got And, Or, and Not are all expressible, as perceptron units. So and, or, and not are all expressible as perceptron units. Hey that's great because if we have AND, OR, and NOT, then we can represent any Boolean function. Well, do we know that? We know that if we combine them together, we combine these perceptron units together Can we, can we express any perceptron, oh sorry, any boolean function that we want using a single perception? So, what do we normally do in this case? So ,what's the most evil function we can think of? Yes indeed, we'll when we're woking on, on decision trees The thing that was so evil was the XOR, the called parity more generally. Right. So, alright. I mean, may, maybe if we can do that, we can do anything. So, let's, let's give it a shot.

1.1.57 XOR as Perceptron Network Question

Alright so here's what we're going to do. We're going to try to figure out how to draw sorry, compute XOR as instead of a single perceptron, which we know is impossible, we can do it as a network of perceptron. Just to, to make it easier for you, here's how we're going to set it up. That we're, we've x1 and x2 as our inputs We've got two units. This first unit is just going to compute and add and we already know how to

do that. We've already figured out what weights need to be, here and here. And what the threshold needs to be, so that the output will be the and of those two inputs. So, that's all good. But ,what we don't know ,eh, what, what, it turns out to be the case, that the second unit, with now three inputs, X1, X2, and the and of X1 and X2, can also be made to, or can be, can be, now, we can set the weights on that, so that the output is going to be X or. So, what we'd like you to do is, figure out how to do that. How do you set this weight - Is the input of X1, this way which is the and input, and this way which is the X2 input, and the threshold. So that ,it's going to actually compute an X or. And, and just so you know, this is not a trick question. You really can do it this time.

1.1.58 XOR as Perceptron Network Solution

So, okay, so, how we, how we going to solve this? Okay, so, I guess the first thing to do is if you look at the table you have at the bottom, it tells us what the truth tables are for AND and XOR, alright? So, we know that Boolean functions, can all be represented as combinations of and or N not. So, I'm going to recommend you feel out that empty column with OR. So, OR is like that. Right. And you'll notice, if you look at AND OR and XOR that, OR looks just like XOR except ,at the very last row. In the second, okay good, uh-huh, and in that row. Right, and, AND on the other hand, tells us a one only on the last row. So what, I'm going to suggest that we really want that last node to do in your drawing, is to compute the or of X1 or X2. And produce the right answer, except in the case of the last row, which we only want to turn off when and happens to be true. So ,really what that node is, is computing or minus and. Alright, so how do we make this or minus and? So the way we did or before Well we did it a couple of different ways. But one is we gave weights of one on the two inputs. And then a threshold of one. And that made, ignoring everything else at the moment, this unit will now turn on if either x1 or x2 are on. And otherwise it will stay off. Right. So what's the worst case? The lowest value that you can get. Is when one of those is one and one of those is zero, which means that the, sum into those will be, in fact, one. Yeah. Right? So, if the AND comes out as being true, it's going to give us some positive value. So, if we just simply have a negative wait there, that will subtract out. Exactly in the case ,when AND is on. It's not going to quite give us the answer we want, but it's a good place to start to think about it. Alright, so like just a negative weight, like negative one. Mm-hmm. Alright. So does that work? Not quite. Alright, and why doesn't it work? Because if, well certainly when and is off then we really are just getting the or, that's all good. Yeah. But if both x1 and x2 are both on, then the sum here is going to be two minus the one that we get from the AND which is still one. So, minus one isn't enough? Minus with both, maybe we can do more than that. Maybe we can do minus two What happens if we do minus two? Then we've got ,X1 and X2 if they're both on, then we get a sum of one minus two plus one or zero. Which is less than our threshold so it will output zero. And in the other two cases, right, when and is off than it just acts like or. So this actually kind of does the right thing. Its actually OR minus kind of and times two. [LAUGH] Right. And there you go. And of course there's an infinite number of solutions to this.

1.1.59 Perceptron Training

Alright. So in the examples up to this point, we've be setting the weights by hand to make various functions happen. And that's not really that useful in the context of machine learning. We'd really like a system, that given examples, finds weights that map the inputs to the outputs. And 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 the Perceptron Rule, and the other is called gradient descent or the Delta Rule. And the difference between them is the perception rule is going to make use of the threshold outputs, and the, the other mechanism is going to use unthreshold values. Alright so what we need to talk about now is the perception rule for, which is, how to set the weights of a single unit. So that it matches some training set. So we've got a training set, which is a bunch of examples of x, these are vectors, and we have y's which are zeros and ones which are the, the output that we want to hit. And what we want to do is set the, set the weights so that we capture this, this same data set. And we're going to do that by, modifying the weights over time. Oh, Michiel, what's the series of dashes over on the left. Oh, sorry, right. I should mention that, so one of the things that we're going to do here is were 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. So there's a, there's a very convenient trick for actually learning them by just treating it as a, as another kind of weight. So if you think about the way that the, 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, what I did here was took the actual data, the x's, and I added what is sometimes called a, a bias unit to it. So basically, the input is one always to that. And the weight corresponding to it is going to correspond to negative theta ultimately. So, just, just again, 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 some, instead of theta. Centric, yeah. It certainly makes the math shorter. So okay, so this is what we're going to do. 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 X is our input. And what we're going to do is we're going to change weight i, the, the, the weight corresponding to the ith unit, by the amount that we're changing the weight by. So this is sort of a tautology, right. This is truly just saying the amount we've changed the weight by is exactly delta W - in other words the amount we've changed the weight by. So we need to define that what that weight change is. The weight change is going to be find as falls. 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. So what's going to happen here, if they are both zero so let's, let's look at this. Each of y and y hat can only be zero and one. If they are both zeros then this y minus y hat is zero. If they're both ones and what does that mean? It means the output should have been zero and the output of our current. Network really was zero, so that's, that's kind of 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. Okay, so 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 cut the learning rate and use that as the the weight update change. So essentially what we are saying is if the output is already correct either both on or both off. Then there's gong to be no change to the weights. But, if our output is wrong. Let's say that we are giving a one when we should have been giving a zero. That means our, the total here is too large. And so we need to make it smaller. How are we going to make it smaller? Which ever input XI's correspond too, 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 this, this little learning rate. Alright, 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 kind of 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, the input output pairs. So, we'll have a chance to get in to really build things up, but we're going to do it a little bit at a time so we don't overshoot. And that's the rule. It's actually extremely simple. Like, you, actually writing this in code is, is quite trivial. And and yet, it does some remarkable things. So let's imagine for a second that we have a training set that looks like this. It's in two dimensions, again, so that it's easy to visualize. That we've got. A bunch of positive examples, these green x's and we've got a bunch of negative examples these red x's, and were trying to learn basically a half plane right? Were trying to learn a half plane that separates the positive from the negative examples. So Charles do you see a, a, half plane that we could put in here that would do the trick? I do. What would it look like? It's that one. By that one do you mean, this one? Yeah. That's exactly what I was thinking, Michael. That's awesome! Yeah, there are isn't, isn't a whole lot of flexibility in what the answer is in this case, if we really want to get all greens on one side and all the reds on the other. 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, right? That there is a way of separating the positives and negatives with a line. And 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. In fact, which I guess is really the same as saying that it will actually find it. It won't eventually get around to getting to something close to it. It will actually find a line, and it will stop saying okay I now have a set of weights that, that do the trick. So that's happens if the data set is in fact linearly separable and that's pretty cool. It's pretty amazing that it can do that, it's a very simple rule and it just goes through and iterates and, and solves the problem. So. Charles Sened solves the problem. So. I can think of one. What if it is not linearly separable? Hmm, I see. So, if the data is linearlly separable, then the algorithm works, so the algorithm simply needs to only be run when the data is linearlly 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. Well there is one way you could do it. Whats that? You could run this algorithm, and see if it ever

stops. I see, yes of course, there's a problem with that particular scheme, right, which says, well for one thing this algorithm never stops, so wait, we need to, we need to address that. But, but really we should be running this loop here, while, there's some error so I neglected to say that before. But what you'll notice is if you continue to run this after the point where it's getting all the answers right. It found a set of weights that lineally separate the positive and negative instances what will happen is when it gets to this delta w line that y minus y hat will always be zero the weights will never change we'll go back and update them by adding zero to them repeatedly over and over again. So. If it ever does reach zero error, if it ever does separate the data set then we can just put a little condition in there and tell it to stop filtering So what you are suggesting is that we could run this algorithm and if it stops then we know that it is linearly separable and if it doesn't stop Then we know that it's not linearly separable, right? By this guarantee. Sure. The problem is we, we don't know when finite is done, right? If, if this were like 1,000 iterations, we could run it for 1,000 if it wasn't done. It's not done, but all we know at this point is that it's a finite number of iterations, and so that could be a thousand, 10 thousand, a million, ten million, we don't know, so we never know when to stop and declare the data set not linearly separable. Hmm, so if we could do that, then we would have solved the halting problem, and we would all have nobel prizes Well, that's not necessarily the case. But it's certainly the other direction is true. That if we could solve the halting problem, then we could solve this. Hm. But it could be that this problem might be solvable even without solving the halting problem. Fair enough. Okay.

1.1.60 Gradient Descent

So we are going to need a learning algorithm that is more robust to non-linear separability or linear non-separability. Does that sound right? Non-linear separability. Non linear separability. Non? Yeah think of it. Left parenthesis, linear sep, spreadability left parenthesis. There we go, that's right, negating the whole phrase, very good. So and, Gradient descent is going to give us an algorithm for doing exactly that. So, what we're going to do now is think of things this way. So what we did before was we had a summation over all the different input features of the activation on that input feature times the weight, w, for that input feature. And we sum all those up and we get an activation. And then we have our estimated output as whether or not that activation is greater than or equal to zero. So 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 Not thresholded value is, as close to the target as we can. So this actually kind of brings us back to the regression story. We can define an error metric on the weight vector 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. Right? 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. ¿ Hey Michael, can I ask you a question? Sure. Why one half of that? Mm. Yes. It turns out that it turn, 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. Okay. Just like in the regression case we're going to fall back to calculus. Right, calculus is going to tell us how we can push around these weights, to try to push this error down. Right, so we would like to know. How does changing the weight change the error, and let's push the weight in the direction that causes the error to go down. So we're going to take the partial derivative of the, 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. So that's the partial dif, dif, [INAUDIBLE] So that's the partial derivitive with respect to weight wi, of exactly this error measure. So to take this partial derivitive we just use the chain rule as we always do. And what is it to take the derivitive of something like this, if you have this quantity here. We take the power, move it to the front, keep this thing, and then take the derivitive of this thing. But that, so this now answers your question, Charles. Why do we put a half in there? Because down the line, it's going to be really convenient that two and the half canceled out. So, 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. So so what we're left with then, is exactly what I said, 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. Now, we need to take the partial derivative with respect to weight w i of this sum that involves a bunch of the ws in it. So, when don't match the w i, that derivative is going to be zero because the, you know, changing the weight won't have any impact on it. The only place where this, changing this weight has any impact is at x of 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. So this is what we're left with. Now the derivative of the error with respect to any weight w sub i. Is exactly this, this sum. The sum of the difference between the activation and the

target output times the activation on that input unit You know? That looks exactly like, almost exactly like the rule that we use with the rule that we used perceptron before. It does indeed! What's the difference? Well, actually let's Let's write this down. 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.

1.1.61 Comparison of Learning Rules

So here's our update rules what they end up being. The gradient descent rule we just derived says what we want to do is more the weights in the negative direction of the gradient. So if we negate that expression that we had before and take a little step in that direction we get exactly this expression. Multiply the. The input on that weight times the target minus the activation. Whereas in the perceptron case what we were doing is taking that same activation, thresholding it. Like, determining whether it's positive or negative. Putting in a zero or a one. And putting that in here, that's what y hat is. 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, calculus. [LAUGH]. I guess that's not really an answer is it. It's, the gradient descent rule is good because it's more. Robust. To to data sets that are not linearly separable, but it's only going to converge in the limit. To a local optimum. Alright is that, is that the story there Charles? As far as I'm concerned.

1.1.62 Comparison of Learning Rules Quiz Question

So once we see these two things next to each other, it kind of raises the question, why, don't we just use a gradient decent type ,on an error metric that's defined in terms of y, hat instead of this, the activation a? because y hat ,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, there's no need for that. So, it seemed there's a, bunch of different possible reasons for that. It could be, well we don't do that, because, it would just be computationally compactable. It's too, it's too much work. Another possibility ,would be, well to do the gradient descent, you'd have to be able to take the derivitive and if we use it in this form, it's not differentiable. So, we can't take the derivative. Another one is, well sure we can do all that, it's not intractable and its not, not differentiable. But, if we do that then the weights tend to grow too fast, until you end up getting unstable answers, and then, the last possible choice that we will give you is. You can do that but you can get multiple different answers and the different answers, behave differently and so this is really just to keep it from being in illdefined.

1.1.63 Comparison of Learning Rules Quiz Solution

So why don't we do gradient descent on y hat? Well there could be many reasons but the main reason is it's not differentiable. It's a just discontinuous function. There's no way to take the derivative at the point where it's discontinuous. So this this activation thing. The, the change from. Activation to y hat 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, it's not. So the derivative is basically zero, and then that. 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, of course, doesn't really give us any information either. So this, this algorithm doesn't really work, if you. Try to take the derivative through this discontinuous function. But it does kind of, you know. What if we made this, more differentiable? Like, what is it that makes this so undifferentiable? It's this, it's this really pointy spot, right. So you could imagine a function that was kind of like this, but then instead of the point spot, it kind of smoothed out a bit. Mm, like that. So kind of a softer version of a threshold, which isn't exactly a threshold. But it leaks this differentiable. Hm. So that would kind of force the algorithm to put its money where its mouth is. Like if that really is the reason, that the problem is non differentiable, fine. We'll make it differentiable. Now, how do you like it? I don't know, how do we like it now [LAUGH]? Well, I'll tell you how much I like it when you show me a function that acts like that.

1.1.64 Sigmoid

Challenge accepted. We're going to look at a a function called the sigmoid. Sigmoid meaning s-like, right, sig, sigma-ish, sigmoid. So we're going to define this, this, the sigmoid using the letter. Sigma 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, one over one plus e to the, e to the minus a, and what do we know about this function? Well, it is. Ought to be clear that as the activation gets less and less and less, we'd want it to go to zero, and in fact it does, right. So, as a goes to negative infinity, the negative a goes to infinity. E to the infinity is something really, really big. So it's one over 1 plus something really big, which is like 1 over something huge, which is almost zero. So, the sigmoid function goes toward, this function that we defined here, goes to zero as the activation goes. To negative infinity, that's great, that's just like threshold, and as the activation gets really really large, we're talking about e to the minus something really large, which is like e to the almost, or like e to the negative infinity which is like almost zero, so one over one plus zero is essentially one. So on the one limit, it go towards zero, and the other limit it goes towards one, and in fact we can just Draw this so you can see what it really looks like you know, minus five and below it's essentially at zero, and then it makes this kind of gradual, you can see why it's 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. So instead of just an abrupt of transition to zero, we had this gradual transition between negative five and five. And this is great because it's differentiable, so. What do you think Charles, does this answer your question? It does, I buy that. Alright good so if we have units like this now we can take derivatives which means we can use this gradient decent idea all over the place. So not only is this function differentiable but the derivative itself has a very beautiful form. In particular it turns out... That if you take the derivative of this sigma function, it can be written as the function itself times one minus the function itself. So this is just, this is just really elegant and simple. So, if you have, you know, the sigma function in your code, there's nothing special that you need for the derivative. You could just compute it this way. So we would, it's not a bad exercise to go through and do this. Practice your calculus, we just did this together but it's not that fun to watch. So I would suggest doing it on your own, and if you have any trouble we'll, we'll provide additional information for you to, to help you work that out. But when you do it on your own make sure that no one is watching. [LAUGH] Well they can watch, they just probably won't enjoy it very much. So, so can we say anything about why this form kind of makes sense? So, so what's neat about this is. As we, as our activation gets very, very negative, then our sigma value gets closer and closer to zero. And if you look at what our derivative is there, it's something like zero times something like one minus zero, whereas the derivative as you get to very, very large as, that's like sigma's going to one. And you get 1 times 1 minus 1 minus 1, so essentially 1 times 0. So you can see the derivatives flatten out for very large and very negative a's. And when a is like, zero, so what happens when a is like zero? Boy, what does happen when a is like zero? Charles, what happens if we plug zero into this sigma function? You get one half. Is that obvious? Oh, I see, because e to the minus a, that's zero, so e to the zero is one, one over one plus one, so a half. And then our derivative at that point is a half times a half, or a quarter, so that's kind of neat. Mm-hm. So so this is really, this, it's, it's in a very nice form for being able to work with it. But it's probably worth saying that. Surely you could use other functions that are different, and there might be good reasons to do that. This one just happens to be a very nice way of dealing with the threshold in question. Yeah and there's other ways that are also nice. So again, the main properties here are that as activation gets very negative it goes to zero, as activation gets very positive it goes to one, and there's this smooth transition in between, there's other ways of making that shape.

1.1.65 Neural Network Sketch

Alright so we're now in a great position to talk about what the network part of the neural network is about. So now the idea is that we can construct using exactly these kind of sigmoid units, a chain of relationships between the input layer, which are the different components of x, with the output. Y, and the way this is going to happen is, there's u, other layers of, of units in between. That each one is computing the weighted sum, signoided, of the layer before it. These other layers of units are often referred to as hidden layers, because you can kind of see the inputs, you can see the outputs. This, this other stuff is, is less constrained. Or indirectly constrained. And what's happening is that each of these units, it's, it's running exactly that kind of, you know, take the weights, multiply by the things coming into it, put it through the sigmoid and that's your activation, that's your output. So, so 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 that there's all these sort of crazy non linearities in between. And so, this leads to an idea called back propagation, which is really just at its heart, a computationally beneficial organization of the chain rule. We're just computing the derivatives with respect to all the different weights in the network, all in one convenient way, that has, this, this lovely interpretation of having information flowing from the inputs to the outputs. And then error information flowing back from the outputs towards the inputs, and that tells you how to compute all the derivatives. And then, therefore how to make all the weight updates to make, the network produce something more like what you wanted it to produce. So this is where learning is actually taking place, and it's really neat! You know, this back propagation is referring to the fact that the errors are flowing backwards. Sometimes it is even called error back propagation. Nice, so here's a question for you Michael. What happens if I replace the sigmoid units with some other function and, and let's say that function is also different Well, if it's differentiable, then we can still do this, 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. Hmm. That's a big win. Does it still act like a preceptron? Well, even this doesn't act exactly like a preceptron, right? So it's really just analogous to a preceptron, because we're not really doing the hard thresholding, we don't have guarantees of, of convergence in finite time. In fact, the error function can have many local optima, and what, what we mean by that is this idea that we're trying to get the, 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, well good, then 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. Oh so that makes sense, so if we think about sigmoid the sigmoid and the error function that we picked right. The error function was sum of squared airs, so that looks like a porabola in some high dimensional space, but once we start combining them with others like this over, 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. Right, exactly right and 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 now 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 this, that there's, it's smooth but there's these Place where it goes down, comes up again and goes down maybe further, comes up again and doesn't come down as far and you could easily get yourself stuck at a point like this where you're not at the global minimum. Your at some local optimum.

1.1.66 Optimizing Weights

one of 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 and then you start to wonder boy is there some other way that I can optimize these weights i'm trying to find a set of weights for the neural network that well that what that that tries to minimize error on the training set and so gradient descent is one way to do it and it can get stuck but there's other kinds of advanced optimization methods have become very appropriate here in fact there's a lot of people in machine learning who think of optimization and learning is kind of being the same thing that what you're really trying to do in any kind of learning problem is solved this this high-order very difficult optimization problem to figure out what the learned representation needs to be so i just want to mention in passing so various kinds of advanced methods that that people brought to bear there's things like mom using momentum terms in the gradient which basically where the idea in the momentum is as we're doing gradient descents imagine this is our error surface we don't want to get stuck in this little Bowl here we want to kind of pass all the way through to get to this bowl so maybe we need to just you know continue in the direction we've been going so instead of you know think of it as a kind of physical analogy instead of just just going to the bottom of this hill and getting stuck it can kind of bounced 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 to better optimize things instead of just thinking about the way that individual weights change the error function to look at combinations of weights Hamiltonians and whatnot there's various ideas from randomized optimization which were going to get to in a sister course that can be applied to two to make things more robust and sometimes it's worth thinking you know what 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 using a structure that's too complex missed when do we when do we see something like this before Charles when we were doing regression and we were talking about overfitting what's a more or less complex network

well there's two things you can do with network you can add more and more nodes and you can add more and more layers good so right so if we do more nodes that we put into network the more complicated the mapping becomes from input to output the more local minimum we get the more we get they have the ability to actually model the noise which brings up exactly the same overfitting issues it turns out there's another one that's actually really interesting in the neural net setting which I think didn't occur to people in the early days but it became clear and clear over time which is that you can also have a complex network just because the numbers the weights are very large so same number of weight same number of nodes same number of layers but larger numbers often leads to more complex network and the possibility of overfitting and so sometimes we want to penalize the network not just by giving it $PERCENT fewer nodes or layers but also by keeping the numbers in a reasonable range set that makes sense that makes perfect sense

1.1.67 Restriction Bias

So this brings up the issue of what neural nets are more or less appropriate for. What is the restriction bias, and the inductive bias of this class of classifiers, and regression algorithms? So Charles, can you remind us what restriction bias is? Well, restriction bias Tells you something about the representational power of whatever data structure it is that you're using. So in this case the network of neurons. And it tells you the set of hypotheses that you're willing to consider. Right, so if, if the, if there's a great deal of restriction, then there's lots and lots of different kinds of models that we're just not even considering. We're, we're restricting our view to just a subset of those. So In the case of neural nets, what restrictions are we putting? Well, 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 exor, and that allowed us to do more. Then we started sticking Sigmoids and other arbitrary functions and to nodes so that we could represent more and more, and you mention that if you let weights get big and we have lots of layers and lots of nodes, we can be really, really complex. So, it seems to me that we are actually not doing much of a restriction at all. So let me ask you this then Michael. What kind of functions can we represent, clearly we can represent boolean functions, cause we did that. Can we represent continuous functions? That's, that's a great question to ask, that's what we should try to figure that out. So, in the case, as you said, Boolean functions, we can. 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. So that one definitely can happen. So what about continuous functions? So what is it? What is a continuous function? A continuous function is one where, as the input changes the output changes somewhat smoothly, right? There's no jumps in the function like that. Well, there's no discon, there's no discontinuities, that's for sure. Alright, now 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, 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. And, essentially 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, that it needs to model. And they, the patches get set at the hidden. 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 Arbitrary. 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, right, so maybe multiple hidden layers and multiple units. So that worries me a little bit Michael, because it means that we're almost certainly going to overfit, right? 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. So, how are we going to avoid doing that? Excellent question. So, it seems like there's, there is exactly that worry. But, 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. And so, it's not like any fixed network can actually capture any arbitrary function. So 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. So that's one thing, the other is hey, well we can do with overfitting what we've done the other times we've had to deal with overfitting. And that's to use ideas like, cross validation. And we used cross validation to decide. How many hidden layers to use. We can use it to decide how many nodes to put in each layer. And we can also use it to decide when to stop training because the weights have gotten too large. So, and this is, it's probably worth pointing this out

that this is kind of a different, different property from the. Other classes of supervised learning algorithms we've looked at so far. So in a decision tree, you build up the decision tree, an you may have over fit, but it is what it is. In regression, you know, you solve the regression problem, and again that may have over fit. 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 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 the, 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, this pattern that we've seen before, that the cross validation error can turn around and, and at this, you know, at this low point, you might have, you might want to just stop training your network there. The more you train it, possibly the worse you'll do. And again, that, it's reflecting this idea that the complexity of the network is not just in the nodes and the layers, but also in the magnetude of the weights. Typically what happens in this turnaround point is that some weights are actually getting larger and larger and larger. So, just wanted to highlight that difference between neural net function approximation of what we see in some of the other algorithms

1.1.68 Preference Bias

Alright, you know the issue that we want to make sure that we think about each time we introduce a new kind of supervised learning representation is to ask what its preference bias is. So Charles, can you remind us what preference bias is? Mike researcher bias tells you what it is you are able to represent. 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. So, perhaps you 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. So that actually brings up a point here which is, we haven't actually chosen an algorithm. We talked about how derivatives work, how back propagation works, but you missed telling me one very important thing, which is how do we start? You tell me how to update the weights but, how do I start out with the weights? Do they all start at zero? Do they all start out at one? How do you usually set the weights in the beginning? Yes indeed. We did not talk about that, that's, it's really important. You can't run this algorithm without initializing the weights to something. Right? We did talk about how you update the weights but they don't just you know, just start undefined and you, you can't just update something that's undefined. So we have to set the initial weights to something. So pretty typical thing for people to do, is small, random, values. So why do you suppose we want 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. Yea kind of. I mean there's also the issue of Well if we run the algorithm multiple times if we get stuck, we like it not to get stuck exactly there again, if do, if you run it again. So it gives some variability, which is a helpful thing in avoiding local minimal. And what do you suppose, it's important to start with small values. Well you just said. In our discussion before that if the weights get really big that can sometimes lead to over fitting, because it let's you represent arbitrarily complex functions. Good. And so, and what is that tell us about what the preference bias is then? Well 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. And of course the usual stuff like we prefer, correct answers to incorrect answers, and so on and so forth. ¿ So, you'd say that neural-nets implement, or maybe we should say, that neural networks implement a kind of bias that says Prefer correct over incorrect but all things being equal, the simpler explanation, is preferred. Well, if you have the right algorithm. If the algorithm starts with small, random values and tries to stop, you know, when you start over-fitting Then you, cause you're going to start out with the simpler explanations first before you allow your weights to grow. so you, about that. So this reminiscent of the principal that is known as Occan's razor which is often stated as entities should not be multiplied unnecessarily. And given that we're working with neural networks, there's a lot of unnecessary multiplication that happens. [LAUGH] But, in fact, this actually is referring to exactly what we've been talking about. So this unnecessarily is, one interpretation of this is that, "Well, when is it necessary?" It's necessary if you're getting better explanatory power, you're fitting your data better. So Unnecessarily would mean, well 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. And 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.

1.1.69 Summary

So that brings us to the end of the topics we are going to talk about in terms of neural nets. There's going to be some interesting stuff for you to do in terms of the homework where you'll be exposed to some other important concepts. But that's, that's all we're going to lecture about for now. So let's just remind ourselves what exactly we covered in the neural net section. So Charles what do you remember? I remember perceptrons. I remember. And perceptron was a threshold unit, a linear threshold unit, and we could put networks of them together. Yes. To produce any Boolean function. What else? Oh, we had a learning rule for perceptrons. Mm-hm. Which runs in finite time for linearly separable data sets. And we learned a general differentiable rule. Adding general we learned about propagation using a gradient set. And we talked a little bit about the, about the preference and restriction by c's of neural networks. Alright, til next time. See you Michael.

1.2 Instance Based Learning

1.2.1 Instance Based Learning Before

Hi Michael! Hey Charles! How's it going? It's going pretty well. How's it going with you? It is a beautiful fall day here in Providence Rhode Island. Oh that's right it's fall, when you are. [LAUGH] Yeah, I think, that's right. So, what we're going to do today, Michael. If you will indulge me. Is ,we're going to talk about a different class of ,uh, learning algorithms and approaches than we've been talking about before. So now the other ones were low class, this is going to be high class? Exactly. And we call them instance based learning. Which sounds very hoity toity and high voluting. Don't you agree? Yeah, sure, why not? [LAUGH] It sounds like it maybe has good posture. It does, in fact, have good posture. Well let's, let's learn about it. I'm, I'm, I'm intrigued. Yeah, so I think that ,uh, what we're going to end up talking about to day is kind of interesting, I hope. But it's sort of different, and what I'm hoping is through this discussion Is that, we will be able to reveal some of the unspoken assumptions that we've been making so far, okay? Unspoken assumptions, it sounds, yeah, okay, that sounds like we should get to the bottom of that. Yes, so let's do that. So, just to remind you of what we have been doing in the ,um, past, this is what was going on with all of our little supervised learning tasks, right. We were given a bunch of training data ,labeled here as you know, x, y One, xy two, xy three, dot, dot, dot, xy zen. And ,uh, we would then learn some function. So, for example, if we have a bunch of points in a plane, we might learn a line to represent them, which is what this little blue line is. 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, right? Okay. Yeah, so like, black is the input here and then the two blue things are what get derived by the learning algorithm. Right. 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. In this case, probably line. And that would, be how we would determine answers going forward. Yeah. That's, that's what we've been talking about. Right. And in particular without reference to the original data. So. I want to propose an alternative and the alternative is basically going to not do this. [LAUGH] So let me give you a, let me, let me tell u exactly what I mean by that.

1.2.2 Instance Based Learning Now

Okay, so here what I mean concretely by not doing this thing over here any more. So here what I'm proposing we do now. We take all the data, all of the training data we had, the xy1, xy2, dot, dot, dot, xyn and we put it in a database Ah-ha. And then next time we get some new x to look at, we just look it up in the database. And we're done. None of this fancy shmancy learning, none of this producing an appropriate function like you know, wx+b, or what ever the equation of a line is. None of that fancy stuff anymore. We just stick in to the data base. People written data base programs before. We'll look it up when we're done. We're done.Period I feel like you've changed the paradigm. Yes, I am a paradigm changer. So what do you think? It's like, it's like disruptive. It's going to throw off the markets. Yeah. It's going to change everything. So, what do you think? well, I mean, so there's a bunch of really cool things about this idea. Which is why I'm excited. So one is it, you know, it doesn't forget, right? So it actually is very reliable. It's very ,um ,dependable, right, so 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, you know, crack potty, smooth version of it. Right, so we don't, yeah that's a good point. So we look at this little blue line over here, you'll notice that say, for this little x over here, we're not going to get back what we put in, so, it remembers, it's like an elephant. Good. So that's kind of cool. Another thing is that there's none of this wasted time, you know, doing learning [LAUGH]. It just takes the data and it, and it's, very rapidly just puts it in the database. So [CROSSTALK] it's fast. It's fast. It's like. I like it when you say nice things about my algorithms. Okay. So it's fast. Anything else, you can think of? Sh, yeah I cant think of one more thing at least. Mm-hm. Which is ,that like you, it's simple. [LAUGH] That's true. I am simple. I am simple and straightforward. I just need a few things to make me happy. Bacon. Bacon, and And Chocolate. Oh nice. Have you ever had chocolate covered bacon? That seems wrong. You know, you would think so, but it turns out it's delicious. It's like if you take fat, and sugar, and you put it together somehow you like it [LAUGH] I'm not making this up you can buy chocolate covered bacon, you're unsurprised to here in America. Okay, so we've got three good things in remember stuff. So, you know, none of this little noisy throwing away information. It's very fast, you just stick it in a database. Using your favorite data base. And looking up is going to be equally as fast. And it's very simple. There's really no interesting learning going on here. So it's the perfect algorithm when we're done. Okay. I mean, I, it feels like there's more that we need to say though. Like what? In particular the way that you wrote this, F of X equals look up of X. If I give you one of the other points in between ,then 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. Mm. So the down side of remembering is, no generalization. [LAUGH] And I guess a similar sort of issue is that when you, when you call it memorization, it makes, it reminds me of the issues that we saw with regard to overfitting So, it bottles the noise exactly, it bottles exactly what it was given. So it's going to be very sensitive to noise. So it's kind of a yes and no. So that's a little scary and, and it can over fit in a couple of ways, I think it can over fit ,um, by believing the data too much that is literally believing all of it and what do you do if you have couple ,uh, speaking in noise, what if you have you know a couple of examples that are all the same. I have got an x, shows up multiple times but each with a different y. Oh,the same x,ah, yeah, and so the look up would return two different things and this algorithm or whatever that you have described so far, wouldn't commit to either of them and it would just say, hey, here is both. Yeah, that seems problematic. Okay, alright. But I feel like, you know, you are going to to tell me, how to fix those things. So I wasn't too worried. Yeah, well, there is gotta be a nice way of fixing it. I think There's sort of a basic problem here, which is that we're taking this remembering and then looking up a little too literally, right? So I stick in the data, and I can get back exactly the data that I got, but I can't get back anything that I don't have, and that seems like something that we might be able to overcome if we're just a little bit clever. Mm. So let's see if we can be a little bit clever.

1.2.3 Cost of the House

Okay Michael, so let's see if we can, work together to deal with this minor triffle of a problem, that ,ah, you've observed with my cooling algorithm, okay. So, 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. [CROSSTALK] And, 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. Okay? Okay, where is this? Where is this? Oh, this is Georgia Tech, as you can tell because, it says Georgia Tech. Oh, I see it now. Okay. So, here's what I want you to do. using machine learning. I want you to look at all of this data, and then I want you to tell me, for these little black dots, whether they are really expensive, moderately expensive or, or inexpensive. But ,I want you to do it, using something like the technique that we talked about before. Okay? So, let's look at this little dot, over here. Which, by the way, I want to point out. this little black dot here by the US Post Office, underneath the rightmost ,e ,over here, is not a point in our data base. But I think by staring at this, you might be able to come up with a reasonable guess, about whether it is moderately expensive, expensive, or inexpensive. Okay, yeah. I think, this is a helpful, example, because, now I see that it does kind of make sense, especially, in this context, to think of the geometric location, as actually being a very useful attribute for deciding how to label the new points. So, that black point that you've pointed out, is in the part of the neighborhood, that has a green dot in it. Like, the nearest dot to it, seems like a pretty good guess as to what, what the value of that house might be, so I'm going to guess green. Yes, and I think ,you would be right. And I like the word that you used there. You talked about, its nearest neighbor, so I like that. I'm going to write that down. Neighbor, okay. So, I'm going to look at my nearest neighbor. Well let's see if this works, for another point. Let's look at another point, that's near an, e, let's see, the first e over here.

This little black point, over here. What do you think? If I, if I looked at my nearest neighbor, what would I, what would I guess? Yeah, this one seems really clear. It's, it's surrounded by red. It's in the red part of town. So, you're guessing, the output is then, purple? [LAUGH] No, I'm going with red. Yes, and I think that that makes perfect sense. So, this is pretty cool. If I have a point that's not in my database ,but, I still, by looking at my nearest neighbour, can, sort of figure out ,ah, what the actual value should be. So, there we have solved, the problem. Yes, seems like a pretty good role. Yeah, just look at your nearest neighbour and you are done. There, so, boom. There is nothing else for you to do. Yeah, except that you didn't do all of the houses yet. Okay, well, what did I miss? The one in the middle and [CROSSTALK] I'm wondering, if maybe you did that on purpose, because, this one has some issues. What are its issues? Besides being too, near 10th Street? well, yeah, apart from that it doesn't really have any very close neighbors ,on the, on the map. So the closest that you get, is, I don't know, maybe that red one? Maybe. But I would be really, I'd be very wary of just using that as my guess, because, it's also pretty darn close to a bluepoint. Yeah. And also not so, far from the green point. That's a good point. 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. And they are kind of close to you. So, any clever way we might around this? I would say, move the black dot, to, No, no, no, no, we are not allowed that before. No? Okay, right it seems, it seems, like it would be helpful. No, no, they are federal laws ,against interesting. I was going to say, yeah, so, alright So, short of that, maybe ,we just need to look at a bigger context? Ahh, that makes sense. So, you're saying my little nearest neighbor thing, sort of worked ,but the problem was I started out with examples ,that were, you know, very clearly in a neighborhood, and now I'm in a place where I'm not so sure about the neighborhood, so I should let I should look at more of my neighbors, than just the closest one.

1.2.4 Cost of the House Two

Okay, so, how man do you want to look at? Well in this case it, you know, I feel like I could draw a, a kind of extended city block zone and capture maybe, I don't know, five of the points. Okay, let's do it. So let's find our five, our five nearest neighbors. So let's see. This is clearly close, that's one over here. I'd say this is close. I'll say this one is close. This one's close. None of the other blue ones are actually that close. And I'd say that's the next closest one, so here are my little five points. That all seem relative near. So what does that tell you? Well, I mean, it's, it feels like it suggests that red is not a bad choice here. Mm It's in a reddish part of town. Yeah, I get that. So, so you think it's a pretty, fair thing to bet that this should be red then? Yeah I mean I think that if you were really asking me seriously I would wonder about that blue point to the right of the highway and whether that had any influence. That's pretty far away. Yeah, it's not that far away. Well in Atlanta, once you cross highways you might as well be an infinite distance away. Well so, okay, but. That's a good point then. So, I guess I was interpreting your notion of distance as being, you know, like straight line distance on the map. But maybe that doesn't make sense for this kind of neighborhood example. Hm, no, that's a good point. So, we've been talking about distance sort of implicitly. But this notion of distance. It's actually quite important. So maybe distance is straight-line distance, maybe it's as the crow flies. Maybe it's driving distance. Maybe it has to take into account the fact that, when you cross highways in Atlanta, you're typically moving into a completely different universe. These sorts of things matter. Yeah. So I could imagine I don't know, like Google Maps distance. Right. Or how many paths can you get there and which is the shortest one given the traffic? There's all kinds of things like that you could do. So. So that's fair, that's fair. But that just says that this, this distant, we have to be very careful what we mean by distance and that's okay. But let's just say for the sake of this discussion that these are the closest points by some reasonable measure of distance. So, in that world, would you be happy if you had to pick a single example? a single output, a single label of red ,uh, blue or green. Would you be happy picking red? Yeah, I mean you know, not ecstatic, but okay. That's fair. So, I like this. So, we, we went from just picking our nearest neighbor to picking our nearest neighbors. And ,what's a good value you think we should, we should stick to with neighbors? We started with one and that clearly wasn't good. You picked, at least not in all cases and you came up with five. So what do you think? What, what, if I'm going to call this algorithm something, what do you think five nearest neighbors? What do you think? What should I call it? Five seems good. I mean I feel like that, that's gotta be universal. The number five? Yeah. Well it is in Atlanta but it might not be univeral in wherever it is you are. We'll call it the Georgia Tech nearest neighbors. That doesn't seem like an algorithm that's going to to be used very much. Fair enough. All right. So what about, we could do as many nearest neighbors as is appropriate. Or maybe we should just make it a free parameter and call it K. Ok, I like that. K nearest neighbors, so we'll have K nearest neighbors. And we'll pick our K numbers. Oh, and you said something fancy there, by the way. You said free parameter. I like that. We should, we should come back to that again. 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 Which is the number of neighbors we're going to use. And some notion of distance. Oh, sure. Which here we were kind of using in the sort of obvious way, but there might be other ways we might want to use distance here. Yeah, like I could imagine if the houses, if, had additional features like how many Square footages they had. Right, stuff like that. That would make perfect sense. So, so really distance, we're using distance here in a kind of in an over loaded sense, because this is something on a map. But really distance is a standard for similarity. Similarity, good. It's kind of standard for the opposite of similarity. [LAUGH] Well distance is just a kind of similarity, right? But in case of, you know, points on the map. Similarity, it sort of makes sense because as you said when we were talking about real estate, location, location, location matters. So, there, similarity really is kind of the inverse of distance. But in other ways, things like the number of veterans you have, whether you're one on 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. Okay, so I like this. I think 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 to this, you know, not having an exact data point in the database. So ,maybe we should turn this into an algorithm. Yeah, let's go for it. Okay, let's do that.

1.2.5 K NN

Okay, so what we have here, again, 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, that's the little x, y points, x y one, x y 2, x y 3, so on and so fourth. You're given some kind of distance metric or similarity function. And this is important because this represents the domain knowledge as I think we, we've already said. You get some number of neighbors that you care about, k, hence the k and n, which also, by the way, represents domain knowledge. Tells you something about how many neighbors you think you should have. And then are given some particular new query point and I want to output some kind of answer, some label, some value. 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. Okay. I'm sort of processing this. So the, the data the capital D. Are those pairs and there's a set of pairs? Yes. Ok. And k smallest distances. So this NN this is a set? Yes. And it consists of all the elements in the data that are closest to the query point? Yep. And the so the query point is a parameter of that. Okay. Yeah. Alright. I think I. Oh. And then it's, then the so it's just return. Yeah, so we haven't figured out what to return. So there's two separate cases we've been talking about so far. One is where, we're doing classification, and one is where we're doing regression. So, a question for you would be, what do you think we should when we're doing classification? Sort of, what we were doing before on the map. What will be a way of returning a proper label? So you want to label, not a, like a weight on a label or something like that? No. I want a label. You have to produce an answer. You have to commit to something Michael. Alright. Can I commit to more than one thing? Nope. Okay. So I would say that a reasonable thing to do there would be. Did we get Ys associated with the things in NN? Yeap. So I would go with they should vote. I like that. I think that's a good one, so we'll simply vote and what does it mean to vote? It means, let's see, so feel like there would be a way to represent it in terms of NN, the set. Like do you want me to write it formally? No. Oh, then I would just say The closest point. Whichever yi is most frequent among the closest points wins. Yeah. Right. So you want to find a, a vote of basically a vote of the yi's, that are apart of the neighborhood set. And you take the plurality. Plurality I see. So it's whichever one occurs the most. Right. What if there's ties? It's the mode. The mode. Right. Right. Mmmm. I love mode. What if they're ties? That's a good point. Well, if they are ties among the output, then you're just going to have to pick one. OK. And there's lots of ways you might do that. You might say, well, I'll take the one. That is say, most commonly represented in the data period. Or I'll just randomly pick each time, or any number of ways you might, you c an imagine doing that. The one that's first alphabetically. The one that's first lexicographically? Hm. What about in the regression case? Okay. So in the regression case our y-is are numbers. Uh-huh. And we have the closest Yi's, so we have a bunch of those numbers and it seems like [LAUGH] if you have a pile of numbers and have to return one, a standard thing to do would be to take the average, or the mean. Yeah. Now let's just simply take the mean of the Yi's, and at least there, you don't have to worry about a tie. That's right. Though, I guess, you know. We didn't really deal with the question of what happens if there's more than k small. It's, like, what if they're all exactly the same distance? All n of them are exactly the same distance. So which are the k closest? Well, there's lots of things you could do there. I guess what I would suggest doing, is, take the, If you have more than k that are close, 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. Okay. That seem reasonable? Yeah, I think that's what college rankings do. Actually, that is what college rankings do, and then they, yeah, that's exactly what college rankings do. So, let's do that. We know that college rankings make sense. [LAUGH]. Yeah, those are, they're scientifically proven to be, Youths. scary, scary to people in colleges. That's exactly right. So, here's what we've got, Michael. So, all we do is we take the training data. We have some notion of similarity or distance. We have a notion of the number of neighbors that we care about. We have a query point, we find the K closest to one, you know breaking ties accordingly. And then we basically average in some way, in a way that make sense for classification, in a way they make sense for regression and we are done. 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. Exactly how you choose to implement the mean or the average operation that shows how to do here. And you could put a bunch of different things here and you end up in, completely, you could end up with completely different answer. Mm. By the way, one thing that you might do, just to give you an example of just, how much range there is here. Is rather than doing a simple vote by counting, you could do a vote that is say, weighted by how far away you are. So we could have a weighted vote. Uh-huh. That might help us with ties. That could help with ties. Yeah. You could do a weighted average. Yes, right. So, you're basically saying that the y values that correspond to x values that are closer to the query point have more of an influence on the mean. Which makes some sense, right? No, I think it makes a lot of sense! So, how would you weight that? What would you do? I would weight it by the similarity. Right, so well in this case, the similarity is we have a distance value similarity, so You would have to, you know, weight it by something like one over the distance. Oh I see. Okay. That seems like a hack. Sure but it's a hack that sort of makes sense. Okay. Okay. So anyway. Simple algorithim. Lots and lots of decisions to make here. All of which could in principle have a pretty big effect. And so, in order to see that, I want to do two quizzes that I hope get to heart of this and maybe give us a little bit of insight into how some of these decisions might matter on the one hand, and exactly just how simple or not simple this algorithm turns out to be. Okay? Awesome.

1.2.6 Wont You Compute My Neighbors Question

Okay Michael, I have two quizzes for you. Okay? Yeah, yeah. Here's the first quiz, and here's the way it's set up. I want you to fill in the empty boxes of this table. Okay? Ooh. Got it. There's a lot of empty boxes. There's a lot of empty boxes. Okay, but Okay, let me make sure I understand what's going on here. So we're looking at three different algorithms that are learning algorithms. Yep. There's one One neural net No Okay, one nearest neighbor. Mm-hm K nearest neighbor and linear regression. Yep And for each one you want to know running time and space. Mm-hm. And this is on n points I assume, yeah, n sort, what does it mean for data points to be sorted? So let's assume we're living in a world where all of our data points are you know in r one. Okay. Oh okay that well that. That could be sorted. That could be. Yeah that could be sorted. And that you know we are going to be out putting some real numbers as well. So they're points on a, on a number line. So to make things simple for you. I'm going to say that the points that your given are already sorted. Oh ok alright. And yeah that makes sense. Its just a scaler. So then a query point is going to come in. And then its going to be some value. And were going to have to find the nearest neighbor or do the linear regression or whatever. Right. Alright now that's for running time. For now space your talking about the space of what. How much space you are going to have to do in order to accomplish your task. How much space you going to have to use in order to accomplish your task? So this is kind of like the the. The space that's representing the class enviro. Or the regression. After training. Yes. So actually that question about after training is important. You'll notice I've divided each of these algorithms into two phases. There's the learning phase. How much time it takes to learn. How much space you need to learn. Then there's the query phase. When I give you some new value and you have to output and answer. What's the running time for that and what are the space requirements for that? Okay? You got that? Yeah I want that for each one. Of these three algorithms. Except for one nearest neighbor which the, it appears as though you filled in for me to get me started. Right so just to get you started and make it easier for you know to know what I'm talking about. I'm talking about big o times here. Right. I'm not going to make you write out big o. Big o is implicit. So if we look at one nearest neighbor, and we ask well what's the running time of learning? Well, it's constant. Right? Because there's no learning. I see. You just take that sorted set of data points and you just pass it along through the query here. Right. Now, you could say that" Well, I'm going to take the data points or I'm going to copy them into this database," and so it's linear. But let's assume they already come in a data base, or some data structure that you can use, okay? Gotcha. Okay, so now that actually brings us to a nice little question about how much space, learning takes here.

And, well because you have to store those points, and keep them around. The space requirements are big O of N. Yeah, that makes sense. Okay, good. So given that as an example. Do you think your one example in your data base. Mm, do you think you can use that to build up labels for all the rest of the phases of the different algorithms? Yeah, I think so. Okay, cool. Go for it.

1.2.7 Wont You Compute My Neighbors Solution

Okay Michael, are you ready? I am afraid so. All right, which one do you want to fill out first? Let's just do them in order, so ,one nearest neighbor. You, you explained, how the training works. We just take the assorted list, and leave it there. Mm-hm. And we have the classifier, or the aggressor itself ,has linear space, and now at query time, we need to find the nearest neighbor, Um-huh. Which we could do by taking the query point, and running through the whole list ,and seeing which one it's closest to, but, because, it's sorted I think we, we outta be able to use binary search and, and in log time, find the closest, point to the query. That's exactly right, you should be able to do that in log base, two time. What if it weren't sorted? Yeah then, like I said, I think you could just scan through the whole list and that would be linear time and that's not a big deal. Right, yeah, we could do linear time, but, because I gave you a sorted list, because, I'm so helpful, you can do it in [INAUDIBLE] time, okay, but. That was, that was very, very thoughtful of you. It was I thought, I thought of her, so what about on the, space side? Alright, so the amount of space that you need to process its query is linear. We don't need to take any special, set aside, space beyond, a couple simple variables. And the data that we're given, which we've already accounted for. Right, so then why would it be linear, if we accounted for it? Did I say linear? I meant constant. Yes, yes, that's right, constant. That's what you meant. That's what you said. That's what happened. [LAUGH] Okay. It's a good thing, this wasn't being recorded, so we could verify one way or the other. [LAUGH] It is a good thing, maybe we'll look it up on Wikipedia, and it'll say confusingly [LAUGH] Linear sometimes use to mean constant. Constant. [LAUGH]. Yeah, that is pretty confusing. Okay, what about k and n? Alright, K and n. So k and n, so the training process, the learning process, is exactly the same, as it is for one year stamper, which is to say you do nothing and you pass all the data forward, to the, The query processor. So it's going to be 1 n. That is correct, nice. Now querying, seems like its a little more subtle. So we can find the single nearest neighbor in log and time. Mm-mm. Where we going to get the other K minus one? So, I'm pretty sure, that ,once we find the nearest neighbor we can kind of start doing a little, Spread out search, from there, until we found the k nearest neighbors. Sure. So, you're saying, you know, you've got these points. They're already in a line, you find the nearest neighbor. You know ,the, the next nearest neighbors have to be within k of the points surrounding it, and so you can just move in kind of, either direction and pick them up as you go. Yeah, something like that. Okay. I mean the way that, the way that I was thinking about it is that, I, I think you can use the same algorithm that you used for merging lists, in merge sort, but here the lists actually corresponds, to being, to the left of the query point, and being to the right of the query point and they are both sorted ,in terms of their distance from the query point. Sure, yeah, I buy that. So, so that ought to give us log n, plus k. Okay, so, do we need to write the k? I'm going to say yes, because, if k is on the order of like, n over 2, then it's going to dominate. If k is on the order of log n, then it's not going to dominate. Mm, that's a good point. So, yeah, we'll do k. I will point out that if k is on the order of n over 2, you're right, it will dominate, and then really this is big o of n. That's right. But if it's on the order of log n, then it's just log n plus n, and so it's a big old long n, log n. But ,you're right, so we should probably keep the k around because, we don't know its relationship to n. Okay, fair enough. Okay, what about the space requirements? We know one bit of relationship, it's smaller than or equal to n. That's true. Because that would be really weird, if I gave you ten data points and asked you for the 20 nearest neighbors. That's the sort of thing you would do. It's the sort of thing you would do, but then ,it would be really confusing. No, no, no, it's the sort of thing YOU would do, again, let's go to Wikipedia. Confusingly, twenty sometimes means ten. [LAUGH] Okay. So what about space? Space, so, I don't understand why, it would ever need more than constant space. So, so we're going to, zip around in that. I mean, I guess, if do it really badly ,we can use K space. To kind of copy over what those, possible nearest neighbors are. But ,we don't need to keep track of them. We can just point to them in place, so it's constant. Okay, yea that's true in fact, because, it's sorted all you really need to know is the beginning and the ending. So, that's two things that's constant. Okay, cool, alright, good, so what about linear regression? Your favorite little algorithm thing that you did? When we talked about this before. I do like, linear regression. The learning in this case, is what we are mapping, real number input to a real number output. The way we are doing that is we are taking, its probably M X plus B sort of form [CROSSTALK]. We need to find, the multiplier and the additive constant. which, It seems, like, in general doing a regression involves, inverting a matrix. But, in this case,I think the matrix that we're talking about is of constant size. So inverting, is, constant time. I think, it's as easy ,as basically

just scanning through the list ,to populate that, that ,constant size matrix. So, I'm going to say order n. Yep. To process the data. That is correct. There's probably a really nice algorithm for that. Yeah, probably some kind of linear regression algorithm. Yeah. [LAUGH] No, I mean like the general linear regression algorithm is, is involves inverting a matrix. Right. Or something like it, something equivalent to it. But here because, we're, it's all in scaler land, I think it's simpler. Yeah, I think that's right. Okay ,so what about space? All right, so space interpreted the data that you passed forward from the learning algorithm, to the regressor, is just, well MX plus B. It's just M and B, which is constant. There's the two numbers. Right, that's 2, 2 is like 1, [UNKNOWN] large values [LAUGH] of 1, so it's constant. Yeah, All right, now at query time, you give me an X. I multiply, it by M and add B, so that's constant time [CROSSTALK]. So, before the query, cost was expensive and the learning cost was cheap. And now we've kind of swapped that around. Yeah, we have, so space would be. Space, oh, that you're asking me? Yeah. Space for the query would be constant as well. Right, exactly, so yeah, so you made a good point here. So earlier on, we had the situation where learning was fast, was constant, and querying was, you know, not as fast. It was, you know, probably logarithmic. But, in the regression case, learning was expensive and the querying was easy, so we swapped around, that's exactly right. So, why would you care about that, well, let's see, I'll point out something, which is though, even though we swapped out what was expensive in terms of time and what wasn't, you'll notice that. It's only logarithmic at query time for these first two but it's linear for the learning time, in, linear regression, so doesn't that mean that linear regression is always slower and worse? No really, because we only have to learn once, but we can query many, many times. Right, right. So I guess if we query more than, you know, n times for example ,it'll certainly be, worse overall. In terms of running time. That's right. Okay. Though, it's, though it's interesting, because like when I see numbers like this, my, my algorithm, hat tells me that I should try to balance them a little bit more. Like, here it's, it's you can make, learning ,essentially free and the other one you can make querying essentially free. Really you want to split those, somewhat evenly. Though it's, not obvious to me how you would do that. Like, square root of n, learning time and then square root of n query time or something like that. Yeah, but you did say something else that was important right? Which is that you only have to learn once. Yeah, It's true. So, the balance you know, really depends on how often you're going to do querying and exactly, what power it gives you. I mean, the trade off is really there. Don't you think? Yeah, I guess so. I mean so, so specifically, in the, in the version where you just query ones. Right. Then the balance thing could be more interesting. Sure, okay, cool. All right anything else you want to observe about this. Let's see, we got the trade off between learning versus querying. So, either you do all your work upfront, or, you put it ,off ,and do your work only, when you're forced to at query time. Yeah, I guess, well one thing, is, I want to point out that there's a nice Mr. Rodgers, reference in the title. That was, that was very cool. Thank you very much. And the second thing, is that it does strike me, in a sense, that what's going on here for the nearest neighbor algorithms, is that you just put off ,doing any work until you absolutely have to. Mm-hm. Which strikes me as kind of a procrastinatory approach. So that's a good word, that you used there, procrastinate. The words that people use, in the literature, are ,uh, lazy. Mm. They say that these are lazy learners, versus something like linear regression, which is an eager learner. Eager. Yes. So, linear regression, is eager, it wants to learn right away, and it does, but 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. I See, so I guess its the case, that, if we never query it, then the lazy learner definitely comes out ahead. Right, that makes sense. Yeah. It's just in time learning, or JIL. [LAUGH] Or JITL, I guess, which doesn't roll off the tongue anywhere near as well. Okay, cool. So we've gotten through this quiz. Would you like to do another one? Yeah, I just have to get this JITL off my tongue. [LAUGH]

1.2.8 Domain K NNowledge Question

Okay Michael, so here's our second ,quiz, is a row. In the last quiz, we talked about running time and space time, but now we're going to talk about ,how the k-nn algorithm, actually works. And in particular how different choices, between distance metrics, values of k, and how you're going to put them together, can give you different answers, okay? So, what I have over here on the left is training data. This is a regression problem and you're training data is made up of xy pairs. X is two dimensional. Okay? So this is a function from R squared to some value in R1. Okay? Mm-hm. So, the first dimension represents, something and the second dimension, represents something. And then there's some particular, output over here. And what I want you to do, is given a query point, 4, 2 produce what the proper y or output ought to be, given all of this training did. You're with me? Yeah. Okay, so I want you to do it in four different cases, I want you to do it in the case where, your distance matrix is euclidean, Okay. The distance metric, in R2? Yes. Oh I see because, we're going to measure the distance between the query and the different data points. Right. Yeah.

Okay. Uh-huh. Mm-hm. So it's euclidean, for a case of one nearest neighbor and three nearest neighbor and I want you to take, for example, in the three nearest neighbor case. I want you take their output and average them. Okay? Okay. Now, in the I also want you to do the same thing. But in the case where instead of using Euclidean distance, we use Manhattan distance. But again, for both 1 nearest neighbor and 3 nearest neighbor And in any case where we have ties, like in three nearest neighbor where we absolutely have to have at least three of these things show up, just let 'em average. Okay? Got you. Now we're doing averaging instead of straight voting, because, this is a regression problem. Got it. Okay. Any questions? Maybe. Let's see. Three nearest neighbor. And so if there's ties, we, we use the college ranking trick of including everybody who's at least as good as the k, largest or k closest. Yes, exactly. Okay, yeah, no I think, I think I can take a stab at this. Okay, cool then go.

1.2.9 Domain K NNowledge Solution

Alright Michael, you ready? Yeah. Okay. What's the answer? Walk me through. I will walk you through. Alright. So let's, let's do this. Let's write the Euclidean distance. Well let's write the Manhattan distance, because I don't, I don't want to take square root to my head. Okay. Let's write the Manhattan distances next to the Xs. OKay. Or the Ys. Alright. Either way. Let's do it next to the Xs. So this is the Manhattan distance or MD as the cool kids call it. [LAUGH] Is that true? Yea. The cool kids called it L one. No, no, no have you ever heard a cool kid ever say something like L one? Well, to me the cool kids are the people at neps who know more math than I do. Yea, do you think any of them are going to watch this video? Actually I'm afraid all of them are going to watch this video. Now I'm really afraid. Mm-Hm so you better get it right, every ones watching. All right, well let me do, let me complete the Manhattan distances. So the first one what you do is you take the 1 minus 4. Mm-Hm. And that's three. And you take the 6 minus 2 and that's 4. And you add the two together and you get 7. Which interestingly is the same as y, but I think that's a coincidence. Okay. [CROSSTALK] And now I'll do all the rest of them 'cause I pre-computed them of Four, six, eight, four, six. Alright, so now we've got it set up so we can do one and three nearest neighbor relatively quickly. So, the one nearest neighbor, the closest distance, is four. Mm–hm. But unfortauntely there are two points that have that two comma four in set number one. We have outputs of eight and 50 because they. Almost agree with each other. Uh-huh. Not. And if we take the average of those two things we get 29. Yep. That is correct Michael. Great now in terms of the three nearest neighbors we have the fours and the sixes. So the four, three nearest neighbors. Yep. Somewhat awkwardly. And the we have the average of those things which is what. Eight, fifty and sixteen and sixty eight which gets us thirty five point five. Right. Obviously. [LAUGH] Okay. Alright, so that was pretty straightforward. And those answers aren't too far off from one another. So what about the Euclidean case? Alright, so one thing to point out. I, I was worried about computing square roots but it occurs to me that I actually don't have to compute square roots because that's the monotonic transformation, and we only care about the orders. Hm, okay. So for Euclidean distance, or as I like to call him casually, ED, Mmhm. We can just take the square differences summed up. Okay, so this would be ED squared. Yes, it would be ED squared. Okay, ED. Good. So the first one, it'll be the one minus four is three, squared is nine. And the 6 minus 2 is 4, squared is 16. And 9 plus 16 is 25. So the first one will be 25. And notice the square of 25 is pretty easy to compute. Yeah, but the other ones aren't going to be. It just so happens that we've got a pythagorean triple on our hands. Mm. I love those. Al right so the remaining ones, the x squared are eight, 26, 40, ten, and 20. Hm, none of those are easily square rootable. Exactly, though 40 feels like it really was trying and failed. Yeah. An eight over shot and now it's a perfect cube. So, eight is the smallest distance. Yep. And again, seemingly, coincidentally, they Y value associated with that, is eight. Hm. So an eight, eight is our answer. Good and that's correct. And the three closest are eight, ten and 20. Mm-hmm. And if we average the Y values for those that's eight, 50 and 68, which gives us an average of 42. The meaning of life, the universe. And pretty much everything? Yes! And that is absolutely correct. So that's kind of That's kind of cool that you get completely different answers depending upon what you do. Yeah, it does seem very different, doesn't it? I mean there's like several orders of magnitude spread here. Well, that's. Maybe not orders of magnitude but orders of doubling. Yes, there are orders of doubling spread. Well, you know what Michael, I actually had a specific function in mind when I did this. Okay! Let's find out which one is the right one! Well, the function I had was Y was equal to the first dimension squared plus the second dimension. So, let's call that X1 and X2, and this was actually the function that I had in place. So, you square the first term and you add the second. Okay, and so like looking at the second last one, for example, seven squared is 49 plus one is 50 Oh, [UNKNOWN]. It's very consistent. Thank you. So what would be the actual answer for four comma two? Okay so four squared is 16 plus two is 18. Which is close to none of them. Right. So there's a lesson here, there's several lessons here. And one lesson I don't want you to take away. So here's the lesson. So

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. And this sort of suggests, that maybe this thing doesn't do very well. Uh, kNN doesn't do very well because none of these are close to 18. That seems a little sad. But I've good new for you Michael. Okay. The good news is that, actually 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, and it has to do with this sort of. Fundamental assumptions in bias of kNN, I happen to pick an example that sort of violates 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. Okay that sounds useful. Okay, so let's do that.

1.2.10 K NN Bias

Ok, Michael, so I'm going to talk a little bit about bias. In particular, the preference bias for K [INAUDIBLE]. So, let me remind you what preference bias is. Preference bias is kind of our notion of why we would prefer one hypothesis over another. And they say all things, 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, Occam's Razor, those sorts of things were the ways that we expressed our preferences over various hypothesis. And kNN is no exception. It also has preference by its built in as does every algorithm of any note. So I just wanted to go through three that I thought of is as being indicitive of this bias. And they're, kind of all related to one another. So the first one is a notion of locality. Right? There's this idea that near points are similiar to one another. Does that make sense to you? Yeah. Yeah. That was really important. It came out nicely in the the real estate example. Right. So the whole idea. The whole thing we are using to generalize from one thing to another is this notion of nearness. Right. And exactly how this notion of nearness works out. Is embedded in whatever distance function we happen to be given. And so, 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 is there like, a perfect distance function for a given problem? There's certainly a perfect distance function for any particular problem. Yeah, that's what I mean. Not one that works for the universe, but one, like, you know, if I give you a problem and you can work on it all day long. Can you find, is there a notion that there"s a distance function that would capture things perfectly? Well, it has to be the case for any given fixed problem. That there is some distance function that minimizes, say, sum of squared errors or something like that. First is some other distance function. Right? Okay. That has to be the case. So there, there always to be at least one best distance function given everything else is fixed. That makes sense. Right. Now, what that is, who knows. Maybe you finding it might be. Arbitrarily difficult. Because there's at least an infinite, there's at least an infinite number of distance functions. Well yeah, I was thinking that, that for latter to find distance functions to be anything we want. What about a distance function that said the distance between all the things that have the same answer, is zero. Mm-hm. And the distance between them and the ones that have different answers is you know, infinity or something big. Yeah. And then, then the distance function, like, somehow already has built in the solution to the problem because it's already put the things that have the same answers together. Right, you could do that, and of course, doing that would require again solving the original problem. But yeah, so. So, such a function has to exist, or, well, you know, there's always noise. What if there's noise in your data, you know? But some such function like that has to exist, the question is finding it. But I think the real point to take there is, there are some good distance functions for our problem and there are some bad distance functions for our problem. And how you pick those is really fundamental assumption your making about the domain. That's why it's domain knowledge. Yeah, that sounds right. Mm 'Kay. So, locality however it's expressed to the distance function, that is similarity. It's 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. Alright. That we are by choosing to average. And by choosing to look at points that are similar to one another. We are expecting functions to behave, smoothly. Alright, so, you know, in the 2D case. It's, it's kind of easy to see, right? You, you, you, you have these, these sort of points and you're basically saying, look, 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. Does that make sense? I mean, it seems like we're defining to be pretty

similar to locality. In this case. So I'm, I'm drawing an example, such that, you know, whatever we meant by locality has already been kind of expressed in the graph. Okay. And you know, by picking, you know, this is really for pedagogical reasons. You know can imagine, this you know, these are points that live in 77,000 dimensions, and it's impossible to actually visualize them much less draw them. And I could try. [LAUGH] But here's, here's three dimensions and here's the fourth dimension. I think I'm going to get tired before I hit seven and seven thousand but, you know, you kind of, you kind of get the idea, right? That, if. In, you know, if you can imagine in your head points that are really near one another in some space, you kind of hope that they behave similarly. Right. Right. Okay, so locality and smoothness. And I think these make sense. I mean, these, this is hardly the only algorithm that makes these kind of assumptions. But there is another assumption which is a bit more subtle I think. Which is worth spending a second talking about, which is, 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, sort of, 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 not only did they matter, they mattered equally. Right. So think about the the last quiz I gave you. Right. It said y equals x 1 squared plus x 2. And you noticed we got answers that were wildly off from what the actual answer was. Well if I know that the first dimension. The first feature is going to be squared and the second one is not going to be squared. Do you think either one of these features is more important or more important to get right? Okay. Right. Trying to think about what that might mean. So, if, yea its definitely the case that when you look for similar examples in the database you want to care more about X1 because a little bit of a difference in X1 gets squared out. Right? It can lead to a very large difference in the corresponding Y value. Whereas in the x2's, it's not quite as crucial. Th, th, the, if you're off a little bit more, then you're off a little bit more, it's just a linear relationship. So yeah, 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. Right so, we probably would have gotten different, I'm not going to go through this but, we probably would have gotten different answers if, in the Euclidian or Manhattan case we had instead of just taking the difference between the first two The first dimensions, we had taken that difference and squared it. And then in the case, including this and squaring it again, and then some of those things that were closer in the first dimension instead of the second dimension would've looked more similar and we might've gotten better answer. That's probably a good exercise to go back and do for someone else. [LAUGH] Yeah, I was thinking of doing it right now, but yeah, probably should leave it for other people. Well you can do it if you want to. So did you do it Michael? I did. And? So it's a kind of now a mix between the Manhattan distance and the Euclidian distance. So, I'm taking the first component, take the difference, square it. Mm-hm. Take the second component, take the difference, absolute value it. And add those two things together. Sure. All right. So if I do that, with one nearest neighbor, I still get that tie, but the output answer ends up being 12. Hm. Which is better than 24.7. And that's better than eight, which is what it was before. So the eight has gone up to 12, which is better than the other one, which I think was 35.5, comes down to 29.5 Close here again to the correct answer which is eighteen. So in both cases it kind of pushed in the right direction, it was using more, of the, the answers that were relevant and fewer of the answers that were not relevant. Right. There you go. So the notion of relevance by the way, turns out to be very, very important. And highlights a weakness of kNN. So this brings me to a kind of theorem or fundamental results of a machine learning that is particularly relevant to kNN but its actually relevant everywhere. Do you think its worth while to mention it? Sure it sounds very relevant. Alright let's do it.

1.2.11 Curse of Dimensionality

Okay, Michael. So this notion of having different features or different dimensions throw us off has a name and it's called the Curse of Dimensionality. Oh, nice audio effect. I did like that effect in post-production. And it refers to well, a very particular thing. So let me just read out what it refers to. As the number of features or equivalently ,uh, dimensions grows that is as we add more and more features we go x of 1, x of two then we add x of three, add more and more of these features. As those features grows or as the number of dimensions grow ,the amount of data ,that we need to generalize accurately also grows exponentially. Now 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, you just, you sort of can't win. I think everybody knows that in the sense that if you look, I've done this experiment actually, if you look in the popular press like, you know, Time Magazine Or New York Times, USA Today. People will use the word exponentially sometimes to mean exponentially, and sometimes to mean, a lot. Yeah that's actually a pet peeve of mine. The whole notion of, Me too. Oh, it's exponentially bigger. No, that's, that's not meaningful. If you're saying I have one point. And then I have another point, and I want to say this one

point is exponentially bigger than this one. That's meaningless! It's also liberally bigger than that one. Exponentially refers to a trend. Again, their,their,their not talking about the mathematical relationship. They just mean a lot. Okay, so they're wrong. And it bothers me deeply but I'm willing to accept it for the purposes of this discussion. Okay. Exponentially means bad. It means that we need more, and more, and more, and more data as we add features and dimensions. Now as a machine learning person this is a real problem right, because What you want to do, or like what your instinct tells you to do is, oh ,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 and more features. You know, 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. Mm. This is a very serious problem, and it sort of captures, a little bit of what the difficulties are in k and n. If you have a di, if you have 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. [CROSSTALK] Yeah, that makes a lot of sense. Yeah, it seems a little scary. So, you know, I think you can say these words, and the words sort of make sense, but I think it helps to kind of draw a picture, and so I'm going to draw a little picture. Okay? Yeah. All right.

1.2.12 Curse of Dimensionality Two

Okay, Michael, so let's, let's look at this little line segment, okay? And then 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, alright? That's kind of K nearest neighbor-y/-ish. So, I'm going to put a little X here, I'll put one here, I'll put one here, put one here, put one here, here, here, here, here, here. Is that ten? Three... six. Nine, ten. Ten. Okay. And 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. Does that make sense? Yeah, so, so it's representing it in the sense that, that point. Uh,when you're trying to estimate values of other places on the line it's going to default as the nearest neighbor to being that point so there's a very small little neighborhood of the red line segment that is covered by each of the green X's. That is exactly right and in fact each one of these green X's represents. How much of this segment? Each of the green X's covers one tenth? That's exactly right. You cover one tenth. Alright Michael, so let's say I move from a line segment now to a two dimensional space. So a little square segment. If that's the right technical term. And I've taken my little ten x's, and I put them down here before. Well, here's something you'll notice; you'll notice that each one of these x's is going to still end up representing one-tenth of all of this space, but you'll also notice that, that, that it's representing now you know. Really really really really big. I see. So one way of putting it is, you know if you think about the farthest point, as opposed to the furthest point which would be incorrect. [LAUGH] The farthest point that this particular first x over here is representing, its got some distance here. Over here, the farthest point from this x, the distance is very, very far away. So, a question would be, how can I make it so that each of the x's I put down represents the same amount of. I don't know diameter or distance as the xs in this line segment over here. So what do you think I have to do? I feel like you need to fill up the square with xs. Yeah, that's exactly right so let's do that. So filling em up Michael as you suggested. You'll notice that at least if we pretend that I drew this right. Each of these X's is now going to end up being the nearest neighbor for a little square like this, and the diameter of these little squares are going to be the same as the diameter of these little line segments. Yeah, I agree for some definition of the word diameter. Yes, and for some definition of our demonstration. Okay, so how many of these X's are there, Michael? Can you tell? You want to count? [LAUGH]. I'm going to multiple [CROSSTALK] cause it looks like you did ten by ten so that'll be 100. That'll be 100. So each one now holds a hundredth of the space, and I went from needing ten points to, of course, 100 points in order to cover the same amount of space. Alright, so that definitely seems like the mild rebuke of dimensionality. Yes. But doesn't seem that bad. Okay well, what happens if I now move into three dimensions? So now, if I want to cover the same amount of, you know, diameter space for, you know, sufficient definition of diameter. I'm going to have to do a bunch of copying and pasting that I'm not willing to do so, you know, there would be more x's here and you know, there will be x's there and an x here and it'll just kind of go and fill up some space and you know, I'm not going to do this whatever but [SOUND] and you'll get x's everywhere. And you notice, I need a lot more X's than I had before. And by the way, I'm just showing you the outside of this little cube, there are actually X's on the inside as well, that you can't see. How many X's do you think

I have? I don't think you drew any X's. You're just like scribbling on the side of the cube. These are X's. You, you were doing so well for awhile, and then just lost it entirely. Well, wouldn't you lose it if you had to write 1000 x's. Hm. No because I would use computers to help me but yes, yes it is very frustrating to have to have that many x's. And so but in particular in this case we're talking about data points in a nearest neighbor method and that, boy that does seem like a big growth from ten to a 100 to 1000. In fact the growth is, exponential. Exponential ¿Right. So if we went into four dimensions, which I'm not going to draw, then we would need not 1,000, but 10,000 points. And if five dimensions we would need 100,000 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. Wow. Right. So this is really problematic right. In my little nearest neighbor method, wanted to be able to say, well, I want to make sure the neighborhood remains small, as I add dimensions, I'm going to need to grow the number of points that I have in my training set exponentially. And that's really bad. And by the way, this isn't just an issue of 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. So the curse of dimensionality does not just to kNN. It is a curse of dimensionality for ML period [SOUND]. You mean for me? Yes. Because of [INAUDIBLE] [LAUGH] Okay. And that seems really problematic because it's very natural to just keep throwing dimensions into a machine learning problem. Like it's, it's having trouble learning. Let me give it a few more dimensions so, to give it hints. But really what you're doing is just giving it a larger and larger volume to fill. Yeah. 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. Zoinks. Mm-hm. There's an entire series of lessons that we will get eventually that, that deals with this issue. The issue of? Dimensionality. Finding the right dimensionality? Yeah. That would be a useful thing. It would. But it's far off in the future. It's like infinitely far in the future. So we'll worry about that in a few weeks. Okay. Okay. All right. So there you go, Michael. Curse of Dimensionality is real and it's a real problem. Where did that term come from, it's a cool term. I think it came from, oh what's his name. Bellman. Oh, Bellman, like the dynamic programming guy. Yeah, the dynamic programming guy, uh,the Bellman of Bellman equation guy. Which we haven't gotten to yet in the course. Which we haven't gotten to in the course but we will get to in the course. Because it's central. So it looks like actually, the element's central to a lot of things. Wow. Sometimes it gives us equations that helps us but sometimes it gives us curses. [LAUGH] Curses Betterman. Foiled again. [LAUGH]

1.2.13 Some Other Stuff

Okay, Michael so we talked a little bit about the curse of dimensionality, but I think it's worthwhile to talk about some other stuff that comes up. We've been sort of skirting around this and you know bring it up in various ways throughout our discussion so far. But I think it's worthwhile kind of writing them all down on a slide and trying to think through them for a little bit. So ,uh, the other stuff that comes up in [UNKNOWN] mainly comes up in these sort of assumptions we make about parameters to the algorithm. So the one we talked about ,uh, probably the most is our distance measure, you know our distance between some X and some query point Q and we've explored a couple. We looked at Eucudean and we looked at Manhattan. And we even looked at weighted versions of those. And this really matters, I've said this before but I really think it bears repeating that 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. So I, so I have a question about these these distance functions. So you mentioned Euclidean and Manhattan, are there other distance functions that the students should know? Like, things that they, that might come up, or things that they should think of first if they have a particular kind of data? yeah, there's a, there's a ton of them. I think Well, first off, it, it's probably worth pointing out that this, this notion of weighted distance is one way to deal with the curse of dimensionality. You can weight different dimensions differently. And that would be one, and you might come up with sort of automatic ways of doing that. That, that's sort of worth mentioning. But you will notice that both Euclidean and Manhattan distance at least as we have talked about them, are really useful for things like regression. Their kind of assuming that you have numbers in that subtraction kind of makes sense. But there are other functions, distance functions that you might do if you are dealing with cases like, I don't know Discrete data, right? Where instead of it all being numbers, it's colors, or something like that. Alright so, your distance might be mismatches. For example, or it might be a mixture of those. In fact, one of the nice things about KNN, is that we've been talking about it with points, because it's sort of easy to think about it that way. But this distance function is just a black box. You can take Arbitrary things and try to decide how similar they are based on whatever you know about the domain and that could be very

useful. So ,you could talk about images right, where you take pictures of people and you know rather than doing something like a pixel by pixel comparison, you try to line up their eyes. And look at their mouths, and try to see if they're the same shape you know things like that, that might be more complicated and and perhaps even arbitrarily computational to determine notions of similarity so really this idea of distance in similarity tells you a lot about your domain and what you believe about it. ,another thing that's worth what what's pushing on a little bit is how you pick k. Well there's no good was to pick k you just You just have to know something about it, but I want to think about a particular case. Well, what if we end up in a world where K=N. Well, that would be silly. Why would it be silly? Well, so if K=N, then what you're doing is you're taking, so in the case of regression for example, you're taking all of the data points and averaging the. Y values together. Basically ignoring the query. So, you end up with a constant function. But that's only if you do a simple average. What if you do a weighted average? A weighted average. So the near, the points that are the query are going to get more weight in the average, so that acually will be diffrent. Even though k equals n, it will be different depending on where you actually put your query down. Exactly. That's exactly right so, for example, if I have a little bunch of points like this say. Where you notice it kind of looks like I have two different lines here and I can pick a query point way over here, all of these points are going to influence me as oppose to these points and so I'm going to end up. Estimating with something that looks more like this because these points over here won't have much to say. But if I have a query point that's way over here somewhere these points are going to matter and I'm going to end up looking something looks a little bit more like this than like that. Now I'm drawing these as lines. They won't exactly look like lines because these points will have some influence. They'll be more curvy than that. But the point is that near, near the place we want to do the query it will look To be more strongly influenced by these points over here or these points over here depending on where you are. Well that gives me an idea. Oh, what kind of idea does it give you? Well, what about instead of just taking a weighted average, what about using the distance matrix to pick up some of the points? And then do a different regression on that substantive point. Right, I like that. So we can replace this whole notion of average with a more kind of, regression-y thing. So it actually, instead of using the same value for the whole patch. Actually, it still continues to use the input values. Yeah. So, in fact, average is just a special case of a kind of regression, right? Mm hm, mm hm. Right? So this actually has a name, believe it or not. It's actually called locally weighted regression. Yeah, so this 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. Neat, Yeah. Add that works out very well. And again, it gives you a little bit of power. So here's something I don't think is very obvious until it's pointed out to you. Which is this notion of replacing the average with a more general regression or even classification function. It actually allows you to do something more powerful than it seems. So let's imagine that we were going to do locally weighted regression and we were going to do, in fact, linear regression. So, what would locally- weighted linear regression look like? Well, if we go back to this example over here on the left basically, you take all the points that are nearby and you try to fit a line to it. And, so you would end up with stuff that looked, pretty much, like this. While you're over here, you would get the line like this, but while you were over here you'd get a line like this. Then, somewhere in the middle you would get lines that started to look like this and And you would end up with something that kind of ended up looking a lot like a curve. So that's kind of cool because you notice that we start with a hypothesis state of lines and this locally weighted linear regression. But then we end up actually being able to represent a hypothesis space that is strictly bigger. than the set of lines. Hm. So we can use a very simple kind of hypothesis space but by using this locally weighted regression we end up with a more complicated space that is complicated, that's made more complicated depending upon the complications that are represented by your data points. So this results, this sort of reveals another bit of power with kNN. Which is, it allows you to take local information and build functions or build concepts around the local things that are similar to you. And that allows you to make arbitrarily complicated functions. Neat. At least in principle. Okay, cool. Alright so you got all that? I, think so yeah. Okay, cool. So then, I think we should wrap up. Nice. Nice.

1.2.14 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 learnerversus 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. [LAUGH] 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. [LAUGH] 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.

1.3 Ensemble B & B

1.3.1 Ensemble Learning Boosting

Hey Charles, how's it goin'? It's going pretty well Michael. How's it going with you? Good, thank you. Good, good, good. Guess what we're going to talk about today? Well, reading off this screen, it looks like maybe ensemble learning, and boosting, whatever that is. Yes, that's exactly what we're going to talk about. We're going to talk about a class of algorithms called ensemble learners. And I think you will See that they're related to some of the stuff that we've been doing already, and in particular we're going to spend most of our time focusing on, boosting. because boosting is my favorite of the ensemble learning algorithms. So you ready for that? Yeah! Let's do it. Okay. So, I want to start this out by, going through a little exercise with you. I want you to think about a problem. Okay. And the particular problem I want you to think about is, spam email. Mm, I think about that a lot. So, normally if we were thinking of this a classification task, right, 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 or you know neural networks or kNN whatever that means with email. We would be coming up with all of these sort of complicated things. I want to propose an alternative which is going to get us to ensemble learn. OK and here's the alternative. 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. Okay, so let me be specific, Michael. We have this problem with spam email. That is, you 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 rules that'll help me to figure that out. And I want you to think about simple rules, so can you think of any simple rules that might indicate that something is spam? Alright I can, yeah I can think of some simple rules. I don't think they would be very good, but they might better than nothing. Like If for example it mentions how manly I am, I, I would be, be willing to believe that was a spam message. So like if the body of the message contains the word manly. Okay, I like that. like that when body contains manly. I like that rule, because I often get non-spam messages talking about manly. So I guess one man's spam is another man's normal email. [LAUGH] I guess that's true. Probably. Any other rules? Sure if it, you know if it comes from my spouse it's probably not spam. OK, so let's see, from spouse. Her name's Lisa. Now we're going to call our spouse. So let's say minus, I'm going to go to plus here, so we know some rules are indicitive of being spam, and some rules are indicitive of not being spam. Okay, anything else? Possibly the length of the message. I guess. Like what? I don't know. I don't know that this would be very accurate, but I think some of this, some of the spam I get sometimes is very, very short just like the, it's like the URL. Like hey, check out this site, and then there's a URL. Hm, I like that. So, we'll just say short. Just contains URLs. Hm, I like those rules. Let's see if we can think of anything else. Oh, how about this one. It's just an image. Hm. I get a lot of those where it's just an image. I see, like in it's it's and if you look at the picture it's all various pharmaceuticals from Canada. Exactly. Here's one I get a lot. Hm, Lots of misspelled words that you end up reading as being a real word. Hm. But I don't know how I'd write that as a rule. Or you could just list the words. Like rules that, words that have already been modified in that way. I guess so. Yeah, kind of a, kind of a black list, a black list of words. Okay so, words like, I would say manly, but you were saying prawn. Or whatever that says. yeah, so they're and they're tons of these. Right? I mean, another one that's, that's very popular if you're old enough anyway is this one, remember this one? Oh, sure that was sometimes a virus, right? Yes. Our young, our younger viewers will not know this but this was one of the first big spam messages that would get out there. Make money fast. And there's tons and tons of these. We could come up with a bunch of them. Now, here's something they all kind of have in common, Michael, and you've touched on this all ready. 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. Right. So the word manly is evidence but it's not enough to decide whether something is spam or not. It's from your spouse, it's evidence it's not spam, but sometimes you get messages from your spouse that are in fact spam, because in fact, she didn't actually send them. You know, and so on and so forth. And sometimes she did email from Princes in Nigeria. I didn't. And they're not always spam. I, I actually do, but any case, sometimes people are asking you for money, and maybe that's message you want to ignore, but it isn't necessarily spam. And some people are very interested in getting like this and don't consider it spam, right? So, so, okay, so I can see that these would all maybe provide some evidence, but it seems really hard tp figure out the right way of combining them all together to I don't know, make a decision. Right, this is exactly right.

And by the way, if you think about something like decision tree, you could. There's really a sort of 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. Right? So, we need to figure out how to do that here and that is the fundamental notion of ensemble learning. But wait, isn't, couldn't you also do something similar with something like neural net. Right? Where each of these now becomes a feature and we're just trying to learn weights for combining them all together. So That would kind of satisfy what you were talking about. True, I mean I think the the difference here in this case and and I think you're absolutely right but one difference here 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. But you're absolutely right. You could think of neural networks as being an ensemble of little parts. Sometimes hard to understand, but an ensemble nonetheless.

1.3.2 Ensemble Learning Simple Rules

So, the characteristic of ensemble learning is first this, that you take a bunch of simple rules, all of which kind of make sense and you can see as sort of helping. But on their own, individually, do not give you a good answer. And then you magically combine them in some way to create a more complex rule, that in fact, works really well. And ensemble learning algorithms have a sort of basic form to them, that can be described in just one or two lines. So let me do that and then we can start wondering a little bit how we're going to make that real. So here's the basic form of an ensemble learning algorithm. Basically you learn over a subset of the data, and that generates some kind of a rule. And then you learn over another subset of the data and that generates a different rule. And then you learn over another subset of the data and that generates yet a third rule, and yet a fourth rule, and yet a fifth rule, and so on and so forth. And then eventually you take all of those rules and you combine them into one of these complex rules. So, we might imagine in the email case that 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. And 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. And 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 set of the data, but aren't necessarily good at a large set of the data. And then after you've collected these rules, you combine them in some way, and there you go. And that's really the beginning and the end of ensemble learning. So wait. So, when you say manly was in a lot of the positive examples. Do you mean like it distinguishes the positive and the negative example? So it should also not be in the negative examples. That's right. That's exactly right. So 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. And, and now why does it, why are we are looking at subsets of the data? I don't understand why we can't just look at all, the whole data. Well, if we look at all of the data, then it's going to be hard to come up with these, these simple rules. That's the basic answer. Actually, ask me that question a little bit later, when we talk about overfitting, and I think I'll have a good answer for you. Okay, so here we go Michael. 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.

1.3.3 Ensemble Learning Algorithm

Here's the Ensemble Learning algorithm. We're done, Michael, we're done with the entire lesson. We don't have to do anything else anymore. We know that we're supposed to look over subset of data, pick up rules, and then combine them. So, what else do you need to know in order to write your first Ensemble Learning algorithm? So, I'm already kind of uncomfortable with this notion of" combine," right? So, like, I can think of lots of really dumb ways to combine things. Like, choose one at random or, you know, I don't know, add em all up and divide by pi I mean so, so presumably there's gotta be some intelligence in how this combination is taking place Yes, you would think so, but your not at all bothered about how you pick a subset? Oh ,I was imaging you meant random subsets. Oh ,so you'd automated assumption about how we were going to pick a subset. You just [CROSSTALK] weren't sure how to combine them. Well actually let's explore that for a minute. Here's kind of the dumbest thing you can imagine doing. That turns out to work out pretty well. We're going to pick subsets, by, I'm going to say uniformly. Just to be specific about it. So ,we're going to do the dumbest thing that we can think of, or one one of the dumbest things you could

think of. Or maybe ,we should say simplest and not dumbest so as not to, to, to make a value judgment. That you can think of doing which would be to just uniformly randomly Choose among some of the data, and say that's the data I'm going to look at, and then I'm going to apply some learning algorithm to it. Is that what you were thinking of Micheal? Yeah. Okay, so just pick a subset of the data, apply a learner to it. I'll get some hypothesis out, I'll get some rule out. And now I'm going to combine them, so since were being simple. Why don't we try doing something simple for combining. Let's imagine, Michael, that we're doing a regression. What's kind of the simplest thing you could do if you have ten different rules which tell you, how you should be predicting some new data point? What's the simplest thing you could imagine doing with it? So, okay, so each of them spits out a number. I guess if we kind of equally believe in each of them, a reasonable thing to do would be to average. Great. So, a very simple way of combining, in the case of regression, would be to average them. We'll simply take the mean. And by the way, why wouldn't we equally believe in each of them. Each one of them learned over a random subset of the data. You have no reason. Well. To believe one's better than the other. There's a couple of reasons. One ,it could be a bad random subset. I don't know how I would measure that. I could be a good random subset. Yeah. Then we'd want, we'd want that to count more in the mean. But, but I guess what I was thinking more in terms of maybe for some of the subsets you know, it gets more error than others or it uses more complex rule than others or something. I could imagine that. Actually maybe we can explore how this sort of idea might go wrong. Let's, let's do that. Maybe we can do that with the quiz. You like quizzes, right? They're important.

1.3.4 Ensemble Learning Outputs Question

Okay, so here's a quiz for you Micheal, here's the setup, you ready? You've got N data points. The learner that you're going to use over your subsets is a zeroth order polynomial. The way you're going to combine the output of the learners is by averaging them. So, it's just what we've been talking about so far, and your subsets are going to be constructed in the following way. You uniformly randomly picked them and you ended up with N different subsets or disjoint, and each one has a single point in it, that happens to be one of the data points. Okay i think i get that Right, so if you look over here on your left you got a graph of some data points and this is one subset This is another subset, that's another subset, that's another subset, that's another subset, that's another subset, that's another subset, got it. Yeah, now what do you want to know about it. Now what I want to know is when you do your ensemble learning you learn all these different, you learn all these different rules and then you combine them ,what is the output going to be? What does the ensemble output? And you want a number? I want a description and if the answers a number that's a perfectly fine description. But I'll give you a hint, it's a short description. A short description of the answer. Okay, I'll think about it. Alright.

1.3.5 Ensemble Learning Outputs Solution

Okay Michael have you thought about it? Do you know what the answer is? Yeah. I think, you know, you asked it in a funny way, but I think, what you're asking maybe was pretty simple. So let, let me, let me see if I can talk it through. So ,we've got n data points, each learner is a zeroth order polynomial. So you, you said the ensemble rule is that you learn over a subset, a zeroth order polynomial is just (no period) Well, we said that the thing that minimizes the average. Sorry, that minimizes the expected error, or the squared error [INAUDIBLE] it's just the average. So, if the sets are indistinct sets, with one data point each, then each, of the individual learners, is just going to learn the average. Then they get, not the average sorry. The actual output value of each individual point is the average, and then the combining algorithm, to combine all the pieces of the ensemble into one answer, combines with the mean. So ,it's going to combine the mean of those each of which is the data point, so it's the mean of the data points. So, the ensemble outputs, I don't know, I'd say average or mean? Yes. Or zeroth order polynomial of the data set, or, you know, one node decision tree, or ,uh. A constant? Which happens to be the mean of the data. Haven't we seen this before? It seems to come up a lot, when we are outputting very simple hypotheses. Right. And the last time we did this, if I recall correctly, this is what happens ,if you do an unweighted average with kNN where k is equal to n. Oh, right. Like, like, right. An N-NN. N-NN. Mm. Mm, so we should probably do something a little smarter than this then. And, I thought that we might look at some of the housing data, because, no one's started looking at the housing data yet. [LAUGH] Okay, so let's look at that right quick and see if we can figure out how this works. And then see if we can do something a little bit better, even better than that. Okay?

1.3.6 Ensemble Learning An Example

Alright, Michael, so, here's what you have before you. You have the same housing data that we've looked at a couple of times before. I've, for the sake of readability, I've drawn over, some of the data points so that they're easier to see. This is exactly the data, that we've always had. Okay? Okay. Now, you'll notice that I marked one of them as green, because here's what we're going to do. 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. Okay? 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. Okay? Okay. So that sounds like, cross validation. 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. Okay. Okay, Michael, so the first thing I'm going to do is I'm going to pick a random subset of these points. And just for the sake of the example, 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. And by the way, I'm going to choose those randomly, and I'm going to choose them with replacement. So we're not going to end up in the situation we ended up just a couple of minutes ago where we never go to see the same data point twice. Okay? Yeah. Alright. So 5 subsets of 5 examples, and then I'm going to learn a third order polynomial. And I'm going to take those 3rd order polynomials. I'm just going to learn on that subset, and then I'm going to combine them by averaging. Want to see what we get? Oh, yeah, sure. So here's what you get, Michael. Here I'm showing you a plot over those same points, with the five different 3rd order polynomials. Can you see them? Yeah. They're, right. There's like a bunch of wispy hairs. Just like most third order polynomials. And as you can see they're, they're kind of you stare at them and you see their kind of similar. But some of them they veer off a little bit because they're looking at different data points. One of them actually very hard to see because it's only one like this. Actually veers off like this because just, purely randomly, it never got to see the two final points. I see. But they all, but they all seem to be pretty much in agreement, like between points three and four. There's a lot of consistency there. Right. Because just picking five of the subsets you seem to be able to either get things on the end, or you get things in the middle. And maybe one or two things on the end it sort of works out. Even the one that doesn't see the, the last two points still got to see a bunch of first ones and get this part of this space fairly right. Cool. Okay. So the question now becomes how good is the average of these compared to something we might have learned over the entire data set? And here's what we get when do that. So what you're looking at now Michael, is the red line, is the average of all of those five third order polynomials. And the blue line, is the fourth order polynomial that we learned when we did this with simple regression, a couple of lessons back. Okay. And you actually see them pretty close. Why is one of them a fourth order, and one a third order? Well what I wanted to do is try a simpler set of hypothesis, than we were doing, than when we were doing full blown regression. So third order simpler than fourth order. So, I thought we'd combine a bunch of simpler rules. Then the one we had used before and see how well it does. You want to know how well it does? I would! Well 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. It typically is the case that the blue line always does better on the training set, the red points, than the red line does. But the red line almost always does better on the green point on the test set or the validation set. Interesting. That is kind of interesting. So wait, so let me get this straight. It seems sort of magical. So, so it learns an average of third degree polynomials, third order polynomials, which is itself a third order polynomial. But you're saying it does better by doing this kind of trick than just learning a third order polynomial directly. Yeah, why might you think that might be? I have a guess, you tell me what you think. Wow, so well, I mean, you know, the danger is often over fitting, over fitting is like the scary possibility. And so maybe by, by kind of mixing the data up in this way and focusing on different subsets of it. I don't know. Somehow manages to find the important structure as opposed to getting misled by any of the individual datapoints. Yeah. That's the basic idea. It's kind of the same thing, at least that's what I, I think that's a good answer. It's basically the same kind of argument you make for cross validation. Alright. You take a random bunch of subsets. You don't get trapped by one or two points that happen to be wrong because they happen to be wrong because of noise or whatever. And you sort of average out all of the variances and the differences. Hm. And often times it works. And in practice this particular technique of ensemble learning does quite well in getting rid of overfitting. And what is this called? So, this particular version, where you take a random subset and you combine by the mean, it's called bagging. And I guess the bags are the random subsets? Sure. [LAUGH] That's how I'm going to think of it. That's how I'm going to think of it. It also has another name which is called bootstrap aggregation. So I guess the different subsets are the boots. [LAUGH] No,no, no, no bootstrap usually refers to pulling yourself up by your bootstraps. Yeah, I like my, I like my answer better. So, each of the subsets are the boots and the averaging is the strap. And there you go. So, regardless of whether you call it bootstrap aggregation or you

call it 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.

1.3.7 Ensemble Boosting

Okay so, let's go back and look at our two questions we were trying to answer. And so far we've answer the first one, learn over a subset of data, defined a rule by choosing that subset uniformally randomly and applying some learning algorithm. And we answered the second question, which is how do you combine all of those rules of thumbs by saying, you simply average them. And that gave us, bagging. So Michael, I'm going to suggest an alternative to at least the first one. And 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. So let me throw and idea at you and you tell me if you think it's a good one. So 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. So what do i mean by that? What i mean by that is. We should pick a subset based upon whether the examples in that subset are hard. So what do you think of that? Well, I guess it depends on how we think about hard, right so it could be that it's hard because some, in some absolute sense right, or could be that it is hard relative to you know, if we were to stop now how well we do Yeah, and I mean the latter. Oh. Okay. Alright. Well that I feel like that makes a lot of sense. I mean, certainly when I'm you know, trying to learn a new skill, I'll spend most of my energy on the stuff that I kind, that I'm kind of on the edge of being able to do, not the stuff that I've already mastered. It can be a little dispiriting. But it, but it I think it, I make faster progress that way. Right and if you, if you go back to the example that we, we started with, with spam right? If you come up with a and you see it does a very good on some of the data, some of the mail examples, but doesn't do a good job on the other. Why would you spend your time trying to come up with more rules that do well on the email messages you already know how to classify? You should be focusing on the ones you don't know how to classify. And that's the basic idea here between, the basic idea here behind boosting and finding the hardest examples. Cool. Okay. So that answers the first one. We're going to look at the hardest examples, and I'm going to define for you exactly what that means. I'm going to have to introduce at least one technical definition. But ,uh, I want to make certain you got 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. Okay. We're voting except this time,this time we are going to do weighted mean. Now why do we want to do weighted mean? Well I have to tell you exactly how we are going to weight it but the basic idea is to avoid the certain situations That we came across when we looked at the data before, when taking an average over a bunch of points that are spread out, this 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. But just keep in your head for now that we're going to try to do some sort of weighted average. Some kind of weighted voting. Okay? Sure. One of the things that's scaring me at the moment though is this, like I have this fear that by focusing on the hardest questions, and then, and then sort of mastering those, what's to keep the learner from starting to kind of lose track of the ones it has already mastered? Like how, why does it not thrash back and forth? So that's going to be the trick. Behind the, the particular way that we do weighting. Okay So I will show you that in a moment, and it's going to require two slightly technical definitions, that we have been kind of skirting around, this entire conversation. Okay? Sure.

1.3.8 Ensemble Boosting Quiz Question

Alright so, the, the whole goal of what we're going to add for boosting here is we're going to, we're going to expand on this notion of hardest examples and weighted mean. But before I can do that, I'm going to have to define a couple of terms. Okay. And you let me know Michael if, if these terms make sense. So, here's the first one. The first one is error. So how have we been defining error so far? Usually we take the square difference between the correct labels and the, what's produced by our classifier or regression algorithm. That's true. That is how we've been using error when we're thinking about regression error. How about, a notion of accuracy. About how good we are at, say, classifying examples. So let's, let's stick with classification formulas. Well, that would be the same as squared areas, except that it's not really

meeting the whole powers [INAUDIBLE] area. That is to say, if the outputs are zeroes and ones, the squared area is just whether or not there's a mismatch. So it could just be the total number of wrong answers. Right. So, what we've been doing so far is counting mismatches. I like that word, mismatches. And we might call an error raid or an error percentage as the total number of mismatches over the total number of examples. And that tells us whether we're at 85% or 92%, or, or whatever, right? So that's what we've been doing so far. But implicit in that, Michael, is the idea that every single example is equally as important. So, that's not always the case. Now you might remember from the first talk that we had. We talked about distributions over examples. We said that, you know, learning only happens if you're 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 into account when we've been talking about error. So here's another definition of error and you tell me if you think it makes sense, given what we just said. So, this is my definition of error. So the subscript D, stands for distribution. So we don't know how new examples are being drawn, but however they're being drawn they're being drawn from some distribution, and I'm just going to call that distribution" D", okay? mhm Right. So 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, and C is whatever the true underlying concept is. So I'm going to define error as the probability, given the underlined distribution that I will disagree with the true concept on some particular instance X. Does that make sense for you? Yeah but I'm not seeing why that's different from number of mismatches in the sense that if we count mismatches on a sample drawn from D, which is how we would get our testing set anyway. Then I would think that would be you know if it's large enough a pretty good approximation of this value. So here Michael, let me give you a specific example. I'm going to draw four, 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. Hm. Okay? And then I'm going to tell you this particular learner output a hypothesis. Output you know, a, a potential function that ends up getting the first one and the third one right, but gets the second and the fourth one wrong. So what's the error here? Mm. So let's just make sure that, that everybody's with us. Let's do this as a quiz. Okay, so let's ask the students what they think. So here's the question again. You've output some hypothesis over the four possible values of x, and it turns out that you get the first and the third one right, and you get the second and the fourth one wrong. If I look at it like this, what's the error rate?

1.3.9 Ensemble Boosting Quiz Solution

Okay, Michael what's your answer? It looks like, half of them are right and half of them are wrong. So, the number of mismatches, is, two out of four or a half. Right, that is exactly the right answer ,because ,you got half of them right and half of them wrong. But ,it assumes ,that your likely to see all four of the examples, equally often. So, what if I told you, that, that's not in fact the case. So ,here's another example of error for you. What if I told you that each of the points, is, likely to be seen ,uh, in different proportions and in fact in these particular proportions. So you're going to see the first one ,half the time. You're going to see the second one ,one 20th at a time. You're also going to see the fourth one, one 20th at a time and you'll see the third one ,four tenths of the time. Alright, so you got it Michael? One half, one 20th, four tenths and one twentieth. Got it.

1.3.10 Ensemble Boosting Quiz Two Question

Okay. So, now I have a different question for you. Actually, I have the same question for you, which is, what is the error rate now. Go.

1.3.11 Ensemble Boosting Quiz Two Solution

Okay, Michael what's the answer? Well, it's still a half. But I guess we, we really should take into consideration those probability. So the number of mismatches they have, but the actual number of errors, the expected number of errors is like well, a 20th plus 20th, so like a 10th. So it's 90% correct, 10% error. Right. That's exactly right, so, 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. You buy that? Yeah. Okay, so, that notion of error turns out to be very important for boositng 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 god 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. And 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. And I'll, I'll describe in a minute exactly how that's done. But isn't it the case that this distribution doesn't really matter? You should just get them all right. Sure. But now it's a question of how you're going to get them all right. Which brings me to my second definition I want to make. And that second definition is a weak learner. So there's this idea of a learning algorithm, which is what we mean by a learner here. As being weak. And that definition's actually fairly straightforward so straightforward in fact that you can sort of forget that it's really important. And all a weak learners is, is a learner that no matter what the distribution is over your data, will do better than chance when it tries to learn labels on that data. So what does does better than chance actually mean? Well what it means is, that no matter what the distribution over the data is, you're always going to have an error rate that's less than a half. So that means sort of as a formalism, is written down here below. That for all D, that is to say no matter what the distribution is, your learning algorithm We'll have an expected error. That is the probability that it will disagree with it through actual concept if you draw a single sample that is less than or equal to one half minus Epsilon. Now epsilon is a term that you end up seeing a lot in mathematical proofs and particularly ones involving machine learning. And epsilon just means a really, really small number somewhere between a little bigger than 0 and certainly much smaller than 1. So, here what this means technically is that you're bounded away from 1 1/2. Which 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 gained way back when with decision trees. So does that all make sense Michael? I'm not sure that I get this right. Let's, maybe we can do a quiz and just kind of nail down some of the questions that I've got. Okay, sure. You got an idea for a quiz? Sure.

1.3.12 Weak Learning Question

Okay Michael so you, let's make certain that you really grasp this concept of weak learning okay? Mm-hm. So, here's a little quiz that I put together to test your knowledge. So, here's the, here's the deal. I've got a little matrix here, it's a little table, and across the top. Are three different hypotheses. So, hypothesis one, hypothesis two, and hypothesis three. So your entire hypothesis base consists only of these three hypothesis, hypotheses. Got it? Got it. Okay, your entire instance space consists entirely of only four examples; X1, X2, X3, and X4. Got it? Got it. 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. So, in this case hypothesis one gets examples 2, 3, and 4 correct. But gets example one wrong, while hypothesis three gets one in four correct, but two and three incorrect. I see. So, there's no hypothesis that gets everything right. Right. So does that mean that we don't have a weak learner, because then there's some distributions for which any given hypothesis is going to get things wrong. Maybe. Maybe not. Let's see. Here's what I want you to do. 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 3 hypotheses will in fact be able to find that does better than chance. That is, has an expected error greater than half. Okay. 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 half. So greater than half in this case would mean three out of four, correct? Oh no, no. Oh, you're using, you want to use that definition that you, that actually took into consideration the distribution. Exactly. That's the whole point. If you, if you always need to have some distribution over your examples to really know what your expected error is. Alright. And if there is no such evil distribution, should I just fill in zeroes in all those boxes? Yes, all zeros means no such distribution. You can do it in either case. 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. It had better add up to one. Alright, I think I can give that a shot. Okay, go.

1.3.13 Weak Learning Solution

Okay Michael, you got answers for me? Yeah, I think so. The first thing I notice is that if I put equal weights on all four examples, like I, I decided that instead of solving this problem by thinking, I would just try a couple examples, and see if I found things in both boxes. So, if I put equal weight on X1, X2, X3, X4. Mm-hm. Then hypothesis one, H1 gets three out of four correct, that's three quarters. That's better than a half. So. Well done. Then I fill that in, in the good boxes, quarters all the way down. That's a turtle, 'because it's turtles all the way down [LAUGH]. No, no, it's not though, it should be quarters all the way down. I thought you'd may be draw a quarter. I, I can't draw a quarter, also I can't draw a turtle obviously but still. [LAUGH] Agreed. Alright, good. You'd think, anyone, do you think anyone listening to this is old enough to get turtles all the way down. Yeah, that's a great joke. Everybody knows that joke. And if people don't know the joke, then we should pause this thing right now, and you should go look up turtles all the way down. And then come back. Okay. It's a, it's a really great joke if you're computer scientist. Yes, and if you don't think it's a good joke then you should probably be in a different field. Okay. [LAUGH] What about the evil distribution? So then I started to generate. Okay, well what if, the, the, the issue here is that, because we spread all the, the, the, the probability out in the first hypothesis really good. So I said okay, well let me put all the weight on the first example. The x1. Okay. So what did that look like. Now h1 did very badly. It gets, it's 100 percent error. And h2 is 100 percent error. But h3 is 0 percent errors. So so. yes. So, so putting it all putting all the weight on x1 is no good. And if you look x2, x3, 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 I kind of lucked into putting a half on both the first and the second one. because i figured that, that ought to work, but then i realized, oh wait a second that's an evil distribution, because if you choose h1, h2, or h3, they all have exactly 50% error on the half a half distribution. Very good. So 1/2, 1/2, zero, zero, is a correct answer. Now I don't know if there's others. You know, certainly X, putting all the weight on X2 and X3 is no good, because H2 and H1 both get those. Putting all the weight on X3 and X4 are no good, because H1 gets all of those correct. In fact we have to have some weight on X1, right. Otherwise H1 is the way to go. Right. So, yeah. No, that's interesting. So what does that mean in this case? What do you mean, what does that mean? So what does this tell us about, how do we build a weak learner for this example? So what it tells us is that 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 instant set. Interesting. Now is there a way that we can, like, okay, so this example has no weak learner. Is there a way to change this example so it would have a weak learner? Um...I'm sure there is. Like if we change that x2, x, h3, if that was a check instead of an X. Which one? X2 H3. So if we made that a green one, here I'll, I'll make it a green one. By using the power of computers. Woah, special effect. Yes. So now there's no way to put weight on any two things and have it fail. I don't know, my intuition now is that this, this should have a weak learner. Okay, well, how would we prove that? I don't know, but may be we should end this quiz. Yeah, I think, we should end this quiz. And leave it as an exercise to the listener. I'm pretty sure I can figure this out. By the way, we should point a couple of things here though, Michael. That one is that, the if it weren't the case, if we had more hypotheses and more examples. Perhaps an odd number of them and we have 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 just because you'd have more choices to choose from. What made this one particularly hard is that you only had three hypotheses and none of them was, not all of them were particularly good. Sure, yeah. I mean if you have a bun-, you can have many, many hypotheses and they're all pretty bad then you're not going to do very well. Well, it would depend upon if they're bad on very different things. But you're right, 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. Interesting. Okay, this is more subtle than I thought. So that's, that's interesting. Right. So what the lesson you should take away from this is. If you were just, to think about it for 2 seconds you might think okay weak learner. That seems easy. And often it is, but if you think about for 4 seconds you realize that's actually a pretty strong condition. You're going to have to have a lot of hypotheses that are, many of which are going to have to do good 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.

1.3.14 Boosting In Code

All right Michael, so here's boosting in pseudo code. Okay, let me read it out to you then you can tell me if it makes sense. So we're given a training set. It's made up of a bunch of xi, yr pairs. You know, x is your input and y is your output. And for reasons that'll be clear in a moment All of your labels are either

minus one or plus one. Where minus one means not in class or plus one means you're in a class. So this is a binary classification task. That make sense? So far. Okay. And then what you're going to do is, you're going to loop at every time step, let's call it lower-case t. From the first time step one, to some big time in the future. We'll just call it capital T and not worry about where it comes from right now. The first thing you're going to do is you're going to construct a distribution. And I'll tell you how in a minute, Michael. Okay, so, so, don't worry about it. And we'll just call that D sub T. So, this is your distribution over your examples at some particular time T. Given that distribution, I want you to find a weak classifier. I want your weak learner to output some hypothesis. Let's call that epsilon sub T. The hypothesis that gets produced to that time step. And that hypothesis should have some small error. Let's call that error Epsilon sub T, because it's a small number. Such that it does well on the training set, given the particular distribution. So, I'm just rewriting my notion of error from, the other side of the screen. So there are times we want you to find a weak classifier. That is, we want you to call some weak learner that returns some hypothesis. let's call it h sub t that has a small error. let's call that epsilons of t. Which is to say that the probability of it being wrong that is disagreeing with the training label is small, with respect to the underlying distribution. So just to be clear there, the epsilon could be as big as slightly less than a half. Right? It doesn't have to be teeny, tiny. It could actually be, almost a half. But it can't be bigger than a half. That's right. And, and no matter what happens. Or even equal to a half. but, you know, we can assume, although it doesn't matter for the algorithm that the learner is going to return the best one that it can. With some error. But regardless, it's going to have, it's going to satisfy the requirements of a weak learner, and all I've done is copy this notion of error over to here. Ok? Awesome! Ok. So, you're going to do that and you'll do that a whole bunch of times steps, constantly finding hypothesis at each time step h sub t with small error epsilon sub t constantly making new distributions, and then eventually, you're going to output some final hypothesis. Which, 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 week classifier with low error. Keep doing that you have a bunch of them and then combine them somehow into some final hypothesis. And that's high level of algorithm for boosting, okay? Okay, but you've left out the two, two really important things, even the part from how you find we, weak classifier, which is where do we get this distribution and where do we get this, this final hypothesis? Right, so let me do that for you right now.

1.3.15 The Most Important Parts

Okay Michael, you've called my bluff. You, you said I've left out the most important parts, and you are right. So, I'm going to tell you ,how to construct the most important parts. Let's start, with the distribution. So, let's start with the base case, and that is the distribution, at the beginning of time, D sub one. So, this distribution, is going to be over each of the examples and those examples, are ,indexed, so, over i. And I'm simply going to set that distribution, to be uniform. So, how many examples do we have, Michael? Let's pick, let's pick a letter. Let's call it n. Okay. Why not, cause we do that for everything else and I'm going to say that for every single one of the examples they happen one over int times, that is uniform distribution. Now, why do I do that, because, I have no reason to believe, for the purposes of this algorithm, that any given example, is better ,than any other example, more important than any other example, harder than other example or anything else. I know nothing. So, see if you can learn over all of the examples. You with me? Yeah, cause I feel like if it actually solves that problem, we're just done. So [CROSSTALK] [CROSSTALK] Yes and, and that's what you always want. But that's the easy case. So I start out with uniform distribution, that's what you usually do ,when you don't know anything. But, what are you going to do ,while your in the middle? So, here's what I am going to do Michael. At every time step T, I'm going to construct, a new distribution, Dis of T plus 1. Okay so, here's how we're going to construct the distribution at every time step. Okay? I'm going to create the new distribution, T plus 1 to be E for each example, I. - to be the old distribution, and times T, times E to the minus alpha T, times Y sub i, times H of sub T, of X of I, all divided by Z sub T. [LAUGH] So that's pretty obvious right? [LAUGH] So what do each of those terms mean? I mean ,I know it's intuitively obvious ,even to the most casual observer, but ,let me just try to explain what each of the parts mean. So, we know that the D is our distribution and it's some number, where, over all the examples, it adds up to one. And it's a stand-in, we know, because I said this at the beginning, 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. So, I'm going to take the old distribution for an example, of, 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. So, there's a cute little trick here, we know that h of t always returns, a value ,of either minus one or plus one, because ,that's how we define our training set, you always have a label of minus one or plus one. So, ht is going to return minus one or plus one for a particular x sub i.

Y of i which is the label with respect to that example, is also always going to be plus one or minus one. And alpha t is a constant, which I will get into a little bit later just right now think of it as a number. And so what happens, Michael, if the hypothesis, at time t for a particular example x of i agrees, with the label, that is associated with that x of i? Well, hang on, you say the alpha's a number. Is it a positive number? A between 0 and 1 number? A negative number? What kind of number? Does, does it not matter. I think it matters. That's a good question. It, it matters eventually. But right now, that number is always, positive. Positive, alright. So, like, a [UNKNOWN]. Almost like a learning rate-y kind of thing, maybe. It's a learning rate-y kind of thing, sort of. Alright, so, good. So the y, okay, I see, I see. So, the y times the h is going to be. 1 if they agree, and minus 1 if they disagree. Exactly, so if they both say positive 1, positive 1 times positive 1 is 1. If they both say negative 1, negative 1 times negative 1 is 1. So, it's exactly what you say when they agree, that number is 1. And when they disagree, that number is minus 1. Alpha Sub T, which i define below, is always a positive number. You can trust me on this. The error is always between 0 and 1. And it just turns out that the natural log of 1 minus a number between 0 and 1 over that number, always gives you a positive number. And if you don't believe me, you should play around with the numbers till you convince yourself. So, that's going to be some positive number. So, that means when they agree, that whole product will be positive. Which means, you'll be raising e to minus some number when they disagree that product will be negative which means you'll be raising e to some positive number. So, let's imagine they agree. So you're going to be re raising, e, to minus some number, what's going to happen to the relative weight of d sub t of i?

1.3.16 When D agrees Question

So, Michael wants us to do a quiz. Because Michael likes quizzes cause he thinks you like quizzes. And so, I want you to answer this question before Michael gets a chance to. So just to be clear here's the question again. What happens to the distribution over a particular example i when the hypothesis ht that was output by the example. Agrees with the particular label, Y-sub-i. Okay, so we have four possibilities when they agree. One is the probability of you seeing that particular example increases. That is, you increase the value of D-sub-t on i. Or the probability of you seeing that example decreases. That is, the number d of t of i goes down, or it stays the same when they agree or well it depends on exactly what's going on with the old value of d and alpha and all these other things. So ,you can't really give just one of those other three answers. So those are your possibilites. The other radio buttons [LAUGH] only one of them is right. And go.

1.3.17 When D agrees Solution

Okay Michael what's the answer. Alright, so you kind of were walking us through it, but basically if Yi and Ht agree, that means they're both negative or they're both positive. They're equal to each other. So when we multiply them together we get one. One times whatever our alpha thing is, some positive number is going to be positive. We're negating that, so it's negative E to the negative something is something between zero and one. Less than, less than one. So, that's going to scale it down. So, it looks like. And you know assuming that everything else goes ok with, with the way that ,uh, the normalization happens right? It seems like it could be depends on the normalization. So by the way. That's a good point. The the x sub t. Is in fact, what ever normalization constant you need at time T, in order to make it all work out to be a distribution. Correct. Then not going to, not going to change. True. But is some of them are correct and some of them incorrect, the ones that are correct are going to decrease. And the ones that are incorrect are going to increase. That's right, so what's the answer to the quiz. Depends. That's true, it does. That's exactly the right answer. It depends ,on what else is going on, you're correct. Now. But I feel like it should be decreases, like that's really, mainly what happens. That's also fair. The answer is, if this one is correct, that is they agree, and you disagree on at least some of them, at least one, one other example, it will in fact decrease. So I could ask a similar question, which is, well what happens when they disagree? And at least one other example agrees. Then what happens? Yeah, then they, then that should increase. Oh. Right. Oh. It's going to put more weight on the ones that it's getting wrong. Exactly. And the ones that it's getting wrong must be the ones that are harder. Or at least that's the underlying idea. All right, Michael, so you got it? So you understand what the equation's for? Yeah, it look. It seemed really scary at first but it seems you know marginally less scary now because all that it's doing, it's doing it in a particular way. I don't know why it's doing it in this particular way. But all it seems to be doing is. The answers that it, it had, it was getting wrong... It puts more weight on those and the ones that its getting right, it puts less weight on those and then you know, the loop goes around again and it tries to make a new classifier. Right, and since the ones that its getting wrong are getting more and more weight but we are guaranteed or atleast we've

assumed that we have a weak learner that will always do better than chance. On ,ah, 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.

1.3.18 Final Hypothesis

So that ties together this, what constructed E does for you, and connecting it to the hardest examples. So now, 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, they way you do that combination in the step is basically by doing a weighted average. And the weight Is going to be based upon this alpha sub T. So the final hypothesis is just the s g n 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 alpha sub T's. And remember, the alpha sub T is one half of the natural log of one minus epsilon T over epsilon T. That is to say, it's a measure of how well you're doing with respect to underlining error. So, you get more weight if you do well Then if you do less well or you get less weight. So what does this look like to you? Well its 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 thresh holding function where if its below zero you say you know what? Negative and if its above zero you say you know what? Positive and if its zero you just throw up your hands and And return zero. In other words, you return literally the sign of the number. So you are throwing away information there, and I'm not going to tell you what it is now, but when we go to the next lesson it;s going to turn out that that little bit of information you throw away is actually pretty important. But that's just a little bit of a teaser. We'll get back to that there. Okay so, this is boosting, Michael. 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, which I show you how to do over here, and then simply bringing everything together, which I show you how to do over here. Alright yeah, I think it doesn't seem so bad and I feel like I could code this up, but I would be a little happier if I had a handle on what the, why alpha is the way that it is. Well there's two answers. The first answer is. You use natural logs because you're using exponentials and that's always a cute thing to do. And of course, you're using the error term as a way of measuring how good the hypothesis is. And the second answer is, it's in the reading you were supposed to have done. [LAUGH] So, go back and read the paper now that you've listened to this and you will have a much better understanding of what it's trying to tell you. Thanks You're welcome. I'm about helping others Michael you know that.

1.3.19 Three Little Boxes

So, Michael, I want to try to convince you other than the fact that it's an algorithm with symbols that, it sort of works, at, at least informally. And then, I'm going to do what I always do and refer you to actually read the, the, the text to get the, the details. But before I do that, I wanted to go through an example if you think that would help. I would like an example. Okay. So, let's go through an example. So, here's a very simple example. So, I got three little boxes on the screen. Can you see them? Yeah. 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. Hey those boxes are really nice, did you get help from our trusty course developer? I did in fact did get help from our trusty course developer. And when I say help, I mean he did this. Oh thanks Push Car. Yes Push Car is wonderful. Now what's really cool about this is that Push Car is already let you know that we're going to be able to do this in 3 simple steps. And I'm going to be able to animate it. Or at least hopefully it'll look animated by the time, [LAUGH] we're done with all the editing. So just pay attention to the first box for now, you have a bunch of data points; red pluses and green minuses, which is the opposite of what we usually do Push Car. But either way it's red pluses and green minuses. [LAUGH] With the appropriate labels and they all live in this, this part of the plane. By the way, what do you call a part of the plane? I know you have line segments, what's like, a sub part of a plane? Looks like a square to me. Yes it is, but I mean, what do you call them? You, you don't call it a plane segment, do you? What do you call it? A region. A square region, fine. So it's a square region on a plane. And we want to figure out how to be able to correctly classify these examples. Okay, so that is nothing new there. We just want to be able to come up with something. So now we have to do what we did like in the quiz is that we have to specify what our hypothesis space is. So here's our hypothesis space. So the hypothesis space is the set Of axis aligned semi-planes. You know what that means? Mm, no. Well for the purpose of this example it 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. I see. Okay, good. Right. And their

axes align 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. Okay, so I'm going to just walk through what boosting would end up doing with this particular example or what a boosting might do with this particular example given that you have a learner. That always chooses between axis aligned semi planes. Okay? Yeah. So let's imagine we ran our boosting algorithm now in the beginning it's step 1 all of the examples look the same because we have no particular reason to say any are more important than the other, 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 hypotheses 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. That is what I was guessing. Of course it was. And what I'm saying here is that everythign to the left of this line is going to be positive and everything to the right is going to be negative. So if you look at this what does this hypothesis do? So it gets correct, correctly labeled positive. The two pluses to the left. Right? Correct. And it gets correct all of the minuses as well. Correct. Right? But it gets wrong the three pluses on the right side. So it gets, this wrong, this wrong, and this wrong. Right, the Three Plusketeers. Exactly. [LAUGH] The Three Plusketeers. That's actually pretty good. So I'm just you know 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 our little alpha you end up, our little, our little formula for alpha, you end up with alpha equal to 4.2. That's not obvious to me but. Is is See, see, see it's not always obvious. [LAUGH] Okay. Good. So there you go and that's just what happens when you stick this particular set in there. So now we're going to construct the next distribution. Right? And what's going to happen in the next distribution? So the one's that it got right should get less weight and the one's that it got wrong should get more weight so those three plusketeers should become more prominent somehow. That's exactly what happens. They become, I'm just going to draw them as much thicker and bigger to kind of emphasise that they're getting bigger, and it's going to turn out that everything else is going to get smaller which is a lot harder to draw here. So i'm just going to kind of leave them their size, so they sort of get normalized away. Okay? I would guess as to what the next plane should be. I think that we should cut it. Underneath those pluses but above the green minuses. And that should get us three errors. The two pluses on the left and the minus on the top will be wrong. But they have less weight than the three pluses we got right, so this going to be better than the previous one. So, that's possibly true. But it's not what the learner output. Oh! Let me tell you what the learner did output though. This learner output by putting a line to the right of the three pluses, because he's gotta get those right in saying that everything to the left is in fact, positive. So, does that seem like a reasonable one to you? Well, it does better than half. I guess that's really all what we're trying to do, but it does seem to do worse than what I suggested. Well, let's see, it gets the three that mattered that you were really, really doing poorly right but then so did yours. And it only, and it picks up still the other two which it was getting right. And it gets wrong these three minus' which aren't worth so much. So is that worse than what you suggested? No, it gets wrong, oh, the three minuses. Oh, it gets correct those two red pluses on the left. So it gets three things wrong. So that's just as good as what I suggested. Okay, I agree. Okay good. So the error of this step by the way, turns out to be 0.21 and the alpha at this time step turns out to be 0.65. So that's pretty interesting, so we got a bunch of these right and a bunch of these wrong. So what's going to happen to the distribution over these examples. Alright, the ones that it, again, the ones that it got wrong should get pushed up in weight and which ones are those, those are the, the three green minuses in the middle patch Right. They should become more prominent. The pluses, the three, the three plusketeers should become less prominent than they were but it still might be more prominent than they were in the beginning. And maybe because in fact the alpha, let's see the alpha is bigger so, it will have actually a bigger effect on bringing it down. Yeah I guess so, but it, there'll still be more prominent than the other ones that haven't been bumped. Yeah the ones that you, the, the two, the two red pluses on the left have, you've never gotten them wrong. Hm. So they're really going to disappear. So, if we do, If I do my best to, If I do my best to kind of draw that you're still, you're going to have. These pluses are going to be a little bit bigger than the other pluses, but they're going to be smaller than they were before. The two, the three greens in the middle are going to be bigger than they were before. But those two pluses are going to be even smaller, and these two minuses are going to be smaller. So, what do you think the third hypothesis should be. Quiz. Oh, I like that.

1.3.20 Which Hypothesis Question

Okay, so Michael wanted to have a quiz here, 'because Michael again, likes those sort of things and, and I like to please Michael. So, we came up with three possibilities, one of which we hope is right. [LAUGH] And I've labeled them here in orange, A, B, and C and put little radio boxes next to 'em, so you could select

'em. So which of those three hypotheses are, is a good one to pick next? So, A is a horizontal line that says everything above it should be a plus. B is a, another horizontal line that says everything above it should be a plus. And C is a vertical line, like the last two hypotheses that we found, that says everything to the left should be a plus. So, which do you think is the right one? Go.

1.3.21 Which Hypothesis Solution

Alright Michael, what's the answer? Alright so of those others, well C is pretty good, because it does separate the pluses from the minuses. We, we even liked it so much we used it in round two. Mh-hm. But it doesn't as good to me as A, because A actually does a good job of separating the very, the more heavily weighted points. So I would, I would say A. So in fact that is what our little learning system shows. It shows A. Now, through the trick of animation, I leave you with A. And that is exactly the right answer. By the way, Michael, if you look at these three hypothesis and their weights, you end up with something kind of interesting. So if you look at this third hypothesis that's chosen here, turns out they have a very low error, you'll notice that the errors are going down over time, by the way, of 0.14. And it has a much higher alpha 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. Okay Michael, so I cleaned up a little bit so that you could see it. If you take each of the three hypothesis that we produced, and you weight them accordingly, you end up with the bottom figure. No way. Absolutely. That's. Kind of awesome. So what you're saying is that, even though we were only using half planes, or, or axis-aligned semi planes, for all the weak learners, that at the end of the day it actually kind of bent the line around and captured the positive and negative examples perfectly. Right. Does that remind you of anything else we've talked about in the past? Sh. Everything. Nothing. No, I dunno, I mean so with, with decision trees you can make the shapes like that, and That's true. And the fact that we're doing a weighted combination of things reminds me of the neural net. Yeah. And it should remind you of one other thing. I'm imagining that you want me to say nearest neighbors, but I can't quite make the connection. Well, you recall in our discussion with nearest neighbors, when we did weighted nearest neighbor. In particular we did weighted linear regression, we were able to take a simple hypothesis, add it together in order to get a more complicated hypothesis. That's true, because it's local. Right, exactly because it's local, and this is a general feature of Ensemble methods that if you try to look at just some particular hypothesis class. Let's just call it H, because you're doing weighted averages over hypotheses drawn from that hypothesis class. This hypothesis class is almost all low, 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. I'm not surprised that you can combine simple things to get complicated things. But I am surprised that you can combine them just with sums. And get complicated things because sums often act very, you know, sort of, friendly. Right it's a linear combination not a nonlinear combination. Actually, Michael part of the reason you get something nonlinear here is because you're passing it through a non-linearity at the end. The sine. Yea, that's a good thing, we should, we should ponder that.

1.3.22 Good Answers

Okay Michael, so we've done our little example. I want to ask you a quick question and try to talk something through with you and then we can start to wrap up. Okay. Awesome. Alright, so, here is my quick question. Now, in the reading, which I know you've read, there's a proof. That shows that boosting not only, you know, does pretty things with axis of line semi-planes, but also that it will converge to good answers and that it will find good combined hypotheses. You know, we could go look at the reading and write down a proof that shows that boosting does well. Umm. And there's one in the reading. Or we could talk about an intuition. So if someone were to come up to you. If a student were to find you somewhere and said, I read the proof, I'm kind of getting it, but do you have a good sort of intuition about why boosting tends will do well? What do you think you would tell them? Could you think of something simple? I've been struggling with this for a while. No. [LAUGH]. Okay, well, then let me try something on you and you can tell me if it sort of makes sense. So this is just an intuition for why, for why boosting pins will do well. Okay, so what does boosting do? Okay. 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 become increasingly important. Right, that's what boosting does. Yes? Yes. Right, that's what this whole, whole bit of D is all about. It's all about re-weighting based on difficulty and hardest. And 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, right? Mm hm. That's what it would mean not to do well, agreed? Yeah. Okay. So how many things could it not get right? How many things could it misclassify? How many things could it get incorrect? Well, I'm going to argue Michael, that, that number has to be small. There cannot be a lot of examples that it gets wrong. So do you want to know why? Do you want to know my reasoning for why? Yeah. 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, more than half of the time I am correct, that number of things I'm getting wrong has to be getting smaller over time. Because let's imagine that 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. Is that make sense? It does, but it, but what scares me is, okay, 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. Yeah, why is that? I don't know. But did you wanna, are we, we working up to some kind of you know, log n kind of thing where each time you are knocking off half of them and therefore. I don't know. Do you remember the proof. The proof. I mean what goes on is that you get, sort of, this exponentially aggressive weighting over examples, right? Yeah. 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 get that we're, the we're quickly ramping up the weights on the hard ones. I don't get why that's causing us to get fewer things wrong over time. So like, when you should, in your, 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. Right. Like, 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? Well, certainly it's the case that if you keep getting something, right, you will, get them. Well, so here's what happens over time, right. Is that over time, every new hypothesis, it gets to get a vote, based upon how well it does on the last, difficult let's say, distribution. So the ones that are even if the ones that you were getting right you start to get wrong, they are going to, if you get them increasingly wrong, that error's going to go down and you're going to get less of a vote, right. Because e sub T is over the current distribution. And it's not over the sum of the voted, over all the examples you've ever seen. Understand. So does that make sense? Is that right? I don't know. I don't have the intuition, it seems like it could be, you know, could we keep shifting the distribution. It could be that the error is going up. Like if the error could be low, why can't we just make it low from the beginning. Right. Like, I feel like the error should be going up, because we're, we're asking it harder and harder questions as we go. No, no, no, because we're asking it harder and harder questions, but 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. I mean that's why having, being able to always, that's why having a weak learner is such a powerful thing. But why couldn't we like on, on iteration 17, have something where the weak learner works right at the edge of it's abilities, and it just comes back with something that's a half minus epsilon. That's fine. But it has to always be able to do that. If it's a half minus epsilon, the things it's getting wrong will have to go back down again. No, no I understand that. What I'm saying is that, 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, imagine, imagine the case that you're getting, right. You, 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 round 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 vote on that do well on different parts of the space. Right? Because you're always forced to do better than chance. So. Yeah but that, that's not the same as saying that we're forced to get better and better each iteration. That's right, it's not. So it's, yeah again, 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. Well, I feel like it should be, it should be something like, look we had, look at what the, so okay, so we generate a new distribution, what is the previous, what's the previous classified error on this distribution, it better be the case. I mean if it were the case that we always return the best classifier that 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 a half 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're still going to, well you won't have a third error actually. 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 an example I just gave. Less than a third. So, if your error is less and a third, then the weighting goes up more. And so, the one that you just got wrong goes up, 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 a half. 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, coicidentally, got right two times in a row. Become so unimportant. 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 a half. 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, you're basically saying you must pick up information all the time. Hm. And then your non uni. Well uniform is the wrong word but you are kind of. You know, 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 you. You, 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, 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 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. Cool.

1.3.23 Summary

Okay, Michael, so that was a great conversation, what have we learned? Alright, well we talked about ensemble learning which was the idea of instead of just learning one thing, if it's good to learn once, it's even better to learn multiple times, In multiple ways. The simple version that we concentrated on first was this notion of bagging. Where what we did is instead of just learning on the whole data set, we would sub-sample bunch of examples from the training set, different ways, and train up different classifiers or different learners on each of those and then merge them together with the average. Okay, so if I can summarize that, we learned that ensembles are good. [LAUGH] We learned that even simple ensembles like bagging are good. We talked about the fact that by using this kind of ensemble approach, you can take simple learners or simple classifiers and merge them together and get more complicated classifiers. Mm, yeah, so we can take. We can. Combining simple gives you complex. Anything else? And we talked about the idea of boosting where you can Oh, maybe this is why it's called boosting. You can take something that has possibly very high error but always less than a half, and turn it into something that has very low error. So we learned that boosting is really good. And, we talked a little bit about why, that's good. By the way, there's a whole bunch of other details here too, right? Boosting also has the advan, as does bagging Not only has these little properties you've talked about before, but it tends to be very fast. It's agnostic to the learner. As you noticed, that in no time, did we say, try to take advantage of what the actual learner was doing. Just that it was, in fact, a weak learner. Hm. So I think that's important. It's agnostic. Meaning you can plug in any learner you want? Yeah. So long as it's a weak learner. So there's something we learned about. We learned about weak learners that we defined with that meant. And, we also talked about ,um,

what error really, really means. With respect to some kind of underlying distribution. What do you think Michael? That seems like useful stuff. These are useful stuff to me. I'm going to throw one more thing at you, Michael, before I let you go. Okay, you ready? Yep. Here's a simple fact. About boosting that turns out in practice. You know our favorite little over-fitting example. Do you know how over-fitting works? You have a training line that tends to get better, and better, and better. Maybe even going down to zero error. But then you have test error Which gets better and better and at some point it starts to get worse. Mm. And at that point you have over fitting and I think, Michael, you asserted it at some point or maybe I asserted that ,you always have to worry about over fitting. Over fitting is just the kind of fact of life. You got to come up with ways to deal with it or sort of over believing your data. Well, what if I told you that in practice When you run boosting, even as you run it over time so that your training error keeps getting better and better and better and better, it also turns out that your testing error keeps getting better and better and better and better and better and better and better. That seems too good to be true. It does seem too good to be true. It turns out it's not too good to be true. And I have an explanation for it. Tell me. Not until next time. alright, see you then. See you then. Bye.

1.4 Kernel Methods and SVMs

1.4.1 The Best Line Question

Hi Michael, how are you doing? Oh, good thanks. What the, what's on tap for today? Well, as you can see on your screen today we're going to talk about support vector machines. Hmm. And at the end of it, we are going to circle back to boosting, so that I can try to explain to you why it doesn't seem to over fit as much as you might think or at least not in the ways that you make it. Ooh, that would be cool. I was worried that that was maybe a command. What was a command? The thing that you wrote on the screen. [INAUDIBLE] [LAUGH] Man you know that reminds me of one of my favorite little thing about police. That any combination of the word police is a legal sentence [LAUGH] I see. Please. Please. Police Please please. Please police. Please please please please please please police. Those are all legal sentences. All right, we'll please stop. [LAUGH] Man, this is why I hang out with you, Michael. Ok,so today we're going to talk about support vector machines and I'm going to do something unexpected. I'm going to start out The beginning of this with a quiz. With a quiz. Yes, with a quiz. So here's the quiz for you, Michael. I've drawn on here ,uh, some points, some labeled positive and some labeled minus, representing two different classes. And, you'll notice that you could draw a line to separate them, therefore they are Linearly separable. Exactly. So the question I have for you is simply this, which is the best line? And here's how you're going to show me. Here's how you're going to show me. I'm not just going to ask you to draw some random line because that's too hard for us to get feeback on. So instead what I've done is I've drawn three green dots. Here on the sort of upper left and three red dots here on the bottom right and what I want to you to do is select one of the green dots and one of the red dots and since you'll have two dots that will define a line and that'll be the way you indicate to me which of the lines is in fact The best line. The best line. The best line. And I'm not going to even tell you what best means. You tell me what best means, when you justify why you would choose one over the other. If I think they're all the best, I can choose any one I want? That's right. Ok. So you got it? I think so. OK. Go.

1.4.2 The Best Line Solution

Alright, Michael. So, you got it? Which line do you think is best? Alright, so if I choose one green and one red [CROSSTALK] that means there's nine different possible lines. But they all separate the points. That is correct. I'm pretty sure. I, you know, I think they're intended to even if they don't quite. But I think they do, I think they actually all separate the positives from the negatives [CROSSTALK]. The positives on the upper right side and the negatives on the lower left side. So, I mean the only one that seems further special is the middle one. If I choose the middle green and the middle red it's kind of, you know, ecstatically pleasing, there's a lot of space on, on each side. Okay. So, I'm going to go with that but it's not clear, you know again, best if I was a photographer that might be the best. Hm, you are a photographer. So, let's pretend I can draw a straight lines. That's pretty good. That's the line that you've chosen. Yes. And you think that's best. Well, I'm going to give you a hint, Michael, and tell you that you were correct. That's a pretty good hint. Yeah. But now I want you to figure out for me why that line is better than this line. Interesting. Alright, well so one thing that's, that second line. The. Orange line, as oppose to the orange line, has against it, in some sense it seems to get really, really close to that bottom minus point. Maybe it's a little too close. Like maybe that minus is near other minuses that we just can't see. And since we drew

really really close, a line really really close to it, it could be we ended chopping of some of the minuses that. That we might want to have kept on the other side. And so maybe by drawing it in the middle it's sort of, we have the biggest, I don't know like demilitarized zone. No, I like that. So that is a very good explanation for why you would prefer the middle line over the. Other line that isn't the middle line. And you can make a similar argument for any other of the nine lines that, that or the other eight lines that you could possibly draw. Let me draw another line for you and you tell me if you think this one is good or bad and why it might be better or worse than the middle one. See if you give me a different answer. I can take one of the parallel lines which again we will pretend is a straight line and put it there. Or I can take in fact the other line that is meant to be parallel to it and why aren't those lines better? Or just as good. I guess I'd use the same explanation as before which is that these, the, the line that was, is really close to the pluses maybe is a little too close for comfort and it, it gets very close to that plus. The, kind of, I don't know. I could point to it, but maybe you should point to it. The plus that's really close to the line. I'm pointing to it. Because Again if there's a just, you know, other data points near there, it's going to make this distinction between the positives and the negatives that isn't really warranted by the data. Right, so I could make the op the opposite argument with you, Michael, which is that listen, they all separate the points. They completely explain the data as we see them. So, all three of those lines explain the data. In fact, all nine of those lines explain the data. So, why aren't they warranted by the data. What's the problem that you might run into if you put one line very close to the positives or one line very close to the minuses? Well, again...so, so...it may be that they fit this data. But, this is just a sample from the population. And so... You know, we don't know what's going to happen really close to that other plus is. So like in the nearest neighbor algorithm for example which you always try to make me remember, it's going to want to, you know, if it's closer to the pluses, it should be a plus. Right. That makes sense and I think that's exactly right so your intuition and I think the intuition that people should take with them is that lines that are too close to the positives. Or lines that are too close to the negatives, sort of have this feature where 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. 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. And why is that? It's because they're believing the data too much. So, given that we're trying to avoid overfitting, what could you say about the middle line? So, here's the argument that I would make, and you tell me if you buy it, Michael. This line, or the line it's intended to represent anyway, is the line that is consistent with the data while commiting least to it. That seems like a good way of saying what I was feeling, yeah. It is funny though because like, overfitting up to this point, we you, we were generally talking about overfitting as being something where there's a great deal of model complexity in some sense. Mm-hm. 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 interesting that they, they, they kind of maybe behave as if they are. Right, and in fact they, they're, it's a more, sort of, literal interpretation of the words over and fit, right? You, 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 right. So this basic idea of, uh,uh, finding the line of least commitment in the linear separable set of data, is the basis behind support vector machines. So what I want to do next is I want to see if we can come up with some equation that would help us define such a line. It's easy in this case cause we're staring at it but if you imagine these points were in 700 and. Thirteen thousand dimension it would pretty difficult to find such a plane just by staring at it so let's see if we can try to work out how you go about finding this least commitment line okay Cool.

1.4.3 Support Vector Machine

Okay Michael. So, let's, let's write down a few equations. Let's try to be a little bit more formal. A little bit more mathematical about this idea. So, I'm going to try to encapsulate, what we just talked about, to the line of least commitment. By drawing another line. So, 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, to 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, and giving mis-classification. And then the middle line, is sort of in the happy medium. Okay? So, what you really want from the lines, all the lines that are possible, between ,these two boundary lines. If I can use that language. Is that, somehow, this distance here, is as big as possible. Can you see that? Yeah, though it seems like the gray line, could be pushed out a little more, right? because ,the minuses don't bump into it. That's a good point Michael and I'm going to fix that, by

putting a minus sign here. [LAUGH] Okay [CROSSTALK] I see, this is [CROSSTALK] I did the best I could under the circumstances. Data, revisionist history. No, there's just an invisible point, and I just made it more visible. For the sake of the reader. Okay, so, I've got these two lines, these are sort of as far as I can go, without stating to do mis-classification with my, my separating line, and the line in the middle, we've already argued is the sort of the best one because it provides the least commitment. So, that means, you want to have a line, such that, it leaves as much space as possible, from the boundaries. Alright, Michael. So let's see if we can figure out exactly, what that line is like. So, the first thing, that I want to do is, is introduce a little bit of notation. Right? So we all remember what the equation, of a line is. It's Y equals MX plus B, that's just a general equation for a line. But, 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. And generally ,when we write about hyperplanes, we describe them as some output, let's just call it y is equal to w transpose plus b. here, because of what we're, 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 are the parameters ,or W, represents the parameters for our plane. Along with b, which is what moves it out of the origin. Okay, are you with me. I think so, but that's, so may be we should get rid of that top ,Y, because, that Y is different kind of Y. Alright so the top Y is talking about the Y dimension of the plane and in the second equation, that Y is kind of folded into the X. And we have a new y, which is actually, the output of the classifier. Right. I like it, so let's get rid of that first y which is just an equation for a line and let's ask what each of these things are. So, let's just say that again for clarity's sake. I think, you make a good point, Michael. Y here, is going to be our classification label ,right, whenever we're talking about using a linear separator, effectively, what we've been talking about, which I realize now we never ever actually said explicitly, 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. And in this case, in particular, 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. Okay? Yeap. This is our classification label y. W represents again, the parameters of the plane. Along with b, which is what moves it in and out of the origin. So, this is now, effectively, what our linear classifiers actually look like. Even in multiple dimensions, with hyperpoints. Okay? Cool. So, let's take that, and and, and push it, to, to sort of the next level. Let, let's figure out exactly, what we would expect the output, of our hyperplane ,to be in this example, that I, 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, on the one hand, but also, that it has the property of being, as far away, from the data as possible, while still being consistent with it. So, if you're on the decision boundary, for this particular line, which again, is w transpose x plus b. What would be, the output ,of this classifier for any point that lies along the line? so, right. If that's decision boundary, that's where it's kind of not sure ,if it's positive or negative so that should be zero. Right, so the equation of this line or this hyperplane, is w transpose x plus b equals zero. For, some set of parameters w and b, we don't yet know what they are. But, what we do know, that this is the definition of a hyperplane. Where the equation for a hyperplane, and since it's at the decision boundary ,it should give me neither a positive or a negative output. Okay? Yep. Okay now, one question we can ask ourselves then, if we look at these other lines is well what's the equation for the other grid lines? That are right at our positive or negative examples. So to help you answer, that, I want to talk about what the labels themselves ought to be. So ,just like we did with boosting, let's say that our labels are always going to be, from the set minus one and plus one. We know ,that our labels ,are minus 1 and plus 1. And so we're going to take advantage of that fact, by saying well. The line that brushes up, against ,the positive example. We want it to be the case ,since, we know those labels are going to be plus 1. That, the output ,of that particular line, that particular linear separator, would be plus 1, on the very first point, that it encounters. Does that make sense? Yeah Okay That way the things, that are, that it's you know that are kind of past the line are going to be above 1 and the things are before the line, in that kind of demilitarized zone, are going to be between zero and 1. Right, so in fact given what you just said, what is the equation of that line? Oh I see. So, it should, be the W transpose x plus b equals, 1 for the top gray line. That's exactly right. And by a similar argument, where would you say the, the line of the hyperplane should be for the bottom gray line? Analogously, it seems like that one should be minus one. Right. So, we, we have the decision boundary, we're looking at, and we know that the equation of the line is w transposed x plus b is zero. We know that if we slid that line, towards the positive values, we would end up with w transposed x plus b equals one. And if we slid it towards the negative values, we'd end up with, equals minus one, now we can ask ourselves, how this helps us. And it 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 our positive and negative examples, so that would mean then, and I hope you buy this Michael, that the distance ,between the two gray lines, which are parallel, to that line, needs to also be maximum. Yeah,

that's exactly what we want. Right, so we want this vector here, to have, the maximum link 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. Okay?And I don't know what those x values are, but I do, I'm just going to call, them x1 and x2. That seem fair? Hm, I guess that seems as good a name as any. And that is going to define the two points that I have, and the distance between them ,is in fact going to be or the, 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 your boundary decision line is from the data and we ant that to be maximal ,because then we made the least commitment to the data. So, let's write that down as algebra. The equation for our positive line so to speak ,is ,therefore w transpose x1 plus b equals 1. 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 gives me some point on that line, okay? It gives me, puts me in some particular place, on that line. And similarly, I can do the same thing, for my negative line. And get w transpose x2, plus b equals minus 1. Now, we want the distance between, these two, hyperplanes or these two lines, in this example, to be maximal. And in order to figure out what, that means, we need to know exactly, what that line is. So, it's, it's the difference between the two. So, we can just use our, 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. Oh, I like that, we get a quiz. That is the correct answer. Even though it seems like a tightness match, in fact quiz, is the correct answer [LAUGH]

1.4.4 Distance Between Planes Question

Okay. So here's the quiz. Michael is going to answer it, but we want to give you a chance to answer it first. I've got these two equations of two different hydroplanes, 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. Do you understand that, Michael? Or, have I, have I told them enough, or not? Yeah, I think I'm just going to subtract the second equation from the first equation. It seems pretty straightforward. Okay, it seems reasonable to me. But remember, the output that I want you to figure out here is exactly what the distances between those two planes, okay? That is, between what's represented by X1 and X2, okay? Go.

1.4.5 Distance Between Planes Solution

Okay Michael, what's the answer? Well, there's the answer to the question that I thought you were asking and then there's the question that you then at the end actually asked. It seems like at the end you asked what is the difference between the two, the distance between the two lines and I I feel that, like that's just The, like the norm of x1 minus x2. But, the difference between these equations is going to be well ,uh, W transpose. Well why not write down what you're, what you're telling over in the side over here and then we can put the final answer in the box. Okay. Okay, so what now? W transpose. X1 minus x2, equals two. Right, so you used ,ah, the power of subtraction to make that work. Okay, very good. Okay, so that's the difference between those two equations, now how am I going to go from there to figuring out the distance between x1 and x2? I still feel like its just that norm of x1 minus x2. Okay, but I want you to tell it to me in terms of w. And what I want you to tell it to me in terms of w, because the only thing we have to play with here is w, well w and b. That's what defines our line. And so I want to find the right line, the only thing I get to play with, W and b. So, I'd like to know something about the relationship between w and the distance between x1 and x2. Well, w transpose times their differences too. [LAUGH] That's true. That's not telling us the distance, though. So what is the distance in terms of W? Well what if I told you w was a number? What would you do if it was just a simple scalar and you had this equation, and I wanted to know what x1 minus x2 was. What would you do? And I wanted it in terms of W? Yeah, I wanted to know what x1 minus x2 was equal to. Oh, I see. So, if I divide it by w, that would be helpful' cus then x1 minus x2 would be two over w. Right, but you can't do that because w is a vector, and you can't really divide by a vector, at least not in the world that we're talking about. So how are you going to... Make that work. Would you like a hint? Sure. Well here's a hint ,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. Now what is dividing W by the length of W, give you? So, right. 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's sphere. Right. No, that's exactly right! Alright, in fact it is a sphere because it's a, it's a hyper sphere I suppose. right! So we do that and that effectively is like giving you a value one, like you said it's a unit sphere. And so now we're actually talking about the difference between the vector X one and the vector X two projected onto the unit sphere and that's equal to two over the [CROSSTALK]. Wait. That's what X minus two is perjected onto the W vector The normalized W vector. Right, the doubilized norm, the normalized The double normalized U vector, right [LAUGH] So does that help you? Does that actually answer the question? Doesn't seem like it does. no, it does. So is that, okay, alright, hang on [LAUGH] ,so ,the x one minus x two, dotted with W. So W, W, we don't know. It could be anything. Can it? So. Mm-hm. We've taken x one minus x two, projected it onto W. So it's like the, the length of x one minus x two, but in the w direction. Exactly. That's exactly right. So, the link, which, by the way, is, well, is exactly right. So, what we've just done is we have found the link of x1 and x2 in the W direction. What do we know about w with relationship to the line? W is the parameter to the line. Yes, but in particular. As with, in the case with any hyper plane, W actually represents a vector that's perpendicular to the line. And since we chose X1 and X2, so that they would in fact, their distance or the difference, would in fact be perpendicular to the line. What we've just done is projected. That rose 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. So we maximize the length. Of x1 minus x2 with doing what with W. I see so the thing on the left is,infact, have we answered the quiz yet by the way, or are we still working on that? I'm going to say we are still working on it. Oh men ,alright this is a hard quiz. The thing on left, not just were the braces are, that actually turns out to be the distance between the two. Hyperplanes. Right, let's let's give that a letter. Let's call it M. Mm. Mm. And, we're saying that equals two over the norm of W. And that's, 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. Right. So it set Ws to all zeroes, and we should be golden. Right, 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 outta be able to use it for something and turn in into some other problem we might be able to solve. Might be able to maximize for ,uh, so that we can actually find the best line. And it turns out we can do that. Have we answered the quiz yet? Oh yeah we did. Which is in fact what I wanted so the answer is. Wow. Somebody gets that, that would be pretty impressive. That would be very impressive. Or anything similar to this I would accept. [LAUGH] In fact I probably better will. Okay good. 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 two over the length of w and turn it into a problem where we can solve this directly.

1.4.6 Still Support Vector Machines

Okay. So, we're still talking about support vector machines, although I haven't told you what support vector machines are yet, we're getting there, Michael. Bear with me. And what we got from our last discussion is that what we want to do somehow is maximize a particular equation, that is, 2 over the length of W. And as a reminder, W 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 2 over the length of W while classifying everything correctly. But, while classifying everything correctly is not a very mathematically satisfying expression, but 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, and I'm just going to write, I'm going to write it out for you, Michael, and see if you can, you can guess why this works. So, what I've written here is YI times W transpose XI plus B greater than or equal to 1 for all I. That is, for all of our training data examples. So why

does this work? Well, what we really want is that the, that linear classifier, WTXI plus B, is greater than or equal to 1 for the positive examples, and less than or equal to negative one for the negative examples. But you cleverly multiply it by the label on the lefthand side, which does exactly that. If YI is 1, it leaves it untouched. And if YI is negative 1, it flips everything around. So that we're really talking about less than or equal to minus 1. That's, that's very clever. It is very clever, and I'm going to pretend that I came up with that idea myself. So, it turns out that trying to solve this particular problem, maximizing 2 over W, while satisfying that constraint, is a little painful to do. But that 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 2 over W, we can instead try to minimize 1/2 times W squared. Now can you see that those will always have the same answer? Yes, so, well, not the same answer, but it will be minimum, the point that maximizes one will minimize the other. Yeah, because the, 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 exactly, you know, changes the direction, of what the answer is. And the squaring is, is, makes it monotone. It doesn't, it doesn't, it magnifies it but it doesn't change the ordering of things. So yeah. That, that, that seems fine. I don't why that's any easier, but it seems the same. Well, do you want to know why it's easier? Cause I'll tell you. Please. 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. Awesome. 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 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 the MATLAB. But there's a whole set of classes out there, where they teach you how to do quadratic programming. We could take an aside, I could learn all about quadratic programming, and then we could talk about it for two hours. But it's really beside the point. 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. Okay? Okay, fair enough. Okay, fair enough. So, in particular, it turns out that we can transform, again, this particular quadratic programming problem into a different quadratic programming problem. Or actually, truthfully, into the normal form for a quadratic programming problem, that has the following form. So here's what this equation tells you, Michael. We have basically started out by trying to maximize the margin. And that's the same thing as trying to maximize 2 over the length of W, I think I convinced you of, 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 1/2 times the length of W squared, subject to the same constraints. And then notice, because we happen to know this, 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 1/2 of W squared, we can try to maximize another function that has a different set of parameters, which I'll call alpha. And that 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 1/2 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. Holy cow. Now, it's so obvious how you get from one step to the other I'm not going to bother to explain it to you. But instead tell you to 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, and that is the same thing as maximizing 2 over the length of W, and you buy that it's the same as trying to minimize 1/2 times W squared, 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. Okay? All right. All right, so trust me on this. This is what it is that we want to solve. Now, it turns out that we can run little programs to solve this, and you end up with answers. But what's really interesting is what this equation actually tells us about what we're trying to do. So let me just show you. This'll be just, talk a little bit about the properties of this equation, and the property of the solutions to this equation for a second. Okay? hm. Okay. So let me move a few things around so that we can look at it

1.4.7 Still More Support Vector Machines

Okay. So we've done a little bit of moving, moving stuff around, and kept the same equation of before. Remember, our goal is to use quadratic programming to maximize this equation. So let me talk a little bit

about the properties of the solution for this equation. So here's the first one. 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 that's the little W, not the big W That's the little W, not the big W. That's right, okay? Neat. Yeah, that is kind of neat. So it's really easy to do. And of course once you know W it's easy to recover B. You just find the value of X, you stick it into W, you, that you know is equal to plus 1, and then poof, you, you can find out B. So you can recover W directly from this and you can recover B from it in sort of the obvious way. But here are some other properties that are a little bit, little bit more interesting for you. So 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. So here's the one that I'm going to tell you. It turns out, okay, that alpha, each of those alphas are mostly zero, usually. So if I told you that in the solution to this, most of the alphas that you come back are going to be zero, what does that tell you about W? So W is the sum of the data points times their labels times alpha. And if the alpha is zero, that 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. That is exactly right. So basically, some of the vectors matter for finding the solution to this, and some do not. So 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. Yeah, well the ones with non-zero alphas. So you basically built a machine that only needs a few support vectors. Oh. So the, the data points for which the corresponding alpha is non-zero, those are the support vectors? Yes, 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. Now Michael, let me let me do a quick quiz. Yea, yea!

1.4.8 Optimal Separator Question

Okay Michael, I've drawn a little teensy tiny graph on the screen. Can you see it? Yes. Okay, and some points are positive, some points are negative. And let's just imagine, for the sake of argument, that the green line that I've drawn in between them is in fact the optimal separator. It probably isn't, but let's just pretend that I drew the right one. Now, I've just told you that a lot of the alphas are going to be zero, and some of them are not. So, thinking about everything that we've said, and thinking about what it means to build, this, what it means to build, an, an optimal decision boundary, maximizing a margin. I want you to, t, point to one of the posit examples that almost certainly is not going to have a non-zero alpha, and one of the minus examples that almost certainly is not going to have a non-zero alpha. That is, are not a part of the support vectors. You want something that does not not have a zero? Don't confuse me. Do you want [LAUGH] something that's, that, that has a zero alpha or a non-zero alpha? I want something that has a zero alpha. Got it. That is, in some sense, doesn't matter. Okay, go.

1.4.9 Optimal Separator Solution

Alright Michael. So. You think you got an answer? Yeah. It's interesting. 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 non zero, sorry, I would put zero alphas on the lower left hand minus and the, one of the upper pluses. Yes, and, you know, I haven't actually worked out the answer here. But, both of these pluses probably don't matter. Certainly, this one doesn't. certainly, this minus doesn't matter. Maybe this one doesn't matter, either. But the point that she raises, is exactly right. The, the points that are far away from the decision boundary, and can't be used to define the contours of that decision boundary. Don't matter, whether they're plus or minus. Does that make sense? Yeah, cool. Does this remind you of anything? Nearest neighbors? That's almost always the answer. Why does it remind you of nearest neighbors? because only the local points matter? Oh, that's a good answer. I was going to have a different answer. Know what my answer was? What? 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. Oh, I see. 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. Right, so it's just another way of thinking about instance-based learning, except that rather than being completely lazy, you put a lot, some energy into figuring out which points you could actually stand to throw away. Interesting. Okay. Yeah, I think that's kind of interesting. I think it's kind of cool. So good. So you got that. Well let me show you one more thing, Michael. Alright, so you got this notion of there being very few of the, the support vectors that you need, but I want to point out something very important about some of the parameters in this equation. So we just got through talking about the alphas, right? 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 xs come into play with one another is here. So Michael, generally speaking, given a couple of vectors, what does xi transpose xj actually mean? It's the dot product. Right, and what is the dot product? It's like the projection of one of those onto the other right? Yeah, and that ends up giving you what? A number. Yes. Does that number kind of represent anything? And if you say the dot product, I will climb through the screen and kill you. What about the length of the projection. Right, and what does that kind of represent for you? Well, I guess in particular if the x's are, well if there are five going to each other than it's going to be zero. But if they kind of point in the same direction, they're going to be, it's going to be a large value, and if they put in opposite directions it's going to be a negative value. So it's sort of kind of indicating how much they're pointing in the same direction. So, I guess it could be a measure of their similarity. Rightl, I think that is, that is exactly right. 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 boundry. 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.

1.4.10 Linearly Married

Okay, Michael. So, I've drawn a little thing on the screen for you. because want to illustrate a problem. So, you see this little graph that I have? It looks just like all the other graphs you've drawn so far. More or less. I think there are fewer points, but I think you're right. It more or less looks like the same one. And, we found the line that is linearly separating the two clouds of points. [LAUGH] And you agree, that's a technical term. And you agree that it linearly separates them. And you're even willing to accept, for the purposes of this discussion, that is in fact the line that maximizes the margin. Sure. Even if it's not. Okay, cool. So, this is easy right? So, what are you going to do, Michael, if I take this now and I add one more point? And here is the point that I'm going to add. Hmm. It's another minus. I can think of two things to do. One is I can put a vertical line through it, and that makes things nearly separable again. That's usually not allowed. And the other one is, since it's a different color I feel like I could just erase it. No. No, all right. That's not acceptable. I control the pen. . I mean in some sense the margin is now negative, right. because this, this point, 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. That's true. It's not linearly separable. Okay. Can you think of some clever way to fix it? Well, again, I mean, I feel like, I mean I was making a joke before. But maybe a reasonable thing to do is to have some kind of, you know, I'm allowed to delete some number of points thing. Or, or, find the line that linearly separates the positives and the negatives, while at the same time, minimizing the number of things that are on the wrong side. Right. So you wouldn't be linearly summarizing, separating them. But you'd be finding a line that make sort of the minimal set of errors while also maximizing the margin, if you kind of were allowed to flip a few points from positive to negative or negative to positive. I think that's what you said. Yeah. And then you need to have kind of new knob now to trade off those two things. Right, and it turns out you can do that. We're not going to about it. Instead we're going to make a homework assignment about it then, and let the students think about it a little bit. But I think even that little clever thing that you came up with, even though it is used, won't work in another case that I'm thinking of. So let me draw that case for you. Okay, Michael, how does that look? Well, the good news is it's not exactly like all the other graphs you've drawn but it is, it is very similar to it. There's going to be a linear separator that falls between that bottom plus and the line of minuses. And there's a maximum margin one. So its, this seems all within the bounds of what we've been talking about. ¡¡ That's true. You're very smart. What if I add just a few more points? Okay, that doesn't seem like just a few. [LAUGH] Wait, so I guess this is different from the other example because where before, as before, it looked like there was maybe like an outlier or an intruder. Now, it's like there's a ring around the whole thing. Oh, is that why they're linearly married? Yes. Ha! All right. So now, you know, you can draw lines all day long, and it's just not going to slice things up. That's right. So now we have to come up with some clever way of managing to make this work, or we're going to have to throw away support vector machines altogether. And I like support vector machines, so I want to avoid doing that. So here's the little trick we're going to do. I am going to change the data points without changing the data points. Ouu. That's going to be a neat trick. That seems possible and impossible. Here's what I'm going to do, Michael. I am going to define a function, okay. Here's the function. I'm going to create a little function here. And this function's going to take a data point, okay. And because we've been using to many X's and I's and Y's and J's, I'm simply going to call it Q, okay. Now, Q is one of the points that are in, it's in the same dimension as these other points. So in this case, it's two dimensions in a plane, okay. And I'm going to transform that particular point Q into another kind of point. But I'm going to do it in another way that doesn't require cheating, okay. You ready? Sure, Im perplexed

but okay Okay. So what is Q? Q is in the Y, is in the plane. 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. How does that look, Michael? Strange. So you took, so Q is a two dimensional point. So it's got, Q1 and Q2 are its two components. Yup. And you're saying, you're going to 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 the sec, that value is the second component squared. And now, just because apparently it wasn't weird enough, you're going to throw in a root 2 times the product of those two as the third dimension. Okay. That's right. You're, you know you have an interesting sense of style. I do. I kind of like it. Now 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 just because, why not? Okay. Okay? All right. Now, Why did I do this? 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. So let me remind you what that equation was. All right, Michael. So I've written up the equation for you, as I promised I would remind you, of the quadratic program that we're trying to solve. I didn't write down all of the constraints and everything. I'm hoping that you remember them. And I wrote it up there for a reason. And the reason I wrote it up, is because you'll recall, just not too long ago, I asked you to talk about XI and XJ, and what it looks like in this equation. And what I think we agreed to, or at least I know I 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, fee or fi or whatever the right pronunciation is, what would happen if I took two data points, and I did the transpose or the dot product between them. What would I get? So, let me just write that out. Or, I don't know, you can try telling me if you want to. So let's make that, let's make that let's make that simple, Michael. And in fact, let's make it so simple we can make it into a quiz.

1.4.11 What is the Output Question

Okay, Michael. Here's a problem I want you to solve. 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. Okay? Sure. And 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, or phi. You got it? Yeah. Okay, so. X transpose Y, gets turned into phi X transpose phi Y. What's the answer? What's the output? In terms of X1, X2, Y1, Y2. Go.

1.4.12 What is the Output Solution

All right Michael, you got the answer? I'm still carrying some squareds. You want to talk it through? Okay. Sure. So, x is really x1x2 and y is really y1y2 and phi x is now this crazy triple x so I, so I wrote x1 squared x2 squared, square root of 2x 1x 2. Yeah. That's the vector that we get for phi x. Then for phi y, I get y1, it seems like it would be helpful to see this. You want me to right it down? Sure. Okay. So that turns out to be the same as what did you say? x1 squared, x2 squared. Root 2, x1, x2. Root 2, x1, x2. Okay. And then the y vector gets transformed to the same thing, except for with ys, y1 squared comma, y2 squared comma, root 2, y1, y2. Okay. So, then, the, the dot product is just, the, the products of the corresponding components summed up. So x1 squared, y1 squared plus, Okay X2 squared, y2 squared, Mm-hm 2 x1, x2, y1, y2 That's right. So here's a question for you, Michael. Does that look familiar? Based on your years of thinking about algebra. Oh, thanks for writing it that way! I see. We can, is this right? So it's, it's like, we can factor this. Mm-hm. It's like x1 plus x2 times y1 plus y2. Yeah, that's right. Wait, is that right? No it's not right. X1 Y1 Mm-hm. Plus X2 Y2. Whole thing's squared. Right. So, let's right that down. So if you factor it out, you're right, this is exactly equal to, X1 Y1 plus X2 Y2. Squared. And what's an even simpler way of writing that? I see. x1 y1 plus x2 y2 looks like a dot product itself. It looks like the dot product of x and y. Oh, it's x transpose y on the inside, and then we square it on the outside. That's exactly right. So now do you see why there was method to my madness when I created the phi function? Not yet. So you're saying, if we're just dealing with dot products, now I'm still a little confused. So, so it is the case, that you define these in an interesting way, so that the dot product became the square of the old dot product.

Right, so now let me make two observations. Okay. Here's observation one. What's x transpose y? I mean geometrically, what does that represent? The length of the projection of y onto x. No I mean geometrically, like go all the way back to geometry, third grade. We didn't do transposes in third grade. [LAUGH] I know we didn't do transposes, but you did equations like this, or at least later you learned they were equations like this. Here, pretend you're in third grade, and I said talk to me about geometry. What kind of words would you use? Oh, what's in geometry? Triangles. Circles. Yeah. Circles! Did you say circles? Sure, but only because I thought you might have wanted me to. Did you say circles? That's a really ugly looking circle, sure. Sure. 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. Interesting. So if and this is my second point, Michael. 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. Right, and 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, of course. Of course. Because, I see, because they are the ones that were closer to the origin. Right. So they get raised up less. Wow. Okay. So using that third dimension. Right. Now, this is a cute trick, right? 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? Because it fits the circle pattern that you wanted. But 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. So you're saying 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 that x1 err 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? Yes, that's exactly right. That's crazy. It's so crazy it has a name. And that is 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. So we never really used phi. 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. So is it important that such a phi exists? Or is it just the case that we can, you know, we could, we could throw in a cubed, we could throw in a fourth, we could do a square root and a log, like can we do anything we want there in that Xi transpose Xj? Or are we constrained to only use things that somewhere out there, there is a way of representing it as a regular dot product? Well, that is an interesting question. The answer is you can't just use anything, but in practice it turns out you can use almost anything. And the other answer to your question is, it turns out for any function that you use, there is some transformation into some dimensional space, higher dimensional space, that is equivalent. Whoa. 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. So, which part is the kernel? So, the kernel is the function itself. So, in fact, let me, let me, 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.

1.4.13 Kernel

Okay, so I've the cleaned up the screen a little bit, Michael to, to make this a little bit clearer. Now, let's look at this xi transpose xj. And I'm now going to replace it with something. So, 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. And again, 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. Just like we were injecting domain knowledge when we were thinking about k-nearest neighbors. Yes, everything has domain knowledge and everything ultimately comes back to k-nearest neighbors. I don't know why and I don't know how, but it always seems to. So the k in k-nearest neighbors, and the k in kernel really stand for knowledge. Oh, wow, that's pretty good. We should write a paper with that title. [LAUGH] So the room neatness here, the neatness here two fold. One is, 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. But, the second bit is, 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. Right? So is, and that is a kernel and another kernel is X transpose Y? Yes, that's something I would call a kernel. And the other kernel we talked about was just X transpose Y by itself. That's, that's a kernel too, isn't it? Oh no, no. That's right. That's right. That's absolutely right. So, that's a different kernel, you're absolutely right. Just X transpose Y. Is another kernel. Actually we can 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. Hm, and I would, yeah, okay, good. And that should look familiar. It reminds me of the regression lecture. Exactly, where we were doing polynomial regression. So 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. And there're lots of other kernels you can come up with, Michael. So. Here's just a couple. I will just sort of leave em. Leave em up to you, to think about. And there's, there's tons of them. So, here's one that I, I happen to like. So, that's a a sort of, radial basis kernel. Does that look familiar to you? Well, to me, make sure I understand that it's doing the right thing. So if x and y are really close to each other, then it's like, 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 kind of like the sigmoid where it, it transitions between zero and one but it's not exactly the same shape as that. Right in fact it's symmetric, that the square of the of the distance between you in making an actual distance, makes it always a positive value there. Or at least a non-negative value there. And so it becomes symmetric, so it looks a lot more like a, like a gaussian with some kind of width which is represented by sigma. And there are tons and tons of these. 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. And there're tons and tons of these you can come up with. And there's lots of, been a lot of research over the years on what makes a good kernel function. The most important thing here, I think, is that it really captures your, your domain knowledge. It really captures your notion of similarity. You might notice, Michael, that since it's just an arbitrary function that returns a number, it means that X and Y or the, 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, I don't know, strings or graphs or images? Absolutely. You could think about two strings. 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. You could talk about words like cat and lion, and decide those are more similar than cat and mosquito, for example. because they, because of the ears? Yeah. Mainly because of the ears. All right. But then I, then I think I understand. Okay. Good. So you might be curious, Michael, whether there are any bad kernel functions. There is actually an answer to that. 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. Have you ever heard of the Mercer Condition? I've heard the word. I actually used to live near Mercer County in New Jersey. You did? Yeah. Oh. So then I guess it's the condition of living near where Michael used to live. Now, so 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. Gotcha.

1.4.14 Summary

Okay, Michael, so that gets us to the end of Support Vector Machines. So, let's recap. What have we learned? Well, we learned that support vector machine is not a command or a political statement. We talked about how a margin is a, is a useful concept in trying to understand how well a linear classifier might generalize. Okay, I like that. Lemme write that down. Margins are important. So we learned about margins In particular their relation to generalization and overfitting. Okay. In particular, we, we would like, given the choice, to find a linear separator that has the largest margin. Right, right. So maximizing margins. Right. At least when it comes to margins, bigger is better. Then we talked about how you could actually find. A linear separator that has maximum margin. I think we turned it into a quadratic program. Yes. We found an optimization problem for finding maximum margins and they turned out to be quadratic programming. And it was the dual of the quadratic program that showed us how, what the support vectors were. The support vectors were the points from the input data. That were necessary for defining that maximum margin separator. Right, right. So, we actually figured out what support vectors were. And then we tied all that in to instance based learning and other kind of ensemble methods. And so you could sort of think of support vector machines as being eager lazy learners. Or only as lazy as necessary to represent what you needed to represent. Because the, the classifier was based on just a subset of the data, or, or, or the raw data was being used for defining the classifier. Exactly right. Alright, so is there anything else? Oh. Oh. Right. Then there was the whole idea that, well, linear doesn't always seem like enough, but, we can project. Data into a higher dimension space and do the comparisons there. And that only made one little change to the algorithm. In particular, the dot product could be turned into some other similarity metric. And you called that the kernel trick. Right. love the kernel trick. And as you say, we took, basically, X transposed Y. And generalized it to a generic similarity function k of x comma y. And that actually ended up representing all of our domain knowledge, which will come up again and again throughout this course. What are the levers we have For expressing domain knowledge. Okay so, anything else for writing down Michiel? I think you said that kernels had to satisfy the merciful condition. No, no, no, mercer condition. Oh, okay, well that makes more sense. The mercer condition is interestingly quite merciful, because it's ok to use kernels that don't necessarily satisfy the mercer condition, often in practice it still works. Okay, well good, I think that is mostly it. So i guess we're done. Wait a second. Didn't you say that you were going to explain boosting to us? You kind of suggested maybe you'd do that during the boosting lecture. But then you said it would be during the SVM lecture. And now the SVM lecture's over and we still don't know why boosting doesn't over fit. Oh. That's a good point. That's a good point. I had forgotten about that. Thank you Micheal, this is why I keep you around, to remind me about stuff like this. Okay, let me take a moment then, just a moment. To see if we can tie together, over fitting and boosting about what we just learned about SVM's.

1.4.15 Back to Boosting

Alright, so back to boosting, Michael. So as you recall the little teaser I left you with last time, is that it appears that boosting does not always over-fit. And a little graph. That's true, but 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. So, given what we talked about with support vector machines, and what we spent most of our time thinking about, what do you think the answer is? Well I, I don't think I would have asked again if I, had a thought about it. But you mean You want me to connect it to support vector machines, somehow. Well the, the thing that was fighting over fitting in support vector machines, was trying to focus on maximum margin classifiers. Here, let me, let me try to explain to you why it is

that you don't have this problem with With overfitting at least not in the, in the typical way as you keep applying it over and over again like you do with something like neural networks. And it really boils down to noticing that we've been ignoring some information. So, what we normally keep track of is error. So error on say a training set is just, you know, the 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 given. Make sense? Yes, a lot of the algorithms we talked indirectly have something like that. So, like in a nearest neibhbor method, if you are doing five nearest neighbor and all five of the neighbor agree, that seems different than the case with vote one way and two vote the other. Right. And in fact, that's a really good example. 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. Okay. So what does that mean in the boosting case? Well as you recall, the final output of the boosted classifier is given by a very simple formula. And here's the equation here that h of x is equal to the sine of the sum over all of the weak hypothesis that you've gotten of alpha times h. So the weighted average of all of the hypothesis, right? And you just simply, 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. Produces zero. Just throw up your hands. So I'm going to make a tiny change to this formula, Michael. Just, just for the purpose of sort 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. Now what does that end up doing? Okay, so the weights. I'm getting a. There's Alphas in the SVM's too, so I'm getting a little confused. So that I'm. I think these Alphas all have to be non-negative. Right. But they kind of like this support vector values, in that there could be zero, if, if that hypothesis isn't come into play? Well, but they want in that case, the, the alpha is always set to be the natural log of something. Oh, oh, oh, and also these alphas are applied to hypothesis whereas the alphas in the, in the SVM settings were being applied to data points. That's right. So, unfortunately in machine learning, people in, invent things separately and re-use 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. Gotcha, okay. So this, 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. Right. And what it ends up doing like often is the case in these situations, is it normalizes the output. So it turns out that this value. Inside here is always going to be between minus one and plus one. Okay? But otherwise it doesn't change anything about what we've been doing for boosting. So you might ask why did I go through the trouble of normalizing it between minus one and plus one? Why indeed? Well it's 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. Okay? Is that an x or a plus? That's a plus. Okay. So it's a positive example and it's near plus one. Right. So this would be something that the algorithm is getting correct. Yes, and 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. Hmm. 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. Because for example I could also have a plus value way over here. So I am very, very confident in my very, very incorrect answer. Mm. So this is my daughter, for example. [LAUGH] She's very confident whether she's right or wrong. [LAUGH] Okay. And so now imagine there's lots of little points like this. And if you're doing well, you would expect that, you know, very, 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. Okay, so does this make sense, Michael as a picture, Oh yeah. What, what might be going on? Absolutely. Okay, good. So now I want you to imagine that we've been going through these, these training examples, and we've gotten very, very good training error. In fact, let's imagine that we have negative training error. I'm [LAUGH] Wow. In fact, let's imagine that we have no training error at all. So we, 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. So here's what ends up happening in practice, right? 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 and drifting and drifting until it's all the way over here, this minus starts drifting and drifting and drifting untili it's all the way over here. And the same happens for the pluses. And as you keep going and 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 and up and 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. So what does that look like to you, Michael? This picture? Yeah. I mean that there's a, there's a big gap between the left most plus and the right most minus. Which, you know, in the context of this lecture reminds me of a margin. That's exactly right. Basically what ends up happening is that as you add more and more weak learners here the boosting out rhythm 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. And what do we know about large margins? Large margins tend to minimize over fitting. That's exactly right. So it, 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 over fitting even as you add more and more learners is that you're increasing the margin. And there you go. And if you look in the reading that we gave the students there's actually a detailed descritpion about this in a proof. Cool. Okay. So, there you go, Michael. Do you think, then, that boosting never overfits? [SOUND] Never seems like such a strong word. I mean, the story that you told says that it's going to try to separate those things out, but I guess I guess it doesn't have to be able to do that. I mean, it could be that for example all the weak learners are I dunno very unconfident very inconsistent. Hm. Okay, well you know, maybe, maybe it's worthwhile to take a little diversion here to take a five second quiz. I think it's worth the time. Done!

1.4.16 Boosting Tends to Overfit Question

Okay Michael. So here's a quick quiz. So we just tried to argue that boosting has this annoying habbit of not always over fitting, 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. And we don't want that to happen. So here's a little quiz to see if we can figure out the circumstances under which boosting might over fit, or tends to over fit. So here are five possiblities. Let me read them to you. Tell me if they actually make sense. So here's possiblity number one, boosting will tend to over fit if The weak learner that it's boosting over, always chooses the weakest output that is, it, among all the hypothesis that it finds that do better than chance over the training with whatever given distribution. It always pick the one that is still nonetheless closest to chance, while still being better. Well, why would it do that? Just to be difficult. Alright, and so you want to know, whether that makes it over, would [INAUDIBLE] make it over fit? [UNKNOWN] boosting overfit. Okay. Alright. The second one is weak learner actually ends up using...or the weak leaner itself that boosting is using is in fact a neural network learner. And just for a little specificity, let's say this is a neural network that has many many layers and many many nodes. So, you know, it's a big powerful neural network, alright? the other option is... Boosting has a lot of data. So you're trying to learn, your training data is actually very, very, very large. You have lots and lots of examples. The fourth case, is that, the true underlying hypothesis,the true underlying concept, is in fact non linear. So you can't just draw a line. And then the fifth case is that we let boosting train much too long. Whatever that means. Let's just say we let it train a lot. Not just a thousand iterations but a hundred billion iterations. Okay. Okay. Billions and billions of itterations. Okay. You got it? Yeah. Alright. Go.

1.4.17 Boosting Tends to Overfit Solution

All right, Michael. What's the answer? All right. Well, let me start off with what I think the answer isn't. So, 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 training too long. Oh, nice to know you were listening. [LAUGH] Boosting training too long, seems like not a good reason for it to overfit. You're correct. All right. Boosting tends to overfit if it's a nonlinear problem. So, that doesn't seem right. I mean I guess, no, this one just doesn't seem right at all. Like I don't see why, why the problem being linear or nonlinear, has anything to do with overfitting., Okay. 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 of, you know, there's a lot to fit. There's a lot going on there. It's unlikely to overfit. Right, and 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 training zero generalization error. because it'll work on the testing data as well, because it's in there. Right. All right. 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. Sure. So, if, and now we're doing boosting over that. So 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. OK, so you're, you're, so for now let's put a little question mark to it. You think that might be the right answer, but you want to think about it some more? Yeah let me, 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, like I feel like the only one, the only one of these choices that is likely to be true is the second one. And that is, in fact, correct. So let me give you an example of when that would be correct. So let's imagine I have a big powerful neural network that could represent any arbitrary functions. Okay, 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. Interesting. And not only will it overfit, but it'll just, it'll be stuck in a horrible loop of error. Right. So that's why this is the sort of situation where you can imagine boosting 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. Interesting. Now, I do want to have a little semantic argument with you for a moment, Michael. You used the word strongest at some point, when you were talking about using the weakest output. And I just want to point out that, that doesn't really mean anything. What do you mean, it doesn't mean anything? Well, so what's a strong, what would you call a strong learner? One that is far away from it. If a weak learner just has to do a little bit better than a half, it seems like a strong learner would be something that would be very close to being accurate. Right. Of course, on the other hand, if by that definition all strong learners are also weak learners. Sure. Because anything that does better than a half is still doing better than a half, which is all it requires to be a weak learner. Yeah, but that's kind of true of people too. Like a strong person is also a weak person. No. Well it depends how you define it. So, if you say a weak person is someone who can at least lift their own arms, then strong people are also weak people in that they can lift their arms. Yes if you define it that way and if I define blue to be purple, then I can say blue is purple. But that's not how people define weak people. They define weak people, by saying they can't lift more than, not that they can lift at least as much. I see. So it's this piece of terminology that boosting uses that is in error, not me. That's one interpretation. It's not the one that I would use, but it's one interpretation. When you say something like a strong learner, I mean, it makes sense to use that kind of term, and sort of throw it around, and say, well, by a strong learner I mean someone who's, or a learner that's going to overfit, or is going to always do really well on the training data. But in kind of a technical definition it's very difficult to sort of pin down. So don't get too caught up what a strong learner means if you want to write a proof. Seems fair? Good point yeah, also, also that this whole notion that strong is sometimes defined as not weak. And it is not the case that if you have something that's not a weak learner that it's, then it's a strong learner. In fact, it's no lear, no learner at all. Exactly. So, a weak learner's just defined in a way that basically says, it gives me at least some information. Good. Let me just throw one more thing in here and then we can stop talking about this. There's another, a couple of other cases where boosting tends to overfit. The one that matters the most, or comes up the most, is in the case of pink noise. Did you say, peak noise? I said, pink noise. I even wrote it in red, which looks like pink. It's a strong pink as opposed to a weak pink. [LAUGH] I'm sorry. There's no way for that to be obvious from what we've talked about, but as a practical matter, pink noise tends to, cause boosting overfit. Okay, but this is not a term I'm familiar with unless you're critiquing the musical stylings of a particular performer. [LAUGH] No. Although I did recently see, see them in concert. But that's a whole other conversation. Okay, so pink noise just means uniform noise. I thought white noise was uniform noise. No, white noise is Gaussian noise. Okay, so pink noise is uniform noise and white noise is Gaussian noise. This is why, Michael, by the way, if you ever try to set up a studio or a cool stereo system in your house, you want a pink noise generator. So that it covers all the frequencies equally, not just the white noise. generated. Hm. But boosting tends

to overfit in those sorts of circumstances. And you can read more about it in the notes if you want to. But the one that I want I really want people to get is, that if you have an underlying weak learner that overfits, then it is difficult for boosting to overcome that. Because fundamentally you've already done all of your overfitting and it's, there's really not much for those things to do. Okay. Got it? Got it. Excellent. It all ties back into margins, and it's all one big story, which I think is the lesson of all of machine learning.

1.4.18 Summary For Real

All right Michael, so we're back. We have learned a bunch of stuff about SVMs and you forced me to live up to my promise to connect margins. Which is kind of the big cool thing about SVMs, the sort of fundamental theoretical construct that's underlying SVMs. You forced me to connect that back to boosting. So I think we've learned everything we need to learn about support vector machines for now. What do you think? If you say so, sounds good. All right. Well, I will talk to you next time, Michael. Thanks. Bye.

1.5 Comp Learning Theory

1.5.1 Learning Theory

All right, so that quiz maybe was ill-placed, in that it was about what this is not about. What this is about is computational learning theory. And 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, 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, and why certain algorithms are effective and ineffective. Okay, so Michael, so can I ask you a question then? Sure, please. So, it sounds to me like you've just justified this in the same way that a computing theoretician might try to justify theory. Are they related? Right. That's a very good observation. 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 a-, analyzing algorithms in computing. So yeah, that's exactly right. Okay, great. In fact, let's let's, let's leverage that analogy. And we'll do it in the context of a quiz.

1.5.2 Resources in Machine Learning Question

So often in the theory of computation, we analyze algorithms in terms of their use of resources. And it's usually time and space. So like, when we talk about an algorithm running in say n log n time, we're saying something about this, the time that it takes. So if you say that it uses n-squared space, we're talking about the, the amount of memory that it takes up as the function of the growth of the inputs. So what sort of resources do you suppose might matter in the context of computational learning theory? So we're just, we're trying to select among algorithms. We want algorithms that use their resources well. What source of resources would we analyze? If there's multiple possible answers, just fill in one of them. We'll just see if it's any of the ones on our list.

1.5.3 Resources in Machine Learning Solution

All right. So Charles, what do you think would some reasonable resources to try to manage in a learning algorithm? Okay, well I was thinking of three. Because three is my favorite number. Two of them are what you already have written up there. 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. That's right, so if in particular we have a learning algorithm and they, they do more or less the same things, but one runs in n-squared time and the

other runs in exponential time, we'd really like to have the one that runs in the shorter amount of time. Or, in particular, if there's a, if we define a learning problem and we say well, We could do this computation, but its MP hard. Then maybe that's a problematic way of defining the problem. So yeah, time definitely matters. Space for the same reason. And those are the same things that happen in in, or that are relevant in regular algorithms. What about anything specific to the machine learning setting? Okay, so that was my third one. So 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. Yes I like the, I like the word samples. Though data is probably pretty good. Thing to stick in there as well or examples. Those should all be okay. Yes. Indeed. Yeah. We, 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. Oh, that makes sense.

1.5.4 Defining Inductive Learning

All right. So we should take a moment to define inductive learning. So 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. So 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. So one of them is just a simple thing like, 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 you know, got a, 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. How many, how much data does the does the algorithm get to see? . Is there a letter you like for that, Charles? no. Okay. Is there a letter you like? I don't know, sometimes I like M for number of samples. But I thought maybe you went, you would want N because, you know, things tend to grow with N. Yeah. I did want N, but then I thought well, we can't use N because we use N for everything else. [LAUGH] Fair enough. 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. Why, why do you think that could come into play, Charles? Well, if you don't have a very complex hypothesis class, then there's some things, well, do you mean the complexity of the class or the complexity of the hypotheses in the class? That's a good question. Do we mean the complexity of the class or the complexity of the hypotheses in the class? Well it depends on how we measure complexity. But the complexity of the class could be like the sum of the complexities of all the hypotheses in the class, so it could be both. Hm. Well if, if you mean, you know, 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, well, represent much. It will be hard for you to learn anything complicated. So that could matter. Sure. That's right. Now, could you see a downside to having a complexity class, I'm sorry, a hypothesis class that is very, very complex? You mean like my daughter? sure. I think you could, it would be much easier to overfit. Right. So getting something that actually works well might be challenging. You might need a lot more data to kind of nail down what your're, what you're really talking about. So it's a bit of a double edged sword. All right. So then, there, another issue is, well, you know, it may be easy to learn if you don't have to learn very well. [LAUGH] So the, 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. And and so those, those are kind of the main complexity related quantities that I, that I wanted to talk about. But 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. Let me just first say that when I talk about the manner in which training examples are presented, there's, there's two that I think are really, really important to look at. One is batch and 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. A big batch, exactly. Or it could also be presented on line, 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. Mm-hm. We haven't really talked about algorithms that work that way, but it is useful in the context of computational learning theory to have both kinds of algorithms. They have

different sorts of behavior. All right. So let's talk about the manner in which training examples are selected.

1.5.5 Selecting Training Examples

All right, so turns out it, it matters how we select training examples when when a learner is needing to learn. So 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, so what do you think, Charles? Are there, there any ways you can think of for selecting training examples? Well, so, I, I, I so here's how I think about it. 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. I see, so in this case, the learner would be selecting the training examples. The learner would say, I, you know, here's, here's an input x, would you please tell me c of x? Right, that's exactly what I mean, right, right. 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's, it actually is trying to help the learner. Sure. So it's a, friendly teacher. Okay, I've had some of those. Good, all right so, the, it could be the, the learner asking to try to get data. It could be the teacher asking to try to lead the learner to something good. And, anything else? Well, so, I like this, this way of thinking about it. Because it sort of makes sense in my world as a professor, but it doesn't actually sound like either of things is what we've been looking at so far. What I think we've been looking at so far is that somehow the Xs and the CXs, 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. Yeah, I like that. 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, like, the world around them. And, and, the things could come from any of those. I guess another possibility would be that questions are asked specifically in an evil way. All right, so all of these different kinds of distributions or, or ways of selecting training examples are actually going to come into play in the stuff that we're talking about. Let's let's use the first couple to get a sense of why these things might be different from each other. So let's go back to this notion of 20 questions. Okay.

1.5.6 Teaching Via 20 Questions Question

Right, so let's go back to, to 20 questions that we talked about a number of lectures ago. And we're going to consider the following analogy. That in, in some sense, what is a, what is a learner? What is an inductive learner trying to do? It's trying to find the best hypothesis in some class H. And the hypotheses all map input to yes or no. Let's say that what we're trying to do in twenty questions is, deal with our hypothesis class H, which is maybe a set of possible people. And so, these are the possible answers to the 20 questions problem. And, X is the set of questions that you can actually ask. So X, they're kind of like training examples. That's how the learner's getting information about what the right answer is. But no question, in and of itself, might actually be sufficiently revealing of what the hypothesis is. All right does that, does that analogy make sense to you, Charles? Makes perfect sense to me. All right, good. So, now let's 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, you know, 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. So why, why are these different from each other? What does the teacher have that the learner doesn't have? Well, the teacher actually knows the answer. Indeed. Alright, so let's let's just do a quick quiz and say, 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? Okay. Go.

1.5.7 Teaching Via 20 Questions Solution

Alright we're back. So let's, let's, let's think this through, so the teacher gets to suggests to the student any, any question. The teacher knows the answer, and the teacher would like the learner to get the answer as quickly as possible. Not requiring the full 20 questions if, if possible. So what do you think, what do you think the teacher could do as a good strategy here? For 20 questions? Yeah. So, if, if we think of the

set of possible people as being these sort of hypotheses. And you know which of the hypothesis is correct and which is wrong, then presuming the teacher can keep all this in her head, I guess the right thing to do would be to always pick an X, such that it gets rid of as many of the hypotheses is possible. But what, what, do I mean by that? I mean, it gives the most information. Well, but there's a, there's a very strong kind of information that you could get in this particular case. So, so, so let's imagine that this set of questions is, is this sort of, you can ask any question in the universe. So, I give you some question, I oof. Let's, let's try this because remember we did this before, you thought of a person, and you had me ask questions. So I want you to think of a person. Okay. Alright, and I want you to tell me what question I should ask, that's going to figure out who the person is as quickly as possible. Okay. Is this person as good in his field as his namesake is in his field? Is it Michael Jordan? Yes. Is it the Michael Jordan who's a professor at Berkley? Yes. Okay. You could have asked if I say, is it Michael I Jordan? Yeah, I don't think I knew the initials of the two Michael Jordans. Yeah. Michael Jordan, for, for people who don't know is a professor at Berkley and one of the giants in machine learning. And he is literally the Michael Jordan of machinery. [LAUGH] Alright, but, but, but you could've done this in one question, right? If you had to ask me, if the first question was, is the person, and then the actual person. You knew that the answer to that would be yes. And so I would've gotten it in one step. Hmm, that's true. So, given a, you know a, a sufficiently, helpful teacher. You can, you can do this in one. Well, that make sense, right? So basically, if my questions include, is this the correct hypothesis, then you should always just ask that question. Right. But that seems, that seems like cheating. [LAUGH] Yeah, I would, I. You're right and all I was really trying to do is show that, it's a very different problem, when the, the question has to come from the learner's side. That kind of question from the learner's side would be pure folly. Because it would, it, it, the learner would not know what person to pick. If the person started, if the learner started to ask questions like, is the person Michael I Jordan? The answer's very likely to be no. In which case that's not very helpful. Now, a really helpful teacher could potentially change the target. But that's, that's definitely cheating. Mmm. OK, fair enough.

1.5.8 The Learner Question

All right. So the learner, if the learner is choosing which questions to ask the learner is in a distinctly more difficult situation because it doesn't actually know what the right answer is. So whereas the teacher could ask questions there were maximally informative. That is to say, whittle down the hypothesis set to one in a single question. The learner doesn't, doesn't have that ability. The learner can't know which, what whether the question that's being asked is going to have a yes or no answer. So again, assuming that the learner can ask any question it wants, but doesn't know what the right answer is. How many questions is it going to need to ask before it can it has whittled down the set of possible people to just the one that is the right answer? So I gave four helpful formulae here. All in terms of the number of people that we have to choose from. Is, do we have to ask questions that is the size of the set of people? Is it going to be more like the log of size of the set of people? Since there's, you know, lots of difference ways that the person can be chosen. Do we have to look at two to the number of people? Or again can the learner be as powerful as the teacher, and get the whole thing in just a single question? Alright, so, is, is, is this a little clearer Charles? Can you, can you work this one through? Yeah. I think I can work this one through.

1.5.9 The Learner Solution

Well so, 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. And from the teacher's point of view, I might be able to pick a question that gives you the answer right away, depending upon what h is and what x is. But from the learner's point of view, I sort of can't. So what does the learner know? 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. Let's say l. And if the answer is no then it's going to be n minus l, right? And so you said that in the case of the teacher, pick a question so that this yes is going to whittle it down so that this is just going to be the one right answer. That's exactly right. That, that's assuming that x is set up in such a way that you could ask that kind of question. Basically a question could be of the form, is it this or this or this or this or this? Any pattern of answers that we want, we can construct the corresponding question. Okay, that makes sense to me and I don't think it changes, my thought process, so, so, that's good. 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? Yeah, I'll write that down. I like that. Now, you could say, that's what the teacher did. But, let's take what the teacher did, the teacher knew that there was a hypothesis where it would whittle it down that, since the answer was yes, it would whittle it down to one. But if the answer had been no, it would have only gotten rid of one. So since the teacher knew the answer the teacher could pick that question. But 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. Alright so how many. Let's see so under the assumption the target hypothesis was chosen from H say, uniformly at random. Right. 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 covers every, the hypothesis base covers everything. Then the best you can do is eliminate about half of them, on average. Well, so, but in particular, if you have a question like this, x. That, you now, you ask it when there's n possible people. And then, after you ask it if the answer is yes, you're at l possible people. And otherwise, you're at n minus l. Which branch are we going to go down? Well, we don't know. It could be either one. ha. But do, can we put a probability on it? Yes if it covers everything. It's about a half and a half. You said it was uniform, right? Uniformly chosen? The, the right answer, the, the target person is uniformly chosen. But this question isn't, isn't necessarily of that form, right? This, this question could be very skewed, it might only have one thing on the left. And n minus 1 on the right like you said. Right, well then I, that's not a very good question to ask. No, that's right. But I wanted, I want to be able to distinguish good from bad questions by saying we want questions that, that, that are going to leave us with the smallest expected size set. Oh, well then since all possibilities are there, the smallest the, the best you can hope for is about a half. Yeah, and in particular, eh, with this, this particular question's going to do, is it's going to have a probability of n over l of going down the left branch and getting an l. And it's going to have a probability of n minus l over n of going down the right branch and getting n minus l. And so that could be one way of actually scoring this, this quesiton The question that has the best score, as you pointed out, is going to be one where l is about the same as n minus l, in fact, half would be a really good choice. Right, I thought that's what I said. Yeah. Oh, okay, good. But, but I just wanted to write down the formula. Okay, that makes sense, that makes sense. Okay, no, it's always good to write down the formula. 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 you'll start out with n, then n over 2, then n over 4, then n over 8, then n over, and eventually you'll get to n over n which will give 1. So you'll only have one, possible answer. And so that'll take you a logarithm amount of time. Right, 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 hypothesis, 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. Right.

1.5.10 Teacher With Constrained Queries

All right. Now, you might get a misleading impression from what I just presented there, because I did this sort of odd trick that's not really very kosher. Specifically, 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 [INAUDIBLE] and literals or their negation. So here's a, 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. All right, so Charles, let's make sure that that you understand this. So let's let's look at, some inputs. Alright. So let's say our input is this. X1 is 0. x2 is 1. x3 is 0. x4 is 1. x5 is 1. What is this formula going to evaluate in this case? Okay so, a 0 is false and 1 is true I assume, because, that's what we do in these things. X2 and x4 don't matter. I just have to look at x1, x3, and x5. So, x1 had better be true, and it isn't, so I already know the

answer. It's false. Very good. All right, let's let's try a couple more of these. Okay. So, I would do the same thing I did before. I would look at x1, x3, and x5. So in this case, x1, cleverly, I've noticed this 1 so, that doesn't give me the answer right away because you're an evil teacher. x3 is also 1 but x5 is 1, and according to the hypothesis it has to be 0 for this to be true, so 0. I can do the same kind of thing for the third line, so x1 is true like it needs to be, but x3 is false instead of being true so also 0. And then in the bottom, let's see. X1 is true, which it needs to be. X3 is true, which it needs to be, and X5 is false, which it needs to be, so the output is 1. Very good. All right. I got it.

1.5.11 Reconstructing Hypothesis Question

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.

1.5.12 Reconstructing Hypothesis Solution

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. Is the answer you got it. Huh I got it right. So the thing to notice is that 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.

1.5.13 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 1. But it's still linear, it's still linear 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 1 0. And you're going to tell me that 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. Yes, is there a happy way of saying it? There isn't a happy way of saying that. But there is, there are other questions that have happier answers. Okay, like what?

1.5.14 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 let's 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 in the formula that every variable is present. In, in both it's positive and negated for. 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.

1.5.15 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 is that training set need to be for the learner to be able to create successful hypothesis. 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

1.5.16 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 like that, 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. All right. Good. So I think what we'll do next is a quiz just to make sure that we kind of get this concept and how it works. And we'll do that next time. Okay sure. I like quizzes.

1.5.17 Terminology Question

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 in return true, ignore the inputs in 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.

1.5.18 Terminology Solution

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.

1.5.19 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.

1.5.20 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 fine 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 appoximatly correct. 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 let's be approxiiamtly correct. But we can't be approximately correct all the time, so let's only probably be approximately correct. So using your, your, 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.

1.5.21 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 1 minus Delta, ouput a hypothesis h, little h, 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 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. .

1.5.22 PAC Learnable Question

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.

1.5.23 PAC Learnable Solution

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 algarythm 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.

1.5.24 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. [LAUGH]

1.5.25 Epsilon Exhausted Quiz Question

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 distribtiuon 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.

1.5.26 Epsilon Exhausted Quiz Solution

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 [CROSSTALK] 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 x or. Since we know x or 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 0.5 to, to one, but this is the, this is the value that I was really hoping you'd be able to reason out. Okay, well that all made sense. Good, nice work. Thanks.

1.5.27 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? 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.

1.5.28 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 law 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 long is the 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 convienent 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 virgin 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 aught 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!

1.5.29 PAC Learnable Example Question

This puts us in a good position to be able to go back to that question we looked at before, so let's 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 corrisponding to returning each of those 10 bits, seperate hypthesis, one returns the first bit, one returns the second bit. And so on, and now we have a target of Epsilon equals point 1, so we would like to, return a hypothesis whose error is less than or equal to point 1, and we want to be pretty sure of it, the error, or failure probability needs to be less than or equal to point 2, and let's just say for concreteness, that our distribution of our imput is uniform. Ok. 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.

1.5.30 PAC Learnable Example Solution

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.

1.5.31 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 learnerversus 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. [LAUGH] 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. [LAUGH] 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.

1.6 VC Dimensions

1.6.1 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, it's complicated, but it's really interesting. So let's, 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. Alright well let's do a quiz.

1.6.2 Which Hypothesis Spaces Are Infinite Question

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.

1.6.3 Which Hypothesis Spaces Are Infinite Solution

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 theres not just an infinite number of those there is alif one 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. O, 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 ans 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 theres 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 hypothese 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. But it, but it, you know? Is 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 neuro-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 neuro-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.

1.6.4 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, 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 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 spec-, 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, 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.

1.6.5 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.

1.6.6 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 infiniite. 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.

1.6.7 Internal Training Question

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, the largest, 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.

1.6.8 Internal Training Solution

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, yeah like that. 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 possiblities 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 chatter, 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 cause 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. Hm. I am the [INAUDIBLE] powerful.

1.6.9 Linear Separators Question

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. Go.

1.6.10 Linear Separators Solution

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 and 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. [LAUGH] [SOUND] 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 3 points on the number line and there's just 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. [LAUGH] 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's 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 [CROSSTALK] 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 1s that are connected by the minus, they cross each other, 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 seperator. 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 argued 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 let's see, 2D space or 3D space or 4D space or 5D space? is the VC dimension still three or does it keep getting bigger?

1.6.11 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, lineal 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.

1.6.12 Polygons Question

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 holygo-, 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]. [LAUGH] Okay, so let's go to a quiz. Go.

1.6.13 Polygons Solution

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, let's 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, ant 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 dimention 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?

1.6.14 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.

1.6.15 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 hyptothesis 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.

1.6.16 Summary

All right, so that gets us to the end of talking about VC dimension. So Charles, what did we learn? Well, we learned about VC dimension. [LAUGH] I think that was probably the key thing. Indeed, which, which was capturing this notion of shattering. Right. We learned that VC dimension, the relationship between VC dimensions, VC dimension and parameters. Yeah. Very good, and in fact so like we'd even be able to have a guess to say, well what if you have a neural network and you're thinking of adding additional nodes to the hidden layer. What do we suppose that would do to the VC dimension of what you can represent? So, we didn't really talk about it, but it does occur to me when you put it that way that it's pretty subtle, right? Because it's the, it's not just that it's related to the number of parameters. It's related to the true number of parameters. Because you can always come up with inefficient ways to represent your parameters. So, for example, if you have a real number I could represent that as two parameters. Everything to the left of the decimal point and everything to the right of the decimal point, but that doesn't make it really two separate parameters. It's still just one parameter. All right, that's fair. We also saw how VC relates to the size of the hypothesis space for finite hypothesis spaces. And I guess we learned how sample complexity relates to VC dimension, and in fact, all these things are themselves related. And we learned a little bit, or some tricks, or at least went through some examples of how to actually compute the VC dimension. In particular, we learned that you need to give an example to find the lower bound, and you need to prove an upper bound, Good. And one other thing that I'd want to add to that, is that VC dimension captures this notion of PAC Learnability. Cool. I think that's enough for one session. All right. See you next time, Michael. Bye. All right. See you next time.

1.7 Bayesian Learning

1.7.1 Intro

HI Michael. Hey how's it going? So I want to talk about something today Michael. I want to talk about Bayesian Learning, and I've been inspired by our last discussion on learning theory to think a little bit more about what it is exactly that we're trying to do. I'm in the mood beyond specific algorithms to just think more generally The sort that learning people want us to do, learning theory people want us to do and I think Bayesian Learning is a nice place to start. Sound fair? Yeah, that sounds really cool, I think that might be a nice formal framework for thinking about some of these problems. Good. Good. So, I'm going to start out. By making a few assertions, which I hope you will agree with, and if you agree with this then we'll be able to kind of move forward and ask some pretty cool questions okay? So Bayesian learning, so the kind of idea here behind Bayesian learning is this sort of fundamental Underlying assumption about what we're trying to do with the machine learning. So, I've written it down here, here's what I'm going to claim we're trying to do. We are trying to learn the best hypothesis we can given some data and some domain knowledge. Do you buy that as an assertion? Yeah, it's, and pretty much everything we've talked about so far has had a form kind of like that. We're searching through a hypothesis base and As you've pointed out on multiple occasions there's this kind of extra domain knowledge that comes into play for example when you pick a like a similarity metric for something like k nearest neighbors Right and of course we always have the data because we're machine learning people and we always have data. So this is what we've been trying to do and I'm going to suggest that we can be a little bit more precise about what we mean by best and I'm going to try to do that and see if you agree with me. Okay, so I'm going to rewrite what I've written already except I'm replacing best with most probable. Okay. So what I'm going to claim is that what we've really been trying to do with all these algorithms we're doing is we're trying to learn the most likely or the most probable hypothesis given the data and whatever domain knowledge we bring to bear. You buy that? Interesting. I'm not sure yet. I mean, so is it the hypothesis that it's most likely to be returned by the algorithm? No, it's the hypothesis that we think is most likely, given the data that we've seen. Given the training set and given whatever domain knowledge that we bring to bear on the problem,

the best hypothesis is the one that is most likely, that is Most probable. Or most l, probably the correct one. Interesting. Well, are we going to be able to connect that to what we were talking before? Which is generally we were selecting hypotheses based on things like their error. Yes. Exactly. We are going to be able to connect that. We are definitly going to be able to connect that. But. Okay. I can;t go forward unless I can convince you that it's reasonable to at least start out thinking about best being the same thing as most probable. Yeah, I'm willing to go forward with this. It sounds interesting. So if you're willing to move forward with this, then I want to write one more thing down and then we can sort of dive into it. So if you buy that we're trying to learn the most probable hypothesis, the most likely one, the one that has the highest chance of being correct given the data, and our domain knowledge, then we can write that down in math speak pretty simply. It's the probability of, some particular hypothesis h, drawn from some hypothesis class. Given some amount of data which I'm just going to refer to as D from now on. Okay? And that's just, exactly what we said just above when we talk about the most probable age, given the data. Okay? Well wait. Two things. One is so D is not distribution which we've had in the past. That's true. So I guess as long as we keep that straight. And the other one is No that's, you're just telling me the probability of some particular hypothesis h. That's right. So, we want to somehow, given this quantity we want to find, the best or the, most likely, of the hypothesis given this. Does that make sense? Yes. So we want to find the argmax, of h, drawn from Your hypothesis class. That is we want to find the hypothesis drawn from the hypothesis class that has the highest probability given the data. Perfect. Okay, good. So we're going to spend the next 45 hours. [LAUGH] Exploring this expression. Okay so that's like what, like 6 hours per letter. [LAUGH] Yeah and that's fine because its important.

1.7.2 Bayes Rule

Alright Michael. So like I said, we're going to spend all this time trying to, to unpack this particular equation. And the first thing we need to do is we need to come up with another form of it that we might have some chance of actually understanding of actually getting through. So I want to use something called Bayes' rule. Do you remember Bayes' rule? I do. Okay, what's Bayes' Rule? The man with the Bayes makes the rule. Oh wait, no, that's the golden rule. That's right, no. The Bayes Rule, is, it relates, it, I don't know. I think of it as just letting you switch which thing is on which side of the bar. Okay, so. Do you want me to give the whole expression? Yeah, give me the whole expression. So if we're going to apply Bayes' Rule to the probability of h given D. We can move, turn it around and make it equal to the probably of D given H. And it would be great if we could just stop with that, but we can't. We have to now kind of put them in the same space. So, we multiply by the probability of H, and then we divide by the probability of D. And sometimes that's just a normalization and we don't have to worry about it too much. But that's, that's the bay, that's Bayes' rule right there. So this is Bayes' rule. And it actually is really easy to derive. It falls it follows directly from the chain rule in probability theory. Do you think it's worthwhile? Showing people that or just they should just accept it. Well, I mean, you could just, you might be able to just see it. Just, the, the thing on top of the, the normalization, the probability of D given h times probability of h. That's actually the probability of D and h together. Right. So the probability of h times the probability of d over h as you say also the chain rule basically the definition of conditional probability in conjunctions and if you move the probability of d over to the left hand side you can see we're really just saying the same thing two different ways. It's just the probability of h and d. So then we're done. No, that's right. So I can write down what you just said. And use different letters just to make it more confusing, so Oh good. You can point out that the probability of A and B, by the chain rule, is just the probability of A given B, times the probability of B. But because order doesn't matter, it's also the case that the probability of A and B. Is the probability of b given a times the probability of a. And that's just the chain rule. And so if these two quantities equal to one another's exactly what you say, I could say well, the probability of a given b is just the probability of b given a times the probability a divided by the probability of b. And that's exactly what we have over here. Good. So now that we've mastered that all your Bayes are belong to us. [LAUGH] How long have you been saying that? The...just, only about 3 or 4 minutes. [LAUGH] Fair enough. Okay, so we have Bayes's rule. And what's really nice about Bayes's rule is that while it's a very simple thing, it's also true. It follows directly from probability theory. But more importantly for machine learning, it gives us a handle to talk about. What it is we're exactly trying to do when we say we're trying to find the most probable hypothesis, given the data. So let's just take a moment to think about what all these terms mean. We know what this term here means. The, it's just the probability of some hypothesis given the data. But what do all these other terms mean? I want to start with this term, the probability of the data. It's really nothing more than your prior belief of seeing some particular set of data. Now, and as you point out, Michael, often it just ends up to be a normalizing term and typically does not matter, though we'll see a couple of cases where it

does matter, helps us to, to sort of think about a few things. But generally speaking, whatever it is Since the only thing that we care about is the hypothesis, we're trying to find that, the probability of the data doesn't depend on the hypothesis, so typically we ignore it, but it's nice to just be clear about what it means. The other terms are a bit more interesting. They matter a little bit more. This term here, the probability is the probability of the data given the hypothesis right? Mm. Seems like learning backwards. It does seem like learning backwards but what's really nice about this quantity is that unlike the other quantity, the probability of the hypothesis given the data, it's actually, turns out to be pretty easy to think about the likelihood that we would see some data given that we were in a world where some hypothesis, h, is true. So there is a little bit of subtlety there and I, let me, let me unpack that subtlety a little bit. So we've been talking about the data if its sort of a thing that is floating out in air, but we know that the data is actually our training data. And it's a set of inputs and let's just say for the sake of argument we are going to do classification learning, it's a set of labels that are associated with those inputs. So just to drive the point home, I'm going to call those d's, little d's. And so our data is made up of a bunch of these training examples. And these training examples are whatever input that we get coming from a teacher, coming from ourselves, coming from nature, coming from somewhere and the associated label that goes along with them. So when you talk about the probability of the data given the hypothesis, what you're talking about, well, what's the likelihood that. Given that I've got all of these Xis and given that I'm living in a world where this particular hypothesis that I would see these particular labels. Does that make sense Michael? I see. Yeah, so, so I can imagine a more complicated kind of notation where, we're, we're kind of accepting the Xs as given. But the labels is what we are actually saying is something that we want to assigned probability to. Right so its not really that the x's matter in the sense that we are trying to understand those. What really mattes re the labels that are associated with them. And we will see an example of that in a moment. But I wanted to make sure that you get this subtled. So in a sense then I guess you're saying that the probability of D given H component, or, or quantity, is really like running the hypothesis. It's like, It's like labeling the data. Okay Michael, just to make sure we get this. Let's imagine we're in a universe, where the following hypothesis is true. It returns true, in exactly the cases where some input number X, is greater than or equal to 10 And it returns false otherwise. Okay? Yup. Okay. So here's a question for you. Let's say that our data was made up of exactly one point. And that value set x equal to 7. Okay? What is the probability that the label associated with 7. Would be true. Huh. So you're saying we're in a world where h is holding and that the h, h is being used to generate labels. So it wouldn't do that right? So, the probability ought to be zero. That's exactly right and what's the probability that it would be false? 1 minus 0 [LAUGH] which we'll call 1. Which we'll call 1. That's exactly right. So it's, it's just that simple. That, the probability of the data given the hypothesis, is really about, given a set of x's, what's the probability that I would see some particular label. Now, what's nice about that is, is, as you point out, is that, it's as if we're running the hypothesis. Well, given a hypothesis, it's really easy, or at least it's easier usually, to compute the probability of us seeing some labels. So, this quantity is a lot easier to figure out than the original quantity that we're looking for. The probability of the hypothesis, given the data. Yeah, I could see that. It's sort of reminding me a little bit of the Version Space, but I can't quite crystallize what the connection is. Well that's, it's good you bring that up. Because I, I think in a couple of seconds I'll give you an example that might really help you to see that. Okay? Okay.

1.7.3 Bayes Rule p2

So, let's look at the last quantity that we haven't talked about so far. And that is the probability of the hypothesis. Well, just like the probability of D is the prior on the data, this is in fact your prior on the hypothesis. So, just like the probability of D is a prior on the data. The probability of H is a prior on a particular hypothesis drawn from the hypothesis space. So in other words, in encapsulates our prior belief that one hypothesis is likely or unlikely compared to other hypotheses. So in fact what's really neat about this from a sort of AI point of view is that the prior ,as its called, is in fact our domain knowledge. So if every angle that we've seen so far, everything that we've said there's always some place where we stick in out domain knowledge. Are prior belief about the way the world works. Whether that's a similarity metric for Knn It, it's something about which features might be important, so we care about high information gain and decision trees, or our belief about the, the structure of a neural network. Those are prior beliefs, those are, that represents the main knowledge. And here in Bayesian Learning, here in this notion of, of Bayes' Rule, all of our prior knowledge sits here in the probability or prior probability over the hypotheses. Does that all make sense? Yeah its really interesting I guess. So we talked about things like kernels and similarity functions as ways of capturing this kind of domain knowledge. And I guess, I guess what its saying is that its maybe tending to prefer or assign higher probability to hypothesis that group things a certain way. exactly right. So, in fact, when you use something like Euclidian distance in KNN, what you're

saying is,'Well, points that are closer together ought to have, similar labels, and so, we would believe any hypothesis that puts points that are physically close to one another to have similar outputs, we would say, are more likely than ones that put points that are very close together to have different outputs. Neat. So let me just mention one last thing before I give you a quiz, okay? So, see if this makes sense, I'm a see if you really understand Bayes' rule. So let's imagine that I wanted to know under what circumstances the, probability of a hypothesis, given the data, goes up. What on the right side of the equation would you expect to change, go up or go down, or stay the same, that would influence whether the probability of a hypothesis goes up. So the probability of the hypothesis given the data, what could make that combined quantity go up, so one is looking at the right hand side, the probability of the hypothesis, so, so if you have a hypothesis that has a higher prior, has, is more likely to be a good one. Before you see the data then that would raise it after you see the data too. Right. And I guess the probability of the data given the hypothesis should go up. Oh, which is kind of like accuracy. It's kind of like saying that if you pick a hypothesis that does a better job of labeling the data, then also your probability of the hypothesis will go up. Right. Anything else? I guess the probability of the data going down. But that's not really a change from the hypothesis. Right. But it is true that if those goes down, then the probability in the hypothesis can and the data will go up. But as you point out, it's not connected to the hypothesis directly. And I'll write in equation for you in, in just a moment that'll kind of make that, I think, a little bit clearer. Okay, but you got all this, right? So I think you understand it. So we got Bayes' Rule. And, notice what we've done. We've gone from this sort of general notion of saying we need to find the best hypothesis, to actually coming up with an equation, that sort of makes explicit what we mean by that. That what we care about is the probability of some hypothesis given the data. That's what we mean by best. And that, that can be further thought as, the probablity of us seeing, some labels on some data, given hypothesis. Times the probability of the hypothesis, even without any data whatsoever, normalized by the probability of the data. So let's play around with Bayes' rules a little bit and make certain that we all, we all kind of get it. Okay? Sure. Okay.

1.7.4 Bayes Rule Quiz Question

Okay Mike, are you ready for a quiz? Uh-huh. Okay, so, here, let me, let me set up the, the situation for you. So a man goes to see his doctor, okay, because his back hurts or something. Aww. And she gives him a, I know, it's really sad. It's his, the left side of his lower back, he's been playing too much racquetball. Anyway, so a man goes to see a doctor, and she gives him a lab test. Now this test is pretty good, okay? It returns a correct positive. That is, if you have the thing that this lab test is testing for, it will say you have it 98 percent of the time, okay? So it only gives you a false positive two percent of the time. And at the same time, it will return a correct negative, that is if you don't have what the lab test is testing for, it will say you don't have it. 97% of the time, so it has a false negative rate of only 3%. Wait, hang on. So, just, what's his problem? Oh, that's the question. So, the test looks for a disease. So, give me a disease. Spleen? Okay, I like that. So the test looks for spleentitis. Now spleentitis is such a rare disease that nobody's ever heard of it, And it turns out that it's so rare that only about this fraction of the population has it. Okay? Mm-hm. That make sense? So we're looking for spleentitis. It's a very rare disease, but this test is really good at determining whether you have it or determining whether you don't have it Can I tell you that, its, spleentitis appeared zero times in google. [LAUGH] So it really is quite rare. It really is quite rare. But what does google know? OK, so you got it all Michael? Yeah. So its a really rare disease and we have a very accurate test for it. Good. Man goes to see the doctor. She gives him a lab test. Its a pretty good lab test. Its checking for spleentitis, relatively rare disease and the test comes back positive. Oh. Yes. So, test is positive. So, here is the quiz question. Should we be net, notifying his next of kin? Yes. Does he have spleentitis? You said, just said he had spleentitis. No, I said the test says he had spleentitis. Or the test looks for spleentitis, and the test came back positive. So, does he have spleentitis? Yes or no? Alright, before I try to answer that can I just, ask for clarification, can I get a clarification? Please. So the 98 is a percentage and the 97 is a percentage, is .008 also a percentage? No it's not. So if I wanted to convert it to a percentage it would be .8%. Got it. Alright, now I think I have, what I need. Okay, alright, so, you think about it. Go.

1.7.5 Bayes Rule Quiz Solution

Okay Michael, what's the answer? Does he have spleentitis? Yes, does he have spleentitis? I don't think we know, for sure. Mm? What do mean by that? Well, I mean. It's a noisy and probabilistic world right. So the test told us that things look like he has spleentitis and the test is usually right. But the test is sometimes wrong and it can give the wrong answer and that's really all we know, so we can't be sure. Okay but if

you had to pick one. If you had to yes or no, like our students they did when they took the quiz. Which one would you pick? Yes or no. So, I guess C the pants. I would just say, yes because the test says, yes but if I guess I was trying to be more precise, I may go through and work out the probability and I guess if it's more likely to have than not to have, then I'd say and otherwise I'd say, no. Okay. So how would you go about doing that? Walk me through it. Based on the name of the quiz, I think I'd go with Bayes' Rule. Okay. So [LAUGH] I like that. So Bayes' Rule, is everyone recall, is the probability of the hypothesis given the data is equal to the probability of the data given the hypothesis times the probability of the hypothesis divided by the probability of the data. So, [LAUGH] Let's write all that out. So what is the probability of spleentitis, which I'm just going to write as an s. Given. We're making jokes about spleentitis, but we don't want that to be confused with splenitis, which is a real thing and probably not very pleasant. So apologies to anyone out there with splenitis. But this is spleentitis, which is really totally different. Is splenitis a real thing? Yeah. :Really what is it? Enlargement and inflammation in the spleen and the spleen as a result of infection or possibly a parasite infestation or cysts. So what you're saying is that's gross and we don't want to think about it. OK good so Woo okay, so the probability of getting splentitis and again is a very rare thing and probably isn't even real. Totally, its totally different, its definitely not real Yea definitely not. Given that we gotten a positive result and you say that we should use Baye's rules so that would be in this case what? So it's the same as the probability of the positive result given that you have spleentitis. Mm-hm. Times the probability, the prior probability of having spleentitis. Mm-hm. And I want to say normalize, but like divided by the probability of a positive test result. And what would be, the probabili. The other option is that you don't have spleentitis. Mm-hm. Even though you got a positive result. And that would be equal to? The probability of a positive result given you don't have spleentitis. Mm-hm. Times the prior probability of not having spleentitis. huh. Divided by the, again the same thing. The probability of the test results. So that's, those two things added together, needed to be one. Right. But as you point out. If we just want to figure out which one is bigger than the other. We don't actually have to know this. Hm, good point. So we can ignore it, okay. Okay, so, let's compute this. So, what is in fact, the probability of me getting a plus, given that I have spleenitis? Right. So it says in the setup, the test results correct positive 98% of the time. So, I, I think that's what it means. It means that if you really do have it, it's going to say that you have it with that probability. Okay, so That's just point nine eight. OK? And that's times the prior probability of having spleentitis which is? .008. Right. .008. And what's that equal to? It is equal to. 0.0078. 0.00784. Okay, fine. We can do the same thing over here. So what's the probability of getting a positive if you don't have spleentitis So, the probability of a correct negative is 97%. That means if you really don't have it, it's going to say you don't have it, so probability of positive result given that you don't have it, that should be the 3%. That's exactly right. Times the prior probability of not having spleentitis which is? .992. 1- .008. That's right, and that is equal to? .02976 So, which number is bigger? The one that has the larger significant digit. Which one of those two is that? I mean, obviously, the one that's bigger is the, you don't have it. That's right. So the answer would be no. And in fact the probability is almost 80%. Yeah. Which is crazy. So, it's like, you go into the doctor, you've run a test, the doctor says congratulations, you don't have speentitis, because the test says you do. That's right. [LAUGH] So, what does that tell you? That seems stupid. That does seem stupid, but what does it tell you About Bayes' Rule. What is Bayes' Rule capture. What is thing that make the answer no, despite the fact, you have a high reliability test that says yes. I. Okay. So I guess, I guess the way to think about it is, a random person showing up in the doctors office, is very unlikely to have this particular disease. And even the tiny, little, small percentage probability that the test would give a wrong answer is completely swamped by the fact that you probably don't have the disease. But I guess this isn't really factoring in the idea that, you know, presumably this lab test was run for some other reason. There was some other evidence that there was concern. Or the doctor just really wanted some more money, because She needs a new boat. Yeah, I know a lot of doctors. I do too. And most of them don't work like that. Yeah most, well most of them have PhD's not MD's. So, another way of summarizing what you just said Michael, I think, is that priors matter. I want to say the thing that I got out of this is tests don't matter. Well, tests matter. Like what's the purpose of running a test if if it's going to come back and say. Well it used to be that I was pretty sure you didn't have it and now I am still pretty sure you don't have it. Well the point of running a test is you run a test when you have a reason to believe that the test might be useful. So what is the one thing, if I could only change one thing without getting completely ridiculous, whats the easy well, I don't know whats easy, whats the easiest thing for me to change about this setup. I have three numbers here. This one, this one and this one. What would be the easiest number to change? Well, in some sense none of the seem that easy to change but I guess maybe what you're trying to get me to say is that if we look at a different population of people then we can change that .008 number to something else, like if we only give the test to people who have other signs of spleentitis. Then then it, it would probably be a much bigger number. Right, so changing the test, making the test better might be hard, presumably

you know, billion of dollars of research have gone into that, but if you don't give the test to people who you don't have any reason to believe have Spleentitis, just walking off the street, as you put it, a random person walking off the street, then you can change the priors, so some other evidence. That you might have splentitis would lead the prior to change, and then the test would suddenly be useful. So this, by the way, is an argument for why you don't want to just require that everyone take tests for certain things. Because if the prior probability is low, then the test isn't very useful. On the other hand, as soon as you have any reason to believe We have strong evidence that someone might have some condition, then it makes sense to test them for it. So it's like a stop and frisk situation. It's exactly like a stop and frisk situation. I'm looking at you [INAUDIBLE]. Okay But in some sense, you're use of the word prior is a little confusing there. So it's not that we're changing the prior, it's that we're...we have some additional evidence that we can factor in. And I guess we can imagine that that's part of the prior, but it seems like it's posterior. Yeah, it does. And it, but... One way to think about it, you actually, I think you just captured it in what you just said, right? Which is you can think of as a prior. Well, a prior to what? So it's your prior belief over a set of hypotheses, given the world you happen to be in. If you're in a world where random people walk in to take a test for splentitis, then there's a low prior probability that they have it. If you're in a world where the only people who come in are people who are from a population where the prior probability is significantly higher, then you would have a different prior. It's really a question about where you are in the process when you actually formulate your question. So would it be worth asking people how, how likely would it have to be that you have spleentitis to make this test at all useful? Right, that would change a positive, a positive result would actually change your mind about whether someone has it. yeah, actually that, I think that's something that I, I'll leave for the for the, for the interested reader, where would that prior probability have to change so that getting a positive result, I would be more likely to believe that you actually have it than not. That does bring up a philosophical question, though, which is So what, just because the priors have changed, doesn't mean that suddenly the test is useful, or that the test is going to give you an answer that somehow distinguishes and is this positive. And from a mathematical point of view, the question of whether this number is 0.008 or, or 0.8, you know, 8 10ths of a percent, where does it change? Does it change at 5%? Or does it change at 50%? Or does it change at 500%? It probably changes at 500%. You know, what, where is the place in which suddenly a positive result would make you believe they actually had spleentitis or whatever disease you're looking for. Okay? Okay.

1.7.6 Bayesian Learning

Okay, Michael, so we've gotten through that quiz and you see that Bayes' rule actually gives you some information. It actually helps you make a decision. So I'm going to suggest that, that whole exercise we went through was actually our way of walking through an algorithm. So here's a particular algorithm that follows from what we just did. And let me just write that down for you. All right, so here's the algorithm, Michael, so it's very simple. For each H in H, that is, each candidate hypothesis in our in our hypothesis space, simply calculate the probability of that hypothesis given the data W which we know is equal to the probability of the data given that hypothesis times the prior probability of the hypothesis, divided by the probability of the data. And then simply output whichever hypothesis has maximum probability. Does that make sense? Yeah. Okay, so I do want to point out that since all we care about is computing the argmax, as before, we don't actually ever have to compute that little bit so, and that's a good thing because we don't always know what the prior probability on the data is, so we can ignore it for the purposes of finding the maximal hypothesis. So the place you removed it from, it seems like that's not actually valid, because it's not the case that the probability of h given d equals, it's the probability of d given h times the probability of h. It just means that we don't care what the probability is when we go to compute the argmax. That's right, so, in fact, it's probably better to say that I'm going to approximate the probability hypothesis given the data by just calculating the probability of the data given the hypothesis times the probability of the hypothesis and just go ahead and ignore the denominator. Precissely because it doesn't change hte maximal age. Yeah, so it's, it's nice that that goes away. Right, because it's hard to know, often what the prior, what the prior probability over the data is. It would be nice if we didn't have to worry about the other one, either. Which other one? The probability of h, where's that coming from? right, so where does that come from? So that's a deep philosophical question. Sometimes it's just something you believe, and you can write down. And sometimes it's a little harder. And that's actually good that you bring that up. When we compute, our probabilities this way so it's actually got a name, it's the MAP or the maximum a posteriori hypothesis and that makes sense, it's the biggest posterior given all of your priors. But you're right Michael that often it's just as hard to say anything particular about your prior over the hypothesis as it is to say something about your prior of the data and, so it is very common to drop that. And, in dropping that, we're actually

computing the argmax over the probability of the data given the hypothesis. And, that is known as the maximum likelihood hypothesis. I guess you can't call it the maximum A priori hypothesis, because then it would also be MAP. Exactly, although I've never thought about that before. By the way, just just to be clear, we're not really dropping this, in this case, what we really said, is that, our prior belief is that all hypotheses are equally likely. So we have a uniform prior that is, the probability of any given hypothesis is exactly the same as the probability as any other given hypothesis. I see, so you're saying if, if we assume that they all are equally likely, then, the choice of hypothesis doesn't change that term at all, the p of h term, so it really is equivalent to just ignoring it. Exactly, in some constant, we don't even have to know what the constant is. But whatever it is, it's the same everywhere and therefor it doesn't affect the other terms or, in particular, affect the argmax computation. So that's actually pretty cool right? Once you think about what we just did. We just took something that was very hard. Computing the probability of a hypothesis given the data and turned it into something much easier that is... Computing the probability of you seeing the data labels given a particular hypothesis and it turns out that those are effectively the same thing if you don't have a strong prior. So that's really cool, so we're done right? We now know how to find the best hypothesis You're just finding the most likely hypothesis or the most probable one and that turns out to be the same thing as just simply finding the hypothesis that best matches the data. We're done its all, its easy. Everythings good. So,the math seems very nice and pretty and easy but is isn't it hiding a lot of work to actually do these computations? Well, sure well well look you know how to do multiplication that's pretty easy right? [LAUGH]. So I guess the only hard part is we have to look at every single hypothesis. Yeah, that's just a slight, little, you know, issue. So, mathematically meaningful, but computationally questionable. Hm. So, the big point there, is that it's not practical. Well, unless the number of hypotheses is really, really small. But as we know, a lot of the hypotheses spaces that we care about, like, for example, linear separators, are actually infinite. And so it's going to be very difficult to use this algorithm directly. But despite all that, I think that there's still something important that we get out of thinking about it this way in just the same way that we get something important out of thinking about vc dimension. Even if we're not entirely sure how to compute it in some particular case. This really gives us a gold standard, right? We have an algorithm, at least a conceptual algorithm, that tells us what the right thing to do would be if we're capable of computing it directly. So, that's good because we can maybe prove things about this and compare results that we get from some Real live algorithms to what we might expect to get but also it turns out it's pretty cute because it helps us to say other things about what it is we actually expect to learn. And I'm going to give you a couple examples of those just to sort of prove my point, sound good? Yeah. Okay.

1.7.7 Bayesian Learning in Action

Okay Michael, so let's see if we can actually use this as a way of deriving something maybe that we already knew. So I'm going to go through a couple of these because I I actually think, well, frankly I just think it's kind of cool. But, I I'm hoping I can convince you it's sort of cool too and that we get something out of it. Okay, so let me set up the word, I'm going to set up a a problem, and it's going to be a kind of generic problem, and I'm going to see what we can get out of it, okay? So this is machine learning, so we're going to be given a bunch of data, so there are three assumptions that I'm going to make here. The first is that we're going to be given a bunch of labeled training data, which I'm writing here as x sub i and d sub i, so x sub i is whatever the input space is, and d sub i are these labels. And let's say, it doesn't actually even matter what the labels are, but let's say that the labels are classification labels. Okay? Hm. All right. And furthermore, not only we're given this data as examples drawn from some underlying concept c, but they're, in fact, noise-free. Okay? So they're true examples that tell you what c is. Okay? Mm-hm. So I'm going to say, in fact, let me write that down because I think it's important. They're noise-free examples. Okay. Like di equals c of xi. That's right, for all xi. So, the second assumption, is that the true concept c, is actually in our hypothesis space, whatever that hypothesis space is. And finally, we have no reason to believe that any particular hypothesis in our hypothesis space is more likely than any other. And so, we have a uniform prior over our hypotheses. So it's like the one thing we know is that we don't know anything. That's right. So, sometimes people called this an uninformative prior because you don't know anything. Except of course I've always thought that's a terrible name because its a completely informative prior. In fact its equally as informative as every other prior in that it tells you something that all hypothesis are equally likely. But that's I thought it was called an uninformed prior. Is it? So its just an ignorant prior is what you're telling me. Yeah. Okay. Well, then maybe that's the problem. I just always had a problem with it because people keep calling it uninformative and the really mean uninformed. Okay. In any case, so these are our, these are our assumptions. We've got a bunch of data, it's noise free, the concept is actually in the hypothesis base we care about and we have a uniform prior. So we need to compute the

best hypothesis. So given that we want to somehow compute the probability of some hypothesis given the data, right? That's just Bay's Rule. So, Michael, you've got the problem right? Yes. [LAUGH] okay. So in order to compute the probability of a hypothesis given the data, we just need to figure out all of these other terms. So let me just write down some of the terms and you can tell me what a you think the answer. Okay. Well, what was the question? The question is, while we want to compute some kind of expression for the probability of a hypothesis given the data. So given some particular hypothesis, I want to know what's the probability of that hypothesis given the data, okay? Yeah. Okay, you got the setup. So, we're going to compute that by figuring out these three terms over here. So, let's just pick, one of them to do. Let's try the prior probability. So Michael, what's the prior probability on H? Did we say that it was a finite hypothesis class? It is a finite hypothesis class. Then it's like, one over the size of that hypothesis class because it's uniform. Exactly right, uniform means Exactly that. Okay so we've got one of our terms, good job. let's pick another term. How about the probability of data given the hypothesis. What's that? The probability, so I guess noise free, and we know that it's noise free so it's always, so they're always going to be zeros and ones. Mm-hm. So, and it's going to be a question of whether or not the data is consistent with that hypothesis. Right, if the labels all match. Right. What we expect them to be if that really were the hypothesis, then we get a one, otherwise we get a zero. That's exactly right. So let me see if I can write down what I think you just said. The probability of the data, given the hypothesis, is, therefore one if it's the case, that the labels And the hypothesis agree for every single one of the training exercises. Right? Yep Is that what you said? Good. And if any of them disagree, then the probability is zero. So that's actually very important. It's important to, to understand exactly what it means for, to have the probability to get a hypothesis, as we mentioned before. That the English version of this is, what's the probability that I would see data with these labels in a universe where H is actually true. Which is different from saying that H is trure or H is false. It's really a common about the labels that you see on a data. In a universe, where H happens to be true. Okay, but you know, it's occurring to me now that you wrote that down, that we've talked about this idea before. When? Well, so, like there's a shorter way of writing that. Which is D of H equals one if H is in the version space of D. Huh, that's exactly right, that's exactly right. So, in fact, that will help us to compute the final term that we need, which is the probability of seeing the data labels. So, how do we go about computing that? Well, it's exactly going to boil down to the version space as you say, let me just write out a couple of steps so that it's pretty Kind of easy to see. It's sometimes easier in these situations to kind of break things up. So, the probability of the data sort of formally, is equal to just this. So we can write the probability of the data as being, basically, a marginalized version of the probability of the data given each of the hypotheses times the probability of the hypotheses. Now, this is only true in a world where our hypotheses are mutually exclusive. Okay so let's assume we are in that world because frankly that's what we always assume and this little trick is going to workout for us because we are going to get to take advantage of two terms that we already computed naming the probability that the data given the hypothesis and the prior probability of a particular hypothesis so we know that prior probability of a hypothesis is right, its just one over the size of the hypothesis space and how am I going to substitute in this equation for the probability of the data given the hypothesis? So, I don't know. I would write that differently. I mean, it's basically it's like the indicator function on whether or hot HI is in the version space of D. Right, that's exactly right. So in fact this is not a good way to have written it. Let's see if I can come up with a, a good notational way of doing it. Let's say, for every hypothesis that is in the version space of the hypothesis space given the labels that we've got. Okay? How's that count? Okay. So rather than having to come up with an indicator function, I'm just going to define vs as the subset of all those hypotheses that are consistent with the data. Yeah exactly Okay, and so whats the probability of those? One It's one and it's zero otherwise, so then, we can simplify the sum and it's simply what? ? The sum of the one, ooh! The one of each doesn't even depend on the hypothesis. mm-mh! I see wait I don't see oh yes I do, I do it's one over the size of version space. No its the size of the version space over the size of the hypothesis space. That's exactly right. Basically for every single hypothesis in the version space we're going to add one and how many of those are? Well the size of the version space number of those. And multiply all that by one over the size hypothesis space, and so the probability of the data is that term. So now we can just substitute all of that, into our handy dandy equation up there, and let's just do that. So the probability of the hypothesis given the data, is the probability of the data given the hypothesis Which we know is one for all those that are consistent, zero otherwise. The probability of the prior probability over the hypothesis is just one over the size of the hypothesis space, and the probability of the data is the size of the version space Over the size of the hypothesis base which, when we divide everything out, is simply this. Got it? Got it. So, what does that all say? It says that, given a bunch of data, your probability of a particular hypothesis being correct, or being the best one or the right one, is simply uniform over all of the hypotheses that are in the version space. That is, are consistent with the data that we see. Nice. It is kind of nice. And by the way, if it's not

consistent with it, then it's zero. So, this is only true for hypotheses that are still in version space and zero otherwise. Now notice that all of this sort of works out only in a world where you really do have noise free examples, and you know that the concept is actually in your hypothesis space and, just as crucially that you have a uniform prior for all the hypotheses. Now this is exactly the algorithm that we talked about before right. We talked about before what would we do. To kind of decide whether a hypothesis was good enough in this sort of noise-free world. And the answer we came up with is you should just pick one of them that's in the version space. And what this says is there's no reason to pick one over the other from the version space. They're all equally as good or rather equally as likely to be correct. Yeah, that follows. Yeah. So there you go. So it turns out you can actually do something with this. Notice by the way that we did not pick a particular hypothesis space, we did not pick a particular form of our instance space, we did not actually say anything at all about exactly what the labels were other than that they were labels of some sort. The strongest assumption that we made was a uniform prior, so this is always the right thing to do. At least in a Bayesen sense in a world where you've got noise free data, you have to find that hypothesis space, and you have uniform priors. Just pick something from the consistent set of hypotheses.

1.7.8 Noisy Data Question

Alright, Michael, I got a quiz for you, okay? Sure. So, in the last example we had noise free data. So I want to think a little bit about what happens if we have some noisy data. And so I'm going to come up with a really weird, noisy model. But hopefully it illustrates the point. Okay. Sure. Okay so i got a bunch of training data, its x of i d of i and here's how the true underline process sort of works. So give us some particular x of i, you get a label which is d of i which is equal to k times x of i where k is some number So one of the counting numbers, one, two, three, four, five, six, seven, eight, and so on and so forth. And the probability that you actually get anyone of those multiples of x of i is equal to one over two to the k. Now why did I choose one over two to the k? Because it turns out that the sum of all those two to the k's from one through infinity happens to equal to one. So it's a true probability distribution. Hmm, okay. So it's just a neat little geometric distribution. So, you under understand the setup so far? I think so, so before hypothesis were producing answers then we looked for them to be exactly in the data. Now we're saying that the hypothesis produces an answer, and it gets kind of smooshed around a little bit before it reappears in the table, thats the noisy part. Right, so you, you're not going to be in a case now, that if the hypothesis disagrees with the label it sees. That in fact that means no it can't possibly be the right hypothesis because there's some stochastic process going on that might corrupt your output label, if you want to think of it as corruption, since it's noisy. Okay? Okay, yeah sure. Alright? Okay, so here's a set of data that you got. Here's a bunch of x's that, that make up our training data one, three, 11, 12, and 20. For some reason they're in ascending order. And the labels that go along with them are five, six, 11, 36, and 100. So you'll notice that they're all multiples of some sort of the input x. Okay? Alright. Now I have a candidate hypothesis. H of x which just returns x. That's kind of neat. So it's the identity function. So, what I want you to do is to compute the probability of seeing this particular data set in a world where that hypothesis, the identity function, is in fact true. The identity function plus this noise process. Yes. And one other question quickly this, this noise process is supplied independently to each of these inputs, outputs, pairs? Yes, absolutely. Okay, then, yeah, I think I can do that. Uh-huh. Okay, go.

1.7.9 Noisy Data Solution

Okay, Michael. You got the answer? Yeah, I think, well I can work through it, I don't actually have the number yet. Okay, let's do that. So, alright, so in a world where. In a world where. Where this is the hypothesis that actually matters. We're saying that X comes in, the hypothesis spits that same X out. And then this noise process causes it to become a multiple. And the probability of a multiple is this one over two to the case. So, the probability that that would happen from this hypothesis. for the very first data item. The one to five, would be 1 32nd. That's the probability that a one would produce a five by this process. Okay. How do you, how'd you figure that out? Cause the k that we would need the multiplier would have to be five. And so the probability for that multiplier is exactly one over two to the five which is one 30 second. Okay. And so then I would use that same thought process on the next one which says that it is doubled and the way that this particular process would have produced a doubling would be if with, with probability a quarter. Uh-hm. And, the next data element would have been produced by this process with probability at half, because it's k will be 1, and 1 over 2 to the k would be half, Okay, I like this. Right? The next one will be an 8th, because its tripled, Uh-hm. And the last one is also a multiplier of 5, just like the first one, so that will be one thirty second as well, Mm-hm. Alright but now we need to assign a probability to the

whole data set, and because you told me it was okay to think about these things happening independently, the probability that all these things would happen is exactly the product. Right. So I'll multiply a 32nd and a quarter and 1/2 and an 8th and a 32nd, so that's like a factor of 5 plus 2 is 7 plus 1 is 8. Plus another 3 is 11 plus another 5 is 16 and 2^16 is 65,536. So it should be 1 over, oh you already wrote it. 65,536. Yea that. Yes that's absolutely correct Michael. Well done. Okay so, that's right, but you did it with a bunch of specific numbers. Is there a more generic Is there a general form that we could write down? Yeah, I think so, we're doing something pretty regular once I fell into a pattern. So, I took the D, and divided by X, so D over X tells me that the multiplier that was used, so that's like, the K. So. D over x gave you the k. And it was one over 2 to the that. Okay, so one over 2 to the that. And it was then the product of, of that quantity for all of the data elements, so all the i's. So product over all the i's of that. Okay. But we have to be careful because If it was the case that for any of our xi's the d wasn't a multiple of it, that can't happen under this hypothesis and the whole probability needs to go to zero. Right. So they all have to be divisible otherwise all bets are off. Okay, so in other words if d of i mod x of i is equal to zero and this formula holds and it's zero otherwise. Exactly. Okay. Sounds good. Okay, great Michael. So that's right and that was exactly the right way of thinking about it. And now, what we're going to do next, is we're going to take what we've just gone through. This sort of process of thinking about, how to generate data labels. for, you know, noisy cases and we're going to apply to it what I think you will find will be a pretty cool derivation. Sound good? Awesome! Excellent.

1.7.10 Return to Bayesian Learning

Okay Michael, so that was pretty good with the quiz. I want to do another derivation and I want you to help me with it, okay? Hm. Cool. Okay, so Michael, we have a similar setup to what we've had before. We're given a bunch of training data, XI inputs and DI outputs. But this time we're dealing with real valued functions. So the way DIs are constructed is there's some Deterministic function f, that we pass the fs through. And that gives us some value. And that's really what we're trying to figure out. What is the underlying f? But to make our job a little bit harder, we have noisy outputs. So, for every single DI that is generated, there's some small error, epsilon that is added to it. Now, this particular error term, is in fact drawn from a normal distribution with zero mean and some variance. We don't know what the variance is. It's going to turn out. It doesn't actually matter. There's some variance going on here. The important thing is that there's zero mean. So, you got it? And it's important that it's probably the same variance for all the data. That's right, in fact, each of these epsilon sub i's are drawn iid. And is that f, are we assuming it's linear? Nope, we're not assuming that it's linear. Okay. It's just some function. All right, I'm with you. Okay, so you got it? Yep. All right. So, here's my question to you. What is The maximum likelihood hypothesis. Do we know f? Can I just say f? No, we don't know f. All we see are x sub i's and d sub i's. But we know there is some underlying f. And we know that it's noisy, according to some normal distribution. I don't know how I would find that. Well let's try to walk it through. So. We know how to find the maximum likelihood hypothesis, at least we know an equation for it. The maximum likelihood hypothesis is simply the one that maximizes this expression. Right. That was when we assumed a uniform prior on the hypotheses. Exactly. And so we, this is sort of the easiest case to think about Where it turns out that finding the hypothesis that best fits the data is the same as finding a hypothesis that describes the data the best. If you make an assumption about a uniform distribution, or a uniform prior. Okay, so. This is all we have to do now is figure out what we're going to do to expand this expression. So what do you think we should do first? The probability of the data given the hypothesis. Right. So each we assumed IID. Mm-hm. You actually helpfully even wrote that down. So we can expand that into the product over all the data elements of the probablity of that data element given the hypothesis. And x. Okay, so let's do that, Michael. Let's write that out. So, finding the hypothesis that maximizes the data that we see, as you point out, is just a product over each of the independent data that we see. Or datums. So that's good. That's one nice step. So we've gone from talking about all of the data together to each of the individual training data that we see. So what do we do next? What is the probability of seeing one particular P sub i, given that we're in a world where H is true. So okay, given that H is true that means whatever the corresponding xi is, if we push that through the f function, then the di is going to be F of XI plus some error term so I guess if we took di minus F X I, that would tell us what the error term is and the we just need an expression for saying how likely it is that we get that much error. Right, so, what is the expression that tells us that? I'm guessing it's something that uses the normal distribution, it probably has an E in it. [LAUGH] I think that' s absolutely right. So, let's be particular about what you said. So, when you say that we should push it through F of X, let's be clear that that's basically what H is supposed to be. Our goal here is, given all of this training data, let's recover what the true f of x is. And that's what our H is. Each of our hypotheses a guess about what

the true underlying deterministic function F is. So, if we have some particular labels, some particular value D sub 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. So, we can actually write down Gaussian. Do you remember what the equation for a Gaussian is? Yes. It's exactly something that has an E in it. That's right. So I'll see, I'm going to start writing it and you see if you remember any of what I'm writing down. E to the... No. Okay, good. It's 1 over. E to the. No. Okay. Square root of, it's, it's coming back to you now. 2 pi sigma squared. Okay. Times... I was going to put that in after. Oh, okay. So now you get your E, so E to the what? It's going to be the value, which, in our case, is, like, H of XI minus DI. Yeah. And then I feel like, we probably square that? Yep. And then we divide by sigma squared? right. Really? Yeah. Sweet! And your missing one tiny thing. There needs to be another two. Yes. And in fact it's minus one half. Got it. So, this is exactly the form of the Gaussian in the normal distribution. And what it basically says is the probability me seeing some particular point, in this case D of I. Given that the mean is H of X. Which is to say that's the underlying the function. Is exactly this expression. e to the minus one half, of the distance from the mean, squared, divided by the merits. Okay. And that's just, you either remember that or you don't. But that's just the definition of a Gaussian. So that means the probability of us seeing the data is the product of the probability of us seeing each of the data items. And that's just the product of this expression here. Good. Now, we need to simplify this. We could stop here because this is true, but we really need to simplify this and I think it's pretty... Not to hard to do. It's pretty easy. Mm..hm. What kind of trick do you think we would do here to simplify this? So, first thing I would do is, noticed that the 1 over square root 2 pi sigma square doesn't depend on i at all, and maybe move it outside the pi but then realize, well, actually since we're doing an argmax anyway, it's not going to have any impact at all. [CROSSTALK] I would just like cross that baby out. I like that. No point in keeping it. All right, now I'm hoping that the other sigma squared we can make that go away too. So I'm tempted to just cross it out, but I'd rather, I'd be much more happy if I had a good explanation for why that's okay. Well, so what's the normal trick, so we're trying to maximize the function, right? What you just said is we can get rid of this particular constant expression because it doesn't affect the max. What's making it hard for you to get rid of the sigma squared here is that it's being passed through some exponential and you can't remember off the top of your head what clever work you can do with constants inside of exponentials. So it would be nice if we could get rid of the exponential. Very good. So because log is concave. No, because it's monotonic. um-hm. We can take the log of the whole shabang. So this is going to be equal to the argmax of the sum of the log of that expression, which is going to move the thing to the outside and the log of E, so that's going to be good, so it's going to be the sum of the superscript thing, the power. Right. So let's write that down. Okay, so just to be sure that that was clear to everybody, let's just point out that we basically took the log of both, the natural log of both sides, and so we said, instead of trying to find the maximum hypothesis or the maximum likelihood hypothesis by evaluating this expression directly, we instead evaluated the log of that expression. And as you'll recall from intermediate algebra, the log of a product is the same as the sum of the logs, and the log of E to something is just that thing. As long as we do natural log. As long as we do natural log when we have E. If we were doing something to the, 2 to the power of something, we'd want to do log base 2. Okay. Got it. And you said to do it to both sides but we really didn't need to do it to both sides we just needed to do it inside the things we taking the argmax. That's correct. Okay, so we've got here. So, is there any other simplifying that we can do. Yeah, yeah now it seems much clearer so the. The negative one half divided by sigma squared all can move outside the sum cause it doesn't depend on I at all. Right. And then the sigma squared you said that before you said that that wasn't going to turn out to matter. Both sigma squares ended up, you know, getting tossed into the rubbish heap. That's right. And I want to be careful with the negative sign. Like I feel like the half can go and the sigma square can go but the negative has to stay. You're right. The half can go. And the sigma squared can go. So that leaves us with this expression. So I've taken, gotten rid of the one half, like you suggested. Got rid of the sigma squared like you suggested, and I moved the minus sign outside of the summation. And I'm left with this expression. I have a thought about getting rid of that minus sign. Well how would you get rid of a minus sign? So the max of a negative is the min. Right, so we can get rid of the minus sign by just simply minimizing instead of maximizing that expression. We end up with this expression. Nice. That's much simpler than where we started. The e is gone. It's much simpler. We got rid of a bunch of e's. We got rid of a bunch of turns out extraneous constants. We got rid of multiplication. We did a bunch of stuff, and we ended up with this. You know, we got rid of two pis. It's kind of sad I would like some pie. Mm, I wonder what kind of pie it was? Pecan pie? [LAUGH] Okay, so we got this expression, and that's kind of nice on your own you say, but actually it's even nicer than that. What? What does this expression remind you of Michael? The Sum of Squares. This is exactly it. This is, in fact. The sum of squared error, which is awesome. Yeah, whoever decided it would be a good idea to model noise as a Gaussian was really on to

something. Mm-hm. Now, think about what this means, Michael. We just took, using Bayesian Learning, a very simple idea of maximizing a likelihood. We did nothing but substitution, here and there. With the noise model. We got rid of a bunch of things that we didn't have to get rid of. We cleverly used the natural log. Notice that the minus sign can be taken away with the min. And, we ended up with sum of squared error. Which suggests that all that stuff we were doing with back propagation. And, all these other kinds of things we're doing with receptrons is the right thing to do. Minimizing the sum of square error, which we've just been doing before. Is in fact the right thing to do according to Bayesian learning. Right in this case meaning meaning what a Bayesian would say. Meaning what a Bayesian would say which I believe is sort of right by definition. More importantly here it is. They certainly believe it. Well, they, they do frequently. Oh! I see what you did there. No one will get that but, but us. Anyway, the thing is this is the thing you should minimize if you're trying to find the maximum likelihood hypothesis. Now, I just want to say something. That is beautiful. Absolutely beautiful. That you do something very simple like finding the maximum likelihood hypothesis and you end up deriving sum of squared errors. So, just to make sure that I'm understanding. because I see some beauty here, but maybe not all of it. We didn't talk about what the hypothesis class here was. Right, so, if you don't know what the hypothesis class is... You're, you're kind of stalled at this point, but if we say the hypothesis class is say linear functions. Mm-hm. Then, what we're saying is we can do linear regression, because linear regression is exactly about minimizing the sum of the squares, right? So linear regression comes popping out of this kind of Bayesian perspective just like that, so is, is that part of what makes it so cool? That is part of what makes it cool, but I just think more generally about gradient descent right? The way gradient descend works is you take a derivative by stepping in this, in this space of the error function, which is sum of squared error. I see, so you get gradient descend too. Yes, you get all of the stuff that people have been doing. Now, there's a piece of beauty there, which is that we derived things like gradient descend and linear regression, all of the stuff we were talking about before and we have an argument. For why it's the right thing to do at least in a Bayesian sense. But there's an even deeper beauty here, which is tied in with ugliness, which is the reason this is the right thing to do, is because of the specific assumptions that we've made. So what were the assumptions that we made? We assumed that there was some True deterministic function that was mapping our x's to our in this case our d's and that they were corrupted say transmission error or line noise or however you want to think about it. They are corrupted by some noise that has a very particular form. Uncorrelated, independently drawn, Gaussian noise, with mean zero. So the less pretty way of thinking about it is. Whenever you're trying to minimize the sum of squared error, you are in fact assuming that the data that you have has been corrupted by Gaussian noise. And if it's corrupted by some other noise, or you're actually not trying to model determinance function, of this sort. And then you are in fact, possibly, in fact most likely doing the wrong thing. I mean are there other noise models that lead to some other kinds of learning. Sure, pick any other model in here that does't look Gaussian at all, and you would end up with something else. I don't know what you would end up with because. You know, you couldn't do all these cute tricks with natural logs but yes, you would end up with something different. And one question you might ask yourself is well, if I try to do minimizing the sum of the squared errors, or something for which this model was not the right one, what sort of bad things might happen? Here let me give you an example, let's imagine that we're looking at this here, and our X's are, I don't know measurements of people. Okay? So height and weight. Something like that. Mm-hm. And in fact let's make it, let's make it let's make it even simpler than that. Let's imagine that our x is our height. And our outputs, our d's, are say weight. And what we're trying to learn is some kind of function from height to weight. Now, this doesn't make a lot of sense to have a true [INAUDIBLE], but I'm trying to make a point here. So what we're saying here is that we, we measure our height and then we measure weight. That there's some simple relationship between them that's captured by f. But, when we measure the weight, we get a sort of noisy version of that weight. Okay? That seems reasonable. But what's not reasonable is we're saying. Our measurement of the weight is noisy, but our measurement of height is not. Because if the x's are noisy, then this is not a valid assumption. I see. So, it seems to work a lot of the time and we have an argument for when it will work, but it's not clear that this particular assumption actually makes a lot of sense in the real world. Even though in practice it seems to do just fine. Okay, got it? I think so though I feel like if the error if you put an error term inside the f along with the x and f is say linear. Mm-hm. Then maybe it pops out and it just becomes another part of the noise term and, and it all still goes through. Like I feel lines are still pretty happy even with that. No I think you're right. Lines would be happy here because linear, I mean linear functions are very nicely behaved in that way. But of course, they'd have to be the same noise model in order for it to work the way you want it to work. Yeah. They'd have to both be Gaussian. They have to both have zero mean, right? And they'd have to be independent of one another. So your measuring device that gives you an error for your height would also have to give you an independent normal error for the weight. Yeah. Though I feel like my scale and

my yardstick actually are fairly independent. And they're Gaussian? . Oh mine is clearly Gaussian. Yeah. Yeah. Well at least they're normal. They're normally are. Mm-hm. Okay good. So let's move on to the next thing Michael. Let's try one more example of this and, and then I hope that means you got it, okay? Sure. Beautiful.

1.7.11 Best hypothesis Question

So before we go on to the next example, Michael, I wanted to do a quick quiz, just to make certain you really get what's going on here. The, the sort of power of looking, using Bayesian learning. The, the main insight, I think, I, I want to drive home here, is something you said. Which is that, when we were doing regression before, when we were talking about the perceptrons, we actually had in our head a particular kind of function, a particular hypothesis class. In here with what been talking about with Bayesian learning, the answer tto finding to sum of squared errors was independent of the hypothesis class and only dependent upon the key assumptions that we were making, mainly that we had labeled the data with certain form, and that that data was generated by a process that took deterministic function and added some Gaussian noise to it. So, here's the quiz. Here's your training data. You've got a bunch of Xs and a bunch of Ds. These are the values that you have to learn. And I want you to tell me, which of these three functions over here, these three hypotheses is the best one under this assumption. Got it? mod? Are we allowed to do that? We are allowed to do that because It's just a function. It's just a function, man. Interesting. It's just a function. So we've got a linea-, a constant function, a linear function, and we've talked about those, but we've also got a mod function. alright, and we've got a uniform prior over these threes hypotheses. Yup. Okay. Yeah, I think I can do that. Okay. Go.

1.7.12 Best hypothesis Solution

Alright, we're back, what's the answer Michael? So, you want me to work it through? Sure. So what I did first is I made it to, I extended the table that you had. Okay. To include each of these, the output for each of these three functions. What I'm basically, what I like to do is compute the squared error for each of these three functions on that data and then choose the one that has the lowest squared error. Make total sense to me. Sounds good enough to be an algorithm. So you want me to write out the table? Well, I mean, I started to do that, and then there was like one too many steps, and I just threw out my hand and just wrote a program. Okay, so, we'll just say" Insert code here", because that's what you did, that was the step. And, what did your code tell you? Well, let me start with the constant function, because that's the easiest piece of code. So, I'm saying what's the difference between each of the D values and two. Squaring it all and summing it up and I get 19. And I can do the same thing, instead of using two I use x over three take the difference of that to the D values and square that and I get 19 point four, four, four, four, four, four. Then I can do, right, so now at this point I'm rock and rolling. I can actually just substitute in my nine, and I get 12? Not, not something-odd 12? No, just 12, so the error's 12. So that has the smallest error. So even though that's sort of a crazy, like, stripy function right. Like, it increases linearly and then it resets at 9. Mmhmm. It actually fits the state of the best. That is correct. Your code is correct, Michael. Well done. Well actually, looking at this data that sort of makes sense to me, right. Because if you look at the first three examples. Of the data, the outputs are very close. But the outputs of the next three are much bigger, and by doing a mod nine, what you effectively do is say, this is the identity function above this line. And then below the line, it's as if I'm sort of subtracting nine from all of them, and that makes them closer together. Hm. And so it just happens to work out here. But surely that's just because we came up with a bad constant function and a bad linear function. Do you think there's a better linear function? So I mean because it's the squared error, we're really just talking about linear regression. Right. So I can just run linear regression. So I get an intercept of 0.9588 And a, and a slope of 0.1647. Okay So that's, so that's my linear function of choice. Okay, so that's, what was, what was that again? So x times, you know, it's like a six, I guess, like 0.165 probably. 0.165x Plus Mm hm. Plus 0.959. So that's our function, that's our best linear function, the function that minimizes greater. So it better end up being, it better end up being less than 19.4, or I'm a liar. Mm-hm. And now I need to take the difference between that and D square it, and sum. 15.7. Hm. So that gives you 15, I'm going to say 15.8. So that is better. Yeah, so it's better than the X over three, but it's also worse than the mod nie. Hm. and the best constant function, has to be worse, because the linear functions include the constant functions as a subset, so this is, that 15.8 is. Is better than the best constant function too. Oh its easy to do though right? Because the best constant function is just the mean of the data. What is the mean of the data? 2.17. huh two is pretty close. Yeah that's interesting. Well that's Kind of in the middle of the pack I guess. That sort of works right because two the error for two was actually lower than the error for

x divided by three. And for what it's worth the error for 2.17 as constant function to 2.2 is 8.8, 18.8, 18.8 sorry. Yeah, you're not the [INAUDIBLE]. Yeah, eight would have been less than everything. Okay. So, what have we learned here? That sometimes you want to use mod. Yeah. If your data is weird. [LAUGH] You have you have definitely modified my box. Well I'm glad you found it mod. Hmm. [LAUGH] PUNS. Okay, good, so I think that was a good, that was, that was a good exercise. So I'm going to give you one more example of deriving something and then we're going to move on. Cool. Okay.

1.7.13 Minimum Description Length

All right, Michael. So I, all I have written up here for you is, are a maximum a posteriori equation, right? So the best hypothesis is the one that maximizes this expression. Nothing new, right? So I want to do a little trick, the same trick that you did before. So, you noticed that when we had E to the something, that we could use the natural log on E to get rid of everything. So I am going to try to do the same thing here. In the nat, why did the natural log work again? Well, it's the inverse of the E, but it let us turn products into sums. Right. And the other reason it worked is because it's a. Oh, it's monotonic. Right, it's a monotonic function and so it doesn't change the argmax. So, I'm going to do the log of both sides here. But this time I'm going to do log base 2, for no particular reason other than it'll turn out to help later. So, I'm just going to take the log of this entire expression, which, because it turns products into sums, gives me this. And by the way for those of you who haven't noticed, I drew in a little bit of notation here. When you write just LG, it's just log base 2. Okay, so, we agree that the answer to this equation and the answer to this equation is the same. And now I'm going to do one other little trick, exactly the trick that you used before. I'm going to change my max into a min, by simply multiplying everything by minus 1. Okay, I don't quite see where you're going here. But you agree that we haven't changed the answer. I agree that we haven't changed the answer. Okay. Do a log in there, do a minus sign in there that took us from a max to a min, but I haven't changed the answer. Now, do you recognize anything about these expressions? I'll give you a hint. Information theory. Okay. So, information theory is usually entropy, which is like sum of P log P stuff. Right. I'm not seeing that. Well, there you, there's your log and there's your P. Sure. [LAUGH] [LAUGH] It's not P times that though. That's true. But we know from information theory, based exactly on this notion of entropy, that the optimal code for some event with probability P has length minus log base 2 of P. So, that just comes straight out of information theory. That's where all the entropy stuff comes from. Okay. So, if we have some event that has some particular probability P of happening, the best code for it has this structure, minus log of P. Okay. So, if we take this fact that we know, and we apply it to here, what is this actually saying? This is saying that, in order to find the maximum a posteriori hypothesis, we want to some how minimize two terms that can be described as lengths. Okay. I can see that. So my question to you is, given that this definition over here, that an event with probability P has some length minus log P, what is this the length of? So that would be the length of the probability of the data given the hypothesis. Mm-hm. And the length of the hypothesis, or the probability of the hypothesis. Well no, it's just the length of that hypothesis. Oh, because the event is what has the length. Oh, I see. So it's the length of the data, given the hypothesis, and the length of the hypothesis. Right. So let's write that out. But I was just doing, like, pattern matching there. It's not clear to me what a length of a hypothesis is. Hypotheses are functions. I don't know how to take a tape measure to a function. That's fair. So this is the length of the hypothesis. Right? Yep. So, you said you don't know what that means. But, let's think about that out loud for a moment. What does it mean to have a length of a hypothesis? That's really sort of the number of bits you need to describe a particular hypothesis, right? Okay. Okay. And in fact, that's exactly what it means. That's why we use log base 2. So, if we want to minimize the length of a hypothesis, what does that mean, the number of bits that we need to represent the hypothesis? The number of bits that we need to represent the hypothesis is, I guess, in some representation, or, so in this case I guess it would be some optimal representation. We are taking all the different hypotheses and writing them out. The ones that are more likely have a higher P of H, because that's the prior. And those are going to have smaller lengths than the optimal code. And the ones that are less common are going to have longer codes. Well, let's make it more concrete.

1.7.14 Which Tree Question

So here are two decision trees, which one has the smallest length? Go.

1.7.15 Which Tree Solution

Okay, Michael. Which of these two decision trees is smaller? [LAUGH] The one on the right is smaller. That's exactly right because it's easier, it's, it's easier to represent it in sort of almost any obvious way that you could think of. It has fewer nodes, so smaller decision trees, trees with fewer nodes, less depth, whatever you need to make it smaller, have smaller lengths than bigger decision trees. So that means that if all we cared about was the second term here. We would prefer smaller decision trees, over bigger decisions trees. Which we do. Which we do. Now what about this over here? The, what does it mean? So this is pretty straight forward. You got this right? That the length of. Well, I mean guess what's weird that you, you're kind of moving back and forth between a notion of a prior, which is where the p of h came from and a notion of Well, you know, if we're going to actually have to describe the hypothesis you're going to have to write it down in some way, and this gives you a way of measuring how long it takes to write it down. But I guess what this whole derivation is doing is linking those two concepts, so that you can think about our bias for shorter decision trees as actually being the prior, right? Actually being the thing that says the smaller ones are more likely And vica versa, that when we think about things that are priors, that are assigning higher probability to certain things, it's kind of like giving them a shorter description. Right, so infact if you were to take this example literally here, that we prefer smaller trees to bigger trees, this kind of a bayesian argument for occam's razor. And pruning. And pruning. Well, you, often use razors to prune, so it makes perfect sense. Ok, so this is kind of straight foreward, that basically smaller trees are smaller than bigger trees. It sort of makes sense. Now, what about this over here? What does it mean to talk about the length of the data given a particular hypothesis. Uh...I could think of one interpretation there. So like, if the hypothesis generates the data really well, then you don't really need the data at all, right? You just have...you already have the hypothesis. The data is free. Right? But if it deviates a lot from the hypothesis, then you're going to have to have a long description of where the deviations are. So maybe it's kind of capturing this sort of notion of how well it fits. Right, that's exactly right. So I like that explanation so let me write it down. So here we literally just mean something like size of h. But over here we are talking about, well sort of error right? if the hypo, if just exactly what you said if the hypothesis perfectly describes the data, then you don't need any of the data. But let's imagine that the hypothesis gets all of the data labels wrong. Then when you send the hypothesis over To this person. This, this sort of person we're making up who, trying to understand the Daden hypothesis. And you're also going to have to send over what all the correct answers were. So, what this really is, is a notion of miss-classification error, or just error in general. If we're thinking about regression. So, basically, what we're saying is, if we're trying to find the maximum a posteriori Hypothesis. We want to maximize this expression. We want to find the h that maximizes this expression. That's the same as finding the h that maximizes the log of that expression, which gives you this. Which is the same as minimizing this expression, which is just maximizing this expression but throwing a minus one in front But these terms actually have meanings in information theory, 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 literally occam's razor. What is important here In reality is that these are often traded off of 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. Hm. So this actually has a name, and that is the minimum description, and 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. Now, if you think about it for a little whiel Michael you'll realize that yea this sort of makes sense at the hand wavy level at which I just talked about it. But, you do have some real issues here about for example units. So, I don't know if the units of the size of the hypothesis are directly comparable to the counts of errors or you know sum of squared errors or something like that and so you have to come up with some way of translating between them... And some way of making the decision whether you would rather minimize this or you'd rather minimize that if you were forced to make a decision. But the basic idea is still the same here. That the best hypothesis is the one that minimizes error without paying too much of a price for the complexity of the hypothesis. Wow. So I've been sitting here thinking about, so with decision trees, this notion of length feels... Like you could translate it directly into bits right like you actually had to write it down and transmit it, it makes a lot of sense. But then I was thinking about neural networks. And, and, and given that a fixed neural network architecture it's always the same number of weights and they're just numbers. So you just transmit those numbers. So I thought, hmmm, this isn't really helping us understand? ? ? ? ? and then it occurred to me that those weights, if they get really you're going to need more bits to express those big weights. And in fact that's exactly when

we get over fitting with neural nets if we let the weights get too big. So like this gives a really nice story for understanding neural nets as well. Right. That the complexity is not in the number of parameters directly but in what you need to represent the value of the parameters. Wow. So I could have ten parameters that are all binary, in which case I need ten bits. Or they could be arbitrary real numbers, in which case I might need, well, an arbitrary number of bits. That's really weird. Yeah, but the point here, Michael, I want to wrap this up. The point here is we've now used Bayesian learning to derive a bunch of different things that we've actually been using all along, and so again the beauty of Bayesian learning is that it gives you a sort of handle on why you might be making some of the decisions that you're making. It seems like this raises the theory question that you threw at me in a previous unit. Right. Which is like well so if it doesn't really tell us anything we didn't already know, how important is it? Well in this case, I think it is important because it told us something that we were thinking and tells us in fact we were right. So now we can comfortably go out in the world minimizing some of squared error when we're in a world where there is some kind of Gaussian transmission noise. We can go about trying to Believe Occam's Razor because Bayes told us so. [LAUGH] Thanks to Shannon. And so on and so forth. We can do these things and know that in some sense, they're the right things to do, at least in a Bayesian sense. Neat. Okay, good. Now one more thing, Michael, I'm going to show you. Which is that everything I've told you so far is a lie. [SOUND]

1.7.16 Bayesian Classification Question

Okay, Michael, so here's a quiz. Now it's a pretty straightforward quiz. I just want you to use everything that you've learned so far. Okay, so you have three hypotheses. Let's call them h1, h2, and h3, okay? Mm-hm. For the sake of argument. Here's what each of these hypotheses Outputs for some particular x. h1 says plus. h2 says minus. h3 says minus. Okay? Mm-hm. Now here we've, already made it easy for you and we've computed the probability of, a particular hypothesis given some set of data. I'm not showing you the data but I'm showing you the answer for it. Okay? So the probability of h1 given the data is 0.4, h2 is 0.3, and h3 is 0.3. Got it? Wait hang on, so, okay, I see that corresponds here about this given data, what's x? x is some input, it doesn't matter, just like it doesn't matter what the date is. [LAUGH] 'Kay. Just call it so that, that, x is x, okay? It's just some object out in the world and each hypothesis labels it. Plus or minus. 'Kay. Or can label it plus or minus. And H1 decides if for that X, it's positive, and the other two hypotheses decide that it is, in fact, negative and H1 has probability that is the maximum a posterior probability h1 is .4, h2 is .3 and h3 is .3. So my question is, very simple. Using all of the magic we've done, this is just to make sure you've got it, Michael, I dunno, we've done a lot of derivations, we've walked away from some things [LAUGH] we gotta make sure we get back to basics here. What is the best label for x? Is it plus, or is it minus? I see why this is tricky. Okay. And go.

1.7.17 Bayesian Classification Solution

And we're back. What's the answer, Michael? Okay, so it depends. What does it depend on? I've given you everything. This is straightforward. Well, so, okay, I guess. The here, so here's what I'm seeing. So I'm, what I'm seeing is that hypothesis one is the most likely hypothesis. It's not just the most likely, it's the most a posteriori. Well, that's what I mean by likely. Right, is the map hypothesis? It's the maximum a posteriori hypothesis. So if we say, what is the label according to the map hypothesis? Boom, it's plus. Yes. But, if we're saying what's the most likely label. So the most likely label is, is, we have to actually look over all the hypotheses and in a sense, let them vote. So the probability that the label is minus is actually 0.6, which is greater than 0.4, so if I had to pick, I would go with minus. And you would be correct. So I did a little tricky thing here for you Michael. You've been complaining about my titles, because everyone said Bayesian learning and I changed the title here to Bayesian Classification. Ohhh. Because in fact the problem here, we've been talking about all along is, what's the best hypothesis. But here. I ask you what's the best label? Hm. And exactly as you point out, finding the best hypothesis is a, is a very simple algorithm. Here I'll write it for you because we did it before. For every H in hypothesis set, simply compute the probability that it is the best one, and then simply output max. That's how you find the best hypothesis, but that's not how you find the best label. The way you find the best label is you basically do a weighted vote for every single hypothesis in the hypothesis set, according to the weight being the probability of that hypothesis given the data. Okay. So the best, if you can only output hypothesis and use that hypothesis, in fact, you would say plus. But if you asked everyone to vote, just like we did with boosting, just like we did effectively with KNN and all these other kind of. Weighted regression techniques we've used before, you need to do the voting. And I, and I feel like I could probably derive that using rules of probability. Right, because really what we want is we're trying to maximize the probability of the label, given the data, and I think

the probability laws would tell us that's equal to the sum over all hypotheses of the hypothesis and the label given the data, which is, like, the probability of the hypothesis given the data, times the probability of the label given the hypothesis, and that's what we did, we summed up. You know, the probability of the label given the hypothesis is either one or zero. That's your left column. And then we're summing up the probabilities that corresponding to the pluses. And we're summing up the probabilities corresponding to the minuses and choosing the largest one. So, this is what you just said written down as an equation. basically, the most likely value. Is the one that maximizes this expression. And this follows directly from Bayesian's rule where now instead of trying to maximize the hypothesis given the data, you're trying to maximize the value given the data. And I think it's pretty straightforward to derive that but I'd like to leave it up to the students to do it on their own. Okay, so Michael, in some sense everything that I've told you before is a lie, in that I've led you down this path that somehow, finding the best hypothesis is the right thing to do. But the truth is, finding the value is what we actually care about. Finding a hypothesis is just a means to an end. And if we have a way of actually computing the probabilities for all the hypotheses, then we should let them to vote in order to find the best actual label or the best value for it. Got it. All right. Good.

1.7.18 Summary

Okay Michael so this wraps up all this Bayesian Learning stuff. What have we learned today? We did Bayes rule. We learned Bayes rule. We even learned how to derive Bayes rule. And it was super useful because it let's you swap, kind of, causes and effect. So I like the way you put that, Michael that we're swapping causes and effects. Sort of mathematically when we think about Bayes rule, what that really let's us do is. Instead of having to compute the probably of a hypothesis given the data We instead view to compute the probability of the data given the hypothesis, which is typically a much easier thing to do. And what makes it of course Bayes rule in general is that you weight that by the prior probability over the hypothesis. Which in fact is one of the important things that we learned which is that priors matter. So anything else we learned? Yep, we did the MAP hypothesis, Maximum a posteriori. Right. We learned about HMap, and we also learned about HML. ML, right. The maximum likelihood hypothesis. ¿ Right. And what's the maximum likelihood hypothesis? How's it relate to the maximum a posteriori hypothesis? It's the MAP that you get when the prior is uniform. Right. Alright. And we, oh, we connected up maximum a posteriori and least squares. Yeah, that was pretty, I really liked that. So, we basically der, we deroved. We derived a bunch of things we'd been doing before. And short of showed that there's actually a good argument for them. At least if you're Bayesian. There are good arguments for doing some doing sum of squares. There are good arguments for Occam's Razor. We'd actually be able to give real justification for doing them other than, well sure it makes us one of them. Right so that includes the minimum description length story. Mm-hm. And then finally, you told me that was all a lie, and you said that really what you want to do is this other kind of way of picking that actually factors in the probability of all the different hypotheses and having them essentially vote. Right. What we really care about, is classification. We're learning in the end and so we also learned about Bayes classifiers. So in fact, what we described before, which is voting of hypothesis. Turns out to be the Bayes optimal classifier. I didn't say that, but it is very important to note. In fact, what you should be noting there is not only is it the Bayes optimal classifier, it's the Bayes optimal classifier. And what that means is that on average you cannot do any better than basically doing a weighted vote of all the hypotheses according to the probability of the hypothesis given the data. You cannot do any better than this on average. So again, what Bayesian learning gives us and what Bayesian classification gives us is a way of talking about optimality and gold standards. What'd you think, Michael? That's really neat. I like it. I mean, I have to tell you, I really think that this stuff is kind of cool. It's always nice to be able to take things that actually work and explain them according to some framework, some underlying theory. I wonder though, it seems like all these Bayesian equations lead us to the question, of how we actually infere probabilities from various different quantities and observations. So is there a way to do that? So I think the answer is yes. And maybe you should go figure it out and then tell me about it next time. Okay. [LAUGH] All right, as you wish. As you wish. Stay tuned. Anyway, this has been a lot of fun, Michael. I will talk to you later. Thanks. Bye.

1.8 Bayesian Inference

1.8.1 Intro

Hey Charles. Hey Michael. So like I get to lecture near you today. Yes you do. I can even see you. This is, this is crazy. I sort of don't have my regular pad. This makes me a little uncomfortable. But you look very dashing in your nice blue suit. Thanks. We're going to record some live action stuff today. Mm. [LAUGH] All right so. Do you remember last time we were talking about Bayesian learning? I do, because I led that. Right. Good point. And so one of the questions that I asked as a follow-up was, these quantities, these probabilistic quantities that we're working with. Is there's anything that we need to know about how to represent and reason with them. And you said that I should look into it. Yeah, because I, I just, I yeah you should look into it. So I did. So, and it's cool. And so I figured it would be fun to tell you about it. Okay, well I look forward to it. Thanks! And also I want to point out, we're using a different color scheme today. Isn't that a nice blue? It is a nice blue, its sort of a relaxing blue. As opposed to that blue blue that we used before. It's like Cerulean... Is it? No. It's more like periwinkle. No, it's definitely not periwinkle. Oh you're right, it's not periwinkle. Periwinkle's a Navy. No, it's too light to be navy. All right, so so good, so right. It turns out that there's this concept called Bayesian Networks, which is this wonderful representation for representing and manipulating probabilistic quantities over complex spaces. And so it fits in really well with the stuff you were talking about last time.

1.8.2 Joint Distribution

Alright, so to make this work, we're going to need to build on this idea of a joint distribution. It's not going to be obvious right away what this has to do with machine learning, at all. But, I, it's going to connect. So, just bear with me for a little bit. Alright, so to talk about this concept, what we're going to do is look at an example. And the example that I think might work, that would be nice and simple, is the notion of storm and lightning. So, here's a little picture of storm and lightning. And what we're going to do is say, let's say, on a random day, at 2 PM. You look outside. And, what I want you to do is say, what fraction of the time, is, is each of these different possible combination of things happening? So, for example, what's the probability that you look out and there's a storm and there's lightning at the same time? So, what do you think? On a random day? Yeah, random day at 2 PM. And we can be in Atlanta since that's what you're familiar with. Is it summer? Because that happens more often in the summer. Sure, let's say summer. It's fairly high at 2 PM. Let's say it happens a quarter of the time. Wow, that's a rainy summer. Mm-hm. Alright. Now, that's not the only possibility though. It could also be that there's a storm but no lightning. Right. That happens more often at 2 PM in the summer in Atlanta. Let's say it's mm, .4. Wow. Alright. Now what's the probability that you look at the window and there's no storm but there is lightning. Maybe 5%. And what's the probability that you look out and there's, you know, it's nice clear there's no storm no lightning. Coincidentally I picked numbers that made it easier for me to subtract from one. So, it's 0.3. [LAUGH] Right and so these, there's only, these are the only four possibilities. We're saying. And they, so they have to add up to 100%. And so I, yeah it had to be 30 at this point. So, it's actually more likely that there's a storm than not, according to what you said. It's Atlanta in the summer at 2 PM. There you go. Alright. So, this is a joint distribution. And now we can actually ask various kinds of questions about this. Oh, you know what would be a good form for asking a question. I don't know. I'm looking at you quizzically. Nice. Using the fact that we are in the same place. We are going to do a quiz.

1.8.3 Joint Distribution Quiz Question

You ready for a quiz? Yes, I am. Okay. Here's what I'd like you to do. I'd like you to use these probabilities that we have written down here, that, that constitute the joint distribution of, when you look out do you see a storm, do you see lightning? And use these numbers to answer some other questions that aren't directly in here but you can figure it out. So, the first one is to say, what's the probability when you look out the window at 2 PM in the summer, in Atlanta, that there's no storm, okay? Okay. And, then the question is to say, what's the probability that if there is a storm, there is also lightning, okay? So the probability of lightning given that there is a storm. And we've done some stuff with conditional probability. So these concepts should be familiar with you, but you should be able to connect it up with, you know, the numbers in the table. You ready? I am ready. Go.

1.8.4 Joint Distribution Quiz Solution

All right. Let's here it. Okay. So here's the process that I went through. I'm just going to talk this out. I haven't actually worked it out in my head yet. So what's the probability that there isn't a storm? Well the way you have this drawn it actually makes it pretty easy to see. I can just look at the cases where storm is false, and it turns out there's two of them. And I can just add those probabilities over there, and I get .05 plus .30, and that gives me .35. That's great. Yes, so that's exactly what you did. So you went through, and now all that matters in the universe are the cases where they're not a storm and that ended up being these two numbers. And you said, well Those are two different cases that can happen. We'll just add their probabilities because they're not overlapping and you've got .35. Great. All right what about the second question? Okay, so that's probability that there's lightning in a world where there's a storm so I'm going to do a very similar trick. I'm going to look at the cases where storm happens to be true. And conveniently they're the first two rows and I have two cases, so we know the probability of there being a storm is 0.65 which is good, because 0.65 and 0.35 add up to one. But that's not the probability of there being lightening, given there is a storm. So, of those two cases, there's only one where lightening is happening, windstorm is happening, and that's 0.25. But 0.25 isn't enough because it's only 0.25 out of 0.65. Hm. So the correct answer would be 0.25 divided by 0.65. Which is, some number. 5 13th's? Yeah. It's 5 13th's. And, though I'd rather that people fill it in as a fraction. As a, wait. That is a 5 13ths is a fraction. Good point. As a point something something. A decimel. So, 5 13ths is obviously 0.4615. And there you go. Is that right? Yes. That was perfect. Yeah so its usually when there's a storm, its not lightningy. It's less than half the time. That makes sense. It does because otherwise lightning would be happening all the time. Well when its storming. It could be that its very likely when its storming. It is likely when it's storming, but it wouldn't be happening every time its storming because otherwise it would be lightning all the time when its storming. RIght. And often there's breaks between lighting. In fact, most of the time there's not lightning, at least outside my window. At 2pm. In the summer.

1.8.5 Adding Attributes

Alright, so that wasn't so bad. You are able to compute some probabilities from this joint distribution. So let's see what happens when we start talking about more variables. More propositions that could be true or false. What I did is I filled in thunder as another variable and thunder can be true or false in each of these cases. And I wrote down what the probabilities could be from my experience in Atlanta in the summer. I was, I was around over last summer, and in 2004, so let's, so I'm an expert obviously, so I'm able to estimate these probabilities to the nearest percent. Anyway the point is, that one of the things you should notice here is that each time we add one variable what happens to the number of probabilities that we have to write down? Well in a world where it's binary it goes up by two. A factor of two, right? A factor of two. Not just, not just two more, but like, twice as many. And so if we have a complicated scenario that we want to be able to reason about, and it's got, I don't know, a hundred variables, that's going to be a lot. That's, that's, I can't even, I can't even think about that. Yeah, it's like two to the hundred is. That's, that's not even a real number. It's technically a real number, but it's an, it's an unimaginably large number. There's only like four numbers, one, two, three, many, and too many. So it's going to be really inconvenient as we start adding more of these and especially if we add variables like, you know, remember the restaurant example that we worked on when we were doing decision trees. Oh yeah those were the days. Then there was variables like food type, and what was the deal with food type? It had lots of values that it could take on. Yeah, yeah like five or something like that. Thai an, American and Italian. Right and so if we had, add variable like that it's going to multiply the number of probabilities that we need by five. So this is going to get really big really fast. So would it be nice if we had an more convenient way of writing it out in this distribution? Yeah, it would be nice. So it turns out that we can factor it. But I thought we already had a factor of two? Well that was a joke but it actually is pretty close to being the truth, which is the idea that instead of representing all, so, so, in this case, there's eight numbers. Instead of representing them as eight numbers, we're going to represent it by you know, 2 times 2 time 2. So we really are going to essentially factor it. putting, putting things into pieces that we can recombine, smaller pieces that we can recombine into, into larger pieces. And it, yeah, it turns out that actually works out really well.

1.8.6 Conditional Independence

Alright, I'm going to hit you with a definition first. Hit me. So, conditional independence is this idea that goes like this. We're going to say that some variable that makes up the joint distribution is conditionally independent of some other variable, Y, given Z, if it's the case of the probability distribution governing X,

so 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. So that is, if it's the case that for all possible values, little x, little y and little z for the variables big x, big y, and big z, If it's the case that the probability that big X, the random variable big X, equals, takes on the value of little x, given that big Y takes on the value of little y and big Z takes on the value of little z, equals the probability that big X takes on the value of x given big Z takes on the value of z. If those are equal for all possible ways of filling in the values of the variables, then we say that x is conditionally independent of y given z. Right, so you see we dropped Y from the right-hand side of the probability expression. Okay, so it's sort of less things we have to worry about, if it's the case that we really didn't need it in the first place. Fewer. Fair enough. So that's pretty similar to normal independence. Okay, so what's normal independence? So normal independence, we say the probability of x and y is equal to the probability of x times the probability of y. That's right. Which means if we think about the chain rule, we also know that the probability of x and y is equal to the probability of x given y times the probability of y. So that means that the probability of x given y is equal to the probability of x, for all values of x and y. So this is actually implying. So [INAUDIBLE] if it equals that. Oh, that means that P(x) times P(y) equals P(x given y) times P(y). If we cancel those, we get px equals. Okay. That's what you wanted to say. Right. So, since, What independence means, right, is that the joint distribution between two variables is equal to the product of their marginals. That's just. You know 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 you wrote down for conditional independence. Right, the only thing that we added is this notion that it might be the case that we don't have such a strong property as this where it's always the case that you can write the probability of x given y just with the probability of x. But in the context of some, of knowing some value z, it might be true. And that's what conditional independence gives us. As long as there is some z that we stick in here, that gives us that property, that's great, we can essentially ignore y, when we are talking about the probability of x. Okay, that's pretty cool. That means more powerful or something. Yeah, and in fact if you remember you mentioned the word factoring. You can see here that we are down a probability as the product of two other things. We are factoring that probability distribution. That's what independence let's us do. And conditional independence let's us do that in, in more general circumstances. So let's apply this content back to what we were talking about before. Okay.

1.8.7 Conditional Quiz Question

So, here's a quiz using this notion of conditional independence. So, bear with me for a second, because this is a little bit weird the way that I wrote it. But, what I'd like you to do is find a truth setting for thunder and lightning. So like, true/true or true/false or false/true or false/false. Such that, the following thing holds true. That the probability that thunder takes on that value, given that lightning takes on the value that you give, and the storm is true, ends up equaling the probability that thunder takes on that value given lightning takes on the value that you gave and storm is false. Right, so a setting here so that basically the value of storm doesn't matter. So, whatever I put in the upper left box has to be what I put in the lower left box. What I put in the upper right box has to be what I put in the lower right box. Right and in fact we're just not going to give you boxes for the other ones. We'll just give you the two top boxes and automatically fill in the bottom box. Okay, that seems reasonable.

1.8.8 Conditional Quiz Solution

Alright, so how are we going to figure this out? By you letting them figure it out while I figure it out. [LAUGH] I think you should figure this out. Okay let's figure it out. It might not be obvious just looking at it blankly so why don't we just throw in some values here. So, for example we can do this. Mm-hm Which is, it gets filled in in both places. So the probability that thunder is true given that lightning is false and storm is true, what is that number? Well, so we just have to find the place in our little eight-row table where lightning is false and storm is true. Lightning is false and storm is true, uh-huh. Which is there. Uh-huh. And the probability that thunder is true is 0.04 divided by 0.4. Oh cause we're asking about thunder right. Was what's the probability that thunder is true given that the other two things lightning is false and storm is true so that's going to be divided by the point 4. That's the setting that we're in. Right and Point 04 divided by point 4 is point 1 Right so maybe we'll get lucky and it will work out the same with the other one. So where do we have to look for that one? Well now we have to look in the row where lightning has false and storm is false. Okay. Down here. And look at the case where thunder is true, and that's .03. .03

divided by .3 which is also .1. Woo hoo! So that works as an answer. It turns out that, in fact, no matter what you type into these two boxes, it does, in fact, work. And what does that tell us? Well, it tells us that it doesn't matter what the value of storm is. We can figure out the value of thunder by only looking at the value of lightening. So, that is to say, that the probability of thunder given lightning and storm is equal to the probability of thunder given lightening or that we have conditionally independent variables. Yes, that's right. Storm is conditionally independent of thunder, given lightning. Right. So, the probability of thunder giving li-, given lightning and storm, is equal to the probability of thunder, given lightning. That means that thunder and storm are conditionally independent, given lightning. Or thunders conditionally independent of storm, given lightning. Sure. Very good. Alright. So now what we're going to do next is say, Okay well given that we have this nice property. And yeah, I, I worked a little bit to make sure that the numbers, worked out. It doesn't always happen this way, but here we had some nice conditional independence and what, we're going to do next is look at a nice representation of that, kind of information.

1.8.9 Belief Networks Question

So the concept of a belief network, sometimes also known as Bayes Net. Sometimes also known as Bayesian Network. Sometimes also known as a graphical model. And there's other names, but it's the same idea over and over again. And the, and the idea is that what we're going to do is we're going to represent the conditional independence relationships between all the variables in the joint distribution graphically. In terms of of a little picture like this, where there's nodes corresponding to all the variables. And, edges corresponding to dependencies that need to be explicitly represented. So, the way that this works is, what we can do is we can fill in the prior probability of storm, which we can get by just marginalizing out. So we've, we've already done an exercise like this. So this is a number you should be able to figure out. Then because of vary well, this is also true that that you can figure out what the probability of lightning is, given storm and also given not storm. And these are numbers that you can just get by marginalizing out. Finally, the probability of thunder, normally you'd have to condition that on both storm and lightning. But as we already talked about, it's actually conditionally independent of storm given lightning. So, all we need to figure out is the probability of thunder given lightning, and the probability of thunder given not lightning. And once we have these, in this case five numbers, that's enough to work out any probability we want in the joint, just by multiplying corresponding components together. So, what I'd like you to do is actually fill in these boxes as a quiz. And to help you out we copied the numbers over from the previous slides so that you actually have the [LAUGH] values that you need to fill in this table. because otherwise that would have been kind of mean.

1.8.10 Belief Networks Solution

Alright Charles can you work out these numbers? I can. So the first one is pretty easy because we did that once when we were talking a couple slides back. We did. We just look at the case where a storm is set to be true. Those are, those two mega rows there and those are .25 and .4. We add that up and we get .65. We're pointing out that since we know that S is .65, we know that not S is .35. Good. Okay. Although that table really has two numbers in it, we only need one of them. Right. Yes. Very good point. because it's constrained by needing to add up to one. Then we do something similar with lightning. We look at the cases where lightning is true. And s is also true. Yep. There's just one case like that. Huh? Huh, there is only one case like that. Right, but what we really want to know is what's the probability that lightning is true given that storm is true. So we need to think about both cases where storm is true and say of these, what's the probability that storm...that lightning is true. And it's .25 over .65. Right. Which is .385 rounded up. because you're a cowboy. Which means that... The probability of it, of not L given S is one minus that or .615. That's right. Okay. So we do the same trick with probability of L given not S and we find the case where lightning is true but storm is false and that's .05, or we have to do it out of both cases where S is false and so it's .05. Divided by, point .05 divided by .35 which is, 1 7th. And 1 7th is approximately .143, rounded up. And so not L given not S is .857. [LAUGH] Nicely done. I use subtraction in my head. In your head yeah, but it was like with carries and stuff that was nice. And right, so let's see. And, does these sorts of things make sense. Of not a storm, it's kind of unlikely that we'll see lightening. Or, if there is a storm, it's moderately common that we'll see lightening. Okay, that makes sense. Okay, good. So, now we do the same trick again with thunder. Except now, instead of looking at L and S, we look at Thunder and, and lighting, so we need to look a case where thunder is true and lightning is true, so that would be, point, that's all the cases where lightning is true, so it would be .2 divided by .25 Alright and why are we looking at the case where storm is true? Why are we doing it? Because it's conditionally independent of storm. It doesn't

matter. [CROSSTALK] Information, so it doesn't matter which rows we look at. What matters is we look at a case where thunder and lightening are both true, and we compare that to thunder is false and lightening is true. So that's this number. Those add up to the 0.25, we get 0.2, over the 0.25, which is 0.8. Right. So it's very likely to hear thunder if you see lightning. That makes sense. And there's only a 20% chance that you don't hear thunder when you hear lightning. It's lightning not thunder, yup. Mmhmm. And so we do the same thing in the case where we have thunder and there's not lightning. So we find that row. Okay. Not lightning and there is thunder. There's one. Right and we do the same trick we did before and we get, .04 over .4. Which I think we did last time, actually, and we get .1. We did. So, if it's, if there's not lightening out, it's very unlikely to hear thunder. Alright. Alright and just to drive this point home. That was great. Just to drive this point home. What if it was the case that it mattered what's value storm had, how would we fill in this table. Well we'd have to look at a lot more rows. Well in particular we couldn't draw this kind of belief network if that were the case, right? Right. Because it wouldn't be conditionally independent. So we'd have to draw basically another edge. Here, and what that represents is that thunder, to work out to what the probability of thunder is, you have to look at storm and lightning, all the joint combinations of those to make it work. ¡¡ And that grows exponentially as you add more and more data. ¡¡ And that's right, and that's something that threw me when I started to look at this, because the picture looks a lot like a neural net. Right? In a neural net, you've got these nodes, you've got arrows going into the nodes, and when you have a bunch of arrows going into the same node, you just end up like adding all those different influences together, weighted by what's, what it has on the weight. This belief network representation is an entirely different animal. In particular, now, what we're really saying is, to work out the value of this node, you need to know what's going on in all combinations of what the inputs are. And so, as you pointed out, so astutely, that grows exponentially as you have more variables coming into the node. Higher in degree. Hm. So this is not just a network. It's a graph. And so we can talk about parents and children right? So, basically, the number of numbers you have to keep track of is exponential in your number in your parents. I mean it's a, yes. Though it's not exactly a tree. Doesn't have to be a tree so the parents relationships are kind of weird. Like in particular, if you use parent terminology in this graph, what you're saying is that lightning has one parent which is storm and thunder has two parents which are storm and lightning. So it's, storm is it's own grandfather and parent. So let me ask you a quick question, Michael. So earlier on when you were describing this, this graph, I noticed you used the word dependencies. You said we're going to capture the dependencies. Hm. So if you erase the red line between storm and thunder, I'd be happy to. So you erased that, should I read this as storms cause lightning, and lightning causes thunder. You can do that, but you would be wrong. Oh okay. You can not infer that there is a cause of relationship just because there is an arrow between them. These arrows are just telling us about the relationship between the probabilities and not anything about the physically processes that underlie them. Okay so let me make sure I understand, what you are saying is, it would be very natural to look at a belief network or a Bayesian net or a Bayes Nets or graphical model. And read the arrows as causes, and therefore read them as talking about dependencies. But actually what's happening here is that these things represent conditional independencies. So, it is not true that lightening is dependent on storm and thunder is dependent on lightening. So much as is the case that storm and thunder are conditionally independent given lightning. That's, that is a good point. I guess I never really realized that dependence. You use the word dependence. Sometimes it means a physical dependence. Like, in the real world it's dependent. Here I'm just talking about statistical dependence. It's really just talking about the fact that we can derive numbers from other numbers, and not that You know things cause other things. So yeah, that's a really good point. It seems like that was an easy place to get slipped up. Okay. Cool.

1.8.11 Sampling From The Joint Distribution Question

Alright, so now that we have a handle on this kind of representation, let's look at some things we can do with it. So, here's an example of a Bayesian network with five variables. A, B, C, D, E. And let's pretend that each one has some set of possible values. Could be true/false. Could be red, green, blue. Whatever it happens to be. And these arrows again tell us about our conditional dependence relationships. So how would we go about actually well, say sampling from this distribution? So let's say that we wanted to just as an example see what A, B, C, D, and E, might look like in a, in a randomly selected example from the distribution that this network represents. So turns out what we can do is that if we sample from A. Now A is specified has no incoming arrows so it's not conditioned on anything in particular so we can sample directly from A's distribution. We can do the same for B and now C. If we want to sample from C, we need to, make use of what values have already been selected for A and B. Because C is conditioned on A and B. But we can sample from that distribution. Each, each value of A and B, each joint value of A and B gives

a distribution over C. And we do the same thing for D and the same thing for E. And we're done. What we've sampled from is actually the probability distribution, the joint probability distribution. So does that seem like a useful thing to be able to do Charles? It does seem like a useful thing to be able to do. Yeah, so here's just a quickie quiz. So just write a one word description that says, well in this sampling you'll notice I went a, b, c, d, and e. What ordering do I need to do if I have a belief net like this specified by this graphical structure with the arrows? If I want to be able to sample it, I need to do it in a particular order. Some orders are, are going to be problematic because we haven't actually, you know, sampled the variables that it depends on. So, what ordering should we select for A, B, C, D, E? In general, what, what is the name for that. So that we can actually do this kind of sampling trick this way. Okay.

1.8.12 Sampling From The Joint Distribution Solution

All right Charles, so, so, what do you think the answer is here? Actually I don't know what you're looking for here. Oh, okay. Well, so one thing that's true. We had to sample the, the variables from A to E. Mm-hm. And that's alphabetical order. So do you think that's what I was looking for? Maybe in this case but I would think that that wouldn't be generally true. True. Right. So, yeah, alphabetical is not what I was looking for. So, there's it's a graph theoretic property that says we want to basically put the nodes in order, so that you always put the things that have incoming links that haven't been visited yet after the ones where you, they have been visited. Oh, so it is a lot like alphabetical or a lot like lexo-, lexicographic, but it's topological. There we go. Yeah, that's what I was looking for. So, topological sort. Which makes perfect sense. Right, and so this a standard thing that you can do with a graph, and it's very quick to, to actually compute one of these. It does depend on a particular property, though. Let's see. Topological only makes sense if you really can go from no parents to parents. So, it cannot be cyclical. You can't have arrows that take you back. So, E can't be a parent of A and also have A be one of its parents. That's right. So it must be acyclic. Must be acyclic, right. And that's going to be true in these cases, because we're always going to set it up so that in a, in a Bayes net, the variable that we're each variable depends on other variables. But they all, it ultimately has to bottom out. There can't by cyclic dependencies. So, it is a directed acyclic graph. So, what would it mean if there were cycles? I don't know. I don't know what to do with such a graph. It just doesn't mean anything at all, I guess. Yeah, I mean, there, there is a family of undirected models. Mm-hm. But we're talking only about the directed ones here. So, the directed ones yeah, it'd have to be acyclic for the, for the probability distribution to be meaningful. Well, that makes sense. I'm sure we could make something up, but this is, typically this is how it's done. It's, it's, we constrain ourselves to acyclic graphs. Well, if a Bayesian network is supposed to capture conditional independencies, then if you add cycles, that's like saying there are none, right? I'm not even sure what that means. I could make it mean something. So here, we, we want the probability of A, conditioned on probability of A. Well, maybe that's like probability of what, what A was one time step ago. Or it could mean that it, you know, that, that we've actually putting constraints on the joint assignment to all the variables. But, yeah, it's not really, it doesn't really, it makes things more complicated and that's not he model that, that is the typical one Okay, fair enough.

1.8.13 Recovering the Joint Distribution

So another important thing that you can do with this representation is recover the joint distribution. Remember a couple, a couple slides ago we looked at the issue of how can we go from the distrib, joint distribution to specifying what the probabilities are, the conditional probability tables, they're called, at each of these nodes. But we can actually go the other direction as well. 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. So the probability that that value of A would be taken times the probability that that value of B would be taken times the probability that that value of C would be taken, conditioned on those are the values that were chosen for A and B. So it's just like in the sampling case. Right, and that's much more compact a representation. That's a good observation, yeah. So how, if these were Boolean variables, how many values would we need to specify for the joint distribution in the standard representation, where you just assign probability to everything. Well if I ignore the fact that there are some constraints that we might be able to take advantage of, it would be 2 to the 5th, because there are five variables. Right, but here we've broken it down into smaller chunks so, the probability of A, it's just specified by single number. Probability of B is specified by a single number. Probability of C is specified for a single number for each combination of A and B. That's four of them. This also requires four values and

this requires four values. So this is really, what, it's like 2 to the 5th minus 1 I guess. Because, if I tell you the first 31 values, the last, the 32th value, it's just 1 minus the sum of the other. This is 14 numbers versus 31. You are right, it is more compact, 31 is bigger. Right but let's imagine that all of the variables were in fact completely independent of one another, then you would have 5, you would only need 5 numbers. It would be the product of the unconditionals. Yeah, which is what we'd get if we had kind of like just a set of weighted coins. If they're unrelated to each other, but each one has some probability of coming up heads, the probability of getting some, some particular combination like, A is heads and B is tails and C is heads and D is heads and E is heads. We could just break that down to the probability of the individual events. So then all of the, just like with the joint distribution where you have this exponential growth, because you need to know everything. 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. Right, so the fewer number of parents, the more compact the distribution ends up being.

1.8.14 Sampling

Earlier I mentioned sampling and I asked you whether that sounded useful, and you said it was. So, let's do a little exercise. Why? Why [LAUGH] is that a useful thing? Why is it good idea to be able to sample from a distribution? Well, because it's one of the two things that distributions are for. What does that mean? Well 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. Okay. That's a little bit circular in the sense that it didn't tell me why it was useful to generate them other than it's one of the things you can do. Well, you didn't ask me to actually make sense. But I mean, this is the, the thing that you use distributions for. Now why would you want to do that? Yeah. So, 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. So it's like flipping a coin, or I want to flip a coin and find out whether I'm going to get heads or tails. It would be nice if I can do that in a way that's consistent with whatever the underlying bias of the coin is. Okay, so yeah, if this distribution represented something complex, we might, you know, for whatever reason need to simulate that world and, and act according to those probabilities. So, yeah, that, that's a reasonable one. What else, what if, what if I showed you this, if i took this distribution that we used for the lightning and thunder example. Mm-hm. What if you wanted to get a handle on it? How can we use sampling for the distribution to give you some insight into how the storms work? Okay so let's see, I've, I've, I've got this representation of the joint distribution, but it's just a representation of the joint distribution. If I want to asked a question like, well what's the chance that it's, oh let's say, storming outside if I've heard thunder, I could go through and, and, you know, back compute the reverse of the conditional probability tables. And I could do things like, or I could just generate a bunch of samples where I had thunder and I can just see how often the storm was also true. Does that make sense? It does, though I'm not going to use the words that you just used to write that down. Okay. I'm going to call that approximate inference. 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. So yeah, that, that turns out to be a really good way to do it. In fact, sometimes I think that's a lot of what people are doing when we're, 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. Hm. So, how would you do that? How would I do what? How would you do this approximate inference? We're going to get to that but I wanted to. Oh, okay, cool. But there, but there's one or two other things about sampling that I wanted to mention. Okay. Another thing that I could imagine using this for is this notion of visualization. Which may be, I mean this in a, 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. So, I bet if I was to run that if I was to draw a bunch of samples from the lightening thundering set, you would have a better feel for how likely different things are. Just you as a person might get a sense of how these things work. So, you can imagine in, in a medical domain a doctor who's, who's thinking about prescri, prescribing a particular kind of drug for a particular kind of person, if the information about drug interactions and so forth was, was represented as a big belief net, it might be hard to look at it and know anything. But if it ge, if you use that to generate a bunch of artificial patients you might start to get to feel for oh, you know what, these kinds of people tend to react badly in these kinds of circumstances. That's

still a kind of approximate inference, right? That's right. So this is, this is a kind of an in the machine sense, and this is kind of in the human sense. Okay, I like that. So let's see, let's see if I, if I understand this. So the, the nice thing about the storm, the thunder, and the lightning example is that it has pedagogical value. Because it's easy for a student to look at that and go okay, I understand what's going on here. One because there's only three nodes and two arrows, and the other is because, we think we understand how storms, thunder and lightning work. Right. Yup. Or most people do. So that makes a lot of sense. Of course the downside of it is, we think we understand it. And so it's hard to see why you would need to do samples, I mean, there's just a couple of probability distributions and we kind of know what it means. But in the real world, there are perhaps hundreds and hundreds of variables with complicated relationships and conditional independencies that, that aren't necessary intuitive just by looking at the graph. And 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, you know, really glock all the 25 different conditional probability tables. Does that sound right? Is that. [CROSSTALK] Yeah, yeah. What you're trying to say? That's exactly right. Thanks. Okay. I want to draw your attention to this, this word here for a moment. This notion of approximate inference. Now generally we don't like approximations when we can do things, things exactly. So why are, why are we not doing things exactly? because it's hard. It's hard, that's exactly right. So or, or, even if it weren't hard, it may, it may be in some cases faster. So I would be, I'm not going to do it now, but I'd be happy if I guess if there's ground swell of support among the students. To I can go through the argument as to why this inference is hard. There's a nice little reduction to problems, N, NP complete problems like satisfiability. But it turns out roughly that if you could do inference exactly on any belief net that you want, then you could solve very, very hard problems efficiently using that idea. So it's, it's cute, but it's kind of takes us a little bit off our path, so I'm not going to get into that. Okay, so sampling is useful, Michael, which I always suspected in my heart, and now we've got some good arguments for why it actually is.

1.8.15 Inferencing Rules

So, okay so let's, let's actually do some inferencing just to, to kind of get a feel for it. For certain kinds of networks we can do things exactly. And we're going to look at one of those examples in just a moment. But, it turns out, helpful to remind ourselves of some rules of probability in inference that will help us do that. So, here's just kind of a little cheat sheet. For you, so, marginalization is this idea that we can represent the probability of, of a value, at, by summing over some other variable and looking at the joint probabilities of those. And if, if you've trouble remembering this one, this, this's how I like to think about it, if we're trying to figure out the probablitiy of x, then one way, one thing we can do is break it up in. Break the world up into, well 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. The 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 we drop that then what is that implying? Just go ahead. Well, if you drop that then it implies that they are completly independent of one another. Right, 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. And as I recall, the order on the left doesn't matter, so, you have the probability of X times the probability of Y give X, but you could have written the probability of Y times the probability of, X given Y. Yes. And, actually, let's do a quick quiz. Okay.

1.8.16 Inferencing Rules Quiz Question

All right. So, 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. Go.

1.8.17 Inferencing Rules Quiz Solution

Did you get it? Yeah I did actually. so, so this one I think I understand completely. 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 of 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. Good. So it has to be the one on the sec, the second and just to make sure if you look at the second product the probability of x given y the second multican? Is it multican? Hm, factor. Factor? Let's say factor. The second factor, this 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. Yeah. So this is actually just one way you could just 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. So that's exactly right, the second one. Right. So this, this, so this makes sense to me. This is why when you look at a network, a Bayesian 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 it's again it's just the way the probabilities are being decomposed. Hm. And the last of these three equations just Bayes rule, this time written correctly where the denominator has to be the probability of x, and we've gone over this a couple of times. I don't, I don't need to, to describe it again, but what Would like to, just, bring to your attention to this three together turn out to be kind of our, you know, three musketeers in working out the probability of various kinds of events. Excellent.

1.8.18 Inference By Hand Question

All right. So 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. Let's imagine that we've got two boxes. Onee 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. And 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. All right. So the draw that we make, we have a green ball. 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? So let's, to make, maybe to help point out how this is connected with Bayes net inference, Charles, why don't you help me draw the Bayes net that corresponds to this problem. Okay. So, if I think about it as a process, which now means I'm, I'm thinking about this as things causing the other, the first thing that you did in the process is you picked the box. Good. All right. So let's say, so the first variable in the net is going to be the box variable., Right, and then once I had the box variable over there, I can then pick, the second thing in the process is I pick a ball. So, in this case you're calling it 1. So I make the first pick. And is it, do we need an arrow there? Yeah, because the, you pick the box and then that let's you pick which ball that you have. So, which ball you pick, the color of the ball you pick, depends upon the box so to speak. Good. And so, the probabilities here are going to be, it's going to look like this. All right. So the second variable here is what, what color ball you get when you do the first draw from the box. Ad 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. So for the second draw, well, clearly, that sort of 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. Okay, so now 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. Sure. But it also depends upon which box you're in. So you need an arrow from there as well. And what would be really nice is if we were in the storm, lightening and thunder case where, if I knew that it was, what ball 1 was, I would know what ball 2 was, but that's not true. Because in a case, for example, when ball 1 is green, it doesn't tell me what ball 2 is unless I also know which box I'm in. So, we have to draw the arrow from box to ball 2. Indeed. Right. And so there's a lot of, a lot of probabilities that we have to write down. But let's, let's just write down a piece of that table. 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. hm. because that's what we're ultimately going to need here. So now ball 2, in the case where we were drawing from box 1, that probably that's green. In the case were the first ball had been green, it leaves just 2 out of 3, right. hmm. And 1 out of 3 yellow and no blue. But on the other hand, had we drawn from box 2 first, and again, we had gotten green, now it's green one fourth, zero yellow, and blue three quarters. RIght. And there's yeah, we need this same thing where the other case, where ball 1 is yellow and ball 1 is blue. But we are not going to need those numbers for this problem.

Right. All right. So now that we have written it as a Bayes net, is that, is that helpful at all? So what we're, we haven't asked the question yet. So maybe it's time to ask the question and then we could work on the answer. Okay. All right. The question is, what's the probability that the second draw is blue, given that the first draw had been green? Go.

1.8.19 Inference By Hand Solution

All right, so can you use this Bayes net to help work things out? Yeah, actually it make it a lot easier. I was, I was thinking about how I would do this and, and wouldn't involve writing a whole lot of equations and doing a whole lot of stuff but actually, just by writing out the Bayes net we ended up, and filling out these tables we ended up doing that. So, the, the bottom table is, basically tells me the probability of, ball two being some color. In a world where ball one is known to be green. Because we just broke down that part of the table, so we don't have to do it for every other one. And, you know, if I knew that I were in box one, then the probability of it being blue in a world where ball one was green is in fact zero. And if I knew I were in box two. Then the probability of it being blue in, where ball one is green, and where box two is three quarters. So I only care about that last column. All right. And now I just have to choose the row or choose how to distribute the likelihood over the row. So all I really need to know is, what's the probability of me being in box one and being in box two. All right, which we have in the table as well, as a half. Right. So that means the probability of it being ball two. Being, ball two being blue in a world where ball one is green, is just the probability of ball two being blue, given that ball one is green. And we want to know the probability two is blue given that one is green but when you look at the table and all we care about is that last column, all we really want to know is, well, we know the answer when box one, when we're in box one, when box equals one, it's zero, and we know the answer when box equals two, it's 3 4s. So if we were going to do a sample, for example, which we talked about earlier, we would just sample a bunch of times, and we would get 0 sometimes and we would get 3 4s sometimes. And that would be great, except of course, we want to compute this exactly. And we know how to compute it exactly, because we actually know the distribution over, how many times box would be equal to 1 and how many times box would be equal to 2. It would be half in each case. So, I really like, I think you've made this easier by giving us the table. So, actually writing out the Bayes net. So we want to know the probability that the second ball is blue given that the first ball is green. And that's just equal to the probability that the second ball is blue. Given that the first ball is green and we were in box one. Because if we knew that, we knew we were in box one and the first ball we drew was green, it'd be really easy to compute the probability of the second ball being blue. It's right there in the table at zero. Is this, is this the way that you think it should be written? Almost, but not quite. That would be the easy thing to do because we know that answer. We know the probably that box is equal to 1. It's just a half. But it's not just the probability that box is equal to one, it's the probability that box is equal to one in a world where we knew the first thing we drew was green. Gotcha. And if we had that then it would be easy to figure out the, the products there to figure out two is blue in a world where the box one is green. Boxes equal to 1 and the first ball that we pulled was equal to, was green. And then we will just add that to the probability that the second ball we drew was blue. Given that the first ball that we drew was green. And we were in box two. We were drawing from box two. And that would have to be weighted by the prober-, probability that box was two in a world where the first ball that we grew, drew was green. Good. Very good. And in fact, this rule that you kind of worked through follows just algebraically from two of the rules that we just talked about. It's the combination of the marginalization rule, which let's us introduce this box variable. But the way that we wrote it before, it was, you have to and it in. But then we actually then applied the chain rule to split that into a conditional probability. So, so this is all valid at the moment. And are these quantities that we, that we know? Well, we certainly know the very first term in each of the two summands. Can it be summands? Let's say they're summands. If they're not, we'll get nasty emails from people. The first part's probability. Second ball is blue given that the first one is green in red box one. And the probablity that the second ball is blue given that the first one is green in red box two. That's easy, that's actually in the table. That's easy, that's in the table. And it's zero in this case, and three quarters in this case. Right, so it's zero in the first case and it's three quarters in the second case, straight outta the table. Now all we have to do is figure out how often we're in box one and how often we're in box two and if you didn't think it through you would just have the probability of box equals one and the probability of box equals two. But we have to remember we're in a world where the first ball we picked was green. So now we just have to compute each of those terms. So how do we do that? So we want to know what the probability is that boxes, we're in box 1 given that we picked a green ball first. Well that one's actually much easier to think about because Bayes' rule will give us, will allows us to express this in quantities where we do know the answer. Because we have the tables. So that would be the

probability that the first ball was green given that we were in box 1 times the probability that we're in box 1 divided by the probability that the first thing we picked is green. So, the probability that we get a green ball if we pick box one, is just well, it's three quarters. Yep. It's. A different three quarters than the other one though. Yeah. Those, those two three quarters aren't the same three quarters. This, this way. Because sometimes, two three quarters are not the same two three quarters. In this case, there are three green balls and one, what we're pretending to call yellow because it's easier to write than orange, ball. And so three of the four of them are green, so if we were in box one, we close our eyes, we'd get three of those. So what the probability that we're in box one? Well, it's right there in the table, to Bayes' net, it's one half. Now we just have to figure out well, what's the probability that I would get a green ball the first time I picked one? Right. And so one easy way to do that is, we actually do this, this whole process again on box two, and then just normalize. Or we could break this apart using the, using the marginalization rule. Yeah, which one do you want to do? The first one I think. Okay. So figuring out the probability the first one is green isn't, isn't as easy as it looks. You can't just say, well there are five green balls, but there's a total of nine balls, and so it's 5 9th, because those nine balls aren't distributed equally on both sides of the boxes. So you really have to, you still have to know which box that you're in, in some sense. Right. But we can kind of skip that step. Okay, so I like this, so what's the probability that the first ball is green given that we're in box two, well it's just 2 5ths. Prove by looking at the screen. And what's the prior probability that we're in box two? Well, it's just a half because that was given to us on the table. And so, we still don't know the prior probability of, of the first ball being green, but it turns out we don't have to because there are only two boxes and so we can just normalize and the right thing will happen. So, three quarters times one half is equal to three eighths. And 2/5 times 1/2 is equal to 2/10 or 1/5. And that's right. So 3/8 is also 15 over 40. 1/5 is 8 over 40. Why do we do that? Because we want to be able to add them up and normalize and so that means if you added those two together and put them in the denominator, that would give you 23 over 40. And, so how much is 15 40ths of 23 over 40ths well, it's 15 out of 23. And so, without ever directly computing the probability that 1 equals green. We know that the probability of us being in box 1, given that the first ball pulled was green is 15 over 23. Which was a lot of work to do considering that we knew we were going to multiply it by zero. [LAUGH] Which meant none of this work mattered. Okay. Or we did it because we love probability. No it was, it was kind of helpful because we needed to know how to normalize these two numbers. Right, so it was useful but, I mean, just the whole thing we already kind of knew. Yeah. That [LAUGH] that was going to be zero. But this one we didn't know. Right, this one we didn't know, and so now we know that the, the other case is 8 23rds, and we're done. So 0 times 15, divided by 23 is 0, and three quarters times 8 23rds is 24 over 92. Right, and we can, there's a factor of 4 in both of those. So it's actually 6 23rds. That's what I said. Woohoo! Wow. [LAUGH] Boy it would be nice if we had an algorithm to do this for us. Man, and the algorithm shou, shou, should not involve me. [LAUGH]

1.8.20 Naive Bayes

Alright, so what we'd like to do is work up to an algorithm that can actually do some of these inference steps instead of having to think it through each time de novo. So what I'm going to do is, let's hearken back to an example that we looked at before which is about spam detection. Do you, do you remember the spam example? I do remember the spam example. That was way back in the boosting lecture, right? Yes, I think you did that one. I did, it was an excellent example. There you go. So, we didn't think about it in a Bayes net setting, it was in a classification setting we were trying to come up with the rule, but let's think of this as a Bayes Net where there's a bunch of different variables that can be true or false about any given email message. It can either be spam or not. It can contain the word Viagra or not. It can contain the word prince or not. It maybe contains the word Udacity, or not. Mm. Right? And, so, just as we think about these as these random variables. If we're trying to build a belief net or a Bayes net with these variables. We have to say. kind of, what's dependent on what. In terms of representing the probabilities. So how would you, how do you think we should draw arrows to, to relate these to quantities to each other. I think that the arrows should go down from spam to the other features of spam mail and I'll tell you why. Because if, I like this notion of generation that you talked about a little bit earlier. It seems to me if you know. Spam mail or not. It sort of generates certain words. And as written as these are like words I mean I know the, the spam example these are you know, kind of stand ins for features. But they're sort of features of spam mail. Yeah I think that's a really good way to think about it. So, in some sense what we're saying if we draw the bayes net in this way, then any given email message has some probability of being spam. And given that it's spam, it has some probability of containing different sets of possible words. Right. So, I would say that, well what, so what do you, oh let's see if we can actually fill in some of these values. So given that we have a spam message, how likely do you think it would be to contain a word like, well let's say

the word viagra. Fairly high. It might be 0.3, but a non-spam message might be, I don't know, like 0.001. Right. Something like that. So how about a word like prince? Well I get a lot of email about Prince because I'm a Prince fan. Yeah, I was thinking that. That's why I thought it would an interesting example. So, if in your spam messages, how likely is it for Prince to come up? Fairly low. Maybe like 0.2 because you're talking about the Nigerian princes and whatnot. On the other hand among your non spam messages how likely is it for prince to come up, do you think? Well I get a lot of non spam, so, its still relatively low, but not as low as .001. Alright, so, let's say .1. Okay. That's a lot of prince spam. You can never have enough prince spam. Alright, so in the messages that you have that are spam, how often does the word Udacity come up? I guess, it's pretty low. I don't think I've ever seen a spam that mentions Udacity. Alright, what about your non-spam email? Again, increasingly, it's getting higher and higher. [LAUGH] Almost as much as I get prince mail. All right, so we'll call that .1 as well then. Okay. All right, so now we have, oh an, an what's the probability of spam versus not spam? [INAUDIBLE] Probability to have spam is pretty low, I'm going to say, at this point, actually; it's not that low. At this point, it's probably half my mail. Wow. All right, I'm going to say .4 Alright, so this is now, Bayesian network structure that actually is, it's not exactly generating spam, but it 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? Well, Since it says Naive Bays I think I would use Bayes rule. That would be naive of you. 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. Oh I do see that, so this is where those conditional independences come into play If I'm reading this network right, each one of those attribute values is conditionally independent of each other, given that you know the value of SPAM. Excellent. So then that means that the first quantity there is actually a product of each of those conditional probabilities. Yeah, so this is a really convenient structure. Because it really just decomposes into all these separate helpful quantities. So 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. The probability that's, Contains viagra given that it's spam, which we have. That number is 0.3. That probability that prince doesn't appear in it, given that it's spam and that is that it doesn't contain prince given that it is spam. So that should 0.8, cause 1 minus the 0.2. And that it's not udacity given that it's spam. Is going to be 1 minus this 0.0001, should be 0.9999. All right. So this is the case when things, when it is spam, and if it's not spam, we can do this same thing and get a product, and that we can normalize, to get what the, the relative probabilities between it being spam and not spam. So then I'm a big fan of normalization, but of course this makes me think about, since it's sort of a classification problem, we only really care about knowing which one's more likely. We don't really care about the probability, right? Do we have to normalize? Yeah, yeah because we do care about the probability. Oh we do? Yeah because we're... I asked" What is the probability of spam given these other quantities. Oh, I see. But you're right. So the observation that you're making is a really good one. Which is that 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. Okay. Because this this seems a little bit specific. Alright so the general form for this, is that if we're trying to figure out the probability of, of some kind of a a root node like this, when you have all these little bristly things coming down. You can think of it as a probability of a value given a bunch of attributes. And that's going to be equal to the product of the probability that each of those attributes would be generated by that. Underlying this v. This, this the label or the or the underlying class. Times the prior probability that v and then we just normalize by all the different possible values of, of v. This, this quantity across all the possible types of v. So so this is one way of actually getting a very general kind of inference done, and there's, as you were pointing out, Charles, there's a. There's a really nice reason to think about things in this form, because it does let you do a kind of classification. So essentially if you think of, 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, if we have a way of generating attribute values from classes. What this let's us do is to go the other way. That we observe the attribute values and we can infer the class. Nice, so what's the equation for that? Right, so the, the maximum oposterior class if you're just trying to find whats the most likely class given the, the data that you've seen. You can just take an arg max over all the different possible values of that, that root node of the prob, its probability times the product of all the attribute values given that class. So this would actually let us if you're, if you're been paying attention, we could, in this particular case, compute map spam. Which is a palindrome. Wow. That is spectacular. You did not see that coming did you? No I did not.

1.8.21 Why Naive Bayes Is Cool

So this idea of Naive Bayes, where you have a network that has a label producing or, or conditionally producing a bunch of attribute values, is just a really cool and powerful idea. So one of the, one of the issues is that, even though inference in general is, is is a very difficult problem it's NP hard. To work out what these probabilities are, when you have a naive Bayes structure, it's cheap. It's, it's 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 so far, we've only been talking about Bayes Nets in, in 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 odd, 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, 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. So this is, you know in, 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. Which is mostly what this, this mini course has been about. So, that's really great to have a connection, 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 on, in, in any of these directions. And it turns out it's wildly successful empirically. So, my understanding is that 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. So yeah so it's like unclear why we'd even have any other algorithms, right Charles? Well, there's no free lunch. But I, I gotta say I, I you know there's this as a famous man once said it works in practice but doesn't work in theory. And I'm trying to figure out how this can possibly work. So I noticed it's called Naive Bayes. And, I think I know why now. Alright. One is that it's well it's naive and in fact painfully ridiculous to believe that the bayesian net that you wrote up there in the upper right-hand corner represents the real world most of the time. Hm, I see, and why is that? Well because what the, what the network says is that all of the attributes are conditionally independent giving that you know the label, that just can't be true. We talked about this before where we were using Bayesian 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. So, one question I have. I have two, we'll save the second one though. One question I have is, how in the world can it possibly be the case that this works in practice? Hm, that's a good question. It does. Moving on. [LAUGH] No, that's not satisfying. No? How about, how about I give it a guess? Okay? Alright. Now, now that I yelled at you, why don't I, why don't I give it a guess. [LAUGH] I think it comes back to one of the conversation we had in the previous slide. When I was saying well we don't have to care. We don't care about probabilities. And you said we do care about probabilities because of the question your asking and that was fair. But once were down to classification. The probabilities really don't matter. Right all that matters is that you get the right answers. So its okay I guess if the probabilities you get are long. So long as they're sort, sort of in the right direction right. That you end up getting the, the right label as a result. Yeah, that's a good point. That in fact we're introducing this idea in the context of, of Bayesian Inference it might actually not be so good at that even if it is particularly good at classification. Oh, oh actually I think I have a good example so, so here, here write this down. So let's imagine there are four actually you can use the network that you have up there okay Good. So let's say that the first attribute, I'm just going to call it A and the second attribute I'm going to call B, and let's say we're really, we're really lucky and our naïve assumption is right and they really are conditionally independent. But let's say the third attribute, is actually just another way of writing down A, and the fourth attribute is just another way of writing down B. So, clearly there are interrelationships between the attributes, right? The third attiribute is the first one, the fourth attribute is the second one. There's not way around that. And so you'd think Naive Bayes would fail. But, actually, looking at your equation right below there where you're doing counting, I actually think, it'll work just fine. Why? Because all you're really doing is double counting the sort of weight of attribute A, but you're also double counting the weight of attribute B and they'll cancel each other out. And you'll get the right answer. When you do the arg max, but these When you do the arg max You get bad probabilities. The probabilities end up being kind of squared of what they should, what they're supposed to be. But that's okay because the ordering is preserved. Right, exactly. And so, even if you're unlucky and the fourth attribute wasn't B but it was something else, C. It doesn't matter if you double count A as long as it still gives you the right label.

And you can imagine that if you have weak inner relationships or, you know, you have enough attributes and, and so on that you would still get the right, you know, yes this is the correct label, even if you've got the probabilities wildly wrong. Okay, so I'm willing to believe that that could happen in practice. Okay. So in fact, my guess is that Naive Bayes believes it's answer too much. But it doesn't matter if it happens to be right. All right and did you have other issues with it? So the second problem I have actually boils down to that equation you wrote there. So it's really nice and neat that you can compute the probabilities of seeing an attribute, given a value by just doing counting. But, I don't have an infinite amount of data, right? Not on a bad day, no. No. Or even on a good day I usually don't have an infinite amount of data. So what if I'm unlucky enough that for some particular attribute value, I have never seen it paired with that label, V. Right. So then, that means this numerator will be zero Right. So. Well that numerator is zero, but since the computation involves a product by just having one attribute value that I've never seen before. I'm going to end up saying well the probability of that entire product of seeing that value given a set of attributes is also going to be zero. So one unseen attribute, basically says it doesn't matter what else is going on. Which seems a little weird, right? You, you, you'd think that you, if all the other attributes are screaming yes, yes, yes, yes, it should be positive. But just because you haven't happened to have seen any examples of some other one single attribute, that shouldn't be enough to do veto. Good point, so 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, there's smarter and less smart ways of doing that, but no, you're absolutely right. That, that is, that zeroing out problem is a real thing and you have to be a little bit careful. Hey, hey I just had a thought. So, if you, you have to do that, because if you don't do that, then you're believing your data too much. You're kind of over fitting. Ooh. Over fitting comes up again. Oh, oh, it's okay, okay so, so, so, so, so bear with me on this Michael. So if you're over fitting by believing the data, and you're fixing it by smooth, I usually spell it with a V, but whatever. If you, you'd think that by being smooth, then you're making an assumption. There's a kind of inductive bias, right? Your'e, you're saying that I go in with the assumption that they're sort of all things are at least mildly possible. Good. Huh. Yea, that's, that's right. Okay, Naive Bayes is cool, you've convinced me. Nice.

1.8.22 Wrapping Up

So I was thinking of talking to you more about sampling, but it seems like it might work out best to just have some hands on experience with it so we're going to put those things on the homework. So given that we're actually in a position now to, to kind of wrap up the whole Bayes net inference piece that we were talking about. So do you want to help remind me, Charles, what were the things that we covered? Sure, I can help you with that. We covered Bayesian Inference [LAUGH] I'm sorry. I'm punch drunk. I'm going to choose not to pay attention to that. Instead, write Bayesian Networks. We talked about the Bayesian Network representation of joint probability distributions. Right. We did a lot of examples of how to actually do inference with networks. You know, exactly how do we, do we compute probabilities of particular values. We mentioned sampling. That's right. And then we did a Naive Bayes. Well first we did say that, that in general it's hard to do exact inference. It's actually hard to do even approximate inference. Mm-hm. But we talked about a special case of bayesian networks, that was called naive bayes with the naive part being, that we're assuming that attributes are independent of one another. Condition on the label. Right. And this was actually helping us make a link between all this bayesian stuff. The bayesian rabbit hole we went down. And classification, which is the core machine learning topic that we've been spending a lot of time on. So the other thing that I really liked about this notion, this link to classification, Michael, is that when I was talking about Bayesian learning, what we ended up with at the end is this nice idea that we had a gold standard, right? We had a sort of way of talking about what the right hypothesis was and, ultimately, what the right classification was by computing these probabilities. And sometimes, we couldn't do it because, typically, you can't actually do the for loop that requires you compute conditional probabilities of hypothesis given data. Over say an infinite number of hypothesis, but at least we kind of knew what the right thing was and we made right assumptions we could do things like derive, oh I don't know, a sum of squared errors or various other things that you might do and that was all very cool. But what you've done here when you do inference. Is at least with a Naive Bayes case, you've shown us a way that we can do classification using these things, that actually is tractable, and is the right thing to do under certain assumptions. I really like that. And the other thing that I think is worth mentioning is that not only does it link this Bayesian learning to classification. But it connects classification back to this general notion of Bayesian learning, Bayesian inference where, you don't have to worry about just figuring out the most likely label given a bunch of attributes. But because it's a Bayes network and you can compute anything from it, you could try to ask

well what's the likelihood that I see some particular attribute or set of attributes, given a label or given a subset of attributes on all those kind of things that you could do. With the Bayesian learning. So inference gives us this power to not just do classification, but to do a larger set of things beyond classification. I think that's kind of cool. Cool. Yeah, well said. The, the For, and another thing, kind of in that same space is that it handles missing attributes really well. So whereas things like, oh. You know, decision trees and so forth, if you give me an example that doesn't have one of the attribute values and you've hit that part of the decision tree where you need to know that attribute value you're stuck. Whereas in this naive base setting, you can still do the probabalistic inference over the missing attributes because all the things are linked by probabilities. Nice. All right. So I think, you know, you'll, you'll get a much stronger handle of this when you go through the, the homework problems. But I think that's enough for Bayesian inference. And I think that actually wraps up classification and regression more generally. Right. So we're done with supervised learning. Well, one's never done with supervised learning. But we're at least done with this part of the course. Because there's always more to supervise learn. That's right. And in particular you'll get a nice example of this, because you'll be taking an exam. [LAUGH] And your input will be the exam, and then we'll give you a label back. [LAUGH] I guess that's one way to think about it. Well and then they'll get to generalize beyond that for the next time they take the exam. Very good! All right. Well, well thanks very much, this has been fun. Thanks Charles. This has been fun. I will see you in the second mini course. All right. Bye.

CHAPTER

2

UNSUPERVISED LEARNING

2.1 Randomized Optimization

2.1.1 Optimization

Hi everybody, welcome to our second mini course in machine learning. This sections going to be on unsupervised learning. That sounds fun. Hi Michael. Oh, hey Charles, I didn't even know you were here. That's because I'm not really there. Oh. [LAUGH] I'm glad to hear your voice regardless. Good, I'm always happy to hear your voice Michael, and I'm looking forward to hearing about un-dash-supervised learning. Well, unsupervised learning's going to be a series of lectures that we do. But today's lecture Is on PZTTM-NIIAOOI. Ok... Which is randomized optimization. I took the letters of "optimization" and I randomized them. [LAUGH] You're such a nerd, I love it! Okay. Alright so so the plan is to talk about optimization, and in particular to focus on some algorithms that use randomization to be more effective. But let's let's talk a little bit about optimization for a moment, because that's something that has come up, but we haven't really spent any time on it. So, what I'm thinking about, when I, when I talk about optimization, I imagine that there's some input space X, which is, you know kind of like in the machine learning setting and we've also, we're also given access to an objective function or a fitness function, F, that maps any of the inputs in the the input space to a real number, a score. Sometimes called fitness, sometimes called the objective, sometimes called score. It could be any number of things. But in the setting that we're talking about right now, the goal is to find some value, x*, I didn't mean to put space, such that the fitness value for that x* is equal to or maybe as close as possible to the maximum possible value. So that's like [INAUDIBLE] like we were doing with [INAUDIBLE]. Yeah, exactly, right. So of all the possible x's to choose, would we like to choose the one that, that causes the function value to be the highest. Okay. Yeah, I, I wrote it this way, though, because I thought it would probably be helpful if it's like, yeah, because we'd be pretty happy with the with an x* that isn't necessarily the best, but it's like near arc max, right? Something that's close to the best. Okay. So, so this is a really important sub-problem. It comes up very often, I was wondering if you could think of examples where that might be a good thing to do. Like in life. Well, Like in life. Computation, which is life. Computation is life. And life is computation. Well, believe it or not, I was actually just talking to someone the other day who's a chemical engineer and works at a chemical plant. And he says there's all these parameters they have to tune when they mix their little magical chemicals. And if they do it just right, they end up with something that, you know, is exactly the right temperature, comes out just right. If they're wrong at all, if some of their temperature is off a little bit or anything is wrong, then it ends up costing a lot of money and not coming out to be as good as they want it to be. So, you know, factories and chemicals and optimization and parameters. Factory, chemical. I'm not sure that's a general catergory of problem but I guess, I guess, maybe the one way to think about it is We'll call it process control. So if you've got a process that you're trying to put together. And you have some way of measuring how, how well it's going, like yield or cost, or something like that. Then you could imagine optimizing the, the process, itself,

to try to improve your score. Okay. Yeah I think that's what it is. You know route finding is kind of like this as well. Right. So just. You know, find me the best way to get to Georgia. What about root finding? Oooh, I see what you did there. Mm mm. So you could think of root finding as being a kind of optimization as well. If you've got a function and you're trying figure out where it crosses the the origin. You might add you might set that up as a optimization problem. You might say well of all the different positions I could be in, I want to minimize the distance between the axis and the value at the point where I chose it. Find me the best one. Which, you know, is going to be right there. Actually, you know, when you put it like that. How about neural networks? Nice. So, that's, yeah let's get it back to the learning settings. So, what is, what do you mean by neural networks? Well, I mean, everything we've been talking about so far. X, you know is some kind of stand in for parameters of something. A process or I don't know whatever. So if the weights are just parameters of a neural network, then we want to find the x that I guess minimizes our error over some training set, or, or upcoming test sets or something. Yeah, so minimizing error is a kind of optimization. It's not a max in this case, but if we I guess, negate it you want to maximize the negative error, that it's maximized when the error is zero. Right. Cool, is any other learning related topics that you can think of that, that have optimization in them? Well I would guess anything with parameters, where some parameters are better than others and you have some way of measuring how good they are, is an optimization problem. So, decision trees? Sure. So, so what's the parameters there? There the order of the nodes, the actually nodes, like what's the first node? Yeah, the whole structure of the tree. So it's not a continuous value like a weight, but it is a structure where we could try to optimize over the structure. There's nothing in the way that I set this up here that makes it so that X has to be continuous or anything like that. We just need a way of mapping inputs to scores. Okay. So all of machine learning. Everything we did in the first third of the class is optimization. There is some optimization in each of the pieces of it. Yeah, exactly. because often what we say, given this training set, find me a classifier that does wel on the training set and that's an optimization problem. Hmm.

2.1.2 Optimize Me Question

Alright! So, let's I mean, what we'd really like to get to are algorithms for doing optimization, and I think to do that it's helpful to do a quiz. So, here is a little a quiz that I put together, I call it Optimize Me. And what I want you to do is, is here's two different problems, there's two different input spaces, two different functions, and I want you to find the optimal x-star or something really close to it, if you can't get it exactly right. So, the first question we've got the input space x is the values one to 100. And the function that we're trying to optimize is x mod 6 squared mod 7 minus the sin of x. Holy cow. Yeah, it's, it is an awesome looking function. Do you want to see it? I, I might be able to plot it for you. Sure, I'd love to see plots, because then maybe I'll know how to figure out the answer. Yeah oh yeah well maybe I should not then, maybe I should wait until we already have an answer and then yeah, let's, let's wait but I'll show you after it. It's, it's, it's crazy. [Laughs] Alright it's beautiful. So okay good, so and the second problem we've got x as the set of real numbers, so all, any, any possible number in the reals. And the function that we're trying to optimize is negative x to the 4th plus 1000x cubed minus 20x squared plus 4x minus 6. Alright, so you understand the questions? I understand the questions, yes. Good, good luck with them.

2.1.3 Optimize Me Solution

Alright, everybody, welcome back. So, let's, let's work through these. Charles, what is, what are you thinking for, for this first one. I'm thinking I should write a program, and just run it through, 'because that's the easiest thing to do. So what you're saying is this f function is so kind of wacky that it's hard to imagine actually analyzing it, and doing anything with it. But, on the other hand. The X, the space of X values is very small. Right. Good so right so we can enumerate all the possible values and just see which one gives the highest output. Yeah that's pretty straight forward if I just wrote the code, in fact it's like one line of code. So I wrote it for you. Oh good. I'm nice that way. You are nice that way, you optimized our time. Our did, for what it's worth, this is what the function looks like. I'm not even sure that's a function. Isn't that beautiful? It's something. It looks DNA gone wrong. Yeah, totally wrong, there's just mutations everywhere. It's like a bow tie that then tried to hide itself in another dimension. I mean like, you can see there's this interesting up and down-ness to it like a, like the sine wave part. Mm. But they're, they're interdigitated with each other. It's all, it's just crazy to me. Interdigitated. That is an excellent word. Yeah, it means, it means to put you're fingers together. Hm. So, So what happens if you plot that but with, connecting lines? Oh, oh, hee, hee. Sure. [LAUGH] Wow. You get somethin' like that. Wow. This is a kind of unfriendly function. Yeah, this is not the friendliest function I have ever seen. And now it looks a bit like,

actually it looks a bit like an insane bow tie. Nice. Electric bow tie. Yeah. So it turns out that there's two points that are really really close to the top. There's this one at 11 and there's this one I don't know a little bit after 80 or so. But the 1 and the 11 is actually the largest. Hm. This function goes too loud. [LAUGH] It does indeed. So, I'm going to say 11. I wanted to make a crazy function that was the, was the first thing that popped into my head. Actually I did, the second one was a mod three at first. Hm, well, it's an interesting thing, right because if you tried to reason this out, it would be very difficult to do. I mean, first off, mod isn't very nicely you know, it's very easy, it's not very easy to deal with algebraically in closed form but you can start reasoning and say, well, the dominating factor is mod seven in the sense that you're never going to be greater than six and of course the mod six is never going to be greater than five so the best you can get there is five and then you minus the sign, which is going to be some small number so you can kind of pick all the values that are going to give you something like. Five there, and then whichever one has the smallest Sine, that's the one you go for. I wonder if that would get you anywhere close to the right answer. So, interesting that that's not quite true here because 11 mod six is, no 11 mod six is five, right? Hmm. Squared is 25. Mod seven is four. Is that right? Anyway. Hm, okay. let's do the second one. Okay. Alright so it turns out that, again, 11 and 80 are really close but 11 is slightly larger and it's just almost five, but not quite. Alright, so for the second problem, now we can't use of enumerating because our X's constitute the entire space of the, reals. So, there's like, 200 of them. More. So we need to do something else. And I think what you were pointing out to me is that because this is a nice polynomial. We can actually use calculus to figure out where the maximum is. Right, just take the derivative and set it equal to 0. So unfortunately, I made that a little harder than than probably you'd appreciate. So the derivative would be this, and setting it equal to 0 would give us this equation, which unfortunately is a cubic. Right. So it should have three solutions. Perhaps, and you could just check each of them. But solving the cubic is sort of a pain. Mm mm. I guess I could have been a bit nicer. You could have been. Sorry. But it all depends what you're optimizing. So how would you go about solving this now? I would go on Google and I look up how do you solve cubics. [LAUGH] Sure, that's not a bad idea. Any other things, because it is nice and smooth, right? So let's, here, let's, let me actually find it for you. So here's the function, and I just plotted a subrange of it, from minus 500 to 1000. You know you could ask, could it be that the maximum is actually someplace outside this range. You know how you'd answer that? No. So because it's a fourth order polynomial, like think of like a, a quadratic, Yeah. It's got the one bump in it, a cubic can have as much as you know one up bump and one down bump. And a qua, and a quartic ca, a 4th degree polynomial can have an up bump, a down bump, an up bump and then back down. M-hm. So we're actually seeing all the activity here, outside this range it just plumettes negatively. It, it can't turn around and come back up again. So because we can see the two bumps we are good. OK. And in this case, it looks like the bump that we want is somewhere between 500 and a 1000, somewhere in the 700 kind of zone and well so one thing we could do because it's really well behaved, we can kind of zoom in on the function at that range. Mm-hmm. So there we are now zoomed in from between 500 and 1,000. We can see, okay, well, actually it looks like the peak is maybe between 600 and 800, you think? I'd say 700 and 800. Between 700 and 800. All right, well let's take a look at that. Ooooo. Nice and Pointy. Alright. So how bout that? 740. How bout 740 and 760? So we really are kind of getting into the nitty gritty here. So, so, for the quiz we'll you know, we accepted anything, between you know, 745 and 755. But in fact we can keep zooming in and we can use calculus, and we can, we can really hone on what that, the tippy top is there. But that's not so important right now, we could also something called Newton's method, you know Newton's method? I remember it. Newton, newton's method said if, you know, guess a position like you know, 455 sorry 755, and use the derivative at that point which is actually really easy to compute, doesn't involve solving the, the cubic, it just involves, Writing down the cubic and evaluating at that point, we can actually you know, fit with the straight line is here. And we can take it, take steps in the direction of that line, it's a, it's a gradient ascend kind of method. Yeah. And, and that'll allow us to hone in, as we get closer and closer to the top. The slope is going to flatten out, we're going to take smaller steps And this process converges to whatever the peak is, I think it's a little bit under 750. That looks about right.

2.1.4 Optimization Approaches

So in terms of optimization approaches, we actually just looked at a couple of different ideas. We, 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. When is that a good idea and when is that not a good idea? Well, it's probably a really good idea when you have just a small set of things you need to generate and test. Yeah, it really does require that we have a small input space. And it's also particularly helpful if it's a complex function, right? Because there really isn't any other choice if the function has kind of crazy

behavior, like in the mod example that I gave. Alright, for things like calculus, like, just solving for what the optimum is analytically. When, when is that a good idea and when is that not such a good idea? Well, it's a good idea when you have a function where you can do that. Where, there, well, first off, it has to have a derivative. Right, and so, it seems like, well, how are you going to write down a function that doesn't have a derivative. Well for the thing we're trying to optimize is some kind of, it could be, crazy because we defined it to be crazy. We can define it with little, you know, like, ifs and things like that, to make it only piecewise continuous. Or it might be the, the, 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. Right, or if the inputs are discrete. Right, like in the first example, right. So it might not actually give us any feedback if the derivative is, is, if we're not moving in a smooth continuous space. Hey is mod, does mod have a derivative. Almost everywhere. Just not everywhere. Yeah. There's that, you know, because mod kind looks like ner, ner, ner, ner. So it has this nice derivative everywhere except for at these, at the jumps. Which happen pretty often. Yeah, there's a lot of them. But they're, you know, measure zero. Okay. Like, if you just picked a random number, it wouldn't be at the jump. So it's not only important that the function has a derivative, but the derivative, we need to be able to solve it, solve that derivative equal to zero. Newton's method can be helpful even outside of that case, where we have a derivative, and we have time to kind of iteratively improve, right? Just keep querying the the function. Creeping up on what the optimum turns out to be. So, okay, but then what do we do if these, these assumptions don't hold? So we have a real, what would that mean? It would mean, what, big input space. Still have a complex function. Complex function. No derivative, or difficult to find derivative. I knew I had that derivative around here someplace. [LAUGH] You know it's always the last place you find it. [LAUGH] Indeed. Yeah. Exactly. And, and actually I, I, I should have mentioned one more thing about Newton's method. Which is that the function has a derivative you iteratively improve. And it really 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 a, a curve say like, like that. because Newton's method is just going to hone in on the, the local peak. So that would be bad if you have lots of local maxima in this case, or optima in this case. Yeah, yeah, yeah, yeah. So, so that's, you can list this as well, possibly many local optima, right. The, the, 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. Makes sense. So, as you might expect that our answer to this hard question is going to be randomized optimization. The topic of today's lecture.

2.1.5 Hill Climbing

So here's an algorithm that you'd probably either already know about or would be able to come up with rather quickly. Which is the idea of 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 we, here the neighborhood might be, you know, 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 than we are now then move to that point. Otherwise, we stop because we are at a local optimum. So, what this is going to do is it's going to iterate moving up and up and up and up this curve, always in a better and better and better direction until it hits this peak here. Then, it's going to look on both sides of it. The neighborhood, everything in the neighborhood is worse. So, it just stops there and this is the x that it returns. Which is, you know, not bad answer. It's not the best answer. But, it's a good answer. Hm. What if you had started out just a little bit more to the left? Say, on the other side of that valley. Oh, like here? Yeah. Hm. So then, well, let's see. What would it, what would it do? We start there. We take a step. We, we say, what's the neighborhood? The neighborhood or the points, just a little bit to the left, and just a little bit to the right. One of them is an improvement. So it takes the improvement. And again, it keeps finding it more and more improved. And then it gets to this top of this little bump and it says, okay, both the points in my neighborhood are worse than where I am now. Hurray, I'm done. Hm. So, that's not even worst of the local optima, you could actually get stuck here as well, which is even lower. Well, you could get stuck anywhere. Well, at any peak. Yes, exactly right and that's the lowest of the peaks.

2.1.6 Guess My Word Question

Alright. I want you to guess my word, using this hill climbing approach that we've just been talking about. And unfortunately, well, we tried this will actual words and the space is a little bit big and frustrating. But what we can do instead is pretend that a five bit sequence is a word. So, I'm thinking of a word and by word I mean five bits. And what we're going to do is I'm going to give you, for each time you guess a 5-bit sequence, I'm going to tell you the number of correct bits. So in each position it's, it's, if you matched what I am thinking in that position, then I give you an additional point for that. Okay. And you're going to now step through the hill-climbing algorithm. Okay, so three things. 1, if you had done eight bits, that would be a word. And. Oh. Nice. 2, four bits is a nibble. Okay, here's the third thing. We need to define a neighbor function. So, I'm going to define a neighbor function. So, I can think about this. This is all one bit differences from where you are. That's good and, and an interesting question is, was it your job to come up with that or was it my job to come up with that? Now is it, is it part of the problem or is it part of the algorithm that's trying to solve the problem? Huh. I'm going to say it's a part of the algorithm that's trying to solve the problem. That's how I think of it, too. Alright, so, we have to start somewhere, so let's just pretend for the sake of argument that we start it at all five zeros. And I'm going to give you a score of two for that. Okay. And so now I have to do all possible neighbors, and there are five of them. So there is one followed by four zeros. 01 followed by three zeroes and you know, the rest of the identity. Alright, here is all your neighbors for the x, you're aware. Mm-hm. And here is the scores for all of those neighbors. So, there are three of them that are maximal but for the sake of simplicity and drawing, I am going to stick with the first one. So you can get rid of the bottom four. Alright, so here is our new x. Okay. And now I have to try all different. So one of the neighbors would be all zeroes. But I already know what that is. So I don't have to worry about that. That's good. So the algorithm, the pseudocode that we wrote didn't do that, do that optimization but in general if as long as the neighborhood function is symmetric then you can save yourself a little bit of effort. Right, so, or we can just keep track of everything that we've ever seen, although that gets very space inefficient very quickly. And since it's the identity maker's [UNKNOWN] with the one matrix. [LAUGH] Okay, so what are the numbers to that one? Wow, that's pretty cool! So, you said we're getting there very quickly. So, it's going to be, so, [CROSSTALK] In fact, we now, have enough information, that smart human being could actually jump to the, optimum. Yeah, I think we do. I was actually thinking about that. Let's see what happened.

2.1.7 Guess My Word Solution

All right, let's figure it out. well, so, first off, I think the first thing, if I'm right is. So, is the following statement true? Once we know four of them are right, we will stumble onto the right answer when we do the neighborhood. Well, so, okay. So, that's a good idea. We could, actually, do one more step. Mm-hm. And see what happens. So yeah, because, in this particular case, that because of the fitness function is the way that it is, the number of correct bits. There's always a neighbor that is one step closer to the target. Right. So we're always going to be incrementing as we go. Mm-hm. So this is a very friendly fitness function. Mm-hm. It's only, only a global optimum. Right. So, do you want to do that? And then maybe we could talk about how we could have been even more clever? So, let's just pick the first one. Okay. And so, now we have to do the fourth one, so it's 10110, and then the fifth one is 10101. Okay. All right, so this if is x and these are the neighbors of x, these are the scores. And it turns out that one of them actually has all five matches. So that was in fact the pattern I was thinking of. Here, just to prove it 10110. And if we now continue the search from here, all the neighbors are going to be worse, obviously. Right. So, how could we have figured out the sequence without that extra information that I just gave you? Well, if we look at the two that have four, we can basically just see that where the, the, they must have three in common. Which they do. one, the first, the second, and the last one. Okay. And each is off by exactly one. So, that means that, well, since I know the answer, I know that you just take one, you just take one from each. So either they have to both be ones or they both have to be zeros. All right, and so do we know that the 01 doesn't work? We probably know that the 01 doesn't work. because that, then the answer would be 10000, 10000. Which we already have. Which is exactly, we already know is a three. Right. So then it has to be 10110, which in fact it is. Which it was. Cool Right, so this was in fact a very nice fitness function. and, and, and, you know, it seems to me that we had a, a, nice little advantage here in that we really understood the structure of our space. But if this were a 5000 bit sequence, where the fitness function was some weird thing. Where, in fact, we didn't even know what it was. We just knew that there was one. Then, that would've been a whole lot harder. actually, here's a question for you, Michael. If I had told you, if I had given you this, but I haven't, didn't tell you what the fitness function was. Just that when you gave me a sequence, I evaluated it for you. Uh-huh. We wouldn't have been able to do any of the stuff that we were doing. I mean, we'd still

be able to do it, but we wouldn't have been able to jump ahead. Like, we just, like we could've done with those two that were four, because we wouldn't even know what the maximum fitness value was. Yeah, we would have to done this last step to, to find that one of them has a score of five. And then we would have to done that step again to make sure that there wasn't anything locally better than five, because we wouldn't of known that if we didn't have a kind of semantic understanding of the fitness function. But, but the rest of the algorithm was perfectly implementable without knowing that, right? All, all that you were basing your decisions on was, you gave me bit sequences, I gave you scores, and you used those to figure out what next bit sequence to ask. It, it didn't use the fact that the fitness function had a particular form. That's right. That's right. Now, this wouldn't, this particular problem is very, very well behaved, because it has one global optimum, this bit sequence that has a score of five. Nothing else had a five. And once you're at five, and, and no other bit sequence actually could you be stuck. Right, so it was always the case that you could always get closer to the target pattern, so your score can always go up. But, in general, that's not how things work. That you may actually be able to get stuck in local optima and we need a way of dealing with that.

2.1.8 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 are some places that this could get stuck? There, there, and there. And others to boot. 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 like, you know, sort of what you do if you were trying to solve a problem and you got stuck. You're like, okay let me just try to solve it again. So why is this a good idea? Hm. Well one it, it, it takes away the luck factor of I happened to pick a good starting place. Although I suppose it replaces it with the luck factor that I randomly happened to pick a good place. But that's okay because I'm going to keep randomly picking good places. Or randomly picking starting places Yeah. 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, you know, a factor, a constant factor, which is how many times you are willing to do a random restart. Yeah, that's a good way of thinking about it. That it's, it's really just random hill climbing, repeated how every many times you feel like you have time to repeat it. And it can actually do much, much better. Now, if there really is only one local. Sorry, if there is only one optimum and there is no local optimum, then what's going to happen when we do random restart hill climbing? We'll just keep getting the same answer. Yeah, over and over again. So, could be that we, that we might keep track of that and notice, okay, you know, what we seem in a space where these random restarts aren't getting us any new information. So, you might as well stop now. Well, that's one answer but I could think of another answer. What's that? So, maybe you just keep starting too close to the same place you were starting before. I mean, it may be random, but you can get unlucky in random right? 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. Yeah the randomness should do that, at least on average. But you're right, so we could try to be more systematic. So here might be an example of a kind of function where that would be really relevant. So imagine we've got, 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, the real hill. The top that we really want. The optimum. So yeah, we, we might not want to give up after a relatively small number of tries because it could be that this tiny little, I know, I'm going to call it a basin of attraction. Mm-hm. I like that. And if it's small enough, it might take lots of tries to hit it. In fact, it could be a needle in a haystack. In which case, there's only one place that you could start that has that optimum. And that could take a very, very long time to luck into. In fact, the hill climbing part isn't doing anything for you at that point. Yeah, but you know. If you're in a world where there's only one point that's maximum. And, but there's only one way to get to it by having to land on a single point. Then, you're in a bad world anyway. [LAUGH] Right, I mean, there's going to be, nothing is going to to work out. Nothing is going to help you in that world. Yeah. Right, and in fact, I would, I would claim since everything has some kind of inductive bias. Right you're, you're making some assumption about like local smoothness or something. that, that makes sense here because if you, if you are worried about a world where there is always some point of the infinite number of points you could be looking at say. That is the right one and there is no way of finding it other than having to have stumbled on it from the beginning.

Then you can't make any assumptions about anything. You might as well look at every single point and what's the point of that. Fair enough. Mm-hm. So the, the assumption here I guess is that there, you can make local improvements and that local improvements add up to global improvements. Right. Hey I got a question. I got a semantic question for you, a definition question for you. Sure. You decided that you didn't, if you only had one optimum, you didn't want to call it a local optimum. So, is a local optimum, by definition, not a global optimum? So, if I was being mathematical about it, I would define local optimum in a way that would include the global optimum but it's just awkward to talk about it that way. because it feels like the local optimum is someplace you don't want to be and the global optimum is someplace you do want to be. But yeah that's right the global optimum is a local optimum in the sense you can't improve it using local steps. Okay.

2.1.9 Randomized Hill Climbing Quiz Question

I think we can kind of get at some of these issues with a randomized hillclimbing quiz. So what was, kind of what is the advantage of, of doing hillclimbing and, and what does it lead to in terms of being able to find the maximum. So, so here, [LAUGH] yeah, Charles, this is going to be a quiz, and I, I guess you're not particularly happy about it. But let's, let's take a look here. We've got an input space of 1 to 28, the integers. And so here it is kind of out on the screen, all the values from 1 to 28. And here's the fitness function, or the objective function for each of those points. It sort of follows this, not exactly sawtooth, kind of piecewise, jaggy, I don't know what to call it. Drunk. [LAUGH] It is what it is. [LAUGH] And, here's the the global optimum. And what we're going to do is, we're going to run randomized hill climbing. And we're going to run it this way. We're going to assume that the algorithm chooses randomly, in both, if both directions improve. So it just flips a coin 50-50, then it does whatever evaluations it needs to do to figure out which way to move to go uphill. We'll use as of the neighborhood of x to be the thing immediately to its left and immediately to its right, unless of course you're at 1 or 28, in which case there's just one neighbor. So we're just going to use a simplification of the hillclimbing that we had before. Which is, it's going to check the neighbors, check both the neighbors if there's two of them, and if one of them is an improvement, it's going to go in that direction. If neither of them improvement then it's going to declare itself at a local optimum. And if both are an improvement, it's just going to flip a coin to decide where to go. And once it gets to the, to the local optimum, let's say it sort of climbs up this hill and ends up at this peak here, it's going to realize that it's at the, at a local optimum, and then it's going to trigger random restart and pick a new x position to start at. And now, the question is, how many function evaluations on average is it going to take to find x star, the, the tippy-top of this peak? So, you know, maybe you could write code to do this, or maybe you could do it with math. This is probably a little bit more involved than most of our quizzes, but it should give you a chance to really dive in and get a feel for this notion that it matters kind of where these basins of attraction are. Alright, you think that's clear enough to give it a shot? Sure. Alright, let's go.

2.1.10 Randomized Hill Climbing Quiz Solution

All right, so there's there's a bunch of ways you can imagine doing this. And here's, here's how I would suggest looking at it. So, for each of these 28 positions, there's some number of steps that it's going to take before it either resets, that is to say, you know, re, re, re-chooses at random, or has actually reached the tippy top. So, let's, let's actually evaluate these. So if you, if you happen to start at 1, how many steps is it before you realize that you, that you're at 1? You have to evaluate 1, you have to evaluate 2, you have to evaluate 3, and you have to evaluate 4 to see that you're at local optimum. Mm-hm. So, there's going to be 4 evaluations if you start at 1. If you start at 2, then it's going to be 1, you're going to have to evaluate 1, 2, 3, and 4 again. If you start 3, then you only need to evaluate 2, 3 and 4 to know that you are at a local optimum. Hm-mm. If you start 4, then you have to evaluate 4 and then if you randomly step in this direction, you are also going to need to evaluate 2, 2 and 3. All right. So then, I, I finished writing down what all these different numbers are. For each, at each position here, I wrote down in red, how many steps it's going to take before it hits a local optimum including the ones here around the actual peak. And I wrote two numbers for the places that were these cusps where it can go either way. So, now, we can actually work out what the expected number of steps to the global optimum is. It turns out for 22 out of 28 of these points, the average 5.39 before they discovered that they're at a local optimum, at which point we have to start the whole process over again. And so whatever the expected value of reaching the local optimum is, we have to, we incur that same cost again. That makes sense. Good. Okay. So, for 4 out of 28. These guys here. Then, they're going to take 4 steps on average and end up with the peak. Then, the only other things that left to figure out or what happens at these two weird cusps where just a flip. It's going to flip a coin and either

get stuck on a local optimum that's not global or go to the global optimum. So, in 2 out of 56 of these cases it's going to have chosen to go to the local optimum. The non global local optimum. And it just so turns out that it's 10 in both of those cases. It's 10 steps before you realize you're stuck again. At which point it's going to start the whole process all over again because it's stuck. Then in 1/56th of the cases, we're going to be here and choose to go to the right and, and take six steps to get to the global and here 1/56th of the cases we're going to take five steps to get to the global. Okay, so just to be clear, so the way you got 28 is that there are 28 numbers and you uniformly chose where to start. Yep. Where you got 56 is, you decided that for two of those numbers 15 and 20 you end up getting them 1/28th of the time. And then half of the time you do. The local optimum and half the time you end up doing the global optimum. And it just turns out that in the case, in the cases where you end up going through the local optimum, those both happen to take 10 steps. And that's how you got 2 out of 56. And the last thing to notice is that for point number 4, where you could go to the left or to the right, since you do each half the time, it's like saying the number of steps there is just uh,11 over 2. On average. Yeah. Very good. So that was folded into this 5.39 number. You're right. Thanks for pointing that out. Okay. Sure. So this all makes sense and obviously if you just add those all up you can solve for V and the answer is [COUGH] exactly. So, just algebraically simplifying this equation gives us V equals 5.36 plus 0.82 V. Solving for V gets us 29.78. So, it only takes 29.78 function valuations before we first hit the maximum. Ta-da. Michael? Yes sir. So, that seems unfortunate. Because here's a really simple algorithm, for i equals or, hey let's say x. For x equals 1 to 28. Compute f of x. If it's bigger than my previous one, then hold on to it. And then just return whatever is the biggest number I see. So, that's linear and that's 28 evaluations, which is less than 29.78. Yeah, that's a good algorithm in this case. But you, I, I guess your point is that this, this whole idea of hill climbing and using this, this local information is actually costing us a little bit compared to just enumerating all the possible inputs and just testing them off. Right. Which I guess makes sense because the numbers are really small and [UNKNOWN] and you have a whole bunch of local optima, and so it's going to take you a long time to luck into the right one. But, it seems kind of sad that there's nothing clever you can do here to do better than 28, which is kind of what you want to do. Well, there's a couple clever things we could do, right? So one is, we could random restart faster. It turns out [INAUDIBLE] magic [INAUDIBLE]. So here's a question. What happens if you randomly restart after each function evaluation? How many function evaluations do we have to do, on average before we reach the maximum? I don't know. Oh, well, I guess it would be less than 28. It would be exactly 28. Oh it would? Yeah. 28's less than 28 so that's, that's fair. It's less than or equal to it, sure. Yeah, yeah, that's what I meant. And even better what happens if we actually keep track of points that we visited in the past? Then we don't pay the cost for re-evaluating them. That's right. So, then the only question is how long does it take us to luck up into getting between 15 and 20. Yes. Well, once we. Yeah, once we fall into that zone, then we're going to be done shortly afterwards. Right. so, how much of a, how much of a win is that on average? That's a good question. It's a little bit hard to tell. Because what's going to happen is if we follow this algorithm with the additional thing, additional attribute that says. If you already know the function valuation from som, from some previous iteration. We just reuse it, we don't get, we don't pay the cost for that. Then what's going to happen is we're going to get dropped off in random places in this space. We're going to hill climb visiting some things along the way. And possibly get drops in place where we've already done the function evaluation. Right. And therefore don't have to pay that cost again. But it's not as simple as saying how many different hops do we take before we land in this basin because if we land in say this basin here, the second one from the left. If, if we you know get dropped onto point 14, it's going to stay in that zone for a while, which is actually kind of bad because it's taking lots of function evaluations in that same part of the space. But look. But what you just said is important, right? Since we're only, we're only going to pay the cost of doing the function evaluation once, that means in the worst case. We would end up hitting the three bad basins before we hit the fourth basin. In fact, we would end up hitting the three bad basins enough times that we would end up visiting all of the points in them. So we'd land in 0.14 which would get us all the way over to the peak and just to the left, and then we would land in 5.5, which would get us to the right, so we cover everything in the basin. So, the worst case is we cover every single point in all three of the basins before we get lucky enough to land in the, the good basin. Right. And then. But, the moment we land in the good basin, we will get to. Am I right? We will get to avoid at least one of those numbers. No. Not necessarily. So, for example, if we start at 15 and we randomly move to the right. To know that we're at the optimum here, we have to check the point 19. So we've actually touched everything. No, we didn't touch 20. Well, we may have already, unluckily, hit 20 because it's part of the right hand basin as well. That's true. But anyway, the point is that you're right. If we keep track of which points we've already visited, then we can't do worse than 28. Like, even if we're really, really unlucky, we can't do worse than 28. Right. Maybe, maybe on average it will be a little bit better than that. But but only one or two. But it will still be better. I think it still be better. On average, on average it will be better. Yeah,

its true. I think, I think in this case it's, it's just a contrived example with lots of local optima and just a linear space so that there's a the, the attraction base ends up being relatively small compared to the size of the space. Hm. Okay, that makes sense. So, what, what should I learn from this? So I guess what I learned is randomized optimization or randomized hill climbing, here, may not do better than evaluating all this space in the worst case, but it won't do any worse. And every once in a while, you will get lucky and do much better. At least if your cost is function evaluations, as opposed to steps. That's right. Yeah and also, well and I, and, the thing that I was hoping, that you'd notice is this idea that randomized hill climbing, depends a lot on the size of the attraction base and around the global optima. So if we make that really big, then this is a huge win. If we make it really small then it's less of win. No and that makes a lot is right. So if, if, if the third one, had gone all the way from say number 3 to number 26. Then you would fall into it immediately and you would just, you would get there very quickly. Right. Okay, yeah, I see that, that makes, that makes a lot of sense. So, so when you asked me before what the advantages are, the advantages are, you know, you, you're in a place where the attraction basin for your global optima are in fact, relatively big. If you could, once you luck into that part of the space and there's lots of ways to luck into that part of the space, you can get there very quickly. So it's a big win. Agreed. Okay, good. You know, okay, that makes sense to me, I like that.

2.1.11 Simulated Annealing

Alright, we're going to look at another algorithm that kind of takes the idea of random restarts one step further. Instead of just waiting until you hit a local optimum to, 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. And the algorithm that we're talking about in concretely is called Simulating Annealing. 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. And you can think of this as being related to the notion exploring and exploiting. The hill climbing is what it's doing 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. So, which is better then, exploring or exploiting? Well you have to trade these off really carefully. So, 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. So, since I'm always trying to connect this back to stuff we talked before. 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. Does that make sense or is that too much of a stretch? I doesn't make sense to be me but I'm sure it make sense to you. [LAUGH] It makes sense to me. So you want to think of that as being kind of like overfitting. Well, you know, the fundamental problem of overfitting, right, is believing your data too much. And, and dealing with just the coincidences of what you happen to see. And that's like 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. So obviously as you need to take, as you said, 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. I don't, I don't know how deep that resemblance is. Superficial. They're like twin cousins. Twin cousins from another family branch. So the simulated annealing algorithm that we're talking about is related to an algorithm called Metropolis-Hastings, which I always thought sounded really cool. And the whole idea of it actually gets into metallurgy. So here I drew a picture of a sword. And we want the sword, when we're making a sword if we're a blacksmith and we're making a sword, we want it to be nice and hard. And the way to make it hard and sharp, and, and, and well structured, is for all the molecules to be, be aligned in the same way. So they fit together very snugly and they kind of lock in. And, so, there is a bit of an optimization process that has to happen. There is all these different ways that the molecules can be arranged and we would like to settle into an arrangement that has the lowest energy. Well, it turns out that the way blacksmiths have figured out to do this is they will make the sword and then do repeated hea, heating and cooling, and that actually show, they can see that that actually strengthens the blade, and, and what's really happening at the molecular level is it's giving the molecules an opportunity to realign themselves, to find an even better solution to the problem of how they can all fit into that space. So, this annealing is, is this idea of

heating and cooling. And we're not really literally heating and cooling, though of course when we run the computer it is getting a bit hotter. We're just simulating the idea of the temperature changing.

2.1.12 Annealing Algorithm

So, that's kind of just the motivation for the simulating the Annealing Algorithm, but there's an actual algorithm and it's, it's remarkably simple and remarkably effective. So, the algorithm goes like this, for, we're just going to 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 of t, from the neighborhood of x. And then what are we going to do? Well, we're going to move our x to that x t, probabilistically. So in particular, we've got this probability function pxxtt, which is going to help us decide whether or not to actually make the move. And it's, and, and the form of that function is written out here. So, the probability that if we're currently at x, and we're thinking about moving to x sub t, little t, and the current temperature is capital 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. Right? So we always hill climb when we are at a point where we can do that. So that's just rigorall hill climbing. It's kind of like hill climbing, exactly. 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. Okay. But, what if it's not an improvement? Well if it's not an improvement, what we're going to do is we're going to look at the fitness difference between the point that we're evaluating, and the point where we are now. Look at the difference between those two, divide that by the temperature, take E to that, and interpret that as a probability. And we either make the move or not with that probability. So, alright, so let's, we've gotta dive in a little bit to this expression to make some sense out of it. So, what happens, Charles, if the point that we currently visited x, then we visit a neighbor point X T, what if their fitnesses are really close to each other, say you know, just Infinitimally close to each other. Well, if they're infinitesimally close to one another, then that means that difference is going to be very close to 0. Right, and so we get 0 over doesn't matter what the temperature is, we get 0. E to the 0 is 1, so we make the move if it's you know, infinitesimally smaller than, than where we are now. Mm-hm. Alright, what if it's a, what if it's a big step down? Well, then, that means that number's really, really, negative. Yes. And the negative divided by some positive number T is the temperature, so T's always greater than or equal to 0. [BLANK AUDIO] Yeah, let's say that. Okay. In fact, probably making it equal to 0 could run us into trouble. So let's just say that it's bigger this year. So it's Kelvin. So it's in Kelvin. Okay, so [LAUGH]. we, that'd be a really, 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. Good, right. So, in other words, if a giant step down, we probably won't take it. Right, okay, that makes sense, so you're sort of smoothly going in directions that kind of look bad as a function of how bad they look, and sort of exaggerating that difference through an exponential function. [LAUGH] Right? [LAUGH] Sure. If, if that, if that gets you going. Well so 3 isn't 3 times worse than 1. It's you know, 2 to the 3 times worse than 1. I see, or E to the. Well so, so good, but let's, let's look at one more thing with this equation here. let's, let's say that there's You know, a moderate step down. Mm-hm. So, it's not so huge that when we divide it by T, it's essentially negative infinity. Let's just say it's a smaller thing, you know, - 5 or something like that. Okay. Then what, what happens? Now we get some E to the that is going to be some probability between 0 and 1. What does the T, what does T do? In this case. What if T is something really big? What if, what if, what if T is something really small? Well, T is something really big. Let's say, infinity then, it doesn't matter what the difference is. It's effectively going to be 0. Right, so really big temperature here means that we're going to have some negative thing divided by something else. This is going to be, essentially 0. It's going to be E to the 0, so it's going to be 1. So, when the temperature's really high, we're actually willing to take downward steps. So, we don't even sort of notice that it's going down. Right. There's lots of randomness happening. Right. So, in fact, if key is infinity, it's really, really, really, really hot then even if the neighbor, is much, much worse off. He'll be less worse off than, infinity, and so, basically you're always just going to jump to the next person. To the next neighbor. To the next point. Right. So we're very likely to accept. So the, so it's going to move freely. If T is really small, as T approaches 0 what happens? Well let's just take the extreme case. So T was 0 or effectively 0, then that means any difference whatsoever basically becomes infinity. Right, it gets magnified by this very, very small T. Right. And so then it's not going to take any downwards steps. It's only going to go uphill. Yeah, so that's kind of the essence of the Similarity Annealing Algorithm, so maybe we can talk about some of its properties. OK.

2.1.13 Properties of Simulated Annealing

What are some of the properties of simulated amealing? No we already talked about two of them: this idea that as the temperature goes to zero, it acts, it's acting more and more like hill climbing, only taking steps that improve the fitness. And as T goes to infinity, it's like a random walk around the neighborhood. So you, wherever point where it is, it's willing to go to any of the other points in the neighborhood and it just doesn't, it doesn't really pay any attention to the fitness function anymore. So huh, so that makes sense right if I wanted to, to go with the analogy. High temperatures like a lot of heat, which is a lot of energy and so molecules bounce around a lot. That kind of has the effect of flattening out the fitness function. T goes to zero, that's no energy. It's like you've frozen; molecules can't move and so you only move to places with lower energy or in this case, higher fitness. So that all makes sense. So how do I get from one to the other? You see you had any You had in the algorithm decrease t, but do we need to decrease t quickly, slowly, what's the right way to do it? So in practice, we wanted to decrease the temperature slowly, because it gives it, the system a kind of a chance to explore at the current temperature before we start to cool it out. So, well, one way to think about it is, if you think, if you consider some kind of function that we're trying to optimize, when the temperature is high, it kind of doesn't notice big valleys. It's, it's willing to kind of wander through big valleys. As the temperature gets cooler, the valleys become boundaries and it starts to break things up into different basins of attraction, but smaller gullies it still is able to walk, walk over. As the temperature gets even lower, those become barriers as well. And so what we really want to do is give it a chance to kind of wander to where the, the high value points are before cooling it so much that it can't now bridge the gulf between different local optimums. Ok, that makes sense. So, property as it's sort of bouncing between exploration and exploitation. I buy that. So where do you end up? Where do you end up? Well, if we're lucky, we end up at the global optimum; yay. [LAUGH] Is there anything, you know, we could say that we could characterize how often we end up there, or are we going to have to do randomized restarts and do simulated annealing a million times? So, 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 end at any, any given point x in the space is actually e to the fitness of x divided by the temperature and then normalized, because this is a probability distribution over the input space. So this is, this is pretty remarkable right? So this is saying that it's most likely to be in the places that have high fitness, right? because that's those are where it's going to have the highest probability of being. And 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 star, the, the, the optimum. And as the temperature is, is higher, it's going to smooth things out and it'll, you know, be randomly all over the place. Hm. So, so that's why it's really important that we eventually get down to a very low temperature. But if we, again, if we get there too quickly, then it can, it can be stuck. Hm. Right, because it's spending again, time proportional to the fitness. So, it is spending time ending up at points that are not the optimum, if they're say, close to the optimum. Okay, so you know, I actually now that you, you spell that out, I actually recognize this distribution, it has a name it's called the Boltzmann distribution. That's exactly right, and if there's people listening that have a physics background, they're going to probably be poking their eyes out because, huh, it's the case that we, we're using a number of physics concepts, but we don't quite use the same notation and we don't quite do things exactly the way that a physicist would. So, there's definitely a relationship and the Boltzmann distribution is used quite a bit in, in the physics community, but it may not be exactly like that so don't, try not to panic. Yeah, it's more like an analogy, like so much of life. It's a simulated Boltzmann Distribution. It's like a simile. [LAUGH]

2.1.14 Genetic Algorithms

All right. There's one more class of randomized optimization algorithms that is really attractive. It's, it's, the, it's very interesting to people. It has proven itself time and time and again. And it's this notion of genetic algorithms. So the main insight that's exploited in the genetic algorithm setting is this. So let's imagine we've got a two dimensional space. And, it's you know, hard for me to actually draw a, a fitness surface over a two dimensional space. So just kind of think of this as being one of those maps. Those contour maps. And so imagine that we've got, one dimension that X now comes in, in these two different dimensions. And What we're trying to do is find the peak, which happens to be there. So what if, we actually evaluate 3 different points, so these green points here, we actually check what the values are at these points. So what we find is that, from this initial point, this green point here. If we increase our dimension 2, we get a better value. But, it's also the case. It's starting from that point. If we increase on dimension 1, we get a better value. So, maybe, what we, we ought to do, is take kind of elements of these 2 solutions, these 2 inputs and combine them together, and move out on dimension 1 and dimension 2, and

maybe that will actually give us a good score as well. And in this particular case, it puts us in the base interaction of the local maxima. So, this turned out to be useful in many spaces, especially spaces that can be specified combinatorially like this. Where there is this separate dimensions that contribute in various ways to the overall fitness value. Ok Michael, that sort of makes sense. But, what does this have to do with genetics or algorithms for that matter? Well, it's an algorithm in that we're doing, it's an optimization algorithm, and the genetic part, is because what we're going to do, is were going to build an analogy with Biological evolution. Mm, analogies. In particular, instead of thinking about these input points, these little green dots, we're going to think of them as each input point is like an individual, and a group of them taken together is like a population. Mm-hm. It's really the same idea, but we're just giving it a different name. Okay. The the idea of local search where you make little changes to a, to an input, we're going to now call that mutation. All right? Where you take an individual and kind of tweak it a little bit Oh, like we did in the the, the example we did before where we define the neighborhood as every one, difference in every single bit. That's right. So, so the mutations can happen along neighborhoods. It's the same kind of concept as that. Okay. And then, you know? And, and, you can see that there's, the mutations happening over X. So, I assume that you get X [UNKNOWN]. [LAUGH] I think that's a fair point. I like your science. Yeah. [LAUGH] Yeah, that's right. It is science. The, those 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. So, what crossover does, is it takes different points, like this green point and this green point and instead of moving them just to their own neighborhood, it 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, that feels a little more like evolution. So, this is kind of like. Dare I say sexual reproduction. Right, where the two parents can actually form a new kind of offspring that you know, if you're lucky is has all the positive attributes of both of the parents. Like my children. Uh-huh, and if you're unlucky, it has the worse attributes of the parents, like other peoples children. [LAUGH] Exactly. And so, and finally what we were calling iteration before in the context of genetic algorithms, we can call it generation. Because we're going to take, a population of individuals and kind of you know, mate them together to create a new population of individuals. And we're going to, what we hope is improve iteration by iteration. Okay, that makes sense. So, If I can just push on this a minute, it seems like, if it weren't for crossover, this is really like doing random restart, except instead of doing restarts you just do 'em all at once cause we have parallel computers. Yeah, I think that's I think that's fair. I think that's quite fair actually. Okay. So then the thing that makes it more than that is, crossover. That somehow these parallel, random searches are bleeding information back and forth, help maybe bleeds the wrong. [LAUGH] Yeah, you don't want to get too biological about this. Yes, right. Well, so they're sharing fluids, metaphorically, with one another [LAUGH]. And conveying information that way, just the way genes do. Right. And then, and so that's the sort of interesting concept that now we have information, not just in the individual, like we're moving that one individual around and trying to find better and better solutions. But the population as a whole, represents a distribution over individuals. And that, that distribution might actually be a useful thing to guide the search for higher scoring individuals. Okay. That, that makes sense.

2.1.15 GA Skeleton

Here's a skeleton of the al, of an algorithm that implements a, a GA. So, [COUGH] what we need to do is we start off with some inital population and usually the population size is fixed to some constant. We'll call it K. So we just generate a bunch of, of random individuals to get things started. Then what we're going to do is we're going to repeat until essentially things converge. We're going to compute the fitness of all of the individuals in the population. And then we're going to select the most fit individuals. So, how do you suppose we might decide on which ones are the most fit? The ones you're most likely to go to the prom with? Some people choose people differently than just whether they're fit. I guess it depends what you mean by fit. Do you mean like fit, like physical fitness? Well, I meant fit like well, I did. But I guess fit, like, whatever the fitness function tells you is fit. Great! Okay, so, since we applied the fitness function f to all the individuals we have scores for them, and if we want to select the ones that are most fit, which ones do you think those would be? I guess you're saying those would be the ones with the highest scores. Yeah. So that is, that's definitely one way to do it, where what happens is you take the, 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. But there's other ways you can do it as well. One, this is sometimes called truncation selection. But 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. So does this get back to exploitation versus exploration then? I think it does. I think it, I was certainly having that thought when I was saying it out loud. That this is a, it's a similar kind of idea 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 irregardless, no. Irregardless is a word. Is it really? Yes it's actually a word. Irrespective is what I wanted to say. Irrespective of the fitness of those individuals. Well irregardless I understand what you mean. Alright, so then that's going to, we declare some of them as most fit. Then what we're going to do is 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 to. So instead of just taking the combination of the two parent individuals, we take their combination and then we make little, little local changes to it, to mute, mutate them. And we let that, the, the value of that pair that, that new offspring replace one of the least fit individuals in the population. So Michael, can I ask you a question? Sure. I'm having a hard time what is crossover means. Can you draw me a little picture? Sure. That we, this might be, you know rated R. That's fine. No, no, no. No, you're right because we're going to do it well so, it turns out that this is sort of generic, we can define crossover in many different ways. But I think you're right, I think it's worth looking at some concrete examples because it gives you some insight into why this operation might be useful. Okay.

2.1.16 Crossover Example

So let's do a concrete example of crossover. And so, it turns out that the crossover operation is always going to depend critically on how you represent the input space. So, let's say concretely that our input space is, is eight bit strings. So here's two parents, 01101100 and 11010111. And we're going to introduce them to each other. All right, now they know each other. Mm-hm. And now we're going to use them to create a new individual, a new offspring. So they're really going to know each other. All right. 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 now we can ask. How can we use this? To generate a new individual that uses elements of the two individuals that we have. So if you can think of any ways to do that. I have some ideas. [LAUGH] Yeah okay, what you got? I have lots of ways but I don't think any of them can be reproduced for the purposes of this learning lesson. All right well so let's, let's do ones that really kind of stick to the bit patterns. [LAUGH] I swear there's no way you can say this without getting in trouble Michael. All right. But that's okay. So how about, here's an obvious one. Right, if we really push the genetic notion as far as we can then each of those things represent, I don't know, alleles or some other biological term. And so what happens in genetics, right, is you mix and match your chromosones and alleles together. So why don't I say one child is The first four bits of this handsome Charles fellow and the last four bits of this beautiful and wonderful Sheila person. Alright, I see what you're doing there. So what you're saying is we're going to pick, well maybe this isn't quite what you said but I'm going to imagine what you said we're going to pick a random number along the sequence at the half-way point. And what we're going to do is now mix and match and create two offspring. One uses the first half of Charles and the second half of Sheila and then the other one is the other way around. And as you can see it's this last bit that determines the sex. Anyway, so these are the two offspring that these individuals have generated. And this particular way of, of combining where you randomly choose a position and then flip flop, is called one point crossover. Alright, so now I want you to think about this for a second Charles. So I don't know if it's an inductive bias, but what kind of bias do we put in? When we say well we're going to choose one of these points and we're going to flip flop based on where that point is chosen. What is that, what is that going to, what kind of offspring is that going to generate? Huh. Also, you know what, I see 2, I see 2 kind of assumptions built there. So, maybe that an inductive bias, so. Or a bias of some sort. So, one assumption is that locality of the bits matter. Good. Right. So the first by picking halfway through, you are saying, the first four bits are somehow related and the lsat four bits are somehow related because otherwise they wouldn't make any sense. Now to talk about them being together and that brings it to my second point which I guess really is just a first point. Which is that it assumes that there are subparts of the space that can be kind of independently optimized that you can then put together. Right, and in particular when I say sub spaces to optimize, I mean that they're independent part of the subspace, so that's actually the example that you gave before you said Well there's these two dimensions and each dimension kind of matters independently and the total reward or the total, the total fitness is some kind of linear combination of them. And so I can put them, cause 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, in that, kind of like the example we did

when we were doing bit guessing. That that you can be heading in a good direction, that they're pieces that are right and if we reuse those pieces we can get even righter. Sure. Alright so, and if it is the case that the sequence of the ordering of bits matters, we have this locality property. This is actually a fairly sensible thing to do. But can you imagine any other way of combing these bits together to get to get offspring? Well, I can think of lots. Well, so let's, let's focus, you know, you have many different possible ideas, but 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. We don't really, the ordering doesn't matter anymore. So keeping them clumped together like that. We don't think that that's a useful thing. Okay. Well, what would that mean? Tell me what that means. Well, so, The one point crossover, when we talked about that. It really matters that you know, the two bits that are next to each other are very likely to stay connected, right? That is, it's, it's unlikely that the split will happen to happen exactly between them and so we'll tend to travel as a group. But, if we don't think it's important that the bits that are next to each other need to travel together. If we say that It should be equally likely for any of the bits to kind of remain together. We need to cross over a lot more than just that one time. In a sense we might need to cross over every time. Well so, what I'm trying to get at here is this notion that what we could do is we could generate individuals by just scrambling at each bit position. Okay. So. The first bit position, maybe which stays the same, the second one flips, the third one stays the same, the fourth one stays the same, the fifth one flips, the sixth one stays the same, the seventh one flips, and the eighth one stays the same. 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 current that maybe 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. Okay. 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 right so we, we imagine that we get our genes from our parents but the, 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.

2.1.17 What Have We Learned

So that's really all I wanted to say about genetic algorithms. I mean, 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, this is, it's a, it's a useful thing. Some people call genetic algorithms the second best solution to any given problem. Hmm. So, it's a good thing to have in your toolbox. But I think that's, that's really it. That's all I wanted to say about randomized optimization. So what have we learned? Well, we learned about random optimization period. That that there is a notion of optimization in the first place. So, we talked about optimization in general and then we, what was the randomized part? Well, we'd take random steps where we start off in random places. Or, we'd do random kind, well actually that's really it. You take random steps, you start off in random places and it's a way to overcome when you can't take a natural gradient step. That's right. So did we talk, and we talked about some particular randomization. Er, sorry, randomized optimization algorithms. Let's see. There was randomized hill climbing. And we had 2 flavors of that. Right. We did simulated annealing. And, we did genetic algorithms. And don' t forget, that 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. A later in this particular sub-unit we're going to be talking about, finding good clusterings. And so, this notion of finding something that's good, finding a way to, to be optimal is pervasive through apache learning. Oh that make sense. Well, there, well, if we're trying to remember all these other things we learned. We also learned that AI researchers like analogies. [LAUGH] Both simulating annealing and generic algorithms are analogies. And they don't just like analogies, they like taking analogies and pushing them until they break. Indeed, actually hill climbing [LAUGH] is an analogy too. Yeah, actually, right? Every single thing that we did is an analogy to something. Okay, that's good. The other thing that we learned, which I think is important, is that there's no way to talk about cross, crossover without getting a bunch of people in the studio to giggle. [LAUGH] Yeah, genetic algorithms make people blush. Okay, that's pretty good, but Michael you know, I'm, I'm looking at all this and now that you put all these words together on one slide, I have 2 observations I want to make. That that are kind of bothering me. So, one is, I'm looking at hill climbing that makes sense, hill climbing restarts makes sense, simulated annealing makes sense, [INAUDIBLE] but you know what, they don't really remember a lot. So, what do I mean by that? 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, and creating swords with black smiths, and all that

other stuff you talked about and in the end, at every step, the only thing you remember is, where you are and maybe where you last were. And 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. And that just, that sort of bothers. So that's one thing. The second thing is, what I really liked about simulating annealing, other than, you know, the analogy and hearing you talk about strong swords, 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. That's, that's outstanding. Yes, very good, very good question. It is true that these are all kind of amnesic, right? 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, these are kind of the simplest algorithms, but you can re-combine these ideas you know, sort of cross-over 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. Right, they become taboo regions. To stay, with the idea that you should stay away from regions where you've already done a lot of evaluations, so you cover the space better. And then there's other methods that are, 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. Okay, so go ahead. Well, so I kind of added time. Maybe maybe you could look into this, I can give you some references and maybe you can report back. Fine. [LAUGH] You'll learn, you'll learn very well by doing that. Yes sir. [LAUGH] Alright, we'll see you then, then. Okay, I'll see you then, then too. Bye, Michael. Bye Charles.

2.1.18 MIMIC

Here's what I like to do. You pointed out some issues that you were concerned about and I thought that maybe you could go and look into it a bit more and you did. And so why don't I turn things over to you so that you can tell us what you've found out. Okay, well thank you Michael. Hi again. Hi. Thank you Michael. So I did go and I started trying to deal with these issues. So just to recap a little bit, there were two problems that I had, more or less. And they were, that we had all these cool little randomized optimization algorithms. And, most of them seemed to share this property that the only thing that really happened over time, is you started out with some point and you ended up with some point, which was supposed to be, you know, the optimal point. And the only difference between the first point, and the last point that you did or, the one millionth point or however how many you iterations you had, is that, that 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. Now you could argue that that 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. Now, one thing I really liked about some [UNKNOWN] is that you were very clear about what the probability distribution was. So what I decided to do is to go out there in the world and see if I could find maybe a class of algorithms that took care of these two points for us. And I found something, you'll be very happy hear, Michael. Yeah, I would love to, to find out what it is. It turns out that I wrote a paper about this, almost 20 years ago. [LAUGH] How did you find that? I just said, well if I wanted to start looking someplace, I should look at home first. And I stumbled across Oh. this paper that I wrote. I see, I see. So learning about machine learning, really begins at home. That's exactly right. So, I had to re-read the paper because, you know, it is a coupl of decades old. And I will point that a lot of other work has been done since this that refines on these ideas. But, this is fairly straightforward and simple, so, I think I am just going to stick with this work. And the paper's available for everyone listening this to right now or watching this right now. So you can read it and all of it's gory details. But I just want to go over the, the high level bit here because I, I, I really think it kind of gets at this idea. So, in particular, the paper that I'm talking about introduced an algorithm called Mimic, which actually stands for something, though I forget

what. And it really had a very simple structure to it. The basic idea was to directly model a probability distribution. Probability distribution of what? Well, I'll tell you. And, you know, like I said, Michael, I will, define exactly what this, probability distribution is for you for a second and, and hopefully you'll, you'll buy that it seems like a reasonable distribution to model. And given that you have this, probability distribution that you're directly modeling, the, the goal is to do this sort of search through space, just like we did with all the rest of these algorithms. And to successfully refine the estimate of that distribution. Hm. And the idea is that if you can directly model this distribution and refine it over time that, that will in fact convey structure. Structure in particular of what were learning about the search space while we're doing the search. Exactly, 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. Yeah, that seems like a really useful thing. So I'm just going to give you, again, this, this simple mimic algorithm that, that sort of captures these basic ideas, because I think it's fairly simple and easy to understand, while still getting some of the underlying issues. But do keep in mind that there's been literally decades of work since then, and optimization space where people have really taken these kinds of ideas and refined them to be sort of much more mathematically precise. But this, I think, does get the idea across, and I happen to understand it, so I thought that I would share it with you. Hm. Seem fair? Yeah, that sounds really exciting. Excellent.

2.1.19 A Probability Model Question

Okay, so here's a probability distribution I'm going to define for you Michael. You'll notice that it's a probability over our Xs. And it's parameterized by theta. So theta is going to turn out to stand for threshold. Okay? Got it? Mmhm. Okay, so here's a probability distribution. It is 1 over z sub theta. For all values of x such that the fitness function is greater than or equal to theta. Yeah. And it's 0 otherwise. So do you understand that? I think so. So I assume that Z sub theta here is, is going to be some 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. That's exactly right. So another way of saying this is that the this probability is uniform over all values of x whose fitness are above some threshold. Yeah so, I'm, I'm, the image I have in my hand is 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, that collection. That's right. And everything below it, we will simply ignore. Doesn't happen, it doesn't get sampled form. Exactly. Okay so because you demanded it Michael, here's a quiz. [LAUGH] Alright. So, two questions in this quiz. The first question is, theta's some threshold and so let's imagine the fitness function has to have some minimum value and some maximum value. Okay? And let's call those theta submin and theta submax respectively. And so I want you to describe in one or two words P sup data min of x. That is the probability distribution whether threshold is its minimum value, and P sup data max of x. That is, the distribution where theta is at it's maximum value. You got it? So, just to, just to be clear, sort of the maximum meaningful and minimum meaningful values Right. Because I can imagine them, you know, if it's just sort of way outside the range of the fitness function then it's kind of trivial, I guess. Right, well, let's, it is, but let's assume that theta min is the lowest value that the fitness function can take on and theta max is the largest value of the fitness function. Okay, yeah, okay, that's how I was imagining it. Okay. Okay, yeah, I'm good to go. Okay. So go!

2.1.20 A Probability Model Solution

Okay Marco, you got an answer for me? Yeah, yeah. I, this is, the way I was thinking about it was essentially, well for the theta min. Wait, so we're maximixing, right? yes. All right. So theta max, then, is going to be the largest value if, that f can return. Yep. Which is actually the optimum of the function. So that probability distribution assigns a probability of well one, if there's a unique optimum, it'll decide a probability of one to that single point. That's exactly right, but you were getting at something else when you said if there's one optimum. What if there are multiple optima? Then it's uniform over all of them. Right. So, it is the distribution that generates only optimal points. So, it's a distribution over optima, which is just you said and we'd accept optimum point or anything that sort of captures this idea. Okay. Okay? Well, what about theta submit? So, if it's the minimum that the function can achieve then it, 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. That's exactly right. So it is in fact, simply, the uniform distribution. Nice. Now, this is going to be pretty cool, because in this, in this slide right here we basically have the mimic algorithm. I'm not seeing it. Well I'm about to tell you, we are basically going to try to estimate this particular distribution; P sub theta of X. We're going to start out with P sub theta of min, P sub theta min of x, which is the uniform distribution. So we're going to sample uniformly from all of the points, and then somehow use those points

to do better and better, and better and better estimates until we get from uniform to a distribution of only the optimal points. And that's the basic idea. So, okay, so, and, and the first one being something that's really easy to sample from, because we're just sampling uniformly from the input space. Right. And the second being 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. Exactly. So we'll start out here. And we're, the goal is to start out here and then to end up here. Gotcha. And so let's actually write out a out rhythm that's does that. Yeah, because that's not an algorithm that's just a goal. But many algorithms start out as goals. Hm.

2.1.21 Pseudo Code

Okay, so Michael here's some silicone that represents an algorithm of the sort I just described after the last quiz. So here's the basic idea, just see if you can picture this. We have, were in the middle of this process, we have some data at sometimes, step t, and were simply going to generate samples that are consistent with that distribution. So, we're going to shoot our probability distribution is not just something that gives us a probability but something from which we can sample. So, generate samples according to P su theta sub T. Generate a bunch of those samples and now that we have these samples, we're going to come up with a new theta. T plus one. And that theta T plus one is going to be the best samples that we just generated. So, let's say the top half. Now, you'll notice that this should actually remind you of something. Does it remind you of any of the other randomized algorithms that we've looked at so far? Maybe a little bit like simulated annealing. No. Hm. Because it does, you know, change the probability of going up. What we're choosing. Oh, we're choosing different percentile. That's a lot like genetic algorithms. Right, it's exactly like genetic algorithms. Except instead of having this population that moves from one bit to other, we generate samples that's like our population. So, we generate a population here. We pick the most fit of that population by retaining only those that are the best ones. And all the stuff that you said before about, well maybe you don't want to pick the best ones. Maybe you want to sample it according to a probability distribution proportional to the fitness. All those things you talked about before would apply here, but let's just stick to the simple case where you, you take the best ones. And, now that we've got this new population, rather than using that population again, we estimate a new distribution that's consistent with those. And then we just lather, rinse, repeat until we come to some conversions. Okay, so let me just make sure I'm following. First of all, when you say P sup, you don't mean the food. I do not, because I do not like pea soup. And you don't mean sup meaning like the largest possible value of theta, because when I was hearing p-sup theta, I was, I kept thinking that you mean the best theta. No, soup is short for superscript. [CROSSTALK] When people say p-sub-i, which means subscript. Oh, yeah, okay, that, that makes sense. Mm-hm. And, okay. So now now that's [LAUGH] that's clear. The, this notion of estimating a probability distribution, I guess that's where it feels really fuzzy to me. because if you do the estimate as a kind of, I don't know, sort of particle filtery kind of thing, where you just keep instances. Mm-hm. Then it feels really, a lot like the genetic algorithm. Right. Except, remember, one of the things that we cared about, a structure. So this structure that we're maintaining remember this, this complaint that I had that we weren't somehow keeping track of structure in a nice way? Yeah. Is all going to be hidden inside the 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. And I'll, I'll get into some details about that in the very next slide, I just want to make certain that you understand. The high level bit here. The high level bit. And the high. Yeah, I mean I am not, I am not connecting it with theta in the sense of the previous slide. So theta is our threshold, right? So, basically, we have some distribution. Now, remember what is P superscript theta? It is you know, it is a probability distribution that is uniform over all points that are, have a fitness value that is greater than theta, greater than or equal to theta. Right. So I have some distribution here. If I just generate samples from that distribution then what this means is I'm going to be generating all the points whose value, whose fitness is at least as good as theta. And then I'm going to take from those the set of points that are much higher than theta, hopefully and use that to estimate a new distribution. And I'm just going to keep doing that. And what would should happen is, since I'm consoling taking the best of those, is that theta over time will get higher and higher, and higher, and I'll move from theta min over time, the theta max. And when I get to theta max, I've converged, and I'm done. I see. So all right, so the theta right, so I guess I was confused because I, I didn't completely absorb the second line yet. So that we're updating the theta and it's specifically going to be tracking the, I'm not sure what the nth percentile is. [INAUDIBLE] Adding the number of samples that we draw. Oh no. I'm sorry. Nth means you know, 50th percentile or 25th percentile or 75th percentile. So, okay. So, if you set that to be the median. Mm-hm. Which is the 50th percentile. Then what we're doing is we're, we're sampling. We're taking the top half of what's left. We're somehow refitting a distribution into 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're sampling from kind of a more fit collection. Right. That's exactly right. Okay. So this should work or at least it should match your intuition it would work if there are, sort of two things that are true. The first is that we can actually do this estimate. Yeah, that's the part that scares me. Uh-huh. Sure. Well given a finite set of data, can we estimate a probability distribution. So that's the first thing that sort of had better be true. And the second thing that needs to be true and this is, I think, a bit more subtle, is that the probability distribution for a piece of theta and a probability distribution for say, piece of theta plus epsilon. That is a slightly higher value of theta, should be kind of close. So in other, and what does that mean? What that means. Is that generating samples from P sub theta of t should give me samples from P sub theta of t plus 1. So I have to that, that means that not only do I have to be able to estimate the distribution, I have to do a good enough job of estimating this distribution such that, when I generate samples from it, I will also generate samples from the next distribution that I'm looking for. Okay? Okay. So if those things are true. Then, I, I hope, you can be sort of imagine that you would be able to move from the part of the space that a theta min, and eventually get to the part of the space, the theta max. Yeah, it seems like it should climb right up. Yeah, I agree with that. That's just, assuming that you could represent these distributions in between. Again, the picture I have in my head is an, like a bumpy mountainous landscape. And you put a slice through it. And, in the beginning, yeah, right. So in the beginning it's the whole space, and that should be easy to represent. At the end it's a single point, and that's easy to represent. But in the middle. Yeah, like that. But in the middle, it's this sort of crazy, there's a blob, then there's a space, then there's more blob. Little blob with a hole in it, so like that might be a much harder distribution to represent. Right, and that's going to turn out to be an empirical question but as I, as I hope you'll see in a couple of slides it tends to work out pretty well in practice. Cool. Okay. And by the way when it doesn't work out well in practice, there's all these cute tricks that you can do to make it work out pretty well in practice, and we'll talk about that towards the end, okay

2.1.22 Estimating Distributions

Okay Michael, so, let me see if I can convince you that we actually have some way of actually estimating these distributions. So, all I've written up here is the chain rule version of a joint probability distribution, okay? But just to be clear, what these subscripts mean here, every x that we have is made up of a set of features. So, there's, let's just say there is N of these features. And so really, X is a vector. There's feature one. There's feature two. There's feature three, dot, dot, dot. All the way up to feature N. Okay? Sure. What I really want to know for my piece of theta and I'm going to drop the theta's here for the purpose of just describing this generic distribution. Is I want to say, well the probability of me seeing all of the features yeah, all of the features of some particular example is just the joint distribution over all of those features. Now of course, we could just estimate this, but that's going to be hard. Why's it going to be hard? Because. Well, the first distribution is conditioned on a lot of things, so it's an exponential sized conditional probability table. Exactly, so this is exponential table and if it's an exponential table then we need to have an enormous amount of data in order to estimate it well and this is sort of the fundamental problem. But, we have addressed this in earlier lectures, earlier lessons Michael. In the inference lecture maybe? Yes, in the inference lecture. Whew. Where we can try to estimate this incredibly painful joint distribution by making some assumptions about conditional independence, and in particular I'm going to make, one kind of assumption. And that is, that we only care about what's called Dependency Trees. Okay, so what's a dependency tree Michael? Do you remember? No I have absolutely no idea, I don't think I've ever heard of such a thing. So the Dependency Tree is a special case of a Bayesian network, where the network itself is a tree. That is, every node, every variable in the tree, has exactly one parent. I see. So sometimes in Bayesian networks, they talk about polytrees. Polytrees just mean that if you ignore the direction of the edges, you have a tree. But you're actually talking about the, the edges have to go in a particular direction. Everybody's got one parent. Right, exactly right. And in, and in, so, you should see these as a directed graph, although I almost never draw them that way. Because, you know, it's sort of 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, and what does that mean? That means that every node, every random variable, depends on exactly one other random variable. Yeah. So we can rewrite this joint distribution here now as a product over each of the features depending upon only its parent. I see. So the first capital pi there means product. And the second lower case pi there means parent. Exactly. So when I compare this representation of a distribution, dependency to representation versus the full joint. What nice properties do I get from doing it this way versus doing it that way? Well the, I guess the main positive is that you're only ever conditioned on, well it's gotta be at most one, right? Not exactly one? Right. At most one, so, so in fact you, you picked up on a nice rotational

thing here. The parent of the tree, the root of the tree doesn't actually have a parent. So just assume that in this case Pi returns itself or something and so it's the unconditional distribution. And that could happen at any time. Got, gotcha, alright, yeah and so, like if nobody has any parents, then that's the same as Naive Bayes, but here it can happen with those one parents. No. No? Naive Bayes has one, exactly one, everyone has one parent and it's exactly the same one. Oh, right, good point. So, if everyone has no parents, then you're saying that all of the features are in fact, independent of one another. Gotcha. Okay. Alright. But the, but that's not what you asked. You asked comparing that probability distribution to the full joint, the main thing is that you're, no ones ever conditioned on more than one, other feature and therefore the conditional probability tables stay very, very small. Right. In fact do you, how small do they stay? Well it depends on how big this, are there, are there binary features that we're talking about? Yeah let's say they're binary. So then it's like two numbers. Right. It's like you're probability when your parent is true and your probability when your parent is false. Well that's each table. But what about representing the entire thing? So it's going to be linear in the size of the number of variables or. Right. Features. Right and the number, the amount of the data that you need is going to be fundamentally. In order to get this tape, in order to get this tree, it's going to turn out that you only need to keep track of quadratic set of numbers, quadratic in the number. We're not going to be given that tree, we're going to have to figure that out? Well, 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? Yeah, that' doesn't exactly follow from the algorithm sketch because you could also say well, then you need to figure out that a tree is the right thing at all instead of something else like a box. Oh, that's fair, but what we're going to do is were just going to assume that we're going to do the dependency trees, now why are we doing this? Were actually doing this for a couple of reasons Michael, your, your points pretty your points well taken. 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 how many relationships do you want, what kind, which of all the possible relationships you could have where you only are related to one other thing, do you want to have. Well, 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, as you point out, not having any parents at all. In which case, you would be estimating simply this. Yeah, I wouldn't say that the next simplest. That's even simpler. But it doesn't, doesn't allow any of the inter-relationships. Any of the co-variants essentially information to be captured. No, that's fair, that's a fair point. So, we could have started with something like this except here you must you 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. So that's the basic idea. Now I want to be, 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. I buy that. Okay and one other thing worth noting is I you'll I hope you'll see this is a couple of slides if you don't see it immediately, is that at the very least this will allow us to capture the same kind of relationships that we get from cross over in genetic algorithms. Huh? And that was a bit of the, the inspiration for this. That cross-over is representing structure. In this case structure that's measured by locality, and this is kind of the general form of that. So you are saying if there is some locality, then whatever, however we're going to learn the dependency tree from the samples is going to be able to capture that. And therefore, it will kind of wrap its head around the same kind of information that that cross over's exploiting. That's exactly right, and in fact, as we'll see in one of the examples that I'll give you in a moment, that it can do better than that because it doesn't actually require locality the way that crossover does. There's one other thing that's worth mentioning here about this distribution and why it's nice. It captures relationships. But the second thing is, something that you, you sort of alluded to earlier is that, it's very easy to sample from. Right? So 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. Right? 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. Right. I get that and 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. Exactly, that's what we're going to do next and that's going to involves math.

2.1.23 Finding Dependency Trees

Okay Michael, so what I'm going to do, I'm going to step through how you find dependency trees and just stop me if it doesn't make any sense, okay? Sure. But hopefully it's fairly straightforward, at least if you understand information theory. And again I want to stress that although we're going to, 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, okay? Exciting. Okay, so the first thing we have to remember, Michael, is that we have some true probability distribution which I'm just going to write as P, that we're trying to do. That's our P soup theta in this case. But the general question of how you represent a dependency tree doesn't depend on theta or anything else, there's just some underlying distribution we care about, let's call that P. Okay. And we're going to estimate it with another distribution which I'm going to represent as p-hat because, you know, for approximation. That's going to depend upon that parent function that we defined before. Okay? Alright. So somehow. You sound skeptical, Michael. Well, because I saw that you wrote those train tracks. And you figure they must mean something? Well I know they mean something. It's a, it's an information theory thing but I'm not going to remember what it is. You're going to, it's going to, you're going to say something like kl divergence or something. That's exactly right. So, somehow we want to not just find a p-hat sub theta, that is a dependency tree that represents the underlying distribution, but we want to find the best one. And so best one here sort of means closest one, or most similar, or the one that would generate the points in the best possible way. And 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. You remember what the KL Divergence stands for, Michael? Kullback Leibler That's right. And it has a particular form and in this case noncontinuous variables. It has basically this form. And that's basically a measure of the divergence, that's what the D stands for, the divergence between the underlying distribution P that we care about and some other candidate distribution P-hat that we're trying to get as close as possible. So if these are the same distributions, if P-hat and P are the same distribution, the Kullback–Leibler divergence is equal to zero. Mm. Mm-hm. And 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. 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 P-hat, that is as close as we can get to P, okay? And we have a whole unit that Pushcar will do to remind everybody about information theory where this comes from. But basically this is the right way to define similarity between probability distributions. You just have to kind of take that on faith. Okay. Well, I'll, I'll pause this and go back and listen to Pushcar's lecture and then I'll be ready for you. Okay, beautiful. Okay. So what we really want to do is we want to minimize the Kullback–Leibler divergence the 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. So Michael, if you were, if you were paying close attention to the algebra, you will realize that well p log p, now that you've come back from Pushcar's lecture is just entropy. Yep. So or it's minus the entropy. And so I can rewrite this as simply minus the entropy of the underlying distribution, plus the sum of the conditional entropies, for each of the Xis, given its parent. Which has some, you know, sort of intuitive niceness do it, but, whatever. This is what you end up with just by doing the substitution. P log P gives you minus entropy of P minus P log P hat, which gives you the conditional entropy according to the function, the parent function pi. Okay. Well, in the end all we care about is finding the best pi. So, this term doesn't matter at all and so we end up with a kind of cost function that we would like to minimize. Which I'm going to call here J. Which depends upon pi which is just the sum of all the conditional entropies. 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. Does that make any intuitive sense to you? Yeah. I think so. Cause we want, we want to choose parents that are going to give us a lot of information about the values of the corresponding features.

2.1.24 Finding Dependency Trees Two

Right. So if, in order for this to minimized you would have to have picked a parent that tells you a lot about yourself. All right. 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 sort of sum of [UNKNOWN] conditional [UNKNOWN]. You with me? Yeah. Okay, 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 this. Actually, I know how you go about computing it, and let me

tell you, it's excruciatingly painful. But it turns out that there's a cute little trick that you can do to make it less excruciatingly painful, and what I'm going to do is I'm going to define a slightly different version of this function. And I'm going to call it, j hat. So as I said before, 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 prime, where I've added this term. 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 this depends upon pi and so doesn't actually change the proper pi. That makes sense? Yeah, except I keep thinking about pie. Mm, pie. Mm, I'm sorry, you got me distracted. Okay but do you see how minimizing this versus minimizing this should give me the same pi? I do. I mean, it's sort of like adding a constant if you've got a max. It doesn't change which element gives you the max. Right. But by adding this term I've actually come up with something kind of cute. What is this expression, Michael? Do you know? It looks kind of familiar from Information Theory. Is that cross-entropy? No, though sort of, but no. Is it mutual information? It is in fact, mutual information. In fact. It's the negative of, mutual information. So, minimizing this expression, is the same thing as maximizing, neutral information. Now, I went through this for a couple of reasons Michael. One is, I think it's easier to, kind of see what mutual information is. But also because, this going to induce a very simple algorithm. Which I'll show you in a second, for figuring out how to find a dependency tree. And the trick here is to realize that, conditional entropies are directional. Right? Wait, so conditional entropies is, is, oh, is that, that's the quantity that we started up? Right. At the top with the hq, okay, huh. So, this is, this is directional right? Xi depends upon this Yeah. And if I were to do h of pi given xi, I would be getting a completely different number. Mutual information on the other hand is bi-directional. Interesting. 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 so we wanted to minimize a couple of liable deductions because that's what Shannon told us to do. We work it all out and it turns out we really want to minimize the kind of cost function another way of rewriting this of conditional entropy. We threw this little trick in, which just allows us to turn those conditional interviews into mutual information. And what this basically says is that to find the best pi, the best parents, the best dependency tree, means you should maximize the mutual information between every feature and its parent. Nice. Right. And that sort of makes sense, right? I want be associated with the parent that gives the most information about me. Charles, I have a question. Yes? Just a little bit confused. The the xis, what are they, where's that bound? Oh, by the summation. The, what summation? You know, the summation that's always been there the entire time I've been talking. Oh, I'm sorry I missed that. I don't know how you missed it man, its right there. Yeah, I mean, you didn't actually put the index in the summation so I was, guess I was, I was confused. My fault right, so just to be clear of anyone who noticed that the [INAUDIBLE] version is a summation over all possible variables in the distribution And so we've done here is carry that all the way through and so these two steps here in particular. 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 mutual information, the sum of the mutual informations, between every feature and its parent. Hmm, interesting. Right, and again I think that makes a lot of sense, right? You, basically the best tree is, the best dependency tree is the one that captures dependencies the best. I see. Alright. That's cool. Alright, so now we need to figure out how to do that optimization. Exactly right. And it turns out it's gloriously easy.

2.1.25 Finding Dependency Trees Three

Alright Michael, so see if you can tell me where we're going here. So, just as a reminder I've erased a bunch of stuff so I can make some room. We're trying to maximize this cost function, J prime sub 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. So here, I've drawn a bunch of nodes as part of a graph, and all the edges between them. Each one of these nodes is going to represent a feature, an X of I and what's going to go on the edges, is going to be the mutual information between them. So, I know 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 sub graph of this graph, in particular a tree. That maximizes these sums. So, any ideas how I do that, Michael? First off, does that make sense what I just said? Yeah, yeah, yeah, I mean it's, it's very interesting right. So we're trying to pick, each, each time we pick an edge, it's actually saying that what we, we're going to care about the relationship between this pair of variables. And we're only allowing edges. We're not allowing, because they, they, that involves 2 nodes. We're not getting sets of 3, or 5, or however many nodes larger than that. Mm-hm. And so what we want, is we want to be considering a subset of edges, that form a tree, that has the highest total information content. This is correct. And that's a minimum spanning tree. Well, not quite. That's a maximum spanning tree. That's

exactly right. It turns out that finding the tree consistent with this graph, such that this is true, is in fact the same thing as finding the maximum spanning tree. Which we all vaguely remember from our algorithms class. So you said, it was terribly painful, but it's actually not. This is a No, I said, it's, it's terribly easily painful. So, that's really neat. We turn to 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 some, of a well-known, well understood computer science problem, of finding the maximum spanning tree. If we find the maximum spanning tree. Then we have found the best dependency tree. So, to be honest, I don't remember maximum spanning tree. Minimum spanning tree, can we just solve a maximum spanning tree problem by, I don't know, negating the edges or, or Yes. Adding, okay. That's exactly right. So, everywhere in there, where you pick a max, you pick a min or you just label them with the negative of the mutual information and it's the same thing. I think that this particular form of writing as a maximum mutual information, is easier to think about. Okay. And what you'll do when you find the maximum spanning tree, is you'll end up with, in fact some tree. And you're done. Well, so now you need to tell me, because the structure was a directed structure. So, dude, so can we then pick any node then call it the root and then do a, like a depth first reversal or something? Absolutely. So, first off, I want to point out there's 2 different algorithms that you could use. Do you remember the two algorithms you could use to find the maximum spanning tree? Or the 2 famous ones anyway? I'm going to go with Prim and Kruskal but I know there's others. No, those are the two that, that they teach you in algorithms class. And it turns out for this particular one, you want to use prim. Because, prim is, Proper. [LAUGH] It is proper, prim and proper. Prim is the proper algorithm to use whenever you have a strongly connected graph, it just happens to be faster. You mean densely connect to graph. What did I say? Strongly. Okay, densely connected. Right, a densely connected graph and by the way, this is a fully connected graph, which is about as dense a connected graph as you can get. [LAUGH] So, Prim's algorithm runs in time quadratic or polynomial anyway in the number of edges. And so it's actually fairly efficient as things go, because as we know in theory if it's polynomial, it might as well be free. There you go. [LAUGH] So use Prim's algorithm, you find the Maximum Spanning Tree, and you are done. Alright, I'm going to need reminding where we, where this is all plugging in. Okay. because. That's exactly what I was going to. because, this is a cool diversion, but. Right, so this is actually worth pointing out this was a bit of a diversion that we had to do. Let's go back to the original algorithm that we had, and just point out what we would be doing here.

2.1.26 Back to Pseudo Code

Okay Michael so let me take you back to the pseudo code that we, we have for MIMIC and see if we can plugin some of the stuff that we we just talked about just as a way of refreshing. So as you recall, the goals to generate samples from some Pea soup theta sub T of X. So in this case were just going to estimate that by finding the best dependency tree we can, that is the best Pi function, okay? 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 this. We simply started a node, generate according to it's unconditional distribution and then generate samples from each according to it's conditional distribution given it's parents. So, first we generate a sample from here. Then we generate one from here, then from here, then from here, and then from here. Now where do we get these conditional probability tables, Michael? That's a good question. I think it's pretty direct from the mutual information. Exactly. 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 now, for every single feature, X of I, how often that was say, 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, yes? Yep. And 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. So, already have the conditional probability table for each of those as well. So, 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. So given that we can generate samples and time linear and the number of features, and we're done. So this part is easy. By the way you asked me a question before. I just want to make it clear, that we come back with an undirected graph. And, you can pick any single node as the root, and that will induce a specific directed tree. And it actually just follows from the chain rule. I'll leave that as an, an exercise to the, the viewer. But there you go. So now we know how to generate samples. From some tree that we have. We generate a bunch of these samples. We 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. Got it? Cool. Right. So, 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. 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.

2.1.27 Probability Distribution Question

Okay so, we're going to have a quick quiz. Michael insists on having one, because Michael likes that sort of thing, and I like Michael, so I'm going to go with him and give you a quick quiz to see if you follow along with what we talked about. Now this quiz is less about the details of mimic itself. Than it is about the probability distribution. So I've been harping on this idea that mimic doesn't care about your probability distribution. You should just pick the best one. So we want to make certain that you get that by giving your three problems and three probability distributions you might use and see if you can tell us which one goes with which. Okay? So here are the three problems. The first problem is you're fitness function is maximize the number of 1. So the first thing you need to, to see is that; we're going to assume in all of these cases, that our input value is x binary strings of length N. Okay? So, 00000001001011. 11100111, whatever. There's going to be a string of 100. So, they're binary digits. And the first problem we want to maximize the number of 1s that appear in that sample. Do you get that, Michael. Yeah. I mean it's, I mean that's going to be maximized by just making everything a 1, right? That's right. That's right. Okay, but, but we want, we want to think about mimic finding that. Right, because you don't know that, that's the true underlying fitness function. That just happens to be what it is. I see. Okay. Okay. So, in the second problem, we want you to maximize the number of alternations between bits. So, 101 01 01 is better than all 1s, much better the all 1s in fact. because the fitness you'd get from maximizing alternations. Between 01 0101 is in fact, N minus 1 which is the largest that could be. But if you add all the digits being the same, your fitness would be 0. You see that Michael? And minus 1, oh, I see and minus 1 because it switch every time it switches. Right. Got it. So what would maximize the number of alternations and say, hey. Five digit string. Like 01010. Or? The compliment of that, 10101. Right. So two values both act as maxima here. Okay. The third problem is you want to minimize two color errors in a graph. Now that one is a little bit harder to describe, so I'm going to draw a little graph for you. So, the two color problem you might remember is given a graph with some edges, like say, this graph here. You want to assign a color to each node such that every node has a different color than its neighbor. Okay? So, we assume there's only two colors here. One is zero. So let's say I assign This the color, value of 1. This the color value of zero. This the color value of zero. This the color value of 1. And this the color value of 1. So how many errors are there? Michael. Wait, we're, oh, we should be trying to maximize. But, okay, so I guess not. So minimize two color errors. So an error would, in this case would be, an edge that has the same color on both ends. And I see one of those, the bottom sort of rightish. Right. So here these do not match. So that's good. These do not match. These do not match. But these two match. Even though that one doesn't. So there's exactly one error. So in these first two examples we're trying to maximize the fitness. Here we're trying to minimize the cost just to make things. Slightly more difficult for you. If we removed this edge here, then you'll notice that this has no mismatches whatsoever, and this would be an optimum. Got it. Okay? So we need to figure this out. Okay? So there we go. To maximize the number of ones in your string or maximize the alternations by adjacent bits. Or to minimize the number of two color errors in a graph. Alright. Those are the three problems. Now here are the three distributions. The first distribution, these are in alphabetical order, is a chain. So a chain would be in kind of Bayesian network speak, a chain would be a graph. Like this. 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. So, the way you would generate samples is you would generate a sample from here. And then generate a sample on here dependent upon this value. Given that value, you generate a sample here. Given that value, then you generate a sample given that value. Got it Michael? I think so. So, and, and we are imagining that the ordering is given. So, that, so, we know which ones depend on which. Right. So, in fact, it's not just a chain. It's a specific chain. And is it the same chain as, say, the ordering of the bits? Yes. In this particular case, it's the same chain as the ordering of the bits so, I'm going to call this that chain. [LAUGH] Alright. Okay. The second one is what we've been talking all about along. It's a dependency tree. Unlike in the case with this chain here where. I am giving you a specific chain. This is the first bit, the second bit, the third bit, the fourth bit and

so on to the nth bit. 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. So, if I were to draw that. It would be a bunch of nodes with no edges between them, directed or otherwise. And so, the joint probability across all your features is just a product of the unconditional probabilities. So, the simplest probability distribution you can have. Okay, You got that, Michael? Yeah, so. And, okay, if I'm understanding correctly, each of these is representable by a dependency tree. Yeah, each one is, actually. So then why, why would any one be better than any other one? I don't know, Michael, you tell me? I guess in the case of, well could be, you could think of it maybe in the terms of number of parameters that need to be estimated. 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. Yep. In the chain case, we have a conditional probability per node. So it's, like 2n parameters. Mm-hm, yep. And in the dependency tree case, it's. I think what you were saying is that we're estimating kind of n squared parameters. And then, then pulling the tree out of that. Exactly. Now, of course, like you point out Michael. A chain is a dependency tree. Independents are a dependency tree and a dependency tree is surprisingly enough a dependency tree. [LAUGH] But these numbers and parameters do matter, because although a dependency tree can represent and independent set of variables. The way your going to discover that their independent is that your 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 over fit in the case where all the variables or all the features are independent. Okay. Okay. Alright. So, you got it? Alright. Nope. One more question. Yes? Well, other then I need to fill the boxes with the numbers one, two, or three, right? Yep. Of the algorithm that they're best for, or the the problem that they're the best for. Got it. And each one is used exactly once? Yes. Okay, and then finally just to kind of make sure that I am wrapped head around why we are doing probability distributions at all. So in the context of mimic, we need some way of capturing the, I guess the space of answers that do at least a certain level. They, they do at least this will and [UNKNOWN] the fitness. So, it, it, it. Hm. I guess, I guess I feel a little turn off because. We're not trying to represent the fitness functions, right? You're not telling me which dependency structure should I be using to represent this fitness function. Right. Instead I'm asking you to figure out which distribution will represent the optima, the structure between the optimal values. Huh, okay. Alright, I'm not sure I 100% get it, but I think I get it enough to answer this quiz. Excellent. Okay, so go.

2.1.28 Probability Distribution Solution

Okay, Michael. Okay, 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. We're, we're, you know, local information is enough. Is that, is that enough to say that the answer, that the one goes in the last box for independent? That's correct. But it's, I guess, I guess what I don't quite understand is in, in the case of the, so we want to know, what's the probability, yeah, it's a funny thing we're representing, right? Yeah. So if we, the, like, the third position, we're saying, well what's the probability that if this is a 1 that we're going to be above a certain value, and what's the probability if, if it's a zero, we're going to be above a certain value. Mm-hm. And that doesn't really exactly capture the way the fitness function works. But I guess, oh, I guess because we're drawing from, we're drawing the rest of the things at random, the rest of the values at random? Mm-hm. Yeah, we are. So is that where the meaning comes from, then? Yeah, think about it this way. Remember, the probability distribution that we care about, p sub theta, is basically uniform for all x's such that the fitness value is greater than or equal to theta. So imagine we start out with theta equal to 0, let's say, which is the lowest value that you could get? Well then, what's probability of bit 1 being a 1? Okay For all values greater than or equal to, well, it's 1/2. Okay. Because every single value is okay. And 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. Agreed? I do agree with that, yeah. Okay. Now, at the very, very end, when I have all 1s as the maximum value, that's also easy to represent. Because the probability, what's the probability of bit 1 being 1? For the optimum value. 1. Yes, the probability of bit 2 is? 1. And the probability of bit 3? Point, no, 1. Exactly, and they all can be independently represented. So we can definitely represent the minimum theta value. Streams, yeah. Yeah, so, we can represent the maximum. The question here is, or at least in terms of, you know, you're likely to find the answer. The question here is, can this distribution represent values of theta in between? So imagine theta was, let's say I had four bits here, like we're doing here. And let's imagine theta was 2. Good. Now, 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. Right. And 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? Well it's n choose 2, which is, something. 6. 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. And 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 sub theta. because it will sometimes be able to generate with probability greater than 0, say, a value of all 0s. Yes, okay, good. That was what I was concerned about. And so this is, this is just an approximation, even in the simple case. Right. And so, and that's actually pretty interesting, right? So, yeah, 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, or optima, 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 theta greater than 2, 3, or 4, even. Got it. Okay. Alright, so I, so, given that, I think I can do the next two without too much more difficulty. So, for number 2 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. Mm-hm. And the chain gives you exactly that information without anything extra. Right. So I would put the 2 in the first box. Yep. And finally, well, there's only one box left. So I'd put the 3 in the middle box. But I guess the interesting to, to look at here is that it is surprisingly appropriate, right? So, that the coloring problem, it is specifically saying, well, the, my value depends on, you know, what's a good value for me depends on, I guess, several of my neighbors, not just one. Right. But I feel I could, you could capture a lot of the necessary information by finding a good dependency tree. Right. And it turns out that in practice, as our readers and listeners will see, from one of the resources that we're making available to them, 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 one, 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 l, 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. And 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. So, in fact, that gets us to the last bit of things I wanted to talk about, which are sort of practical features of mimic and what we kind of learn by thinking this way.

2.1.29 Practical Matters

Okay. So here's some practical matters, Michael. I mentioned one of them before, and that is that 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 Randomize 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. So, the chain example before 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 their basically being drawn in multiple directions at once. The quiz also brought up another point which is worth mentioning here was that mimic and with anything that tries to do this kind of search through probability [UNKNOWN] is that it's an issue of representing everything. That is it's not enough just to be able to represent a probability distribution of the [UNKNOWN]. You really want to be able to represent everything in between as you move through probability space toward your answer. This is the universal symbol for moving through probability space. You don't just want to represent here at the end and here at the beginning which is pretty easy because uniform distribution, but can you represent this point? Can

you represent this point? And 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 optima because you get randomize your search for optima. Hm. I see. 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 important sampling and rejection sampling And all these kinds of tools that we have for representing probability distributions that you ca 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 to me about Mimic or what to get out of here is that representing structure does matter. But you pay a price. And 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. Let me just share a fact with you Michael. Okay. I have run this algorithm on many, many, many examples and I've compared it to simulated [INAUDIBLE]. I've compared it to [INAUDIBLE] algorithms. I've compared it to randomized hill climbing. And it works pretty well for the sorts of examples I've come up with. And here's a little fact that I just want to give you. 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 for something like Simulated Annealing, but for Mimic, it might take only a hundred. And this is consistently true, so 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 Is it an argument for always using it? Can you imagine one now? because it might give a worse answer? Well, in practice it doesn't. So, these are cases where both simulated [UNKNOWN] and mimic, or randomized hill climbing or genetic algorithms actually eventually do find the answer. Mimic just finds it in orders of magnitude, fewer iterations. But you're counting iterations. Mm-hm. You didn't control for the fact that Different algorithms, can take different times for a single iteration. That's exactly right. So, what do you think? If I were to compare simulated annealing to mimic, which do you think take, the, takes more time for any given iteration? Well simulated annealing just does this little tiny step, right? It like, computes a bunch of neighbors and then does a probability comparrison, and then, takes a step. Mimic is drawing this sample, estimating a bunch of parameters, then yeah. I guess the other way around. It's drawing from a distribution. It's 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's, 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. Almost certainly. So, when would you think that MIMIC would still be worth using in a case like that, where we know that we can get to the answer, but simulated annealing will take orders of magnitude more iterations, MIMIC will take fewer iterations? So when would it still be worth it to take the one with fewer iterations even though east, each iteration is expensive? Prints algorithm, quadratic this that and the other. Oh yeah, grab at that part, hm.

2.1.30 Practical Matters Two

Here. Let me put it to you this way Michael. What is MIMIC giving you for all of that work that it's doing in building dependency trees and, and you know, running Prim's algorithm and finding maximum spanning trees. I'm going to say structure because that's the word that you've been using repeatedly. You get structure. And, and 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 for 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 or whatever it is you're doing in, in estimating your probability distribution. So, why would it be worth it to do that? What's the other source of expense? Well, so, all right. So there's all this computation within an iteration, but what it's, what it's doing, what it's trying to do is trying to find inputs that have high scores and so you do have to compute the scores for all those inputs. So the, the fitness calculation is very important. Right, 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 figure, compute it for some particular x, I pay some huge cost in time. Well, have you told us how many function evaluations there are in an iteration? because it seems like, it could be a lot in MIMIC. Well, you get one, basically for every sample you generate. Yeah and, and so, but so. I guess, you can compare iterations but you can also compare samples. Right, so they would depend upon how many samples you feel like you need to generate, and of course you can be very clever because. Remember, theta will generate a bunch of samples for theta plus

1. Mm-hm. So, if you keep track of the ones that you've 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. But let's imagine that at every iteration, you generate 100 samples. So, at most, you're going to have to evaluate f of x, 100 times. So, MIMIC is still a win over something else, if the number of iterations that it takes is 100 or more than 100 times fewer. Wow. Right? Yeah. So, can you think of functions, fitness functions, real fitness functions that might actually be expensive to compute? Well, yeah, sure. I mean a lot of this stuff that is, is important is like that. So, if you're trying to design a rocket ship or something like that, that. Fitness evaluation is, is doing a detailed simulation of how it performs. And that could be a very costly thing. Right, actually it's, that's a good example because MIMIC has been used for things like antenna design. It's been used for things like 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, 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, you know, to figure out the answer. Another case where it comes up a lot is where Your evaluation function, your fitness function, involves human beings. Oh, interesting, yeah. 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. [LAUGH] Yeah, they're not measured in megaflops. That's right. So, you end up with cases where fitness functions are expensive, something like this becomes a big win. And that's a general point to make, though, that when. 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 just in terms of whether you are going to over-fit 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. Cool. I get it. Okay, cool. So normally what we would do next is we would say what, whatever we learn, but I actually kind of think we just did that. [LAUGH] indeed. So, in fact, let me do something about that.

2.1.31 What Have We Learned

So, Michael what have we learned? That. I think you're right. Okay, well, Michael, I was happy to be able to share some of this with you and to remind myself of something I did 20 years ago. [LAUGH] And it answered the concern that you had, so it fit into the over all structure. Excellent, oh, it fit into the structure. Well done, Michael. Oh, I see what I did there. Yeah, very well done. Well, I will talk to you later Michael. Thank you so much for listening. Sure, thanks. Bye.

2.2 Clustering

2.2.1 Unsupervised Learning

Dun, duh, dun, dun, duh, dun, dun, dun. Dun, duh, dun, dun, duh, dun, dun, dun. Dun, duh, dun, dun, duh, dun, dun, dun. Dun, duh, dun, dun, duh, dun, dun, dun. Hi, I'm Michael Litman and welcome to clustering and expectation maximization. I dunno, I feel a little drama might help kind of wake us up this morning. How are you doing Charles? I'm doing fine. I just had a lot of ice cream. [LAUGH] Dude, you really shouldn't eat ice cream first thing in the morning. It wasn't first thing in the morning. That was after I had a muffin, and a breakfast burrito. So it was third thing in the morning. You frighten me. As you may recall, the mini course that we are doing now is on unsupervised learning and we haven't really kind of set down and, and introduced these concepts. So, I, I thought it maybe worth taking a step back, even though we are going to talk about some particular algorithms today, to talk more generally about what unsupervised learning is. So up to this point in the first mini course we talked about supervised learning. Supervised learning, in supervised learning we use labeled training data to generalize labels to new instances. And what unsupervised learning is is making sense out of unlabeled data. So when I wrote these down I started to think boy, they're more different than you'd expect just by sticking an un in front. How so? Well they don't seem to really, the definitions don't seem to have all the much in common. You know, it's not like this one uses labeled training data to generalize labels to new instances, and this one uses unlabeled training data to generalize labels to new instances. Like I would think that you'd have the definition of one. You just stick an extra un in there. Well, that's kind of true, right? So you, if you, if you do some data description and make sense out of unlabeled data. And then I give you a new piece of data. Somehow, using that description, you would know how to describe it. Maybe. I mean there's definitely some, some unsupervised learning algorithms that do some amount of generalization. Mm-hm. But but

it's, it's not as, it's not as unified. I think, in some sense the concern here is that unsupervised learning is just not as unified a problem, or as well defined, or crisply defined a problem as supervised learning. Oh, I think that's definitely true. So just as labels, and I think we talked about this in the intro to the whole course. That supervised learning, we can sometimes think of as function approximation, which is learning how to match inputs to outputs. Whereas, unsupervised learning is data description. It's about taking the data you've got, and finding some kind of more compact way of describing it. Sure, I buy that. And it makes a lot of sense.

2.2.2 Basic Clustering Problem

But even though the unsupervised learning problem isn't really that crisply defined, we can define a basic clustering problem and, and be somewhat crisp about it. So, that's what I want to do next. I want to say that one of the classic unsupervised problems is clustering, taking a set of objects and putting them into groups. And so, here's what we're going to assume. We're going to assume that we're given some set of objects X and we're given information about how they relate to each other in the form of inter-object distances. So we're going to assume that for objects x and y in the set of objects we have D(x,y) which is the same as D(y,x). Which says how far apart they are in, in some space. Now they don't actually have to be in some metric space. They just need to have this, this distance matrix D to find. So that makes sense, now let me ask you, so you said they don't have to be in a metric space. So you mean it literally does not have to be a real distance? It doesn't have to be a, a metric. Yeah, I don't, I don't think we're going to depend on that. I think we just need some kind of way of measuring how far apart things are. And it doesn't mean that they're embedded in some two dimensional space or three dimensional space. There's just numbers that relate them to one another. So they don't have to obey the triangle inequality, for example. I don't think so. I don't think we're going to depend on that in any way. Okay. So this is just like KNN because everything is like KNN. Where you have this notion of similarity and distance and that's where your domain knowledge is. So you've got a bunch of objects and your domain knowledge is these distances, these similarities between objects. Exactly, right. And in fact it's, it's surprisingly similar to KNN in that they surprisingly depend on similarness. Hm. So that's good. So KNN is similar to everything, so its distance is small to all things. [LAUGH] Wow, that is very meta. Alright. So that's, that's the input and the output that a clustering algorithm needs to, to create is a partition. Which is to say we're going to, I'll write it this way. That P of, P sub D. The partition function defined for some particular distance set D, for, for some object x, is just going to be some label, but it's such that if x and y are going to be assigned to the same cluster, the same partition, then they're going to have the same output of this PD function. Okay. That make some sense? That does make sense. What would be a trivial clustering algorithm based on this definition? Right, it's taking its input the objects and then the distances. And then it spits out partitions. A trivial one would be, well, a trivial one would be put everything in the same partition. Yeah. So that would be one. So that would be, that would look like this. P, for all x in the set of objects, PD of x equals, let's say, 1. So that means that all objects are in partition 1. We're all humans. That's right. Okay. Here's another trivial one. Every single object is in its own partition. Yes, good, that's the other one I was thinking of. Which maybe we could write like this. We'll just use the object name itself as the name of of the cluster. And now every object is in its own cluster. Now, we didn't define, as part of this, this problem definition, whether we should like the one where everybody's in the same cluster or the one that everyone's in different clusters or something in between, it's not really in the, in this definition at this level of detail. Hm. Well, that's fine. I mean, we can all be humans and we can all be unique snowflakes at the same time. [LAUGH]

2.2.3 Single Linkage Clustering Question

I think in part because clustering tends not to have one consistent definition it's very algorithm driven. So there's different algorithms for doing clustering and we can analyze each of those algorithms somewhat independently. There isn't really one problem and then there's a bunch of different algorithms for attacking that one problem. Each algorithm kind of has its own problem. So, I think the, the easiest way to go through and talk about unsupervised learning and clustering is to start going through some algorithms. So I'm going to start with single linkage clustering, because I think it's in many ways the simplest and most natural. Okay. And it has some very nice properties. So it's, it's, it's not a terrible idea, it's been very useful in statistics in a very long time. So, so here's how single linkage clustering goes, let's imagine we've got these objects. That we'll call them these little, these little red dots. We're imagine that they're objects and just because it would be hard to write down all the D values between them. Let's just literally use the 2 dimensional space, the distance on the slide here as the distances. Okay. Okay. So, you can, so if you were,

if you were asked to cluster this how would you do it? Like, what do you think the, the groupings ought to be? Well, just staring at them they're either two or they're four. Mm. Or maybe, there's five. So, I would probably put the three on the left that there together, together. And I would put the four on the right that are together, together. Or I might notice that the two at the top of the ones on the right are more together than the other two and sub divide them some more. But, if I were just far enough away, I would definitely say so that seems like a reasonable clustering, but also the one that took all four of those and put them together is a reasonable clustering. Right. Yeah, yeah. That means that sort of agrees with what my eyes are thinking as well. Alright, so so it'd be good if we can recover that kind of structure algorithmically, so let's let's talk through this, this algorithm. It's called single linkage clustering, sometimes SLC. Or "slick". nice. It's a, it's a hierarchical agglomorative cluster of algorithm, or HAC. [LAUGH] I like that acronym even better. That's an Andrew Moore joke, for what it's worth. Oh, it's a slick HAC. Nice. All right. So here's what we're going to do. We're going to consider each object a cluster to start. We're going to do this interatively. So we start off with N objects, like we have here. One, two, three, four, five, six, seven. And we're going to define the inter cluster distance as the distance between the closest two points in the two clusters. So in the beginning, all the points are in their own clusters. All the objects are in their own clusters. So the inter. Cluster distance is exactly the interobject distance. Okay? Okay. But, little by little, we're going to actually be aggregating things into larger clusters and we have to define what it means for how, how close is one cluster to another. And we'll say the close, the closest between two clusters is the closest of the closest two points between those two clusters. All right. So, so now what we're going to do is we're going to iterate. We're going to say merge the two closest clusters, because, you know, they belong together. And then we're just going to repeat that n minus k times To leave us with K clusters. So K is now an input to this algorithm. Okay. All right. So help me out. Let's step through this. What what would we merge first according to this algorithm? Well, they're either on cluster I, according to my eyesight, which is, of course, perfect. I would take the two left most ones, upper left most ones. Probably and merge them together. They look closest to me. Alright let me label them just to make your life a little bit easier. Oh that'd be nice. Alright. SO what do you think? I would bring A and B togheter. ALright. They're now a cluster. So we had seven clusters now we have six. Oh by the way we're trying to get down to two. Okay and then I would put C and D together. Yeah, that's what I was thinking too. Then I would probably want to put c, d, and e together, because they're in alphabetical order. [LAUGH]. All right, well, just to be concrete about this, why, what is the next step that we're supposed to do? What we're supposed to do is say which pair of clusters is closest. And again, the, the, distance between, say, I don't know, the a b cluster and the c d cluster is not the average over all the points or the furthest points but the closest ones. So, like I think c and d, the,the distance between the a b cluster and the c d cluster is exactly the distance between b and d in this case. Mm-hm. Does that make sense? Yeah. Alright. So we know what's the currently closest pair of clusters. from, from where I am looking, it's either e to the c and d or g to a and b and they look very similar to me. E to C and D or G to B and A, that's what you said? Yeah. Probably G to B and A, staring at. I tried really hard to make it so that they're be an easy answer, but you're right, I, they're really close. So let's, let's I think it doesn't matter too much. we'll, we'll say that that one's neck. So now we'll say, now we'll do a quick quiz. Okay. So what merge does, single link clustering do next? Just give the pair of points I would be connecting. Okay. Say, separated by a comma.

2.2.4 Single Linkage Clustering Solution

Alright. So, what do you think? Well I'm actually pulling out a piece of paper To measure things on the screen? You know, because my screen is actually flat on the ground or flat on the table. So, it looks like e and f are closest. Yeah, though I think we probably should be willing to accept ef, df, or bg. No. Because they're all really similar. E and f is close, b and g is close. But d and f is actually pretty, is clearly farther than the rest of them. Proved by. D and. Really? because it looks like an equilateral triangle to me. Maybe it is, actually. I take it all back. Yeah, actually you're right. You're right. It's the angle. Well actually d and f I think are closer than e and f. Alright. Good! So that yeah. So the, so the, I would say that any of those, any of those would be fine next. Mm-hm. And, I think interestingly won't change the answer, in the end. In the end. Yeah. So let's let's continue with this process now that the quiz explanation is done.

2.2.5 Single Linkage Clustering Two

So let's finish this clustering process. So what were you thinking of doing next? Well, D and F and B and G, we both thought were pretty close. So let's just do D and F. Well, D and F doesn't matter much anymore because they're already in the same cluster. Oh, that's a good point. That's a good point. So what is their

inter cluster distance? It's zero. Yes. because they're in the same cluster. That's a fun point you make there, Michael. Sure. How about B and G then? I like B and G. And what do we do next? Well, we have two, two nice clusters. So, actually if we were to merge the next cluster. We would end up with, I guess b and d together. Doesn't really matter because it's going to merge the last two clusters together. Yeah. So, in particular what I was what I was getting at is we don't do any emerging next because now this repeat loop is done. So that's, this is what we end up with assuming that we had started off setting K equals two. Um-hm. Then, then we're done at this point. We have what looks like a backwards R and a funny looking, actually they look like Hebrew letters to me but that's probably not a universal perspective. So just one other thing to point out quickly. So we can represent the series of merges that we did using a structure that's kind of like this. We merged a and b first, and then we merged c and d. And then we merged, I think d and e? Yeah. Or the c, d, and e. And then we merged e and f. And then we merged g and the, and the ab cluster. And then that left us with the two clusters. So, this, in some sense it, we've captured the same information about what's connected with what. But this is kind of a nice way of representing it, because it actually gives us a hierarchical structure. A tree structure. A kind of hierarchical agglomerative structure. And there's lots of applications where it's actually useful to have that, that extra bit of information. because we can now very easily, look at remember, in the beginning you said that you could see three clusters? Well you can see if you just cut this tree structure here, we get, we get the three. Oh. Not quite. We almost get the three clusters that you wanted. Here it looks like f is separated from the others. It depends which, which distances we thought were closer. And also if you combine those two clusters again, the last two clusters, then basically you, you have a true tree with a single root. That's right. And, and that's kind of, that summarizes all the different cluster structures that you can get out of single link clustering. Hm, I like that. So I have a question, Michael. Shoot. Well I have two questions. The first question is do you know the difference between further and farther? I do, though. I sometimes use them interchangeably. Yeah, it turns out interestingly enough that that definitions are further and farther switch every 100 years or so Hm And unfortunately I was born in a time period where they're switching meanings again and so it drives me nuts. But in any case for our listeners, further farther means physical distance, and further means metaphysical or metaphorical distance. So one lives farther down the road, but goes further into debt. And you should almost never confuse the two, the people currently are using further as if it means farther. So I have a question here for you, which is you define the inter cluster distance as the distance between the closest two points in the two clusters Yep, that's what it says right here. Yeah, as if that were an immutable fact of the algorithm. Is it, or could we do something else? What would happen if we picked average distance or distance between the farthest two points as opposed to furthest two points or some other measure? [LAUGH] Well, we don't know if this is physicaly distance or metaphotical distance, right? Or metaphysical distance, it's just d. That's true. So it's like, we'll call it forthest, with an o. [LAUGH] Please don't. The forthest point. Yes good. And average is another one. That's true. Yeah, these are different things you could define. I, my recollection of this is, they're no longer single link clustering. Single link clustering is defined to be closest. But you also have other things like average link clustering, which I think is the one with the average. And then there's maybe max link clustering. I forget exactly what it's called where it does the for list. I think I saw a talk once as I didn't understand which had to do with median distances. And that somehow you could prove interesting things because of it as opposed to the mean, which is what I think you mean by average there. Yeah. Average meaning mean. Yeah, I mean, generally, things like median are really good if the numbers on the distances don't matter, just their orderings. Mm. So, sometimes people dif, describe that as being metric versus non-metric. Non-metric, median is usually a non-metric statistic. Average is, is a metric statistic. It actually, the, the details of the numbers matter a lot. Okay. Well that all makes sense I guess the basic idea is that what I was saying before that the distance was some notion of where your domain knowledge comes in. I guess it's also true that the, how you define intercluster distance is also a kind of domain knowledge. It's another parameter to the algorithm. That seem fair? Yeah. Okay.

2.2.6 Running Time of SLC Question

So there's a couple of really nice things to be able to say about single-link clustering. So one of them is that it's deterministic, right? So we run the same algorithm, and unless there's ties in the distances, it, it will give us an answer, which is, which is nice. Doesn't require any kind of randomized optimization. Another thing about it is that if you're doing this process in a space where the distances represent edge lengths of the graph then this is actually the same as a minimum spanning tree algorithm. Hm. And another nice property is that the running time of the algorithm is actually relatively friendly and can be characterized so let's, let's think that through. So let's imagine we've got n points or objects that we want to cluster. And we've got K clusters that we want as our as our target at the end. Which of these different running times

best characterizes the running time of single link clustering?

2.2.7 Running Time of SLC Solution

All right. What you got? So I look at this and believe it or not I, I came to an answer by two completely different ways. Okay. And do they, do they agree? They, they happen to agree, but one of them is kind of lame. It's sort of the thing that you would do if you were doing the SAT or the GRE and you wanted to quickly get a guess at the answer. Oh, I see. And it's that k, very small k and very big k are kind of both easy. It's the one in the middle where it gets hard. And so any function that depends on k directly, in a way where as k gets bigger it gets harder, as k gets smaller, it gets easier, is probably not right. Which made me think was probably n cubed. And so then I tried to think about how the algorithm worked. And so, the worst case way of thinking about the algorithm is at every single time you have to figure out which two points are closest to one another. Alright, hang on a sec. So we're going to repeat k times, and the hard case is when it's, like, n over twoish, right? Right. And what, what is it that we do at each step? Well, we find the two closest points. Good. Right, and that, and how long, and yeah, and how long does that take, it's exactly, it's n squared. because you have to do all possible combinations. It's a little better than n squared, right, it's, you know, but it's big O of n squared. Right, and it's possible that if we store those distances really cleverly we can get it down a bit more, but, yeah, n squared seems like a nice way to characterize it. So don't we need to do some work to merge the clusters together so that the next time through we get the right set of distances that we're looking through. We could do that and that would work fine and that's probably the right thing to do, but just kind of at a high level, what you just really want to do is, you just want to find the closest two points that have two different labels. Okay. So what I associate with every, with every object x is the label that it currently has. So you're going to look at each pair. Ignore it if the, the labels are the same, on both of the clusters that the two objects are in. Mm-hm. And otherwise find the minimum. Right. And what, and I can. That sort of works. And I could do that in time linear in the number of pairs. So it's still n squared, I don't have to sort or anything like that. True, though we could, it could be a little bit smarter to not reconsider the same pairs over and over and over again. That's right, you could. If you could use, you could probably use something fancy like Fibonacci heaps, or hash tables or something. And in fact, I know that there are people who've gotten entire PhDs on clever ways of doing this without having to consider all points. By dividing points up into little boxes where things tend to be closer to one another. Actually looks a lot, very hierarchical as well. Fair enough. Okay, so we do, we do, we do a, that takes n squared time. How many times do we do a? Well, we do it about n times. Alright. So that gets us the n cubed. And is it, is it clear that the other ones are definitely wrong? Well, since we can do this in n squared time, and the number of clusters can be large, the first one is a bad answer. The second one is a bad answer. How about the last one? Is it, is it possible that we could actually do this in linear time? no. Well, for, for, for very small or very big k, maybe. Like, so k equals 2, yes, that's easy, that's linear, k equals n, that's easy. because we started out that way. I see, but for in between k's. because you still have to find the closest distance, and there's, you have to find distances and the only way to do that is to consider all the distances. Yeah, at the very least the, the number of inputs that you need to even sniff at is quadratic. Right. Because the d can be any kind of structure. So, yeah, okay. So this is really the only answer that, that is even close. But I think it could come down a little bit from n cubed. Oh. I'm sure it could. I mean, because I just came up with the silliest, the, the simplest algorithm that you could think of that wasn't absolutely stupid. So I think that would. You could certainly do better than n cubed. But not much better than n cubed. All right. End of quiz. Yay.

2.2.8 Issues With SLC Question

Here's another quiz. Just to kind of practice this idea of what singling clustering does and to think about it a little bit more. But also to wonder with, if maybe is it, isn't exactly what we think of when we think of clustering. So so here's a set of points that I drew in the upper left corner here in black. And the question is, if we run singling clustering with K equals two. Which of these three patterns will, will we be getting out? Okay, I got it. All right, good, let's let's give everybody a chance to think about it. Okay. Go.

2.2.9 Issues With SLC Solution

All right, so, so how does this work, Charles? How do you do single link clustering? Again, assuming the distances are distances in the plane, just because it makes it easy. Right. So you just keep finding the clusters that are closest, and you defined it as where the, the two closest points in two clusters are what

makes them, is what defines their distance. Mm. And so if I look at this, if you look at, look at the big outer circle. you'll, you'll notice that each of, you start with any one of those points and it's always going to be closer to one of its neighbors on the circle than to, say, one of those four points in the middle of the circle. Right? They're close to each other. They're close to each other. If we start doing mergings, it, it ought to look something like this, right? Where we've got, I don't know. [SOUND] Maybe. [SOUND] But little by little it's going to be linking these things up. Mm-hm. And that's the key thing there that you were drawing before that the bridge between the two big circles, every point there is always going to be closer to one of the points on the outer circle clusters than it's ever going to be. To anything else. And so, you're going to end up making, one big cluster, out of the outer shell. Which is kind of weird. It is a little bit weird. You get these kind of stringy clusters because two things are close to each other if they're close anywhere. Right. So, I don't know. To my eye, the best one is the bottom right. But it wouldn't do that because k equals 2. Right. So to my, for what's, for what's left, I guess I like the upper right. But that's not what single lane clustering would do. It would actually kind of walk around the circle and link things up. Like in the lower left.

2.2.10 K Means Clustering

So we're going to talk about a different clustering algorithm that, that at least doesn't have that weird property and it has some other nice, positive properties. But, it, there's trade-offs. There's always trade-offs in clustering. So k-means clustering is what we're going to talk about next. And, and the, the feel, the basic flow of the algorithm is like this. We're going to first pick a K. K is going to be the number of clusters that we're going to try to produce. And the way it's going to do it is by picking K of the points at random to be centers. And then we're going to repeat the following process, each center is going to claim it's closest points. So, we're going to cluster all the points based on how close they are to the k centers that we've defined. Then, we're going to recompute, based on that clustering that we just did. Recompute the centers. And the center would be the average of the points in its cluster. And then we're going to tick tock back and forth. We're going to go back and repeat 'til convergence. For the new centers claiming their closest points. The new groups of points computing their average and back and forth like that. So let's step through an example and then and then we'll try to analyze what this actually does. Okay, that makes sense. Alright. I'm going to do K equals two. In this example the same set of points that I used in the previous example. And I'll just you know, kind of randomly. Choose two of them to be the two clusters the red cluster and the green cluster alright? Okay. So that's this, that's this first step done. Now what we're going to do is each of these centers is going to claim. Their closest point. So we are going to take all the points in this space and assign them to one of the other of these two clusters. And it ends up looking like this. Alright? So this is now the points that's closest to the green centers have all been put into a green blob. The ones that are closest to the red center, all get put into the red blob. Okay. Okay? So, that's, that's step two. Now what we do is recompute the centers. So the center of the red blob is still pretty close to where it was before but if you look at the green blob, the center of the green blob ought to be to the right. You see that? I do, I do. In fact it should be a lot more to the right cause there's a whole lot of points on the right. Yeah, though, I did this sort of by eye, so it doesn't, yeah, you're right. It should be, it should be closer to the clump on the right, but I, I didn't quite do it that way. Okay. So I moved it just a little bit more to the right. See that? Mm-hm. But now we can repeat this process. We can say," Okay, now everybody join the group that you're closest to." And one nice thing that happened now is that the group of points on the left all joined together in red and this sort of weird hammery thing on the right became green. And now we're going to recompute our centers again. We're going to say, where's the center of the clusters given the way that they've been painted. And that, again, move things a little bit to the right in both cases. We ask every point to join the team that they're closest to. And we get that. And that actually turns out to be exactly the same clustering that we had a moment ago. So when we recompute the centers, they remain unchanged. And so we've converged. So it, it seems to have clustered things reasonably given that I didn't actually run this. I just did it by hand. Okay. Can I ask a question? Oh, please. Okay. You seem to have drawn this in such a way that the centers are always one of the points. But that's not really meant to do, is it? No, no. It's definitely not what I meant to do. It's it realy is just. It, it, doesn't. The center is not necessarily a point that's in the collection of objects. Right. Yeah thanks for pointing that out. Okay, so this makes sense and that look better than what single linkage clustering, or single link clustering, or single linkage or whatever it is called, clustering did. At least for this kind of example, yeah, it, it produced kind of more compact clusters without giant holes in them. M-hm. So so do you have any questions about this? About what it does. Yes. So, so, [LAUGH] I think I might have some questions Michael. And they may even be questions you'd like to here. So I asked one question about what the synergy looked like. So I have a couple of questions, one is does this always

converge? Does it always come up with a good answer? Yes, those are good questions.

2.2.11 K Means in Euclidean Space Part 1

Alright, so what I'd like to do is work up to a proof that K-means does something good. [LAUGH] And to do that I think it's helpful to have little bit more precise notation than what I was doing before. So here we go, let's, we're going to work up to doing K-means in a Euclidean space. And the notation that I'm going to use is, we've got at any given moment in time in the algorithm, there's some partition, p super t of x. And these clusters are defined the same way that we did before, which it just returns the, you know, some kind of identifier for cluster x is in and this is at iteration t of K-means. We also have another kind of more convenient way of writing that which is C sub i of t, which is the set of all points in cluster i. It's really just the same as the set x, such that p of x equals i. Does that make sense? Yeah, okay that makes perfect sense. You've got some kind of partition, and everything in the same partition belongs together in something you're calling C for cluster. Good, and we're also going to need to be able to refer to the center of a cluster because we're in a Euclidean space, it's meaningful to add the objects together. So, take all the objects that are in some cluster, Ci at iteration t and divide by the number of objects in that cluster. So just summing those points together. This is also sometimes called the centroid, right? Right, so it's the mean or the centroid. Yeah it's like. That's right. I mean in one dimension it's definitely the mean. In higher dimensions it's like a per dimension mean. Oh, so in fact you're going to end up with K means. Oh, I see what you did there. Mm. So now, here's an equation arrow version of the algorithm, that we start off picking centers in some way for iteration zero, and we hand it over to this process that's going to assign the partitions. In particular, the partition of point x is going to be the minimum over all the clusters of the distance between x and the center of that cluster. Alright, so it's just assigning each point to its closest center. Does that make sense? Mm-hm, and that all those bars with the sub two just mean Euclidean distance. Yeah, we're just, that's right, exactly. We just computing the distance between those two things. And we hand that partition over to the process that computes the center. So it's, it's now using this other representation of the clustering Ci, it's adding the points together, divided by the number of points in that cluster, and it hands those centers back to this first process to recompute the cluster. So we're just going to be ping-ponging back and forth between these two processes. Okay. Good? Sure, and t keeps getting updated after every cycle. Right, yes. So, this is where t gets incremented, t plus plus. So this is the algorithm, and an interesting fact about it is: Do you remember when we talked about optimization? I do remember when we talked about optimization. And optimization is good because it's finding better and better answers. So if we can think of this as being kind of an optimization algorithm then that could be good. It could be that that what we're doing, what we're going around here is not just randomly reassigning things, but actually improving things. Okay. Well how are you going to convince me of that? Alright, let's do that.

2.2.12 K Means as Optimization

We're going to look at K-means as an optimization problem. So remember when we talked about optimization we worried about configurations, I think we called them inputs at that time but I think it's going to be helpful to think of them as, as configurations here. There was a way of scoring configurations and we were trying to find configurations that had high score. And for some of the algorithms we needed a notion of a neighborhood so that we could move from configuration to configuration trying to improve. Mm-hm. So in this setting, the configuration, the thing that we're optimizing over, is the partitions, the clusters, and we also have this kind of auxiliary variable of where the centers are for those clusters. And what we need now is a notion of scores. How do we score a particular clustering? So do you have any thoughts about what would be a better or worse clustering according to the kind of K-means algorithm? Well, the thing about what you were saying earlier, about what we were trying to do with creating these clusters, and I look at this notion of a center, so something pops in my head. So, you would like to have centers or partitions that somehow are good representations of your data, and why does that matter? Because you said in the very beginning that we often think of unsupervised learning as compact representation. So if you want to have a compact representation it would be nice if you don't throw anything away. So, I'm going to say that a good score will be something that captures just how much error you introduce by representing these points as partitions or in this case as centers. Does that make sense? Okay. I guess that's a, that's a perfectly reasonable way to think of it. I, another way to think of it is in terms of error, right. Yeah, which I guess is the same idea. Like if we're, if you think about the object as being represented by the center of its cluster. Mm-hm. Then we want to know how far away from the center it is. Right. And the farther away it is, the more error you have in representing it. Right. So, here is a concrete of writing down what we think the

scoring function could be. So we're going to say that the error, it's kind of the negative score, right? This is something that we want to minimize even though generally, we've been talking about optimization as maximizing. Here is something we want to minimize. That if you give me a particular way of clustering it and you define the centers based on, say, that cluster, what we're going to do is we're going to sum over all the objects the distance, the square distance actually, between the object and its center. Right? So p of x is the cluster for x and center sub that is the center for that cluster. So that drives home the idea that we're talking about Euclidean distance here because x would have to be in the same space as the centers are by definition. Okay. Yeah, and I'm not sure exactly how we're going to define neighborhood. But one way to define neighborhood is that the neighborhood of a configuration, which is a p and a center is the set of pairs where you keep the centers the same and you change the partitions. Or you keep the partitions the same and you move the centers. So you're basically changing one of these at a time. Oh, that's very clever, Michael. Thanks.

2.2.13 K Means as Optimization Quiz Question

Great. So now, looked at this way, you can think of the k means procedure as actually moving around in the space and it follows one of the optimization algorithms that we talked about. So here's three of the algorithms that we talked about: hill climbing, genetic algorithms and simulated annealing. These are all randomized, optimization algorithms and one of these is kind of a lot like K means so which one do you think it is? Okay let me think about it.

2.2.14 K Means as Optimization Quiz Solution

What do you think? I think it's hill climbing. Me, too. It's certainly not simulated annealing because there's no annealing. Yup. Simulated or otherwise. You might think it's genetic algorithms because you've got all of these centers that are all moving around at once, but they're all a part of the ultimate answer. They're not populations. So it's really hill climbing. You just take a step towards a configuration that's better than the configuration you had before. That's right, except what we haven't said yet is the way we defined hill climbing, it actually evaluated the score and chose a neighbor that maximized the score. And we didn't show that these steps actually do that. We just defined them in a particular way. Here's the wicked cool thing. It really does do hill climbing. It really does find the neighbor that maximizes the score, minimizes the error in this case. So let's show that because it's kind of neat. Every time I see this I'm always surprised and delighted. Oh, I'm looking forward to being delighted.

2.2.15 K Means in Euclidean Space Part 2

Alright so let's see if we can figure out what happens to this E function as in one step we update the partitions and in the other step we update the centers. So let's look at the partition one first. The way that this partition is being defined is we, for each point loop over all the clusters and find the cluster whose center is closest to x in terms of squared distance. What happens to E when we do that? Well we move a point. ¿From one cluster to a different cluster only if it causes the square distance to the center to go down. So that means that the error, either stays the same if the point stays in the same cluster or it goes down if it goes to a better cluster. That makes sense. So, can only go down. Well, it can never go up. It's different than saying it can only go down. Agreed, because it could stay the same. What happens if it stays the same? I guess, not necessarily anything interesting. Right. But, certainly, when we converged, it stays the same. Right. Alright. Now let's look at the other side here. So that's what happens when we move things into partitions. And in some sense that seems easy. Because we only move things if it causes the error to go down so it really is a lot like hill climbing Mm-hm On this side though what happens when we move the centers so could it be the case that when we move the center to some other place that the error goes up? No. And why do you say that? Because the average is going to be the best way to represent a set of points. On average, we should be able to demonstrate that. I think we already have. We did this earlier in, in the course when we were talking about minimizing the squares. You are right. So, basically, you could take that equation and you could just take the derivative of it. Set it equal to zero and it will turn out to be exactly the average. You're right that's exactly right. The error equation E that's right, yeah so this is like really kind of neat. When we moved points around we move it to reduce error and we move centers around we always move it to the center even though this is a continuous space we always jump to the center that actually has minimum error under the assumption that we're holding the partition steady. Right. So this is just great. So put them together, you're guaranteed to be, let's see, what's the math term? Monotonically

non-increasing in error. Monotonically non-increasing in error, very nice. And does that imply that thing has to converge? Could we be monotonically non-increasing in error forever? You could, in some worlds, but not in this world. I, I think I could argue that. Alright. So a monotonically non-increasing function, is a function that never gets bigger. So you could end up in a case where you hit some point, like say zero error, and you keep going. So, why wouldn't that happen here? Here's the argument. You ready? Sure. There are a finite number of configurations. Hm. There have to be a finite number of configurations 'cause there's a finite number of objects, and there's a finite number of labels they can have. Mm-hm. And once you've chosen a label for a bunch of the objects, the centers are defined deterministically from that. Right, so even though it is an infinite space as we're tick-tocking back and forth, if we don't move the partitions then the centers are going to be where they were. So, the centers are quite constrained even though it's continuous. Right, so, the only tricky part to that is that you could have a point that can go to either of, let's say, two partitions, because the distance is the same. So you have to have some way of breaking ties, such that you always get the same answer. For example, I will just say that if I, as a point, can go to any of two partitions, I will pick whichever one has the lowest number. Good idea. So breaking ties consistently and you gave a particular role for that is going to guarantee that we at least don't kind of spin around not improving. Right so let's see if what I just said makes sense. So tell me if you buy this, they have a finite number of configurations. If I always break ties consistently and I never go into a configuration with a higher error, then at some point. Basically, I will never repeat configurations. I'll never go back to a configuration that I was at before. And at some point, I'll have to run out of configurations because there are a finite number of them. Yeah, nicely done. So it converges, in finite time no less. Finite time. Could it be exponential time? because there is a lot of possible partitions. So how many different configurations are there, there K to the N because you can assign K to the first object, K to the second object. K to the third object, so k times, k times, k times, k times, k all the way up to n, so that's k to the n. So, that's a lot of possible configurations, but regardless, there's a finite number of them and I suppose in practice, it's not like you would look at every single one of them. Because you're going to jump around very quickly because, of the way distance metrics works. They point close together, they're always going to be close together. So you're never going to try, assigning each one to all possible configurations if they're close together. Yeah, it tends to converge pretty fast. So let's summarize some of these properties. Okay.

2.2.16 Properties of K Means Clustering Quiz Question

Alright, so let's summarize some of the properties that we've been talking about just so that they're actually written out. One is that each iteration as we go through this k-means process. Each iteration is actually polynomial. It's pretty fast. We just have to go through and look at the distance between each center, each of the k centers and each of the n points, to reassign them. And then we have to run through all the points to redefine the clusters. There could be an extra factor of d in here, if these are d dimensional points. Sure. That makes, that makes sense, alright? Then you were just going through the argument as to why the number of iterations would be finite and exponential. And the exponential is like k to the nth. And what you said was the first of the n objects can be in any of k clusters, and the second can be in any of the k clusters, and the third can be in any of the k clusters. So we get a total of k to the nth different ways of assigning points to the k clusters. But I do think what you said in response to that is right which is in practice, you're not going to do an explanation under iteration. Right right, in practice it tends be really, really quite low. It just kind of clicks into place. Because it's a, it's the same thing for reason why the average is, even though it's a continuous space, it's still finite. Distance is a really, really strong concern. Right, and so the error on each iteration is going to decrease if we break ties consistently. And as you've been pointing out to me, there is one exception here whereas if things are assigned to clusters, it could be that things don't improve, that it stays exactly the same, but that can only happen once. Because then the clusters are going to get assigned according to the consistent tie breaking rule, and the next time through we're going to see that we've converged. But there is something that we didn't talk about that I think is important to think through. So here's a set of six points. Okay. If I was going to ask you to make three clusters of this, what would you do? I would put a and b together, c and d together, and e and f together. Indeed. Alright. So that is the, that's the optimum here. So, the question is, write down a way of assigning the three cluster centers to points, so that it will be stuck there and not get to the, the clustering that you just found. So, just write down a, b, c, d, e, f. Three of them separated by commas defining where the centers should start so that it will actually not make progress. Okay.

2.2.17 Properties of K Means Clustering Quiz Solution

Okay, so what do you think? I think the answer would be a and b and any of those four: c, d, e or f. Alright. So let's think about what would happen in that case. So, if we start off the centers there. The first step is going to be for every point to join whichever cluster it's closest to. So, a is just going to be with a. Mm-hm. B is just going to be with b. And then d is going to slurp up all these other four points. Right. All right. So now in the next iteration, it's going to recompute the centers. And a and b aren't going to change. This cluster, the center's going to change to here. And now it's never going to make any additional progress. So those are the three clusters it'll find if it starts off with those kind of bad initial points. So how would we go about avoiding that sort of thing? yes. I was going to ask you exactly that question. So, given that we're thinking about this as a hill climbing process, that's a local optimum. And we had a pretty good way of dealing with local optima and that was random restarts. That's one possibility. Another thing that people sometimes do is they'll do a quick analysis of the space and actually try to find initial cluster centers that kind of are spread out. So pick a cluster center and then pick the next cluster center to be as far away from all the other cluster centers as you can. So kind of do like the, I don't know, the convex hole of the space and try to pick points near the corners or something like that? Yes, yeah, that's right. Hm. So I think you've done another thing too, now that I say that out loud. Which is, you've been choosing these random places to start your centers as always being one of the points. I guess you didn't have to do that. Yes. But it probably helps that you do. It certainly keeps the centers near the points. Another thing you can do to get things started is just randomly assign everybody the clusters but that can often have the property that all of the senders of those clusters end up being kind of really close to each other. So, by picking points to be the clusters it does have a tendency to spread things out. Okay, that makes sense. That's not a proof, though. No, a proof by that makes sense. [LAUGH]

2.2.18 Soft Clustering Quiz Question

All right. We're going to talk about another clustering method. And just to transition from the previous one, let's, let's take a look at this data set here. We've got seven points. A, b, c are all close together on the left. E, f, g are all close together on the right. And d is equidistant from, say c and e. In the k means clustering setting. If we're looking for two clusters. What's going to happen to d? So, one possibility is it ends up with the group on the left. Another possibility is it ends up with the group on the right. Another possibility is that, depending on what happens from the random starting state, it might end up on the left or the right. And then finally, it's shared between the two clusters. because it doesn't really belong in either of them. So it's sort of going to partly join both. Oh, I see. So d is exactly in the middle between c and a. Okay. That's what I intended to draw, yeah. Okay, good. I think I can figure out the answer.

2.2.19 Soft Clustering Quiz Solution

Alright. So what do you think? I think the answer is, three. It sometimes would be left and sometimes would be right, depending upon where you randomly start. So for example if you start with your random centers on a and b. Then I think what'll happen is, d will end up on the right, as that point gets dragged over towards the weight of the right, it'll drag d with it. If you started out with both points on f and g, then I think d would get dragged to the left cluster as the left most cluster got dragged to the left. I think if you started it with a and g, then it would depend upon how you break ties. And since I always break ties lexiographically, because I like saying the word lexiographically, it would end up on the left. So I think the answer has to be the third choice. But, I'll say one thing, I wish it were the fourth choice. I would like to grant this wish by talking about the next clustering algorithm which, in particular, does soft clustering. And, soft clustering allows for the possibility that a point can be shared you know, I'm a little bit of this, I'm a little bit of that. I'm a little bit of country, I'm a little bit of rock and roll.

2.2.20 Soft Clustering

So to do soft clustering, we're going to use a similar trick to what we've used in some of the other lectures, which is to lean on probability theory so that now points instead of coming from one cluster or another cluster can be probabilistically from one of multiple possible clusters. Does that seem like a good idea? It does. I feel a song coming on. Lean on probability when you're not strongly believing one thing. No, that doesn't work. Yeah, that almost worked. Almost. Yeah. So that's because it's soft clustering or soft assignments instead of hard ones. I like it. All right. So to do this, we're going to have to connect up

a probabilistic generator of process with the data that we actually observed. So let's assume, and there's many ways to go down this road, but we're going to go down the road this way. Assume that the data was generated by, what happens is we're going to select one of K possible Gaussian distributions. So we're going to imagine that the data's going to be generated by draws from Gaussians, from normals. Mm-hm. Let's assume that we know the variants, sigma square, and that the K Gaussians are sampled from uniformly. Okay. And then what we're going to do is that given that Gaussian we're going to select an actual point, an actual data point in the space from that Gaussian. And then we repeat that n times. So if n is bigger than K then we're going to see some points that come from the same Gaussian, and if those Gaussians are well separated they're going to look like clusters. Assuming they have very different means. Alright. That's what I mean by, we'll separate it. Yeah exactly so. Oh, yeah, yeah, yeah, okay. Good. Alright, and in particular, what we'd like to do now is say, alright, well, now what we're really, happening, is, we're given the data, we're thinking kind Bayesianly, right, we're given the data and we want to try to figure out what the clusters would have been to have generated that data. So we're going to try to find a hypothesis, which is this case is just going to be a collection of k means, not to be confused with K-means. Mm. That maximizes the probability of the data. Right? So find me K-mu values, which are the means of those Gaussian distributions. So that the probability of the data given that hypothesis is as high as possible. And this is an ML hypothesis. And ML of course stands for Michael Lipman. I don't think that's right. Machine learning. That's closer, but not quite right. Maximum likelihood. That I think is correct. Alright. So that's now the problem setup. I didn't actually give you an algorithm for doing this, but presumably it's going to depend on various kinds of things and probability theory and optimization, but is it sort of clear what we're shooting for now? It is, it is. And I actually think the fact that we're looking for k means probably menas that we are going to end up tying it back to k means. Maybe so, but again, it's a softer kind of k means, it's a softer, gentler kind of k means. Mm. I think some people call it K Gaussians. Does that sound right? No. Alright then. I mean it sounds correct, but it doesn't sound right. [LAUGH] Alright then.

2.2.21 Maximum Likelihood Gaussian

So one of the things that is going to be helpful as a subcomponent to this process is to be able to say. Well, if we've got a set of points in some space, and we're trying to find the maximum likelihood Gaussian with some known variant. Mode is the maximum likelihood Gaussian. What is the best way of setting the mean to capture this set of points? So fortunately this is really easy to do. And the reason that it works out this way is the same reason that we've talked about in several of the other lessons. But the maximum likelihood mean of the Gaussian, this mu that we want to set, is the mean of the data, the mean is the mean. That's pretty mean. And it's no mean feat that it works out this way. [LAUGH] And, [LAUGH] what? Oh, just well done. So, in particular, this is really easy to compute. If we know that all this data is coming from the same Gaussian, then finding the mean that maximizes the likelihood is just computing the mean of all the points. Right, we kind of did that, just a few slides ago. Exactly. Okay, alright, so given a bunch of data points that I all know came from some Gaussian, I can compute the mean of that Gaussian by actually taking the sample mean, and I could mean it, okay. So the tricky thing of course, is what happens if there's k of them. How do we set the k different means? And our answer is going to be, there I just wrote it. Do you see it? Nope. That's because, it's hidden variables! Oh. My favorite kind of variables. Variables that you don't have to see. So what we're going to imagine is that the data points, instead of just being x, it's actually x and a bunch of random variables that are indicator variables on which cluster that x came from. So it's not just x anymore it's x and let's say a bunch of zeros and then a one. Corresponding to which cluster generated X. Now of course if we knew that, that would be really useful information. We, we're going to have to do some inference to figure out what those values are. But, the, concept is that, by adding these extra hidden variables in, it really kind of breaks up the problem in a convenient way. Okay.

2.2.22 Expectation Maximization

So, this is going to lead us to the concept of expectation maximization. So, expectation maximization is actually, at an algorithmic level, it's surprisingly similar to K means. So, what we are going to do is, we're going to tick-tock back and forth between 2 different probabilistic calculations. So, you see that? I kind of drew it like the other one. Mm Hm. The names of the 2 phases are expectation, and maximization. Sort of you know, our name is our algorithim. I like that. So, what they're going to do is, we're going to move back and forth between a soft clustering, and computing the means from the soft cluster. So the soft clustering goes like this. This probablisitic indicator variable, Z I J. Represents the likelihood that data element I comes from cluster J. And so, the way we're going to do that, since we're in the maximum likelihood setting, is

to use Bayes' rule, and say, well, that's going to be proportional to the probability that data element I was produced by cluster J. And then we have a normalization factor. Normally, we'd also have the prior in there. So why is the prior gone Charles? Well, because you said it was the maximum likelihood. Scenario. Yeah, right. We talked about how that just meant that it was uniform and that allowed us to just leave that component out. It's not going to have any impact on the normalization. Right. So that's what the Z step is. Is if we had the clusters, if we knew where the means were, then we could compute how likely it is that the data would come from the means, and that's just this calculation here. So that's computing the expectation. Defining the Z varaibles from the muse. The centers. We're going to pass that information. That clustering information, Z, over to the maximization step. What the maximization is going to say is, okay, well if that's the clustering, we can compute the means from those clusters. All we have to do is just take the average variable value. Right? So the average of the Xi's. Within each cluster J. What's the likelihood it came from cluster J and then again, we have to normalize. If you think of this as being a 0 1 indicator variable, then really it is just the average of the things we assign to that cluster. But here, we actually are kind of soft assigning, so we could have half of one of the data points in there, and it only counts half towards the average, and we could have a tenth in another place, and a whole value in another place, and so we're just doing this weighted average of the data points. So, can I ask you a question, Michael? Yeah, shoot. So, this makes sense to me, and I, and I even get that for the Gaussian case, the z i variable will always be non 0 in the end, because there's always some probability. They come from some Gaussian because they have infinite extent. So I, this all makes sense to me. Is there a way to take exactly this algorithm and turn it into k means? I'm staring at it, and it feels like if all your probabilities were ones and zeroes, you would end up with exactly k means. I think. I dunno, I never really thought about that. Let's think about that for a moment. Certainly, the case, if all the z variables were 0, 1, then the maximization set would be the means, which is what k means does. Mm-hm. Then, what would happen? We send these means back, and what we do in k-means is we say, each data point belongs to it's closest center. Mm-hm. Which is very similar actually to what this does. Except that here we then make it proportional. So I guess it would exactly that if we made these clustering assignments, pushed them to 0 or 1 depending on which was the most likely cluster. Right, so if you made it so that the probability of you being to a cluster actually depends upon all the clusters, and you always got a 1 over 0. Basically you did, this was like a hidden argmax kind of a thing, or a hidden max or something. Then you would end up with exactly k-means. I think you're right. Huh. Yeah, I never thought about that. Okay. So it really does end up being an awful lot like the k-means algorithm, which is improving in the error metric, this squared error metric. This is actually going to be improving in a probabilistic metric, right. The, the data is going to be more and more likely over time. That makes sense.

2.2.23 EM Examples

So that's sort of EM at the level of the equations, but I thought it would be helpful to actually implement it and kind of mess around a little bit. So do you want to see what happens? I would love to see what happens. So I generated some data here, which comes actually from two different Gaussian clusters. One that's centered over hereish, and one that's centered over hereish. [LAUGH] Just not to be too specific about it, but you can sort of see that there's two clouds of points, right? One in the upper left and one in the lower right. Can you see it? The one in the lower right looks like a clown. I think it's a little bit more of a superhero but that's that's not the point. The point is, is that it's just a bunch of random points. Hm. That's the point. It's a random point, but it is a point. So, your point is that the points are randomly pointed. Yeah, I just want to point out that at random. I see your point. And, so, what I want to do now is run EM and the first thing I need to do is pick the initial centers. Right, so I need to pick the mu's that we think these were generated by. And so a common thing for people to do is just to take two of the points from the data set at random. So I took two points at random and I'll mark them with an x. And now what I'm going to do is run an iteration of EM. And what that means is, I'm going to first do expectation, which is going to assign every one of these points to one of these two clusters. And I'll color them as to which cluster they belong to. And then I will move the centers, re-estimate the means based on that soft assignment. Makes sense. Alright, now so what you can see is something kind of interesting happened. Because these centers started off in the same cluster together, many of them were assigned to one cluster, which was the one that was a little bit more above. And then very few of them ended up getting assigned to the lower cluster because very few points were essentially closer to that one than the other one. When I colored the lower ones red and the upper ones blue. And there's this sort of band of 1s in the middle that had intermediate probabilities, they weren't really deeply one cluster or the other. They were near 0.5? Yeah, exactly, specifically between, including 0.4 to including 0.6. Okay that make sense. Alright, and so those are the green ones. And then

based on the cluster assignments, we estimated the means. So this xis now at the, you know, the mean of the red points. And the green points, because these they're actually shared and this acts as kind of the rest. And you can see, it doesn't really capture yet what the true clustering is because a huge number of the lower right cluster points are blue. They're kind of grouped in with the upper left cluster. But we can run another iteration of the app. And now things have shifted, right? So now it looks like this lower right cluster has claimed a good amount of the points in the lower right and the blue ones are the blue ones. There's just a few little green scattered between them at the boundary and you always expect some of that to happen. Now, the centers have moved and now they are starting to take on their clusters, but we just run this a couple more iterations until we see the x is not really moving any where. Alright, seems to have settled down and you can see it really did pull out the lower right cluster, the upper left cluster as a cluster and just a few points at the boundary where it said well I can sort of believe that that's part of the red cluster, I can sort of believe it's also part of the blue cluster, I really can't decide. And you know what? That's probably right too. I mean they could go either way. Yeah, exactly. I can sort of hallucinate this green point here as being part of the upper cluster. Just kind of on the fringe. But, you know, it works just as well as being part of the lower cluster. Actually Michael I'm kind of curious. Can you, use your magical computer powers to, you know, click on one of those green points and see what the probability is? I could. Yeah, so this, in particular, it's 55% in the first cluster and 45% in the other cluster. Hm. So, it really is. It's hovering near that boundary. That makes sense. What about that red point all the way in to the right? The ones that are all by itself. Excellent choice. So, it is actually. 0.9999996 in one cluster. And 1 minus that in the other cluster. This one is pretty certain coming from the second cluster. I see two things that came out of this. One is EM is much nicer in that it's not forced to make a decision about where those green points go. So that's sort of soft clustering does that. And that's a good thing. Because we don't have that problem that we had before. But one of the consequences of that, and this is not a bad thing, but it's a thing, is that even points that pretty clearly belong to one cluster or another, given that we're staring at them. They do have a really high probability, 0.9999999996, but they all have some non-zero probability. Of ending up, of belonging to the other cluster. And is that, you think that's a good thing or a bad thing? I think it's a good thing. I think it makes sense because Gaussians have infinite extent and even if a point is very, very far away from the center, it has some chance of having been generated from that Gaussian and so, this just tells us what that chance is, it is very, very unlikely. But it is still as non-zero probability match. In, in some sense, it's acknowledging truth. Right? Which is that you can't be sure which of the two clusters, this came form, even if I tell you where these clusters are. Right? Yeah, okay. I, I agree that. Okay, this is cool. So, EM is nice. So, does EM work for anything other than Gaussians? It does, it actually can be applied in lots of different settings. Let's flip over and talk a little about some properties of EM. Okay.

2.2.24 Properties of EM

So we talked about the equations that defined expectation maximization, and we stepped through an example with some actual data in the sense that it was data points. They weren't actual, measured data points. But what I'd like to talk about now for a moment is some of the properties of the EM algorithm more generally. So one of the things that's good about it is that each time the iteration of EM runs, the likelihood of the data is monotonically non-decreasing, right? So it's not getting worse. Generally it's finding higher and higher likelihoods and it's kind of moving in a good direction that way. Unfortunately, even though that's true, it is not the case that the algorithm has to converge. I mean have you ever seen it not converge? I've never seen it not converge. No, because usually there's some kind of step that you take and you just, you make the weight lower and lower. So yeah, or something like that. You know, no, I've never seen it not converge. So it doesn't have to converge. I think you can construct really strange examples that make it do that. But on the other hand, so even though it doesn't converge, it can't diverge, right? It can't be that these numbers blow up and become infinitely large because it really is working in the space of probabilities. And it's, it's pretty well behaved as far as that's concerned. That's a difference between in k means, right? So, the argument for k means, if I recall, which feels like was about a week ago, when we talked about this. [LAUGH] But of course, it was, it was merely seconds ago. Is that there is a finite number of configurations and k means and since you are never getting worse in our error metric. So long as you have some way of breaking ties, eventually you have to stop. And, so, that's how you got convergence, right? Yeah. So, in EM, I guess the configurations are probabilities and I guess there is an infinite number of those. Yet you can do more than guess. So in fact, there are an infinite number of those. Exactly. It wouldn't necessarily follow that you wouldn't converge from that. But that alone is, is one big difference, I guess, between the k means and the, the EM. That's the trade off you get for being able to put probabilities on things. So you've got an infinite number of configurations. You never do worse but you're always trying

to move closer and closer. So, I guess what could happen is you could keep moving closer every single time, but because there's a infinite number of configurations, the step by which you get better could keep getting smaller, and so you never actually approach the final best configuration. I suppose that's possible. But for all intents and purposes, it converges. Right. Exactly. However, it can get stuck. And this you see all the time. I almost never not see it do this. Which is to say that, if there's multiple ways of assigning things to clusters, it could find a way that doesn't have very good likelihood but can't really improve on very well. So there's a local optima problem that is pretty common, and so what do we do when we, get stuck in local optima with a randomized algorithm? Cry. No. We take all of our toys home and randomly restart. There we go. Okay. And the last thing to mention is this, is, is what you just suggested a moment ago in the previous slide. Which is that, it's nothing, there's nothing specific about Gaussians in here. It really is an algorithm that can be applied anytime that we can work with probability distribution. And so there's just a ton of different algorithms that work in different scenarios by defining different probability distributions, and then all you have to do is figure out what the E step and the M step are. How do you expectation to work out the probability of the latent variables. And then, how do you do maximization to use those latent variables to estimate parameters? And usually it's the case that it's the estimation that's expensive. It's difficult because it involves probabilistic inference. Right? So it's just like Bayes net stuff. Mm. And the maximization is often just, you know, counting things. But it isn't, in general, the case that it's always harder to do E than M. There's some well-known examples where M is hard and E is actually quite easy. You know, for any given problem you have some work to do to derive what the E and the M steps are. But it's very general, it's a really it's a good tool to have in your toolbox. I like that. So, basically it's not that hard because it's just a small matter of math programming. Indeed.

2.2.25 Clustering Properties

Alright. So that's all the algorithms that we're going to talk about in terms of unsupervised clustering algorithms. But I would like to kind of pop up a level, and talk a little bit about different properties that clustering algorithms might have. Desirable properties. So, the three that I'm going to talk about are richness. Scale-invariance and consistency, and these are good things to have right? I'd like to be rich and consistent. And Lord knows, you'd like to be scale-invariant. I would. I wish I were scale-invariant. I guess it would be very hard for me to gain weight if that were true. Mm-hm. No matter what the scale is, the number's always the same. Alright, so if you have a clustering algorithm, what does it do? It takes a set of distances d and maps them to a set of clusters. Or partitions. Right. Or a partition. Right. And these are three properties of those kinds of mappings. So richness is the idea that for any assignment of objects to the clusters, there's some distance matrix that would cause your clustering algorithm to produce that clustering. For any clustering that you want to achieve there is some distance matrix where p of d, your clustering algorithm, your clustering scheme, produces that clustering. So that's like saying, all inputs are valid and all outputs are valid. All inputs, which is to say that the distance matrices, sure, those are valid, and anything could be an output. Any way of clustering could be an output. because the, you know, think of the alternative. The alternative is there are certain clusters that you just can't produce. And that seems wrong, doesn't it? That, it ought to be the case that your clustering algorithm should produce whatever is appropriate, and shouldn't be limited in what it can express. Sometimes. You'd even want your algorithms to say, you know what, there's only one cluster here. Yeah. I mean, totally. If I showed you a picture, you might look at it. And you'd say I just see one cluster and it looks like a cloud. The second property that we're talking about, the scale invariance. And this is, I think, much more straight forward, at least in terms of conceptually. So, it just means that if I give you a distance matrix and I double all the distances or halve all the distances. It shouldn't change what the clustering is. That the clustering algorithm should be invariant to what the space of the point is, assuming that we keep the relative distances the same. So this is the NASA problem. So this says that, if I come up with a bunch of clusterings because I've been measuring points in miles, if I suddenly start measuring them in kilometers, it shouldn't change anything. That's right, yeah, change of units. Yeah, that's a really good way of thinking about it. It's not even that I'm scaling the distances, I'm just using inches instead of feet. It should be the case that it's still the same data. All right. And then the last one is maybe the hardest one to visualize. But, it's a really reasonable thing. And you would expect clustering to act this way. So, consistency says that if we have some clustering. If your clustering algorithm produces some clusters. So, let's, let's do a little example of that. Then, shrinking the intracluster distances and expanding the intercluster distances, does not change the clustering. Let me show you that. All right. And now I've edited this so that within the clusters the points have gotten closer together. More so in this one than, than the other. Or not changing them at all. But in this particular case, I shrunk this one a lot, I shrunk this one a little. And then I moved the clusters a little bit farther

apart. This changes the distances a bit and we like our clustering algorithm to continue to consider these clusters, right? It shouldn't introduce new clusters because we've shrunken things. And it shouldn't want to join these things together, because they've gotten further apart. So, that's this notion of consistency and a little bit more cumbersome to describe, but very natural thing to think about in the clustering setting. Well that makes sense right? So, this is where our domain knowledge comes in so, distances are a measure of similarity, right? Yup. So, what consistency says if you found that a bunch of things were similar and a bunch of other things were dissimilar. That if you made the things that were similar, more similar and the things that were not similar, less similar. It shouldn't change your notion which things are similar and which things are not. Yeah, which things are alike, which things want to be grouped together. Yeah, uh-huh. Hm. Good so you think you understand these three clustering ideas? I believe I do. Alright, so let's put them into play a little bit. Okay.

2.2.26 Clustering Properties Quiz Question

And we'll do that with a clustering properties quiz. Oh yeah. All right, so what I'm going to do, is I'm going to give you three different clustering algorithms. For each one, ask whether it has these properties. Does it have richness? Does it have scale-invariance? Does it have consistency? And so the algorithms are all going to be variations of the first clustering algorithm we talked about, single-link clustering. And so, what we're going to do is we're going to run single link clustering, but to make it a clustering algorithm we have to decide under what conditions are we going to stop building our clusters? And I've got three different conditions, then that defines our three different algorithms. So one is, we've got n items that we're clustering. I'm going to stop when I've got n over 2 clusters. 'Kay? So just keep merging, keep building clusters until you've reached n over 2 clusters, and at that point, stop and return what you've got. Okay. Does that, does that make sense? Yes. All right, and you remember enough about single-link clustering for that to be meaningful, but that's, it's where were going to start off with everything in its own cluster, and then merge them together by whatever two clusters are closest together, and then iterate. Yes. Okay. All right, so that's algorithm one. We're going to stop at n over 2 clusters. The second one is we're going to have some parameter theta, and we're going to keep merging clusters until we'd have to merge clusters that are theta units apart. And once they're theta units apart, we're going to say, nope, that's too far to be part of the same cluster. We're done. Okay. Okay? So that, again, it's a clustering algorithm, right? It's going to take these distances, and it's going to turn it into groups. So that's like, only things that are within ten feet of each other could possibly be clusters. Exactly. Okay. Right. Theta is going to define that. And the last one is very, very similar. We're going to keep doing clusters until we'd have to merge clusters that are farther than theta over omega units apart. And omega in this case is going to be defined to be the largest pair-wise distance over the entire data set. That's an omega? Yes. Okay. And that's a capital D, at least now it is. [LAUGH] Okay. All right, good? So if you understand these algorithms, what, what I'd like you to do is say which of these have the richness property, which of them have scaling variants, which of them have consistency.

2.2.27 Clustering Properties Quiz Solution

All right Charles let's see what you think. The first one says we want to have a fixed number of clusters. Well actually the first thing I'll note about that, since we're going to have a fixed number of clusters in doesn't have the richness property. Good point. Because richness would allow for one cluster or it could have n clusters or it could have n over three clusters or it could have n over two clusters. But here you forced it to always have n over two clusters so it can't represent all possible clusters. Agreed. However, you'll notice that there's nothing in here that cares about distances. In the sense that, if I took all of the coints, and I multiplied their distances by two, I would still get the same clusters in exactly the same order. That's the important thing. That it cares about distancees but it only cares about the order not the scale. Exactly. So that means there's scale-invariance. Very good. And then by the same argument, this algorithm has consistency because the points that were, clustered together if they got closer together, they would still be picked up. And the ones that were farther apart, well, they'd still get picked up by each other and it, it doesn't matter. So they're definitely consistent. That's right. And it's a nice little property of single-link clustering. Let's move on to the second clustering algorithm. Nice. Okay so clusters that are theta units apart, well, since theta can change, even if theta's fixed the, the points that I get. Could be various kinds of distances. So, let's imagine that theta was ten feet. If all the points are within ten feet of one another, then they would be one cluster. Yup. If on the other hand, all the points were more than ten feet apart, you would have N different clusters. And, you could do any of them in between. So, this is rich. It is indeed. That's

right. We can always muck with the units. Or muck with theta for that matter. So, that we can group the beginning of our clusters in any. Accommodation that we want. Yeah, but for exactly the same argument that they're rich, they're not scale invariant. Yeah. Because I could just take everything and multiply it by theta, multiply it by the distance by theta and now I will suddenly have n then if I had n to begin with, I could divide by theta, and then I would have one. So it's not scale invariant. Agreed. But the consistency argument still works, because all the points that got clustered together. Because they're within theta of each other. If I made them closer, would still be within theta of each other. And the ones that weren't closer together because they were more than theta apart, would now be even more theta apart and so you do get consistency. Agreed. Excellent. Ok. Alright, last example. Clusters that are Theta W units apart where. I'm sorry, Theta Omega units apart. [LAUGH] Theta divided by Omega. Yeah. Where Omega equals the maximum distance. Well, that's just a way of normalizing distance. In much the same way that I argued for richness of the second algorithm, the same argument applies here. That it is rich. It is rich, because I can just keep shrinking and moving the points around, and it will work out just fine. And so, it's definitely rich. We don't control omega, because that's determined by the distances, but we can change the theta so that things are too close or too far. Yeah, okay, I agree with that. That's sort of the last thing you said for the second algorithm too. So you do get richness. Now, what's interesting to me here is that unlike in the second algorithm where you didn't have scale and variance, you do get scale and variance here because if I try to make things bigger. I'm also going to make the maximum distance bigger by exactly the same amount and so I will always end up back where I was before. Yeah. So whatever you do to scale this, it's going to get undone. Yeah. Very good. Right. Exactly. Anything I do to make it, make them far apart will just make them the same distance. But at least by omega. Agreed. Okay. And, and if we have consistency too, we've got a trifecta. We do, except, we don't have consistency. What? Because if I make the, the clusters that are farther apart, farther apart, then I also change omega. And that could actually change the cluster. It would because, in fact, imagine that I made the points in a set of clusters. Much closer together, but then I move that cluster, you know, sort of infinitely far apart from the rest of the clusters. Then, suddenly my theta divided by infinity, or near infinity, would make the, the radius of allowable clusters so small that no points would be able to cluster with any other point. And so that would screw up whatever you had before, assuming you had clusters before at all. And so, I can construct a world where consistency would be wrong. Oh that's nice, it made a little diagonal of X's. [LAUGH] And I win, three in a row. Well done Michael. So, what little tweak do we have to do to these algorithms to get a trifecta? Yeah, that would be great, wouldn't it? And that's the final algorithm that we'll talk about. Great.

2.2.28 Impossibility Theorem

Charles, I lied. No! I'm so sorry. In fact, it turns out that there is no tweak that you could do to these algorithms to make the trifecta, to have all 3of these properties. And, in fact, there is no clustering algorithm. It is impossible to derive a clustering algorithm that has these 3 properties. So this is proven by Kleinberg, and what he showed is that, once you define these 3 different properties, richness, scale invariance, and consistency, they are mutually contradictory in a sense. So, just like we saw in the example that we went through in the quiz, where we tweak the algorithm and it gets us one but it loses another, that's a necessary property. You just can't have all these 3 hold at once. That's pretty surprising coming from Kleinberg, because John is one of the nicest people I know in machine learning and theory, and you would think that he would have tried to find a possibility theorem, not an impossibility theorem. I'm very disappointed. See, he's got a dark side, is what I'm saying. This is a striking and maybe even upsetting result, right? It's saying that, if you actually sit down and say, well, here's what I would like my clustering algorithm to be, which people hadn't really done very often, once you've bothered to go do all that hard work of saying what matters to you and what doesn't matter to you, it turns out you can't have what you want. You can't always have what you want. But, can you get what you need? In this particular case, maybe. It depends if you only need 2 of these. [LAUGH] Well, so how bad is this? So, so, so, I agree, it is, it is a at least to me, a surprising result that you can't have all 3 of them. And it's a little disappointing because you'd love to have an algorithm that does. Because, I think we agree, all 3 of these properties makes sense. At least, certainly, independently they do. Right. But just how bad is it? I mean when, when you can't have all 3, can you come close to having all 3? Can you have like 2.9 of them? [LAUGH] Well, you can definitely have two of them, because that's what we did in the quiz. hm. Well, it kind of depends on what, you can reinterpret some of these properties and get, and nearly satisfy them. And I can point you to Kleinberg's paper to look into that. But I think I need to be done with this for now. I just wantedto give you a flavor of the idea that, one of the reasons that clustering is hard to pin down is because when you pin it down, it plays dead. Hm. It's like a, it's like an opossum. When you pin it down, it actually it turns out that it still doesn't

really want to do what you want it to do. But in practice, what happens is people do clustering for lots of different reasons. And they're willing to change their clustering, like, look at what actually comes out of it, and change their notion of clustering so that it's more consistent with what they're trying to achieve. It's not great to use in this purely automated fashion, where you just hand it over to the computer and be done with it. But it's still a really powerful thing to do to get to know your data better. Okay. I accept that. I feel better now.

2.2.29 What Have We Learned

Okay Michael, what have we learned. Well, we learned the definition for feature selection, which was getting a subset of the features to feed to some, I think we also talked about supervised training algorithms, or learning algorithms after the selection has taken place. Right. We made a distinction between [NOISE] rapping and filtering. Was that supposed to be filtering? Yeah, I didn't know how the filtering sounds. It sounded more like slurping. Yeah, I'm not sure that my raping actually sounded like rapping either. It didn't, but I wasn't going to say anything. So what else did we learn? Besides, you like sound effects. So we learned about feature selection, we defined that. We discussed the difference between filtering methods for feature selection and wrapping methods for feature selection. What did we learn about them? That wrapping is slow, but actually seems like it solves the problem that matters. Yeah. So let's call that slow but useful. Is that useful in the sense that you defined it? Yes. Mm Hm. Boy these can be useful words. Filtering is simpler, possibly faster, but maybe misses the point. Probably faster, yeah, but ignores bias. Okay, so what else? Is that it? Well so we, and we specifically learned about the distinction between features being useful versus them being relevant. That's right. so, relevant things are things that give you information about the classification problem, or the regression problem that you care about. But useful, features are things that help you to actually do the learning, given some specific algorithm. Right. And you reminded what a Bayes optimal classifier was. [LAUGH]. Which I guess I should have known already. But somehow, I did not understand how it fit into this context. The, the main power Bayes optimal classifiers is that it is sort of the gold standard, it is the ultimate that you could do if you had everything and all the time in the world. Could I say that relevance is usefulness with regard to the Bayes optimal classifier? Yes, actually you could, I like that. It is a special case [UNKNOWN]. Oh wait, there's something else that you talk, that you talked about. Yes. Which was, strong and weak relevance. Right. And in a sense that relevant features have a kind of kryptonite, in the sense that you can make them not strong just by putting a copy of them into this space. Right. I like that, the kryptonite is copy. Well done. That's because you're no longer indispensable. If I have a copy you. That's right, you and now your evil twin now resides in your parallel universe. So what do you think that means for us? Are we, strongly relevant, weakly relevant or useless? [Laughs] And those are your choices. [Laughs]. I am, I'm going to say I am weakly relevant because I think I correlate with truth, assuming the subset of other people in the world is the empty set. Hm. . Fair enough, fair enough. So then by that definition, do I get to be weakly relevant? You are atleast weakly relevant. Very good. Very good. But are we useful. We're going to say yes. As far as our students know. [LAUGH] Well I guess that's the way we will find out ultimately is how well they do on the course. It's true. It's in some sense completely in their hands. Right. So in fact, this course is a wrapper method over features and this is the first iteration. Interesting. Wow, that was deep. I feel like we should end on that. I do too. Okay, well then I will see you next time when we will talk about feature transformation. Transformation. Well done. Bye. Bye.

2.3 Feature Transformation

2.3.1 Introduction

Hi again Michael. Hey, how's it going? It's going pretty well. We are ready to do our last lesson of this mini course on unsupervised learning and randomized optimization. Cool, what's it about? It is about feature transformation. As opposed to feature selection. Cool. So they can change from vehicles into robots, I assume. yeah, typically. Usually it's from robots into cars, but that's a technical detail. Okay. Okay. So I'm going to define feature transformation for you, or at least I'm going to do my best to. It turns out that it's a slightly more slippery definition than the one that we used for feature selection but it has a lot of things in common. Okay? Yep. And you tell me if this makes sense. Okay, so feature transformation as opposed to feature selection which we talked about last time is the problem of doing some kind of pre-processing on a set of features. In order to create a new set of features. Now typically, we expect that new set of the future to be smaller or in some way more compact. Okay? But while we create this new set of features, we want to explicitly retain as much information as possible, and when I say retain as much information

as possible, I probably mean, I think we'll, we'll see as we continue this conversation. Information that is relevant and hopefully useful. Now, the first thing you might ask is, what's the difference between this and feature selection. Is that a question you might ask, Michael? I was thinking about that, though I think you kind of told me at the beginning of the feature selection lecture. well, I told you that I was going to to tell you. I'm not sure I actually told you. Hm. So one thing you might say Michael, is that if I looked at this definition, this seems to actually be consistent with feature selection as well. I'd take a set of features, a feature selection. Then I'd create a new feature set which happens to be smaller. And my goal was to retain as much information as possible and we talked about the difference between relevant features and useful features, but really this sort of describes feature selection as well. You see that? Yeah, definitely. Now the difference is, in fact probably the right way to think about is this that feature selection is in fact a subset of feature transformation. Where the pre-processing you're doing is literally taking a subset of the features. Here, feature transformation can be more powerful, and can be an arbitrary pre-processing, not just something that goes from a set to to a subset. But what we're going to do is we're going to restrict our notion of future transformation to what's called linear future transformation. So, let me define that for you explicitly. So as before with the feature subset, we said that we were taking a bunch of features or a bunch of instances that were in some individual feature space and transforming it into another feature space of size M. Yup. And in the, the feature selection problem, m was meant to be less than N, hopefully much less than N. And that's typically the case of feature transformation, though as we'll discuss towards the very end, it doesn't have to be. But when I say usually we expect M to be less than N, usually means almost always. [LAUGH] Okay. And that's because of the cause of Dimensionality problem. But the difference between what we were doing with future selection and future transformation, because this is exactly what I wrote before, is that this transformation operator is usually something like this, in linear transformation operator. So the goal here is to find some matrix P, such that I can project. My examples my points in my instance space, into this new subspace, typically smaller than the original subspace. And I'll get some set of new features which are combinations in a particular linear combinations of the old features. Does that make sense? I think so. So then that p matrix would want to be N by M. Yes. So the transpose would be M by N and that multiplies by the N dimensional feature space in X and projects it out to an M dimensional feature space. Right. Okay. Yeah. Pretty good. So if we wanted to write that out. We could say that feature selection was about taking features like X1, X2, X3 and X4 and finding a subset like X1 and X2. And that would be feature selection. But feature transformation would be taking something like taking X1. X2, X3, and X4, and translating into something like 2X1 plus X2. Which creates a single new feature. Which is a linear combination of the subset of the. Does that make sense? Yeah, but it could actually make other feature combinations as well, right. It's just projected down to one. Right, it could be projected down to two, or it could be projected down to three Or could even project it out to a different for. Okay. And in principal you could imagine that you could even project up into other dimensions which is something that we've done before. Conceptually, anyway. When we talked about kernels? Yeah, although those were typically non-linear transformations implicit in the notion was doing a non-linear feature transformation but we even did it before we even learned about kernels. Very second thing I think we did. Perceptrons. How do perceptrons go into a higher dimensional space. Well, when we talked about XOR. Oh, right. What we effectively did was we showed that we could project the original. Two dimensional space into what looks like a three dimensional space where the third dimension was a combination of the first two. And then you could actually do it with a linear separator. But that wasn't a linear transformation, that c was a nonlinear transformation. Right. Because we were talking about Boolean verbs. Yeah. But, in the end of the day it was still a kind of feature transformation and in this case a feature transformation where we went to a higher number of features. And today, what we're going to be talking about is linear transformations as opposed to non-linear transformations. And we're going to be focusing specifically on the case where we're trying to reduce the number of dimensions. So the implicit assumption here, right, the kind of domain knowledge that we're bringing to here or the belief that we're bringing here is that. We don't need all N features. We need a much smaller set of features that'll still capture all the original information, and therefore, help us to overcome the curse of dimensionality. It's just that that smaller subset might require bringing in information across the various features. Okay? Yeah. Alright

2.3.2 What Are Our Features

Okay Michael, so what are our features? When we have a bunch of documents let's say full of words? I'm not sure, it could be things like the number of words in the document? That could be, but what sort of, and that's actually perfectly reasonable thing but what's the simpliest set of features that you might start with from which you might then compute more interesting features? Well there's a super, duper big set of

features which are the words. Right, so in fact that is exactly what we have, is we have words. Maybe we have punctuation and, you know. Words are sort of the obvious thing to do. I typed in machine learning, why don't I just return every single document that has machine followed by learning. Maybe you should. Maybe I should and maybe that's a perfectly reasonable thing to do. In the case of machine learning, should I return documents that contain the word Machine but don't contain the word learning? I wouldn't score them very highly. I might not score them as highly, but I might still return them as being at least more relevant than documents that contain neither Machine nor Learning. Right, that are just about turtles, say. Right exactly, although it's turtles all the way down Michael. I see. OK. So if our features are words, which are the sort of the most obvious things to get at, we can compute other things like the number of words or whatever. But basically our features are going to be words or counts of words or something like that. As it's sort of a. Reasonable first step. And, in fact, the very early retrieval systems, which, you know, predate both of us, actually Michael, used simple things like words. Now, there's a lot of way, details to this you could imagine. Like maybe you'd want to transform all plurals into their singular version, get rid of words like the. And there's a bunch of complicated stuff you might want to do. But, it's not particularly relevant for this discussion. So, just assume whatever you want to assume about the kind of words that you have. Okay? Okay. Okay. So, what's the problem with using words? Can you think of any problems with using words? Well, there's a lot of them. Right. That's actually the first thing. There's a lot of words. Which means there are a lot of features. Which means the curse of dimensionality is going to hurt us. I would think that they'd be pretty good indicators of meaning, except I guess there's kind of two complimentary problems. One is that some words mean more than one thing. Mm. I like that. So we have good indicators. Words are good indicators of meaning because, you know. The words, but, you could be in the case where you have words mean multiple things. You said there were two problems, what's the second one. Sort of the opposite, which is, you can say the same thing using completely different words. yes. So that's that many words mean the same thing. In particular, words, the fact that words can have multiple meanings. It's called polysemy. The fact that many words can mean the same thing is called synonymy. So, can you think of a word that might have multiple meanings? That indicates the problem of polysemy? Well, so, you know, learning, in the machine learning example, learning. Sometimes refers to, this statistical process that we've been talking about. But it also refers to the thing that, people do. And in some, in some scenarios, it actually means to teach. Like, I'm going to learn you something. That's true, but that's just too on point. I'm going to pick something else. That I think you will appreciate at the end. Let's think of a word like car. So car has multiple meanings, Michael. Hmm. Can you think of them? I'm mostly stuck on one at the moment. Unless you're thinking of. Which one is that? I'm thinking of the vehicle that you drive around. Yes. But I guess one could also be referring to a list deconstruct or in LISP. Yes, exactly. It's the first in what are those things called? CON, CON cells? CON. Thank you. Right, so car is either an automobile or it's the first element in a cons cell, which all of you people who've heard of lisp knows is an awesome reference. [LAUGH] For those of you who've never used lisp, which is clearly the greatest language every written in all of history, or that ever will be written. Because it is a superset and subsumes all other languages, including natural languages. Imagine this word, instead, is apple. And so apple can refer to a fruit, or it can refer to a computer company, or to a music company for that matter. Hm. But, I prefer car. So, car can mean automobile, or it can mean the first element in a cons cell. If you're using Lisp. Now sticking to this car example synonymy is a similar problem in that car and automobile often refer to the same thing. So you see this. So polysemy is a problem multiple things and synonymy is a problem meaning the same thing. And a paticular word like car in this case. Can cause both polysemy problems and synonymy problems. Yeah, I can see that. So in the example you gave before about machine learning, you would, if you'd just returned documents that had machine and learning right after each other, then you'd miss all the stuff on, for example, data mining. Right, in fact, that's a very good example so, there's a huge split in the community between those who care about data mining and those who care about machine learning. And often the people in one camp don't believe the people in the other camp are doing what they're doing. But for the person who isn't religiously commited to one of those camps or the others, when you talk about machine learning you probably also care about data mining. When you talk about data mining you probably also care about what people inside the field might call it instead of machine learning. And so you would be missing out on a whole swath of papers or interetinst discussions if you happend to put in the word machine learning. But similarly if you put in the word like data mining you might end up getting all kinds of documents that are about the data or about the data than you get when you literally mine for ocal. Who knows? And it would cause you huge problems for getting the exactly relelvant set of documents that you want. So we can actually talk about this in terms of, the kind of errors that you get in say supervised learning, what kind of errors do polysemy give you, Michael? False, well there's false positive and false negative and this one of the things where we, its going to return things that aren't relevant, its going to say

they're positive when they're not. So you've given me the answer, false positive. Okay. Looks like false positive. By contrast synonymy gives you what? True positives. No, No wait, I negated the wrong word, false negatives. Exactly. In other words it's going to, wait is that right? Does the, so false negative means, oh it's going to tell you something's not there, when actually it really is. Right, so Typing in car will not just give me the automobile articles about my Tesla, which is a black P85. With black leather, [LAUGH] fully loaded and is like the greatest car on the planet, but it will also give me really awesome, but in this case not what I'm looking for, documents about LISP. Meanwhile, when I type in car, I will actually get all these things on automobiles. But I won't get articles that talk about automobiles without actually using the word car. So you can imagine that these sorts of problems come up all the time. You've got a set of features, in this case words, which have this problem that although they're good indicators. They are insufficient, that is we have a set of features that will generate false positives and false negatives on their own and more to the point, doing feature selection will not solve this problem. I can throw away a bunch of irelivent words and even useless words, but I am still going this problem of generating false positives. For our polysemy and false negatives, or synonomy. And this goes beyond simply, information retrieval and text retrieval into any generic problem where you have possibly a large set of features that have this problem with false negatives and false positives.

2.3.3 Words Like Tesla

So, what we're going to get out of feature transformation is the ability to combine these features, these words, somehow, into some new space where hopefully we will do a better job of eliminating our false positives and false negatives by combining words together. So, we're going to leave this example for now, but just to give you an intuition about how a proper kind of feature transformation might help you just let me point out that if I type in a word like car, you would expect, since I know what car kind of means. Let's say automobiles in this case, I should pick up documents or score documents higher if they don't contain the word car, but they do contain the word automobile. Or perhaps they contain a word like motor or race track. Or Tesla, right, that somehow words like Tesla, and automobile, and motorway, and anything else you can think of that has to do with cars probably are highly correlated or highly related to cars in some way. And so, it would make sense to combine words like automobile and car together into new features to pick up documents that are somehow related together that way. Does that make sense? So that's the intuition I want you to think of for the three algorithms that we're going to go over next. Okay? Yeah, I could definitely see how synonymy is going to be a win, when we index the documents, if we include, well, if we map them down to a lower dimensional feature space where one feature's used for anything sort of car related. I'm not sure how that's going to help with polysemy, but I definitely see how this feature transformation idea could be a win with synonymy. The way, so that's right, and I think it's, it's much easier to see how it works with synonomy then with polysemy. The way it will, it could in principle help you with polysemy is that it will combine a bunch of features that together eliminate the ambiguity of any particular word. So as you type in more words together it'll start to pick those thing up while also eliminating synonomy. So we'll see. The way it's going to work in practice actually is that if you think of it. Now we are talking about unsupervised learning. But if you think of this as a supervised learning problem, then you can imagine how you can ask yourself how to combine sets of these features, these words together such that they can still give you correct labels. And if you can solve that problem you will end up solving both polysemy and synonomy. Or at least minimizing their impact. Cool. OK. So let's go over this specific example to far more abstract examples and talk about three specific algorithms. .

2.3.4 Principal Components Analysis

Okay, so the first linear transformation algorithm we're going to discuss, is something called Principal Components Analysis. Okay Michael? Sure. Now just to be clear here, the amount of time it would take me to derive Principal Components Analysis and work it all out in its gory detail would take forever and so I'm not going to do that, I'm going to leave that to reading. But I want people to understand what Principal Components Analysis actually does and what its properties are, okay? Yeah. Okay, so, Principal Components Analysis has a long history. It's a particular example of something called an eigenproblem. Mm. Okay. It's a particular example something called an eigenproblem which you either already know what that means or you don't. And if you don't then it means you haven't read the material that we've given you so I'm going to ask you to do that. But, whether you have or have not let me just kind of give you an idea of what Principal Components Analysis actually does. And I think the easiest way to do that is with a very simple two dimensional example. So here's my very simple two dimensional example. All

right, so you see this picture Michael? Yep. So this is a bunch of dots sampled from some distribution that happens to lie on a two dimensional plane like this, okay? Yep. Now, what, so this is in fact two dimension. So we have two features here, we'll just call them x and y. We could have called them one and two, it doesn't really matter, this is just the x, y plane. And let me tell you what Principal Components Analysis actually does. What Principal Components Analysis does, is it finds directions of maximal variance. Okay, does that make sense? Variance of what? The variance of the data. So if I had to pick a single direction here, such that if I projected it onto that dimension, onto that direction, onto that vector. And then, I computed the variance. Like, literally the variance of the points that were projected on there. Which direction will be maximal? I would think it would be the one, that is sort of diagonal. It kind of, blobs along that particular direction. Right, that's exactly right. And, and to see that, imagine that we projected all of these points onto just the x dimension. That's the same thing as just taking that particular feature. Well, if we projected all of them down, we would end up with all of our points living in this space. And when we compute the variance, the variance is going to be something that captures the distribution between here and here. Does that make sense? Yep. Similarly, if we projected it onto the y axis. Which is the equivalent of. Those are examples of feature selection. Yes, of fea, it's exactly, it's a, it's a feature selection. It's equivalent of just looking at the second feature here, y. I'm going to end up comput having a variance that spans this space between here and here. By contrast, if we do what you want to do, Michael and we pick a direction that is about 45 degrees if I drew this right. We would end up projecting points between here and here. Now, it's not as easy to see in this particular case but the variance of points as they get projected onto this line will have a much higher variance than on either of these two dimensions. And in particular it turns out that for data like this which I've drawn as an oval that you know sort of has an axis at it's 45 degrees, this direction or axis is in fact the one that maximizes variance. So, Principal Components Analysis, if it had to find a single dimension would pick this dimension. Because it is the one that maximizes variance. Okay, you got it? Sure. Okay. Now. What's the second thing what's the second component that PCA or Principal Components Analysis would find. Do you know? I don't know what you mean by second. This is now a direction. That ha, that has high variance. Yes. No. The first one. Yes. because it seems like you know either the x or the y is pretty high or something that looks just like that red line but it's just a little bit tilted from it would also be very, very high. Right, that's exactly right so, in fact what Principal Components Analysis does is it has one other constraint that I haven't told you about. And that constraint is, it finds directions that are mutually orthogonal. So in fact, since there are only two dimensions here, the very next thing that Principal Components Analysis would do, is it would find a direction that is orthogonal or we think about it in two dimensions, as perpendicular to the first component that it found. I see. So there really only, really only is one choice at that point. That's right. Or. You know there is two choices because you, doesn't matter which direction you pick in principal.

2.3.5 Principal Components Analysis Two

So this is called the first component, or the principle component, and this is called the second, or second principle component of this space. Okay, does that make sense? Yep. Now, here's what's interesting about principle components analysis. You might ask me exactly how you do this. There are several, several mechanisms for doing it. For those of you who have dealt with linear algebra before, something like this singular value decomposition might be familiar to you. It's one way of finding the principal component. But principal components analysis basically has a lot of really neat properties, so let me just describe some of those properties to you. The first property is, well, the two that I've written here. It finds directions that maximize variants and it finds directions that are mutually orthogonal. Mutually orthogonal means it's a global algorithm. And by global here I mean that all the directions, all the new features that they find have a big global constraint, namely that they must be mutually orthogonal. It also turns out that you can prove. Which I will try to give you a little bit of evidence for. But I'm going to prove formally, that the PCA actually gives you the best reconstruction. Now what do I mean by best reconstruction? What I mean is, if you think of each of these directions that it found, in this case. You found this one first and found this one second. The first thing you, I, I hope you see is that if I return these two dimensions, I have actually lost no information. That is, this is just a linear rotation of the original dimensions. So if I were to give you back these two different features, you could reconstruct all of your original data. You see that? Wait. When you say features you mean, what we're going to give it is, for each of those little black data points, we're going to say how far along the red axis is it and how far along the orange axis is it. Right. So it really is just a, a, a kind of a relabeling of the. of the dimensions. Right, so just like when I think about these points in x and y space, the original feature space, whenever I give you value here for this feature I'm just describing how far along a black dot is on this dimension or this projection. Now the second dimension or the second feature

just tells me how far along a dot is. On this particular dimension or axis, and similarly about projecting on the read and on the orange, I'm telling how far along a point is along this axis and along that axis. So if I were to return, if I were to take X and Y and transform them into this new one and two, I would have given you different values than I did from X or Y, but they're actually just the same point. And so I've thrown away no information. So that's a pretty good reconstruction. That's a pretty good reconstruction. But what principle components analysis does for you. Is if I take just on of these dimensions, in particular, the first one, the principle component, I am guaranteed that if I project only in to this space and then try to reproject into the original space. I will minimize what's called L2 error. So do you understand that? I'm not sure let me check. So, you're saying if we instead of, all right we take the little black dots, they now have a red dimension and orange dimension one two, and now if we reconstruct using only the first dimension, I guess it just puts the black dots on the red line. Yep. And so there is no, they have no existance in that second dimension. Correct. And now you're saying of all the different ways that I could do that to kind of project it to a, to a linear sequence. This is the one that's going to have the smallest. L2 error which if im not mistaken is the same kind of reconstruction error we talked about in all the other times we talked about reconstrucion. So it's like squared error. Thats exactly right, its squared error. This particular notion is called a [UNKNOWN] norm but thats really just talking about distance. What this means is that if I project onto this single axis here, and then I compare it to where it was in the original space, the distance, the sum of all the distances between those points will actually be the minimal that I could get for any other projection. Cool. And you can, you can sort of prove this. It kind of makes sense if you just think about the fact that points. Always start out in some orthogonal space. And, I'm basically finding in scaling and a rotation such that I don't lose any information. And I maximize variance along the way. By maximizing variance, it turns out I'm maximizing or maintaining distances as best I can in any given dimension. And, so, that gives me the best reconstruction that I can imagine. Now, you might ask yourself, Is there anything else nice about principal components analysis given this reconstruction error? And there's another property of PCA that is very, very useful. And it boils down to the fact that it's an eigenproblem. What happens when you do principal components analysis is you get all of these axes back, and in fact, if you start out with. N dimensions, you get back N dimensions again, and the job here for a future transformation as you might recall, is you want to pick a subset M of them hopefully much smaller than N. Well, it turns out that associated with each one of these new dimensions that we get. Is its eigenvalue. That eigenvalue is guaranteed to be non-negative, it has a lot of other neat properties. But what matters to us here is that the eigenvalues monotonically non-increase, that is, they tend to get smaller as you move from the principal to the second principal, to the third, to the fourth, to the fifth, to the sixth, and so on to the nth dimension. And so, you can throw away the ones with the least eigenvalue. And that's a way of saying that you're throw awaying these projections, or the directions, or the features, with the least amount of variance.

2.3.6 Principal Components Analysis Three

okay, wait. So let me see if I can echo some of that back. So, it's almost as if what we're doing here is we're doing a transformation into a new space where feature selection can work. Exactly, and in fact, here's something kind of interesting for you. It turns out that if the ion value of some particular dimension is equal to zero, then it means it provides no information whatsoever in the original space. So, if I have a direction, if I have a dimension that has an eigen of zero, I can throw it away and it will not affect my reconstructioner. I'm trying to remember if that makes it irrelevant or useless. Well, it certainly makes it irrelevant. If there's no variance, that's the same thing as saying it has zero entropy, because it never changes. So it's irrelevant. Now whether it's useful or not, well, probably not, but it might be useful for something like, our simple, [INAUDIBLE] example that we used last time. Gotcha. Okay. So does this all make sense? So one question I have is in the example that we just kind of worked through there was a blob of data and when we drew the red line through the maximum variance direction it went through the. The xy origin. Um-huh. Does it have to? Is it necessarily the case that it's going to? Or you know is the algorithm restricted to have to put things through the origin? Well so my answer to you is that, that's actually a very complicated question. And in principle it, so to speak it, it doesn't really [LAUGH] matter. But in practice what people do and their doing something like PCA is they actually subtract the mean of the data. Or the central of the data from all the data points. So it ends up being centered around an origin, for the origin. But what this means is that you can then interprit what we're doing is finding maximum variants. As capturing correlation. Okay. That's helpful. And otherwise, if you don't do that, what you end up with is effectively a principle component that at least intuitively kind of captures the notion of where the origin should be, and it's not terribly helpful for what it is we're trying to do. So my answer is, no it doesn't, and yes it does. [LAUGH] Okay? Sure. Okay. So, that's, that's basically PCA. It's got all these neat properties. Let's sort of summarize

them again. It's, well, actually. Let me add a couple beyond these. It is a global algorhythm. It does give you best reconstruction error. Which seems like a very nice thing to, thing to have. And, it tells you, which of the. New features that you get out are actually important with respect to this notion of reconstruction by simply looking at their corresponding eigenvalues. They also have a couple of other practical properties. In particular, it's very well studied. And what that means in this case is that there are very fast algorithms that do a very good job, even in weird cases. Even with large data sets, at least if they're appropriately sparse, of being able to compute these things. People have been working on this problem again, since before we were born Michael, and they've gotten really good at finding principle components even for what would be very difficult spaces. But this does lead me to a question though which is another question, which is okay, so you've got an algorithm that probably gives you the best reconstruction. But what does it have to do with classification? This is kind of like the question we had before about relevance versus usefulness. So we find a bunch of projections which are relevant in the sense that they allow you to do reconstruction. But it's not clear that if you threw away some of these projections, the ones with low eigenvalue, that even though you'd be able to reconstruct your. Original data is not clear that would help you do classification later, can you see how that would work out? Sure, I mean like, it just could be that that's not where the information is about what the labels ought to be. Right, so imagine for example that one of your original dimensions is in fact directly related to the label and all the rest are just, say goush and noise. But the, the variance of that particular direction is extremely small. It might end up throwing it away. It'll almost certainly end up throwing it away. Which means you'll end up with a bunch of random data that doesn't actually later help you do classification. And that's because this looks a lot like, if I can do the analogy, a filter method. I was going to say that, yeah. Oh. So this is kind of like filtering. And in fact it's going to turn out that the other two items we're looking at too are like filtering although their particular criterion might be more relevant. We'll see. Okay? Alright, so do you understand principal components analysis? Again, I'm not asking you to understand exactly how you would run a principal components analsyis algorithm. That stuff's in, in all the text, but just to sort of understand exactly what is is trying to do, what it's trying to accomplish. Yeah, I think so. So, it's, it's taking the data, it's finding a different set of axis, that are just like regular axis in that they're mutually orthogonal. But it lines up the varience of the data with those axis, so that we can drop the least significant ones, and that gives us a way to do. Feature selection. But the whole thing is a feature transformation algorithm, in the sense that it first moved the data around, to be able to do that. That's exactly right. So, you do, transformation into a new space, where you know how to do filtering. Got it. Excellent.

2.3.7 Independent Components Analysis

Okay Michael, so the second algorithm that we're going to look at it just like principle components analysis, except it's called independent components analysis. Okay, and the major difference is really the difference between the first word principle and independent. So the main idea here is that PCA is about finding correlation. And the way it does that is by maximizing variance. And what that gives you, is the ability to do reconstruction. What independent components analysis is doing, or often called ICA by those in the know, is it's trying to maximize independence. Very simply put, it tries to find a linear transformation of your feature space, into a new feature space, such that each of the individual new features are mutually independent and I mean that in a statistical sense. So, you are converting your XI, your X1, X2, XI... And there's some new features space let's call it I don't know let's call it a Y, Y1,Y2 YI... Such that each one of the new features are statistically independent of one another, that is to say their mutual information is equal to zero. Does that make sense? And this is going to be a linear transformation? It's going to be a linear transformation. T, to make them statistically independent. So, I find some linear transformation here, which is going to take my original feature space, which I'm representing with these X's, and transform it into a new feature space, such that, if I were to treat each of these new features as random variables and compute their mutual information, I would get for all pairs a mutual information of zero. That's part one and the second thing that it's trying to do is that it's trying to make certain that the mutual information between all of the features, y and the original feature space x is as high as possible. So in other words, we want to be able to reconstruct the data. We want to be able to predict an X from a Y and a Y from a X. While at the same time making sure that each of the new dimensions is in fact mutually independent. In a statistical sense. I think I'm going to need an example.

2.3.8 Independent Components Analysis Two

So it is important to get an example I, I think that the, the best way to kind of see an example here is to draw a little picture for you that tells you what the sort of underlying assumption behind ICA is. So here's the kind of, or at least the way I see what the fundamental assumption of ICA is. And that is this. You're, you're assuming that the, there's a bunch of. Things in the world. And these are going to be hidden variables. And they've got some properties that, that are associated with them. Their random variables. They are mutually independent of one another. That is if I know the value of one it doesn't tell me anything about the value of the other. But we don't know what they are. And what we get to see are observables. This observable are given rise to by the value of the hidden variables and they somehow combine in a sort of linear fashion and our job, the job of any learner in this case, any unsupervised learner in this case is given your observables try to find the hidden variables. Under the assumption that these hidden variables are in fact independent of one another. So, what's a concrete example of that? I'm going to give you a concrete example of that then I'm going to give you a demo that we found on the web. Okay? Okay. So, one problem where we know this is true is something called the blind source separation problem. So what' s the Blind Source Separation problem? It also has another name which is the Cocktail Party Problem do you remember the Cocktail Party Problem? I think so, yeah. So, describe it to me. So, not that I go to a lot of cocktail parties, but if you're in a, in a large group of people like in a cafeteria or something like that. Mhm. And you're trying to listen to a conversation. It can be very challenging because theres all these different conversations happening simultaneously. Mm-hm. And we need to be able to pull out that one source that, that we're, that we're caring about, so that we can listen to it while kind of separating it out from what all the other noise sources are. So what you just described to me is that somehow you have a bunch of people, so there's Charles, there's Michael and there's a push car. And there all talking at once. And let's a imagine in place of ears we have microphones. And these microphones are actually recording everything that well everything that they hear. So this means that this first microphone is going to hear me. The second microphone is also going to hear me and so does the third microphone... However, they're going to each hear me at a slightly different volume at a slightly different delay. Okay? Because they're placed sort of randomly around the room. You with me? I think so, yeah. Okay, surely, Michael, who just keeps talking and talking and talking mostly about puns, is also going to be picked up by each of the three microphones, and Pushgar, mainly complaining that we aren't doing enough quizzes, is also going to be heard by the three microphones. In this case if we look at our other examples, the people are the hidden variables. That is, they're the things that are causing events to happen in the world. But what we have are observable's. Are the microphones, and what it is that they're recording. Now, if we're in a small enough room, it turns out that physics tell us that, even though our voices are all nonlinear and sound doesn't travel in a very linear fashion the way that we would like, it turns out that in a small enough room with all of the reflections and everything, each of these microphones are well modeled as receiving unknown linear comment, combination of all of the sources. So this means that each one of these microphones actually contains a, a different linear combination of each of our voices. Hey, you with me? Yeah, it's interesting. Right. And so, really, if I were to give you these three recordings and I wanted to figure out what say, Michael Lipman was saying, all that information is here, but I can't extract Michael Lipman from microphone one, or microphone two, or microphone three. But given all of these microphones, I can in fact recover, just Michael alone or just Charles alone or just Pushcar alone, because ICA exactly tries to find this notion of mutual independence without losing any information form the output. And this model here, of three people talking, mostly independently, and being recorded by linear combination by different microphones is exactly the model that ICA was designed to represent. So, just to sort of drive this home, let me give you an actual example that we found on the web that they do exactly this problem. Okay? Sure.

2.3.9 Cocktail Party Problem

Okay, Michael, so I pulled up this webpage. We will provide a link to it, on our own webpages. And you'll notice there's a really nice copyright here, and this is all free to use. So I'm not, I'm not doing anything wrong here. They're going to use Independent Components Analysis, as a way of recovering, original sounds. So, here you see that I've clicked onto a police car. Somebody talking in a commercial and, let's say, this dude here. And, what it's going to do is it's going to, generate sounds from each of these three sources that are all independently generated. And, it's going to mix them. So if I click on these icons here, of these microphones. See, these look just like what I drew before. You'll be able to hear each of their sounds combined together. So I'm going to put my microphone very close to it so you can hear it. [SOUND] [FOREIGN] And here is another one of the microphones. [SOUND] [FOREIGN] Okay, so, Michael, could you hear that? Yes. Okay. So, what I did is, I played two of the, the, the mix sounds together. And, you

know, to the untrained ear, they sound very much alike. But I think the, the most important thing is, all three of these sounds are over on top of one another, and as human beings we actually do a pretty good job on being able to focus on one of them, because I don't know, we're designed to do that. But machines have historically had a terrible time of doing this. And independent components analysis was the first sort of generic algorithm that was able to solve this. So, we're going to use ICA now to separate the sound sources. And as you see, we get these little gramaphone things. And I'm going to play, each one of these and see if they did a good job of separating those into the original sources. [FOREIGN] The guardians of the electronic stock market NASDAQ who have been burned by past ethics questions are. [SOUND] Okay, so wasn't that kind of cool Michael? I mean we, we took these completely mixed up sounds and we were able to recover the original, by using independent components analysis. That's fascinating. I don't still understand what it's doing, but its pretty neat that it could do that, because it sounded pretty mixed up to me. Well it was, it was in fact each one of these three sources, that we eventually heard were in fact arbitrarily, linearly, combined into each of these separate microphones. And there's really sort of no way in which you'd be able to recover the original sounds, because, there's no reason for you to be able to do that. But people do it all the time. And now with something like independent components analysis, you can also do it. And the reason it works with independent components analysis. And that fundamental assumption that it's making, is that whatever is generating in these cases these sounds, the sources, are statistically independent of one another. Which happens to be true in that case. And they're combined together in a way that is a linear combination. Now there's a lot a details here. And again when you read the materials you see exactly how independent components analysis works. But, intuitively all it's doing is taking this particular model, and by using mutual information directly to find things that are independent of one another, while not losing any information, from the observables, it's able to reconstruct these sounds. And it's able to do this incredibly quickly, and incredibly well. Under a large number of conditions.

2.3.10 Matrix

Okay Michael. So let's try to be a little bit more detailed about, kind of, the mechanics of how you, how you would make this work. The algorithms for finding Independent Components Analysis, are in all the reading. But, I do think it's pretty easy to kind of get lost. If you don't, sort of, turn these matrices into actual numbers. So, let me just, sort of, give you a quick example of how we would do this specific thing here. And see if that helps, okay? Mm-hm. Okay. So, how would you go about turning this into a problem that something like, you know, an actual algorithm that uses numbers could do. Well, it turns out it's fairly straightforward. And let's just take this, this particular example here, with people talking into a microphone. Basically you create a matrix. Which are samples of all of the sounds. So, in our original space, we have, you know, this handsome person here. We have this other, let's say, similarly handsome but in a different way person here, and we have this other person, with my poor attempt to draw hair over here, and they're talking. Well, what we end up doing is we basically take the sound wave and we sample it. And what does it mean to sample it? Well, you know, you represent sound in a computer as a sequence of numbers, just like you represent pictures as a sequence of numbers, and in fact, you represent words as a sequence of numbers. And so we basically turn these sound waves into a matrix full of values. So, as I described before, Michael, each one of these microphones. Is actually getting a linear combination of speech from each of these three individuals. So if we think of microphone one, it is seeing some kind of sound wave, which is again a linear combination from each of these people generating a sound wave. The same is true for microphone two and similarly. For microphone three. Now, of course, we're talking about recording these and we're thinking about computer programs, which means we take this continuous sound wave and we sample it at a very fast rate. And what we end up with is a sequence of numbers that represent the particular pressure that we're getting in these sound waves. So, these sound waves get converted into a sequence of numbers and I'm just going to make up some numbers for this. And now what we have is a matrix where each row represents a feature in our original feature space. And by original here, I mean, the feature space that we see each of our microphones and every column represents a sample. Say at time0, time1, time2, and so on. Does that make sense? Yes. And so, since each of these is a linear combination of these voices we get here. Our goal is to find a projection like we did before and the original definition of the problem, such that if we projected this on to here, we would end up with a new feature space that corresponds individually to each of these people. And you could do this with anything, it doesn't have to be sounds, but if we think about the. Adhoc query problem that we went through earlier, each of these would be words, and say their presence of words would be your features, and each of these columns would be documents, say. And the goal would be to find a projection given a set of documents that allowed us to recover some underlying structure that was useful for classification, or for information retrieval. Does

that make sense? Yeah. And can you say, what does it mean for that sequence to have mutual information? so as you recall. Because I know that you listen to Push Cars. A discussion about mutual information and entropy and so forth. All mutual information is is a measure of how much one variable tells you about another variable. And, the assumption here is that each of these people talking has no mutual information. That is, they're talking independently of one another, or the sounds that are coming out of their mouths, don't allow us to predict the sounds that are going to come out of another persons mouth. So it may mean that these people are talking to one another as we often do, Charles might be talking to Michael, who's talking to push cars, who's talking back to Michael, who's talking. To Charles, but the exact sound waves that we see are actually independent of one another. So if that turns out to be true, what independent component analysis is trying to do is trying to recover features, in this case. It turns out the corresponding individual speakers, such that their sound waves, or the values that you see, are statistically independent of one another. Okay? And that's what this does. But, since we can always come up with arbitrary projections that are statistically independent, it's important that whatever our new transformation gives us. It actually has some relationship with the original values that we saw. So some how we want to be able to learn from this, into this in a way that we don't lose any information. In much the same way that PCA was trying to minimize the loss of information, and so we not only want each of the individual new features, in this case people to be statistically independent, we want the amount of data that we get from these people. Compared to the amount of data that we got originally from the microphones, to actually strongly predict one another. And if the mutual information between the microphones and our candidates for the people's voices have high mutual information, then it means that we haven't lost anything. And so those two things together, those two constraints. Forces into a situation where, if this model is true, we construct the original voices.

2.3.11 PCA vs ICA Question

Okay Michael, so I'm going to see if you understood that fire hose of words that I threw at you by giving you a quick quiz. Thanks. Yeah, you're welcome. So, what I have down in the middle here is a bunch of phrases that might describe PCA, might describe ICA, might describe both of them or might describe neither. Okay? And what I want you to do is check off under PCA each one of these that applies. And check off under ICA each one of these that applies. Okay. Does that make sense? Sure. Okay. Go.

2.3.12 PCA vs ICA Solution

Okay Michael you got answers for me? Think so. Okay good. Alright so let's look at the first one. Mutually orthogonal. Does that apply to PCA, ICA, both, or neither? So, it was one of the defining properties in PCA, so I would say PCA. Okay that's fair enough. That is actually correct. What about ICA? I don't, I don't know. It's not one of the defining features. It wasn't, it wasn't even thinking about orthugonality. That's right. And in fact ICA by finding independent. Projections are almost all but guaranteed to find ones that are not mutually arthugoial, it doesn't care at all. So in fact this is what makes PCA a global algorithm since it has this global constraint of mutual arthugoality. Where ICA really in its definition that cares about that, so this should be unchecked. Okay. I'm going to put an X there to represent unchecked. Okay, got it? Yeah. Okay. What about mutually independent? So, that was how ICA was trying to construct its, I don't want to say features, yeah I guess the, the transformed features. That's right. It was trying to create them to be mutually independent, so I would check ICA in that case. And, PCA didn't use that language at all, so I would just not do that. Okay. That's fair. I will point out though, that it turns out that PCA is trying to do something that sometimes will create things that are mutually independent. But we'll see that when we answer the next question. But you're right, PCA does not care about mutually independents. Okay, what about the third phrase, maximal variance? Alright. Again, I feel like this was one of the defining features of PCA. So it was trying to choose dimensions that maximize variance. That's right, and what about ICA? In terms of variance? Again, the, the, there, it wasn't really discussed in that context. Right. ICA is specifically interested in notions of independence, statistical independence. Not inpu, not in issues of varience. So in fact ICA does not care about maximizing varients. Now that we have gotten this far let me point something out, it turns out that because of this arthugonal deconstraint this lack of independent contraint, but this gold and maximizing varience, there are cases under which PCA happens to find independent projections. What's really going on here with these three constraints or, or lack of at least one in this case, is that PCA is tending to find things that are uncorrelated. By maximizing variance along arthogonal dimensions is finding uncorrelated dimensions. And that makes some sens given what it's trying to do. But that is not the same thing as finding things that are statistically independent. Again there is this particular case, there is a

specific case where that does work out, and that's when all of your data is in fact Gaussian. Oh, interesting. And why Gaussian? Because it turns out that the distribution that maximizes variance is in fact the normal distribution. That's my interpretation of a normal distribution. So maximizing variance means that what PCA is doing is it's trying to find a bunch of orthogonal Gaussians. More or less. Does that make sense? Yeah but, you were saying that it, and it aligns with ICA in that case? In that case yes, because it turns out that uncorrelated ends up being independent under very specific distributions. But that's a coincidence it's not a normal fact. Ha, ha. But by the way this, this is probably worth pointing out here something that at least I think is kind of interesting here. Which is since ICA is only trying to find things that are. Independent of one another. Here's our little symbol for independence. As opposed to things that are uncorrelated, things that are [INAUDIBLE], it turns out that whatever it is PCA is doing, it is not working under the same underlying model as ICA. let's think about what ICA is trying to do. ICA is trying to find a bunch of these prjections all of which are statically independnet. RIght? Mm-hm. What happens if I take a whole bunch of statically independent variables and I add them together? In other words I create linear combination of them. What am I going to get? A bunch of sums of things that are independent? Right and what does that tend towards in the limit? I want to say that the law of large numbers tells us that it turns normal. That's exactly right. If I take a bunch of independent variables and I sum them together, that is, I create a linear combination, I in fact end up with a gouache. And that is the central limit theorem. So one argument you might make is that if you believe in a world where there are independent causes giving rise to observables, which is what ICA believes, then whatever you do, you should not be maximizing variance, becasue you're guaranting the summing together otherwise independant variables. Oh, I see. I see, I see. So by trying to find things that are maximal variants it's trying to mix together through the central unit theorem all these things that are independent, so it's, it's, it's, it's, specifically not teasing apart the independent things, it's trying to smush together the independent things. Right. Under certain assumptions about the distributions of these individual variables. So another assumption that I see us making is not just that these variables are independent, but that they are hightly non-normally distributed. And if that's the case, then ending up with things that look like gaussians, has got to be exactly the wrong thing to do. If that assumption holds to be true. Okay. What about maximum mutual information? My understanding of what you describe for ICA said that this is what it's trying to do. It's trying to find a new feature space where the features are maximally. The mutual information between them is large as possible. So I would check the ICA in that case and not the PCA. Let me be Let me clarify soething you said. How can it be trying to maximize mutual information while also trying to make things that are mutually independent. I think you would describe them both in the same language. So what is it trying to make mutual independent? Different features. The, the new, the new transform features. Right. So each of the new transform features is independent with all the other new transform features. So what is it trying to maximize mutual information between? Oh, the, I see. The the information that content of the original features and the new features. Right. So this is about joint mutual information between all the original features together and All of the transform features. There it's trying to maximize mutual information. But inside the new transform features it's trying to make them pair-wise mutually independent. Alright. I think I said that wrong because I understood it wrong. So thank you for clarifying. You're welcome. But now you understand it right. Maybe. Okay, let's go with that because you got the checkmark right. What about PCA? Just X that. I don't understand what that would mean. Right. So if I were to put something here that it could get a check mark for, what I put down is maximal reconstruction. right. Right. And notice that maximal reconstruction of your original data is not the same thing as maximizing mutual information. Thought of course in the limit they work out to be the same. Interesting, okay. But the one project that maximizes variance is not necessarily the same as the first projection you find for maximizing mutual information. So these things really are doing two completely different things. Okay, last two what about ordered features? So, in PCA, it was actually assigning, you know, taking the maximum variant to dimension first and then the next, Mm-hm You know, after that's been subtracted out, whatever has the largest remaining variance and so forth. So that the features end up coming out in, in a very specific order. And it has the property that you could drop the, the last group of features if you want to still have as high a reconstruction area as possible, given the number of features that you keep. So I would check PCA for that. Okay, good. What about ICA? You didn't say anything about the ordering or how you'd actually find features. It seemed like in the blind source separation example you gave It just, came out with the three, so I'm going to say not ordered. That's right and in fact, if you think about the blind source separation example, how in the world would you order people anyway. I mean, other than in the obvious way. It just doesn't really mean anything. I say it doesn't have a notion of causes being more important than other causes merely that they're independent. So, it doesn't really worry about ordered features. It turns out in practice That you can actually try to order the features by using something called kertosis. Which is the fourth central moment of a distribution. But that's really just something that's

useful in some specific cases, almost by coincidence. ICA, itself, does not particularly care, about ordering. At least not classical ICA. Okay, what about the last one, bag of features? So, I. Would view what you just said about ICA as implying that what ICA produces is a bag of features. There's no particular ordering to them. That's right. It's just a collection of things that make up the whole. I guess, you know, PCA, after you've thrown away whichever features you don't want. The features that remains are just features. They could be treated as a bag, I guess. So I don't know if I would check that or not. I. For symmetry I guess I would say not. Okay, but I'm going to say yes because in ordered set of features is still a bag of features. But we would accept either. Either a check or an x, both are sort of fine. So then, what do we really learn from this, Michael? I think what we've learned is these things have fundamentally different assumptions and are really trying to do completely different things. Okay. The only thing they have in common is that they're still trying to capture the original data somehow. Alright. I understand that, but I also learned the opposite, which is that they are really closely related and are trying to do very similar things. Yeah. But their underlying models are different. So maybe that, that's actually a good point, Michael. So maybe a better way of saying it is, their sort of fundamental assumptions are different. Even though they're trying to do the same thing, which is capture the original data in some new transform space that is somehow better. But if you think about it that way, there are two different optimization functions, two different fitness functions, two different cost functions. So even though they're trying to do the same thing. Reconstructing. Keeping the data around. Their basic assumptions about the way that data is constructed is very different. Okay. Okay.

2.3.13 PCA vs ICA Continued

Okay, so, let me give you just a few more examples, Michael, of a, how PCA and ICA differ. And I'm going to do this mainly by talking about certain things that ICA does. And this is really just for your edification, but I think it really helps, to think about, those sort of underlying models and how they differ, by thinking about how they react differently to different kinds of data. So, just, here's a couple of examples. The one we covered right away, was the blind source separation problem. And of course, what, what we recall is that ICA was, in some sense, designed to solve the blind source separation problem and in fact ICA does an excellent job at solving the blind source separation problem. Meanwhile, PCA does a terrible job at solving the blind source separation problem and that's just because it's assuming this kind of Gaussian distribution, it just does not recover what's going on. But here's something that ICA does differently from PCA. Is because it's got this kind of blind source root in it, it's actually directional. And what I mean by that is, you recall I drew this sort of matrix before, where we had features this way. And we had samples of those features like sound, time samples of sound this way. It turns out that for PCA, it doesn't matter whether I give you this matrix or I give you the transpose of this matrix. It ends up finding the same answer. Hm. And that should make sense, right? Because it's just basically finding a new rotation of the data, and these are effectively just rotations of each other. For the purposes of, of, if you just kind of think about it geometrically in space. ICA on the other hand, gives you completely different answers, if you give it this versus giving it this. So ICA is highly directional. And PCA much less so. But ICA, because it's making this kind of fundamental assumption, does end up generating some really cool results. I'm going to give you another example of one. That's like a blind source separation. In fact, I'm going to give you two. Okay? So, imagine you had a bunch of inputs of faces. Okay? So, here's my input faces. I give you bunches and bunches and bunches of faces. What do you think PCA would do? What do you think the first principle component of PCA would be over pictures of thousands and thousands and thousands of faces. Over all darkness of the image. Actually that is exactly right. The first thing that PCA tends to do with images, we are actual, we are talking pictures, not just sketches here, is it finds the direction of maximum variance and that tends to be brightness or luminous. Which kind of makes sense because that's typically the kind of thing that kids vary the most. So in fact, the first thing people often do when they're trying to use PC on faces is they normalize all of that away. Because the first principal component isn't terribly helpful. It's just kind of giving you the average light. What do you think the second thing it would find it would be? Hair versus not hair? No. Interestingly enough. What it ends up finding is sort of the average face. Which is kind of like the question you asked me earlier about what happens if you go through the origin. Mm. So, there's actually names for this. It's called the Eigen Faces because, you know, as I noted before, it's an Eigen problem. And it basically finds kind of the average face. And that's kind of weird. But it works for reconstruction. I don't know what the average face here would look like. Probably something like this. Yeah, so you can see how useful that would be. Not even clear that's a face. But it works for reconstruction. Do you know what ICA ends up finding? Noses? Yes, it finds noses. It finds eye selectors. It finds mouth selectors. It finds hair selectors. And, I think, intuitively the way I would think about this is because PCA

is doing this global orthogonality thing, it's going to be forced to find global features. ICA I didn't say was global, and that's because it's basically local, it doesn't care about orthogonality. So it tends to find parts of. If you feed it the metrics the right way, it tends to find parts of. And so it ends up finding these little selectors. So this has a really nice outcome in cases like natural images or natural scenes. So what do you think happens Michael, if I take natural scenes, you know, pictures of the forest and grass and walking around. Just things that I would see if I were just walking out in the world and taking bunches of pictures. And I feed it into ICA. Well the answer for PCA, by the way, is the same as before, is you get brightness, you get sort of the average image, things that really make sense if your goal is to do reconstruction. But ICA actually gives you something different. What do you think the independent components, the underlying causes, so to speak, of natural scenes are? I'm still thinking about the face parts. But, by analogy, it seems like it would be the, you know, things in the world like, trees, and rocks, and ground. Not quite, and that's I think in part because, there are too many of those things that kind of overlap in too many different ways. It actually finds something more fundamental. Edges. That's exactly right. ICA finds edges. Now for me, that is incredibly satisfying. It says that the independent components of the world are edges. Now there are two things that come out of this. One is just the satisfying feeling you get by realizing that in it there's a algorithm that, on it's own, recovers something fundamental like edges. That's very nice. But the second thing that's nice about it is, once I use ICA to do my feature transformation and discover that what it's learning are edge detectors, well then I can just write edge detectors. I can just write algorithms that are very fast, very efficient, that can go through images of natural scenes and pull out the edges. It's unclear how to do that with, you know, different principled projections, but if edges are the fundamental building blocks of natural scenes then there are people who know how to write edge projectors very quickly. By the way, you get a similar result, I won't talk about it, but you get a similar result in our information retrieval problem where you have documents. And you can read this in the material we gave you. But what ICA ends up giving you for documents are topics. And they're very easily interpreted that way. Collections of words that select for specific topics that are themselves made up of collections of words that select for topics and collections of uncorrelated words that get rid of distracter documents. So the independent components of scenes are edges and the independent components of documents are topics. And that feels very nice, but in both of these cases, both with edges and with topics, it turns out that the form of these topics are something that you can compute very, very quickly. And sort of independently of the underlying ICA algorithm. So why does that matter, Michael? Well, remember what I said originally about unsupervised learning and what it was good for, was understanding your data. Doing kind of data analysis from a human being's point of view. So what something like ICA, and other algorithms we might imagine, do, is they allow you to do analysis over your data, to discover fundamental features of them, like edges, or topics or noses and eyes. And then once you understand that's what's making up your data, you can simply write code that will select for it, or figure out what the important parts of your data are correspondingly. And so here, we haven't just done this feature transformation problem for the sake of doing classification, feature transformation actually helps us to understand the fundamental underlying causes, or at least structure of our data.

2.3.14 Alternatives

Okay Micheal. So we've talked about PCA and ICA. And they both work remarkably well in the specific domains that they're designed for. And they've been applied for decades on a wide variety of problems for doing this sort of future transformation. But I'm going to just very briefly describe two other alternatives, and sort of give you a notion of the space, okay? Sure. Okay, the first one is kind of irritating, but I feel obligated to share it with you. Ad it's called, well it's got many different names, but I'm going to call it RCA just because I like the symmetry. And RCA stands for random components analysis. So what do you think random components analysis does? This is also called random projection. I'm going to guess, instead of finding dimensions with, say, high variance, it's just going to pick any direction. That's exactly right. What RCA does is it generates random directions. Then I guess it projects the data out into those directions. That's exactly right. It's like saying it picks a random P to a random matrix to project your data onto. This matrix is, in some sense, just any random linear combination. And you want to know something? It works. It works remarkably well. At what though? In terms of like reconstruction? At reconstruction. Well not particularly well at reconstruction. But you know what it works really well, it works really well if the next thing you're going to do is some kind of classification. Hm, Now why is it you think that it actually works? Can you imagine why just picking a bunch of random directions and projecting onto those random directions might work? Well, it does mix things together differently. I don't know why the original data wouldn't work. Then this would work better. Unless the original data is

somewhow is purposely made to not work. Well remember what we're doing here, right. We're starting with N dimensions. And we're projecting down to M dimensions, where M is significantly lower than N. So, I started with a bunch of dimensions. Now remember, the real problem here is not that I can't gather the data from the N dimensions. It"s that there's a whole bunch of them, cursor dimensionality. Yes. So I need to have a lot of data. So if I don't have a lot of data, at least certainly not an exponential amount of data, so to speak, and I project a lower dimension, why would random projections still give me something that helps with classification? So it's, it's as if it's maintaining some of the information from these other dimensions, even though there's fewer of them now. They're all kind of mixed together, but they signal might still be there. That's exactly right. And because you project into a lower dimension, you end up dealing with a cursor dimensionality problem, which is sort of the whole point of this, or one of the whole points of this in the first place. And really, a, a way I think of summarizing what you're saying is, that this manages to still pick up some correlations. So if I take random linear combinations of all of my features, then there's still information from all of my features there. So in practice, at least in my experience, Michael, it turns out that M, the number of lower dimensions you project into, for a randomized components analysis, or randomized projections, tends to be bigger than the M that you would get by doing something like PCA. So you don't end up projecting down to sort of the lowest possible dimensional space. But you still project down to a lower dimensional space that happens to capture your correlations, or at least capture some of the correlations, which often ends up working very well for a learner or a classifier down the road. You can actually see how, in this case, you might even project into another set of dimensions M, where those number dimensions are actually bigger than the number of dimensions you started out with. This, in some sense, is almost exactly what we did with perceptons in solving X or. Basically, you're projecting into higher dimensional spaces by doing this. Does that all make sense? Yeah, I think so. I mean it makes sense to me that it would be not as efficient in some sense, as PCA, because it sort of reminds me of, you know, if I want to, if I want to paint my wall, I can very carefully paint all the little pieces of it. Or, I could just splatter stuff at it. It generally takes more when you splatter because you're not being as systematic, but it does, ultimately, cover your wall. [LAUGH], Yeah. I think that's an interesting analogy, and I'm going to go with an apt one. Okay, so, this sort of thing works. What advantages does it actually have over PCA and ICA? Can you imagine one? There's one in particular which I think sort of jumps out at you. RCA? Well, I don't know. Is that, is that a good quiz question maybe? Sure, let's make it a quick quiz.

2.3.15 Alternatives Quiz Question

Okay, so here's the quiz question then. What is the big advantage of RCA and there is a single word to describe it. So your job Michael is to look into my brain and imagine what word it is I'm thinking of and I gave you a hint by saying it jumps out at you, which means if you don't come up with it you're going to feel really bad. [LAUGH] Okay, so you ready? Ready. Okay. Go!

2.3.16 Alternatives Quiz Solution

Okay Michael, what did you come up with? I have a bunch. Well, so, the big advantage of ICR, RCA, it's cheap. It is cheap. It's simple. It is simple. It's easy. It is easy, in a sort of comical or complexity sense. Sort of practically free. Those are all words like, free, easy. Those are, those are the words that came to mind. You're saying that that's, none of those is your word. No, none of those was the word that I was coming up with. How about, works? No. It did start with the letter f, though. Famous. No. Foul mouthed. That's two words. Friendly. No. Alright, I need more letters. I'm still going back to famous. Fabulous. Fast. That's right, it's fast. But I would give you, definitely give you cheap or easy. Cheap is like fast. So is easy. [LAUGH] Alright. [LAUGH] At least your, that's what the slang term meant where I grew up. Anyway, okay, so anything that captures fast, cheap, easy, whatever would do here. But what's irritating about this and actually things like k means and k and n and a whole bunch of other algorithms, is that they're so freaking simple, and yet they manage to work. And that can be kind of annoying sometimes because, you know, as machine learning people, we want to come up with these complex things that work, but as a very brilliant man once said to me, you must earn your complexity. So, RCA, randomized components analysis, or really randomized projections, has this very simple way of thinking about the world. I'll just randomly throw things together, and it turns out to pick up correlations on the way and worked pretty well. And its big advantage, other than working sometimes, is that it's very, very fast. PCA can take hours in supercomputers, and ICA can take hours upon hours in even more supercomputers. But generating a bunch of random numbers, computers are really good at that, and it happens to go really, really fast.

2.3.17 Alternatives Two

So, the second alternative I just wanted to mention very briefly is something called LDA, which is short for linear discriminant analysis. So, any guess on what linear discriminant analysis might do, Michael? It'll make, well, linear. So it's got a linear in it. Discriminant reminds me of, like, classification lines. Yes. So, what linear discriminant analysis does is, it finds a projection that discriminates based on the label. This actually will feel, this feels a lot like supervised learning, right? You take advantage of the label, and you come up with a transformation, such that you will, in fact, project things into different clumps or clusters, or otherwise separate them based upon their label. So, you can imagine using a lot of algorithms, as we've used in the past. In some sense, what they're doing is, linear discriminant analysis. They're finding lines or finding linear separators between different clumps of points. And if you think about the binary case, for example, you could think of SVM as doing something like this, where what you've done is, you transform your points not into a specific label plus or minus, one or zero, yes or no, but you instead project it onto the line such that it would clump things accordingly. And you use the value of that projection as a way of re-representing the data. Does that make sense? Yeah, and this, this approach seems a little different from all the other ones, in that it's actually paying attention to how the resulting components are going to get used. Right. That's exactly right. It actually is going to try to help you, specifically, with the classification. Alright, that's exactly right. All three of the examples that we gave before, feel almost like filter methods, right? In that they have some criterion they're trying to maximize. Even though randomized projections is, who knows what it's trying to maximize besides randomness. But they don't care about the ultimate learner, or the ultimate label that's associated with them. Whereas LDA does care explicitly about the labels and wants to find ways to discriminate, finds sort of projections or features that make it easier for you to do that discrimination. Now it also does not care about what learner's going to happen next, but it does care about the label. So it does end up doing something slightly different, and works pretty well in a world where you have very simple things that can be lineraly discriminated. Okay? Yeah. So that's it for the alternatives. I actually think that's pretty much it for, feature transformation. So why don't we wrap up. [CROSSTALK] Just a quick, just a quick question, though. [CROSSTALK] So, LDA, I've heard of LDA in this context before meaning something else. Like, latent Dirichlet allocation? Yeah, but that happened long after linear discriminant analysis and we haven't moved past the 90s. Well, I'm just saying that it does seem. And I think it is actually an unsupervised approach. Oh latent, oh no, absolutely. But it is well beyond the scope of the discussion that we're going to have here. All right. As you wish. But it is in fact. Every time you see a D you can think Dirichlet. Which is the computer science pronunciation. Other people pronounce it differently. Yes, because the correct way of pronouncing it is sort of beyond my. It has another syllable in there. Is it German or something? Yeah. Who is Dirichlet. Is it Dirichlet? [LAUGH] That sounded German. [LAUGH] Okay, so let's wrap up Michael. Alright. Okay.

2.3.18 Wrap Up

Okay Michael, so we've gone on a journey of discovery. [LAUGH] Through unsupervised learning, and so my question to you is, what have we learned? I think we learned a little bit about ourselves. And a little bit about America. So what A's have we learned today, Michael? So there was PCA. Mm-hmm. ICA. Mm-hmm. LDA. Mm-hmm. RCA and USA. [INAUDIBLE] USA, we're number one! Whoo! [LAUGH] Okay, we're just going to erase that little bit. [LAUGH] [LAUGH] Okay, yeah, okay, so we learned about a lot of A's today. Uh-huh. Which is the same grade that all our students are going to get, I am sure. That would be great. Or that's if they're truly independent. If they aren't independent, then the central limit theorem says, there will be a normal distribution across grades. Mm-mmm. Mm-mmm. Ring that bell curve. Aw, yeah, baby. Okay. So we learned about PCA, ICA, LDA and RCA. What else did we learn about? Well, I think that was it. But we talked about specifically. we, we talked in detail about the relationships between some of these. Mm-hm. In particular, these are all examples of feature transformation. That's right. Okay, so we found out about the relationships between different transformation analysis. Oh, here's something we learned. We learned that the A doesn't just stand for analysis. In the algorithms, but it actually does stand for the analysis of the data. Because that's unsupervised learning. That's right. And that in particular I gave some examples where ICA tells you what the underlying structure of the data is. You can use it to find structure. So that, for example, the independent components of natural scenes are edges. So it's interesting, because I feel like the other time that you emphasized structure was when you were talking about, mimic which was a piece of work that you did when you were a graduate student. Yep One would almost want to guess that maybe you worked on ICA when you were a graduate student. I actually did. My very first paper as a young graduate student was on mimic and my very last paper as a young graduate student. My actual dissertation, was on independent components analysis. I had that sense from the number of strong

points you felt the need to make [LAUGH] about ICA. Well listen man, really, structure runs my life. As you know, everything about my life is well-structured. Yeah, sure. [LAUGH] Okay, did we learn anything else? So, yeah, so I mean I feel like we spent a lot of time talking about so P, ICA is a more probabilistic kind of modeling, method and PCA is a more I want to say linear algebraic, modeling model. That's a really good point Michael. So, we didn't say it explicitly this way, but actually, even in our own work it often comes up that sometimes, you want to think about, information theory. You want to think about probability. And sometimes, you really just want to think about linear algebra. And you could see PCA as being really about linear algebra. And sometimes only coincidentally being about probability. Where as ICA is all about probability and information theory, and only coincidentally ever about linear algebra. Yeah, that's helpful. That does seem to be a fundamental split in a lot of work that happens in machine learning. Yeah and it, and it makes some sense. I mean, we, we know what the right answer is in probability, but we know what the right answer is in linear algebra. And I guess it's, it's often the case Michael, would you agree that. That the linear algebra approach is often easier to think about, or easier to do in practice and sometimes, it can be interpreted as if it's probability. And that typically breaks down on the edge cases, but you know, you can kind of work around it for sort of common cases. Yeah, that, that the linear algebra algorithms are often cheaper to implement, cheaper to execute less prone to local minima issues. There's sort of a well defined answer that they're, that they're finding. But it's often not quite the answer that you want, and the probability methods give you the answer that you want, but can be very hard to find. Right. And in fact you can see that in ICA and PCA, in that PCA is very well understood. There are lots of fast algorithms for it. And you know that the principle components always exist. Interestingly, we didn't talk about this but by contrast, ICA with its more information, theoretic and probabilistic roots, has a very specific model. And it isn't always the case that that model fits, and so in fact, sometimes you can't find independent components. [INAUDIBLE] Because they don't actually exist, except in the most trivial sense. So it's both more expensive, because of the way you end up searching the space. And it doesn't always produce an answer. But when it does produce an answer, it tends to produce very satisfying ones. Well, I think that's a good place to stop, in the sense that I wanted to know just one more interesting fact about ICA. And now that I've got that, I feel fully satisfied. Well, there's another fact I can tell you which is that it's the only one of these algorithms that start with a vowel. And now I'm more than satisfied. It is always my goal to leave you more than satisifed. [LAUGH]. Alright, then! OK. Well I think we are done with this entire sub lesson mini course thingy. About unsupervised learning. Well that, well that's great! About unsupervised learning. What does that, what does that leave us to do? Doesn't lead us to do anything, at least not with this particular, mini-course. I think actually the description we had here, what we've learned for this particular, lesson actually applies, even going backwards to some of our other lessons. And what we're going to get to do next is, decision problems and reinforcement learning. Ooo, exciting. It is exciting. But I think first people probably have some homeworky stuff to do, projects to do in the context of this minicourse. Yes, and probably an exam of some sort. [LAUGH] [LAUGH] Good luck to everyone on that. Yes, we're absolutely sure you'll do fine. Be sure to go over these lessons and be sure to read all of the material, because there's a lot of detail in the material that wouldn't make a lot of sense for us to cover in this format. But do give it a read, come back, look at the stuff that we've talked about, it should help you understand the intuition behind whats really happening there. Excellent. Well this is great, thanks, thanks Charles, i learned a lot, I did too. Bye Michael i will hear from you soon. Alright, awesome. Bye.

2.3.19 Introduction

Hi, my name is Bushger, and today we are going to talk about information theory. Now, information theory is not really a machine learning algorithm, so we'll, kind of, first understand why we need to learn information theory. Usually, you could teach a whole course on information theory, but for now, it is sufficient to know the basics. So 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, now 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 why. 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.

2.3.20 History

Information theory has a very interesting history. Claude Shannon was a genius mathematician who was working at Bell Labs who came out with this information theory. He's also called as the father of the information age. Why it is interesting, is because at the time Bell Labs had a communication mechanism set up and they had just figured out long distance communication, but they had no idea how to charge people. So you could send a message and they would charge you per message, or they could find out how many words were in those message and they would charge you per word, but none of them made sense because you could sometimes write shorter sentences and can be much more information. So it really became necessary to find out what is information and Shannon was the first person to ever try to work on that problem and figure something out. But information theory has also a background from physics and that is why it has words like entropy in it. The physicists who studied thermodynamics were the first scientist to actually understood information. If you are interested in learning more about the physicists background of information theory, you should read up on Maxwell's Demon. Maxwell's Demon is a very famous part experiment that Maxwell came out with. In physics we believe that energy can neither be created or destroyed, but Maxwell's Demon proves that energy can, can ordered into information and the combination of energy and information can neither be created or destroyed. But let's come back to the big world and let's discuss how Claude Shannon looked at it. So his task was to send messages from one place to the other and try to figure out which message has more information. So, let's start with a simple example.

2.3.21 Sending a Message Question

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 has both the sides which are very similar looking. So its a biased coin so every time I flip its going have the same state. While when I flip this 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. All right? If you also observe the fair coin I have like five heads and five tails. So the probability, so it, so it wa, so it is a fair coin. I got five heads and five tails, so it is a fair coin. So, if I had to transmit this 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? Write down your answers here.

2.3.22 Sending a Message Solution

So, in the first case, we will need ten bits for each of those ten flips. But in the second case, we don't need any bits because the result of the flip is always going to be the same. You don't even have to send this message. The folks at San Francisco will already know what is the result of those ten flips. So realize what we've discovered here, if the output of this coin is predictable, you don't need to communicate anything. But if the output is random, you need the communicate the result of each and every flip. So more information has to be translated. If the sequence is predictable or it has less uncertainty, then it has less information. 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.

2.3.23 Sending a New Message Question

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. They occur, 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 atleast 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?

2.3.24 Sending a New Message Solution

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 be present 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, it occurs 50% of the time. So we can directly ask if it is A or not A. So that'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 one 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 be present 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?

2.3.25 Expected Size of the Message Question

Now let's convert this into a quiz. Now remember, that we use this tree to find out our representation. We need one bit for sending A, two for D, and three each for B and C. So, if you send a message in this language, what is the expected message size? Write your answer down here. Remember the unit for this answer is going to be in bits. Go.

2.3.26 Expected Size of the Message Solution

Okay, so will you work this out? You will need to know the frequency of A, B, C, and D. But we already know that. You will also need to know how many bits each of those symbols require. Now we also know that. So, we will 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. Now, 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 as 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. So the formula of entropy is given as this.

2.3.27 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 that tells you something about this variable. 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 of X comma Y. And, as you can predict, it is given by this particular formula, which is the joint probability distribution between X and Y. And, as you predicted, it is given by this particular formula where P of X comma Y is the joint probability of X and Y. The other measure is called as 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. Now, 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.

2.3.28 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 as mutual information. It is denoted by the symbol I. And it is given as, the entropy of y, subtracted by the entropy of x given 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.

2.3.29 Two Independent Coins Question

Okay, so this is the quiz. Assume that you have two coins, A and B. Both are fair coins, and you flip both A and B. We will assume in this case that A gives us no information about B, which is how it works in the real world. So the probability of A and B is 0.5. Try to find out the joint probability of AB, conditional probability of A given B. The entropy of A, B, and the joined entropy, the condition entropy and imaging information. 'Kay, refer to the nodes or formulas from previous videos and try to answer these questions.

2.3.30 Two Independent Coins Solution

Okay. 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 [UNKNOWN] 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. And 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. 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. And if you substitute the values, we get the conditional entropy as 1. Mutual information between A and B is given by this formula. And 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.

2.3.31 Two Dependent Coins Question

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.

2.3.32 Two Dependent Coins Solution

Okay. So, let's start with the joint probability. Now 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. Now, 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? Okay. Let's use this formula, and see if our answer changes. Okay, 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. Okay, the conditional entropy comes out to be 0. What is a mutual information between A and B, then subtract the entropy of A, and the conditional entropy of A given 0, which is 1 minus 0, which is 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.

2.3.33 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. And you must have heard this term in our previous lectures. So 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. And 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, and usually in supervised learning you are always trying to model our data to a particular distribution. So in that case our distrib, one of our distributions can be of unknown distribution. And we can denote that as P of X. And then can sample our data set to find out Q of 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.

2.3.34 Summary

So, let's summarize what we have learned now. So we, we first understood what is information and we found out that information can be measured in some way and we meausred 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 ourselves to a term called a KL divergence, which is very famously used as a distance measured between two distributions. So this is 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. And yeah. Thank you.

CHAPTER

3

REINFORCEMENT LEARNING

3.1 Markov Decision Processes

3.1.1 Introduction

Hi Michael. Hey Charles. How's it going? It is going quite well. Thank you very much for asking. How are things going with you? I can't complain because I've been barred by a, a judge. [LAUGH] I'm pretty sure you can still complain. Complaining is a fine art that requires lots of practice. Guess what we're going to do today? I'm reading decision making and reinforcement learning which is very exciting. It is. Except the hyphen. Except for the hyphen? Yeah the hyphens wrong. Why? because it's not decision dash making. You're right. I normally don't do that but people always want to put a dash so what I'm going to do is for your Michael. Just for you. I'm going to remove the dash. Thank you. And I'm going to put it over here where it belongs. [LAUGH] No. There. Well, Michael, this is the first of our discussions on the last mini course of machine learning, Reinforcement Learning, which I believe you know a little bit about. I am very interested in Reinforcement Learning. Excellent. So I'm just going to do a little bit of background. It's going to be relatively short and straight forward. We're going to do a couple of quizzes, because I know how much you like quizzes. And then we're just going to go back and forth and see what we get to learn about Reinforcement Learning. Sound good? Excellent. Excellent, okay so, let's get started.

3.1.2 The World 1 Question

Michael, here is a world. In fact, for the purposes of this discussion it is the world. Okay. So imagine the entire universe is well described by this picture over here. Okay? Now this is called a grid world, which is something that people in reinforcement learning love to think about, because it's a nice approximation for all the complexity of the entire universe. Now this particularly world is a three by four grid. So you have one comma one, two comma one, three comma one, four comman one. You have one comma two, one comma three and so on and so forth. For the purpose of this discusision we can think of the world as being a kind of game where you start out in state. Which we're going to call the start state. And your able to execute actions, one of these four, up, down, left to right and the purpose is to wonder around this world in such a way that eventually you make it to the goal over here, this little green spot. You see that Michael? Yep. And under all circumstances, you must avoid the red spot. No. Exactly. Now for this particular example, up does what you think it does, down does what you think it does, as do left and right. But if you find yourself at a boundary, such as right, up here in thi upper-left-hand corner, and you try to go up, you just stay where you are. If you try to go left, you just stay where you are. But if you go right, you do actually end up in the next square. Got it? Think so. Okay. Three last things. One, this little black, space here, is a place you can't enter into, so it acts just like a wall. This green space is the goal, and once you're there it's over. The world is over and you get to start over again. Hm. And once you enter into the red spot,

the world is also over and you have to start over again. So you can't go through the red square to get to the green square. Okay, you got it? Yeah. Excellent. So, here's the quiz. Given this particular world, with the physics I just described to you, and given these actions that you can take, up, down, left and right, what is the shortest sequence of actions that would get us from the start state to the goal state? And you can just type in the words up, down, left and right, separated by spaces, or commas or anything like that. Okay. [LAUGH] Say semicolons are allowed, colons are not. Yeah, good. I am glad you made that distinction. Well, it's an important one to make. Okay, you got it? Yeah, I think so. Alright, cool. So let's see what you get then. Go.

3.1.3 The World 1 Solution

Okay Michael, you go an answer for me? This isn't meant to be a hard quiz. Oh good. Cause I actually have two answers for you. Oh, I like that. So I would say right, right, up, up, right. Okay, so, right, right, up, up, right. But I feel like I could have also said up, up, right, right, right. And you could have. Both of those are acceptable. right, right does what you think it does. You move right and then right and then up and then up and then right. And that takes five steps. Or you could have gone up, up, right, right, right, which would also have taken you five steps. So that's pretty easy, right? Yeah, I thought so. Rag. So yes, either of these would be correct. We're going to stick with this one in particular, since they're equal, because we gotta pick one of them. So this does point out something that often in a decision problem like this, there are in fact multiple answers or multiple optima, if I can tie it back to our randomized optimization discussions. Okay, so you got the quiz, you got the world, you understand what you can do here. Yes? Yep. Okay, cool. Now I'm going to throw in a little wrinkle.

3.1.4 The World 2 Question

I'm going to change the world just a little bit Michael. Okay, that last one was really easy, and it was easy for a bunch of reasons, but one of the reasons it was easy is that every time it took an action it did exactly what you expected it to do. Now I want to introduce a little bit of uncertainty, or stochasticity into the world. So, let me tell you exactly what that means. When you execute an action, it executes correctly, with probability of 0.8. So 80% of the time, if you go up, it goes up. Assuming you can go up. If it goes, if you say down, it goes down, assuming you can. Left, it goes left; right, it goes right. Got it? Yeah! Now, 20% of the time, the action you take actually causes you to move at a right angle. Now, of course, there are two different right angles you could go to, if you go up, you could go either left or right at a right angle. And so that 20% gets distributed uniformally. Okay? Does that make sense? Yeah, I think so. And so if you, if you're in the start state and you try to go up, then there a 10% chance that you tend to bump into the wall. Yes. And then you just stay where you are I guess. Right. So if you, if you decide that you'd have x here, you have a 80% chance of moving up, you have a 10% chance of moving to the right. And you have a 10% chance of moving to the left but, of course, you'd bump a wall and you would end up right back where you started. Got it? Yeah. Okay, good. So, here is my quiz question for you Michael. You recall, we came up with two sequences and the one that I decided to keep was up, up, right, right, right. My question to you, is, what is the reliability of the sequence, up up, right right, actually getting you from the start to the goal, given these probabilities, this uncertainty? And so, we're just going to try that sequence, and ask whether or not it actually got us to the goal. Exactly. And when I say, reliability, I really mean, what's the probability of it actually succeeding? Interesting. Okay. Alright. So let's see if we can figure out the answer. You're ready? I'm ready. Alright. Go!

3.1.5 The World 2 Solution

Okay Michael do you have an answer? I have an answer indeed. Okay, what's the answer? Okay, maybe I don't have an answer indeed. But I have I have the ability to compute one. Okay. I'm, I'm doing some math. So why don't you walk me through the math you're doing. I got .32776. That is correct Michael. Woo-hoo. That was trickier than I thought. Okay well, show me what you did. Alright, so the first thing I did is I said, okay, well this is not so hard, because, from the start-state, if I execute up, up, right, right, right. Each of those things works the way it's supposed to do independently with probability .8. Yep. And so, .8 raised to the 5th power, gives me the probability that that entire sequence will work as intended. Exactly. And what is .8 to the 5th? Do you know? 32,768. with a decimal point in front of it, because you know, powers of 2. Wow. That is correct. But I notice that 32768 is not 32776. No they differ by a little smidge. Mm-hm. So this is what occured to me next is, is there anyway that you could have ended up falling into

the goal from that sequence of, of commands not following the intended path. So since actions can have unintended consequences, as they often do. Mm-hm. I was going to ask okay, so if I go up in the first step, there's a probably .1 that I'll actually go to the right. Yep. From there, if I go up, there's a 10% probability that I'll actually go to the right. Mm-hm. From there, the next thing I do is take the right action, but that can actually go up with some probability, .1. Mm-hm, yep. And then another .1 to get to for the next right action to actually cause an up to happen. Mm-hm. And then finally, that last right might actually execute correctly and bring me into the goal. So I could go underneath the barrier, instead of around the barrier with that same sequence. It just isn't very likely. Okay. Well how unlikely is it? Alright. So I did .1 times .1 times .1 times .1 times .8. huh, so the .1 to the 4 times .8 and that's right so this is the probability that the, 4 of the sequences, 4 of these go wrong. In fact exactly the first 4 go wrong. And the last one goes right. Right. And that's equal to, some very, very small number. And when you add it up. You end up with .32776. In fact that's equal to 0.00008. And that's how you get that number and that's correct. And you'll notice that this this sequence happens to work out to be the second sequence that, we had options for. Yeah, the other thing that took us to the goal, right. Yeah. Was exactly the probability of executing that action executing that sequence of transitions given the first command. Yeah, and I think it would work the other way, too. No, it wouldn't quite work the other way. If you had done the sequence right, right, what is it? Right, right. Up, up right? Yeah. In order for that one to work out, you'd add .8 to the 5 and working. And, for it to work out wrong but work out right, the right would have to send you up and then the ups would have to send you right, and then, yeah, so it would actually work out to be the same. Yeah. So no matter which of the two sequences you came up with, they would have the same probability of succeeding. Neat. Nice. That's actually kind of cool. So, good job, Michael. Very good job on the quiz. A lot of people forget this part. And, in fact, if you forgot that part, and got, just this part right, we actually let you pass. But it was wrong. I was kind of expecting you to get it wrong, but I'm glad you got it right too. Thank you Michael, I appreciate your faith in me. [LAUGH] I wrote the quiz. Actually, I stole this from a book, Oh. which, whose little words are now showing up in front of you, exactly where you can go through the details of this quiz. Okay, alright, Michael. So you might ask yourself why I brought this up, and the reason I brought this up is because, what we did in the first case where we just came up with the sequence up up right right right, is we sort of planned out what we would do in a world where nothing could go wrong. But once we introduced this notion of uncertainty, this, this randomness, this stochasticity, we have to do something other than work out in advance what the right answer is, and then just go. We have to do something a little bit more complicated. Either we have to try to execute these, and then every once in a while, see if we've drifted away. And then re-x, re-plan, come up with a new sequence wherever it is we happen to end up, or we have to come up with some way to incorporate all of these uncertainties and probabilities. So that, we never really have to think, rethink what to do in case something goes wrong. So, what I'm going to do next is I'm going to introduce the framework, that's very common, that people use as a way of capturing this stuff, capturing these uncertainties directly. Okay? Hm. You ready? Yeah. Excellent.

3.1.6 Markov Decision Processes 1

So this is the framework that we're going to be using through most of the discussions that we'll be having at least on reinforcement learning. The single agent reinforcement learning, and it's called the Markov Decision Process. This should sound familiar to you, Michael. Well, you did say we're going to talk about decisions. That's true, and we need a process for making decisions. And we're going to introduce something called the Markovian property as a part of this discussion and I'll tell you exactly what that means in a moment. So, I'm just going to write out this frame work and just, and tell you what it is and what the problem it produces for us. And then we're going to start talking about solutions through the rest of the discussion. So a Markov Decision Process tries to capture worlds like this one by dividing up in the following way. We say that there are states. And states are a set of tokens that somehow represent every state, for lack of a better word, that one could be in. So, does that make sense to you, Michael? Yeah, I think so. So what would the states be in the world that we've been playing around in so far? So, the only thing that differs from moment to moment is where, I guess, I am. Mm-hm. Like, which grid, grid position I'm in. Right. So, I feel like each different grid position I could be in is a state, maybe there's a state for being successfully done or unsuccessfully done? It's possible. But let's stick with the simple one. I like that one because that's really, I think, easy to grasp. So, there are at least, of all the states one could reach, there's, well let's see there's four times three minus one, since you can never reach this state. Although we could say it is a state we just happen to never reach it. So, at most if we just think of this grid literally as a grid there are something like twelve different states. And we can represent these states as their X,Y coordinates, say. We could call this, the start state as say 1,1, which is sort of how I described it earlier. We could describe

the goal state as 4,4. And say this is how we describe our states. Or frankly, it doesn't matter. We could call these states 1,2,3 up to 12. Or we could name them Fred and Marcus. It doesn't really matter. The point is that they're states, they represent something, and we have some way of knowing which state we happen to be in. Okay? Sure. Okay.

3.1.7 Markov Decision Processes 2

Alright so we've got states. So next is whats called the model or the transition model. Sometimes people refer to it as the transition function, and basically it is a function of three variables. It's a state, an action, which I haven't defined for you yet and another state. In fact since I haven't defined what an action is for you yet, let's skip that and actually do define actions for you. So the third part of our model are actions. Actions are the things that you can do in a particular state. So what would the actions be in this world? Well the four different decisions I could make were the up, down, left and right. Though it's maybe a little confusing because I think those were also the four possible outcomes. That's true. Well no, there are other outcomes. You could stay where you were. Right. Right. So these are actually the actions. These are the things that when I'm in a given state I'm allowed to execute. I can either go up, go down, go left or go right. You'll notice that as I described before there was no option not to move. Mm. But there could have been, and there could have been other actions, like teleport, or there's anything that you can imagine. But the point is that your action set represents all of the things that the agent, the robot, the person, or whatever it is you're trying to model, is allowed to do. Now, in its full, generalized form, we can think of the set of actions that one can take as being a function of state Because there's some stage you can do some things and some stage you can't do those things. But most of the time people just treat it as a set of actions. And the actions that aren't allowable in them particular states have no effect. Alright, so we understand states. They're the things that describe the world. We understand actions. Those are the things that you can do, the commands you can execute when your in particular states. And what a model describes, what the transition model describes is in some sense the rules of the game that you're playing. It's the physics of the world. So it's a function of three variables: a state, an action and another state, which, by the way, could actually be the same state. And what this function produces is the probability That you will end up transitioning the state s prime, given that you were in state s, and you took action a. Got it? I think so. so the s prime is where you end up, and the s, a is what you're given. So it's, so you plug these three in. Oh I see, and you get a probability Mm-hm. But the probabilities have to add up to one if you if you sum it up over all the s primes. Right. That's exactly right. So for example if you think about the deterministic case where there was no noise then this is a very simple model. If I'm in the state the start state. And I take the action up, then what's the probability, I end up in the state immediately above it? Was that a 0.08? No, in the, in, when we first started in the deterministic world. Oh, that was probability one. Right, and what would be the probability, you ended up in the state to the right? Probability zero. Right. In fact, the probability that you end up at any of the other states is zero in the deterministic world. Now what about the case when we were in the non-deterministic world where an action would actually execute faithfully only 80% of the time. If I'm in the start state and I go up, what's the probability that I end up. In the state above. That was 0.8 Was the probability, I end up in the state to the right. That was 0.1. And, what was the probability I end up where I started. That was also 0.1. Right, and 0 everywhere else. And, that's just sort of the way it works. So, the model really is an important thing. And the reason, it's important. Is it really does describe the rules of the game. It tells you what you, what will happen if you do something in a particular place. It captures Everything that you know about the transition u, of the world is what you know about the rules, got it? You called it physics before? I called it the physics of the world. Huh. These are the rules that don't change. But they're very different from real world physics. Well, yeah, although they don't have to be. I mean in some sense, you could argue that a mark off decision process, what we described so far, these three things In fact, do describe the universe, the states are, you know, in the positions of all the atoms. The positions and velocities of all the atoms in the universe. The transition miles as you do, take certain positions in the world whatever they are how the state of the universe changes in response to that. And the actions or whatever those set of actions could be. And it can be probabilistic or it's not probabilistic. It's definitely probabilistic. The transition models are by their very definition probabilistic. Gotcha.

3.1.8 Markov Decision Processes 3

Now, I actually snuck something important in here, I actually snuck two things that are important in here. The first is, the, what's called the Markovian property. Do you remember what the Markovian property is Michael? From what? From, I dunno. Actually, where does the Markovian property come from? I'm going

to say Russia. Okay, yeah, from the, from the Russian. Yeah, so I have Russian ancestors, and they passed onto me this idea that. Markov means that you don't have to condition on anything past the most recent state. That's exactly right. The Markovian property is that only the present matters. And they had to pass that down to me you know, one generation at a time, because you know, Markov property. Exactly right, that was very good Michael. So what does this mean? What this means is our transition function, which shows you the probability you end up in some state as prime, given that you're in state S and took action A, only depends upon the current state S. If it also depended upon where you were 20 minutes before then, you would have to have more S's here. And then you would be violating the Markovian property. So is this like, do historians hate this? Well, you know, one never learns anything from history. No, you're supposed to learn from history, or you're doomed to, I don't know, let me make something up, repeat it. [LAUGH] Fair enough. Historians probably don't like this, but there is a way for mathematicians to convince them that they're okay with it. And the way that, you, mathematicians convince you that you're okay with this is to point out that you can turn almost anything, into a Markovian Process by simply making certain that your current state remembers everything you need to remember from the past. I see. So, in general, even if something isn't really Markovian, you need to know what you were, not only what you're doing now, but what you were doing five minutes ago. You could just turn your current state into what you're doing now and what you were doing five minutes ago. The obvious problem with that, of course, is that if you have to remember everything from the beginning of time. You're only going to see every state once and it's going to be very difficult to learn anything. But, that Markovian property turns out to be, turns out to be important and it actually allows us to solve these problems in a tractable way. But I snuck in something else, Michael. What I snuck in is this idea about the transition model, is that nothing ever changes. So this second property that matters for Markov decision processes, at least for the sets of things that we're going to be talking about in the beginning, is that things are stationary. That is these for the purposes of this discussion, it means that these rules don't change. Over time. That's one notion of stationary, okay? Does that mean that the agent can never leave the start state? No, the agent can leave the start state any time it takes the action, then it gets another start state. Then how is it, how is it stationary, then? It's not stationary, the world is stationary. The, the transition model is stationary, the physics are stationary, the rules don't change any. The rules don't change. Right. I, I, okay. I see. Then there's another notion of stationary that we'll see a little bit later. Okay, last thing to point out about the definition of a mark on decision process, is the notion of reward. So, reward is simply a scalr value, that you get for being in a state. So for example we might say, you know, this green goal is a really good goal. And so, if you get there, we're going to give you a dollar. This red dual, on the other hand, is a very, very bad state. We don't want you to end there. And so, if you end up there we're going to take away a dollar from you. What if I don't have a dollar? Someone will give you the dollar. The universe will give you the dollar. [LAUGH] And then take it away? Or, the universe will take it away. Even if you don't have it. You'll have negative dollars. Oh, man. Okay, so Reward is very important here in a of couple ways. Reward is one of the things that, as I'm always harping on encompasses our domain knowledge. So, the Rewards you get from the state tells you the usefulness of entering into that state. Now. I wrote out three different definitions of r here, because sometimes it's very useful to think about them differently. I've been talking about the reward you get for entering into the state, but there's also a notion of reward that you get for entering into a state and taking an action. There's a rewar, or, being in a state and taking an action, there's a reward that you could get for being in a state, taking an action, and then ending up in another state as prime. It turns out these are all mathematically equivalent. But often it's easier to think about one form or the other. But for the purposes of the, you know, for the rest of this discussion, really you can just focus on that one, the reward of the value entering into a state. And those four things, by themselves, along with this Markov property and non-stationarity, defines what's called the Markov Decision Process. Or an MDP. Got it? I'm a little stuck on the, how those could be mathematically equivalent. Well we'll get to that later. Would you like a little bit of intuition? Sure. Well, you can imagine that just as before we were dealing with the the notion of making a non-Markovian thing Markovian by putting a little bit of history into your state. You can always fold in the action that you took to be in a state or the action that you took to get to a state, as a part of your state. But that would be a different Markov Decision Process. It would, but they would work out to have the same solution. Oh, I see.

3.1.9 Markov Decision Processes 4

Speaking of solutions, this is the last little bit of thing that you need to know. And that is. This defines a problem. But, what we also want to have, whenever we have a problem. Is a solution. So, the solution to the Markov Decision Process, is something called a policy. And, what a policy does. Is, it's a function, that takes in a state. And returns an action, in other words, for any given state that you're in, it tells you the

action that you should take. Like as a hint? No, it just tells you, this is the at. Well, I mean, I suppose you don't have to do it, but the way we think about Markov Decision Processes, is that this is the action that will be taken. I see, so it's more of an order. Yes, it's a command. Okay. So that's all a policy is. A policy is solution to a Markov Decision Process. And there is a special policy, which I'm writing here as policy star, or the optimal policy, and that is the policy that maximizes your long-term expected reward. So if all the policies you could take, of all the decisions you might take, this is the policy that optimizes the amount of reward that you're going to receive or expect to receive over your lifetime. So, like, at the end? Well, at yeah, at the end, or at any given point in time, how much reward you're receiving. ¿From the Markov Decision Process point of view, there doesn't have to be an end. Okay. Though in this example, you don't get anything, and then at the end, you get paid off. Right, or unpaid off. Right. If you fall into the red square. So actually, your question points out something very important here. I mentioned earlier when I talked about the three kinds of learning that there, supervised learning and reinforced learning were sort of. Similar, except that instead of getting Ys and Xs we were given Ys and, Xs and Zs. And this is exactly what's happening here. Here what we would like to have if we wanted to learn a policy is a bunch of sa pairs as training examples. Well here's the state and the action you should've took, taken, here's another state and the action you should've taken, so on and so forth. And then we would learn a function, the policy, that maps states to actions. But what we actually see in the reinforcement learning world, in the Markov Decision Process world, is we see states, actions, and then the rewards that we received. And so in fact, this problem of seeing a sequence of states, actions, and rewards. It's very different from the problem of being told. This is the correct action to take to maximize a function. Or find a function that maps from state to action. Instead, we say well, if you're in this state, and you take this action, this is the reward that you would see. And then from that, we need to find the optimal action. So Pi star is being the F from that previous slide? Right. And R is being Z? Yes. And y is being a. And s is being x or x is being s Got you. Right. So but, I'm, okay I'm a little confused about this notion of a policy. So we have the, the, the thing we tried to do to get the goals was up, up, right, right, right. Yes. I don't see how to capture that as a policy. It's actually fairly straightforward. What a policy would say is: What state are you in? Tell me what action you should take. So, the policy, basically is this: When you're in the state, start, the start state, the action you should take is up. And it would have a mapping. For every state that you might see, whether it's this state, this state, this state, this state, this state, this state, this state, or even these two states, and it will tell you what action you should take. And that's what a policy is. A policy, very simply, is nothing more than a function that tells you what action to take at every, in any state you happen to come across. Okay, but the, but the. The question that you asked before was about up, up, right, right, right. Mm hm. And, it seems like, because of the stochastic transitions. You might not be in the same state. Like, you don't know what state you're in, when you take those actions. No, so, one of the things for what we're talking about here, for the Markov Decision Process. Is, there're states, there're actions, there're rewards. You always know what state you're in, and you know what reward you receive. So does that mean you can't do up, up right right right? Well, the way it would work in a Markov Decision Process, so what you're describing is is what's often called a plan. You know, it's, tell me what sequence of actions I should take from here. What Markov Decision Process does and what a pr, a policy does is it doesn't tell you what sequence of actions to take from a particular state. It tells you what action to take in a particular state. You will then end up in another state because of the transition model, the transition function. And then when you're in that state you ask the policy what actions should I take now? Okay. Right, so this is actually a key point. Although we talked about it in the language of planning, which is very common for the people who di, for example take any ag course, the thing about this in terms of planning, what are the things that I can do to accomplish my goals? The Markov Decision Process way of thinking about it, the reinforcement way of thinking about it, or the typical reinforcement learning way of thinking about it, really doesn't talk about plans directly. But instead, talks about policies. Which from which you can infer a plan, but this has the advantage that it tells you what to do everywhere. And it's robust to the underlying stochastic of the word. World. So, is it clear that's all you need to be able to behave well. Well, it's certainly the case, that if you have a policy and that policy is optimal, it does tell you what to do, no matter what situation you're in. 'Kay. And so, if you have that, then that's definitely all you need to behave well. But I mean could it be that you wanted to do something like up, up, right, right, right which you cant write down as a policy? And why cant you write that down as a policy? Because the policies are only telling you what action to do as a function of the state not sort of like how far along you are in the sequence. Right unless, of course, you fold that into your state some how. But thats exactly right, the way to think about this is. The idea of coming up with a concrete plan of what to do for the next 20 time steps is different from the problem of whatever step I happen to be in, whatever state I happen to be in, what's the next best thing I can do? And just always asking that question. Hm. If you always ask that question, that will induce a sequence, but that sequence is actually

dependent upon the set of states that you see. Whereas in the other case where we wrote down a particular policy, you'll notice that was only dependent upon the state you started in and it had to ignore the states that you saw along the way. And the only way to fix that would be to say, well, after I've taken an action, let me look at the state I'm in and see if I should do something different with it. But if you're going to do that, then why are you trying to compute the complete set of states? Or I'm sorry, the complete set of actions that you might take. Okay. Okay, so there you go. Now, a lot of what we're going to be talking about next Michael, is, given that we have MDP, we have this Markov Decision Process defined like this. How do we go from this problem definition to finding a good policy, and in particular, finding the optimal policy? That makes sense. Good. And there you go.

3.1.10 Sequences of Rewards 1

Alright. So having gone through that exercise, Michael, I think it's, it's worthwhile to step back a little bit and think about the assumptions that we've been making that have been mostly unspoken. And I'm going to say that the main assumption that we've been making in some sense boils down to a single word. And that word is stationary. So let me tell you what I mean by that and why by kind of illustrating what it is we've been sort of doing for a little while. Okay? Sure. Okay. So the first thing I'm going to say is that we've actually been. Kind of assuming infinite horizons. So what do I mean by that. When, when we think about the last grid world that we were playing with, we basically said well, you know, I want to avoid going to the end as quickly as possible if I have rewards of a certain value or whatever. Because, you know, the game doesn't end until I get to an absorbing state. Well, that sort of implies. That you basically can live forever. That you have an infinite time horizon to work with. Now can you, can you imagine why if you didn't have an infinite time horizon to work with you might end up doing something very different? Different then what, what we're doing in the grid world? Right, so here let me let me show you the game that we were, the grid world that we were doing before. Might help you think about it. So here's the grid world we had before. And as you recall. We had a particular policy that sort of made sense. Here, I'll, I'll write it out for you again. And this was with a case where we had a reward of minus 0.04. Remember? We just did this. Remember? Yep. Okay, and this was the policy that turned out to be optimal, and in the future I want you to pay attention to here is that when you're over right here near possible end state, rather than going up, it made sense to take the long way around. Because you're going to get some negative reward but it's a small enough negative reward compared to where you might end up. Okay with a positive one. Yeah, I see. And that makes some sense. Well, that only makes sense if you're going to be living long enough that you can take the long route around. What if I told you you only had say three times steps left, and then the game is going to end no matter where you end up? Well, it might be, it might make more sense to take some risk than just try to take the short way because there's really no chance you're going to get to the plus1. I'm entirely convinced of that though because there's still a chance you'll fall into the minus 1 along the way. Right, so the exact val, whether it makes sense to take the risk or not is going to depend upon two things, we've already talked about one of them which is the actual word that you get. If this reward were, you know, negative enough, then clearly it makes sense to just try to end things quickly, right? We just showed that in the last quiz. But another thing that it's going to depend upon is how much time you have in order to get to where you're going. If you've only got one or two time steps before everything's going to end. You can imagine that there are cases where, without changing the reward too much it makes a lot of sense to try to go ahead and quickly get to this plus 1, even though you have some chance of falling into the minus 1. As opposed to, trying to move away, where you're then kind of, all but guaranteed that you're never going to reach the plus 1. So. Whether it makes sense to take the risk or not will depend upon the reward but it's also going to depend upon whether you have an infinite amount of time to get to where you want to get to or whether you have a finite amount of time. And the real major thing I want you to get out of that is that if you don't have an infinite horizon but you have a finite horizon then two things happen. One is the policy might change because things might end. But secondly, and more importantly, the policy can change, or will changee., even though you're in the same state. So, if I told you, if you're in this state right here, and I told you you didn't have an infinite amount of time, but you still had 100 million time steps then, it, I think it's clear that it still makes sense to go the long way around, right? Yeah, I mean the, the probability that this policy is going to last for a million timesteps has got to be tiny. Right. So, I might as well. It's 100 million timesteps might as well be infinity. But if I make that number not 100 million but I make it 2, or 3, or 4. Then suddenly your calculus might change. In particular, your caluculus will change even though I'm in the same state. Right? So maybe this state right here, if I've got a million, 100 million timesteps I still want got to go the long way around, but if I've only got a few time steps, the only way I'm ever going to get a positive reward is to go this way. Does that make sense? I guess so. So, you're saying, for example,

even within the single run, it could be that I'm in a state and I try an action and maybe it doesn't work and I stay where I am. And I try it again, and maybe it doesn't work and I stay where I am. It might then switch to a different action, not because the other one wasn't working, but because now it's running out of time. Right, exactly. So we talked about this notion of a policy which maps states to actions, we talked about this notion about stationarity. So you believe that this sort of Markovian thing said, it doesn't matter where I've been it only matters where I am. And so if i'm in this state, since it only matters where I am, I'm always going to want to take the same action. Well that's only true in this kind of infinite horizon case. If you're in a finite horizon case. And that finite horizon, of course, is going to keep counting down, every time you take a step. Well then suddenly, depending upon the time step that's left, you might take a different action. So we could write that I think, just for the sake of kind of seeing it as some thing like, your policy is a function both the stature and, and the time step you're in. Hm. And that might lead you to a different set of actions. So this is important, this is important, I mean were not, we are not, for, for this course going to talk about this case at all, where you're in a finite horizon, but I think it's important for you to understand that the, without this infinite horizon assumption here. You loose this function of stationarity in your policies. Okay? Yeah. Interesting. Okay. So, that all, I think is, you know, making our something that's obvious, but becomes obvious after someone points it out to you. So, the second thing that I want to talk about, I think, is a little bit more subtle. And, and this notion of utility of sequences. So, as we've been talking, Michael, we have been sort of. Implicitly discussing not just the rewards we get in a single state, but the rewards that we get through a sequences of states that we take. And so I just want to point out a little fact that that comes from that, and where that ends up leading us. And then we'll get to some nice little cute series of math. So. Here's what I want to point what utilities, what we mean by utilities sequences. It means we have some function I'm going to call U for utility over the state, the sequence, sorry. Of states that we're going to see let's call them, S0, S1, S2 and so on and so forth. Well, I think an assumption that we've been making even if we haven't been very explicit about it is that if we had two sequences of states. S0, S1, S2, dot dot dot. And a different sequence S0, then S1 prime and S2 prime, that is two sequences that might differ from S1 on, but all start in the same start state. Okay? If we have a utility for the first, and that utility happens to be greater than the utility for the second, then it also turns out that we believe. That the utility for S1, S2, dot dot dot, is greater than the utility for S1 prime, S2 prime dot dot dot. Alright so these are two different sequences, S one, the S's and the S prime's are two different sequences. Yes. And in the beginning we're comparing them with S0 stuck in front of both of them. And we're saying if I prefer the S0 followed by all the S's, to S0 followed by the S primes, then I have that same preference even with those S0s missing. Right, and so this is called stationarity of preferences. And, another way of saying it is. That if I prefer one sequence of states today over another sequence of states, then I prefer that sequence of states over the same sequence of states tomorrow. So isn't, isn't this just obvious? Because the whatever the rewards for those two cases, we're just adding the reward we get for S0. So. It's going to be the same. But listen to what you just said. You just said, well, it'll be the same, because all we're doing is adding the reward for S0. But what did we ever say about adding up rewards? I thought, I thought that's what we were doing. That's right, that is what we were doing. But we never actually sat down and wrote that down and said, this is what it means. To talk about the utility of a sequence of states as opposed to the reward that you get in one state. Okay, so you're saying that if we, if we are adding rewards, then this follows. Right. Okay. And then I've actually been saying something even stronger, which is, I will show you on the next slide, which is if you believe that this is true, that the utility of one sequence of states is greater than the utility of another sequence of states. Both today and tomorrow. Then it actually forces you to do some variation of what you said which is just adding sequences of states. Or adding the rewards of the sequence of states that we see. That's really interesting. So then, so the adding isn't really an arbitrary thing it follows from this, this deeper assumption. Right, and the reason I bring this up is because. It would make sense if you were to just to grab someone off the street and start talking about Marco Decision Processes. One of two things will happened. Either they'd run screaming from you like you're a crazy person or they would sit and they would listen and if they listen they would just completely buy into the idea that you just add up sequences of rewards. You know, sequences of rewards that you see as a way of talking about how good the states are because that's a very natural thing to do. But it turns out that mathematicall if you have this notion. A sort of stationary of preferences and this sort of infinite arise in world. You really are in a case where this has to be true. And it has to be the case if you have to do some form of addition. Because nothing else sort of can be guaranteed to maintain this property over stationary preferences. I mean, as you said, if I got one sequences of states and another sequnce of states and by just prepending or appending another set of states to it, I'm still going to always guarantee that one's greater than the other. You kind of have to do some form of adding the reward that you see in the states in both cases. because if you don't do that, then eventually this inequality will not hold. So, let me write that down in math terms. And see where that gets us, okay?

Cool.

3.1.11 Sequences of Rewards 2

Alright, Michael. So, let's write that down in math, okay? You like math, right? I do. Okay. So here's the math version of it. I'm going to just say that the utility that we receive for visiting a sequence of states, S0, S1, S2 ellipsis, is simply the sum of all the rewards that we will receive for visiting those states. Sure. Does that make sense? Yeah. Is that consistent with what you were doing when you were thinking about the grid world? Yeah, exactly. But I thought we were going to make that not definitional, we were going to derive that it had to be that way. No, we're not. They can read about it. It'd take me, like, an hour and a half to do it. Alright. And I'm already losing my voice. But. What I do want people, I want you to believe is that the utility of the sequence of states, thinking of it as the sum of all the rewards, makes sense, right? And then if nothing else, at least it makes, it's consistent with what you've been doing as we've been talking about this grid work. Yeah, absolutely. I mean it, it also kind of makes me think about money. Go on. Well, so, so, I mean, that's sort of how money works. If we get some kind of payoff each day, those payoffs get added to each other. They don't get, you know, subtracted or multiplied or square rooted or whatever. They just, you know, they go into your bank account, and things add. So this feels like, kind of like that. It's like money in my pocket. Sure, if you have a big enough pocket. Okay, I'm with you on that. Alright, so I'm going to say that this all makes sense. It, it's really awesome and it sort of doesn't work. And to illustrate why this doesn't work, I'm going to give you a quiz. Yay. because you like quizzes. So I've been told.

3.1.12 Sequences of Rewards 3 Question

So, here's the quiz. You see on the screen this sort of unexplained two little squiggles with a bunch of numbers in front, on top of them or under them or near them? Yeah I, I assume that's a river and on one bank it's the land of the plus ones and the other bank has been infilitrated by plus twos. That's not what I intended at all but I actually like that enough that I'm going to pretend, that that's what I intended. So these work out to be sequences of states. Oh, I see now. Okay, and rewards that you receive. No, no, no, it's a riverbank and on one side of the bank, as you walk along it, you get a bunch of plus ones. On the other side you get a bunch of plus ones and some plus twos. And this just goes on forever, okay? And, I'm not going to tell you what all the rewards are after that except that they, they look similar to the rewards that you see here. Okay. Plus ones and plus twos, okay? And let's say the top, the top one, in fact, nothing but plus ones. And the bottom ones are some plus ones with maybe some plus twos sprinkled in and out, but those are really the only options. Okay? Yeah. Here's my question to you. So, would you rather be on the top side of the river bank, or the bottom side of the river bank? Okay? You think you know the answer already, don't you. Yeah. Okay, cool. Well then, let's, let's give our listeners a, a chance to come up with the right answer.

3.1.13 More About Rewards 3 Solution

Okay, Mikey, you got answers for me? sure. Alright. Wait, should I, do need to make any assumption, or can I make any assumptions I want about the non-blue states? No. It doesn't matter. [LAUGH] Actually. [LAUGH] So the answer is no. Maybe it does. The answer is no. But it doesn't actually matter because Remember, it's all very stationary and Marcovian. So the only thing that matters is where you are. You're stationary Marcovian. Alright, so I think, okay. So let's, let's do the first one, then. Okay. So, so this is, this is pretty cool. So by changing the reward the way that you did, it is now, it's not a hot beach anymore. It's like. You know like, super awesome money beach. Yes, it's like a beach made of bacon. Sure. Okay. Let's think of it that way. Although neither of us is eating bacon at the moment. At the moment. But. [CROSSTALK] Great, because, because then our mouths would be full and it would be very difficult to give this lecture. Alright so, no, no, no,no, no I mean like, even. Alright, never mind. The point is that it's awesome, and there's like diamonds. I don't know, I just saw The Hobbit movie, and there is a room filled entirely with treasures. And so, this is kind of like that. And so now the question is, what should I be doing, like I should never go to the plus 1 or the minus 1, because that ends my treasure gathering. I should just continue to stay in states that aren't those states. Okay. So I would say left for the, for the top state. Mm-hm. Let's say left for the bottom state. This one? The, yes, uh-huh. Okay. And the other two, I need to think about for a moment. So I feel like I'd like to get away from the minus 1. Mm-hm. So, like, you know, to go up. But then that would give me some chance of slipping, wouldn't it? Mm-hm. let's see, I don't want to go to the right. I don't want to go down. I don't want to go. I'm going to go to the left. Mm-hm. I'm going to bash

my head against the wall, because then I get money. Yeah. That's weird. There's really no pain for running into the wall? No. You just stay where you are. Alright and for the bottom one, I feel like it's the same kind of thing. I don't want to go to the right or to the left, cause then there's a probability that I'll slip and end up ending my game. So I'm going to go down. yeah. I like that. So most of these are right, and one of them is wrong. Or, it's not that it's wrong, it's just that's not as right as it could be. Which one do you think that is? I think they're right. But if, if there's one that I could imagine you not being so happy with, which is the bottom one, There are two bottom ones. You mean this one? Yeah. Yeah. What would I like? I think you might prefer if I bash my head against the wall. No, actually, it turns out that this is a correct answer so all four of these are correct. But in this particular state, it actually doesn't matter which way you go. Oh, I see. A more complete answer. Yes. Would be, doesn't matter. Yeah. And is that true of any of the other ones? No. I guess, I guess you're right. The other ones, you have to go in the direction indicated. Right. So, by the way, just like here, it doesn't matter which direction you go in. It's actually true for all of the other states as well. It doesn't matter where you go because In these three states, these are the only states where you can accidentally, you could end the game. And in each of them, you can take an action that guarantees that you don't end the game. So it really doesn't matter about where you do it in the other states. Cool. Right? So that's a good argument. This makes a lot of sense. So at the end of the day, basically, because the reward is positive. And you're just accumulating reward or diamonds and treasure, as you put it. You just never want to leave, so you always want to avoid either of these states. It doesn't matter that there's a +1 here and a -1 here. If I can stay here forever, then I just accumulate positive reward forever. Okay, so what's the next one? So, interesting. So now, now the beach is really, really hot. It's minus 2, so. Mm-hm. I want to get off here as quickly as I can. So definitely the top one, I would go to the right. Just try to get, get that +1 and get out of, get outta here. Yeah. Alright, so then, let's think about the bottom right. Okay. So, in the best of all possible worlds I go around and dump into the +1. And that would take me again if, if I don't slip at all one, two, three, four steps to do that Mm-hm. Actually wait one, two, three, three steps one the hot sand. Yup And I'm getting and -2 for each of those. So that's like minus 6. Mm-hm. So I could get that plus 1 at the end, and that makes it only minus 5. But I think I'd rather just get off the beach. So I'm going to say in the bottom right to actually dive into the pit of pain, because it can't be as bad as where I am. That's exactly right. So, okay, so then the, the one next to it on the left is a little trickier. If I go up to the plus 1, I get a minus 2 Followed by a +1 which is also -1. Mm-hm. So okay. I'm not so sure about that one. Well remember you'll always have some probablitly of ending up back where you started Is that true? Oh because, if I. Well no, no, that's not true if I dive straight into the -1. Then I have 0.8 chance of going to the -1. 0.2 of going up. To the arrow that's labeled, 0.2 going down to the blue box that's not labeled Mm-hm. Wait, 0.2? No, 0.1 Mm-hm. Whereas if I go up, right, I might stay where I am. And accrue -2, like, yeah I feel like I need a calculator for this. Yeah, but you're, I think your intuition is right, though. Just think about it as How much, I mean what do you think has the best chance of most quickly ending things? Well certainly jumping into the -1. Right. I think that's going to be marginally better because there's only one chance of slippage and delay. Where as if I take additional step, there's two chances of slippage and delay. Exactly, that's exactly the right kind of argument to make and, and because the -2 is so big. You're you're going to, you're just better off ending it even if you ended in pain, in the same way that you were in the bottom right hand corner. And so bottom left I would say the cleanest thing would be to jump into the -1. So to go to the right so that I can go up and get into the -1. And what's nice about that move is that is that you either end up in the bottom right hand corner. Or you end up staying where you are, or you end up moving up. But in either case, you are always sort of, you know, you never get farther away, right? Whereas if you try to go up, there's a chance that you will get farther away. Interesting. Is there a way to be sure about these? Yeah, you just have to do the math. [LAUGH] Do we know how to do the math? Yeah, you just start, you just do expectations. You can basically just say, well what's the expected time. Since you know what you're trying to do here is to get to the end. You can just, you can calculate the expected length of time it'll take you to get there. And then just, you know, multiply by all the rewards you're going to get in the meantime. If it helps, though, I will tell you that for any value where the reward is less than minus 1.6284. [LAUGH] This is the right thing to do. [LAUGH] Okay. Just so you know. Alright, so you're saying you actually do know that this is the optimum. The thing that we have here. Yes, I know that this is the optimum. Alright. Let me just draw out the rest of it for you. So it's, it's I think interesting to compare this one. The bottom right one, where you have given me negative rewards to encourage me to end the game. With our main one up here in the upper left. Where you have also given me some negative reward to encourage me to end the game. If you look carefully you'll notice that in this one, you're encouraged to end the game. But you're really encouraged to end the game at a plus 1. Here the reward is so strongly negative, you just need to end the game. And so you end up with a different response, here. Because this is, is just as quick to get to the minus 1 and into pain as to, take this extra step and try to get to the plus 1. And

you end up with a different response here. Here, here, and here. That's really interesting. Yeah, it is. And so, these changes matter. They matter a great deal. So, how am I suppose to choose my rewards? Carefully and with some forethought. Nice. But, you know, again There's lots of ways to think about this. So the way we've been sort of talking about MDPs is kind of the way we've been talking about we talked in the first part of the course. When we talked about having teachers and learners. You can think of your reward as sort of the teaching signal. Sort of the teacher telling you what you ought to do, and what you ought not to do. But another way of thinking about this is that because the rewards define the MDP The rewards are our domain knowledge. I see. So if you're going to design an MDP to capture some world. Then you want to think carefully about how you set the rewards in order to get the behavior that you wish. That's fair. I mean, it seems a little bit like a cop out, but I think it seems like a necessary one. Yeah. I mean, there's really, I mean, again. No matter what you do, you've gotta be able to inject domain knowledge somehow. Otherwise, there's no learning to do. And in this case, the reward basaiclly is telling you how important it is to get to the end. Cool. Okay. Alright. Let's move on.

3.1.14 Sequences of Rewards 1

Alright. So having gone through that exercise, Michael, I think it's, it's worthwhile to step back a little bit and think about the assumptions that we've been making that have been mostly unspoken. And I'm going to say that the main assumption that we've been making in some sense boils down to a single word. And that word is stationary. So let me tell you what I mean by that and why by kind of illustrating what it is we've been sort of doing for a little while. Okay? Sure. Okay. So the first thing I'm going to say is that we've actually been. Kind of assuming infinite horizons. So what do I mean by that. When, when we think about the last grid world that we were playing with, we basically said well, you know, I want to avoid going to the end as quickly as possible if I have rewards of a certain value or whatever. Because, you know, the game doesn't end until I get to an absorbing state. Well, that sort of implies. That you basically can live forever. That you have an infinite time horizon to work with. Now can you, can you imagine why if you didn't have an infinite time horizon to work with you might end up doing something very different? Different then what, what we're doing in the grid world? Right, so here let me let me show you the game that we were, the grid world that we were doing before. Might help you think about it. So here's the grid world we had before. And as you recall. We had a particular policy that sort of made sense. Here, I'll, I'll write it out for you again. And this was with a case where we had a reward of minus 0.04. Remember? We just did this. Remember? Yep. Okay, and this was the policy that turned out to be optimal, and in the future I want you to pay attention to here is that when you're over right here near possible end state, rather than going up, it made sense to take the long way around. Because you're going to get some negative reward but it's a small enough negative reward compared to where you might end up. Okay with a positive one. Yeah, I see. And that makes some sense. Well, that only makes sense if you're going to be living long enough that you can take the long route around. What if I told you you only had say three times steps left, and then the game is going to end no matter where you end up? Well, it might be, it might make more sense to take some risk than just try to take the short way because there's really no chance you're going to get to the plus1. I'm entirely convinced of that though because there's still a chance you'll fall into the minus 1 along the way. Right, so the exact val, whether it makes sense to take the risk or not is going to depend upon two things, we've already talked about one of them which is the actual word that you get. If this reward were, you know, negative enough, then clearly it makes sense to just try to end things quickly, right? We just showed that in the last quiz. But another thing that it's going to depend upon is how much time you have in order to get to where you're going. If you've only got one or two time steps before everything's going to end. You can imagine that there are cases where, without changing the reward too much it makes a lot of sense to try to go ahead and quickly get to this plus 1, even though you have some chance of falling into the minus 1. As opposed to, trying to move away, where you're then kind of, all but guaranteed that you're never going to reach the plus 1. So. Whether it makes sense to take the risk or not will depend upon the reward but it's also going to depend upon whether you have an infinite amount of time to get to where you want to get to or whether you have a finite amount of time. And the real major thing I want you to get out of that is that if you don't have an infinite horizon but you have a finite horizon then two things happen. One is the policy might change because things might end. But secondly, and more importantly, the policy can change, or will changee., even though you're in the same state. So, if I told you, if you're in this state right here, and I told you you didn't have an infinite amount of time, but you still had 100 million time steps then, it, I think it's clear that it still makes sense to go the long way around, right? Yeah, I mean the, the probability that this policy is going to last for a million timesteps has got to be tiny. Right. So, I might as well. It's 100 million timesteps might as well be infinity. But if I make that number not 100 million but I make it 2, or

3, or 4. Then suddenly your calculus might change. In particular, your caluculus will change even though I'm in the same state. Right? So maybe this state right here, if I've got a million, 100 million timesteps I still want got to go the long way around, but if I've only got a few time steps, the only way I'm ever going to get a positive reward is to go this way. Does that make sense? I guess so. So, you're saying, for example, even within the single run, it could be that I'm in a state and I try an action and maybe it doesn't work and I stay where I am. And I try it again, and maybe it doesn't work and I stay where I am. It might then switch to a different action, not because the other one wasn't working, but because now it's running out of time. Right, exactly. So we talked about this notion of a policy which maps states to actions, we talked about this notion about stationarity. So you believe that this sort of Markovian thing said, it doesn't matter where I've been it only matters where I am. And so if i'm in this state, since it only matters where I am, I'm always going to want to take the same action. Well that's only true in this kind of infinite horizon case. If you're in a finite horizon case. And that finite horizon, of course, is going to keep counting down, every time you take a step. Well then suddenly, depending upon the time step that's left, you might take a different action. So we could write that I think, just for the sake of kind of seeing it as some thing like, your policy is a function both the stature and, and the time step you're in. Hm. And that might lead you to a different set of actions. So this is important, this is important, I mean were not, we are not, for, for this course going to talk about this case at all, where you're in a finite horizon, but I think it's important for you to understand that the, without this infinite horizon assumption here. You loose this function of stationarity in your policies. Okay? Yeah. Interesting. Okay. So, that all, I think is, you know, making our something that's obvious, but becomes obvious after someone points it out to you. So, the second thing that I want to talk about, I think, is a little bit more subtle. And, and this notion of utility of sequences. So, as we've been talking, Michael, we have been sort of. Implicitly discussing not just the rewards we get in a single state, but the rewards that we get through a sequences of states that we take. And so I just want to point out a little fact that that comes from that, and where that ends up leading us. And then we'll get to some nice little cute series of math. So. Here's what I want to point what utilities, what we mean by utilities sequences. It means we have some function I'm going to call U for utility over the state, the sequence, sorry. Of states that we're going to see let's call them, S0, S1, S2 and so on and so forth. Well, I think an assumption that we've been making even if we haven't been very explicit about it is that if we had two sequences of states. S0, S1, S2, dot dot dot. And a different sequence S0, then S1 prime and S2 prime, that is two sequences that might differ from S1 on, but all start in the same start state. Okay? If we have a utility for the first, and that utility happens to be greater than the utility for the second, then it also turns out that we believe. That the utility for S1, S2, dot dot dot, is greater than the utility for S1 prime, S2 prime dot dot dot. Alright so these are two different sequences, S one, the S's and the S prime's are two different sequences. Yes. And in the beginning we're comparing them with S0 stuck in front of both of them. And we're saying if I prefer the S0 followed by all the S's, to S0 followed by the S primes, then I have that same preference even with those S0s missing. Right, and so this is called stationarity of preferences. And, another way of saying it is. That if I prefer one sequence of states today over another sequence of states, then I prefer that sequence of states over the same sequence of states tomorrow. So isn't, isn't this just obvious? Because the whatever the rewards for those two cases, we're just adding the reward we get for S0. So. It's going to be the same. But listen to what you just said. You just said, well, it'll be the same, because all we're doing is adding the reward for S0. But what did we ever say about adding up rewards? I thought, I thought that's what we were doing. That's right, that is what we were doing. But we never actually sat down and wrote that down and said, this is what it means. To talk about the utility of a sequence of states as opposed to the reward that you get in one state. Okay, so you're saying that if we, if we are adding rewards, then this follows. Right. Okay. And then I've actually been saying something even stronger, which is, I will show you on the next slide, which is if you believe that this is true, that the utility of one sequence of states is greater than the utility of another sequence of states. Both today and tomorrow. Then it actually forces you to do some variation of what you said which is just adding sequences of states. Or adding the rewards of the sequence of states that we see. That's really interesting. So then, so the adding isn't really an arbitrary thing it follows from this, this deeper assumption. Right, and the reason I bring this up is because. It would make sense if you were to just to grab someone off the street and start talking about Marco Decision Processes. One of two things will happened. Either they'd run screaming from you like you're a crazy person or they would sit and they would listen and if they listen they would just completely buy into the idea that you just add up sequences of rewards. You know, sequences of rewards that you see as a way of talking about how good the states are because that's a very natural thing to do. But it turns out that mathematicall if you have this notion. A sort of stationary of preferences and this sort of infinite arise in world. You really are in a case where this has to be true. And it has to be the case if you have to do some form of addition. Because nothing else sort of can be guaranteed to maintain this property over stationary preferences. I mean, as you said, if I got one

sequences of states and another sequnce of states and by just prepending or appending another set of states to it, I'm still going to always guarantee that one's greater than the other. You kind of have to do some form of adding the reward that you see in the states in both cases. because if you don't do that, then eventually this inequality will not hold. So, let me write that down in math terms. And see where that gets us, okay? Cool.

3.1.15 Sequences of Rewards 2

Alright, Michael. So, let's write that down in math, okay? You like math, right? I do. Okay. So here's the math version of it. I'm going to just say that the utility that we receive for visiting a sequence of states, S0, S1, S2 ellipsis, is simply the sum of all the rewards that we will receive for visiting those states. Sure. Does that make sense? Yeah. Is that consistent with what you were doing when you were thinking about the grid world? Yeah, exactly. But I thought we were going to make that not definitional, we were going to derive that it had to be that way. No, we're not. They can read about it. It'd take me, like, an hour and a half to do it. Alright. And I'm already losing my voice. But. What I do want people, I want you to believe is that the utility of the sequence of states, thinking of it as the sum of all the rewards, makes sense, right? And then if nothing else, at least it makes, it's consistent with what you've been doing as we've been talking about this grid work. Yeah, absolutely. I mean it, it also kind of makes me think about money. Go on. Well, so, so, I mean, that's sort of how money works. If we get some kind of payoff each day, those payoffs get added to each other. They don't get, you know, subtracted or multiplied or square rooted or whatever. They just, you know, they go into your bank account, and things add. So this feels like, kind of like that. It's like money in my pocket. Sure, if you have a big enough pocket. Okay, I'm with you on that. Alright, so I'm going to say that this all makes sense. It, it's really awesome and it sort of doesn't work. And to illustrate why this doesn't work, I'm going to give you a quiz. Yay. because you like quizzes. So I've been told.

3.1.16 Sequences of Rewards 3 Question

So, here's the quiz. You see on the screen this sort of unexplained two little squiggles with a bunch of numbers in front, on top of them or under them or near them? Yeah I, I assume that's a river and on one bank it's the land of the plus ones and the other bank has been infilitrated by plus twos. That's not what I intended at all but I actually like that enough that I'm going to pretend, that that's what I intended. So these work out to be sequences of states. Oh, I see now. Okay, and rewards that you receive. No, no, no, it's a riverbank and on one side of the bank, as you walk along it, you get a bunch of plus ones. On the other side you get a bunch of plus ones and some plus twos. And this just goes on forever, okay? And, I'm not going to tell you what all the rewards are after that except that they, they look similar to the rewards that you see here. Okay. Plus ones and plus twos, okay? And let's say the top, the top one, in fact, nothing but plus ones. And the bottom ones are some plus ones with maybe some plus twos sprinkled in and out, but those are really the only options. Okay? Yeah. Here's my question to you. So, would you rather be on the top side of the river bank, or the bottom side of the river bank? Okay? You think you know the answer already, don't you. Yeah. Okay, cool. Well then, let's, let's give our listeners a, a chance to come up with the right answer.

3.1.17 Sequences of Rewards 3 Solution

Okay Michael, I didn't give you as much time as I normally do because you think you already know the answer. So what's the answer? So, you know, I'd rather get the plus 2's occasionally, so I'll say the bottom one. No. Wait, what? You're not going to tell me the top one's better? no, it's not. Okay then that feels like a trick question. It's not a trick question. The answer is, neither one is better than the other. Oh, so I had to give neither? Or both. Which is better, and I can click on neither. Or you could click on both. Okay. So you thought that the bottom one was better, what was your reasoning for that? Because, sort of moment to moment, sometimes I'm getting plus one's, and that matches what I would've gotten if I had been on the other bank. But then sometimes I get plus two's, which is actually better than what I would have gotten on the other bank. And so, I'm, I'm, I never feel any regret being on the bottom bank. I only feel regret being on the top bank. That's fine. So, what would you say the utility of the sequence along the bottom actually is? 1 plus 1 plus 2 plus 1 plus 1 plus 2 plus dot, dot, dot. Which is equal to? infinity, I guess. That's right. What about the utility of the top one? 1 plus 1 plus 1 plus, that's also infinity. Yep. So the utility of both of those sequences is equal to infinity. Do you think one of them is bigger than the other? You drew the bottom one a larg, a little bit larger. I did. Or longer, anyway. Yeah. But, in the end, the sum of the rewards

that you get are both going to be infinity. So the truth is, neither one of them is better than the other. I still don't think you can say that both are better. You can't get around that. But yeah, I see, I see, neither, I can see neither is better. Or that both are better. Neither is better, both is better, whatever. The point is that, they're both equal to infinity. Hm. And the reason they're equal to infinity is because all we're doing is accumulating rewards. And, if we're always going to be able to get positive rewards no matter what we do, then it doesn't matter what we do. This is the existential dilemma of being immortal. Oh, living forever. Right. So if you live forever then, like why should you care about anything ever? Right I mean, everyone, every all the mortal people are going to die and one day they'll all be, you know, an infinite amount of time in your past. I could do this thing here, which is pleasurable, or I could do this thing right now, that, you know, will, is less pleasurable, but will eventually get me to a better place. But if I'm going to live forever, and I can always get to a better place, than it really doesn't matter what I do. It really doesn't matter. Mm. Because I'm just accumulating rewards, I'm living forever, and I'm going to, infinity is infinity and there's no really no way to compare them. Having said that, your original notion that, look, it feels like I should never regret having taken the second path compared to the first because I will occasionally do better. Seems like the right intuition to have. I see but it's just not built into this particular utility scheme. Right, but it turns out there's a very easy way we can build it into this utility scheme by just making one tiny little change. Would you like to see it? Yes. Beautiful, let's see it then

3.1.18 Sequences of Rewards 4

Okay Michael so here's the little trick. So all I've done is I have replaces the equation at the top with an alternate version of the equation. 'Kay. So it looks there's, you're now exponentially blowing up the reward. I am not exponentially blowing up the reward. Instead of what I've done. Is I've added of the see, the rewards that I'm going to see. For the states that I'm going to see. And I multiplied by gamma to the T. The gamma is between 0 and 1. I see. So... Do you? Well, kind of. So, so it doesn't exponentially blow up. It exponencially implodes. Right. So, so things that are like a thousand steps in the future If we multiply, if we take something that's less than 1 and we raise it to that power, it goes essentially to 0. So it's like it starts off the rewards are kind of substantial and then they get, they trail off quickly. Right. And in fact we can write down mathmatically what this is If you actually, if you stare at it long enough and you remembered your intermediate algebra or your calculus or wherever it is you learned this stuff. You would probably recognize this as a special kind of sequence or series. Do you remember what it is? Television series? [LAUGH]. No, but that's remarkably close. So let's see if we can bound this particular equation. So we know that this version of the equation eventually ends up being infinity if all the rewards are positive. Right? Mm, hm-mm. So what does this end up being even in the case where all the rewards are positive? Well, we can bound this from above by the largest reward that we will ever see, in the following way. So all I've then is said well I don't know what the rewards are but there is some maximum reward I'm going to call it Rmax. And if I pretended that I always got the Rmax reward. So long as that, as long as that's an actual number. A finite number. I know that this is bounded from above by this expression. Well what does this look like. Maybe this does look like a series you remember. Geometric series? Yes. This is the geometric series. And this is exactly equal to this. . And I assume that the summation causes the max to become lower case. Yes, yes it does. That's a, that's a trick that they don't really go over deeply in calculus, but it's true. Oh, that's cool. Alright so, the reward, the maximum reward divided by 1 minus gamma. So when gamma is really close to 0, you're just getting, oh you're getting just that one reward in the beginning and then everything falls off to nothing after that. Yep. And if gamma is really close to 1. You're dividing by something that's teeny tiny, which actually is like multiplying it by something that's really big. So it kind of magnifies the reward out, but it's not infinity until gamma get's all the way up to 1 and that's the same case we were in before. Right, so in fact you'll notice the way I wrote this, gamma has to be between 0 and 1, 0 inclusive but strictly less than 1. If I made it so that it could include 1, well that's actually the first case. Got it. Because 1 to a power is always 1, and so, this is actually a generalization of the sort of, infinite sum of rewards. This is called discounted rewards, or discounted series, or discounted sums, and it allows us, it turns out, to go an infinite distance in finite time. Wait. That makes sense. No. No. [LAUGH] An infinite distance? Yeah. That's at least the way I like to think about it. So, if we're discounted in this particular way, then that gives us a geometric series. And what it does is it allows us to add an infinite number of numbers, but it actually gives us a finite number. Cool. So this means we can still have our, and it turns out, by the way, just to be clear, that this world that we're in. Where we are in between 0 and 1, 0 inclusive is still consistent with our infinite horizons and our stationarity over preferences. So we can add things together and be consistent with those assumptions we were making before Or we can do discounted rewards and we're still going to be consistent with what we're doing before. But the difference

between this case and this case is that by doing the discount, we get to add infinite sequences and still get something finite. The intuition here is that Since, if gamma's less than 1, then eventually as you raise it to a power, it will basically become 0. I mean, that's why it's a geometric series. Which is like having a finite horizon. But that horizon is always the same finite distance away, no matter what, where you are in time. Which means it's effectively infinite. Or unbounded, or however you want to think about it. So it's like you take a step, but you're no closer than where, when you started. Right, so, what does that mean in, a world where you meant to say you're no closer than where you were trying to end up? So that means that even though, you've taken a step forward, the horizon sort of remains a fixed distance away from where you are. Mmm, mmhmm. That's what it means to To, to have this kind of this gamma value here and so you can still treat it as if it's always going to be infinite, even though it gives you a finite value, and that gives you your stationary. So does that make sense? I mean that's sort of the intuition. Yeah. You really are have an in, you're always in infinite, It's always in, infinity. But the gamma value means that at any given point in time, you only really have to think about what amounts to a finite distance. You never get closer to the horizon, even though you're always taking steps towards it. But there still is a horizon that you can see, and I can kill that analogy. [LAUGH] So can we, can you explain to me why it has this form, this R max over 1 minus gamma? I could. There's a You can kind of prove that, in a little cute math way, and I'm happy to take that diversion if you want. Yeah, I think so. I mean, unless, unless you were going to tell me something else. Well how about I tell you something else and then I take that diversion? Okay. Okay, so here's the something else. In the beginning I said, well, this is like going in infinite amount of distance in finite time, which you rebelled against. And my explanation is more about. How infinity looks finite which is not the same thing as going an infinite amount of distance in finite time. The reason I said that is beacuse of the sningularity. What? The singularity so, some of our listeners no doubt have heard of the singularity. You never heard of the singularity? The singularity is like when computers get so fast that you do infinite computation and, oh. Right. So the singularity, for those of you who don't know is is this sort of observation that has been made by many folks. That the sort of limit to computer power growing faster is the fact that it takes us amount, some amount of time to design the next generation of computers. Right, So Moore's law says everything doubles you know, everything 6, 18 months or whatever. But if we could design things faster, then we could actually make it double more quickly. So, one day we'll get to the point where the computer, which I will draw here, like that. The computer can actually design the next generation of computer. Well, when it designs the next generation of computer, that computer will also have the ability to design the next generation of computer. But it will be able to design it twice as fast. And then the next one will be able to design its next successor twice as fast, and so on and so forth. So this little time between generations, will keep Having every single time, which looks remarkably like a geometric sequence. And so once we reach this point, where the computer can actually design its successor, we will then be able to do an infinite number of successors. In finite time. [LAUGH] And that's when the world comes to and end. And that's what's called the singularity. Because we can't understand what happens after that point. So that's what I mean by going an infinite amount of distance in finite time. It just doesn't seem like distance. But, yeah, that's a weird example, for sure. I think it makes sense. [INAUDIBLE] infinite number of doublings in it. Yeah, [INAUDIBLE], no. [LAUGH] Well, we'll, we'll, we'll do we'll do some assignment where we allow people to decide whether this makes sense or one of your analogies make sense. And I'll just remind the listener that I'm the one who'll be assigning final grades. Okay. So with that let's go and do your little diversion. And and then we'll come back to. Doing a whole lot of math. So I actually think this little nice diversion will be warmup to all the math we're going to be doing. Okay? Oooh. Hm. More math. Hm.

3.1.19 Assumptions

So, if we think about the equation that we were doing before, which was you know, the sum of all these gammas times some R max. We can basically take out the R max and what we end up with is something that looks like that, right? You're summing together a bunch of gammas and then multiplying the result by R max. So what does that acutally look like? Well that looks like this. Gamma to the zero, plus gamma to the one, plus gamma squared, plus dot dot dot dot. Eventually multiplied by R max. Right? So let's call this x, okay? Does x include the R max or not? No. So we'll just call the little sequence of, it's going to turn out not to matter. Let's just call the sequence of gammas we're adding together, let's just call that x. Good. Well, if you look at it, this is actually recursive. Right? Because this is an infinite sequence, if you sort of shifted it one over in time, you would end up with just the repeated sequence again. All right? Which means that we can write x in terms of itself. x equals gamma zero plus gamma times X. I see, so it's shifted over, but then you have to multiply it again by gamma, to get up to gamma one, gamma two. Right, exactly. So, so,

this is just, you know, it's just math, that's just all it is. so, we can try to solve for x and figure out what x is, right? Cool. So, what is x? Well, we can subtract from both sides and so we end up with like, something like x minus gamma x equals gamma 0. Right? Seems like we can stop writing gamma 0. Isn't that just one? Yes, but I've already, I've gone too far, Michael. I've alrea, I've already written gamma 0. Alright. [LAUGH] so, this becomes x times 1 minus gamma equals gamma 0, or as Michael so astutely points out, 1. [LAUGH]. Alright. Which means what? It means that. So then we divide by 1 minus gamma. a is 1 over 1 minus gamma. Neat. And that, of course, we are going to multiply by R max. To get the formula that we had. That's, yeah, that's nifty. Yeah, there we go. So, why do we do this? Well, because you like stuff like this and it points out something very simple, which is that geometry is easy. [LAUGH] I don't think that's the kind of geometry that people usually think of. Well, they should be thinking of this kind of geometry. So geometry is easy. And by doing a little cute algebra, we can derive ridiculous equations that turn out to help us deal with go an infinite distance in finite time. [LAUGH] I don't think that's what they're doing, but okay. [LAUGH] So, let's think of this as warmup for what I actually wanted to show you, which is going to turn out to be a whole lot of math. Okay? Alright, let's, let's go for it. Alright, let's do that.

3.1.20 Policies 1

Okay, so Michael, in the spirit of what we just went through in deriving the geometric series, I'm now going to write down a bunch of math. And what I'm going to do is I'm just going to say it at you, and you're going to interrupt me if it doesn't make sense. Okay? That makes sense. It does. Okay, so here's the first thing I'm going to write down. I've decided that by going through this excercise of Of utilities in this kind of reward, we can now write down what the optimal policy is. The optimal policy, which as you recall is simply pi star, is simply the one that maximizes our long-term expected reward. Which looks like what? Well, it looks like this. There, does that make sense? Let me think, so. We have an expected value of the sum, of the discounted rewards, at time t. And, given, pi. Meaning that we're going to be following pi? Mm-hm. So these are the, the sequence of states we're going to see in a world where we follow pi. And it's an expectation because, things are non-deterministic. Or may be non-deterministic. And do we know which state we started? It doesn't matter, it's whatever s zero is. I see. Whatever s zero is, but isn't that random? I mean s one and s two, s three; those are all random. Well, we start at some state, it doesn't matter, so t is starting out a zero. And going to infinity. Okay? So does this make sense? Yes, so then, so we're saying, I would like to know the policy that maximizes the value of that expression. So it gives us the highest expected reward. Yeah, that's the kind of policy I would want. Exactly. So, good, we're done, we know what the optimum policy is. Except that it's not really clear what to do with this. All we've really done is written down what we knew it was we were trying to solve. But it turns out that we've defined utility in such a way that it's going to help us to solve this. So let me write that down as well. I'm going to say that the utility of a particular seque, of a particular state okay. Well it's going to depend upon the policy that were following. So I'm going to rewrite the utility that takes the superscript pie. And thats simply going to be the expected set of states that I'm going to see from that point on given that I've followed the policy. There, does that make sense? It feels like the same thing. I guess the difference now is that you're saying the utility of the policy out of state is what happens if we start running from that state. Yep. And we follow that policy. Got it. Right. So, this answers the question you asked me before about, well, what's S0? Well, we talk about that in terms of the utility of the state. So how good is it to be in some state? Well, it's exactly as good to be in that state as what we will expect to see from that point on. Given that we're a following a specific policy where we started in that state. Hm,. Does that make sense? Kay. Yeah. Very important point here, Michael, is that the reward for entering a state is not the same thing as the utility for that state. Right? And in particular. What reward gives us is immediate gratification or immediate feedback. Okay? But utility gives us long term feedback. Does that make sense? So when reward [UNKNOWN] is the actual value that we get for being in that state. Utility [UNKNOWN] state is both the reward we get for that state. But also, all the reward that we're going to get from that point on. I see. So yeah. That seems like a really important difference. Like, if I say, here's a dollar. You know? Would you poke the president of your university in the eye? You'd be, like, okay. The immediate reward for that is one. But the long term utility of that could be actually quite low. Right. On the other hand, I say, well, why don't you go to college? And you say, but that's going to cost me $40.000. Or better yet, why don't you get a masters degree in computer science from Georgia tech, bu you can say that's going to cost me $6600. Yes, but at the end of it you will have a degree. And by the way it turns out the average starting salary for people who are getting a masters degree or undergraduate degree is about $45000. So is it considered product placement if you. Plug your own product within the product itself? No, I'm just simply stating fact Michael. This is all I'm doing. Just facts. Alright. This is called fact placement. Alright. The point is, there's

a, an immediate negative reward, of say, $6,600 for, I'm going through a degree. Or maybe it's $10,000 by the time, the 15th person sees this. But anyway, it's some cost. But, presumably it's okay to go to college, or go to grad school, or whatever. Because at the end of it you are going to get something positive out of it. So it is not just that it prevents you from taking short term positive things if that is going to lead to long term negative things. I also always you to take short term negatives if it will lead to long term positives. That makes sense. What this does is this gets us back to what I mentioned earlier. Which is this notion of delayed reward. So we have this notion of reward, but utilities are really about accounting for all delayed rewards. And if you think about that, I think you can begin to see how, given you have a mathematical expression delayed rewards, you will be able to start dealing with the credit assignment problem. Cool. Okay, so let's keep going and write more equations.

3.1.21 Policies 2

So, now that we've got utility fine, and we've got this pi star to fine, we can actually do an even better job of writing out pi star. And let me do that. All right, so does this equation make sense, Michael? Let's see, so the policy, is that a star again, or is that a K. That's a star. So it's the optimal policy. All right. The optimal action to take at a state is, well, look over all the actions, and sum up overall the next states, the transition probability, so that's the probability we end up in state s prime. And now we have the utility of s prime, the problem being that that's not defined. Well, it sort of is, we defined it immediately above, at least with respect to some policy. But that's concerning because we don't know. The policy that you want to put in there is gotta be the policy that you're trying to find. Right, so in fact implicitly what I mean here is pi star. So, in fact, let me write that down that whenever you see me write from now on, the utility of a state, I'm almost always going to actually mean the utility of the state if I follow the optimal policy. We might call this the true utility of the state. I see. So I'm just going to write this off to the side here as something for you to remember. So this this says then that the optimal policy is the one that, for every state, returns the action that maximizes my expected utility. With regard to the optimal policy, it feels rather circular. It is rather circular, but you're a computationalist. You're a big fan of recursion. We just went through a whole exercise where we figured out the geometric series by effectively doing recursion. It's a similar kind of situation, for this? It kind of is. So, let me write one more equation down and then you'll be one step closer to actually seeing it. Of course, if we're in an infinite horizon with a discounted state, even though you're one step closer you won't actually be any closer. Well let's worry about that when we get there. So let me write one more equation down. We're never going to get there. It's infinitely long. [LAUGH] Yeah. Wait are you demonstrating something with this lesson by making it infinitely long? [LAUGH] I'm certainly demonstrating something with this lesson. I don't know what it is. So let me write this next equation down. So then the true utility of a state s is then, I'm just basically going to unroll the equation for utility. It's the reward that I get for being in that state, plus I'm now going to discount all of the reward that I'm going to get from that point on. Got it? All right, so once we go to our new state s prime, we're going to look at the utility of that state. Okay, that's sort of fine, modular recursion. We're going to look at overall actions, which action gives us the highest value of that. Oh I see, that's kind of like the pi star expression just above. Yup. All right, so once we figure that out, we know what action we're going to take in state s prime. We're going to discount that, because why? Because I guess that just kind of ups the gamma factor on all the rewards in the future. Right. And then we're going to add to that our immediate reward. Yes, okay I think I follow that. In some sense all I've done is I kept substituting pieces back into one another. So the true utility being in a state is the reward you get in that state, plus the discount of all the reward you're going to get at that point, which, of course, is defined as the utility you're going to get for the states that you see, but each one of those is defined similarly. And so the utility you will get for s double prime say will also be further discounted but since it's multiplied by gamma that will be gamma squared and then s triple prime will be gamma cubed, and so that's basically just unrolling this notion of utility up here. Okay so now it seems like all the pieces are in one place. Right. And so it would be nice if we were done. And I'm going to say that we're not just one step closer, but you can see an oncoming light and it is not an oncoming train, okay. So Yeah, this seems like a really important equation. It is, in fact, it's so important, it's got a name. You want to guess what the name is? Bill That's actually very close. It's Bellman Equation. Bellman equation Esquire. This equation was invented by a guy named Bellman, and it turns out to be in some sense the key equation for solving MDPs and reinforcement learning. Wow. And it's actually even more [INAUDIBLE] than it looks. But basically, this is the fundamental recursive equation that defines the true value of being in some particular state. And it accounts for everything that we care about in the MDP. The utilities themselves deal with the policy that we want to have, the gammas are discount, and all the rewards are here. The transition matrix is here, and the actions or all the actions we're going to take. So

basically the whole MDP is referenced inside of here and allows us by determining utilities, to always know what's the best action to take. What's the one that's going to maximize the utility? So if we can figure out the answer to this equation, the utilities of all the states, we per force know what the optimal policy is. It becomes very easy. So we've sort of taken all that neat stuff about NDPs and stuck it in a single equation. Bellman was a very smart guy. So was he the same Bellman from the curse of dimensionality? Yes. Cool. There can be only one Bellman. [LAUGH] Actually, are there any more Bellmans? I don't think so, I think that they retired, like retiring a jersey. They retired his name. I could've sworn that I saw one at the last hotel that I went to. It was probably the same one. Oh, I get it. Hotel, Bellman, that's really good. Very good. Okay good. Well, so now that we've killed that as much as we could, let's see if we can actually solve this equation, which since this is clearly the key equation since it has a name, okay? Yeah, that would be cool. Especially because it looks like, if you could solve this, you could solve it, right? because then you have u. You could just plug the u in and get the u out. Right. And once you have the u in, and you get the u out, then you got the policy. Right. For u. It's always been for u. [LAUGH] It's for us, Michael, it's for us.

3.1.22 Finding Policies 1

Okay Micheal, so I've erased the screen and kept Bellman's equation, the most important equation ever and we are going to solve it. Oh. So, how are we going to make that work? Okay, so how many, so we wrote this down as a utility of s, how many S's are there? s, n, m, I don't know? Pick a letter. N. N, so we N states which means this isn't really one equasion. This is how many equations? N. Yes it's N equations. How many unknowns are there? Well we have U, the Rs are known, the Ts are known, the only things that's missing is the Us. And there's, oh and there's N of those as well. Right there's N equation in N [UNKNOWN]. So we're done. Yes because we know how to solve n equations in n unknowns, right? If the equations were linear. If the equations were linear. Are these equations linear? I'm going to go with no. Why not? because the max is problematic. That's right. This max operation makes it nonlinear. So, it looks really good for a moment there. We've got n equations, n unknowns. We know how to solve that. But max makes for a very very very weird non linearity. And there's kind of no nice way to deal with this. Actually, one day, Michael, if you ask me, there is a cute little aside you can do where you can turn max into something that's differentiable. Oh. But. That doesn't actually help us here so I'm not going to go off on that aside yet. But even differentiable wouldn't quite be linear. That's right. And it wouldn't help us in this case. Yeah that's exactly right. But the fact that you can have differentiable maxes I think actually [UNKNOWN]. But also unimportant for what we're talking about now. So, we've got n equations, n unknowns, they're nonlinear, which means we can't solve it the way we want to by like inverting matrices or like. You people are in the regression would normally do but it turns out that we can in fact, do something fairly similar, something that actually allows us to solve this even though it is nonlinear. And here's the equation, or the equation, here's, here's the algorithm you use that sort of works, okay? So it's really simple. Just simply start with some arbitrary utilities. Declare those the answer and you're done. That would be one way of doing it but it turns out we can do even better. Wait. Start off with the correct utilities. That would work except we don't know what they are. Oh right. So we're going to start with some arbitrary utilities, but then we're going to update them based on their neighbors. And then we'll simply lather, rinse, repeat. Alright, so what does that mean based on neighbors? So, it means, based on the states, you're going to update the utility for a state based on all of the states that it can reach. So, let me write down an equation that tells you how to update them and then maybe it'll be clear, okay? Yep. So we're going to have a bunch of utilities since we're going to, we're going to be looping through this. And let's just say every time we loop through, that's time t. Okay? So we know what the equation for utility is where the utilities are. It's just the equation that's written up here. So it's r of s plus gamma times the mass over a of. The expected utility. Right? Except we have some estimate of the utility at time t. Okay? and, probably the right thing for me to do would be to write this like as you had or something like that. This is my estimate of the utility. And so now using this, I'm going to want to sort of update the, you know all of my utilities to make them better. And it turns out I can do that by simply doing this. So I'm going to update at every iteration update utilities based on neighbors I'm going to update at every iteration, at every iteration my estimate of the utility of some state S by simply recalculating it to be the actual reward that I get for entering state S, plus the discounted utility that I expect given the original estimates of my utility. Does that make any sense at all? Yeah, I guess so, so. So the s, okay, so, so we need the whole u hat t. Mm-hm. Like at all states, because we're not just using the values that state s to update the values that state s. We're using all the values, at the, at the previous time step to update all the values to the current time step. Yep. So, so all these n equations, they're all tangled together. Right, because of this This, expectation. So really, just to make it clear, this should be a summation over s prime.

3.1.23 Finding Policies 2

So, I'm going to update the utility of s by looking at the utilities of all the other states, including itself, as prime. And weight those based on my probability of getting there given that I took an action. And what action am I going to take? Well, I'm going to take the one that maximizes my, expected utility. So it's sort of like figuring out what to do in a state assuming that this uhat really is the right answer for the future. Right. Now why is that going to work? So I just made up the uhats, right? They started out as arbitary utilities. The next little step about updating utilities based on neighbors makes some sense because effectively is any state that you can reach. Which is determined by the transition function. But, all I'm doing is reaching states that are also made up of arbitrary utilities so, arbitrary values. Why should that help me? Well, it's because of this value right here. This is actual truth. The actual word I get for entering into a stage, is a true value. So, effectively what's going to happen is I'm going to be propagating out the true value, or true reward, the true immediate reward that I get for entering into a state, say. Through all of the states that I'm going to see and propagating that information back and forth. Until ultimately I converge. Still not obvious why we should converge, because we start off with an arbitrary function and it seems like that could be really wrong. So we're like adding truth to wrong. Yes but then, the next time around I'm going to, I've been adding truth twice to wrong, and then more truth to wrong, and more truth to wrong. And eventually, I'm going to be adding, more and more and more and more and more and more and more and more and more and more truth, then it will overwhelm the original wrong. And is that, does it help that the wrong is being discounted? Yes, it helps that the wrong is being discounted. Does it help that the wrong is being discounted? I actually don't know that matters. I'll have to think about that for a moment. It certainly doesn't hurt. But I, I guess the way that I think about this, I mean there's an actual proof you can look it up but in the end I think, I, I tend to think of this as a kind of simple contraction proof that effectively at every step you have to be moving towards the answer. Because you've added in more of the reality, more of the truth. And if you remember the utility of a state is just all of the rewards that you're going to see. So, basically at every time step you have added the reward that you're going to see, and then all of the rewards you're going to see after that. And so you've gotten a better estimate of actually the sequence of rewards you're likely to see from the state. And if that gets better for any of the states, then eventually that betterness will propagate out to all the other states that they can reach or can reach them. That will keep happening and you'll keep getting closer and closer and closer for the true utility of the states until you eventually run out of closeness. Cool. Does that make sense at all? Well does it also help that the gamma is less than one. Yeah it does. the, the way I like to think of this as, is as a sort of contraction proof, that makes, if you've heard of those. So, the basic idea here is that you start out with some noise, but at every step, you're getting some truth, and that truth gets added, and then, the next dideration, more truth gets added, and more truth get, gets added. So, as you have some estimate of some particular state S, you get to update it based on truth. It's actual reward. And you bring in more truth from the other utilities, as well. As this particular utility gets better. That closeness to the true utility then gets spread to all, from, to all the states that can reach it. And because the gamma is less than one. You basically get to overwhelm the past in this case which is the original arbitrary utilities. And so as you keep iterating through this, the latest truth becomes more important than the past less truth. And so you are always getting closer and closer and closer to the truth until you eventually you do. At least that's the intuition that I like to think of. Yeah I, I kind of get that as an intuition, though I'd probably be happier going through the math, but. Well, we could do that, and by we, I mean the students can do that by actually reading the proof. All right. Okay, so cool. So, this right here is a an easy way to find the true value of states. And you do it by iterating and it kind of has a name. It kind of has a name. Yeah, what do you think the name could be? Bellman. Bellman's algorithm. No. No though that's probably reasonable. Utility iteration. Yes, except utility sounds better if you say value, so it's value iteration. And it works. Remarkably well. So, and it doesn't, doesn't give you the answer but it gets you something that is closer and closer to the answer. Right. And, eventually it will converge. You'll get so close to converging it doesn't matter. And, once you have the two utilities, if you recall. We know how to define the optimal policy in terms of utilities. So, if I give you the utilities, then the optimum policy is just, well I'm in a state. Look at all the states I might get to. Figure out the expectation that I'm going to get for a given action. Pick whichever one is maximum and I'm done. So solving for the utilities are the true value of a state. Is effectively the same thing as solving for the optimal policy. Hm. Excellent. That's cool. I think so.

3.1.24 Finding Policies 3 Question

Okay, Michael, so you seemed like you knew what you were doing so I thought I would verify that by giving you a very easy quiz. This doesn't look so easy. It is easy, so here's the quiz. To help you I've written

up both the Bellmans equation and the update that we would give to utilities. I neglected to write the little hats and everything because I just don't think you need that but these are the two equations that you need to be able to think about. This is just what we've written up before. And I've written down the grid world we've been playing with the entire time. And I want you to figure out how value iteration would work from the first iteration and the second iteration for this particular state here that I marked with an X, okay? Okay. Alright, and there are a little more information you need to go. Gamma in this case is going to be equal to one half. Mainly because it's easy to do the math if you do it that way. The rewards, just to remind you, for all of the states except for the two goal or absorbing states is going to be minus 0.04. And my initial arbitrary utilities for all of the states, is going to be 0, except in the two absorbing states where I already know the utilities are 1 and minus 1 respectively, Okay? Okay. You got all that? I think so. You sure? [LAUGH] I feel confident that I'm going to slip up, but Okay, yeah, I think I can take a stab at this. Alright, so gamma's one half, rewards are minus 0.04, arbitrary starting utilities at times 0 is 0, except here at the absorbing states. Tell me how the utility here will evolve after one step, or one iteration, and two steps, or two iterations. Okay then, go.

3.1.25 Finding Policies 3 Solution

Alright, Michael. What's the answer? I think I've already made a mistake. Okay. What's the mistake? Suggesting that we do a quiz. [LAUGH] I'm pretty sure that's always true. Unless Michael, by doing a quiz now and suffering pain, you, in the future, are a better person. In which case, you have made the right long term decision. I see. You're, this is a MDP metapoint you're making. Mm-hm. Alright. So let, but let's, let's just buckle down and do this thing. So. Okay. Alright, so at that state x, we have to consider, according to the equation, we're going to do U sub one at x. And that's going to be the reward at x, which is minus .04. Mm-hm. Plus gamma, which is a half. Yes. Feel free to write these things down. Okay, so, let's see. It's going to be minus .04 plus one half times. Alright. So now we need to do four different actions. Right. So, I don, I would make like a brace-y thing at this point. Not a bracket, but a brace. Alright? Or I could do a bracket, because you're going to notice immediately that it's obvious what the right action is. Okay, alright. Well, we know that the, the right action's going to be to go to the right. Yeah, but even then, you know you don't have to do the rest of the computation because my first guess at all the utilities is that they're 0. Which means you're always going to want to take the action that gets you to plus one with the highest chance. Right. So there's just no point in. I see. Okay. Fair enough. Thank you for. Okay. The shortcut. So, so we only have to do the one action which is to the right. Mm-hm. And so if we go to the right, there's three possible next states we could go in. Yeah. One is back to x, which has a value of zero. Mm-hm. One is to the thing underneath of x, which has a value of zero. And then the last one with probability 0.8 needs to go to the plus one. Which is. 0.8 times plus one. Which is 0.8. 0.8. Okay. And that is? Okay. So 0.8 times a half is 0.4 minus 0.04 is 0.36. Yep. And that is correct. Okay, so to do the same thing for you too, we're going to need the U1 value for a bunch of other states, seems to me. Maybe, so let's right that down. So we know for now the utility here is .36 right? Yeah. And you're saying that in order to do U two, I'm now going to have to, you know, I was able to avoid doing some of the math before because, these are all zeroes. So it was just easy to do. But when I went right, I either stayed where I was, went to the plus one, or I ended up going down. Well I think by the same argument that allowed us to cheat our way out of to cheat our way out before. Okay, It's still going to be best to go to the right, so. Yeah but I know that but the value itself is going to depend on the value in several other states. Yeah well how many other states? Oh, just that one. Just this one, so what was U one of this state? I see. So, presumably, we want to avoid falling into the pit, so the, the best thing we can do is bash our head against the wall, Mm-hm Which will get us a -.04 for that statement. Right. Okay, alright, so maybe this isn't so, so bad. Mm-hm. So now for our u2 values we need -.04 plus a half times point, point one times point 36. Plus. Mm-hm. 0.1 times negative 0.04, so that's minus 0.004, plus 0.8 times one. Oh, which is 0.8 again, just like it was last time. Yep. And I get 0.376. Which is what I get by also getting out your calculator. Okay, so 0.376, and you can imagine how we would do that on and on. I want to point something out, Michael, which is that. You decided to figure what the true utility was for the state under X by bashing your head into the wall. But you know that based on the discussion we had earlier that actually the optimal policy would involve going up instead of bashing your head into the wall. What you did at that point was in fact right because everything else in this utilities were all zero. The best thing you could do is avoid getting, avoid ever falling into minus one, so the policy of the very first of bashing your head into the wall, is infact the right thing to do at that point. But what youll notice is that next time around the utility for the X state Is bigger than zero. In fact, will keep getting bigger and bigger than zero. As you can see, it went from .36 to 0.376. Which means that at some point it's going to be worthwhile to try to go up instead of bashing your head into a wall. I see. So this, so this is kind of cool that

it works. But, it does seem like a really roundabout way of getting there. I mean, is there some way that we could some, I don't know, maybe take advantage of the fact that there's not that many policies? Yeah, so you actually said something, fairly subtle there, so let's see if we can unpack it. So, lemme point out two things, which I think will get us to what you're answering. The first is. Do you realize that the reason that the value iteration works is because eventually value propagates out from its neighbors, right? The first time we can calculate the .36 without really worrying about anything around it because the utilities are all zero, and in fact, based on the initial utilities. Even for this state here, we do the wrong thing. But after some time, this state becomes a truer representation of its utility and, in fact, gets higher and higher. The right thing to do here will go up. You'll also notice that after another time step I'm going to need to start looking at the value of this state as well. Alright. So, eventually I'm going to have to figure out the value, the utilities or the values of all of these states and this plus one is going to propagate out towards the other states. Where this minus one will propagate out less because you're going to try to avoid falling in there. So that makes sense, right? But what's propagating out, Michael? What's propagating out is the true utilities, the true values of these states. But what's a policy? A policy is a function from what to what? States to actions. Right, a policy is the mapping from state to action. It is not a mapping from states to utilities. No, that's what U is. That's what U is. Or that's what you are. So, [LAUGH], so if we have U, we can figure out pi, but U is actually much more information than we need to figure out pi. If we have a U that is not the correct utility, but say, has the ordering of the actions correct, then we're actually doing pretty well, right? It doesn't matter whether we have the wrong utilities. I see. Uh-huh. You'll remember we did this in the, the first third of the class as well when we noticed that we were computing in the Bayesian learning case actual probabilities. But we don't really care about actual probabilities. We just care that the labels are right. There's a very similar argument here. We don't care about having the correct utilities, even though by having the correct utilities, we have the right policy. All we actually care about is getting the right policy. And order matters there rather than absolute value. Does that make sense? Yeah, that's interesting. It's almost kind of like pi is more of like a classifier, right? It's mapping. Inputs to discreet classes and the user kind of like, more like regression where its mapping these states to continuous values. Right and given one, given the utilities, we can find pi, given pi there's an infinite number of utilities that are. Consistent with it. So, what you end up wanting to do is get a utility that's good enough to get you to your pie. Which is one reason why you don't have to worry about getting the absolute convergence in [UNKNOWN]. But it gives us a hint of something we might do that's a little bit better with the policies that might go faster in practise. So, I'm just going to take three seconds to give you an example of that. Okay? Awesome.

3.1.26 Finding Policies 4

Okay so Michael let's see if we can do something that might be faster and just as good as what we've been doing with value iterations. And what we're going to do is we're going to emphasize the fact that we care about policies, not values. Now it's true given the true utilities we can find a policy, but maybe we don't have to find the true utilities. In order to find the optimal policy. So, here's a little sketch of an algorithm okay, so it's going to look a lot like value to ratio. We are going to start with some initial policy, let's call it pi sub zero and that's just going to be a guess, so it's just going to be an arbitrary setting of actions that you should take in different states. Then we're going to evaluate how good that policy is, and the way we're going to it at time T. Is to calculate its utility, which I'm going to call U sub t. Which is equal to the utility you get by following that policy. Okay? And I, I'll show you in a moment exactly how you do that, alright? But I just want to make certain that you, that you, you kind of buy that maybe we can do that. So, We have, given a policy, we ought to be able to evaluate it by figuring out what the utility of that policy is. And, again, we'll talk about that in a second. And then, after we know what the utility of that policy is, we're actually going to improve that policy in a way similar to what we did with value iteration, we're going to update our policy, Time T plus one, to be the policy. That takes the actions that maximizes the expected utility based of what we just calculated for [INAUDIBLE] of T. Now notice it will allow us to change Pi over time because image that we discover that in some state that actually there is an action that is very good in that state. That gets you some place really nice gives you a really big reward and that went on and you do fairly well. Well then all other states that can reach that state. Might end up taking a different action than they did before, because now the best action would be to move towards that state. So these two steps will actually, or can actually lead to some improvement of the policy over time. But the key thing here is we have to figure out exactly how to compute u sub t. Well, the good news is we know how to do that, and it's actually pretty easy. And it boils down to our favorite equation, Doman's equation. [LAUGH] So our utility at time t, that is the utility that we get by following a policy at time t, is just well, the true reward that we're going to get by entering that state plus gamma times the expected utility, which now looks like this. There,

does that make sense? Do you see that? Okay, hang on, it looks a little different from the other equation. So did you mean for it to have T UT is defined in terms of UT and not UT minus one? Yes. Okay, that's interesting. And the max is gone but instead of max, there's a policy. Stuck into the transition function. Yep. A choice of action is determined by the policy. Right. And that's actually the only difference between what we were doing before is that rather than having this max over actions, we already know what action we're going to take. It's determined by the policy we're currently following. Okay, but isn't this just as hard as solving? The thing with the max, you said? Well, what was the problem that we were solving before with the max? That was the Bellman equation. Yes, but we were solving a bunch of equations. How many of them? N. So we were solving n equations, and how many unknowns? N. What's the difference between this, n equations and n unknowns, and the other n equations and n unknowns? Well, n is the same. Mm hm. There's no max, though. There's no max. And what was it that made solving that hard before? It made it, the max made it non linear. The max is gone now. You're saying this is, this is a set of linear equations? Yeah. Because, well, there's, there's just a bunch of sums. And the pi is not like some weird function. This is just effectively a constant. I see. So now, I have n equations, and n unknowns. But it's in linear equations. And now that I have in linear equations and unknowns, I can actually compute this is a reasonable amount of time, by doing matrix inversions, and regression, and other magic hand wavey things. That's very slick. Yeah. It's seems, it's still more expensive than doing the valued [UNKNOWN], I guess. Yeah, but you don't have to, perhaps, do as many iterations as you were doing before. So once you've evaluated it, which we now know how to do, and you've improved it, you just keep doing that until your policy doesn't change. Very cool. Mmhm. And this does look a lot like value iteration to you, doesn't it? Yeah, though is seems like it's making bigger jumps somehow. It is, and that's because instead of making jumps. In value space, it's making jumps in policy space. Which is why we call this class of algorithms Policy Iteration. Cool. Right. Now, this inversion can still be fairly painful, it's, you know, if we don't worry about being highly efficient, you know, it's roughly n cubed, and if there are a lot of states, this can be kind of painful. But it turns out there's little tricks you can do, like do a little step evaluate iteration here for a while to get an estimate and then, you know, kind of cycle through. So there's all kinds of clever things you might want to do, but at a high level without worrying about, you know, the, the details of constants, this general process of moving through policy space. And taking advantage of the fact that by picking a specific policy you're able to turn your nonlinear equations into linear equations turns out to often to be very helpful. So, is it guaranteed to converge? Yes. Nice. Well there, that was easy. I'm, I'm not going to go into it but, you know, there's a finite number of policies. You're always getting better so eventually you have to converge. It's very similar to the, or at least intuitively it's very similar to the argument you might make for [UNKNOWN]. Cool.

3.1.27 Wrapping up

Okay Michael, so, this is it. Why don't you help me remember what we learned in this one marathon session that was not all distributed over multiple months. [LAUGH] Sure. Or years. Mm hm. So, MDPs. So, we talked about mark-off decision processes, and we said what that meant. [LAUGH] Okay, so, that's the first thing we did is we talked about MDPs. And that, that MDP consists of states and rewards and actions and like that, transitions, discounts. So you said discounts and I just want to point out that there are some people who think that that's a part of the definition of the problem and there are some people who think that that's more of a parameter to your algorithm. okay, alright. So there's some people who think of it the right way. Like me. And some people think of it the wrong way. So, I tend to think of the discount as being something that you're allowed to fiddle with, as opposed to it being a fundamental part of the. Then why not fiddle with the rewards? Sure. You are allowed to fiddle with the rewards. Oh! You are very open minded! I am open minded, i believe that rewards should be able to marry other rewards. In any case, the important thing here is that there is some under lying process that you care about that is suppose to represent the world, states, rewards, actions, and transitions and capture that fairly well. Discount is, in some sense, an important part of the problem, because it tells you how much you want to care about the future versus the past. But it's reasonable to think of that as something that you might want to change outside of the kind of underlying physics of the world. I see. Okay, that's fair. Yeah, in some sense, the states and the actions and the transitions represent. The physical world and the rewards and the discount represent the kind of task description. Right. And of course, you say that, but if you, you could decide to define states differently, and doing that would impact both your actions and the transition function and so on and so forth. But the basic idea is right, which is, there's some underlying process we're trying to capture, and I think it's exactly right to say, states, actions, and transitions. Sort of capture that. And rewards and discounts capture more about the nature of the task, you're trying to do in that underlying world. Okay. And in, in that context we talked about two really important concepts. Policies and, value functions? Mm-

hm. Which we sometimes call utilities. Right. And how do utilities differ from rewards? The utilities factor in the long term aspects, and the rewards are just telling you the moment to moment. Right. Utilities are like a group of rewards. Like a gaggle. Or a murder. Of crows. So, that we talked about how we can assign value to an infinite sequence of rewards. Mm-hm. But it helps if we use discounting to do that so that we don't get infinitely large sums. And that allowed us to deal with infinite sequences, but threat them as if their value is, in fact, finite, thus solving the immortality problem. [LAUGH] And, let's see, then we kind of. And the stationarity was a really important part of that. Yep. In fact kind of drove everything. Yeah. And all these things were tied up together in the bellman equation itself. Yes which is an awesome equation and deserves capital letters. [LAUGH] Is that it? And then, well then we solve the Bellman equation using value iteration and policy iteration. Oh, yes. I think that was it. Alright, so are any of these polynomial time algorithms? Well, the way we've been talking about them, no, but you can map these into linear programs and turn them into polynomial problems, or polynomial. So, yes these problems can be solved that way. But actually, that reminds me, of something that we haven't said and that we haven't learned today, that I think is worth mentioning. Which is we been talking about, you know, this section of the class is about over the course is about reinforcement learning. But, we actually haven't done any reinforcement learning here because we know everything, we know the states, we know the rewards, we know the actions, we know the transitions. We have some discount, we have been solving MDP's, but that's not quite the same thing [SOUND] as doing reinforcement learning, however, it's very important to do these things, to make it easier to think about how reinforcement learning works. So, in reinforcement learning, the difference is, you don't necessarily know, the rewards and the transitions or even the states and the actions, for that matter. I see. So when are we going to get into reinforcement learning then? Next time. And when I say we next time, I mean you next time. That's your assignment. Learn all about reinforcement learning and talk about it next time. All right. I gotta go and get to work then. Okay. Well, you have a good one Michael. Thanks for all this. It's really cool. It is cool. Bye. Bye. Bye.

3.1.28 Reinforcement Learning

Hello Charles. Hi Michael. How are you doing? I'm doing okay. Good. It's you know, exciting to be getting to talk about reinforcement learning. Mm. Reinforcement learning. It's my favorite type of learning. Is that true? It is true. Wow. I like machine learning. I like machine learning too. But of all the kinds of machine learning, reinforcement learning is my favorite kind of learning. So, let's, how bout, how bout, giving the opportunity for the students in the class to learn about reinforcement learning by having us tell them about it. Oh, let's do that. You first. Alright. We're going to build up on the stuff you told us about last time. You mean the fantastic, well defined, and well formalized stuff that we talked about last time? Yes, it was fantastic and it laid the groundwork for what I would like to talk about. So you set up Markov decision processes, and I'm going to talk about what it means to learn in that context. Excellent. I find it useful to start off by thinking about a reinforcement learning API, like application programmer interface. So what you talked about is, is this box here. The idea of being able to take a model of an MDP, which consists of a transition function, T, and a reward function R. And it goes through some code and, you know, it comes out and a policy comes out. Right? And a policy is, is like pi. It maps states to actions. And that whole activity, what would you call that? What would you call the, the, the problem of, or the approach of taking models and turning them into policies? Maybe I'd call them planning. Yeah that's what I, that's, that's what I was hoping you would say. Mmhm. Alright, now we're going to talk about a different set up here. We're still interested in spitting out policies, right? Figuring out how to behave to maximize reward. But a learner's going to do something different. Instead of taking a model as input, it's going to take transitions. It's going to take samples of, being in some state, taking some action, observing a reward and observing the state that is at the other end of that transition. Alright? And we'll, I put a little star on that to say well we're going to see a bunch of these transitions. Mm. And using that information we're going to instead of computing a policy, we're going to learn a policy. I see. So we call that learning? Yeah, or even reinforcement learning. Mm. By the way, what makes it reinforcement learning? That's a question. [LAUGH]. It's not a good question, but it's a question. I was, I was going to say good question, but I'll, well maybe, maybe there, it's not that it was a bad question, it's that I don't have a particularly good answer for it. So, maybe we need another slide to discuss that. Okay.

3.1.29 Rat Dinosaurs

All right, so let's do a brief aside about the [LAUGH], this is like a bad history of reinforcement learning so it's not that it's a bad history, it's just badly told. So but the basic idea is this, once upon a time and

we're going to call it, say the 1930s people observed. That if you put an animal, this is supposed to be a rat, in, in a, you know, box, say. And give it choices of looking in place B and place A and, say, one of them has cheese in it and the other one doesn't, but it can't see which because, you know, the doors are closed. And and let's say that we, we consistently do something like this. We turn on a red light whenever the cheese is in the A place. And we turn on a blue light whenever the cheese is in the, B place. And what you observe is if you do this consistently, then what you find is that if the animal sees the stimulus like the red light going on and it takes an action like peeking inside the, the or sticking it's head inside the box A and it gets a reward which in this case would be getting to eat some cheese. That that set up strengthens its response to the stimulus. In other words, in the future when it sees the red light it's going to be more likely to go in and look in box A. And this notion of strengthening, you can think of it as reinforcing the connection. Reinforcing just means strengthening. Hm. So, this was studied for a long time and there's, there's tremendous amount of interesting research about this, but if you start to, you know, think about it as a computer scientist. A natural way of, of kind of thinking about what this problem is, is that you know, a stimulus is kind of like a state, and an action is kind of like an action, and a reward, well, I kind of stole all these words but it's kind of like a reward. And it leads you to this idea that what you really want to do. Is figure out how to choose actions to maximize reward as a function of the state. And so we started to take this concept and we called it reinforcement learning because that's what the psychologists were calling it. But, for us it just means reward maximization. This notion of strengthening is really not part of the story for us at all. So, we're really taking this word and using it, you know, wrong. Hm. Well, that all makes sense except for one thing, which is the idea that this happened in the 1930s. Because we all know that rat dinosaurs did not exist in the 1930s. Yeah, they, well they went extinct in the 40s. Hm, hm. There's this one little epilogue to the story which I find really kind of amusing, and that is the computer scientists did pretty well at figuring out algorithms for solving problems like this. And the psychologists really do care about how stimuli and, and, motor actions and reward all Interact with each other and they've turned to the computer scientists to say, how might the brain be doing this? What, what is the problem that the brain actually might be solving? And so, guess what word [LAUGH] they borrowed back when they started to think about these things. Reinforcement. So, now they talk about reinforcement learning and they don't mean reinforcement, either. They mean reward maximization. So we won. [LAUGH] Yeah, I do not think that was really our plan but but it's, it's nice to know that we had some you know, impact. Well, that's very good you brought planning back into it and so they learned because of our plan, that's pretty good Michael. [LAUGH] All right, so that's the end of this aside. Okay. Let's go, let's go, let's go back to learning things.

3.1.30 API

All right, so now that we're back from our aside, let's go back to thinking about this applications program interface. So I talked about planning and I talked about learning. And it turns out that there's two other sub-processes or sub-components that might be useful to think about that kind of relate these quantities together differently. One is the notion of a modeler. What a modeler can do is take transitions and build a modeler out of it. That makes sense. And a simulator is kind of the inverse, where you can take a model and you can use it to generate transitions. You can actually kind of imagine running around in the world, Just by simulating that row. This is generally not a very hard thing to do. Though there's certainly applications of reinforcement learning where this simulator is extremely expensive. Because it's simulating a lot of things in the world. And this modeling problem you can think of again as a kind of machine learning problem. Right? Trying to map this kind of information into models. So you call them sub-processes. Does this mean that one way to do learning would be to do modeling and simulating so that I had models. And I knew what the reinforcement function was, and then I could just do planning? Yeah, that's a really good idea. In fact, let's look at different ways of gluing these things together. Okay.

3.1.31 API Quiz Question

So your suggestion was to take a modeler and use it to take turn transitions into a model, but once we have a model, you already told us last time how we can use a planner to spit out a policy. You didn't talk about those planners. What are, what's the name of the algorithms that you described last time? Value iteration and policy iteration. Yes, right. So, so what you can do is, is run either of those algorithms in here. They both have the same API right, they both take models and spit out policies. True. Alright, so, so let's do this as a quiz. We'll say, let's use your idea of mapping transitions to model to policy. And what would you call this whole box? Right? And so as a whole box, it takes transitions in and, and produces policy out. So it is solving the reinforcement learning problem. But it's taking a very particular approach to it.

But let's contrast it with the, with kind of the opposite idea. Which is, we can also map a model through a simulator into transitions. And then if we have the ability to do reinforcement learning. We can use, turn those transitions into a policy. So again, as a, as a, composed system. This is turning a model into a policy, so it is a kind of planner, but it's a planner that uses a learner inside and this is a learner that uses a planner inside. So just, as, just out of curiosity I would like to, just ask people what they'd want to call these. I'm not going to grade these, but I just I'm just interested. Like, what would you call this approach? Does that make sense? Just type it in the box. Let's pretend it makes sense. Alright. Go.

3.1.32 API Quiz Solution

All right Charles, so what would you, what would you call let's, let's let's think about this bottom one first. Okay. So what would you call an approach to reinforcement learning that what it does is it builds a model and then plans with it? A planner? Well, it's not a planner. I mean, the planner's inside of it. Sure. The overall system, this sort of blue box. Turns transitions into policies so it's kind of a reinforcement learner. Yeah. But it's one that builds a model inside. I would call that. You know what I would call that, I would call that model-based learning or model-based planning. Actually it's called model-based reinforcement learning. Mm-hm. So this is, this is, in fact, my, you said, reinforcement learning is your favorite kind of learning? Mm-hm. Model-based reinforcement learning is my favorite kind of of reinforcement learning. Mm. But we're not going to get to talk about it very much, so I wanted to at least have this slide to give people a chance to, at least, you know, these are all pieces that you can imagine doing. Right? This, this piece you can think of as being what we did in, when we're talking about supervised learning. And this piece is what you talked about last time. So you know, you could build a model based reinforcement learner that way. Yeah, that makes sense. Alright, how about this other idea, where, where you start off with a model and then you pretend you don't have a model, you pretend you're just in a learning situation by turning that model into, into transitions just by simulating them. Then the learner interacts with the simulator. And then spits out a policy. Well, I could come up with one of two answers. Okay. So I could try to do pattern matching on the answer to the second one. And since that one's model based, then this one's transition based, which is kind of cute. Or I could say, well one difference between at least inside the kind of blue box is it's sort of a model free reinforcement module. because the learner does not ever get to see the model. This is true, the learning piece is model free but we're using model free learner in service of planning. Okay. So, I don't know, I don't have a good this like model free planner, or an RL-based planner, maybe. Like I said, I wasn't going to grade it, I just was kind of interested to see what, what you'd say. Hm. Well I, I was pretty, I felt pretty good about the model-based RL one. Yeah, well this is the actual term that's used in the field. I'm, I'm sure there are some terms that are used here but nothing, nothing really very consistently. Hm. What if I called it the blue box planner? You could call it that, and, but had I, had I drawn it with a black box, it would, it had a better name. mm, oo, that would have been really cool, black box planner. I like that. Okay. But but I want to point out that one of the most successful applications of reinforcement learning are least most celebrated was Jerry Tassara's backgammon playing program, which used exactly this approach. Backgammon is a board game. So we have a complete model of it, but we don't really know how to plan it, it's a very complex, very large state space. So what he did is, he actually used a simulator backgammon, to generate transitions, and then used reinforcement learning approach TD, to, to create a policy for that. So it, his TD Gammon system followed exactly this, this overall pattern. Yeah, it was very influential work and generated many mildly embarrassing Master's theses. [LAUGH] I can only think of one. Oh, actually I can think, that's not true, I can think of two. Oh really? Well, which one are you, you're thinking of yours. Yes, I'm thinking of mine. I try not to think about it. The, the other one is, Justin Boyin, who's a good friend. And, a highly influential reinforcement learning contributor, and now Googler. Who, his Masters thesis was also, it's basically another TD. He did backgammon with RL. Oh, I didn't realize that. I, I actually like him. I think he's very cool. [LAUGH]. So, I'm sure that, his was not mildly embarrassing. I mean, you know, everybody's master's thesis are mildly embarrassing because you're, you're struggling to learn about how to talk about research, and so you're not going to get it perfect. Yeah, and in my case, it was it definitely model free. Right, the only non-embarrassing master's thesis that I'm aware of. Shanon's. Oh sure, alright, well Shannon. What was his master's thesis? It was information theory. Oh, yeah. It's pretty impressive. Yeah. Rob Schapire's master's thesis is pretty cool. I have no doubt. The boosting guy. He did, he did pom de pe learning. For his master's thesis? Yeah. With the diversity representation. I am a terrible human being. [LAUGH]. I mean wow. I mean it is true though, you look at someone like Shannon and you just think to yourself, oh, for his Master's thesis he did, he invented information theory. [LAUGH] You know, like wow, I wonder what he did for his PhD thesis? You know what he did for his PhD thesis? Nobody knows, nobody cares, because he based it on. [LAUGH] He, he had already done the information

theories. They're like here, fill out a piece of paper, you can pick up a PHD. He did something on AI, I can't remember what it was, but it was, it was totally unimportant and nobody cares about it. But information theory? Yeah. [LAUGH]. Thanks. All right, well, yeah, okay. [LAUGH]. II think, I think Schapire's PHD was boosting. Thanks Rob. Make us all look bad, why don't you.

3.1.33 Three Approaches to RL

Alright, we're going to drill down and talk about a specific reinforcement learning algorithm in just a moment. But I wanted to remind everybody about, some of the quantities that Charles introduced in the lesson last time when he was talking about MVP's, and use them to describe three different approaches for solving reinforcement learning problems. So this first box here pi, maps states to actions, and what did you call this Charles? A policy. That's right. And reinforcement learning algorithms that work directly on trying to find the policy, are called policy search algorithms. So the good thing about policy search is, you're learning the quantity that you directly need to use. Right? You're learning the policy. That's supposed to be the output. So that seems like a really good thing. Unfortunately, learning this, this function is very indirect. We don't get direct access to, I was in this state, what actions should I have chosen? Right? So this is, what did you call this, Charles, the the temporal credit assignment problem, right? Right. So, the data doesn't tell you what action to actually choose. Right. And this is why it's not exactly like the way we've formulated supervised learning in the past. Well, at least if we're trying to do policy certs, that's right. But what we're going to do is now consider, well, maybe that's not the quantity that we want to learn. Let's, let's think about learning this function U, which you had said maps states to values. So, what was this guy? U is a utility. Yeah. Yeah the true utility of the state, sometimes I, I think I called it the, the value of the state. Yeah so, so, so sometimes it's referred to as a value function, and learning methods, reinforcement learning methods that target that as, as what they're trying to learn directly, are called value function based approaches. Mm-hm. And, let's say that we, that we try to learn this, we're trying to learn to map states to values, well the good news is, at least if we're acting in the world, we're getting to see, okay I was in some state, I took some action, duh, duh, duh, and I can, I can observe the values that actually result from that, and maybe use that to, to make updates. So you can kind of imagine learning this, so that the, the learning's not quite as indirect. How do we use this to decide how to behave? So we, we need to turn it into a policy. Mm hm. And we didn't quite, we sort of talked about how to do that, in terms of, of looking at the Bellman Equations. We're going to, we're going to, it turns out it's actually kind of hard to use U directly to do that. But we're going to talk about a different form of the value function that's going to make that easy. So this is actually, it turns out it'll be a relatively easy kind of argmax operation, once we have the right kind of value function. Okay. So, there's some computation we have to do, and there's some indirectness to the learning, but, you know, it's okay. Alright, so the other quantities that you told us about were T, which is, I think of it as the transition function, I think you had a different name for it. I just call it the transition model. The, the model, right, yeah. So this is the model, and and the, the R is the reinforcement function, the reward function. Mm-hm. And what do these things do, they take states and actions and the transition function, or the transition model, predicts next states, maybe probabilistically, and the reward function tells you, next rewards, but just rewards. And so this is a model, jointly, and I already mentioned this, but we call methods that learn this quantity, learn these, these functions and then use them to do decisions, model based reinforcement learners, so how do we go from T and R to something like U? Well if we had those, if we had T and we had R, and we could see the S's, and we, and the actions we took, then we could do you know, something like value iteration we did before the learned values. Right, which value iteration was used to solve the Bellman Equations. Right. So that is somewhat heavy weight computation to do, but it's doable. So, what would happens over here, is we actually have fairly direct learning. Why is that? Because when we're trying to learn the transition and reward function, we get state action pairs as input. And then when we receive next state reward pairs as output, so we can solve this as a supervised learning problem. Hm. So, so learning is rather direct, the usage of this is a little bit computationally complex because you actually have to do the planning and then the optimizing to actually develop what you're, you know the policy doesn't come directly out of that. But but this kind of gives you a sense of three different ways, three different ways, places that you can target the pieces of the MVP so that you can do reinforcement learning. I like that. Now, we're going to focus on this, this middle piece. Partly because, you know, it's kind of the Goldilocks situation. It's you know, the learning's not so indirect and the usage is not so indirect. There's just been a tremendous amount of work focused on, on value function based approaches. And it's, you know, remarkably simple it turns out, if you do it right. And also very powerful. That there's lots of ways to use these simple ideas to, to learn hard problems. So, I think this is a good place to focus. Okay. I buy that.

3.1.34 A New Kind of Value Function

So let's talk about a new kind of value function, that's going to actually make that optimization part easier and the learning part easier. So but it's really closely related to the stuff you talked about, Charles, so let's let's start with what you told us. Which is here's a definition of, of you. This is, we're going to define you. Mm. And put you in a box. It's been tried before. It can't be done. All right, so well, you know, if you, if you're a good dog, I'll give you some pie next. So U, [LAUGH] [LAUGH] U is defined for each state, the utility being the state, this is, the long-term value of being in a state is the reward for arriving in that state, plus the discounted reward of the future, and what's going to happen in the future? Well, in the future, to leave this state we're going to choose some action, then we're going to take an expectation over all possible next states. And we're going to arrive in some next state, S prime, and then U is representing the utility of that as well. So this is, is recursive and nonlinear, but we know how to solve this, we can use things like value iteration to do that. Agreed? Agreed. Alright, and you also said, here's how you can use this quantity to decide what to do. That the policy in a state S is, well, let's consider all of the actions we can take to leave that state. We'll look to what their expected values are, so we'll iterate over all the possible next states weighted by their probability of the utility of landing in the state that we'd end up in. Mm-hm. And that, that tells us how to behave. Yes. Alright, so these are, these are the value functions, well the value function of the policy that we talked about before. Here's our new kind of value function. It is sometimes called the Q function. Though, you know, some people in the know don't like that. It's, it's called the Q function because it's the letter Q. But it's, some people have said that it stands for quality but it's just, it's just a letter in the latter half of the alphabet, you know, V was taken, U was taken, you know, W is used for weights, like, Q was available. So, it was brought to bear. So, so this is, this is a, a new definition and you can see it's got elements of the other two. Let me maybe write down what it, it, a way to interpret this. Okay. So, again, here's the, the Q function. And what we're going to think of this as, is, this is the value for arriving in some state, S. And this is, you know, this is the reward we get for, for that arrival. Then what we're going to do is we're going to leave S via action A. So we're going to add to that the discounted expected value that we get for taking action A. It's going to now drop us to some next state S prime, according to this probability, and once we land in S prime, we're going to take whichever action has the highest Q value from there, okay? So that turned out to be the value for arriving in S, leaving via A, and proceeding optimally thereafter. Like once, once we get to a new state, we're going to be choosing the best actions each time. Does that work for you? It makes sense right, because you could, U is basically defined the same way. It's the value for arriving in S and then proceeding optimally thereafter. That's right. And all the, the only thing that we're doing here is we're, we're basically tying the algorithm's hands briefly. We're saying, we're going to force you to leave via ac, via action A. After that, do the normal thing. But just for a moment I'd like you to just take action A. Mm-hm. Okay, that makes sense. So you just, you, you inserted basically a, a kind of, kind of utility step in there that I assume you're about to use in some clever way. Indeed, yes, this is going to turn out to be really helpful, because it's going to allow us to compare the values of different actions without having to actually stare at the model to do it.

3.1.35 Value Function Quiz Question

Alright, so I just introduced this Q function. It turns out that baked into this one little Q function is everything we need for, for dealing with U and pi without knowing the transition function T or the reward function R. And to you know, to demonstrate this for you, I'm going to let you demonstrate it to me. [LAUGH] So, [LAUGH] here's what I'd like you to do. I'd like you to rewrite these equations, this U and the pi equation, in the little boxes, using Q instead of any references to T and R, alright? So, imagine that we have Q, that somebody's already solved this out for us. How can we define pi and U using Q? Does that seem okay? Seems okay to me. I think I might even know the answer. Alright, that would be awesome. So let's, let's give people a chance to think about it and then just tell me what you think. Okay.

3.1.36 Value Function Quiz Solution

Alright, Charles [LAUGH], let's, let's get back to the task at hand, and you thought maybe you had some ideas. So, tell, tell me, how can we define U and pi in terms of Q? Okay. Well, so I guess the first thing to observe is that U is a value. Right? U are a value. Okay. Yes. U is a value. That's correct. [LAUGH] U returns a number. A scalar, in particular, where pi returns an action. Yeah, U of s returns a scalar, that's right, and pi of s returns an action, very good. Right, okay. And, you know, we have the definition for U up there near the top, and it's just the reward and then sort of behaving optimally thereafter, where Q is a reward and then taking a specific action, and and behaving optimally there after. So, we can sort of turn

Q into U, if we always pick the best action. And we know the best action is, it's the one that, you know maximizes the value that you're going to get from that point on. So, I would say that U of s could be defined as Max over a of Q, s comma a. Simplicity itself. Mm-hm. Nicely done. I have no idea how we're going to grade that automatically, because it's hard to type a's under other a's. But, yeah, that's exactly right, that we have the maximization over all possible actions of the Q value in that state. Like, that's just great! Mm-hm How about the policy? So the policy will look exactly the same, the, the, the policy that you want to follow anyway is the one that... Maximizes your value going forward except the differences. It's returning an A and not an actual value, so it should be argmax. Oh, So it's not exactly the same. Right. It's almost exactly the same. Rr... max Right, so this is, it's just so trivial, so, so Q gives us everything we need. If only we had a good way of finding Q. Yeah, if only, if only we could find a Q, a Q for you. [LAUGH] and that's going to be the essense of Q LEARNING. Oh, well done.

3.1.37 Q Learning Question

Alright, I apologize in advance but we're going to do a quiz. Q-learning, The Quiz, you know, because quiz starts with Q. So which of these actually describes Q-learning? Is it the problem of figuring out the best line to wait in? Is it discovering when to come in for your line, when you're in a play? Is it practicing the best bank shot? Or is it evaluating the Bellman equations from data? Alright, do you want [LAUGH] are you willing to do this quiz Charles? I am, I am willing to do your quiz your quiz learning quiz. All right. All right. Let's do it then. Okay. Go!

3.1.38 Q Learning Solution

All right, take me through. Okay. So, the first thing I want to say is, I think it's awesome that you made me do this quiz. The second thing I want to say is, in some sense, every single one of them is correct. Yay! But, you put radio buttons down, not check box buttons, thingys, so I have to pick one. So let's just go through. Figure out the best line to wait in. Well, that is a queue, queue which would make a lot of sense if I were, say, English. Yes, that's right. In fact, you are, in this picture. Oh yeah, great. [LAUGH] That's the English version of Charles, the one with the top hat. Okay. Yeah, and a cane, too. Yeah, I like it. And a cane, yes, well that is typically what I do whenever I go to London. Discovering when to come in for your line, that is also a queue-learning though a different kind of queue. que-learning. The third practicing your best bank shot, although I probably would've said break shot. Ooh, nice. That is also cue-learning. And it's the same spelling. That's the same spelling. That's unfortunate. Well, you pronounce one cue, and you pronounce the other one cue. Yes, well clearly. But, and I spelled them that way too. Yeah, and I know. I know. I can see where the emphasis is. And the fourth one is evaluating the Bellman equations from data. That is you take in the states, the actions, rewards and the next states, and you try to learn an actual Q function, which is just the letter Q. So that is Q-learning, and is the only one that is spelled correctly, and so that is the choice that I would make. Alright, so, just to be clear here. So this all [LAUGH] the purpose here other than to. Demonstrate your ability to show puns? [LAUGH] Is that is just to give the definition. So that, so what Q-learning is going to be doing is it's going to use transitions, that is to say data, to directly produce the solution to those Q equations.

3.1.39 Estimating Q From Transitions

Alright, so what we're going to do to figure out how Q learning works, is we're going to think about what it means to estimate this Q function from transitions. So, let's just remember this is the form of the Q equation, that we've been talking about. And, we can't do this. We can't solve this, because we don't have access to R and T. All we have access to are transitions. So this a really, I guess, I'm going to guess you said this before, but when you, when you write out this equation it really jumps out at me. This is the difference between what I was talking about, solving MDPs and reinforcement learning. In solving MDPs we had R and we had T and now we do not have them, so we have to come up with some other way to solve these kinds of equations. That's right. Okay. So if we did have R and T, then we could solve this? Yeah, this is, I mean, the same things that you talked about, value iteration, policy iteration? It can be formulated in terms of Q. So it's, yeah, there's, this is easy to do, well, [LAUGH] it's polynomial to do if you have access to T and R. But but again, in the learning scenario we don't have the model. What we have are transitions. Okay, okay. So, here's how we're going to use transitions. This is what a transition is. We, we observe that we were in some state S of the MDP. And then action A was chosen somehow. And, then a transition happens. We land in a state. We get the reward for landing in that state. And we find out what state we're in. So that's,

that's the transition. And what are we going to do with it? Well, what we're going to do. Is imagine that we've got an estimate of the Q function, Q hat, and we're going to update it as follows. Here's how we're going to use all these quantities from the transition. We're going to take the, the state and action that we just experienced. And we're going to change it; we going to update it; we're going to move a little bit. Alpha, this is alpha, this is called a learning rate. We're going to move a little bit in the direction of the immediate reward plus the discounted estimated value of the next state. So, we're going to take our estimate Q hat, we going to take the state that we end up in as prime. We're going to look at all the different actions we could take from there and take the maximum. So this together is kind of an estimate of the utility, right? Mm-hm. And this is the utility of the state that we're going to. This altogether is the utility of the state that, that we're in. To the state S. So this is kind of the utility of the state that we're landing in as prime. And this all together is, is related to the utility of the state, right? You can see that it's related. In that we've got the immediate reward, which kind of matches to this. We've got the discounting. We don't have the sum over transitions but we do have the max A and the lookup in the next, in the Q function. Alright so this, this is the Q learning equation. Alright, let me just say a little bit more about this, this alpha arrow notation, which I really like but is not all that standard. So if you, when I write you know something like V gets with an alpha X. What we mean is we're moving alpha of the way from the current value V towards X, which can be written this way. That V gets 1 minus alpha of V plus alpha of x. And so in particular, if you think about this as so if alpha is 0. That's sort of a learning rate of 0, which shouldn't learn at all, and in fact, if you set alpha to 0 here, it's going to zero out X, and it's going to only assign V to V, so nothing's going to change. So learning rate of 0 corresponds to no learning. And if we set alpha to 1, that's like full learning, so we forget everything that we knew and we just jumped to the new value. And that's what happens here, that 1 minus alpha is 0, so the V goes away, and we just get X assigned to V. So does that make sense? That does make sense. And and if alpha were in between 0 and 1 like one half, you're basically making V the average of the old value of V, and the new value X that you see. Good.

3.1.40 Learning Incrementally Question

Alright, let's do a little quiz. Okay. To help us to kind of, make sense of the equations in the Q learning equation by looking at a simpler case. So let's imagine that we got some variable V and some sequencing value X and a sequence of learning rates, alpha sub t. So we're going to draw a series of these X values and use them to update the V values and the learning rate is changing over time and the learning rate is going to have, first of all X is going to be drawn X sub T. Is going to be drawn from some, the distribution of some random variable, big X. Mm-hm. And the learning rates are going to satisfy two properties. One is that the sum of the learning rates, summed up to infinity. But, the sum of the squares of the learning rate, as we go to infinity, sums up to something less than infinity. So can, can you think of a learning rate sequence that has that property? So, the one that I remember is alpha sub t equals one over t. Good. So in fact these's a whole range of possible powers of t that work here, but one over t is a good one. Why is this, why does it satisfy the properties? Well, if you sum up the values up to sum value t, sum up the one over i values, it actually acts like the natural logarithm. And so, as t goes to infinity, the sum goes to infinity. But logarithmically. Which is still you know, infinity. Yeah. But if you look at the sums of the squares, that's actually a well known problem called the I think, the Basil problem, the Basil problem? And it turns out to actually be pie squared over six which is kind of crazy. But there it is which is a finite value. That's intuitively obvious even to the most casual observer. No I really, this, it's, it's, it's whacky. It was an open problem for a long time and then it was finally solved. And it's like, yeah sure pi squared over six. Like where's the pi in here? There's no pi in here. Mm, I didn't find pi anywhere. [LAUGH] That's neither here nor there. The whole, the point is there's, there's a bunch of sequences of learning rates that satisfy this property, and we're going to imagine that we have a sequence of learning rates that satisfies this property. We're going to be updating V sub-t, with a series of X values drawn from, from distribution big X, and what I'm asking is, what do you think this converges to? Do you think it converges to the expected value of X? Do you think maybe it just doesn't converge at all? Do you think it converges to the variance of X? Of, of the random variable X? Or does it converge to infinity, it just keeps growing without bound? All right. Does that seem like a well formed question? sure. All right. So let's give you a chance to think about it. Okay.

3.1.41 Learning Incrementally Solution

Alright, what do you think, Charles? Okay, so I think a couple of things. Let me just talk this out loud. So we're trying to learn incrementally. We're trying to figure out, how if we just kind of keep adding these random variables in a sort of way to what you described before what we would end up at. So why do we

want to choose alpha t with a particular kind of property well if you look at it, it's clear that alpha t at each time step is moving closer and closer to zero. So it means you start out learning a lot and you sort of believe things less and less and less or you let things change you less and less and less over time. So what is that likely to end up at? Well the first thing I notice that well if it's going to do that then it has to converge. Some point. It just makes sense. So I, that helps me to eliminate the second answer. But actually, that also let's me eliminate the fourth answer because in some ways they, that says the same thing. So that leaves us with the expected value of x and the expected value of x squared. For the expected value of x squared. Whichever one it is that is variance. Okay so, I'm going to go with, the expected value of x. So, the expected value of x, right. So, so it turns out that this is actually a wave computing the average by just, you know, repeatedly sampling and updating your values. And, and the way to think about it, is that sometimes the x values that we draw are going to be a bit above the average, and it's going to pull the v value up. Sometimes it's a little bit below the average, and it's going to pull the value down. But in the limit, all those pulls and pushes are going to cancel out, and it's going to settle in on the actual average value or the expected value of this random variable. Right, because in principle, you're going to see an infinite number of those things. And effectively all you are doing is adding them all up, and sort of, you know. Well you're effectively averaging them one little bit at a time. Yeah, that's a good thing about it, that in fact it's adding them up and it's computing a weighted average, but the weights are these alpha's and the weights are decaying over time so we're going to put more weight on the more recent ones, but some more weight on the further away ones but it sort of doesn't matter because the order, since we're drawing this iid, the order doesn't matter and the thing that has to converge to is the mean. Right. I like it. So let's relate that back to what we do in Q-learning.

3.1.42 Estimating Q From Transitions Two

Alright, so this is now the Q in the equation, again. Which again is one of these alphabased things. And, just to be clear, I really do mean, you know, alpha sub t. That we're, that were doing this over time. We're updating our learning rates as we go. It's just, sometimes it's a little irritating to put that, that there. But but yeah. So let's imagine that we're doing that. We're, we're, we're using that same kind of weighted process. Bumping the values around. So, what would, based on what we just talked about, Charles, what would this actually be computing? Well, it would be computing the average value that you would get for following, you know, kind of optimal policy after you take this particular action. Yeah, that it's, it's trying to go to this expected value and, and what is that expected value. So the linearity of expectation says that we can actually move the expectation to break up, break up the sum using the expectation. So this, the expected value of the reward is actually r of s. Mm-hm. The gamma's going to come out because of linearity of expectation. ANd then what we're left with is the expectation over all next dates of the maximum estimated value of the estimated Q value of the next state. But what is this distribution over S prime? It's the distribution that is determeined by the transtiion function. SO it's this. Which is you know this. So that's good. It's I'm kind of cheating though. Do you see how I'm cheating. I think I know how you are cheating but tell me. Well so when I told the story about this alpa arrow updating thing. I said that it convergence to the expected value of this quantity here. But this quantity here... Since it's "Q hat", it's actually changing over time. Right. So this target is actually a moving target. It's changing over time. So we can't quite do this analysis because the first step is a little bit questionable. But it turns out that there really is a theorem that this simple update rule, this Q-learning update rule, this tiny little one line of code, actually solves Markov decision processes. Yeah.

3.1.43 Q Learning Convergence

So this is a remarkable fact about this Q-learning rule, and that is if we start Q hat off pretty much anywhere, and then we update it according to the rule that we talked about. Q for, for when we see a transition s,a, r, s prime, then we update (s,a), the Q value for (s,a), move it alpha of the way towards r plus gamma, max a of the, well basically the Q value of the state S prime. Then as long as we do that, then this estimate, this q hat S A goes to Q S A. The actual solution to the Bellman equation. And I write this with an exclamation mark, because it's like, it's one line of code! It's one line of code, like, how could you not just go out and write this right now? Hm. But the, the, the, let me just, to finish is, this is only true if we actually visit SA infinitely often. So you know, that's an important caveat. That for this to, to actually hold true, for you to really converge to the, the solution, it has to run for a long time. It has to visit all state action pairs. The learning rates have to be updated the way that we talked about before. The next states need to be drawn from the actual transition probabilities but that's, that's cool, if we actually are learning in some actual environment

and the rewards need to be drawn from the rewards function. So, this isn't so problematic. This is a little bit problematic, but it is still very reassuring, this idea that we have the right form of an update rule, so that the thing that we converge to is the actual optimal solution to the MDP. Cool. And we just have to wait til the heat death of the universe, or infinity, and then we're done. Yeah.

3.1.44 Choosing Actions

So, Charles, I kind of cheated. Oh, tell me more. So, Q-learning isn't really an algorithm. Q-leaning is actually a family of algorithms. There's lots of different reinforcement learning algorithms, specific reinforcement learning algorithms that can be reasonably called Q-learning, and they vary typically along these three dimensions. How do we initialize our estimate, Q hat, how do we decay our learning rates, alpha sub-t? And how do we choose actions during learning? Hm. And different ways of making these choices actually lead to algorithms with fairly different behaviors. In particular when we use this in the context of an MDP, well let's, let me, let me ask you. So like, what do you think might matter about let's start with the last one, choosing actions? Well. It see, well there's a bunch of dumb things you could do, right? You could just, just pick an amption, action, action every single time, like the same action every single time independent of what you learned, that's kind of dumb. But, it seems like the obvious smart thing to do is say look, I'm learning, I'm getting better and better, so what I'm going to do is at this next time steps is I actually have to take an action. I'll just pick the action that my Q hat tells me is the best action to take. And I'm done. So all right, let me, let me see if I can capture some of what you just said there. So, one way to choose actions really badly is say, pick some action, call it a sub 0. And no matter what state your in. No matter what's happen so far, always choose that action. Mm-hm. Right, so this possibly can't work. It's going to violate the Q-learning convergence that says we have to visit each state action pair infinitely often, and update them to converge and you know and, and it makes perfect sense. Like if we never try something, like how do you know that you don't like something if you've never even tried it. Like spinach. Exactly. Another idea would be to choose randomly. And this seems kind of good and that we are going to visit. You know, all the states that are visitable and we will try all the actions that are actionable. And we could actually learn Q this way. But, as you pointed out, this is not a great idea because we may have learned Q, but we haven't really used it. We haven't really chosen actions using what we've learned, so it's like we're wise but we are impotent. No, I think it's more like we're wise but we're stupid. I mean. We're wise but we still. We know a lot but refuse to actually do anything about it. Right. In particular in some sense the only difference between the two is the the theorem, right? If I, if I'm just choosing randomly. Wha, what's the problem with them always choosing a sub 0. Well, you're don't going to converge but the real problem is that you don't learn anything. Or you don't take advantage of anything you learned. Choosing randomly is basically the same thing, you basically, you never take advantage of anything that you learn. What's the point in learning a Q function if you are always going to behave randomly, you've learned enough or you've learned but you [CROSSTALK]. [CROSSTALK] right you've actually learned the ultra policy but you're not following it so you're not actually using what you know, so you can't you know it doesn't, it doesn't work all right. So then you had another idea. Which was to use our estimate to choose actions. Yeah. And that seems like a good idea in that we will use it. Is it possible that it won't learn? Well, it will learn something. Well, yeah. It might not learn anything all that good though. So for example, what if we do something like this. So we initialize, now we're back up to this, this first point here. We initialize the, the estimate Q hat so that for every state, a0 looks awesome and all the other actions look terrible. Wait [INAUDIBLE] is that metric awesome? Or English awesome? Oh you're right I'm sorry, I didn't put units on that. That's in Chilean dollars. [LAUGH] Oh okay, so it's pretty awesome then. Okay. So, if you do that, then well let's see what happens. It's almost like always taking a0. The only thing that would save you from taking a0 forever, is if, as you take a0, you learn that, you update your Qs and you keep getting really, really bad results. Really, really bad results, in fact, worse than terrible. yes. Well let's imagine the terrible is worse than terrible. Oh. So, but you're right, yeah you're right. So, so there's, there's at least the case that if this, if this terrible value is actually lower than the value of always choosing a0, then we'll continue to use a0 forever. This is, this is called the greedy action selection strategy. Mm-mh. And that, this is the problem that runs into, it's a kind of local min. Oh, I see. Oh and oh, okay, I see that. So, you, you didn't even have to come up with this ridiculous example. If you, if you well, not ridiculous, but you, extremely. I was going to say, I'm, I'm sorry, ridiculous how? Well, I'm sorry. It, it's a ridiculous situation to be in. It's sort of the extremely unluicky example. I think what you're saying is that you don't like people from Chile. [LAUGH] Oh no I love people from, I love Chile. Especially with you know, just some good beans and some nice meat. But the thing is that you, if you randomly set your Q hat in such a way that a bad action or let's say a suboptimal action ends up looking much better to begin with then the optimal action.

You can get in a situation where you keep choosing the wrong action anyway and so you are only going to learn things that reinforce that action which might be good just not optimal. So you won't actually end up converging onto the true Q hat and that's why you get into a local min. That's right and so in that bad situation and you, I admit it's a little contrived because I've never, I've never even been to Chile is that it wont learn. It'd actually be exactly the same as this first case, always choose a sub 0. Mm-hm. So, that seems problematic too and it interacts in an interesting way with the initialization. Right. So maybe we can do this idea of using Q hat, but we have to be much more careful about how we initialize. Hm. So, you know, what I want to do is something like random restarts.

3.1.45 Choosing Actions Two

So I think that is a clever idea, random research. That was one of the ways that we got unstuck when we were in local optima when we were doing optimization. Maybe this will also come in handy, in this setting. Yeah I like the idea because I came up with it but I can see a couple, you know, but I can see a couple of problems with it. But well you can't see a problem with it yet because it's not even clear what you meant. Or maybe that's the first problem. Well it was clear what I meant in my head, now if you're going to want my mouth to understand what my head meant, well then your just asking for too much, so I really kind of meant, random restarts where you just kind of start it over, over, over, and over again, the problem with that, the problem with that is, it's already going to take a long time, you know, infinity to get to a good answer. And we thought we might have this issue with randomized optimization, certainly this is going to be a problem here, but it still feels like there's something we could do with the randomness idea that would help us to overcome this problem with using what we know. And only using what we know. Alright, alright, so I, I like that direction. So let's think a little bit about what that could mean. So random restarts was one idea that we had when we were talking in optimization about getting unstuck, which is, let yourself get stuck, and then once you realize you're stuck, throw everything out and start over again. Right. And you're right. That's going to end up being really slow, but we had a different way of Of using randomness to get unstuck. In, for example, algorithms like Simulating annealing That's what I was thinking about. Yeah, so Simulating annealing. The idea is Simulating annealing is that you tend to, you take up hill steps. But occassionally, you're willing to take a random down hill step. Right. So, it's kind of this mixture of choosing randomly. And using what you know, what seems to be the best. Right, and so, yeah, yeah, I can see that. So then the random restarts thing kind of works if instead of it being a random restart, it's, it's just a random action every once in a while. Yeah, excellent alright. So simulated annealing like-approach says we're going to take a random action sometimes. So then our exploration policy, our approximate policy, is going to be. To, we're in state s. Figure out the best action in that state according to our estimate and take that with probability I don't know. let's say one minus epsilon. Mm-hm. In otherwise take a random action and see what happens. I have a probability of epsilon. Yeah that's whats left over so. Right so that's why you had one minus epsilon because you want epsilon to be small. So epsilon is going to be. So every once in a while you will randomly act, oh and then that'll, that'll solve the sim annealing problem. Or do what sim annealing helps you with by just taking a random action in what looks like a bad direction, comparatively, sometimes. But it also means that you get to explore the whole space. And so you have a chance of actually learning the true Q. Exactly. So, and yet most of the time, you know, we're spending, assuming Epsilon is small, we're spending a lot of our time taking good actions, or actions that we think are good. But we're always holding out some probability of taking other actions to help us improve our Q hat. Mm, so if you do that infinitely long, and you've, this will, so this will let you visit SA. An infinite number of times, as long as epsilon is greater than zero. That's right and that the mdp is connected, right? So, there could be state action pairs, that just can't ever be reached, in which case you won't reach them. But that's okay, because since you can't reach them, they really don't matter. Yeah, they sort of don't exist. Exactly. Alright so let's let's focus in on this a little bit more because I think that this is now the first idea that we've had for choosing actions that has the property that it will learn and it will use what it learns. Mm, I like that.

3.1.46 Greedy Exploration

Alright, so let's say something that we actually know about this approach to action selection which is called Epsilon Greedy Exploration. And what we know is that if the action selection if GLIE, which means Greedy in the limit with infinite exploration, and that basically means that we are decaying our epsilon for epsilon Greedy. That we start off more random and over time we get less and less random and more and more Greedy. Then we have two things that are true. One is that Q hat goes to Q, which is, which comes from

kind of a standard Q learning convergence result. But, we also have something cooler in this case which is that the policy that we're following, pi hat, is getting more and more like the optimal policy over time. Hm. So, not only do we learn stuff, but we use it, too. And this is an example of the exploration exploitation dilemma. And exploration exploitation is really, really important, not just because they're two words that surprisingly start with the same five letters, like unlikely letters. But also, that it is strike, it is talking specifically about this issue. Exploitation is about using what you know. And exploration is about getting the data that you need so that you can learn. And this is one particular way of doing, it turns out there's lots of other ways of making this trade-off and the reason it's a trade-off is because there's only one of you. There's only one agent acting in the world and it has these two actually somewhat conflicting objectives. One is to take actions that it doesn't know so much about, so it can learn about them and the other one take, takes actions that it knows are good. So that it can get high reward. Hm, that makes sense. You know, I didn't, I never realized that exploration and exploitation share the first five letters. Hm. I always knew that they shared the last five letters. Oh, that's interesting too. Huh. So if you take an r and turn it into an it, you might move from exploration to exploitation. I feel like an entire political movement could be founded on that. [LAUGH]. But I'm not sure exactly what it would be yet. Maybe I'll work on that, before our next lesson. So I have an algorithm. There, there's a standard, lemma. Mm hm. In reinforcement learning theory called the exploration exploitation dilemma. Sorry, [LAUGH], no, lemma. The other kind of lemma. The exploration exploitation lemma, which has to do with taking actions that are either exploring or exploiting. But I have one where you actually do teaching. Mm-hm. You can actually, each time you take an action it's either going to teach the agent something, it's going to explore or exploit. So I call that exploration exploitation, or explore, exploit, explain. Oh, nice. But you could have called it the exploration exploitation dilemma because you used dilemma. And di means two sometimes. And you do the two things. Well, in fact, dilemma does literally mean two. Its a choice between two things. Right, so its a dilemma. Nice Its a lemma about two things. Alright, so there is, it turns out there's a lot of other approaches to exploration and exploitation. And some of them in the, in the model based setting. You can do a lot more with it, a lot more powerful things with it because you actually can keep track of what you've learned effectively in the environment and what you haven't, so the algorithm can actually know what it knows and can use that information to explore things that it doesn't know and exploit things that it does know. Q learning doesn't really have that distinction. It's a much harder thing to do. So, so that's what I wanted to tell you in terms of, you know, thinking about exploration-exploitation, does that make sense to you? It does make sense to me. I think what, the main point I got out of this, or a main point I got out of this, other than our incredible ability to get caught up in letters and coincidences of spelling, is that the exploration-exploitation dilemma really is a dilemma. It's like the fundamental tradeoff in reinforcement learning. You have to exploit because you have to use what you've got, but you have to learn or otherwise you might not be able to exploit profitably. So you have to always trade off between these things, and if you don't, you're bound to either learn nothing or to get caught in local minima. I couldn't agree with you more. In some sense, if you think of reinforcement learning as being the question of model learning plus planning. There's nothing new here because model learning is well studied in the machine learning community and planning is well studied in the planning and scheduling community. Planning. And so, like, what are we adding to this? And what we're adding is the fact that these two processes interact with each other and depend on each other and that's exactly the exploration, exploitation dilemma and that's information has to go back and forth between these two processes that other people understand and we're the glue. Or the glee. [LAUGH] The glee glue. I like it. That's beautiful.

3.1.47 What Have We Learned

So that's at least in a nut shell the reinforcement learning story. There's there's a lot of other topics and I think we are planning to get to some of them in a topics lesson a little bit later, but there's you know, courses and books worth of stuff to study in this area. Just like supervised learning as a whole. But, we're going to just kind of end it there with the idea that we now have a handle on how we can use learning to do decisions with delayed rewards. So, can you help us summarize what we what we learned in this lesson? Okay, sure. I think what did I learn here today, I think I learned a lot of things, some of which had to do with reinforcement learning Mainly that you can actually learn how to solve an MDP. I think that's actually a pretty big deal. Right. Meaning that we don't know T and R. Mm-hm. But we just have access to. The ability to interact with the environment and recieve transitions. And, that's actually that's actually pretty impressive. And, a very powerful thing. Because, we're often not, if we assume the world is an MDP, but we don't know TNR. If we don't have some way of learning in that we don't really have much we can do. And, you've showed me that their is something we can do. Cool. I think that's the biggest thing. We

learned some specific things. In particular, we learned about Q-learning. Several kinds of Q-learning, but one is actually a real word. Yes [LAUGH] indeed they're all real, Michael. We learned about Q-learning. And I think the other most important thing that we learned about is the exploration versus exploitation trail. And with Q-learning, we learned a little bit about when it converges. Mm-hm. And that it's actually a family of algorithms. And different members of that family have different behaviors associated with them. Oh, there's one other thing I wanted to say on that topic, actually. Okay. Which is, that one way to achieve this exploration-exploitation balance, was to randomly choose actions. So to change the we're doing action selection. But there's another one too, which is that we can actually by manipulating the initialization of the Q function. We can actually get another kind of exploration, can you see how that might work? Oh, I know what you do. If you could, if you set, say the Q values to all be the highest possible value they could be. Great, so if we initialized the Q hat to awesome values, then what the Q lettering algorithm would do, even with greedy exploration, what it will do is it will try things that it hasn't tried very much, and it still thinks are awesome. And little by little, it gets a more realistic sense of how the environment works. And. So it's very optimistic? That's right, exactly and it's referred to often as optimism in the face of uncertainty and it's a similar kind of idea that's used in algorithms like, A*, if you're familiar with search algorithms in AI. Oh yes, I remember those. But this is, this is a really powerful idea and it's used in, in reinforcement learning and banded algorithms and planning and And search. Okay, and that makes sense because if everything is awesome. Then your true key value can only go down if awesome is bigger than the biggest Q value you could ever have. And so that means you're going to look at every single action. And as you learn more about them, then you will just get more depressed about them. And that's good. [LAUGH] Yes the world slowly beats you down. [LAUGH] So is that it? Is that all we really talked about? I guess that's about right. We talked about what a que function was. Right. And how that kind of binds everything together and we talked about different approaches including policy search and model base reinforcement learning. Yeah that was very nice. We tied it all back into planning. So one, one thing we didn't talk about is connecting to function approximation, and the issues in machine learning that are really important things things like over fitting. They come up in the reinforcement learning setting, but not in this simplified setting that were looking at here where we learn a separate value. For each state action pair, we're going to have to start generalizing to see the importance of that. And that's, we're going to do in a later lesson. Okay. I like it. And we also learned a bunch of things about letters. Like exploration versus exploitation. In fact, we know enough that we can now get an A in letters. [LAUGH] I like it. Okay. Well, I think we learned a lot, Michael. [LAUGH] Okay. Well, good. Well, thanks. It's very nice to get a chance to talk to you about this stuff. Cool. And so I guess. What are we going to talk about next. Well Whatever it says on the syllabus. I think it's game theory. That's pretty cool. Oh. I see why we're going to do that. Because all we've been talking about the world as if there were just one agent and nobody else. And now we're going to see what happens. When there are other people. Right. Other people show up at the party, next time. On, Machine Learning.

3.2 Game Theory

3.2.1 Game Theory

Hi Michael. Hey Charles. How you been? I've been doing just fine. How've you been? Good. It's been a while. I understand you went to Africa. I went to Africa. The entire continent, or at least parts of it. How's Africa? It's fine. It's, warm. The entire continent? [LAUGH] The entire continent. Or at least Kenya and Nigeria. Okay. Particularly in Nigeria. Which was 80 something degrees and 4,000% humidity. [LAUGH] So, do you want to do a shout out to any of, any of your friends from there? Yeah, I'd like to say hi to Chairman and I'd like to say hi to Prof. Oh, and Good Times. I would like to say hi to Good Times. He was awesome. Okay, so Michael. I am back now so that we can talk about the last little section. That we're going to do in this class, and it's game theory. That sounds cool. What does that have to do with machine learning? So that's an interesting question because, in fact, game theory comes from a tradition way outside of machine learning in A.I. but you'll see in a moment why it is I care about game theory, and really this is a very natural extension to all the stuff we've been talking about with reinforcement learning. Interesting. Are we talking about games like Monopoly? In some sense, we are. Because all of life is like Monopoly. So in fact, what is game theory? Maybe that's what we can do. We can just try to define game theory, and maybe it'll be clear why it is we're worried about this at all. Seem fair? Yeah. Okay.

3.2.2 What Is Game Theory

Alright Michael, so there's lots of definitions of game theory that we could use. One that I like in particular is that game theory is the mathematics of conflict. Hm, [CROSSTALK] that's interesting. I think it's kind of interesting. Or generally it's the mathematics of conflicts of interest when trying to make optimal choices. because I feel like a lot of people have their own conflicts with mathematics. I think everyone but mathematicians have their conflicts with mathematics. I think that's fair. I see. But do you see if you, can you see how worrying about the mathematical conflict might be a sort of natural next thing to think about after you've learned a lot about reinforcement learning? I guess then well the next bullet kind of, kind of suggests a trend. So, so we've been talking about decision making and it's almost always in the context of a single agent that lives in a world and it's trying to maximize reward. But that's kind of a lonely way to think about things, so what if there's other agents in the world with you? Right and of course evidence suggests that there are in fact other agents in the world with you. And what we've been doing with reinforcement learning which, you know, has worked out very well for us, is we've been mostly pretending that those other agents are just a part of the environment. Somehow all the stuff that the other agents do is hidden inside of the transition model. But truthfully it probably makes sense if you want to make optimal decisions to try to take into account explicitly the desires and the goals of all the other agents in the world with you. Does that seem fair? Yeah. Right. So that's what game theory helps us to do and then at the very end I think we'll, we'll be able to tie what we're going to learn Directly back into the reinforcement learning that we've done and even into the Bellman equation. Oh, okay, nice. Yeah, so that is going to work out pretty well but, but we have to get there first and there's a lot of stuff that we have to do to get there. But right now what I want you to think about is this notion that, we're going to move from reinforcement learning world of single agents to a game theory world of multiple agents and tie it all back back together. It's a sort of general note that I think that, that's worthwhile is that, game theory sort of comes out of economics. And then in fact, if you think about multiple agents there being millions and millions of multiple agents, in some sense that's economics. Right? Economics is kind of the math, and the science, and the art of thinking about what happens when there are, lots, and lots, and lots, and lots of people with their own goals possibility conflicting, trying to work together to accomplish something, right. And so what game theory does, is it gives us mathematical tools to think about that. I feel like I feel like other fields would care about some of these things too, like sociology. Right. And what about, I could kind of imagine biology caring about these things, too. Even biology, I like the idea of biology. Biology. Why would biology care about this? Well, I guess the way you described it in terms of lots of individual agents that are interacting. Like, you know, creatures that live and reproduce. I feel like they, they have some of those same issues. Sure. So certainly biology at at the level of entities, at the level of mammals or level of insects, you might be able to think about it that way. But perhaps even at the level of genes and at the level of cells. Little virii and, and bacteria. You could possibly think about it that way. because they're in conflict too, I guess. Yeah. Now there's probably this notion of intention. It's not entirely clear what that means here and I think implicit in the notion of what we're doing here is this notion of intention and explicit goals as opposed to ones that are kind of built into your genes, but I think that's a perfectly reasonable way of thinking about it. I think the, the lesson from this discussion though is that. What game theory sort of captures for us or what would like for it to capture for us, is ways of thinking about what happens when you're not the only thing with intention in the world and how do you incorporate other goals from other people who might not have your best interest at heart or might have your best interest at heart. How do you make that work? And so if you think about that problem then I think it makes sense that it's been increasingly a part of AI over the years, and in some ways machine learning has started to think of it as being a mainstream part of what we do. Cool. Hence, why it's worth talking about today. Okay. Sound good. Gotcha. Okay. Let's, let's try to make this concrete with a very simple sort of example.

3.2.3 A Simple Game 1

Okay, Michael, so here's a, a nice little concrete example to, to think about this. Let's pretend that we're no longer in a world of a single agent like we've been thinking about with reinforcement learning, but we've gone full-blown generality to two agents, [LAUGH] okay? And let's call those agents a and b, and they're going to be in a very simple game where a gets to make a choice. And then b gets to make a choice. And then a might be able to make a choice. So this tree that I've drawn over on the right is going to capture the dynamics of this specific game that I'm imagining. So, these little nodes here, these circles represent states. And we can think about those in states in the same way that we. Talked about reinforcement learning in the past. The edges between these nodes represent actions that one could take. So, this should look familiar, this is just basically a game tree like anyone who's taken a, an AI course might have seen. Okay? I guess

so. It doesn't look like a very interesting game. No. But I guess it's a, sort of abstract example. Yes. It's a very simple game just so that we can get a handle on some basic concepts. So, in particular, if you look at the details of this game, you start out in state one. Ok? And A gets to make a choice between two actions, going left or going right. If A goes right, goes right, she ends up in state three. If she goes left, she ends up in state two. Regardless B gets to make a choice. From state three we can choose to go right, and really that's all that can happen. And this, what happens if B goes right from state three is that, a value of plus two is assigned to A, okay? All of these numbers at the bottom, a the leaves here, are going to be values or rewards if you want to think about 'em that way that are assigned to player A. And, in fact, for the purposes of this game, it's going to be the case that B always get's the opposite of what A get's. So, if A get's plus 2 then B get's minus 2, if A get's plus 4 then B get's minus 4, if A get's minus 1, B get's plus 1, does that make sense? Yeah, though could you write it down so that I won't forget? Okay, that's fine. So, by the way, this is a very specific type of game. here, and it has a name, which I want to get right. This is a two-player zero-sum finite deterministic game of perfect information. So as a, as a title or description of, of this kind of game, does this make sense to you? Do you think you know what they all mean, what all those words mean? So, two players because it's a and b, zero-sum. Because you said the leaves are a's rewards and b's reward is the negation so if you add the two rewards together you're always going to get zero. That's almost right. [LAUGH] Ok. It's not exactly right. Actually, so zero sum really just means that the sum of the rewards is always a constant. And that constant needs to be zero. It doesn't need to be zero. So if it added up to eleven, that would still be zero sum? If it added up to eleven everywhere. Yes. Huh, okay, interesting choice of terminology. finite, I don't know, everything seems to be finite here. There's no infinite number of choices or states or depth. Mhm. deterministic, well, again, thinking about it in an MVPish kind of way, there's no sort of casting transitions in this particular. Picture. Right. So if I'm in, state two and I go right, I always end up in state four, period. Right. Mm-hm. Game. I guess, a game is because it's more than one player? Sure. Of perfect information. That doesn't quite sound like the same terminology that we used in the empty MDP setting. But, I'm wondering if that's like, I know what state I'm in, when I'm making a decision. So, it's like a, like an MDP as apposed to a POMDP. Well it, that's exactly right. It's, it's that you know what state you're in and. Yeah. That's exactly what it means. It's like being the MDP versus the Palm DP. That's a great analogy. Cool. And does it matter that it's a tree like this? because when we were looking at MDPs, we had more complex structures of graphs and things. Well, you can think of this as unrolling the MDP if you want to. So then those states are sort of time stamped and history stamped. For, yeah, for the purposes of this discussion, yes. And that's a perfectly reasonable way of thinking about it. But, okay. But in general, we're going to be thinking about game trees, but actually, we're going through all of this for nothing, because we're going to discover pretty soon that none of this matters. [LAUGH] but, give me a couple of slides to get there, okay? [LAUGH] Sure. Okay. So this is about the simplest or at least the, the least complicated game that you can think about. Two players, zero sum. Finite deterministic game of perfect information. You know, basically, I can look at this tree, I know everything I need to know, and I can make decisions about what action I might want to take in order to maximize my reward. Okay? Good. All right. Now, in NBPs, of course we had this notion of. policies, right. You remember what a policy was Michael? Mapping from states to actions. So in the game theory world, we have something very similar to, policies. We call them strategies. So, all a strategy is, is a mapping of, [LAUGH] all, of all possible states to actions. So for example, here's a strategy, that A might have. When in state 1, go left. And when in state 4, also go left. Seems like a terrible strategy. Does it? Well, just, if nothing else, just in state 4. Sure, but it's a strategy, right? Okay, but it's just, it's a strategy for one of the players. Right, exactly, each player has a strategy. And that makes sense, right? Before when we talked about a policy, and mapped [UNKNOWN] to action, there was only ever one player, only ever one agent. And so we didn't have to worry about what other strategies there were. Here, when we talk about a strategy, it's always with respect to one of the players of the game. Okay, so, question. I've just given you one strategy, which is what A does in all the states A could potentially end up in. How many other strategies are there for A? For A? Okay, that sounds like a quiz. That does sound like a quiz. Let's make it a quiz.

3.2.4 A Simple Game 2 Question

Okay Michael, so here's the quiz, because you made me make it a quiz, I decided to make it even harder, so I want you to tell me how many different strategies are there for A and how many different strategies there are for B. And you are talking about just deterministic mappings only, right? Well, we are in a two player zero sum, finite, deterministic game of perfect information. So, [LAUGH]. So yes, I mean deterministic. Okay, alright. because, you know, it could be that it might be helpful to be stochastic. Oh, that's true, that's true. So in fact, that's a good point. I was going to mention that later, but l'll mention it now. What I just

wrote down here, is actually not just a strategy, it's something called a pure strategy. Hm, it's always about purity with you. It is, purity of chocolate and bacon mainly. But, yes. So, these are simple pure strategies. Okay, so how many pure strategies are there for A and B. Okay. Okay, then let's go.

3.2.5 A Simple Game 2 Solution

Okay, Michael, you ready? Yeah. Alright, what's the answer? So I was thinking about A before you threw in the B. So let me, let me not think about B yet. I'll just think about A. So you had said in state one, it can go either left or right. And in state four, it can go either left or right. So boy, that sounds a lot like two times two equals four. Yes that's exactly right. But generally speaking, so actually walk me through that again Michael. How did you get two times two? So, well I, I had a little choice about how to think about it. One is that in some sense, if you go right from one, then you don't really have to make another choice. Right. But if you go left, then you have this other choice to make of either left or right. So it's, you if, if you're just writing it down as a mapping from state to action, you've got two choices at state one, and two choices at state four. Mmhmm. And so that is two times two, right? You can make, independently choose each of those. Right, that's exactly right. So in fact it's important there that even though if I can gone right on one, I would never, have to make another choice because I can't reach state four. In order to be a strategy, you have to basically say what you would do, in all states where you might end up. Okay, that's fair. Okay, alright so what about B, using that incredible reasoning? [LAUGH] So yeah, so B seems a little trickier because here it can only ever matter whether you're in like if player A sends us down to the left then we have a choice of three. If player one sends us down the right, we have a choice of one, which is really no choice at all. Mmhmm. You can have any color car you want as long as it's black. Yeah, like my Tesla. I was thinking the Model T, but maybe T stands for Tesla. T does stand for Tesla. So, by the definition of how many different you know, sort of reachable strategies it would be one answer but, if you're defining it the way you're defining it, it's going to be three times one or, three. Yeah. And that's exactly right. Good Michael. So, let's actually write down what those strategies are.

3.2.6 A Simple Game 3 Question

Okay, Michael so I have written out biased on upon your rather impressive way of thinking about it. All the possible strategies for a. And all the possible strategies for b. A, can go left, left, left right, right left, right, right. For states one and four respectively. And b can go Left right, middle right or right right. Right. Right. In particular, three in stage three we can only go R. Which makes me a? Conservative? A pirate. Oh. That makes more sense. It does make more sense. Okay. So, now why did I write it this way Michael? Well, I wrote it this way because if I write it this way with all the choices, all the strategies that b has up here and all the strategies that a have here. I actually form a matrix. And I can put in each of cells of the matrix the value of taking a particular from a and a particular strategy from b. Does that make sense? Yeah, that's very clever. Yes, it is very clever. I'm very happy that I came up with it entirely on my own. Okay, so let's start filling in these numbers. Or, instead of filling in the numbers ourselves, we could ask the students to do it by making it a quiz. Nice, I see. Shall we do that? Sure doesn't seem very hard. Okay, so let's make certain everyone here understands and what exactly we're asking you to do. We're saying that if for example A takes this first strategy go left and stay one and left and stayed four. And B takes it's first strategy which is go left and stay two and right and stayed three. What is the value for A that will result? So, let's actually do the first one as an example and ask everyone to do the rest of them. That seem fair? Yeah, that's what I was going to suggest too. Okay, good. So let's see. If A chooses to go left in state one, since A goes first, we'll end up in state 2, right? And then in this first strategy. B goes left in state two, which means we will end up going down this path and the value there is plus seven. So seven is in fact the value of this game with respect to A. Now we know that because this is a two player zero sum finite deterministic game of perfect information. That if a gets a value of seven, b gets a value of minus seven. So we could write minus seven here, seven comma minus seven here. But since we know that it's equal and opposite let's just write down the value for a. Okay? Just for compactness-sake. That seem fair to you? Yeah. Okay, cool. So with that in mind, let's see if we can fill out the rest of this table. Think you can do it? I think so. Okay. Then. Can you give me a place to write everything? It will magically appear. Go.

3.2.7 A Simple Game 3 Solution

Alright Michael. Tell me the answer. let's go across. Alright. Across. So. The first row it's all left, left, but the second left doesn't matter because that's in state four. Oh it might matter. I take it back. So left, left you

did was seven. Left and then, player two goes to medium, middle. Middle. That would be three, so then the pay off is three. Mm hm. If the first player does left, left and the second player does right, right; then we're going to go left, right, left again, or for a minus one. Mm hm. Oh wait, hang on. Left, right, left. Yes, huh, yeah. The next row should be exactly the same as this one, except for in that third case. So it should be seven, three. And the r, the difference there. The r in four only matters in this case where we've gone to the right state in [INAUDIBLE], sorry, right action from state two. And so that should give us a plus four, so four. Correct so far. All right half way there, so, all right now, now, we've got something different so now player one is going to the right, so now actually player one's choice in state four never matters, so the next two rows are going to be identical to each other. And so in the first case, oh, actually [LAUGH] all of these numbers are going to be identical to each other, so player one goes right, player. Or, sorry, player A goes right, player B has to go right. There's no choice. So we're going to get 2s, no matter what now, so it's going to be two, two, two, and then the second row will repeat that two, two, two. Okay. So there you go. Michael, you are correct. You can feel very good about yourself. Very good, very good. So you're right, Michael, we do have we do have this nice little matrix and we have these numbers, and you know what's really interesting about this? That we can figure out what the payoffs are for B without having to do any additional work. That's true. But it's even more interesting than that. Which is, now that I've written down this matrix, nothing else matters. This whole tree, all these rules. None of it matters. It's irrelevant. Wait, wait, but. Okay. Because everything about the game is captured here in this matrix. This is actually called the matrix form of the game, and it comprises everything you need to know about it. It doesn't matter ha, what it means to go left or right or what it means to go middle or whatever. The point is by following this strategy and this strategy, you end up with seven for a. This strategy and this strategy, you end up with two for a, period. And how you got there does not matter. But it does creep me out a little bit that what your saying is that all that matters is the matrix. You know we should write a movie about that. I think it's been done and it's scary. Well, not the first one. It was kind of scary. Like people are trapped in this big box and their slurpyness. But their happy. But they don't know their happy. What do you mean they don't know their happy? [LAUGH] What is means to be, Michael, Michael, Michael. Speaking of which now how do we, how do we figure out. What do we do with this now? So that's the big question right? So, what was the whole point of doing reinforcement learning? The whole point of doing reinforcement learning was to optimize your long term expected reward right? Yeah. And so you pick the policy that would get you to the best place possible, so here's a question I have for you. Give that this is everything we need to know, this little matrix of values. These are all the policies A could choose from. We're calling them strategies here. But, you know, strategies, policies. There are four A can choose from. And there are three B can choose from. What will you think A and B actually do? Okay, so, you know? A wants the rewards to be high. So, seven is the highest, so A should choose the upper left corner. But A can't choose the upper left corner. Why? A can only choose a strategy. B gets to choose some strategy. I see. So A is choosing the row and then B gets to choose the column. Yeah. So then B should choose that also because that's what A wants. No, B wants to maximize what B gets. And remember, B always gets the opposite of A because. Because it's a two-player, zero-sum, finite game of perfect information, deterministic something. So, so you're saying that if we, if A chooses the first row. Say. Mm-hm. Which is the left, left strategy, and B now has a choice between three values and will choose the one that is worst for A, which would be the minus 1, so that would be a terrible thing to do. Yep. Okay then, so then what if A chooses the second row? Okay if A chooses the second row? So now again B is not going to let them have that seven which is kind of sad, but still cant make it to bad for A so B would choose the middle right. Strategy. Mm-hm. And then get, and then that would be a 3 for A, a minus 3 for B. But wait, hang on. So that was done thinking of it as A move, A kind of making the choice first. Mm-hm. That doesn't seem fair. Okay, well then do it the other way. Alright, so if B chooses first, then B could choose the far right column because that's where the minus 1 is. Mm-hm. That's where it's going to be happiest. See, B seems kind of, kind of mean. It's like it's happiest when others are suffering. Well, but A is also happiest when others are suffering. I see. So, if B chooses that column then A wants to choose the second row and get the four, so B could choose the middle column, which then A would choose the two. A would choose what? One of the twos. One of the bottom two rows. No he wouldn't. Oh, right. A is trying to maximize, so A would choose one of the top two rows. Oh, yeah, that makes more sense. Right. Then choose a three which is better for B than having chosen the far right, and B could choose the far left column, but then A would choose one of the sevens, and B would be unhappy with that. So we'd end up with B choosing the middle column and A either choosing the top or the second row. Mhm. So that's kind of the same answer we got the other way. Yep. Huh. That's exactly right. And is that, was that just luck, or did you make this example to make that happen? No, I didn't make this example to make that happen. In fact, the process you went through is exactly the process you would expect to go through, and you will always end up with the, with the same answer. For a two player

zero sum game. You know, deterministic, finite, all those other words. [LAUGH] The process you just went through, is exactly the process you would expect to go through. So let's see if we can, if we can be a little bit more, you know, explicit about the process you went through.

3.2.8 Minimax

So, in particular, the way I heard you write it down was that A must consider the sort of worst case counter strategy by B right? I see, because when A chooses the row, then B was going to make things bad for A along that row, so that's the counter strategy you mean. Right, and in fact, when you try to do it the other way with B. Well, B has to do the same thing. B has to consider the worst case counter, as well. And in this particular case, the way we set it up. Where the values for A. A is always therefore trying to maximize. And B is always trying to minimize A. Which works out to be the same thing as maximizing itself. Does that make sense? Yeah! I mean, other than the fact that, you know, I name my first child Max. I really wanted to name my second child Min. [LAUGH] That actually would have been pretty cool. Why didnt you do that? Because I'm married to someone with more sense then i have. Yeah i understand, i completely understand. Okay so, A is trying to maximize B is trying to minimize they both have to consider the worst case that the other will do. And so whats thats going to is going to force them through exactly the same process you went through. We just figure, I'm going to make the choice so that my opponent makes the counter choice, the worst case choice. I will end up as best as I can. So A is going to basically try to find the maximum minimum, and B is trying to find the minimum maximum. Hmm In fact that strategy has a name, or that way of thinking about it has a name. What do you think it's called? The minimum, maximum. Yes, or Mini max. Which is movie production company, I think. No, that was Miramax. Do you recall that where you have seen Mini max before Michael? In some other class that you once taught or once took years and years ago? Mmm. No. Mini max was exactly the algorythm that we use for game search. And intro to AI. Oh, which was a game tree, which is just what we started with in this case. Even though we turned it into a matrix. Exactly. So in the end, we, you know, this matrix induces a game tree, if you want to think about it that way. Or, game tree induces a matrix, if you want to think about it that way. And the strategy then, in sort of basic AI search, and the strategy in game theory is Minimax when you're in a two player zero sum game of perfect information. So is there a way to do alpha beta pruning on this? All alpha beta pruning does is gives you a more efficient way of finding the answer. I see but it's the same answer no matter how you set it up. Right. Cool. That's right. Okay, so this is pretty cool. So, we have set up a kind of game where we have multiple agents, you know, or at least two agents in this case, who have different strategies. And we actually sort of know, [SOUND], you know, sort of, in a world where it's a zero sum game, and you know the other person is trying to minimize what you get, maximize what they get. That the mini-max strategy would actually give you sort of an answer, in this case, by the way. We say that the value of this game for a is three. If a does the rational thing, and b does the rational thing, that is trying to maximize their own value, you will end up in this situation. That's kind of cool, don't you think? Very cool. I feel like there should be a theorem. There is in fact a theorem. I'm going to write it down.

3.2.9 Fundamental Result

Okay Michael, so here's this theorem that you, you so desperately wanted. You ready? Yep. I'm going to read it to you, because you can't read. In a two player zero-sum deterministic game of perfect information, minimax equals maximin. Alright you told us what minimax was, but you didn't tell us what maximin was. Well maximin is like minimax, except the other way around. So a is trying to. [LAUGH] You know, one side is trying to minimize the maximum, the other side is trying to maximize the minimum. Okay. It's exactly what we described before, just depends upon whether you're looking at it from a's point of view or b's point of view. Oh, I see, like, which, do you choose a column first or do you choose a row first? Exactly, so whether you go a first followed by b, or b first followed by a. You're end, going to end up in the same, with the same result. And that, more importantly, or at least as important, there always exists an optimal pure strategy for each player. In other words, you can solve those games and you know what the answer is. Once you write down the matrix. You just do Minimax, or you do Maximin and you end up with the proper answer. And now you know what the optimal players would do. Now there is a subtlety here which I got it a little bit in the previous slide, when I talked about rational agents. And what we're sort of assuming in everything that we discuss here is that people are always trying to maximize their rewards, okay? So we define the, the reinforcement learning problem that way. That my goal is to find a policy that maximizes my long term expected reward. You know so I'm trying to find, to get the best reward that I can. And what you're assuming here is that everyone else is doing the same thing and they're assuming that everyone else

is doing the same thing. Okay. DOes that make sense? It does thought I'm a little bit stuck on this word optimal at the moment. Right. Well, that's what I'm trying to get at actually. That optimal here really means I'm maximizing the reward that I can get, and I'm assuming everyone else is doing the same thing. And I'm, furthermore, I'm assuming that they're assuming that everyone else is doing the same thing. So, so I guess I'm wondering whether. Whether this theroem is vaccus in a sense that are we defining optimal to be mini max. What we're defining optimal to be, I think. So that's a good questino. I think I would unroll the vaccus one level by saying this. Optimal here, basically has to be optimal in respect to what? And the respect of what here is the underlying assumption that everyone is trying to maximize their rewards. And that everyone knows this. So, in a world where you have perfect information. It's zero-sum. Then, the strategy of Minimax and Maximin give you the same answer. And that furthermore there is always some place where the column and the row cross, the best column and the best row cross. And that is always going to be the solution to that particular game. Now, if we weren't in a two-player zero-sum deterministic game of perfect information, that might not be the case. But in a case where we're in this sort of simplest version, where everyone's being. Rational, that is, optimal, that is, trying to maximize their, their own reward. And assuming everyone else is maximizing their own reward, this is the right thing to do. Now, I've got a little weasel word here as well which we're going to get to in a moment which is this notion not just of a strategy but of a pure strategy. [INAUDIBLE] There's a reason why we have notions of pure strategies because in the end as we get more complicated we're going to have to do it with impure strategies. Mmm. Okay, but are you with me so far? I think so, yeah. So basically all that stuff we did in AI with game trees and search is kind of what you would expect people to do if they knew everything. [LAUGH] So, So, I feel like I could prove this theorum in the case of trees because you just prop, you just kind of commute values from leaves up to the root. Yeah. And, it, it doesn't matter. There is no notion of who goes first or who goes second. So there's just going to be one answer, to that process. It's not obvious to me, how to show it, if you, once you've. Turn the tree into a matrix, that that matrix, I guess because it captures the same information, it ought to be the case that this is still true, but like, I'd have to kind of sit down and think it through. No, and it's, so, so, to help you think it through, I guess what I would suggest is if you have the matrix, you can create a tree that's consistent with it. Because every row and every column represents a strategy. You don't know what that strategy is, but you can, since it's a finite matrix. You can construct a tree that is consistant with that major. In fact, possibly an infinate number of them, I'm not sure, but you can certainly construct at least one that is consistent with it. And then once you have the tree you just do what you said. Good. Alright, good. So we've got this fundamental result and now what we're going to do is we're going to try to be a bit more interesting. But it is important to go through this because now we've got some basic vocabulary and some basic building blocks okay? Yep. Alright.

3.2.10 Game Tree 1

So here's another game tree. We have, again, two players, a and b, and a gets to make a choice, first, to go left or right. And b will find herself in a situation, perhaps, where she gets to choose to go left or right as well. I've drawn these little square boxes, though, to represent chance. So what this means is that if a goes left, you end up in a chance box where you flip a coin and 50% of the time you end up over here, and 50% of the time you end up over here. Along a similar vein, if a goes right and then b goes left, then you end up in another chance node, and 80% of the time you end up here, and 20% of the time you end up here. Alternatively, if b goes right, you always end up here. Does that make sense? I think so. Uh-huh. So what we've done is we've gone from a two player zero-sum deterministic game of perfect information to a two player zero-sum non-deterministic game of perfect information. Okay, so we've relaxed, we gone at least one level up in complexity. Okay, so do you understand this tree? Do you understand this setup? Yeah, I think so. I mean here the stochasticness is happening essentially at the leaves. What does that mean? That there's no choices to make for either player after this randomness happens. But is that. That's not what you mean in general, right? No, that's right. There could be, after here, you end up in a different state where you can then make more choices. But because I don't have enough room, maybe because I want to make it simple, it just sort of ends after that. Okay. But yes, this tree could keep going on and there could be choice, random, choices chance nodes happening everywhere. There could have been a chance node that happened at the very beginning, even. I feel like I could work out the value of this game. Oh, well, how would you go about working out the value of the game? I would probably have it as a quiz. Okay, but what would the quiz look like? It would say, what's the value of this game? Okay. Do you think that just having that be the quiz is the best way for someone to learn or do you think maybe it can be done in stages? Oh, I don't know. Do you have other questions that you want to ask? Well, you know, how would you go about determining the value of the game? I think the first thing that you would do, at least if you were patterning

after what we just did, is you would try to write down a matrix. Sure. So, I'm going to write down a matrix. So that our students will get a chance to, to work out what that matrix looks like and then from there figure out the value of the game. Are you okay with that? Sure. Okay. So if we were going to do that of course. The first thing we'd have to do is we'd have to figure out, figure out what the strategies are. So. What are a's strategies. I would have called them left and right but it, but they're not labeled. Yeah well let's do that we can call them left and right. A can go left or A can go right. What about B's strategies? B can go left or right. And B can go left or right.

3.2.11 Game Tree 2 Question

Here's your first quiz question. In a world where A can go left or right and B can go left or right. What are the values that you would put in the cells of this matrix? Again, this is zero sum. So, these values are from A's point of view and implicitly, there's we know what the values are for B. So, we're just looking for the values from A. Okay? Do you understand the quiz, Michael? Yep. Okay, so go.

3.2.12 Game Tree 2 Solution

All right Michael, you know the answer? Yep. Okay, what's the answer? Do you want to know the value of the whole game? No. I want to know the matrix. [LAUGH] Okay. So right. So, if A goes left, it doesn't matter what B does. And at that point there's a chance node, and it's 50/50 for negative 20, which I feel like ought to be, negative eight. That's right. How'd you get negative eight? because half of 2 is sorry, half of 4 is 2 and half of -20 is -10. So -8. Right, so you just took the expectation of where you would end up. Exactly. Okay. What next? And that, it doesn't matter what b does. So then negative eight is also in the upper right corner of the matrix. Fair enough. The next easy on to do is if both go right. A goes right and b goes right then we get the three. It's just sitting there. Mm-hm. And the last thing requires me to do some multiplication. So Rrr. -5 times 0.8 Is like -4. Mm-hm. 10 times 0.2 is like 2 so -2. That is correct Michael. Shoo. So now we have the matrix. Now, what did we just notice here? Remember what I said about matrices before? It has all the information that you need. Right, so in fact none of this matters. Who cares how we got here? wait, but, but, oh. [LAUGH] I love erasing stuff! So here's the thing, we cannot reconstruct that tree from this matrix. We can't reconstruct, well, actually we could reconstruct that tree from the matrix. No, we can't. Yeah, we, of course we could. No, we can't. Yes, we can, because there's an infinite number of trees we could do, and one of them is that one. I see. We just don't happen to know that its the right one. I don't think that's really what people mean when they say I could reconstruct this. Like I could reconstruct the crime at this crime scene, it could be any of a million things. Like no, we can't construct that specific tree. But you know what it doesn't matter, because the only thing that matters is the matrix. Ahh. And you will notice, Michael, that you did all that multiplication and you multiplied 0.8 times 5 and after a couple of seconds of thinking about which was probably edited out but I know how long it took you do. You came up with the right answer. That's great. But notice that the number you came up with doesn't say anything about expected values and what the probablities are. It doesn't even matter that it's nondeterministic, because once you have these numbers, these expected values, that's what you're going to end up with. So who cares what the original tree was? Who cares if you can reconstruct it? Who cares what the rules of the games are? All you know is I have a choice of two strategies. We call them left and right here, but we could have called them one and two, or Q and Z. It doesn't matter. For the purpose of solving the game. So what is the solution for the game by the way? Oh, A is trying to maximize right? Yes. So A would never, ever, ever, ever want to go left. Mm-hm. So A's going to go right and then B is trying to minimize. So B is going to go left and we get the -2. I believe that is correct. And it makes sense the other way as well, right? That B would not ever choose to go right because then A would choose to go right as well. Gotcha. So you'll still end up here.

3.2.13 Von Neumann

So, here's in fact another theorem for you Michael. So, it turns out the theorem that we wrote down before is still true in the case of non-deterministic games. Oh, cool. Of perfect information, right? By the way, do you know whose theorem this is? Charles' theorem? No, although that would be cool. Von Neumann's theorem? Yes, this is von Neumann's theorem. Oh really? Yeah. Did you just guess? Von Neumann, he's responsible for everything. Do you know, well, you going to remind everyone who von Neumann is? He's my uncle? No. No, you're right. You're right. Von Neumann, so we talk about von Neumann architectures in computer science, that the basic design of a microprocessor is still following the ideas that that he worked

out. Right, the von Neumann Machine. So there you go. So, this is pretty good, right? So, what we did, so what did we learn so far? What we learned so far is, the only thing that matters is the matrix. And once you have the matrix, you used mini max, at least if you're in a two player zero sum game of perfect information. At least if you're in a world of two player zero sum games of perfect information. We can just write down this matrix, throw everything else away and use mini max or maxi min and figure out what the value of the game is, which is the same thing as saying as, we know what the policy is. The kind of joint policy, in a world where everyone is being rational and trying to maximize their rewards and assumes everyone else is doing the same. That's pretty cool, I think. Awesome, so though, you know, I feel like I could make a matrix that wouldn't have this property. This maxi min equals mini max. And, and that we couldn't make a tree out of it. You probably could, but I'm going to guess that it's going to require that it's no longer zero sum. Nooo, I'm going to say no to that. Well, you know what, I, actually now that, I think I understand what you mean and the answer is yes you could. And in fact, that's what we're going to do next when we relax the next little bit. Ooh. So, would you like to do that? Sure, though I'm afraid if we get too relaxed, we might just fall asleep. In fact, why don't we do that, why don't we take a nap and then come back? [LAUGH] the, the joy of being on video, asynchronous napping. Done. Okay, I'll see you in a second.

3.2.14 Minipoker

Okay Michael, so here is a, another little game for us to play. And, what I want you to notice about this game before I describe it to you in detail is that we have relaxed yet another one of the constraints. So, we started out playing two player, zero sum, deterministic games of perfect information. There's also a finite in there somewhere. From now on we're just going to assume everything's finite, because why not? And then, what we did last time. Just a few seconds ago. Is we relax the deterministic part, so we have two player, zero sum, non-deterministic games of perfect information, and now we are going to relax the requirement for perfect information. So now we're going to look at two player, zero-sum, possibly non-deterministic games of hidden information. And this is really important Michael because, this last little bit of relaxation, going from perfect information to hidden information is going to be sort of a quantum leap into the difficult problems of game theory. So this is where it actually starts to get interesting so we have been building a foundation so far and now we are going to get to the interesting and complicated stuff, okay? Wow, one of the things that I learned just now is that the opposite of perfect is hidden. Yes. I always thought the opposite of perfect is imperfect but okay but hidden. I guess if I do not have a perfect face I should hide my face. Thats in fact the atomology of the phrase. Alright, I understand now. Yeah, cool, It's in wikipedia. Here we go, let me describe the game to you, are you ready? This is a version of mini poker where there are a set of cards, but they have no numbers of faces on them they are just red or black, okay. And red is bad, for our hero, Player A, and black is good for our hero, Player A. Okay? I see. So– Wait. Why's it have to be red? I don't know, man. You know, red. You know how it is. Okay. So, here are the rules. They're all written down on the screen, but let me walk through them with you. So A is delta card. Magically. It will be red or black. and, the probability of it being red or black is 50% each. Okay? Yes. Right. So we have a uniform prior over, over the color. Now, remember red is bad for A and black is good for A. So, it's going to turn out without loss of generality, that if A gets a black card, A's definitely going to hold onto the card. Okay? Now A gets this card. B, player B, does not get to see the card. A can choose to either resign or to hold. If A resigns given a red card, then he looses 20 cents. Okay? Okay. Wait. So A's dealt red. A may resign if but only if red. Right. And then A loses 20 cents. Right. Okay. Alright, okay. Okay, so this is a betting game. It's not strange it makes perfect sense its sort of a metaphor for life. Now A can choose to hold instead, hold the card. Thus requiring B to do something. So if A holds the card B can either resign or can demand to see the card. Now if B resigns then A gets 10 cents. Regardless of the color of the card. Okay? Yep. That make sense? Yep. Okay. Now if B chooses to see the card, in fact demands to see the card. Then if the card is red, then A loses 40 cents. But if the card is black, then A gets 30 cents. And since we're betting, this means that whatever A wins, B loses and vice versa. That makes it zero sum. Got it. Okay, is this all make sense? Yeah, I don't know if I can hold all those different numbers in my head. But I, but the basic pattern of it is, that as you say Red is, red is bad, black is good. If A gets a bad card, A can essentially either fold Mm-hm. Right, resign? Or kind of bluff, like hey I've got this great card and then A and if B believes that the card is bad then and calls A or, or, or, and, and just folds then, then A gets, A wins that. But if B says no I think maybe you're bluffing and calls him, then everybody's rewards are more extreme, I guess. ¿Exactly. So it's just a version of poker. A weird version of poker. Simple version of poker. But a version of poker, none the less. There's a minor little detail here which isn't that important, but you know, notice it's written that A will A may resign if its red. Basically, A will never resign on a black card. Because it just doesn't make any sense. And so there's really, it just, there's not point in riding it out. Okay. Because

black is always good, sort of, nothing bad can ever happen to A, if A gets a black card. So there's really sort of no point in riding anything out. But that's just a minor detail. Regardless these are the rules. Okay? Okay. You got it? Sure. I'm going to re-draw this as a game tree which might make it a little easier to keep all the rules in your head.

3.2.15 Minipoker Tree Question

Okay so Michaell here's the tree version, of what I just said. So remember I draw squares as chance nodes. And so, chance takes a, chance. And half the time we end up in a state, where I have, where A has a red card. And half the time I end up in a state where A has, or we end up in a state, where A has a black card. Okay? If A has the black card. Then B gets to choose whether to hold or not, so let me add a little bit of color for you since you wanted some color. This is the place where A gets to make a decision. Okay. Yep. And then this is the place where B gets to make a decision. Got it. Okay? So, A is going to either be in a red state or a black state. Only A knows this. B does not know this, does not know what state He is in. And so let's say A is in a black state. Well, A can only hold in that case. In part because it makes no sense to resign. And then B gets to decide whether to resign, and therefore A gets ten cents or to see, in which case A gets thirty cents. By contrast, when we're in the red state A can either hold or resign. If A resigns he loses 20 cents. If A holds then B gets to decide whether to resign and which case A gets 10 cents, or to see the card in which case A loses 40 cents and B of course gains 40 cents. So this is just a tree version of what I just wrote. Does that make sense? Okay, yeah I can see how this kind of captures the flow of information but I feel like there might be a missing constraint to it. So [INAUDIBLE] I don't know if there's actually a constraint in this thing but let me just point out something. And that is that B has no idea which of these two states his in, and its hidden information. And because B doesn't know what states he's in. He doesn't know whether resign over see. In particular B knew that he was in this left mode state then he would always say see. If B knew he was always in the rightmost state, then he would always resign. But he doesn't know which one, so it's not entirely clear what to do. Neat! Where did you get this game? This game is awesome! . I got this game like I got all of the examples that we're using today, from Andrew Moore. He's clever. He is clever. And I have shamelessly stolen... All of his examples and materials for this. But he said it was okay, by putting up slides in the world and saying, feel free to steal all of the stuff. You want to write his name so that people can credit him? Yeah, I'm going to write it at the very end. When we talk about. Okay. What we've learned. Awesome. Okay. So here's a question for you Michael. We know that I wrote down a bunch of words which describe the game, and then I made it a tree, because I could do that. And it makes it nice and easy to see what's going on. But now we know, that at least everything else we've done. We want to make a little matrix. And so we want to figure out what the value is for different strategies for A and B. So I'm going to assert, and I think it's pretty easy to see, I hope. That A basically has only two strategies. Either A is the type of person that resigns when a card is read. Or A is the type of person who holds, when a card is read. Agreed? Interesting, okay. Yeah. Right. So it really is a conditional policy, right? It's basically If red hold to resign, if black hold to resign, but your point is that black isn't really a choice, red is a choice, and there's only two choices. Okay I can see that as the two strategies sure. Right and of course this does say, this is, this does kind of say when you're in the black case you know you're going to hold. So you know what's going to happen. Ultimately B can either be the kind of person who resigns, whenever A holds or chooses to see, whenever A holds. Right? I see so, like in the previous trees. B would have four different strategies Resigner see in the left state, resigner see in the right state. Here there's this kind of extra connection, this sort of a quantum entanglement between these two states that it, that makes them have to be the same. So there really is just those two choices. That's exactly right, although, I probably wouldn't have used the phrase quantum entanglement. But yes, there's an entanglement, certainly. because we can't tell which state we're in. You're either going to resign or you're going to see. And you just don't know what else to do. Okay so A's a resigner or holder. B's a resigner or a seer. So the question is what numbers go in this matrix? And we're going to figure that out by having a quiz. I kind of saw that coming. Mm-hm, do you think you can figure this out? yeah, yeah. Sure. Sure. Yeah, I think so. Yeah, see, see, see. Yeah, I can do that. Okay, cool. So, let's go then. Go.

3.2.16 Minipoker Tree Solution

Okay Michael whats the answer? Let's start with resigner resigner. Alright, resigner resigner. Resign resigner diner. Alright, E, so resigner vs resigner. So resinger, if A is a resigner that means whenever A gets a red card A resigns. Yes. And that would be a negative 20. Yep. But that's not going to be the answer is it. Because A doesn't always, A doesn't always get a red card A sometimes gets a black card if A gets a

black card [CROSSTALK] than a resigner, resigner that means B is going to resign and, and a plus 10 will happen. Mm-hm. So those are the two possibilities and they're equally likely, so it's a minus 10 divided by 2 which is minus 5. Right. You were correct, sir. Whew. Alright. Which one do you want to do next? Resigner seer. Okay. So again, oh, so the, yeah, good. This is a good choice, because now it's the same as argument as before, except for when we end up in that far right node, and that means minus 20 in half the cases, plus 30 in half the cases Which is a plus 10 divided by 2, or plus 5. That's exactly right. Well done Michael. Thanks. Okay which one next? So holder resigner. So holder reigner. That means when A, gets a card. A is going to hold the card. Mm-hm. And that's true, red or black. Yep. And then, then, it's going to be B's turn, and B is going to, oh, we're doing holder resigner. So, it's going to resign. Mm-hm. So, oh, well, interestingly, I think that takes us to those two leaves, both of which are plus ten. Yep. Because, why does that make sense? Because B, oh, 'cause B doesn't get any, no. Right, right, right, because it's independent of the card. You actually said that when you explained the rules. Mm-hm. So its, its average of plus 10 and plus 10, which ought to be plus 10. It is in fact plus 10. Well done. Okay what about holder's here? Whew, alright, so that's a case where A holds so we go down those branches and we end up, we always end up in one of the blue circled states. Yep. And B sees half the time that leads to minus 40, half the time that leads to plus 30. So that's minus 10 divided by 2, which is minus 5 again? Yeah, that's exactly right. So that's correct, Michael. And that's pretty cool, isn't it? Yeah, I really like this game. I didn't think you could have anything that had sort of poker essence and be this tiny. So yeah, so we live in this really nice little structure here. So I have a question for you. You ready for the question? I'm trying to guess what it is, but sure. Okay, here's my question. What is the value of this game? I was thinking that you might ask that. So, can I, can I step through it, is that okay? Yeah, sure, go ahead. So a's choosing the first row or the second row. So if a chooses the first row, and then b chooses the column, then if it's the first row, then b is going to choose the first column. So a's going to get minus 5. Mm-hm. The same story's going to go through on the bottom row. If a chooses the bottom row, then b's going to choose the seer position which gets the minus 5. Right. So from this so, it seems that the value of the games minus 5. But now let's do the same thing on the b column. So if V, resigns, now a gets to choose resign or holder. And it gets a plus 10. And if B is a sire. Then A chooses between re signer and holder and gets plus 5. Yes. So then from this perspective, the value of the game is plus 5. So, so here's a case where it better not be that we could take a perfect information game and put it into a matrix and get this out, because this is something that can't, like, it doesn't, it doesn't fit your theorem, right? We can't get the value of it by doing minimax or maximin. Exactly, the problem here is that once you move the hidden information. Minimax is not necessarily, and in this case, definitely is not equal to maximin. So von Neumann. Fails. As we all see. Idiot. Yeah, what has, what has he ever done for us, anyway? His theorems and his computer architecture that rules the world. Anyway, so we seem to have a problem here. And the problem is that once we go to this hidden information, as I promise complexity enters in Michael, and now we can't do something very simple with the matrix, find a pure strategy that's going to work. It really is the case that A strategy depends upon what B will do and B strategy depends upon what A will do. And if you don't already know what that's going to be, you don't actually have a value for the game. And in some sense you can get every single one of these values. So, there's got to be some [INAUDIBLE]. I feel like, that's sort of what you'd expect in a game like this. Right. So, if, because of the way that it is. If you know that I'm always a resigner. That I'm always going to, what? [LAUGH] Oh, that when ever I have a red card, I'm going to resign, then. You know that if I don't resign, I have a black card, so you know that you should resign. Mm-hm. Yeah, so it's, it's, it's one of these things where if I am really consistent and never bluff, say, then you can take advantage of me, and vice versa. Like, if you always respond the same way to when I act a certain way, then I can manipulate that. So it's kind of like this game of this sort of mind game like I want you to not know that I know the thing that you don't know that I don't know. Right but the problem is that I know what you know and I know that you know what I know. And that I know that you know that I know that you know that I know that you know that you know what I know. Did you really think that I didn't know that? [LAUGH] And so you end up in this terrible situation. But. There was a key word that you used there, Michael, and it was consistent. And everything we've talked about so far, pure strategies, is exactly the same thing as talking about consistency. So, the way we're going to get around this is, we're going to stop being consistent, or at least, consistent in the same way. Okay? Yeah. And to cheat here, is that we are now going to introduce, instead of pure strategies. Let's see the opposite of pure. Is? Impure. Mixed, that's exactly right. Contaminated. No, so, rather than sticking with pure strategies, we're going to start using mixed strategies.

3.2.17 Mixed Strategy Question

So what's the difference between a pure strategy and a mixed strategy, Michael? Well, it's, it's simply this. A mixed strategy, simply implies, or means, some distribution, over strategies. So in the case of two strategies, like we have here, where you can either, A can be either a resigner or a holder, we're going to simply say that the mixed strategy for A is some value for P. Which is the probability of choosing to be a holder. So do you, you see what's going on here? So the only difference between a mixed strategy and a pure strategy. Is that for a mixed strategy, you choose some probability over all the different strategies that you might choose. So you decide that, you know, going into this I'm going to flip a coin. And half the time I'm going to be a resigner, and half the time I'm going to be a holder. Say. Or 30% of the time I'll be a resigner. And 70% of the time I'll be a holder. Okay? Yep. Where as with pure strategies, you always chose on or the other. So technically, it's the case that a pure strategy's also a mixed strategy where all the probability mass is on a single strategy. Makes sense. So in this case we're going to, in fact, choose P to represent the probability for A of choosing to be a holder rather than a resigner. And so P can be 0%, probability zero, or can be probability one, or any value in between. You with me on that? Yeah, that's neat. Okay, good. To make certain you understand this I'm going to give you a little quiz. Which I have up here on the screen. You ready? Oh, I see it. [LAUGH] It's like those, those very square boxes. Yes. I didn't even realize what, what, what this could be about. It certainly couldn't be a quiz because Charles has never drawn a straight box in his life. I had drew those Michael. It took me 17 hours. Oh man, you are, you're committed to this and I appreciate that. I am committed to this. Okay, so, given that we have a mixed strategy, and we have a probability P of A being a holder, here's my question for you. In a world where B is a resigner, okay? B is always going to choose to resign. What is A's expected profit? To make it easy for you, I copied the matrix over here in the upper right hand corner. Wait, wait, wait. If B is always a resigner. B is always a resigner. Then, what's, and what is A? A is going to choose to be a holder with probability P. Oh, so you want this to be a function of P. Maybe. If it's, if it's, okay, sure, maybe [LAUGH]. It could, well, I mean, yes. It's a function of P, it just might be a constant function that ignores P. It could be, in principle. Okay, now, after you figure that out, I want you to decide, in a world where B is the seer, B always chooses to see the card. What would A's expected profit be, in a world where A will choose to be a holder with probability p. Okay? Hm. Got it? And that's going to be, oh yeah, that could be different because it's a it's a different strategy, though you know seering. Yes Like like if you're a resigner you're a resigner, but if you're a seer then what you are doing is seering Mm-hm. That would be an anagram of resigner. How do you see these things? I don't know. They're just there. I just, they words just kind of mix themselves up. Alright anyway, I'm, I think I am ready to do the function. I'm, I'm ready to stop looking at the letters. Okay? And go!

3.2.18 Mixed Strategy Solution

What's the answer? If B is the resigner, we don't really care about the other column anymore. Mhm. Then what's going to happen is A is mixing between resigning and holding. Yap. And probability P is a probability of being a holder, so whatever P, whenever that event happens it gets ten, and whenever one the opposite of it happens it gets minus five, so I want to say ten p plus one minus p five. Okay, is that your. Let me try that again. Is that your final answer? Ten p. Well I could simplify it. Okay. Well then say it to me again. I'll write it out here so it's easy for you to see. Okay and that is. My answer? That's fair. Do you want to simplify it? You don't have to. Sure, let me simplify it, so it's a minus minus p five and a ten p so that's fifteen P minus one. Minus what? One. Minus what? Five. Uhun, there you go. Thank you. That's correct. We would obviously accept either answer or any combination of those letters. That, [LAUGH] No, I think I might have to do this quiz over actually. No, not the one. I'm talking about either 15 p minus five or ten p minus one minus p times five. Or ten p plus one minus p times minus five. Or any other combination that we can get push car to actually Bothered to, you know. Check. Im, Implement or check. Okay. Well, that was pretty good. And this, of course, is exactly the expected profit. As you put it, P times A as a holder and P times or P percentage 1 minus P percentage A chooses to be a resigner. And so it's just the weighted average between those two values. So, Kent, let me just double check that. So, if P is 0, that means it never holds, it means it always resigns and it gets -5, so that's right and if P is 1, means it always holds, so it should get a +10. And 15 -5 is 10. So, boom! Yeah, it works. You used math there, very good. Okay, what about B? B. So the same story, except on the seer side. Mm-hm. So yeah, I might need that space again. So 5, oh I see, right, minus 5 times p. Mm-hm. Plus 5 times 1 minus p. Mm-hm. If we simplify that we get. There's a minus 5 and another minus 5. So we get minus 10 p. Mm-hm. Plus 1. Plus what? Plus 5. There you go. See you learned. Okay you want to check it? yeah. Oh, that's a good idea. So again if P is 1, then that means you're always which should be the minus 5 and if we put in a 1 there, we get 5 minus 10 is minus 5. And if P is 0, that means that we're always a resigner, and we should

get a 5 for that. And yeah, so we zero out the negative 10 and we get the 5. Exactly. Now, it's not clear to me why we're playing this game. Oh, it is clear to me why we're playing this game, because we want to figure out something about. Strategy that is mixed. Right. So this is how well a mixed strategy does, but not against another mixed strategy. This is against two deterministic strategies. But is it, Michael? But is it? So I'm going to, oh, okay. I'm going to notice something here, which is that you, as you astutely pointed out earlier, we have two functions. Of P or to equations of P, and by the way, do you know what they are? They're lines. Sure, because it's just a, it's linear in P, so that's what linear means. Right, so what would happen you, do you think if we were to actually draw these lines? I think we'd have two lines. Yes, and what would those two lines look like? Let's take a look, shall we? Sure.

3.2.19 Lines Question

Okay, so what I've done here Michael, is I have drawn both of these lines. So here's the line 15 p minus 5. This is a case when b is a resigner. This is my best attempt at drawing it to scale. You start out minus 5, as you point out, and you end up with plus 10. And this is the line where b is a seer. That's minus ten b plus 5, so you start out with plus 5. And you end at minus 5. Okay? Cool. So, what do you notice about these two lines? They make an x? So they intersect. They do intersect. Can you tell where they intersect? How would I solve that? You'd have a quiz. Okay Michael, so here's a quiz. Where do the two lines intersect? Think you know how to figure it out? Yep. Okay. Then go.

3.2.20 Lines Solution

Okay Michael, what's the answer? I don't know, but I would, here's how I'd get it. I'd set the equations of the two lines equal and find the p where the, where those values are equal. Oh, okay. So here let's do that. And I feel like that's like 25 p on one side and ten on the other Okay. So like 10 over 25, which is like 2 over 5, which is, like, 2 over 5. Yes, which is also 0.4. So 0.4, that's exactly right, and it's, you should do it exactly the way you said. Set the two equations equal to one another, do simple algebra, and. So what would have happened if p ended up being, like, not a probability? Then they wouldn't have crossed inside. Good point.

3.2.21 Center Game

So by the way, here's a question for you. We can make it a quiz but I'm not going to make it a quiz. What is the value of the game at p equals 0.4? So I just plug it into those equations, and so negative 10 times 0.4 is like, negative 4 plus 5, is 1. So I'm going to say 1. Mm-hm. So the value here is in fact $0.01. And if you put it in the other equation you get the same thing. Yes, right, because they're equal there. That's actually the answer to the entire kit and kaboodle. If. Sorry, where were the kittens? If A chooses a mixed strategy, where with probability 0.4 he chooses to be a holder. Notice that it doesn't matter whether B is a resigner or B is a seer. You will end up here and there will be a value of plus 1 penny to A. On average. The expected value is. Yeah. plus 1. Neat! Now you might ask yourself, well what if B decides to be a smarty pants and also do a mixed strategy? What do you think would happen in that case? So of course it changes something, but it doesn't change the value of the game, because B, if B is a resigner, A is getting one on average. If B is a seer then A is getting one on average, and an, any average, any convex average of one and one is going to give us one. That's exactly right. So in fact if you think about it, if B tries to do a mixed strategy between these two lines. It's going to have to be for every single point between the two lines, somewhere the average is going to have to be somewhere in between there. No matter how you weight that average. So it's like a bow tie is this space of possible payoffs. Right and since for any let's if I just, If B decided to pick some, you know, some values such that it's in this region of the space. And it's going to end up somewhere, say between here that's fine, or here that's fine, or here that's fine, or here that's fine, or here that's fine. More importantly, right here it's going to have to be between the two lines. Where those two lines cross. So, no matter what mixed strategy B chooses. On average we will end up here, so the value of this game is the expected value of this game and it is, plus 1 for A. Okay so hang on why is that not there are other values that can be obtained. There is a minus 5 a plus 5, true. Why is it plus 1? Well its plus 1 here on average, the expected value of this game. Is plus 1. So you say that the strategy for A is to choose .4. Mm-hm. But why? Like, why is that, of all the different values. What [INAUDIBLE], just because the lines intersect. I don't understand what makes that a good idea for A. Like, it's not like it gets any additional payoff for having things intersect. Well, if a is going to choose a mixed strategy, and b is going to choose a mixed strategy. This is the mixed strategy for a, that guarantees an expected value of plus 1. Meanwhile,

let's imagine this is the way you're going to set it up. So we've been kind of talking in general that a's going to choose the strategy; b's going to choose a strategy, and they're going to go. And that, because we know everyone's rational, you already know what strategy b's going to pick, and b already knows what strategy a's going to pick Right, because we made this assumption about everyone is trying to maximize their own utility, there own reward. So in a mixed strategy, if you know you're going to go for a mixed strategy, you have exactly the same situation. B can figure out well what is it that A should choose and A can figure out what is it that B should choose. So notice that in this particular version of the game, the way it's setup. Even if A announced beforehand to B. Listen I am going to choose to be a holder with probability 0.4. It doesn't matter what B chooses the expected value is this +1. Right? Yeah. But imagine if A said I am going to pick, I'm going to choose to be a holder with probability 1. Well then, what should b do? B should choose to seer. Right. It's exactly the situation we were in before. Okay, but here's the thing I'm having trouble wrapping my head around. So, it's not special that it's an intersection, what's special maybe, is it that, you know, if b is always. Okay; taking what you said before, that a is going to announce a strategy. So for anything that A can announce B is going to presumably do what's best for B which is to just minimize right? Mm-hm. So if you look at that triangle at the bottom. Mm-hm. If you think about that as those lines that V, upside down V shape. Yeah exactly that thing. If you think about that as being the payoffs, the payoff function for A, as a function of the probability P that it chooses to hold. Mm-hm. Then we've chosen the maximum. Right. Right, we're trying to find the, the probability for which A gets the highest expected value. And it's at that, it's at that peak there. Hm-hm. But notice something, Michael. For any case where the two lines cross, you're going to have a function of this form. Well, let's be sure of that, because that's, I wasn't seeing, I was thinking that that was not true. because basically in this case, we have a, a line of positive slope and a line of negative slope. Mm-hm. Right? So, what if we, we can have an intersection between two lines of negative slope. huh. So again, by doing the same exercise you were doing before, where you draw the, you sort of take the minimum of the two lines at all points. Yeah. You'll end up with this. Where do you pick it then? The far left, p equals 0. Yeah. But not the intersection in particular. No, that's true. Okay, alright, I just, wasn't quite getting that. But so, the intersection is special, Because in some cases that is where the max is. It can only I guess the max can only be in those three places, right, it could be far left far right or the intersection. There' s no other way to have a maximum of lines. Right. And b by the way there's another case like this where they never actually cross. Mm. Or like this where they never actually cross. Got it. Right, but in either case, right, it's always going to be those three points. It's going to be the extreme or where they cross. If they happen to never cross, then that point goes away and you just pick the maximum. So what you could, so, in fact, if you wanted to think of an algorithm to choose the right thing to do for A, you basically plot the lines. You take the min at every point and you find the maximum point. Oh I see, so you could just kind of discretized it almost. Yeah, and you're done. Seem reasonable? Well discretized seems problematic but like I get that we're trying to find the probability so that we maximize. Oh wait we maximize the minimum of the two other things. So it's like maximin again. Yeah. Except here, so, in fact, it is exactly min and max or max and min, except, oh, I guess, if you're thinking from A's point of view. It's min and max or max and min but in this case, there's this other parameter, which is the probability. Which is how you determine how you're doing the min and max. Got it, right. In this bigger space, it's min and max. I see, yeah, yeah, yeah, that makes sense. What about so, why is A the one who needs to be random? Why isn't B the one who needs to be random? It's, you end up with the same, you end up in the same place. Because again, remember, you're, you're doing this kind of optimal notion, where underneath all of that is this belief that both people are going to be rational. So the only reason to do this, to take the, you know, maximum minimum, is if you believe that B is always going to try to minimize what you do. But B's in exactly the same situation. Right, from B's point of view, That's what's happening. Huh. It's the same equations? Yeah, it just it gets turned around. That's actually an interesting exercise. Maybe that's the next homework assignment. Okay. So, did that all make sense to you? [LAUGH] [CROSSTALK] It looks a little green and scribbley, but yeah. Sure. That's cool. That's how you know when you're done, Michael. When it's all green and scribbley. [LAUGH] Does this generalize to more than two options? It seems like it's going to get messy fast. Yes, it does. It generalizes to more than two options. Effectively now, you're just doing a max over n things, instead of two things. And you have to search for, there's possibly way more intersections to worry about. Yeah, but all you care about is the minimum. Think of it as you're always drawing the minimum function.

3.2.22 Snitch 1

Okay Michael, so, feels like we've relaxed almost everything we could relax except for one thing. We're now going to look at two player non zero sum. None possibly none deterministic games of hidden information.

Cool. And that's going to turn out to be messy, but get us to the place where I've secretly been trying to get us all along. So, let me describe a game for you. Very carefully, and let's see where it leads us. Okay, here's the game, you've got two people. These two people are criminals. Oh no! Are they smooth criminals? One of them is. [LAUGH] [MUSIC] [LAUGH] Oh, you're terrible. Okay, so we have two people. They're criminals. Okay, and unfortunately they've both been captured by the cops. I have actually no idea how to draw that, but let's just try to draw that like this. The cops come along, and capture them both because they are suspected in a particular robbery. Okay? Hm. And they take them and put them both in jail. So those are jail bars. [LAUGH] Okay? But they don't just put them in jails. They put them in two separate jails. Oh no. Okay. And one cop goes to one criminal and says listen, here's the deal. We know you did it. Okay? We know you did. And your friend over there, he's currently singing, singing like a bird. And he, is going to pin it all on you. So this is your last chance to give us a little help and admit that you two did it. Or, that the other guy did it. You admit that the other guy does it, you say it was his fault, then we'll cut you a deal. Now to make it worse, the cop tells him that there's another cop over there. [LAUGH] Who's talking to the smooth criminal, and is offering him the same deal. Hm. And whoever, goes first, whoever pins it on the other guy first, gets to walk. Okay? Hm. So, if I can get the curly-head guy to defect, okay? Hm. Then, I am going to let him walk and he will spend zero months in jail. Okay. Okay. On the other hand, if the other guy defects, then he's going to get to spend zero time in jail. And since we've got all that we need. We got a confession. To get this guy to go to jail. He's going to go to jail for nine months. Okay? Negative nine months? Yeah. Oh, that's a cost in months. I see. Uh-huh. [CROSSTALK] Yeah that's a cost in months. Okay? So if, curly-head guy defects before the other guy does, then zero. He walks. He pays no cost, other than the cost he's already paid for a life of crime. [LAUGH] Now if he refuses to drop a dime on the other guy, but the smooth criminal decides to defect he's going to lose nine months. That make sense? And the other guy's going to walk. So he's getting the same deal, they're both getting the same deal. You got it? Yeah. All right, now. It's a little more complicated than that, all right? But I, I hope that you're getting all, keeping all of this in your head. It's a little more complicated than that. There's actually four choices here, not just two. It's not just a case of. I defect or the other guy defects. I defect, the other guy doesn't. I don't defect, the other guy does. Okay? You could both refuse to drop a dime on the other. You could cooperate, or you could both rat out the other. At the same exact moment? At the same exact moment. So, there's a big thick wall here. Curly guy doesn't know what smooth guy is doing, and smooth guy doesn't know what curly guy is doing. Right. So the question is, if you have a choice between either defecting or. Which means blaming the other guy? Which means blaming the other guy you defecting from your friendship. Oh. Or you can cooperate, that is be true to your friendship. I can, we both people can defect, both people can cooperate, one person can defect and the other person cooperate, so there's actually four different options there. I fee, I feel a matrix coming on. Yeah, I think there's a matrix coming on. So let's draw the matrix, because I think that makes it easier to see. But you have the background here, right? I think so, yeah. The key piece here is that each person has a choice of either defecting from their friendship or trying to cooperate. Keeping their mouth shut, and, they don't know what the other person is doing. So, what are all the costs here? What are the worst case things? Let's just draw it all out as a matrix.

3.2.23 Snitch 2

So let's just call the smooth guy A, because that's what we've been doing all along. And he can either choose to cooperate, or he can choose to defect. Now by the way, I'm saying cooperate and defect here because they're terms of art, not because these are the words I would have chosen. Okay? B can do the same thing. B can choose to cooperate or B can choose to defect. Now I've set up the game in a particular way here. The, the, the cops have set up the game in a particular way here. If B defects but A cooperates, then B gets to walk and A has to pay the price, nine months in jail. So, I'm going to put into this cell minus 9, 0. So, this means this is the value of this set of strategies for A, and this is the value for B. So the first number. In the pair is what A gets and the second number's what B gets. Okay? Got it. You with me? Yep. Okay, now notice we have to do this now because it's no longer zero sum. It's not going to always add up to a constant. Oh. Well, it is, so far it's a constant negative 9. That's right, oh and in fact it looks this way as well because I have a symmetric deal here. So if A defects and B cooperates, A gets to walk and B goes to jail for nine months. Okay? Yep. So right now, you're right, it's looking like a zero sum game, but it isn't. Because what happens if both A and B drop a dime on each other? Well, they both confessed. So it's a little good, it's a little bit better than having one of them, only one of them, confess. The deal that the DA is willing to give them is that both of them will spend six months in jail. Now these are just, these are just, the, the numbers that are a part of the game. This is, I'm not computing this from anywhere. This is just the way the, the cops have set this up. And I'm going to, now what's an interesting question here is what happens if both A and

B keep their mouth shut? They both choose to cooperate? It turns out that it's not a perfect world because they were caught with a bunch of guns they didn't have permits for. So if neither one of them admits to robbing the bank, they're still going to do a little bit of time. But in this case, it's a small weapon's charge and so each only spends a month in jail. I see, so now it's definitely not zero sum. Mm-hm. Because we have a negative 2 there, a negative 12 there, a negative 9 and a negative 9. Right. Okay. Okay. So, you got the game, you understand it? Yeah, I think so. It's very simple. Now, just looking at this, what's the best possible outcome for the duo? So if they cooperate with each other, then there's you know, one month later, they're back on the streets, back to their criminal activities. Or they're reformed. You never know. Sure. Back to their choice of whether to have criminal activities. The, the mutual defection one, that, where they both defect, it's either, well, 12 months or six months, depending on how you think about it, but they don't do nearly as well. And then the defect and cooperate, it seems like there's a lot of incarceration that will happen. Mm-hm. So, it feels like the best for the, for the two of them is to mutual cooperate. Cooperate, cooperate. Yeah. And that makes sense. This is sort of what you want to happen. Both of them keep their mouth shut. They do a little bit of time, but on average, they do pretty well. Right? Yep. So my question to you is, is that going to happen? Sure, why wouldn't it happen? Well, you tell me. If I know that you're going to cooperate. Let's say that I'm A, okay? And you're B, and I know that you're going to cooperate. What should I do? You should cooperate. Should I? You've chosen, you've chosen this column. All right? I see. Well, if you think of it that way then it's not a joint choice, but it's actually an individual choosing. Then you're better off defecting because then you get off scot-free. Yep. Then, but I go to jail for nine months. I could have a whole baby in that time. Usually takes closer to ten but sure. But you go to to jail for nine months, I don't. So if you are just that cold. I guess because you're a criminal. It doesn't matter. Remember, the matrix is everything. The value of doing this to me is 0 versus minus 1. Alright, that's, that's cold, man. I agree, but it's what the numbers tell you. What if it wasn't criminal? What if it was just, you know, the amount of money that I was going to win in a, in a mini poker game? Yeah, but it's a different kind of mini poker game, because we're both, because. [LAUGH] It's like you beat me. But, by beating me that way, you like, kill, nearly kill me. So? Are you saying that when? What do you mean so? Are you saying you always let me win whenever we play poker? No. Okay then. Because you're cold, is that what you're saying? No. Wait, what? Exactly, right. So the point, Michael, is that if I know you're going to cooperate, you're going to choose this column, then I should defect. Okay, alright, so, fine. So now I'm going to jail for nine months. Mm-hm, if you choose to cooperate. Aha. If, if is good. So what you're saying is, I could drop a dime on you. [LAUGH] You could. So, you could choose to defect, and if I knew you were going to choose to defect, what would I do? Well, you already, you already showed your colors, man. You're, you're defecting, so I'm just switching, I'm just saving myself three months by, by ratting you out. Yeah, so we would end up here. By the way, you know that this game is symmetric, right? no. You defected first. [LAUGH] No, here's the thing. Since the only thing I care about is maximizing my own reward, my own value. I'm going to do the thing that makes sense for me to do in this case which is if you cooperate, defect. If you defect, defect. But you would do the same thing because your whole goal here is to maximize your value. Yeah, well you don't know me like that. Yes I do because it's everything that's here in the matrix. This is the wonderful thing about game theory. All the stuff that you're concerned about is all inside the matrix. Remember the rules of the game don't matter. There's no prisoners here. There are no criminals. There's just, I get a dollar, I lose a dollar, or I lose zero dollars. I lose $9 or I lose $6. Or, for that matter, cents. I lose one penny or I lose no pennies. I lose nine pennies or I lose six pennies. It doesn't matter. This is the value of these particular strategies. And I'm going to want to defect if you cooperate. You'll notice if you defect, I'm also going to want to defect. In fact, if you look closely at this matrix, you should notice something, Michael. From A's point of view, when does it make sense for me to cooperate versus defect? You mean if you have a, some kind of probabilistic policy over cooperate and defect? No, no, not even that. Just in general, is there ever a time as A that I would rather cooperate than defect or vice versa? Oh, I see. So if I know you're going to cooperate, I should defect to, to save myself a month. But if I know you're going to defect, I should defect to save myself three months. Either way, I'm coming out ahead. Right. So in fact, let's take a look at this.

3.2.24 Snitch 3

If I look at the value for me as A, between cooperating and defecting in this first column. It's minus one versus zero, right? Which number is bigger? [LAUGH], zero. Right. If I look at it in this column, it's minus 9 or minus 6. Which number's bigger? Negative six. Right. In both cases, defecting is better than cooperating. So in fact this choice, this strategy, dominates the other. In other words it is always better for me to defect than cooperate. So I will never cooperate, ever. Because it's always better for me to defect. So fine. So I'll never cooperate ever. That'll show you. By exactly the same argument, minus one versus

zero, minus nine versus minus six. Defecting is always better. B will never cooperate. What's the only thing that's left? Pain. Pain. This is called the Prisoner's Dilemma. Huh. You said it was simple, but it seems kind of evil. I didn't say it wasn't evil. Those things aren't opposite of one another. It's not like perfect versus hidden. Or pure versus mixed. It's not easy versus evil [LAUGH]. Evil's often actually easier. It's easy and evil. So we're in a little depressing place here, Michael. You claimed in the beginning, and I agreed with you, that this is sort of the best option, you want everyone to cooperate, because that's sort of what's best for the group. But because defecting dominates cooperating both for A and B you're going to end up here. At least if you reason about it that way. Yeah, I see that. Hence prisoner's dilemma. The only way to break this is if somehow we could communicate and collude with one another. So that we could guarantee that we would both choose to cooperate at the same time. So that wall? What about that wall? What if you put a like a Skype connection or a Google hangout? Well, if you were forced to say what you were going to say at the same time, or somehow be able to punish someone maybe for, you know not doing the right thing, then you might be able to make a different decision. But for this very simple version where I'm going to do it once, even if I could hear what you had to say, if one of us, whichever one of us went first, the second one would always be able to take advantage of that. I do find this kind of depressing. Yes, it's a dilemma. It's a true dilemma. So, this brings us to a more general sort of strategy. This whole notion of strict dominance, works in this case. But, you could imagine very complicated large matrces where it may not work. But it turns out there's a generalization of this notion of dominance. That works remarkably well. And that's what people tend to use to try to solve these kinds of games, to find out what the true value of a game is. So letme describe that for you, ok? Okay.

3.2.25 A Beautiful Equilibrium 1

Alright Michael. I'm going to define a new concept for you. It is called the NASH EQUILIBRIUM. Nice. Okay, here's the set up. You have n players. So we move beyond simply two players. You have n players. And each player has strategies that it can chose from. So here I'm referring to them as, S1, these are all the strategies for player one. As two are all the strategies for player two up to S N all the players for strategy N. Got it? Yep. Okay so each of these are set, so im going to say that the particular strategies, S1 star S2 star S3 star, so on and so forth as N star that is a strategy that each of the N players has chosen. Is in a Nash equilibrium if and only if for each one of those strategies chosen by the n players it is the strategy that maximizes the utility for that particular player, it is the strategy that maximize the utility for that particular player Given all the other strategies that were chosen. Now I actually find that difficult to kind of work through, so let me try to say it a different way. Given that you have a set of strategies as one star, as two star, as three star, as n star. We know that they are in Nash equilibrium, if and only if. If you randomly chose one of the players, and gave them the chance to switch their strategy, they would have no reason to do it. Interesting, okay. So does that make sense? Yeah. Right. So an equilibrium right, just in general, the word, the word sort of makes sense, right, an equilibrium is at a place where everything is balanced. And in some sense, there's no reason for anything to move because they're in balance. So we set that a set of strategies are a Nash Equilibrium if no one person has any reason to change their strategy in a world where everyone else's strategy remains the same. Then you're, then you're in equilibrium, and in particular you're in a Nash equilibrium. Nash equilibrium. Okay good. Do you know who Nash is? Ogden Nash was a poet. Not that Nash. I think there was a TV series with a Nash in it. Yes there was. But you're thinking of John Nash. That's right The Nobel Prize winning person who was the, was featured in the movie A Beautiful Mind. Yep, that's exactly right. And it was, in fact, a Beautiful Equilibrium. Okay, so, there's the notion of a Nash Equilibrium. It was admitted by John Nash, if you believe the movie, in order to pick up women. Which I, you know, you know I never, I've never seen this movie. Oh really? Oh, I watched it and it was, it was surprising how unhelpful it was in explaining what a Nash Equilibrium was. [LAUGH] I'm not surprised. I hope this is helpful, though. So, it really is a kind of difficult concept to completely wrap your, your mind around, but what it really boils down to is, listen, if we all picked a bunch of strategies and we knew that one other person, one of us, we don't know who before hand, but we know that one of us Would have the opportunity to change their strategy after they see what everyone else's strategy is. We'd be in a nash equilibrium only if that person, whoever that person is, has no reason to change their strategy. Gotcha. So, so specifically you've made a distinction between strategies that were pure and strategies that were mixed. Right. And I guess I don't see in this case which kind we're talking about. Right. So, in this particular case, we're talking about. Pure strategies; however, exactly this wording works with mixed strategies. Oh, I see. So you could have a pure Nash equilibrium or a mixed Nash equilibrium. Right. And so, you could, each one of these, instead of choosing a particular strategy and saying they're a Nash equilibrium, you could talk about a probability dristribution over each of these sets of strategies and say those are Nash equilibrium if.

No one would want to change their probability distribution. Got it. So this works for both pure and mixed. Alright, so you think you understand this? [LAUGH] sure. Good, we will test that.

3.2.26 A Beautiful Equilibrium 2 Question

Okay, Michael. So you think you know everything, so here's a quiz. So, here are two matrices, matrix prisoner's dilemma and matrix bunch of numbers that look vaguely symmetric but aren't quite. So here's what I'm going to ask you do do. In each of these cases find the Nash equilibrium. Rut-roh. So do, you told me what it was but you didn't tell me how to find them so that seems kind of rude. It was implicit. It's intuitively obvious even to the most casual observer. Oh, well then, okay. So, and just to remind me now, each row and column is a choice for one of the players or the other player. Yeah. And the first number in the pair is the row player, or A's pay off and the second number is B's pay off. Yes. And we might need probability distributions, or we might not, depending on what it takes to be a Nash equilibrium. Right. okay, I don't [LAUGH] I'm not, it's not like I see the answer yet but it, but I either will be able to find it, or I won't. At least I understand the question. Okay. And by the way, just to make it easier for you there are pure Nash equilibria here, okay? Hm. Alright, so you ready? Yep. Just circle the one or underline it or whatever it is Pushcar does to make it so that you can answer this quiz. You ready? Totally. Go.

3.2.27 A Beautiful Equilibrium 2 Solution

Alright Michael, you got the answer? Yea, I'm ready to try and figure it out. Alright let's go. Let's try the first one. I feel like the first one's going to go rather well. So we need a pair of strategies so that no player is. Motivated to switch. Mm-hm. And you told us they were pure strategies, so we actually have a nice algorithm for doing this, which is we could just check one of them with the definition. Mm-hm. But, but in the case of prisoner's dilemma, I think a natural place to start would be that minus 6, minus 6. Mm-hm. So let's say that A chooses the second row and B chooses the second column, let's see if that's a Nash Equilibrium. So both players need to be happy. So if, would A be happier switching? If A switched, it would be getting minus 9, which is worse. Mm-hm. So A is happy, where A is. And if B switches, B would be getting minus 9, so boom, Nash Equilibrium. Done. Now I didn't verify that other ones weren't a Nash Equilibrium, but you didn't say. Find all of the Nash Equilibrium. You just said find, well you did say find the. So you kind of implied that there's just one. Right. But actually that's true, but you don't have to check this because we already went through an exercise where we, where we knew the answer was minus 6, minus 6. And the way we did that was we noticed that for A defecting the second row is always better than this. Right? And then in particular this row strictly dominates this row. Right. Which implies that if I picked anything on this row I would rather move to the other row. And you can see it. Minus 1 0 is better. Minus 9, minus 6 is better. That's what it means to be strictly dominated. So, I'd never pick this row anyway. And this same argument for B for this column. So, you'll notice that by getting rid of the things that are strictly dominated, the only thing we're left with is this. And, it turns out, in fact, to be a Nash Equilibrium. So, this is correct. So, you just told me how to do my job, which makes me little sad. Well, in this instance. Because I already had the answer to this. But maybe, maybe you were signalling to me how I might attack this next problem. Maybe. Maybe. So, A is choosing a row and gets the first number. So, is there a row, that, were they dominates all the other rows? It doesn't seem that way. Mh-hm But maybe, maybe, oh I could start with B, maybe B, there's a column that dominates all the other columns. But no, it looks like it's totally symmetrical. Yep, so strictly dominated doesn't necessarily help here, and by necessarily I mean doesn't. So that was kind of mean. Thank you. All right, so oh, but we have something else we could do. Yes. So, there is a, the largest number that anybody can get is 6. Mm-hm. And there's a play where both of them can get the 6. Yeah. So there's no way they're going to want to switch away from that, because everyone's getting there kind of maximum out of reward. So A bottom row, B right column, gets us Nash Equilibrium. And that is in fact correct, and you can see it because from here I would always, it would always be worse for me, and it would always worse for me. So, these are in fact the Nash Equilibrium, for these two problems. Cool. They've seemed easier than I was expecting. Mm-hm.

3.2.28 A Beautiful Equilibrium 3

So here are three, fundamental theorems that come out of all of this work on Nash equilibrium. They're all in the screen in front of you. I'll read them for you in case you can't read my handwriting or if there's some typo you discover that forces me to erase some words and then write some new ones. Okay, are we ready? Alright, the first one is, in the n-player pure strategy game, if elimination of all strictly nominated

strategies, eliminates all but one combination of strategies, then that combination is in fact the unique Nash equilibrium. So that's, that's what happened in prisoner's dilemma, we got rid of all but one option and that option had to be the unique Nash equilibrium. You say, eliminate all of them, do we, is it possible that things that we couldn't eliminate in one round, we could eliminate in the next round? Yeah it's possible, so in fact, this is done in an iterated fashion. You get rid of whatever you can eliminate and pretend they were never there. Because the truth is, no one would ever choose them. And then with what you have left, you just do it again. So it is possible. Although not in any examples that we did. Okay. The second one is and this, both of these I think are kind of Obvious. They sort of make sense anyway. At least to me. That any nash equilibrium will survive the iterated elimination of strictly dominated strategies. In other words if you get rid of things that are strictly dominated you will not accidentally get rid of nash equilibria in the process. And that makes sense because, if they're strictly dominated then if you ever end up there, you would want to leave. And therefore can't be a Nash. And therefore can't be a Nash equilibrium, that's right. So, those two things sort of make sense I think. The last one is true but isn't at least painfully obvious, at least not to me, and that is. If n is finite, that is you have a finite number of players, and for each of the set of strategies, that set of strategies is also finite, in other words you're still in a finite game, then there exists at least one Nash equilibrium which might involve mixed strategies. So you're saying there's always an Nash equilibrium? Yes, possibly mixed, for any finite gain. So, when you said I relaxed everything, I didn't actually relax the requirement that it be a finite gain. Fair enough. But you stopped writing finite, so it's sort of the same thing. Yeah, I agree. Because why would you play an infinite game? I don't know. Maybe you got a lot of time on you hands. Mm. That would be like an infinite jest. [LAUGH] Okay. So those are the, the main, results here. So, what did we learn? What we've learned is that with interesting games. We end up in these weird situations where we have to figure out how to solve the game and one really nice concept, is this notion of an equilibrium and in particular the notion of a Nash equilibrium. Cool. And there you go.

3.2.29 The Two-Step

Let me just wrap up by doing a couple quick things. Earlier on when we were talking, you made a kind of offhand comment about, or maybe I made an offhand comment, about the fact that because we're doing this little prisoners dilemma thing, we have this sort of wall here, we're going to end up in this little bad place. Right? With the -6, -6. And there was this kind of notion that what if we could hear each other, would that make a difference? And the answer was, no not really, because who ever goes first is going to lose. And if you try to get people to go at the same time, well in some sense, that's the same thing as having a wall and you can't hear each other if you have to go at exactly the same time. Right, does that make sense? I didn't quite understand. Why does the first, person who goes first lose? Because if I find out whether you're going to cooperate or defect, then I will just then do the thing that makes sense. So if you do anything but defect, I will defect, and then I will win and you will lose. So whoever goes first runs the risk of the other person screwing them over. So. I see. Unless, unless that person defects. Right, but if that person defects, the other person's going to defect. Right, so, but you don't lose any more than you would have. That's true, and, but you still end up in this kind of unfortunate place. So, one question you might have there is, yeah, but that's just because you can only do this once. But what if you could do this twice? So, let's imagine were, going to be in this situation today and then were going to be in this situation tomorrow because, you know, you keep leading me astray. So, whatever happens today I can use as information, for what I might do tomorrow. So, if I decide to go ahead and cooperate this time. And you defect, and I know you're the kind of person who defects, and so when we're in this situation again, I am definitely going to defect. So maybe in that situation, it's in your best interest to go ahead and cooperate because, then we can keep coopoerating together, and in the end it sort of works out. So maybe this notion of not just playing prisoners' dilemma once, But playing prisoner's dilemma twice or three times, or four times, or five times, might come to a different result. Yeah, that seems plausible. I mean, I could imagine saying to you though this channel in the wall, something that says, you know, cooperate with me or, yeah, right, or I will, I will stop cooperating with you. And so, you get more reward if you, if you cooperate with me. Right. So, my question to you is, what does happen exactly in this set? Where I have two versions, two consecutive games of prisoner's dilemma to play. What am I going to do? Ok, well now I mean, we could turn this into a game, right? I mean, that's what you were teaching us how to do. Yeah. So you, so now the game has four, well, I don't know. So one way to do it is to say A has a choice to make on the first step, and [COUGH] that could be cooperate or defect, and then the second step could be cooperate or defect, so there's four possibilities. Right. B has the same four possibilities. But maybe we can actually have A be responsive to B. So A has two things to do on the first step and then two things to do on the second

step for each thing that B did on the first step, for a total of eight combinations. Right. You could do that. [CROSSTALK] So if we made an eight by eight matrix, we should be able to solve it. Right. So, that's pretty easy to do. You just kind of draw an eight by eight matrix in, you know, that many dimensions. And it, you know, you end up right here. I wasn't thinking about that many dimensions. I just thought, like the normal kind of matrix. But, okay. Oh, I see, I see, I see. I mean its eight by eight, so that's kind of a pain. Right. So, its 64 cells and, I guess, I guess we don't want to fill that out. Yeah, but you know. We could fill that out. But, I'm going to help you out here by pointing out You know what? It's not going to make a difference. Oh. So, here. Let's see if I can walk you through why it's not going to make a difference.

3.2.30 2Step2Furious

Let's imagine I have 20 of these games. Okay? Sure. So I'm going to play these games 20 times in a row. Now what you've been doing is you've been going forward. You've said, well, if I can get us the right thing or if I can basically threaten you and say. If you screw me over this time, I'll screw you over from now on. Then maybe I can get you to do the right thing and vice versa, since it's a symmetric game. And then we'll end up doing the right thing all along. And that makes a lot of sense, right? Yeah. If you're going forward. But what if you're going backward, Michael? Let's imagine we are doing [CROSSTALK]. What is amazing to me is that is exactly backwards what you just said. So, let's imagine we are doing these 20 dot dot dot dot, and now I'm on the 20th game, okay? What's going to happen on the 20th game? Well, I mean, I guess I could have built up trust in you from the previous games and think that you're going to cooperate with me, because we've been cooperating together. And then that would be the perfect time to drop a dime on you. Exactly. So, if this is the 20th game and we're in this situation, remember whatever value we've been adding up along the way, that's sunk cost. Right. The only thing that's left is the final game. And the final game looks like this, which means we're going to end up here. But guess what, Michael? The final one is determined. So since we already know the outcome of the final game, the only one that we, the next one that we can look at is the 19th game. Well, we already know what the outcome of the final game is. So this is effectively the last game. Oh. And what is the outcome of that going to be? It's going to be this and backwards, backwards, backwards, backwards proof by induction because the only that proves computer scientists know how to do is proof by induction. It turns out that we will always defect For what it's worth, those are the only proofs worth doing. But, okay. That's true. It's a fine point. Truth by induction and perhaps by conduction [LAUGH]. So, there you go Michael, even if I can play multiple, multiple games to try to build up trust, the truth is the Nash equilibrium, if I filled out that eight by eight matrix you wanted me to fill out. I would still end up in the same place where I would defect both times. Certainly if, yeah okay, so if you're going to be a jerk then, then I have to be a jerk, but yeah, I guess that's right. That's, that seems pretty horrible. It is. And by the way this is not just something I'm making up. This is another theorem that comes out of Nash equilibrium. That if you have an n repeated game then the solution is n repeated, Nash equilibrium. So whatever the Nash equilibrium is for the first game, the one version of the game, is the repeated Nash equilibrium for all versions of that game. Okay, wait a, hang on, hang on. So I. Okay, I buy that. Mm-hm. But couldn't you, well so, in a game that has more than one Nash equilibrium, couldn't we like, al, alternate? We could, and in fact you'll notice that in so far, I have not talked about the case where you have multiple Nash equilibrium. Because in fact, that's it's own problem. If I have one Nash equilibrium, then we know what's going to happen. If I have two Nash equilibria, which one do we choose? Now, what's important to know here, is that if we have two Nash Equilibria then that means that if you're in any one of them, you won't leave it. So it's not like you'll move from one Nash Equilibrium to another, sort of in the general case. Because the fact that it's an equilibrium means that if everyone else is fixed except one person, that person won't choose to move. But. I haven't said anything about how you would choose among multiple Nash equilibria. And that's beyond the scope of this class. Okay. But it is an active researcher and in fact some of my own students have done some work in thinking about. What it would mean to choose and actually the answer always boils down to, let's not worry about that. [LAUGH] All right. But I'm still kind of disturbed by this, right? So it seems like it's sort of saying, is if we knew the world was going to end tomorrow then we might as well be greedy bastards. And, in fact, if we know that the world is going to end at any particular time, which of course it will. We might as well be greedy bastards. That's right. So, be greedy bastards. Oh, no! Or at least, or at least be you know, Nash bastards. [LAUGH] We'll, the Nash is kind of greedy right, it's like I'm always taking an action that's always best for me. Yes, you are. No. Every single thing we've talked about today. Has always assumed that you're always doing what's best for you. The fact that it might also be best for someone else is neither here nor there, you're always doing what's best for you. Right, I get, I get that, but it's, it sounds like this argument is saying that we might as well be like that, like all the time and never really form, like never,

never self-sacrifice even a little bit for greater advantage even to yourself. Well, that's not true. This, it's just all hidden in the utilities, once you've gone through all the what I'm going to do today, what I'm going to do tomorrow, you add it all up, whatever the right strategy is to take. That's the strategy that you take because it's already accounted for all the self-sacrfice you might do that then leads to later good deeds or later good outcome. So the wonderful thing about all of this, is that this matrix is everything you need to know. It captures the future, past and everything else, so everything you need to know is right here. And it's all been reduced to a bunch of pairs of numbers. But you did say something kind of interesting though, Michael, which I think is worth pointing out, which is that everything you say requires knowing when, at least in this sort of iterated version, requires knowing when the world is going to end. So an interesting question to ask would be, what would happen if I knew the world was going to end but I didn't know when. Would that change my behavior. It, it doesn't seem like that should make any difference because it's still going to end. If not today then tomorrow. Or if not tomorrow then the next day. True. But I bet yeah it does make a difference. I'm willing to go and think about that. Okay, why don't you go think about it and let me know. Alright. I'll, I'll tell you about it in the next lecture then. Okay, I like that. Excellent. Okay, cool. So, let's move on. [LAUGH] Okay, fair enough.

3.2.31 What Have We Learned

All right, Michael. So, I think that brings us to the end of what I wanted to talk about anyway. So, can you help me remember what it is that we've learned today? In particular, what you've learned today? Sure. So, I guess the first think I learned is that game theory can make you depressed. And, in fact, in particular that my friend Charles, given the opportunity. Would totally drop a dime on me just to save a month of incarceration. Yeah. Wait, no, no, no, I don't think you summarized that correctly. Game theory is not depressed. It's depressing. Oh, yeah. That's a good point, that's a good point. And Michael is not cruel. He is the victim of cruelty. I don't think so. Because you want to know what the secret here is, Michael? You've got a little matrix of numbers. Those number capture what's going on. The truth, Michael, is that, if we were in Prisoner's Dilemma. I would cooperate with you because my utility is not simply the number of months that I would spend in jail. But it's the number of months you also would spend in jail. Oo. Interesting. So if I, the best way to beat Prisoner's Dilemma, is to change the numbers. [LAUGH] I see. It's like the Kobayashi Maru of game theory. Exactly. So there's an interesting question for you right there, Michael. If I had prisoner's dilemma here, here, let's write it out so you can remember. If I had prisoner's dilemma here, we already know that we're going to end up here, because that's what the numbers tell us to do. But what we'd have to do is change the game. So how would you change the game in prisoner's dilemma? I see. So, if, if we're thinking about it in particular in terms of I care about how long you spend in jail. Maybe not as much as I care about how much I spend in jail. Like maybe half as much. Mm-hm. Then, the payments shift, right? Right. So, now we have like minus1 and a half, minus 1 and a half in the upper left hand corner. Mm-hm. Minus 9 comma minus 4.5, minus 4.5, minus 9 and minus 9, minus 9. So, yeah. So now that, that bottom right becomes a lot less attractive if we actually care about the other person. Right. Well that's, that's, that's, okay, I'm less depressed now. Except of course, that requires that you feel that way internally and that I feel that way internally. There's another way that you could change the game here. Which is, what happens to sniches in jail? They are rewarded. No. No they're not. They're punished. Yes. Oh. So, if your a part of the criminal fraternity, and you don't like prisoners dilemma, then what you have to do is to create a system Where the people who snitch get punished. So it's not just the months that they spend in jail, it's everything else that's going to happen to them if they drop a dime. So you're saying that minus 6, minus 6, ends up being worse? No, what I'm. No, the minus, wait, no, wait, what? Yeah. Oh, the zero ends up getting, oh I see the zero ends up being worse. Because even though your not in jail your going to get I don't know somehow thwarted or, or punished for your past behaviors. Accosted. Interesting. That's right. So that's what you have to do and that works not just with criminals but with the real world. Whenever your in this sort of situation like a prisoners dilemma, you can change the game by changing everyone's utilities. Like for example hiring police officer, police officers or hiring members of the mob to take care of everything. I see. So it almost seems like what you're talking about is a kind of inverse game theory, where if there's a particular behavior that I want to see. How do I set up the payments and rewards so that that behavior is encouraged. Right, and by the way that has a name, and it's called mechanism design. Mechanism design? Yes. I'm not sure I understand either of those words. [LAUGH] Well, that's where you're trying to set up the set of incentives, the mechanisms that you're using to pay people. You're trying to design them in such a way to get particular behavior. This is what a lot of economics is all about. This is what a lot of government is all about. Tax breaks for example, for mortgage interest, encourages you to buy a home, rather than rent a home. I see, by changing the payoff structure. Right. Oh that's neat. And

so that's what we learned today. At least right now. [LAUGH] Okay. Alright. So, let's see. So, just to try to rattle off some of the other things. The whole notion of Game Theory. We talked about, especially the idea that. You can think about a game as a tree or you could represent it as a matrix. And, I believe you said, repeatedly, the matrix has everything. Is that how you said it? Or the matrix is all you need. Yip! Let's see. We talked about minimax and maximin. Mm-hm. We, we relaxed a bunch of constraint on games. Mm-hm. So we, we looked at both perfect and hidden information. Mm-hm. We looked at both zero sum and non zero sum. We learned a lot today. We looked at deterministic and [UNKNOWN] I would want to say, but you called it non-deterministic. And assuming that we can get rid of the first two bullet items that look like maybe they were jokes. I would suggest saying things like, we talked about what strategies are and that they come in different flavors. We talked about the evil prisoners dilemma game. Mm-hm. What else? You gotta give me more, otherwise it's [INAUDIBLE]. Oh, more, good point. Andrew Moore gave lots of really good examples that yes. Michael may be cruel, but Andrew Moore is awesome. He's more cool than me. Andrew Moor is very cool. All of the examples that we've used today, or almost all of the examples we used today actually come from Andrew Moore's slides. Andrew Moore is a professor at Carnegie Mellon or at least he was before he went off to Google. And is a really smart guy who cares very much about machine learning. And game theory and produced a bunch of slides that it turns out lots and lots of people use in their own courses. And his examples were so good for game theory. That I decided to coop them with his permission of course. He tells everyone that they may use them. And, in fact, we have pointers to the slides in the resources links and folders for all of you to look at. And I recommend that you do. Did we learn anything else, Michael? The only other thing that I would want to mention is NASH which is a concept that is nashtastic. It is nashtastic. There are by the way, I should mention briefly. Other kinds of equilibria concepts they're beyond the scope of this class. But there's a whole lot more to game theory as you might imagine. And sometimes when they ask you to in these situations where you can't do what you want to do, you end up in these prisoner's lemonade situations. Other kinds of equilibria can get you out of it. And I'm going to argue without explaining why that the way that they get around this is by introducing Other ways of doing various kinds of communication. and, in fact, I claim they're a particular part of mechanism design. But that's a topic for another day. Okay. Fair enough. Okay. Did we learn anything else, Michael? I don't know. That's what I was thinking about. I mean, that seems like a lot to absorb. And, the other thing is that repeated games, even the prisoner's dilemma kind of unravel if you know when the end. And I was going to look into what happens if you don't know when they're going to end. Okay. So, I guess, that will be something that we will learn. Next time. Right, what we, what will we have will learned? Yes. Future past tense. [LAUGH] Alright, Michael, well, I think that's about it. At least my brain is full. So I will talk to you next time, and you get to lead what I believe is the last full lesson of the course. Oh, exciting. It is excuiting. Alright, well bye Michael. You have fun. I'll see you next time. Alright. Bye Charles. Bye.

3.3 Game Theory Continued

3.3.1 The Sequencing

Hello Charles. Hi Michael. How are you today? I'm doing just fine. How are you doing? Alright. I'm a little out of practice with this lecturing thing. So, I hope this goes well. I'm absolutely sure it will. So, today's lesson is continuing what you were talking about in terms of Game Theory. But, I'm going to be focusing in on what happens when you are worried about making decisions. With more than one player in a sequence. Which we started to get into at the end of your discussion but I'm going to go, more into it. Okay. I really like the logo by the way. Thanks very much. Yeah, it's a, it's a specially game theory logo. I like it very much, I like it very much. I will point out, however, that all sequels should be called the quickening. Yeah, I was going to go with the quickening. Or judgement day, but I didn't, didn't think that made any sense. Hm, that's a fair point.

3.3.2 Iterated Prisoners Dilemma

So let me take you back into what we were talking about last time. We were talking about the iterated prisoners dilemma, and here's the prisoners dilemma payoffs that you wrote down for us. You remember this? I vaguely remember this. And remember it was about these two criminals, Smooth and Curly. And they were, deciding whether to cooperate or defect against each other when, after they've been arrested. Right, and they defect, they always defect. They always defect, right. So in particular, we say well what happens if it's, if they have multiple rounds in which to interact, and so here they are, here are the two of them, and if they've got one round to live, we did an analysis and we indicated that there's really nothing

they can do other than defect against each other. Mm-hm. It's irrational to do anything else. Yep. So we said, all right, well, what happens if we allow there to be more than one round? So now we've got two rounds. And what we realized was that if you got two rounds to go, then these players essentially, face a one-round game because, after this round, what they're going to do in the last round has already been determined. So it's almost as if that round doesn't really matter. There's nothing we can do now that's going to change what they're going to do in that last round, so it's sort of like there's just one round and we're going to defect again. Right, so life is terrible and everyone is out to get everyone else. Exactly, and not only is it for two rounds, but this same argument continues as you go three rounds or more. Oh, it's like a proof by induction. It's kind of like a proof by induction, yeah, well it's proof by ellipsis. Mm, that's my favorite kind of induction. Proof by ellipsis. [LAUGH] So the question then becomes, what happens if the, number of rounds left is unknown, right? So what we've realized is that if you know how many rounds are left, the whole thing comes unraveled and they're just going to defect forever. But we raised the issue of, what happens if the number of rounds left is unknown? Hm. And it seems like it shouldn't really make any difference because if it's say some finite number we just don't know what it is, then it seems like it should still reduce to this same setup that we have. So I was looking into this and it turns out that is it's not the case it actually does make a difference and it's, it's really interesting how it goes and how it connects back with other things we've talked about. Woo, tell me more.

3.3.3 Uncertain End

So here's how I started to think about it. So let's say how can we represent the idea that we have an uncertain ending. We'll one way would be if we had some kind of generic probability distributions over the number of rounds that the games going to be played. But this seems like an, the simplest idea that I could think of. So here, here we have our, two criminals, and what their going to do is their going to play a round of prisoner's dilemma. But at the end of that round, they're going to flip a coin. And with probability one minus gamma, that will be the last round, it's all over. But with probability gamma, they're actually going to play again. And, and they do this after each round. And so each round is, is basically statistically independent of the other rounds. I see. So the set up here is, with probability gamma, the game continues. Now, notice that I chose gamma here. This was a, it's representing a probability here, but in the past, we've used this to represent a discount factor. Mm-hm. But that actually is the same thing, right? In, in, in the normal discounting, we say that the value that you get one step from now, is discounted downward by gamma. And that's exactly what you'd take if you worked out the expected value of a game where you continued making steps with probability gamma. And with probability one minus gamma, it ends if you get zero from then until the rest of time. So every round here could be your last, or not, right? It could be that you actually get to continue playing. Does that make some sense? That makes perfect sense. Awesome. All right, so, so yeah, this is, this is exactly that kind of situation where, well, here, let me, let me ask you a question. What's the expected number of rounds of this game? Well, I'll bet it's finite if gamma's less than one Yes, I even wrote that down. Yeah, I'm smart. Or at least I can read. Sure, but what's the, but, specifically we could actually write it as a, as a, function of gamma. Let's see. If gamma were something like, 99% then I would expect it to be about a 100, right? I think that's right. Yeah. So is that, is that your answer? [LAUGH] My function of gamma is if gamma is .99 the answer is 100. Yeah, something like that. It's not a total function but it's a function. Well, it's a sample. I mean, you do machine learning. Why don't you tell me what the function would be given that sample. Well, we can make it a quiz or I could just tell you. Why don't you just tell me. Alright. So one over one minus gamma is the answer. It works for your example. Um-hm. 1 minus .99 is 100th and we're talking 1 over that so, you get a 100. And, yea we could go through the argument as to why that's that's what it is. But this one over one minus gamma is what shows up all the time. If gamma is zero, then we're talking about one over one. The game lasts one round. That's exactly what we'd expect. Mm-hm As gamma gets closer and closer to one this pro, this quantities getting closer and closer to infinity. So, Right. in fact if you know, it becomes unbounded as gamma hits one. So yeah. So this is the expected of rounds, and so that means like yeah. So as you said if gamma is 0.99, it's a 100 rounds. And we already, reasoned that at a 100 rounds the whole thing falls apart. Right, huh, and I noticed the one over one minus gamma, of course, is just like the way we did discount factors, when we started doing MVP's in the first place. Exactly, yeah, that, that kind of links them together. Hm, that's actually kind of neat.

3.3.4 Tit-for-Tat 1

So if we're going to be talking about strategies in this game that has an uncertain ending, we can't just write down sequences of, of actions anymore. We can't just say cooperate, defect, defect, defect, defect. Or even some kind of tree of possibilities. Because those are going to be finite representations. We need some other representation that allows us to play for an unbounded number of rounds. Mm-hm. And I'm going to start off by presenting an example of such a strategy, one that's, that's very famous for the iterated prisoner's dilemma, and it's called tit for tat. And the structure of tit for tat goes like this, on the first round of the game, A player playing this strategy will cooperate, and then in all future rounds, the player is going to copy the opponents previous move. Does that make sense? It does. So basically, we start, I start out, acting as if, you're going to cooperate with me. And the moment you don't cooperate with me, I will start to defect, and we'll be in the, the old style prisoners' dilemma. Right? Well no, not. What this says is that it actually copies the opponent's previous move. So if, if an opponent goes cooperate, defect, cooperate, defect, cooperate, defect, defect, defect, defect, cooperate, cooperate, cooperate. You're going to see something very similar coming out of the tit for tat agent. I see, I see, I see. In fact, we can represent the strategy as a little finite state machine, like this. Yeah, I like that, okay. And you, so you can see exactly how it kind of proceeds. It starts off cooperating. And then in each, each round it waits to see what the opponent does. That's the green letters here. And then it follows the corresponding arrow, to determine whether it is going to cooperate or defect in the, in the current round. Sure, that makes sense. So in this picture, the black letters here represent my move. And the green letters represent my observation of the opponent's move. Or at least if I'm being tit for tat.

3.3.5 Tit-for-Tat 2 Question

What happens if we follow Tit for Tat? That's a great question. So, let's make it a little more concrete. So, here's a set of strategies that an opponent might adopt. And now the question is, what does it look like Tit for Tat is doing in response to each of these? And I was hoping that you'd, you know, check the corresponding boxes. So basically, for each row, say okay, if you're playing against, if Tit for Tat is playing against always cooperate, what does Tit for Tat do? Does it always defect, does it always cooperate, does it cooperate and then defect, defect, defect, defect, does it alternate between cooperate and defect? And this should, just to give you some practice in, in interpreting the behavior of Tit for Tat. Okay, that works for me. I think I could do this. Go.

3.3.6 Tit-for-Tat 2 Solution

Alright, so let's start off, maybe you can tell me what happens when Tit for Tat plays against always cooperate. So what happens if you always cooperate? Well, let's see, I start out cooperating. Mm-hm. And since the other person is cooperating I will continue to cooperate because that's what they did the last time. Mm-hm. So, I will always cooperate. That's right. Good. Alright, so what about if we play against always defect. Well if we always defect, the first time I'm going to cooperate because that's what you said Tit for Tat is. Hm. But from that point on, I will do what my opponent does, which is defect. So I will cooperate and then defect, defect, defect, defect, defect, defect, ellipses. Yeah, so I put this always defect in there, but actually it can't ever be the right answer right [LAUGH] because Tit for Tat always starts off cooperating. So the, the other three seem like they might be possible, always defect is not possible. Good. Alright. So, what if Tit for Tat plays against another Tit for Tat? Well, I started out cooperating my opponent cooperated. And since I'm going to do what that person does I will cooperate, but since that person's doing what I did, they will also cooperate. And so we will both cooperate forever, so we will always cooperate. Nice. So isn't that kind of interesting? So Tit for Tat even though, if, when it's playing against itself, is a very cooperative fellow. Hm, I like that. But if Tit for Tat is playing against something that defects, it becomes a little bit more vengeful. Yes. Alright, so what if it plays against something that is a little unsure of itself? So it starts off defecting, then cooperating, then defecting, it's sort of almost an anti kind of thing, right? So it's, this is one that starts off with a defect. Right, so it always, so I, first thing I do is cooperate. And then after that, effectively I do what the opponent does one step before. So I basically take what you, I'm pointing to the screen, you can't see me. I take the the D-C-D-C-D, and I just put a C in front of it, because that's what I'm going to do. So, I will do, C-D-C-D-C-D-C-D, ellipsis. So that's your last choice. Isn't the C-D-C-D-C-D, ellipsis in Atlanta? It is, actually. It is the home of all such analysis of diseases in the United States. The Center for Disease Control? Yes. Is that what it's called? Yes, that's exactly what it is. Come to Atlanta and work for us. [LAUGH] Alright, so good. So, that, that's the pattern of, of responses that Tit for Tat makes against this set of strategies. Oh, so we answered my question.

3.3.7 Facing TfT Question

Alright so now that we have a sense of what tit for tat does against various strategies, let's try to think about what we should do against tit for tat, so what do we do if we're facing tit for tat. So I'm just going to break it down to two possibilities it turns out there's actually more, but these these two are pretty instructive. So let's pretend that we have to choose between always defect as a way of playing against tit for tat, or we have to be always cooperate playing against tit for tat. So what I've written down here is what the total discounted reward is going to be or the total reward in this case. As a function of gamma. So, let's start with what happens if you play always cooperating against tit for tat. Well, you already told me that it. Such a thing will result in tit for tat always cooperating. Mm Hm. And that means we're going to play in this box. The cooperate-cooperate box. And that means on every single round, we're going to to get a minus one. Which means over an infinite run we're going to get an average of one, sorry minus one over one minus Gamma. Mm Hm. That makes sense. Okay? You agree with that? I do. Just minus one repeated over and over again. Now always defect as you recall you told me that, that will result, well for the always defect agent against tit for tat, the first thing that is going to happen is it's going to defect while the tit for tat cooperates right? So we're going to get zero for playing that strategy on the first round. Zero doesn't sound very good but look at the alternatives. They're all negative so zero's pretty good. Mm-hm. So it does this sort of you know good thing in the first step then after that tit for tat responds by always defecting in response. Right? And that means we're going to be stuck in this box the defect defect box where you get minus sixes for the rest of ever. Yes. So that means minus six over one minus gamma. But that starts one step from now so we multiply it by another gamma. Okay. Alright. So these are two different expression that represents what our pay off would be for adopting two different strategies. And in fact, if gamma is really high, very close to 1, then this is a really good answer, right? Because it sort of grows, it's like, minus over one minus gamma. So, for high gamma, we're talking about something that's minus one times a really big number. Mm-hm. Whereas this first one is not so good for high gamma because what's going to happen is it's going to get, end up getting minus six on every step. So it's going to do worse overall. But if, if we're talking about the low gamma. Then, let's say, you know, zero for example. A gamma of zero will, will always defect. We'll get zero plus zero. But always cooperate, we'll get negative one over one. And zero is better than minus one. So for really small gamma, like if the games unlikely to last many rounds, you should defect. But is the game is going to last a long time, then you should cooperate. I believe that. Cool! Alright, so then my question to you is: What's the value of gamma for which these two different strategies to play against tit for tat are equally good. I think I know the answer. Woah! That was fast. Let's give everybody else a chance to think about it. Okay. Go.

3.3.8 Facing TfT Solution

Alright Charles, what, you said you had the answer how do we figure it out? It's 1 6th. I guess that's what, which, we don't, I don't know if it will accept that, whatever 1 6th is expressed as a decimal, but yes 1 6th. How did you get that? I saw the number six and figured it had to be 1 6th because you said it was low, but here is what actually I did, well, I was thinking about it is, you said, well when are they equally good? Well, if you always defect, you get minus 6 gamma over 1 minus gamma. And if you always cooperate, you get minus 1 over minus 1 minus gamma, so they're equally good when those two values are the same. Good, alright, and the denominators are the same, as long as gamma's not one, that's fine. Right. If we divide by the negative 6, we get gamma equals 1 6th. Exactly. Excellent. So, so that's interesting, right? So, that's, it's saying that for gamma values that are less than 1 6th, we should be doi, we should just defect because there's no. Well the games not going to last long enough for us to form any kind of coalition. But for things higher than 1 6th. A half. 3 quarters. 0.999. It's going to be better to cooperate than to defect against tit for tat. Or 6 plus epsilon. Indeed. Yeah. I like that. That's actually very cool.

3.3.9 Finite State Strategy

Now, we kind of cheated here. Because I told you there's just these, those two strategies. But there's actually a bunch of other strategies you can play against tit for tat. And it's worth thinking through, how do you compute a best response to some finite-state strategy? So tit for tat is a finite-state strategy in that it has these two [LAUGH], these two states. And the strategies expressed in terms of transitions between those two states. But in general, if we have some kind of finite-state strategy, like tit for tat, how do we figure out how to maximize our own reward in the face of playing against that strategy? So in this picture here that I drew, the states are labeled with the opponent's choice, the finite state strategies choice, okay? Mm-hm. The edges, that's in black, the edges and labeled in green here, are labeled with our choice. So, for example,

if we're in this state of the, sorry, if our opponent is in this state. Mm-hm. We have a choice. We can either cooperate or defect. On this round. Mm-hm. So, the green arrows tell us how that will impact the state of the opponent. And then these red numbers, I just added the information about well I know that if the opponant is about to cooperate and I choose to cooperate. I can just look up in the pay off matrix that that's a -1 for me. Right? Agreed? Agreed. So I just annotated all these edges, all these choices with these extra numbers. So, one of the things that's cool about this is unlike just the payoff matrix representation that we had before, our choice, it impacts the payoff, which is the same as that, but it also impacts the future decisions of the opponent. And that gives us this structure here and also says that maybe this is a slightly harder thing to figure out because of the fact that we can't just maximize our, the number. We actually have to think about where that's going to lead us in the future as well. So two things then. One, I was always fond of saying that the matrix was all that you needed. But that really only made sense when you were just playing once. Yes. That's right. Right? And, two, I look at this and it's a finite state machine but you know what else it looks like to me? It looks like an MDP. Excellent. It is indeed an MDP. Now, it's a, in this case, my opponent's finite state strategy is deterministic, so it's a deterministic MDP, but it is. It's a discounted MDP. Gamma's playing the role of the discount factor. The entries from the payoff matrix are playing the roles of rewards Our action is playing the choice of our action, and the opponent's internal state structure is playing the role our states. So it is, it's an MDP, and so how do we figure out what an optimal strategy is against a finite state strategy? We solve the MDP. Yeah, exactly. So any, any method for solving an MDP can then be used to actually compute the strategy, so what is the strategy going to look like? It is going to be a mapping from states of the opponent to action choices for us. Right, but that's fine because a state does not have to be your state, it's just what matters. What matters in this case is what they opponent is going to do. Right. So now what are the strategies that can be meaningful against tit for tat? So if we cooperate then we're going to stay in this state and it's always going to be the right thing to do to cooperate. So always cooperate is one. If we always, if we defect, now we have a choice again so we could defect from this state which would cause us to defect forever, so always defect is another one. But what's the other thing that could happen? Well, we could tit for tat ourselves. Well, sort of. I mean, so, we could defect, so we could defect against cooperate, but cooperate against defect. Which would actually cause us to do D-C, D-C, D-C. So those are the only, oh I see. No, you're right. I'm not sure how to say it. But the policy is, defect when you're in this state, and cooperate when you're in this state. But the effect of that is to go back and forth against tit for tat. Right. Basically, take this loop here. And those are the only policies that matter. And in this case, we worked out that. Always cooperate is good against tit for tat if it has a high discount factor and always defect is better if you have a low discount factor. But, we can get that for real by solving the MDP. Right and that makes sense and the reason that those are only 3, let me see if I get this right, the reason those are the only 3 that makes sense because if you think of this as an MDP then it has no history, so when you are in C there are only 2 choices and when you are in D there are really only two choices so if you look at the way you've drawn it. You either stay were you are in c or d, or you take the loop. And those are really the only three options because the rest of them would require you remember what you did, you know, a time step or two ago, and there's no way to do that in an MDP, at least not as you've written it. Well, there's a way to do it, it just would never be better than doing it this way. So an MDP always has a mark off deterministic optimal policy. Right. So we only need to consider those. I think that was the same thing that I was trying to say, but with different words. [LAUGH] Ok.

3.3.10 Best Responses in IPD Question

Alright so, now that we have a handle on what it means to compute a best response against some finite state strategy. Let's actually take a look at these. So, so this is a quiz. Imagine that we have a gamma that is large, so greater than a six. Mm-hm. What is the best response to each of these strategies? So this will be a quiz, but let me just make it clear what the, the goal of the quiz is. So, if you're, if you are playing against, always cooperate. Which of these is the best response to it, which of these is going to actually have maximum reward? If we're playing against always defect which is going to have maximum reward? And if we're playing against tit for tat which are going to have maximum reward? Any questions about that, does that make sense? I'm just making certain I have the rows and the columns right. So my opponent is playing on the rows. Yep. And so I'm choosing among the columns for each one of those rows. Yes? Yeah. So I labelled it best possible response for the columns. And these are the opponent's strategies on the rows. Okay. Okay, go.

3.3.11 Best Responses in IPD Solution

Alright. So is this, is it clear what these answers are? Let's find out. So, let's see. we, we already worked out the math on this. So we know that, for gamma, greater than 1 6th, cooperating is better than defecting, in general. So if I'm going against someone who's always going to cooperate, then I should always cooperate. Incorrect. What? Yes, so, so, we didn't actually work that out. What we worked out was what to do if you were playing against tit for tat. If you were playing against someone who was always cooperating, and is completely oblivious to us. No, no, no, no, no, you're right. You should always defect because you're always going to win. Yes, yes, yes. You're always going to win. You're going to get zero on every time step. Right, right. Yeah. No, that makes sense. That's beautiful. Alright, what about always defect? Well, if you're always going to defect, you might as well defect. Indeed. because we're just in the regular old prisoner's dilemma world. [LAUGH] good, alright. So now we, now we have this, this other strange beast here. So our opponent is playing tit for tat. So we could always defect. Right. But we would do, we'd get a higher score if we can convince tit for tat to cooperate with us. For a gamma greater than a 6th. That's right. So that you should always cooperate. That is true, however. Mm-hm. What if we played tit for tat against tit for tat. You'll end up in the same place. Yeah, so that's just as good. Mm-hm. And that's, that's kind of interesting. If you think about mutual best responses. Yes. So that's a strategy that, a pair of strategies where each is a best response to the other, there's a, we have another name for that. Do you remember? No. [LAUGH] You taught, you told us what it was. Yeah, but I probably used different words. It's, it's a Nash equilibrium. Oh. This, that's what a Nash equilibrium is. A pair of strategies where each is a best response to the other. Each, ea there's no way that either would prefer to switch to something else to get higher reward. And that makes perfect sense. You're in an equilibrium. And that is a Nash equilibrium. Okay. So we can use this little table here to actually identify Nash equilibrium. So, what would a strategy be? So, so if one player plays always cooperate, then the best response to that is always defect. Mm-hm. But the best response to always defect, is always defect. So always cooperate is not part of a Nash equilibrium. Right. But what about always defect versus always defect? No, it is. So that's a Nash. Right? Yep. Because they're both doing the thing that is the best response to the other. Right. Alright, this box here, always cooperate against tit for tat. That's not okay, because if a player does always cooperate, it's always better to switch to always defect. Yep. But, check this out. If you are playing tit for tat, and the other player's playing tit for tat, there's no reason to switch because it's actually a best response, it's the optimal thing to do. And that works from both players' perspectives. And that makes sense. So like you said check this out and you've been using check marks that's very good. [LAUGH]. So we're in this situation where we have two Nash equilibria. Indeed, and one of these Nash equilibria, [NOISE] is cooperative. Which is the thing that we were sad about, or at least that I was feeling really sad about, in the, in the last lesson. The idea that, man, there's just, it's clear that they should just try to get along. You explained that you can modify the reward structure, and then they would get along better. But here it turns out, well, no, another thing you can do is just open it up to the possibility of playing multiple rounds, as long as you don't know how many rounds, it becomes possible to to have a strategy that is best off cooperating, and is in fact a Nash Equilibrium. Isn't that equivalent to changing the reward structure? It's definitely related to changing the reward structure. It's a particular way of changing it. A very particular way. But it's. Yeah, because it's not true any more that we can do this in the one shot case. You have to be in the, in the repeated game, setting. Right, so you change your rewards structure to be a sum of rewards. But it's actually an expected sum of rewards, and you don't know where it is you're going to stop. So I guess you're changed, you're changed the game, you changed the, the rewards. But in a, sort of very subtle way. Yeah, the whole game is different, really. Yeah man, you changed the game. [LAUGH]. Sometimes you gotta change the game. Don't hate the game. No, you are supposed to hate the game. Don't hate the player, hate the game.

3.3.12 Folk Theorem

I'm really proud of us for figuring out a way to make the prisoner's dilemma a little more friendly. It turns out, though, that this, this idea, this sort of core idea, is very general and kind of cool. So, it leads us to a topic that we could call, repeated games, and the folk theorem. Mm-hm. So the general idea here is that when you're in the repeated game setting, this possibility of retaliation, the possible of, you know, not being cooperative actually opens the door for cooperation. Because now it becomes, it can become better to just get along and cooperate than to get retaliated against. Right. And that makes sense as long as the retaliation is plausible. Well, we're not talking about plausibility of the retaliation. We're just talking about, right, because the tit for tat, it's, it's not analyzed in that way. It, it doesn't say whether or not it's actually plausible. It just says, well, if you have this strategy and you play against this strategy, there's no incentive to switch. Right. But we'll, yeah, we'll get into this plausibility thing in a little bit. Oh, I, I stumbled across a

word. Okay, go. [LAUGH] But I want to point out one thing first, which actually kind of irritates me. Kind of along the same lines as like, regression is the wrong word and reinforcement is the wrong word. Folk theorem is the wrong word. So, what is a folk theorem. It's two words. In general mathematics, sorry say again? A folk theorem is two words. That's fair, yes. So, I think it's actually two different terms of art. So in mathematics folk theorems are results that are known, at least to experts in the field, and they're considered to have established status but they're not really published in their complete form. There isn't some kind of original publication saying, oh look, I, I found this thing. It's more, something that like, kind of everybody knows, and so you don't really give anybody credit for it. So it's like a theorem that is in the general mm, cloud of understanding. It's sort of among the population, among the group. Wait! Does, does anybody every prove these folk theorems? You can, yeah! All folk theorems are provable. But it's not like you say you know this is Charles Isabel's theorem. Mm. It's like. No, it's just a folk theorem. It's like. Charles, Charles can prove it. We can all prove it. But we, we're not really sure whoever proved it first. It was just one of those things that everybody knows. Oh, so it's like an oral tradition. Yeah, yeah. I think that's a good way to think about it. Okay. I'm just imagining mathematicians sitting around a fire in the winter, cuddled up against one another sharing theorems that have been proved since the beginning of time to one another. Exactly. Right that's the image that I have as well. This, just it's folk theorems. It's this sort of, you know, I learned it from my grandmother and now I'm telling you. Hm. I like that. I like to think of mathematicians that way. [LAUGH] As grandmothers. Yes. So so that's what a folk theorem is in mathematics. However, in game theory it means something different. It is referring to a folk theorem, but it's also referring to a particular folk theorem. So the folk theorem in game theory refers to a particular result that describes the set of payoffs that can result from Nash strategies in repeated games. So it's a funny thing, it's a funny way that they use the word. It's, it's a, it's a folk theorem but it's also the folk theorem. Hm, well, well, maybe it was actually, proven the first time by some guy named Folk. That's a good idea, we should probably, we should push that forward like call him Folke or something like that. Folke's theorem Yeah. But but no. [LAUGH] Oh well, I tried. Yeah that's that's really great idea that we're going to just move on from. [LAUGH] But what I, what I'd like to do next is kind of build up to this notion of the Folk Theorem and it's going to require a couple basic concepts that are not too different from stuff we've already talked about, but we're going to kind of make them concrete and then we're going to show how they all come together to provide us with this idea of a Folk Theorem. Okay.

3.3.13 Repeated Games 1

The thing that I find most useful in trying to understand the folk theorem and, and what it says and how it works is a thing that I call a two-player plot. I don't know if other people have other names for it, but this, this concept is out and, and often discussed. I just don't know what it's called [LAUGH]. But here's the idea of it, it's a really simple idea. It's like a folk plot. It is a, it is a folk, a folk plot, right, I don't know who invented this plot. So here's what we're going to do, remember this prisoner's dilemma game. We've got two players, there's Smoove and Curly. Mm-hm. And, what we're going to do, is we're, there's a bunch of joint, actions that they can take. So Smoove cooperates and Curly defects, or they both defect, or they both cooperate, or one defects defect cooperate in the other direction. So what we're going to do is for each of those joint outcomes, each of those joint action choices. I'm going to plot a dot, put a dot on a two-dimensional plot, where this is the Smoove axis, and this is the Curly axis, okay. Okay. So cooperate-cooperate, remember from the prisoner, prisoner dilemma payoffs, is minus one, minus one. So I put a dot at minus one, minus one. Defect, defect. Is it minus six, minus six? Cooperate defect is it minus nine zero? And defect cooperate, is it zero minus nine? . So do those four points make sense to you? Do you understand like the idea? They do. It may not be so obvious why we do, do it this way, because as you have told us, the matrix is all you need. Mm-hm. But this is really just representing the matrix in another form. But it actually is sort of losing some information. Because these dots don't tell us what the relationship is in terms of if, if one player keeps the same action and the other player changes to a different action. The matrix captures that but this plot doesn't anymore, it's kind of washed out. I see.

3.3.14 Repeated Games 2 Question

All right. Now so just to kind of reinforce this idea and also to let us think about it in a slightly different direction, I want you to consider solving the following quiz. Here's four payoffs that I've put little boxes around. This one at minus 1, minus 1, this one at minus 3, 0, and this one at minus 4, minus 4. And the question is, which of these payoffs can be the average payoff for some joint strategy? And this is in the repeated game setting, [CROSSTALK] okay. So what we're imagining is these two players can coordinate

in any way they want. We're not talking about trying to maximize reward or, or, being Nash or anything like that. They're just going to execute some, some pattern of, of strategies over infinite run. We're going to average the payoffs that the two players get and we'll say, you know is it possible for them to adopt a strategy so that the average per time step gets minus 3 for Smooth and 0 for Curly? Yes or no? So check that box if that's possible. Can it be that they both get minus one on average? If so, check this box. And can it be that they both get minus 4 on average? Check this box. Does that, does that make sense? It does make sense. All right, so I'll give you a chance to think it through and answer it. Okay.

3.3.15 Repeated Games 2 Solution

Alright, so are any of these easy to answer right off the bat? Well, one of them is really easy to answer right off the bat, and that's minus 1 minus 1. Good. Because that's one of the points, so. [LAUGH] Yeah. So, the joint strategy to get the minus one minus one would be for both players to cooperate, right? Again, they're just, you know, they're just willing to do that just for the sake of showing they can make that value. Right, and I think I know the answers for the other ones as well Alright, hit me. So here's how i'm going to, here's how I'm going to, going to talk you through my reasoning so that if any point I'm wrong, you can gently steer me away from embarrassing myself. Basically I was looking at these points and I thought hm, they form a kind of a convex hull. Aha! And I thought well, surely that's just an accident of the numbers and then I thought, oh wait, of course we're talking about averages here. So that means that all sort of possibilities have to be inside the convex hull of the outer points. So if I drew a line between those four points, I would end up with all possible, achievable averages. What, achievable averages, how? How would you achieve things inside the convex hull? I would appropriately average them. So in particular, the first thing I'd notice that minus three, zero, is outside that. Mm. So, it can't be, it can't be something you can achieve. Right, there's no way to make this minus three, zero, certainly not by choosing any of these points, but also no combi, no convex combination, no probabilistic combination of them, is going to do that either. Right. So, this one is just right out. Right, so that leaves minus four minus four, and it seems pretty. That's inside the convex hull, so there is some combination of them that would work. That's right. Any, any sense of what it would be? 2 3rds, I think 2 3rds D,D 1 3rd C,C. And I get that by noticing that D,D is minus six, minus six, and C,C is minus one minus one, and four minus four is two thirds of the way between there. Boom. Cool. Alright, so you're good with that? I'm good with that if you're good with that. It's your quiz. Yeah. I'm excited. Oh good. So those, it's this, this, the minus one minus one and the minus four minus four are, as as you pointed out there is a more general result here. Having to do with the convex hull. Right. Alright, so through the magic of computer graphics, I have a slightly better depiction of this particular region now. As you pointed out this, this convex hull of the points is really important and what it represents is the, we can call it the feasible region, these are average payoffs of some joint strategy they, they may have to collude to do this. And they may not be particularly happy to do that. [LAUGH]. But the fact of the matter is, they can achieve, by working together, payoffs anywhere inside this region. Huh, so that's what my student Liam always meant when he talked about feasible regions. Maybe so, I mean it could be that he meant any number of other things like places he's willing to live when he goes to get a job. No, it was all game theory stuff, I just never knew what he was talking about, but now I do Michael, thanks to you. Sure, hey I'm, I'm happy to help. So this is a really useful kind of geometric way of picturing something that would otherwise be a little bit harder to see, I think in the matrix form. Sure.

3.3.16 Minmax Profile Question

All right, the next concept that we're going to need to understand the folk theorem is the notion of a minmax profile. So a minmax profile is going to be a pair of payoffs, one for each player. And the value for player represents the payoffs that that player can achieve by defending itself from a malicious adversary. So what do you suppose a malicious adversary would mean in a game theory context? Someone who's desperately trying to hurt you. And what does hurt mean? Gives you the lowest score. Yeah, and what does that remind you of from your lesson? My grad students. You think they're malicious? It would explain a few things. Yeah, I don't think they're malicious. They're sweet. [LAUGH] Yeah, I know a lot of them and they're, they're wonderful people. Well, what it reminds me of is, they are wonderful people. It reminds me of zero-sum games. Exactly. So you can imagine thinking about the game that we're playing, now, no longer as being I get my payoff and you get yours, but I get my payoff and you get the negative of my payoff. So you don't, you don't really care about yourself anymore. All you care about is hurting me. And that's, that's the idea of a malicious adversary. I have some ex-girlfriends like that. I'm so. Oh. [LAUGH]. It is. People do get into this mode sometimes. And that, that's actually going to be important in understanding

the folk there. Hmm. So what I'd like to do is figure out what the min-max profile is for this game. So this is a very famous Game theory, game example. Sometimes goes by the name battle of the sexes. That what the b and the s stand for? Sometimes it stands for Bach and Stravinsky. Blech. Those are like composers, I think. You mean like the Backstreet Boys and Sting. Ahh. Alright, that works for me. So, so, let me explain this story. It turns out that Smooth and Curly actually got away, they didn't make any kind of deal, they actually just figured out a way of escaping from the jail. So they're, they're back out on the streets again, and they decided that they'd like to celebrate their freedom by going out to see a concert. And they both decided that in advance, but what they didn't know was which concerts were available. Once they escaped out into the world, they couldn't communicate with each other, they discovered that there's in fact two concerts in the city that night. The Backstreet Boys are playing, and Sting is playing. Okay. Now as it turns out, each of them is now going to have to choose, whether to go to the concert with Backstreet Boys or Sting, and they're choosing independently. Now, if they end up going to different concert's they're both going to be unhappy and get zero's. I see. If they end up at the same concert, then they're going to be happier, but in fact, as it turns out, Smooth really likes the Backstreet Boys and would prefer that they both end up at the Backstreet Boy concert. But Curly really likes Sting and would prefer that they end up at the Sting Concert. That's not realistic. Which part? The fact that I prefer the Backstreet Boys to Sting. What, what do you mean you? I mean this is Smooth. He's a criminal. Mm, that's a fine point that you make there. There is no connection between these characters and ourselves. Real life characters, living, dead or fictional, or mathematical, or instructional. If so, otherwise purely coincidental. Yeah. I could switch these around if you'd prefer. No, no. I'll go with your fantasy. [LAUGH] All right. I, yeah, I think that payoff matrix may look something like we both have twos in, in the s, the same place. Mhm. But anyway, but let's say for the purposes of this example, there's a little bit of a disagreement. Okay, so now what we need to figure out is what the minmax profile is for this game. Okay. Alright, so that's going to be a pair of numbers. Mm-hm. One number corresponds to the payoff for Curly and one number corresponds to the payoff for Smooth. And it should be, the payoff for Curly should be the payoff that Curly can guarantee himself even if Smooth is trying to get him to have a low score. Okay. And vice versa. Smooth's score is going to be the score that it can guarantee, he can guarantee himself even if Curly is trying to minimize Smooth's score. All right. So let's do this as a quiz. So,I want you to find the min-max profile for this game, this Bach, Stravinsky game or Backstreet, Sting game, and put the, the number for Curly in the first box and Smooth in the second box. Okay. Go.

3.3.17 Minmax Profile Solution

Alright, what'd you got? I'm going to say it's one, one. Alright, how would you figure that out? Well first, I would ask you if that's correct. [LAUGH] [LAUGH] We'll see when you try to. [LAUGH] Figure it out. Okay, so, the idea here is I'm trying to figure out what Curly would get, if Smoove was out to get, make certain that Curly got the worst possible value. So, Curly gets to ch Curly gets to choose among rows. And Smoove gets to choose among columns. So, If I chose the B column, then Curly could get a one. You're given choice Curly would go for the first row instead of the second row, because Curly would get a one. If I took the S column, as Smoove, Smoove took the S column, then Curly would choose the second row, and would get two. And one is lower than two. So, Smoove would choose the first column and Curly would have to get the first row and would end up with a one. That make sense? Yes, but here's what you haven't figured out. Yes? What happens if Smoove, no because, because Smoove says, I'm always going to choose the Backstreet Boys. Mm-hm. Then you're right, Curly should also choose that and at least coordinate. Mm-hm. But what if Smoove is random say half and half random between the two options. And I don't know what's going to happen before then? And you don't know what's going to happen, sorry? I know that, I know it's half Curly knows it's half and half but doesn't know which one he's choosing any given time? Exactly. I see. Then, I would end up with, one half and one. Then for choosing B. Uh-huh. Expect the score would be a half, and for choosing S, they'd expect the wor, expect the score would be one. Yep. Oh, I see. And, and then Curly would choose S and get the one. Right. And that's still consistent with, with that? Mm-hm. Can we work out what the what the worst case is? What do you mean? Well I feel like, we should solve it like a [INAUDIBLE] game, like we did in your lesson. Oh. Say that Smoove is actually choosing a pro, probability either X or 1 minus X. So I thought you were asking for like a pure decision, I wasn't even thinking about mixed decisions. Uh-huh, yes, it could be a malicious and randomized adversary. [SOUND] So we are. [LAUGH] You said yuck [LAUGH]. We, we are really talking about my ex-girlfriends. Okay so you just do the math. Do the math. Alright, so if we, if Smoove chooses Backstreet Boys with probability X and Sting with probability of one minus X. Mm-hm. Then, Curly for choosing Backstreet Boys will get X on average and for choosing Sting, we'll get two times one minus six on average. And the useful point is

going to be to discover when these are equal to each other. Yep. So in fact, Smoove, by being malicious and stochastic, can actually force things down to 2 3rds. Hm. And things being symmetric as they are, Curly can do the same. Okay. So, basically, Curly can behave in such a way, that even against a malicious adversary, it could, he could guarantee himself to a score of 2 3rds. Yeah, a malicious possibly mixed adversary. That's right. Okay. But one, one would be right if we were sticking with pure strategies, but why would we do that? That's right, but for the purpose, if that's right, and you can do a version of the folk theorem. In fact, there's lots of different flavors of the folk theorem. The one that we're going to focus on is going to allow for these mixed strategies. Mm-hm. But. In fact in general you could say you know, no I kind of like the mixed strategies, let's just stick with that. No, I like the mixed strategies too, I just wasn't thinking about them. So, I want to point out that in fact the solution that you gave, I think that actually does correspond to what is usually called Minmax, which is the pure strategy. Yeah! So the, [LAUGH] the minmax is, is in fact one, one. The other concept is really important too though. And I think it's sometimes called the security level profile. So instead of the min max level profile the security level profile. And that allows for the possibility of, of mixed strategies. So that gets you down to the 2 3rds, 2 3rds. I think you know, it turns out that there's folk theorems that can be defined with with either of those concepts. I prefer this one. But I do like this name better [LAUGH]. So I, I apologize if that made things confusing. I'm not confused now. I think in the end Michael, the important thing is we were both right. Well, the example that I'm going to next, these two concepts line up. So let's let's do that, and then we don't have to care. [LAUGH] I, I'm all for that. Let's do that.

3.3.18 Security Level Profile

So, here we are, back at the prisoner's dilemma, again. You may recall this picture. Vaguely. Let's add to this, the minmax, or security level profile. So, for prisoner's dilemma, what is the, the minmax? Isn't it d comma d? It is indeed. D comma d. Right so this is value that you can guarantee yourself against a malicious adversary. Malicious adversary is just going to defect on you, and the best thing you can do in that case is defect yourself. Yep. Agreed, agreed? Agreed. Alright, so now let's take a look at the intersection of the following two regions. There's this nice yellow region that we've already got, and then we've got a new region that's defined by this minmax point. This minmax profile. So the region that is above and to the right of this, of this minmax point. Mm-hm. So, the, that's the region. This, this region, alright we already said that this yellow region is called the feasible region. Mm-hm, or orange or whatever color it is. So, I'm thinking we can call this other region the acceptable region. And it, and, the, what I mean by that is if you think about it, payoffs in this region are, smooth, getting more than what smooth can guarantee itself in an adversarial situation. And Curly getting more than Curly could guarantee himself in an adversarial situation. So, these are all, like, you know, better than it could be. So, why not call it the preferable region? The preferable regions, preferable to not being in this region. Mm-hm The intersection of these two Is the feasible preferable acceptable reason? [LAUGH] Exactly. It's kind of, you know, special, from the perspective that it is both feasible and preferable. And now we are ready to state the Folk Theorem.

3.3.19 Folksy Theorem

So here's the Folk Theorem. Any feasible payoff profile that strictly dominates the minmax or security level profile can be realized as a Nash equilibrium payoff profile, with a sufficiently large discount factor. Prove it. What we're going to do is we're going to construct a way of behaving where both players are going to play so that they achieve this feasible profile. And the reason to make that a Nash equilibrium, what we need to do is make it so that it's a best response. And the way that we are going to make it a best response is we're going to say do what you're told. Follow, follow your instructions to achieve that feasible payoff. And if you don't, then the other player is going to attack you, is going to adopt a strategy that forces you down to your minmax or security level, and that's your threat. So the best response to that threat is to just go along and, and and do what you're told to achieve that feasible payoff. The only way that that's going to be stable though is if the thing that you're asked to do, the feasible payoff, is better than the minmax, right? Because that has to be a threat. You can't threaten somebody and say, you know, do this or I'm going to give you candy. It's gotta be do this or I'm going to do, give you something that's less pleasant than what I've asked you to do. Okay, that actually makes sense. Yeah, so this is, this is a really cool idea. I like it. Hey, could you try saying that again, but with a Southern accent, just the Folk Theorem part? Any feasible payoff profile that strictly dominates the minmax/security level profile can be realized as a Nash equilibrium payoff profile with sufficiently large discount factor. I like that because now it's a folksey theorem. [LAUGH]

3.3.20 Grim Trigger

So another way to think about the proof of the folk theorem, is you could prove it with a little strategy that is referred to as grim trigger. Mm. I like that. I like the way it sounds. So here's, here's the basic structure of grim trigger. It says that what we're going to do, is we're going to start off, taking some kind of action or pattern of actions, it's a mutual benefit. And as long as it's cooperation continues this mutual beneficial, behavior will continue. But however, if you ever cross the line, and fail to cooperate with me. Then I will deal out vengeance against you forever. Hm, so once again, we're talking about my ex girlfriends. I don't know why you're obsessed with this. [LAUGH] Maybe it's just, maybe it's just trying to help you understand. So you know kind of what this situation is. Anyways here's, here's what that looks like in the context of prisoners dilemma. Alright so cooperation is the, is the mutually beneficial action. Yes. And as long as you continue to cooperate with me, that's this C arrow here, then I will continue to cooperate with you, but if you ever defect on me, I swear I will spend the rest of my life making you pay. So no matter what you do at this point, defect of cooperate, I will just continue to defect on you. Pain will rain from the sky. Okay, well this makes sense. So, the idea here is that if you know that I"m going to do this, then hopefully it makes sense for us to continue to mutually benefit. Right. So the whole purpose of this is to create, a, a Nash Equilibrium System kind of situation. Right? Where if I'm playing this strategy. And you're playing this strategy, then neither of us has any incentive to cross the line, and so we're just going to continue to cooperate. Crossing the line is going to decrease your reward, so there's no benefit to doing it, so you won't do it. So it, it's nice because it gets us a Nash Equilibrium. Hm. But there's a problem with it. Of course. And you pointed it out before, so let's, let's dive in and make sure that we understand it and see if we can fix it. Okay.

3.3.21 Implausible Threats

The problem is that in some sense, the threat is implausible. And it, and it, in a very kind of real sense. So what's happened is that if you do fake out on me, if you do cross that line, the idea that I will then spend the rest of my days punishing you, forgoing reward myself, right? Not taking the best response against you, seems kind of crazy. Do you agree? Yeah, again, my ex-girlfriend. Yeah, I totally get this. No, but no, I'm saying that nobody would do that. Right, so it, it would be like being in a elevator with a stick of dynamite and seeing that someone has a hundred dollars and saying, give me your hundred dollars or I'll blow us both up. That's not really a reasonable threat because the alternative to me not giving you a hundred dollars is, you die, which seems probably not worth it. That's right, so you could think about the possibility of, okay, I'm going to not give you the $100, you say that you're going to blow me up, but you will hurt yourself more than you hurt me, so it won't be a best response. Not blowing me up and just not getting the $100 and leaving the elevator is better for you than blowing me up. Right. So that is an implausible threat. So the way we formalize this idea, in the game theoretic context, is to say that we're interested. A plausible threat corresponds to something that's called a subgame perfect equilibrium. Okay. So subgame perfect means that each player is always taking a best response, independent of the history. All right, so let's actually look at a concrete example here, let's imagine playing Grim Trigger, against Tit for Tat. So my first question to you is, are these two strategies in nash equilibrio with each other. Yeah I guess so. And why is that? Because, the fact, if I'm playing Tit, if one player is playing Tit for Tat then the Grim Trigger thing doesn't matter anyway because both of you are going to cooperate forever and it doesn't make any sense to deviate. Right, so any strategy that I could choose that's different than Grim Trigger is going to on average do no better, possibly worse. Right. So I might as well stick with Grim Trigger and Tit for Tat has the same kind of feeling about it, that, it's cooperating with Grim. And any, it can't really do anything better so it might as well do that. Right. So the next question to ask is. Are these two strategies in a subgame perfect equilibrium with each other? And the way that you actually test that, is you say, well, they are not subgame perfect. If there's some history of actions that we could feed to these machines, so that, so that, you know, here's, here's what Grim is doing. It's, it's some, some sequence of cooperates and defects, and here's what Tit for Tat is doing. It's some sequence of cooperates and defects. And once we've reached some particular point. Is it the case that one or the other of these machines is not taking a best response that it could actually change it's behavior away from, what the machine says and do better then what the machine says. If that is the case then it's not subgame perfect but if it's the case of all histories. They're always taking a best response. Then, it is subgame perfect. So, so do you see a, a sequence of, of moves that these two players can take where one or the other of them is not going to be doing a best response? It's, can take, right? As opposed to, will take. I don't understand. Yeah, I'm not sure I do either. That's why I asked the question. It's not a, you know made up history, it's like an actual set of moves that are consistent with Grim and Tit for Tat. No no, no no, so it is, it is not necessarily. So we know that if we

actually play these against each other the only history that we're going to see, is. Cooperate forever. Right, Grim is going to do cooperate cooperate cooperate cooperate, Tit for Tat is going to do cooperate cooperate cooperate cooperate. And so they are, and everything's fine. The question is, can we actually go in and alter, the history, so that one or the other in the machines could take a better action than the one that the machine tells it to take. Yeah if Tit for Tat, ever does defect. Alright, so let's take a look at that. So, let's say, on the first move Grim cooperates and Tit for Tat defects. Okay, so let's say that, that's the moment in time. What will the machines do at this point? Well, at this point the and next time step, Tit for Tat will cooperate and Grim will defect. Good and then thereafter. Grim will always defect. And then Tit for Tat will always defect. Right. So the pay of that Grim gets at this point is going to be, well initially high but then very very low. Mm-hm. On the other hand could Grim have changed it's behavior to do better than this? Yeah. Just by doing just, by choosing to cooperate. By choosing to cooperate, so it sort of ignore the fact that, that Tit for Tat did the defect, and instead do a cooperate here, then Grim would do better. So the idea is that Grim is making a threat, but when it comes time to actually follow through on that threat, it's actually doing something that is worse for itself. Than what it would do otherwise. Do, do you see that? I do. So is it subgame perfect? No. And the proof of that is exactly, exactly what you said, Take, take a look at this history. Here's a history where Grim would not be willing to actually follow through on its threat. Right. So it's an implausible threat, and that's bad. So maybe we've now just undone all the awesomeness that we had done. No. Well maybe. I mean the awesomeness was hey look we can actually get machines that are in nash equilibrium and they're cooperative in, in prisoner's dilemma so they're actually kind of doing the right thing. And, turns out well they are but they're, depending on this notion of implausible threats to do it. Mm, how should I feel about that? Well, let's see if we can fix it. Okay.

3.3.22 TfT vs. TfT Question

All right. So let's make sure that we get this concept. So let's evaluate, tit-for-tat versus tit-for-tat, spy versus spy, and ask whether or not they are subgame perfect, or in a subgame perfect equilibrium with each other. And your choices are yes and no. Mm hm. But if you say no, I'd like you to give me a sequence to show that it is not subgame perfect. In other words, that if they were to take this sequence of actions and this one was to take this sequence of actions, it would leave the machines in a position where they would not be willing to follow through on their threat. That it would be better for them to do something else in the long run, assuming that they're still playing against the other tit-for-tat machine. From that point on. Yup. Just imagine that someone could go in and change one thing. Would you still want to follow tit-for-tat the next time step or not? Yeah, I don't know if that has to be just one thing, but yeah. That's right. Change, change the sequence leading up to this point and then say, okay, do you still want to do what tit-for-tat is telling you to do, or would you rather do something else? Right. Okay. I think I got it. Cool. All right. Go.

3.3.23 TfT vs. TfT Solution

Okay. What's your answer? My answer is no. No, I'm sorry, that's wrong. No, no, I think that, well, if it's, if it's right, then we need to provide a sequence that proves it. Okay. So what are you thinking? Well, I was thinking actually something very similar to what we just saw. Okay. Where so tit for tat, what they're going to do is they're going to do cooperate, cooperate, cooperate, cooperate, right? That's what they normally want to do. Exactly. So what would happen if at one point, one of them defected? Okay, just for simplicity let's make it the very first point. Mm-hm. So, tit for tat number one defects, and tit for tat number two also defects. At the very first time? No, it cooperates, because it's done at the same time. Well, I mean so we're, we can feed it anything we want. So we could tell tit for tat two to cooperate. So it's sort of like we've taken over its brain for a brief amount of time. Right. So I'm not yet convinced it's going to matter for this, but the thing is that from that point on, tit for tat two is going to want to, for the next step tit for tat two is going to want to defect. That's right. And, tit for tat one would want to cooperate. Uh-huh. And that's sort of. Sucks. [LAUGH] For tit-for-tat two, right? So, I don't know, so maybe we should try to think through. What is the expected reward, for tit-for-tat two, to actually do this defect at this time? Or wait, no. So, so, sorry to, to stick with the tit-for-tat machine at this time. Well, what's going to happen at that point is, it's going to keep alternating. That's exactly right. So it's going to get the, the rewards corresponding to D versus C, C versus D, D versus C, C versus D. Over and over again. Yeah. So let's thi, let's think about it in the average reward case. So, in the D versus C, if it does D when the other machine does C, then it gets zero. Mm-hm. If it does C when the other one does D it gets minus nine. Mm-hm. And then this alternates. So if we look at the average award, which is basically what you get when the discount factor's very, very high. Mm-hm. It's scoring negative 4.5. Right. Is there any way, that it could behave against tit

for tat, starting from this point that would do better than negative 4.5. Just go ahead and cooperate. Just cooperate forever? Well cooperate the next time and then keep doing tit for tat from that point on. It'll work out to be cooperate forever. On average. That's right. What will get is a minus, minus one. So not being tit for tat at that point but instead, instead turning always to cooperate would actually get it better. So the idea that is should defect at this point, is an implausible threat. Exactly. So this is not sub-game perfect. So yes, you nailed it. Yeah! Does that make sense? It did make sense. Good, alright. So that leaves open the question of, is there a way, to be sub game perfect in Prisoner's dilemma? Can I ask you a question? Sure. Before you answer that. So, I had sort of convinced myself that it didn't matter whether tit for tat number two started out with C or started out with D. I'm trying to decide whether that's actually true. 'Kay, that's a good question. So what will happen, at this from this point on if we now continue. We, you know, we took over the brains for tit for tat, and we forced them to play defect against defect. Mm-hm. And now we release that, and we let them do what, whatever it is that they're going to do. And what is th, what are they going to do? Is [CROSSTALK] They would defect forever [CROSSTALK] Defect forever. Yeah. And is there anything that tit-for-tat two machine, could do to get a better score than that? Cooperate. Yeah, so it could. Cooperating with tit for tat will bring it back into mutual cooperation. Hm. It will actually get a better score. Yeah. So, in one case, it would average to minus one and the other one, it would, it would average to minus three and minus one is better, so, you're right. Good point. Okay. So what matters is that we get we get them defecting. Right. Okay, so that makes sense. So, I, I was right that it didn't matter, although you do get slightly different answers, or slightly different averages. That's right. But in both cases, there's a way of getting a higher average. Right. Okay, cool, that's what I thought. I thought it was something like that. So now, let's go back to what you wanted to do. So, are we going to be able to figure out how to do Prisoner's dilemma in a way that is sub game perfect? Well, how about I propose a machine, and we'll see what it does? Okay.

3.3.24 Pavlov

So here's a machine that is sometimes referred to as Pavlov named after the Russian psychologists who was studying gastric juices and then figured out how animals learn. So I don't know why it's called that in this particular case but, here's what the machine looks like. It says start off cooperating and as long as the opponent keeps cooperating, then cooperate. So, so far it sounds a lot like tit for tat. Yeah. If you defect the then move to the defect state. So, okay still looks like tit for tat. And again, this defect state has two arrows coming out of it. But their reversed from what they were in tit for tat. Here it says, if you cooperate with me, I will continue to defect [LAUGH] against you. But if you defect against me, then I'm willing to cooperate with you. So, that's a little strange, right? That is a little strange, yes. So, you know, the way to get me to stop defecting against you is to defect against me. And then, I I become cooperative again. So, in other words, take advantage of you until you sort of, pull a trigger on me. Seems like it. Yeah. It's a funny thing. So so again, so tit for tat is like repeat what the opponent did. Pavlov is basically cooperate as long as we're agreeing. If we both defect, I'll cooperate. If we both cooperate, I'll cooperate. But if we disagree, if I dis, if I defect and you cooperate or you cooperate and I defect, then I will defect against you on the next round. That sort of makes sense. Really? Yeah, right? I mean, it says: if you're cooperating, I'm going to cooperate. If, we're both going to start out cooperating. And then if you ever defect on me, then you have basically attacked me, and so I'm going to defect on you. Unless you start cooperating again, in which case I think that you're, you're being reasonable, because of what I did, and so now we're going to start cooperating again. Except that's tit for tat [LAUGH]. No, you're right. You're right, I, I take it back. It doesn't make any sense, too many. Yeah, it's weird. It's a little bit weird. too many V's. Too many V's. Too many V's? Mm-hm. In Pavlov. Yes. Two V's. Yeah, but you can think of them being like arrowheads. Oh well then that makes much more sense. Yeah, exactly. Okay. Yeah, so it's this weird thing where I'm going to continue to defect against you until you realize I'm hurting you, you punch me once. [SOUND] And now okay, good, we're even again. We good? Yeah we're good. Oh this is like men on a basketball court. Yeah, sure but, now here's the question. Is this Nash? So, we'll make that a quiz.

3.3.25 Pavlov vs. Pavlov Question

So, here's a quiz. And this is, I chose it this way so that I can maximize the number of v's in one sentence. Is Pavlov v. Pavlov, Nash? [LAUGH] I think I know the answer. Alright. Let's, let's give everybody else a chance to think about it. Okay. Go.

3.3.26 Pavlov vs. Pavlov Solution

Alright, what'd you get? I say yes. And the answer, that comes from what? Well, we both start off and cooperate. And so, you are always going to cooperate and it doesn't make any sense for anyone to ever move away from cooperation. Indeed. Hence, you are at equilibrium, in particular, you are at Nash equilibrium. That is correct.

3.3.27 Pavlov Is Subgame Perfect

So, that's very good. So, you, were able to realize that Pavlov is in Nash equilibrium. But we can go further than that. Or farther, than that. Further. We can go further than that. And show that in fact Pavlov is sub-game perfect. So, unlike tit for tat. It's actually subgame perfect. So let's let's take a look at how we could convince ourselves of that. So I think the important thing to see is that by feeding these two Pavlov machines different sequences we can get into any of these four different combinations of states. They're both in the cooperation state. They're both in the. Both in the defect state, ones cooperate, ones defect, ones defect and ones cooperate. Mh-hm. so is it the case, that no matter which state that those are in, that the average reward is going to be mutual cooperation. So let's check, if they are both cooperating, and we continue with these Pavlov machines then they will mutually cooperate, so yes. Mh-hm. Alright if they're in defect defect then what's going to happen? Well then they both agree, so then they cooperate. So they're going to move to cooperate and then they'll stay there forever, so then that'll be mutual cooperation. Awesome. What if ones in cooperation and ones in defect? Then they disagree. Right. And they move to the other state. More specifically what? Oh, I don't know [LAUGH], I can't remember. I'm trying to keep track of who, who's who. So if I cooperate. and you defect, then let's see the guy who cooperates moves to defect. And the guy who defects moves to defect. Because you, and now you agree, and so you're going to cooperate. Boom. So, when we're in the cooperate defect state, then on the first move. Let's see, you just, yeah, the right-hand Pavlov just defected, so that causes this transition. And this guy just cooperated, which causes this transition, so that we've gone to defect defect. Right. Which means that we're going to average cooperation, because [CROSSTALK] Mm-hm. That's where we're going to get stuck in the long run. Right. And the same thing works through here. Boom. That's actually very cool, and kind of counter-intuitive. Yeah. And truly neat. So sort of no matter what weird sequence we've been fed we manage to resynchronize and then return to a mutually cooperative state. So I have a question for you. Go for it. So presumably this is really cool like mathmatically because now we shold all do, we should all be Pavlovians. Like we're all Kinseans. ANd then we just kind of move forward from there. D people do this? I don't know the answer to that question. I mean other than men on a basketball course. Sure, you can always return to that. Though I'm not sure I'm aware of any analyses of men on a basketball court and whether or not You know, people have analyzed that. Hmm. But how about this. If I find out, I will post something on the instructor's comments. Okay, that sounds reasonable. So Pavlov is subgame perfect. That's awesome. So remind me again why I care that something is subgame perfect? Because it means that, so let's say that you actually, so I'm being this left Pavlov and you defect and me and you're like yeah I'm going to defect on you because I just want to take advantage of you, and you're, you're going to forgive me and I will have gotten this extra bonus for that. And what it turns out is that No, if we do Pavlov versus Pavlov, we're going to fall into mutual cooperation no matter what. So, so, so this defection that I do, this, this threat, this punishment that I deal out to you, you can earn it back, and we can go back into a cooperative state. Right. So, it's worth it to me to punish you, because I know that it's not going to cost me anything in the long run, and it stabilizes your behavior in the short run. Sure. It makes perfect sense. So it becomes a plausible threat. I like it.

3.3.28 Computational Folk Theorem

So this Pavlov idea actually is more general than just the prisoner's dilemma or iterated prisoner's dilemma. And in fact, led to a result that I like to call the computational folk theorem. The idea of the computational folk theorem says that you give me any two player, bimatrix game. What's a bimatrix game? Just that there's two players. [LAUGH] Okay. [LAUGH] So it seems kind of redundant, doesn't it? It does. What makes it bimatrix as opposed to two player zero sum game which you can write down with a single matrix, this is like, each, each player has its own reward matrix. I see. But you're right, I should have, I could've just said bimatrix game and left out the two player. And it's an average reward repeated game. So we're going to play. Rip, Round after round after round. And we're going to look at the average reward. Or, you can also think of it as discounted with an extremely high discount factor. Okay. So you give me one of those games. And what I can do is, I can build a Pavlov-like machine for, the, for any of these games.

And use that to construct a subgame-perfect Nash equilibrium, for any of these games, in polynomial time. Wow. And, so the way that this works is if it is possible for us to have some kind of mutually beneficial relationship, then I can build a Pavlov-like machine. Quickly. If not, the game is actually zero sum like, right? because in a zero sum game we can't mutually benefit, so we cant do anything like Pavlov we're just going to beat each other up. So we can actually solve, linear program in polynomial time, and work out what the strategies would be if we we're playing a zero sum like game. And so either that works, and produces a Nash equilibrium, and we can test that. Or it doesn't work, but at most one player can improve its behavior. And by taking that best response against what the other player does in a zero-sum like sense, then that will be a Nash equilibrium. So there's three possible forms of the Nash equilibrium. But, we can tick through these, figure out which one is right and drive the actual strategies in polynomial time. Wow, that's pretty impressive, who came up with this idea? So this is a result due to Peter Stone and somebody, Oh yeah me. Oh, well that's very impressive, so you managed to find a way to sneak in some of your own work into this class? Here, let's do some more of that. Okay, I'm a big fan. And I think that's fair because I did that way back when on mimic. Mimic. So, yeah, so what the last topic that, this is, that's all I really wanted to say about the Folk theorem and repeated games. What I'd like to do now is move to stochastic games, which is a generalization of repeating games. And, talk a little bit about how this relates Back to things like queue learning and MDPs. Oh, okay. That sounds cool, almost sounds like you're wrapping up. It is, that. And that will be the end of, end of the new material. Wow. Well that means we're coming towards the end of the entire course. I know. We're going to all cry with, disappointment. And I think. And, and I just say this, you know, as a, as an idle suggestion, that the students should demand that we teach more classes. I concur. So let's get there so that they can demand. [LAUGH]

3.3.29 Stochastic Games and Multiagent RL

So what I would like to tell you about is a generalization of both MDPs and repeated games, that is, that goes by the name of Stochastic games, also sometimes Markov games. Mm. I like the name Marcov game better, but I used Stochastic game because that's what people call it and sometimes it's good to use words that other people use. And what what Stochastic games give us is a formal model. For multiagent reinforcement learning. In fact, I like to think of this in terms of an analogy. Which is something like MDP is to RL as stochastic game is to multiagent RL. It's a formal model. That let's us express the sorts of problems that take place in this formalized problem setting. Hm. That sounds very promising. Cool. Alright so let me let me give you a, I'll start off by explaining it in terms of an example and then I'll give a more formal definition because you know, I can't not. So so this is a little game played between A and B. Oh, I should have it between smooth and curly, but At the traditionally it's played between A and B. Mm, and sometimes it's good to use the words that other people use. [LAUGH] I've heard that. I wouldn't say it quite that way. So this is a three by three grid each of the players can go north, south, east and west. And can stay put if that's helpful. And the, the transitions are deterministic, except for through these, these walls here which are called semi-walls. Mm-hm. So these thick lines represent walls that you can't go through, the thin, wall, lines just represent cell boundaries, but this kind of dashed line here is a semi-wall, and that means If you try to go through that, say by going north from, if A goes north from this position, then 50% probability A will actually go to the next state, and 50% probability A will stay where A is. So, the goal is to get to the dollar sign. And if you get to the dollar sign you get a hundred dollars. So if we ignore A for a second, what should B do to minimize the number of steps necessary to get the reward. Go left, and then go up and go up. Oh, I'm sorry. Go west, and then go north and then go north. Yeah, and what should A do ignoring B? Go east and then go north and then north. Yeah. Unfortunately these guys live in the world together, and what happens is, they can't occupy the same square. And as soon as somebody reaches the dollar sign the game ends and the other player, if the other player hasn't reached the dollar sign, gets nothing. I see. So now there's a little bit of contention. So what happens if A and B both try to go, to the same square at the same time? Let's say that we flip a coin and one of them gets to go, first and then the other one will bounce off of the first one. But that's not a problem when it comes to reaching the money. But it's not a problem, yes, right, so the money is kind of like a money pit. [LAUGH] I don't think that's what a money pit is, but okay. And so they can dive in and they both get the money, because they're in the money pit. I like it. So what do you do if you're A? How do you play this game? Oh! Let's think of another thing. Is, can you think of what, what it might mean to, to have a Nash Equilibrium in a game like this? Oh, that's an interesting question. It would mean, well, it would mean, well, what do you mean, what would it mean? It would mean that, neither one of them would want to deviate. It would mean a pair of strategies for the two players. Now the strategies are now multi-step things that say, they're like policies, right? So... Yeah. Like it's a pair of policies, such that neither would prefer to switch. So can you think of a pair of policies that

would have that property. Well, no I'm not sure. I was trying to think about that. I was thinking that kind of, if I were a nice guy what I would want to do is I would want us both to try to go through the, the semi walls, and if we both go through the semi-walls we just go up again and then we, we hit the dollar sign at the same time. And that's very nice. So okay, good. So that, that seems like a cooperative kind of strategy, right? Where they're both you know, 50% oh I'm sorry, 25% of the time both will get through, both will go to the goal together. Hooray. But... 25% of the time neither one will get through and then we're in the same place we were before, so that's okay. That's right. The problem is the other 50% where one of them gets through and the other one doesn't. Right, so what do, what you do if you make it through and the other one doesn't? What do I do, if I get through, and the other one doesn't? Well if I am only going to do this the one time then I just keep going and get the dollar, and the other person loses. Yeah, alright, so what this works out to be, is that A is going to get to the goal. 2 3rd's of the time, and B is going to get to the goal 2 3rd's of the time. Mm-hm. So, alright, so if that's the case, if I say, okay, A, that's what you should do, B, that's what you should do. Then is there a way that either A or B can switch strategies and do better? Well, if B, for example, decides to go west and then go up, what happens? Yes, that's a good question. B will now make it to the goal a 100% of the time, and A will only make it to the goal 50% of the time. So B has an incentive to switch to that, to this strategy if we tell them to both go through the semi-wall. Right. So that wasn't a Nash Equilibrium. B would wanted switch this new policy. Mm-hm. Is this a Nash Equilibrium? No. Wait, is it? No. Because, why doesn't A just choose to go west east? Well, would, would A do better on average by switching to this strategy? Well let's see. no, actually. Oh, no, no, you said half the time they go through. Yeah. So half the time you flip a coin. So half the time I don't make it. Right. But half the time I do. Right. So, actually, it looks the same. It looks the same. That's right. And B would go from 1 to one half. Yeah, that's true. [LAUGH] So, it, A doesn't have an incentive to do it, but B is hoping very much that A doesn't do that. Right. So so, yeah. So that, so there's one Nash Equilibrium where B takes the center. Another one where A takes the center. I guess if, if they do, if we do this coin flip thing, it, it works out this way. If it's the case that if they both if we change the rules here. So that if they collide, neither of them gets to go. Then go, both trying to go to the center is not a Nash equilibrium anymore, because you can do better by actually going up the semi-wall. Right. And so if we, if, if collision means nobody goes through, then, suddenly, you'd want to do the other thing. Exactly. Or one of you goes through the semi-wall and one goes the direct way. Right. So we can see that there's a bunch of different Nash equilibrium here, sorry, Nash equilibria here. And that it's not so obvious how you'd find them, but it is at least clear that they exist and they have a different form than what we had before, because they're not policies instead of these otherwise simplified just you know, choose this row of the matrix. Mm-hm. Cool. Alright. So let's think about how we might learn in these kinds of environments. Oh, okay, I like that already.

3.3.30 Stochastic Games

So, stochastic games, originally due to Shappley, have a bunch of different quantities. State, actions, transitions, rewards, and discount factors. And here's how we're going to do it. We're going to, we're going to say that s, little s, is, is one of the states. And actions could be like little a, but actually, since we're going to focus mostly on two player games for the moment, I'm going to write actions as a and b. Where a is an action for player one and b is an action for player two. Sound okay? Sure. Alright. So next we have the transition function. So, the transition function says: If you're in some state, s, and there's some joint action that's taken, like all the players choose their action simultaneously, a comma b, then what's the probability of reaching some next state s prime? And we can write rewards the same way. So there's reward for player one, given that we're in state s and there's a joint action ab. And there's the reward for the other player, the second player. And a discount factor is you know, like a discount factor. Totally makes sense. So oh its the same discount factor for everyone. Yes! Good a good point. One need not define things that way, but in fact that is the way it's always defined. Hey, not to go of on a tangent here, but sometimes I see NDP's and things like stochastic games defined with a discount Factor being a part of the definition, and sometimes not. Like it's just a part of the prob, definition of the problem or sometimes its a parameter of an algorithm. Which do you prefer? Why do you, why haven't, why have the discount factor actually listed as part of the definition of the game? I have no justification other than it's nice to have listed the things that might be important as oppose to you know, working through algorithms for while and then saying, oh yeah there's this other number that kind of matters too. Okay that's fair. I was just curious. So one of the things that I actually find really interesting is that this model was laid out by Shapley. Mm-hm. One of the, like a former Nobel prize winner. I guess once you're a Nobel prize winner, you're a Nobel prize winner forever. Yeah, [INAUDIBLE]. So, Shapley, the Nobel prize winner, and as we're going to see in a moment, this model is actually a generalization of MDPs, but Shapley published it before Bellman published about MDPs. Oh. This

is pre-Bellman. So MDPs, to some extent, can be thought of as a narrowing of the definition of a stochastic game. Huh. So, all right. Let's do a little quiz. And see that we really understand the relationship between this model and other things that we've talked about. Okay.

3.3.31 Models and Stochastic Games Question

Alright so, Stochastic Games are more general than other models that we've talked about. And so, just to make that case here's a way of making the Stochastic Game settings more constrained. And, by making them more constrained, actually turning them into other models that we've talked about or could talk about. So, I wrote down three different ways of constraining the Stochastic Game model. One says that we're going to make the reward function for one player the opposite of the reward function for the other. The next one says that the transition function has the property that for any state and joint action and next state. If that's going to be equal to the transition probability for state joint action, next state, where we've changed potentially the choice of action for player two. Mm-hm. So basically, player two doesn't matter to the transitions or the rewards for player one, and the rewards for player tow are always zero. So that's, that's again, you can specify this as a Stochastic Game and then in the third case we are saying that the number of states in the environment is exactly one. So I claim that by doing these restrictions. We get out the mark-off decision process model, a zero sum statistic game model, and the repeated game model that we've been talking about in the context of, like, the folk theorem. So, what I'd like you to do is write the letters, A, B, and C in the correct boxes. Okay. Go.

3.3.32 Models and Stochastic Games Solution

Alright, talk me through it. Okay, so I'm going to say that I think I know the answers for this one. And let's start with the first one. So R1 equals minus R2, which you'll notice they're equal and opposite. And in fact if you add them up, that is you sum them you end up with zero. So I'm going to say that's a zero sum stochastic game. Nice. For two, basically you're saying that for all intends and purposes, there's only one agent. Which just makes it a regular Markov decision process. Yeah. So isn't that interesting? That just by the other player irrelevant, then that's what an MDP is. It's like a game where the other players are irrelevant. Yeah, which, both of my children are like that. But okay, I think that's pretty cool. And in fact, I'd be right in saying that R2 doesn't have to equal to zero. As long as it just equals to some constant. Yeah, that's, I mean, constant. Actually, depending on how you think about it, it could be, we could just ignore the whole R2 thing and just say that. As far as the first player is concerned, since the second player really has no impact on anything. It doesn't matter. But the reason I put that in is I got kind of scared that like. I feel like if I lived my life and knew that my actions effected the state and my rewards, but they were also effecting the rewards of somebody who didn't matter. Like I feel like that would actually still have an influence on me. Sure, but then the way you get around that is you would say, well, your R1 is actually equal to your R2. Oh. [CROSSTALK] It would somehow [UNKNOWN]. So, so if I had gone like that, wouldn't that be the case then that we're saying? Oh yeah, I see. That the second player is irrelevant, but the reward, but the first player may be relevant to both. Right. Yeah, okay, yeah I like that a little bit better. Yeah, I mean, once again, it all boils down to changing the rewards. Okay, and so given A and B, I know the answer to three must be C, unless you're tricking me, and it could be A or B again. And which I suppose you could have done, you didn't say they were mutually exclusive. So let me actually argue why it would be C? Well there is only one state and since you're in a stochastic game and you're going to be continually doing this. It means that you're basically doing the same game over and over and over again, so it's a repeated game. Yeah, yeah, yeah so in particular the actions impact the rewards, but they're not going to impact the transitions because you're always just going to back to where you were. The discount factor plays the role of, of decided when the game's going to end, stochastically. And, so yeah, it's exactly a repeated game, this is the one I feel most comfortable about because this really does recover that same model we've been talking about. I like it. Cool! Now, given that we actually are now in a model that generalizes MDPs, it would be nice if we can generalize some of the things that we did with MDPs, like Q learning and value iteration to this, this more general setting. So, that's what we're going to try next. Cool.

3.3.33 Zero-Sum Stochastic Games 1

Now what makes stochastic games more interesting, perhaps, than repeated games, is the idea that the actions that the players take impact not just the rewards, but also future states. Right? And, so this is the same issue that comes up when we're talking about mark off decision processes and the way we dealt with

it in that setting was by defining a value function. So, it seems pretty reasonable to try to go after that same idea again. So what I've got here is actually the Belmont Equation And let's look at this together and let's see if we can fix it because it's not quite right. Okay. For dealing with the idea of zero sum in a stochastic game. Okay so you remember the Belmen equation? We've got Q I star. Mm-hmm. So there was no I before, but Q star is the state. Is to find over state actions, so here we're going to define it over action, joint actions Mm-hm. For the two players, action pairs. The immediate reward to player i for take, for that joint action in that state plus the discounted expected value of the next state. So we need to factor in the transition probabilities So the transition of actually going to some next state s prime is s, sorry t of s, AB as prime. Right? So now, we're imagining what happens when we land in S prime. So what I've written here says, well, we're going to basically look a the Q values in the state that we landed in, and kind of summarize them, summarize that value back up so that we can use it to define the, the value of the state that we left. You with me? I am with you. Alright so if we put this in if, if we say the way we're going to summarize the value for the new state that we land in. Is we think of it as actually a matrix game. That there's payoffs for each action choices of A prime and B prime. And over all of those, we need to kind of summarize well which of those actions in this table of values that we get for s prime. Which of those values are we going to propagate forward and call the value of that state? So what we did in regular MDPs is that we said we'll take a max over all the actions or in this case all the joint actions. Uhun. So what do you think that translates to? Well you wrote down max but that doesn't make sense, that doesn't, that can't be right. Well it translates, it means something. It just doesn't mean what we mean it to mean. That's true, that's fair. So what does it mean and then, and then, how can we fix it? So let's start off with what does it mean. It means that you kind of always assume that the joint actions that are going to be taken Will benefit you the most. So, everyone, everyone is trying to make you happy so, this makes you optimistic? yeah, sort of optimistic to the point of, of Delusion? Yes, very good. Right, it just basically says that whenever we're in a state, the whole world is going to choose their actions to benefit me, and this is not what we get a say a zero sum stacastic game, but a zero sum stacastic game, we should be you know. Like fighting it out at this point. So that would work out if everybody's rewards were the same, or everybody's rewards were the sum of everyone's rewards, or something like that. That's right. If it was some kind of team based game. Mm. Or if everybody was, you know, going to sacrifice their own happiness for the benefit of Qi, or i I mean. Hm. So it's not reasonable to assume that. In fact, what was it that we were assuming when we had a zero sum game that was just a single stage? Right? Just a single game and then we were done. Oh, that people were doing minimax. Right. And maximin. So what if we changed the equation to look like that? So, what I mean by this, is when, when we we evaluate the value of a state. We actually solve the zero sum game in the Q values and take that value and use it in the, in the context of this equation. That seems closer to right. Yeah. I mean, it's not an unreasonable thing to do. It's just to say, I'm going to summarize the future by imagining that we're going to play that game that represents, you know, all the future. Sure. And I'm going to act in such a way to, to try to Maximize that assuming you're trying to minimize it, which makes perfect sense if it's a zero-sum game. Right. I was, yeah, and we're still, we're still acting as if there are only two people here. Yeah, yeah, that's right. It turns out that when you're talking about zero-sum it really implies that there's only two players. Because if you have a zero-sum three player game, it really is just a general sum game. You can imagine that the third player is just an extra factor that's just messing with the numbers to make things sum up to zero. So, yeah, so zero sum really does kind of focus on this two player setting. That makes sense. So we got this modified kelvin equation and we can even translate it into a form that's like Q learning. So the analog of the kelvin equation and the Q learning update in this setting would be that we. If we're in some state, there's some joint action that's taken, there's some pair of rewards that comes and some next state that's visited that the Q value for that state, joint action pair, is going to be updated to be closer to, the reward for player i plus the discounted expected value, or sorry, the discounted Summarized value or v value of the new states as prime, and we'll again, we'll use mini-max to summarize what the values are in that new state. I like it. And that equation is sometimes referred to as mini-max Q, because it's like the Q learning update but just with the mini-max operator instead of a max. That makes sense.

3.3.34 Zero-Sum Stochastic Games 2

So, if we set things up this way, we actually get some wonderful properties coming out. So here are some things we know about this set up for zero-sum stochastic games. Value iteration works, so we can actually solve this system of equations by using the value iteration trick, which is to say, we initialize these Q values to whatever and then we just iterate this as an assignment, right, we just say, you know, equals. So value iteration works. This minimax Q algorithm converges under the same kinds of conditions that Q learning

converges, so we get this nice, you know, Q learning analogue in this multi-agent setting. The Q star that's defined by these equations is unique, so we iterate it and we find it and it's just, there's just that one answer. The policies for the two players can be computed independently, that is to say, if two different players are running minimax Q on their own and not really coordinating with each other except for by playing the game, that the policies that they get out will actually converge to minimax optimal policies. So it really does solve the zero sum game, which is maybe not so surprising because, you know, they are trying to kill each other after all. [LAUGH] Yeah. So, the idea that they'd have to collaborate to do that efficiently would be weird. [LAUGH] I never thought about it like that, but, yeah, that would be weird. The, this update that I've written here can be computed efficiently, which is to say in polynomial time. Because this minimax can be computed using linear programming. Yes of course. And, finally, if we actually iterate this Q equation and, and it's converging to Q star, knowing Q star is enough to figure out how to behave optimally. So we can convert these Q values into an actual behavior, again, by using the, the solution in the linear program. So it's just like MDPs of value iteration with Q-learning? Exactly. It's like we've gone to, to a second agent and it really hasn't impacted things negatively at all. This is, this is all the, pretty much all the things that we want, come out. There, there are some things that don't come out. For example, in the case of an MDP, we can solve these, this system of linear equations in polynomial time. Not just by value iteration, but we can actually set up it as a single linear program and solve it and be done in linear time or, sorry, not linear time, polynomial time. This is not known to be true in the zero-sum stochastic game case, it's not known whether it can be solved in polynomial time. Hm. So there, it is a little harder as a problem, but it's, you know, not harder, not deeply harder and not harder in a way that matters in a machine learning setting. Cool. So this is really great. So let's, let's try to take this same approach and see if we can deal with general sum games. Okay.

3.3.35 General-Sum Games

So okay, so let's think about General sum games, so not zero sum any more. But we're not, you know, restricted, it could be any kind of relationship between the two players. And so the first thing we need to do is realize well, well we can't really do minimax here any more. Right, because that doesn't make sense. Right. That only works with zero-sum games. Well it's only, yeah. That's, well, it sort of assumes that the other player's trying to minimize my reward and that's not the, that's not the concept of the Nash equilibrium. We'd like to do something analogous and find a Nash equilibrium in this general sum setting. So what, what operator do you think we would need in this context here? Nash equilibrium? Yeah, so that would be a very reasonable thing to do, is instead of computing mini max, we actually compute of the two matrix game, right, using Q1 and Q2, compute the Nash equilibrium of that and propagate that value back. It's a well defined notion, right, that we can summarize the value of these two pay of matrices with with a pair of numbers which are the values of the Nash equilibrium. Mm-hm. Alright, so so good. So we can do the same thing in the Q learning setting. Substitute in a Nash equilibrium. And we can call that algorithm Nash-Q, which is, appears in the literature. Nice. Oh minimax Q by the way is something that I wrote about. Nash-Q is a different algorithm. So it's not as cool, is what you're saying. Well, let's let's see how it goes. So this is now an algorithm, you can actually, well, this set of equations it's not exactly clear what it means, but we can think about turning that into value iteration, right? By turning this into an assignment statement. Mm-hm. So, what happens? Well, value iteration doesn't work. No. So, yeah, so if you, you repeat this over and over again, things, weird things can happen, it doesn't, it doesn't really converge, it doesn't really solve this system of equations necessarily. Hm. And unfortunately the, the reasoning here is even harder in the case of Nash-Q because in the case of Nash-Q, it's really trying to solve this system of equations using something like value iteration, but with extra stocasticity. And so it also suffers the same problem. It doesn't necessarily converge. There's not really a unique solution to Q star because you can have different Nash equilibria that have different values. Right. So there isn't really much hope of converting to the answer because there isn't the answer. The the policies can not be computed independently, right, so Nash equilibrium is really defined as a joint behavior, and so we can't just have two different players computing Q values. Even if we could compute the Q values. It wouldn't necessarily tell us what to do with the policies, because if you take two different policies that are both half of a Nash equilibrium, two halves of a Nash equilibrium do not necessarily make a whole Nash equilibrium. Right. because they could be incompatible. So, you know, so far so good, right? Yeah, I can't wait to see what happens next. The update is not efficient unless P equals PPAD, which is to say, computing a Nash equilibrium is not a polynomial time operation as far as we know. It is as hard as any problem in a class that's known as PPAD. And this is actually a relatively recent result, in the, in the last five, ten years. And this class is believed to be as hard as, as NP. So, possibly harder. So it doesn't really, doesn't really give us

any leverage to, computational leverage to kind of break it down in this way. So that's unfortunate. And finally, the last little hope of, well, maybe we can define this kind of learning scenario using Q functions the same way we've been doing, Q functions are not sufficient to specify the policy. That is to say, even if I could do all these other things, efficiently compute a solution of, you know, build the Q values, make them so that they're compatible with each other. And now I just tell you, here's your Q function. Now decide how to behave, you can't. It's, there's not enough information. You're depressing me, Michael. Yes, so this is kind of sad. We go to the general sum case, which in some sense is the only case that matters' because zero sum never really happens. And what we discover is that we lose all, seemingly lose all of the leverage that we have in the context of Q type algorithms. Mm, mm, mm. And that's where we'll stop. Oh. So we're going to end on a high note. No, maybe we should say something before we depart. Let's do that. Come up with something positive to say. Okay.

3.3.36 Lots of Ideas

So even though things are kind of grim, with regard to solving the general sum games. There are lots of ideas that, that have proven themselves to be pretty useful for addressing this, this class of games. It is not the case that any one of them has, has emerged as the dominant view, but, but these are all really cool ideas. So here's one. You can think about stochastic games as themselves being repeated. So, repeated stochastic games. We're going to play a stochastic game and when it's over, we're going to play it again. And that allows us to build folk theorem-like ideas at the level of stochastic games. Oh, that's cool. And so there are some efficient algorithms for dealing with that. So that's one idea. Another one is to make use of a little bit of communication side-channel to be able to say, hey, other player. Here's this thing that I'm thinking about. And it's cheap talk in a sense that, it's nothing that's being said is binding in any way but it gives the two players the ability to co, to coordinate a little bit. And you can actually ultimately compute a correlated equilibrium, which is a, a version of a Nash equilibrium that you know, requires just a, a little bit of coordination, but can be much more efficient to compute. And you can actually get a near optimal approximations of the solution to stochastic games using that idea. Yeah, that's cool. Didn't, didn't I do some work in this space? You did. That's where I got the idea from. Oh okay. There's some, some work by Amy Greenwald looking at how correlated equilibria play into stochastic games and then your, your student Liam and you developed a, a really cool algorithm that actually probably approximates, the solutions. Nice. Another idea that I've heard a lot about lately, that I really like, is the notion of a cognitive hierarchy. The idea that what you're going to do is instead of trying to solve for an equilibrium, you think about each player as assuming that the other players have somewhat more limited computational resources than they do. And then taking a best response to what they believe the other players are going to do. This turns out to be a really good model of how people actually play when, when you ask them to do games like this in the laboratory. Huh. Yeah, the good news about this idea is that, because they're best responses, they can be more easily computed. That, that it's more like, cue learning in MDPs again because you're assuming that the other player is, is fixed. Okay. I'll buy that. And, the last idea I want to throw out is the notion of actually using side payments so that the players, as they're playing together, cannot only take joint actions, but they can say, hey, I'll give, I'm going to get a lot, but if we take this action, I'm going to get a lot of reward. I'm going to give some of that reward back to you, and that will maybe encourage you to take the action that I need you to take so that we'll both do better. And so there's this lovely theory by a father and son duo that they call coco values. Coco sounds awesome but it stands for Cooperative competitive values [CROSSTALK] and so it actually balances the zero sum aspect of games with the mutual benefit aspect of games. So it's, it's, it's a really elegant idea. So basically, the problem isn't solved but there are a lot of cool ideas that are getting us close to solving it. That's right. Yeah. So even though the one player and the zero sum cases are pretty well understood at this point, the general sum case is not as well understood. But there's a lot of really creative ways that people are trying to address it. So, that is good news.

3.3.37 What Have We Learned

Okay Charles, what have we learned? And I mean specifically in the context of this game theory two lesson. That's a that's a good question. We learned about Iterated Prisoners Dilemma. Which turns out to be cool, and it solves the problem, and we learned about how we can connect iterated prison's dilemma to reinforcement learning. What do you mean? Through the discount. Yeah, so I think of that as being the idea of repeated games. Right. Let's see, what else have we learned? So we learned about iterated prison's dilemma, which allowed us to get past this really scary thing with repeated games, connected it with reinforcement learning. The discounting. And then we learned other things like for example, I don't

remember. What, what did we learn? Well so the, the connection between iterated prisoner's dilemma and repeated games was the idea that we can actually encourage cooperation. And in fact, there's a whole bunch of new Nash equilibria that appear when you work in repeated games. That was the concept of the Folk Theorem. Right. The Folk Theorem. So, the Folk Theorem is really cool. And this whole notion of repeated games really seems like a clever way of getting out of what appear to be limitations in game theory. Right. Yeah. And in particular by using things like threats. Right. But only plausible threats. Right, so that was the next thing we talked about. The idea that an equilibrium could be subgame perfect or not, and if it wasn't then the threats could be implausible. But in the subgame perfect setting, they're more plausible. Right let's see and then we learned about Min-max Q. Well there was one last thing we did on the repeated games, which was the Computational Folk theorem. Yes you're right. So basically what we learned is that Michael Littman does cool stuff in game theory. Or at least he does stuff that he's willing to talk about in a MOOC. Yes, so that's, that's there's actually a technical term for that right? MOOC acceptable research? Oh, I didn't know that. Mm-hm. So all these things are by virtue of the fact that they showed up in this class look acceptable. Exactly. Alright, you're right, but then we switch to stochastic games. Mm-hm. And they generalize MDP's and repeated games. Mm-hm. Anything else? Well, that particularly got us to min-max Q and then eventually to Nash Q. But despite the fact that Nash Q doesn't work, we ended up in a place of hope. [LAUGH] We end with some hopefulness. Yeah, and you know, I think that that's actually a lesson for the entire course. That at the end of the day, sometimes it doesn't always work, but there is always hope. [LAUGH] We don't give up and that's, that's, that's how research works. Even when we have impossibility results for things like clustering, or multi-agent multi-agent learning and decision making, we still keep struggling forward. And keep learning, and isn't that what's really important. I think so. Its important for us, and its important for machines. Yes, that is beautiful. I feel like we've made it to a good place, Michael. Perhaps we should stop. [LAUGH] Well it has been, it has been delightful getting to talk to everyone, and it has been very fun getting to talk with you, Charles. And thanks to everybody for making this happen. I agree. And we have one more chance to talk with one another as we wrap up the class. And I look forward to that. So, I will see you then, Michael. Awesome. Do we get to see each other in person for that? We get to see each other in person for that. That will be fun. Yay! Okay, well, bye, Michael. I'll see you next time. Bye. See yeah. Bye, bye.