CS7641 ML Lectures - Unsupervised Learning

I. Randomized Optimization

A. 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 PZTTMNIIAOOI. 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.

B. 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.

C. 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.

D. 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.

E. 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.

F. 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.

G. 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.

H. 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.

I. 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.

J. 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.

K. 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.

L. 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.

M. 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]

N. 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.

O. 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.

P. 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.

Q. 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.

R. 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.

S. 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!

T. 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.

U. 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

V. 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.

W. 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.

X. 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.

Y. 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.

Z. 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.

AA. 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.

AB. 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.

AC. 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.

AD. 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.

AE. 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.

II. Clustering

A. 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.

B. 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]

C. 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.

D. 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.

E. 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.

F. 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?

G. 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.

H. 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.

I. 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.

J. 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.

K. 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.

L. 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.

M. 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.

N. 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.

O. 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.

P. 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.

Q. 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]

R. 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.

S. 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.

T. 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.

U. 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.

V. 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.

W. 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.

X. 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.

Y. 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.

Z. 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.

AA. 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.

AB. 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.

AC. 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.

III. Feature Transformation

A. 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

B. 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.

C. 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. .

D. 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.

E. 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.

F. 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.

G. 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.

H. 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.

I. 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.

J. 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.

K. 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.

L. 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.

M. 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.

N. 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.

O. 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!

P. 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.

Q. 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.

R. 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.

S. *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.

T. *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.

U. 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.

V. 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.

W. 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?

X. 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?

Y. 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.

Z. 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.

AA. 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.

AB. 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.

AC. 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.

AD. 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.

AE. 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.

AF. 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.

AG. 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.

AH. 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.