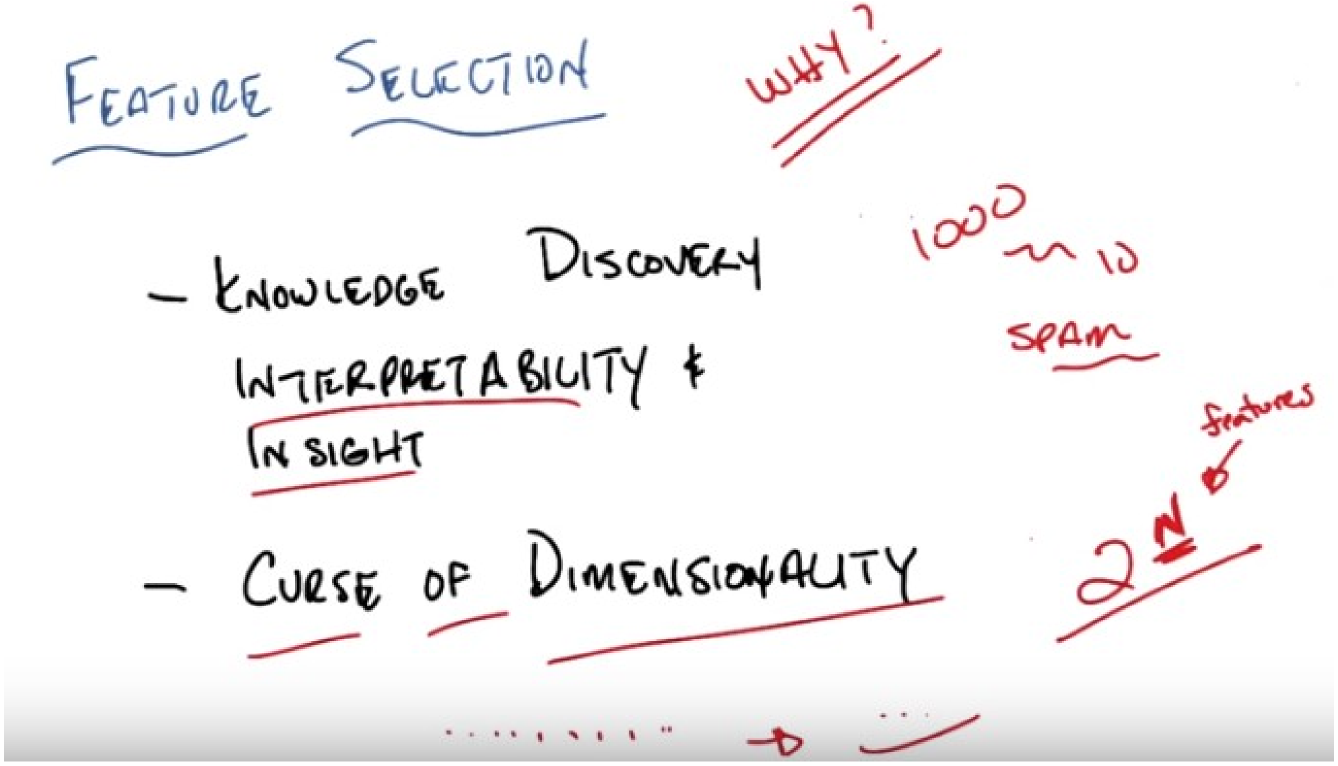
FEATURE SELECTION

1. Hi Michael. >> Hey Charles, how are you doing? >> I'm doing just fine. How are you doing? >> I'm fine, thank you. >> Excellent. We are on our penultimate topic in unsupervised learning. >> Huh, that seems kind of fast. >> It does seem kind of fast, but we'll probably go back and pad things so that it seemed longer. >> [LAUGH] Maybe we could just talk more slowly. >> I'm not really sure I'm capable of that, but I'll do my best to talk more slowly. So, today, what we're going to talk about is feature selection. >> nice. >> Oh. Now it turns out that this is one of two topics that we'll be talking about. That's what I meant by penultimate. And the other is feature transformation, which is something that has a long history, and is very interesting in its own way. But I think it's better to talk about feature selection first. It's a nice crisp problem. Okay? >> Sure and then this is in the unsupervised setting? >> Yes. This is in the unsupervised setting, though it'll turn out that, as is often the case, it's not clear what it means to do something in the unsupervised setting, in the absence of some supervision later down the road. But we'll see that as we, as we talk in a couple of minutes. >> Cool. >> Okay? Alright. So the first thing that we should do I think for feature selection is, define the problem.



FEATURE SELECTION

WHY?

- KNOWLEDGE DISCOVERY

INTERPRETABILITY & INSIGHT

- CURSE OF DIMENSIONALITY

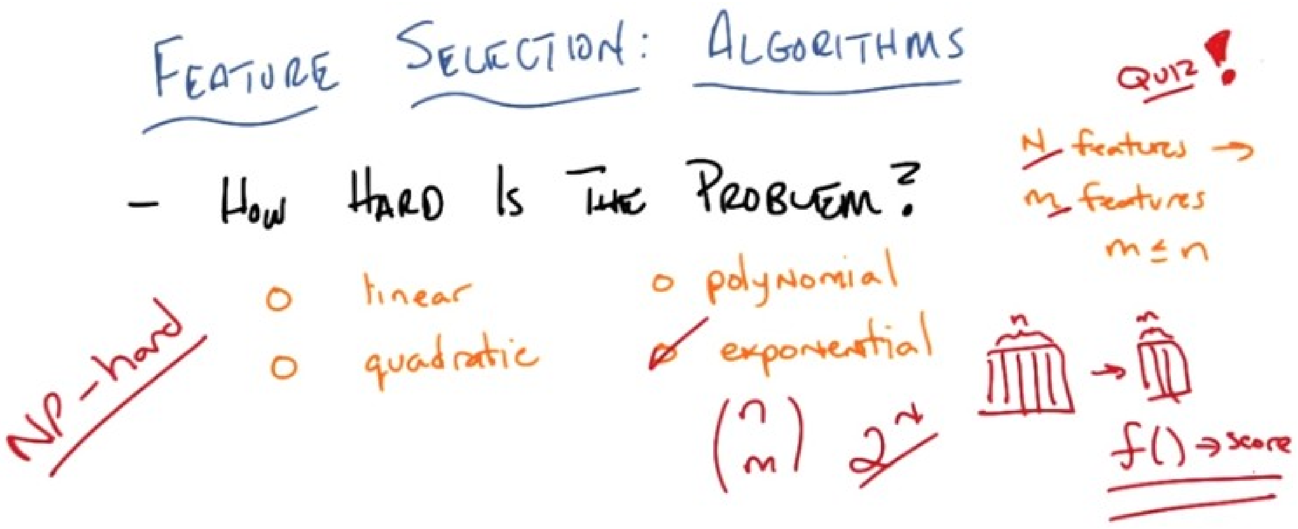
1000 ~ 10

SPAM

2^N features

2. So why might we care to do this, Michael? So, there's a bunch of reasons but really it boils down to two. One is really about human beings and the other is really about machines and machine learning algorithms. And the first one really boils down to what I'll just call knowledge discovery, alright. It is often useful when you're thinking about a set of data in a problem, to be able to interpret the features. To figure out well, which subset features that I've been keeping track of say, when then I'm using on my sensors actually matter. And so really the feature selection problem for what other reasons you might think of doing it or often for interpretability and for insight. So it turns out that of the 1,000 features that I keep track of only a few matter. Maybe only 10 of them matter for doing solving some problem

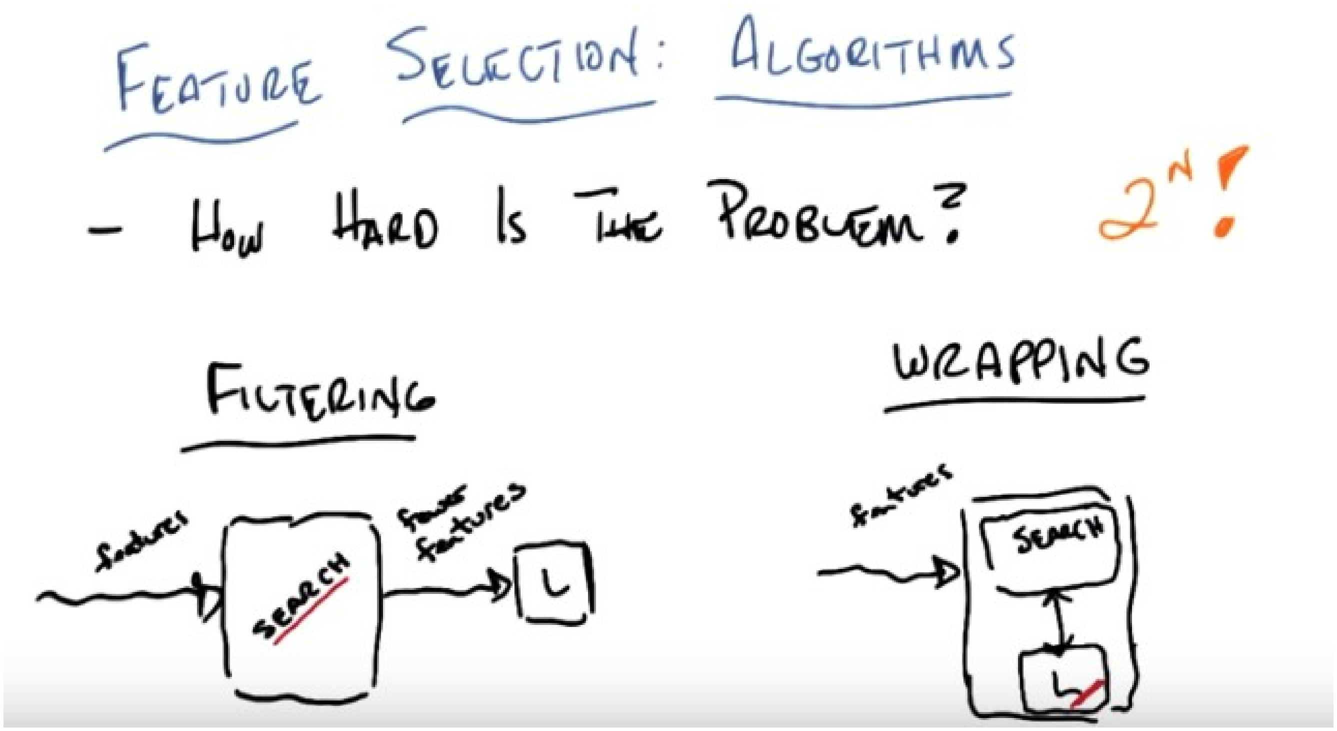
of prediction. So let's think about our spam case for example. There are lots and lots and lots of features that we might care about in spam case, but it may turn out that after we do some kind of analysis that it really all boils down to something like whether you're getting mail from a Nigerian prince, for example. And some of these things that you thought mattered. Don't seem to matter. For for the purposes of solving some problem or, or for doing prediction. And, so, often, this is a very, very useful thing to do. You do this kind of thing all the time, Michael. Actually, if you go back and you look at some of the things that we have been giving for examples. You come up with problems and features taht are, for example, visible to two dimensions as opposed to three or four or five becdause it's easy for you to visualize them and to understand them. And for students to understand how the algorithms work in two d. >> Hm that makes sense. >> Does that all make sense? Okay. So that's one thing and there's, and this really should not be underestimated. People tend to ignore this a lot in the, in the, at least in the machine learning field, much less in the data mining field but this really is a sort of key and an important issue. But there's another reason we might care about if it's algorithmic and that is the curse of dimensionality. So do you remember what the Curse of Dimensionality is Michael? >> Yeah, as you add more features you may need exponentially need more data to kind of fill out the space. >> The Curse of Dimensionality says that the amount of data that you need grows exponentially in the number of features that you have. So it'd be really nice if you could have fewer features. So the problem of feature selection helps us, hopefully at least in a, in a nice world. To go from a whole bunch of features to just a few features, thus making the learning problem actually easier. >> [LAUGH] Looks a little bit like a smile, an alien smiley face now. >> That's right. So we came with an alien frowny face in the beginning and [LAUGH] then we [INAUDIBLE] with an alien smiley face. Okay, so you buy these reasons Michael? >> Yeah, I like how you managed to, you know if X is the input space here, you've managed to map X to Y. >> Michael uses the feature transformation function known as the pun. >> Mmm. >> That's really good, I like that. Okay. Anyway. Okay, so let's keep all this in mind when we're talking about all the algorithms that we're talking about in this lesson, and even in the future transformation lesson. The goal is to take, be able to use, in general, a bunch of features, because you think they might all be important, and then apply some kind of algorithm to get to just the important features. And if you can do that well, then not only will you understand your data better, But you'll also have an easier learning problem. >> Got it.



3. Okay. So, Michael, I want to go into some algorithms, or at least some classes of algorithms but before we do that, I think it's useful to really ask a very simple question, which is exactly how hard is this problem. And to help answer that we're going to do a quiz, because you like quizzes, and I'm going

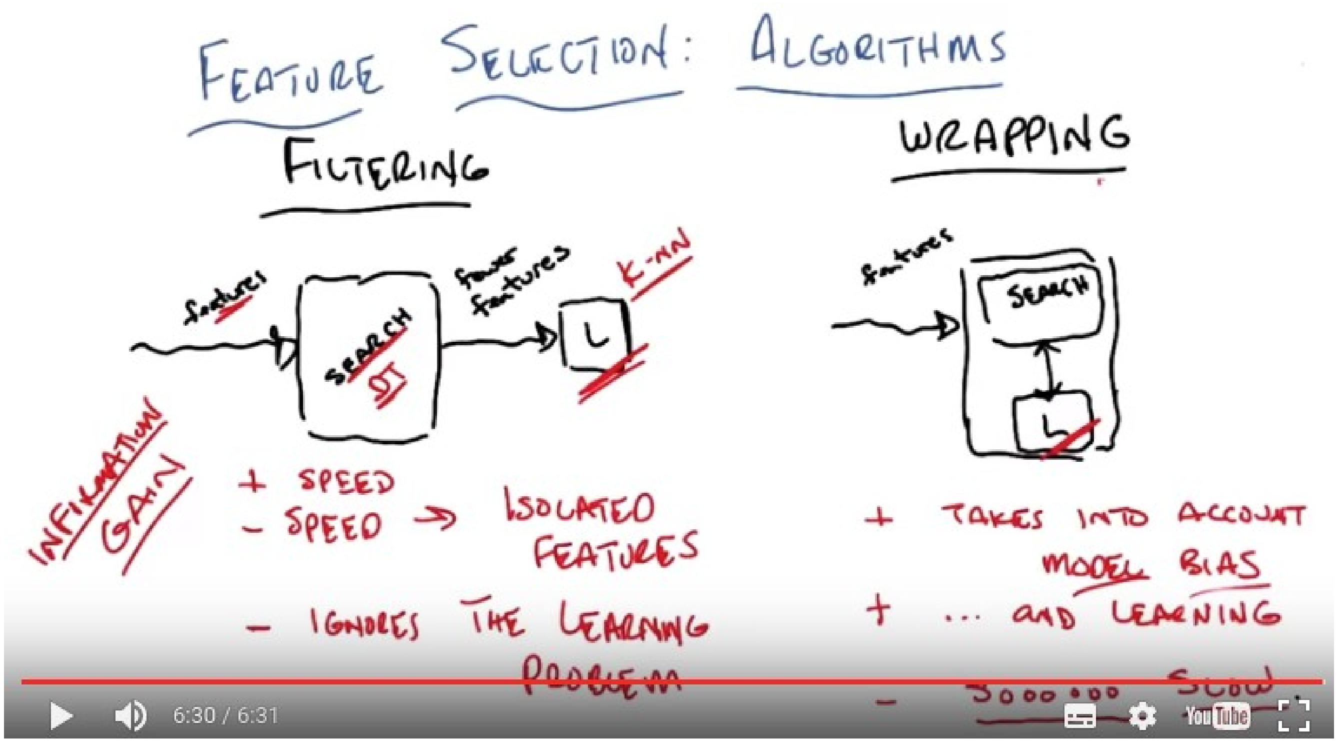
to define my terms just a little bit better but maybe not that much better. So, here's the quiz. You've got a set of data with N features. And your goal is to find a subset of M features. Where the size of that subset is no greater than the original subset. Which makes sense. The original set of features in which makes sense. Okay? And that's it. And so, what I want you to be able to tell me is given this generic problem of starting with N features and finding a subset of M features How long does it take in terms of time complexity to do. Is it linear, quadratic, polynomial, or exponential nm? >> Polynomial includes linear quadratic so you mean sort of of polynomial that's not linear quadratic. >> That's right and I mean quadratic as not linear. >> Got it. >> And I mean exponential that's not polynomial or quadratic or linear. >> It's not. >> Well it could be. You could have 2 to the n plus n squared for example, times 0. You could do it. Don't mess with me man. Okay, so you got the problem? >> Just to make sure I understand. So, so, can you draw like a rectangle? Like our input space x? >> Okay. So we've got some input space x. This is a rectangle. >> And it's got n features, so those are like the columns. >> Okay. >> Alright. And we're going to select a subset m of those n features. So we'll have a like, a skinnier matrix, but not shorter. >> Mm-hm. Right. >> Alright, so and, can I choose any way I want to to define what subsets I like better than which other subsets, because then I can do this really fast. >> So what we're going to do is there's going to be some function, let's just call it f, okay, which given a set of features, return a score. Oh, that looks sort of familiar. >> Mm-hm. And this function may be arbitrarily complicated, it may be really simple, but you don't know what it is. >> Oh. Alright. Then I think, I think I can answer this question now. >> Okay, so let's go.

4. Okay Michael, you think you know the answer? >> I do. So, if you hadn't said, yeah, up until the very end I was thinking along, different lines. So, if it's the case that we just want to choose, if, if this f thing that is scoring is say, constant, then choosing the best subset is easy, I can just choose any subset I want. But you said then, it (f) could actually be arbitrary and it is unknown. So the only way that I can know that I have the best scoring subset is if I try all the subsets. And there's an exponential number of subsets. So I'm going, I would go with exponential. >> That's correct exactly what is the, the form of the exponential? >> Oh, I want to say n choose m. >> Yeah, but you don't actually even know what m is. N choose m, n choose m is right and it gives you an exponential. >> Good, alright. Thanks very much. >> But, the other way is if you don't what m is before you start. Then, it's just 2^n (因為每個 feture 都可能 被選中 或 沒選中). >> It's hard for me to imagine what that means given that you made that part of the input. If you mean just the best subset, then yeah there's 2 to the n subsets. I agree. >> So either one's fine because it gives you the same answer. Sometimes we say, we want half as many or, we want no more than half as many, and sometimes we say we don't know what the best subset is. You figure it out. So this becomes like a kind of clustering problem in its own way except you don't know what k is. But the other way of thinking about this is that of course this problem is hard, of course it's exponential, it is effectively an optimization problem over a set of arbitrary discreet variables. >> That's how I was thinking about it because we always seem to come back to that. >> Right and in fact this problem is known to be NP-hard. And it's exactly because you have to find all possible subsets, so they match the three set. I will not prove that but it does turn out that this is, in fact, exactly the hard problem that you think it is. So given this, and, and by the way, of course, of course this problem is hard. Because if this problem weren't hard then most machine learning would be pretty easy. So, it's not that big of a, it's not that, it's not that surprising that the problem is difficult, so the real question we have in front of us is given that we've got yet another really difficult problem, in fact, a difficult optimization problem, how might we go about tackling it? And it turns out that there are two ways in general that people try to approach this problem.



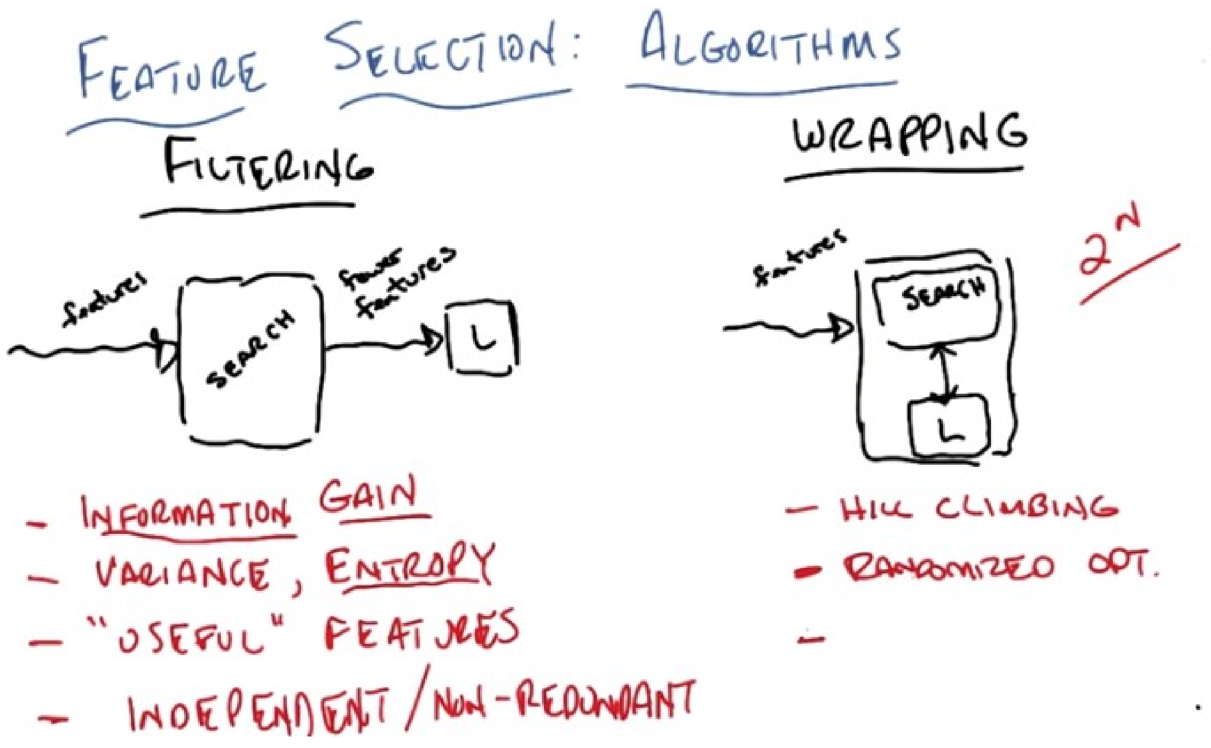
5. >> Okay, so we agreed, Michael, that the problem's pretty hard. It's exponential in the number of features in this case N. And there are sort of two general approaches that people have used to try to attack this problem, broadly speaking. The first is called filtering, and the second is called wrapping, with a W, not an R. >> Hm. >> Well, it has an R too, but it starts with a W. And they actually look pretty similar. So I drew these little cartoons, which I hope would, would make it easier for you to see. So filtering basically works like this. You have a set of features as input. You run them through some algorithm that maximizes some kind of criteria. And then it outputs fewer features. >> Mm-hm. >> And passes those to some learning algorithm, that willl then use it for classification, say. It could be a regression, more generally, but you know, without loss of generality let's assume that we're worried about classification problems here. So you'll notice then that the criterion of how you know whether you're doing well or not is buried inside the search algorithm itself. By contrast, you could do what's called wrapping. And in the wrapping approach, you've taken your set of features, your N features, and you run them into this little box that basically searches over a subset of features, asks the learning algorithm to do something with them. The learning algorithm basically reports how well it does. And it uses that to update this new subset of features that it might look for, and passes to the algorithm (注意此情況下 search 和 L 間的箭頭是雙向的). So, this is called filtering because this is a process that filters the features before handing it to a learning algorithm. And this is called wrapping because the search for the features is wrapped around whatever your learning algorithm is. >> Hm. >> You see that? >> I think so. >> Okay, so you might ask yourself, okay great, so you got two different approaches to this. Which one might you pick? And, unsurprisingly, there are tradeoffs. So, let's talk a little bit about what those tradeoffs are. Actually, before we do that, Michael do you have any suggestions for what would be good or bad about filtering versus wrapping? >> So filtering, it feels like is how we were talking about it before. It's this idea of whittling down to a set of fewer features and then handing it over, kind of one thing after the other. I, an advantage of that seems like it's sort of very flow forward, right? So it, it, the features come in, the search does its thing, it hands it over, everybody's got its job. The disadvantage of that, though, is that there isn't any feedback. So the learning algorithm can't inform the search algorithm as to hey, you know, you gave me this feature, but without that, you took away that

feature, but without it I'm having a lot of trouble doing the learning. Maybe you should put that one back in. The second, the wrapping approach, I guess, would allow for that. >> In fact, that's exactly right. So as I said before, in the filtering case, the criterion or criteria is built inside the search algorithm itself without reference to the learner. By contrast, in wrapping the criteria, or criterion if it's a single one, is actually built inside what the learner wants. It's built inside something like classification error, or sum of squared errors, or whatever it is you measure the learner by. So we can write those down and then stare at them for a little while.



6. So, let's unpack a little bit about what you said Michael. You, you're exactly right in everything that you said and we can sort of write down these things I think more generally be simply noting a few of the pluses and the minuses, the pros and the cons. So in the filtering case one of the things that you, you didn't say explicity but I think was built into what you said. Is a question of speed, so filtering is faster than wrapping because you don't have to worry about what the learner wants, you don't have to worry about paying the cost of what the learner is going to do. You can basically just apply any fast algorithm you might imagine that takes a look at the features and does some sort of filtering. It is of course, as we pointed out before an exponential problem but you can do some kind of an approximation here and you can make a. Basically arbitrarily as fast as you want it to be. Which I think is what you were thinking about originally when we started the quiz. So you do get speed and the price that you're paying for that speed it that you tend to look at the features in isolation. So what do I mean by that. Well you may look at a feature and say this feature is unimportant, for whatever reason. But maybe that feature is important if you combined it with another feature. Say maybe we're in a kind of of parity problem again. And so some feature might look unimportant but actually, in fact, it is important. And the only way you're going to get the kind of speed up that you need is typically by looking at features in isolation. Either adding them or removing them and of course, the reason it gets this speed, is that it ignores the learning problem itself. At least the learner itself, if not the learning problem. Now, in contrast wrapping actually takes into account, model bias, it takes into account whatever the learning bias of the learning alogarithm is, in order to determine what the best sub features are. Which means it's actually worried about the problem of learning, itself. Which, I don't know how to spell, apparently.

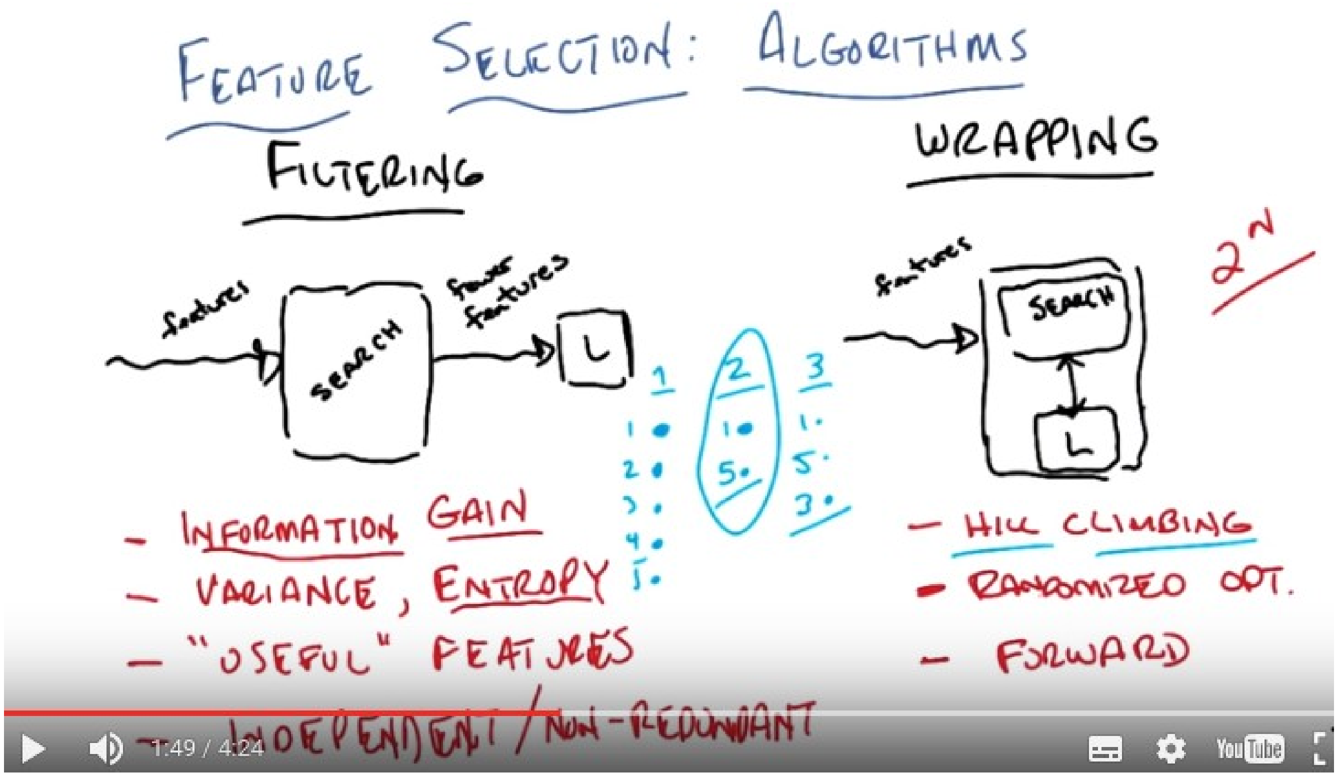
[LAUGH] But in the process, it's much much slower. Because every time it tries to look at a particular subset, a candidate subset, it has to run the learning algorithm. Imagine this learning algorithm is something like neural network learning. This could take thousands and thousands of iterations, you might have to do cross validation, there's all kinds of things you might have to do before you even get to look at the next thing and and get a score. And so wrapping makes a lot more sense because it takes into account the learning problem and im more importantly the learning bias or the inductive bias of the learning algorithm. But, it's going to take a very, very long time to do it while filtering tends to go pretty well. You buy that? >> Yeah. That makes sense. >> Okay. So I'm going to ask you a question before we, sort of move on. Can you think of any algorithms that we've looked at in the, the last mini course, on supervised learning? It basically was already doing filtering. >> So, it's not quite the same, because that's supervised, and now we're talking about unsupervised. But we did look at some algorithms that explicitly decided which features to include, and which ones not. I'm thinking of decision trees, maybe. And, I think boosting kind of has a little bit of this depending on what the weak learner set is. >> Right, that's true. I mean, we, the problem with, boosting is that it's sort of not. It's not picking the features as much as it's picking examples. But certainly with decision trees, you're exactly right. Decision trees are in it's own way a kind of filtering algorithm. So, this might not be obvious, and maybe I could have been clearer about it, you get these features and I think you actually asked me this question. But you get these features, but you do know what the labels are if there's a supervised learner built it. If you're going to have a learner at the end of the day, you, it's not considered cheating in the filtering case. To look inside the learner, but its not consider cheating to understand what the labels are, so you can take advantage of the actual labels of your examples to do your filtering. So in fact if you take the main part of the decision trees which is the information gain thats actually a way of doing filtering on the features. So the criterion is information gain. Find me the features that provide you with the most information given the class label and that gets exactly the subset. >> Cool. Now I don't, I don't quite see how to iterate that though. So I can see how you can get the first feature that way, then are we talking after that we're going to select features that. Add information above and beyond what we've already gotten out of the other features we've selected? >> Sure. So in fact, imagine that my search algorithm here is actual decision trees. So I take a bunch of features. I take a bunch of labeled data. I run my decisino tree algorithim. It picks a subset of the features. Or maybe I do pruning or cross. Whatever. Assisted arbitrarty search algorithm. I just drew a box here. And then what I do is I take all the features that were used by the decision And I pass that to my learner. >> Hm. >> Now the learner is a decision tree learner. I presume we get back the same decision tree. But maybe that's a way of picking a subset of features that are useful for, for example a new network or perceptron or KNN. In which case, I don't quite see how to do the fed, the filtering. >> Well, you run a decision tree algorithm here. [CROSSTALK] >> Oh. >> That gives you the set of features that the decision tree thinks is important. >> Oh. >> And you return that set of features. Basically, the entire, not the tree, but you flatten the tree and then just turn that into a set of features. So imagine, for example, that we actually put inside this box a decision-tree learner. That uses the set of data as input with all of its features and all of the labels that it knows about. It builds a decision tree which gives you typically a subset, well in fact by definition, gives you a subset of all the features. And then the features that come out of the decision tree are passed on to another learned like for a example a perceptron or a nueral network or even something like K and N. [UNKNOWN] >> It always comes back to K and N. >> Everything comes back to K and N. [LAUGH] >> So let me make sure i understand we run a decisoin tree learner but we dont want to use what it actually determined? Instead, we say let's just look at all the things that got split on. And the union of all those now become features to hand off to some other learning algorithm. >> Right. >> But, why would we want to do that if we already trained up the decision tree? >> Because maybe we're worried that the decision tree doesn't do as good a job as you would like. On noisy data for example. Or you're just going to run the decision tree to some depth. Or, even, what you want to do is, you're going to run the decision tree until it actually overfits. And now you know all the features that you could use, and you pass that on to some other learner, which has a different bias. So basically, you use the inductive bias of the decision tree to choose features, but then you use the inductive bias of your other learner in order to do learning. Neat. >> Right, and if you think about the big difference between for example KNN and decision trees, we know that KNN suffers from the curse of dimensionality because it doesn't know which features are important. So this, and but the decisions trees are very good at figuring out which features are important, at least for predicting the label. And so you can ask you're decision tree to give some hint to KNN which has other nice things going for it, which features it should actually pay attention to. >> Cool. >> Alright so there you go.



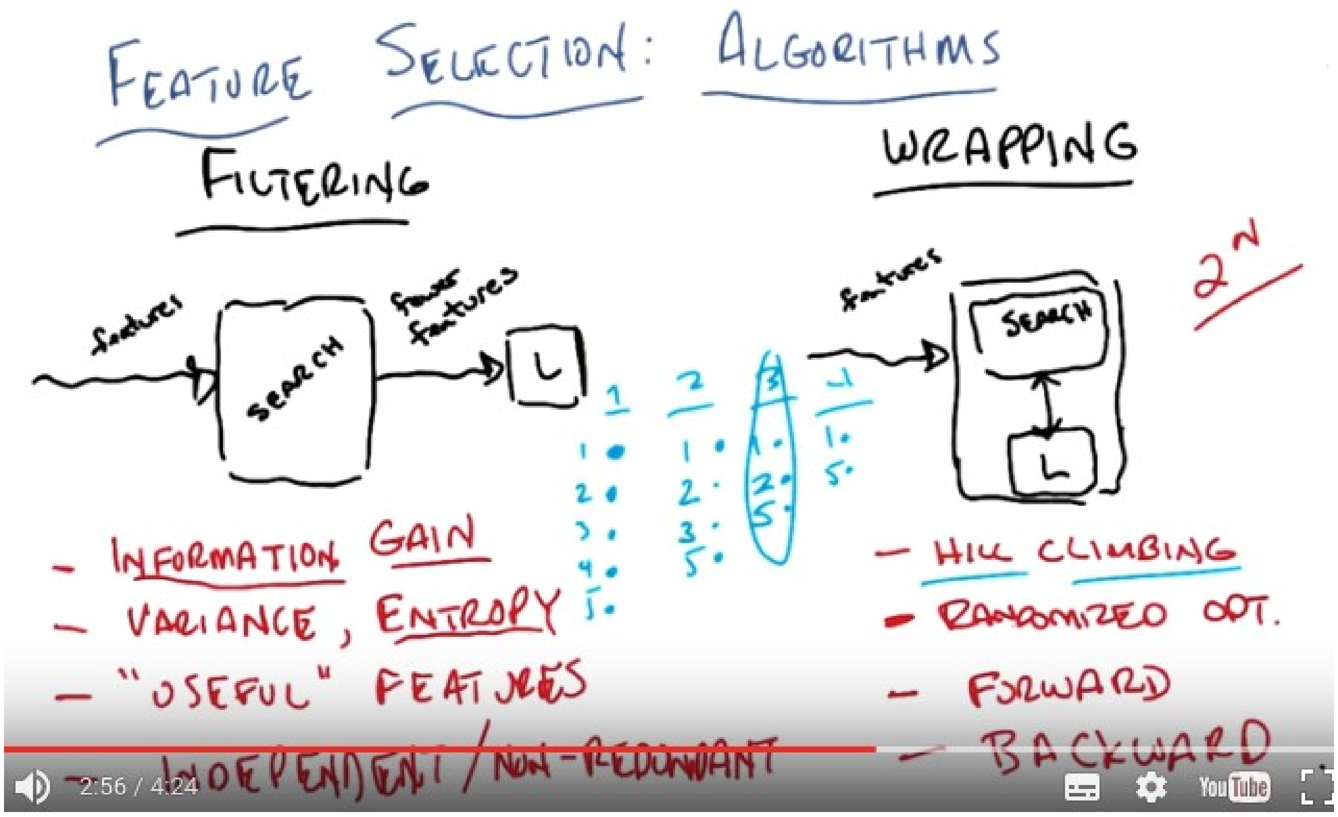
左邊的紅字是 Filtering, 右邊的紅字是 wrapping. Wrapping 最後一點沒有, 是下一段要講的.

7. Okay, Michael. So, I just gave you, you know, one particular way you might do filtering. But let's see if we can expand on that a little bit and come up with some similar ways we might do wrapping. Okay? >> Sure. >> Okay so, I just gave an example of a kind of filtering you might do through decision trees. But really what I'm doing there is I'm defining some criterion. And in this case it was information gain. >> Right. >> This is the way that I will, this is the function that I will use to evaluate the usefulness of a subset of features. How much information does that feature give me about a class label? Can you think of any other kind of filtering criteria we might use? >> So, I don't know, variance, like so, I want features that have lots of different values. That'll do something similar to information gain, I suspect. >> M'kay. Yep. I like that one. So, we might call that variance. Or actually another version of that would be entropy. And, there's tons of ways to do this. Right, so, there's, there's something called the G index which is a kind of diversion of entropy. There's variance, as you telling me here that you're trying to pick features that show up a lot. Anything else? >> I could imagine running a neural net and then pruning away any input features that have low weight. >> Okay. How would we describe that? That's that's actually a kind of information gain too right? Because if the neural network doesn't need it, then somehow it doesn't seem to be terribly useful. In fact why don't we just write that down. >> Oh, I have another one. >> Mm-hm. >> So, if there's features that are linearly dependent on other features

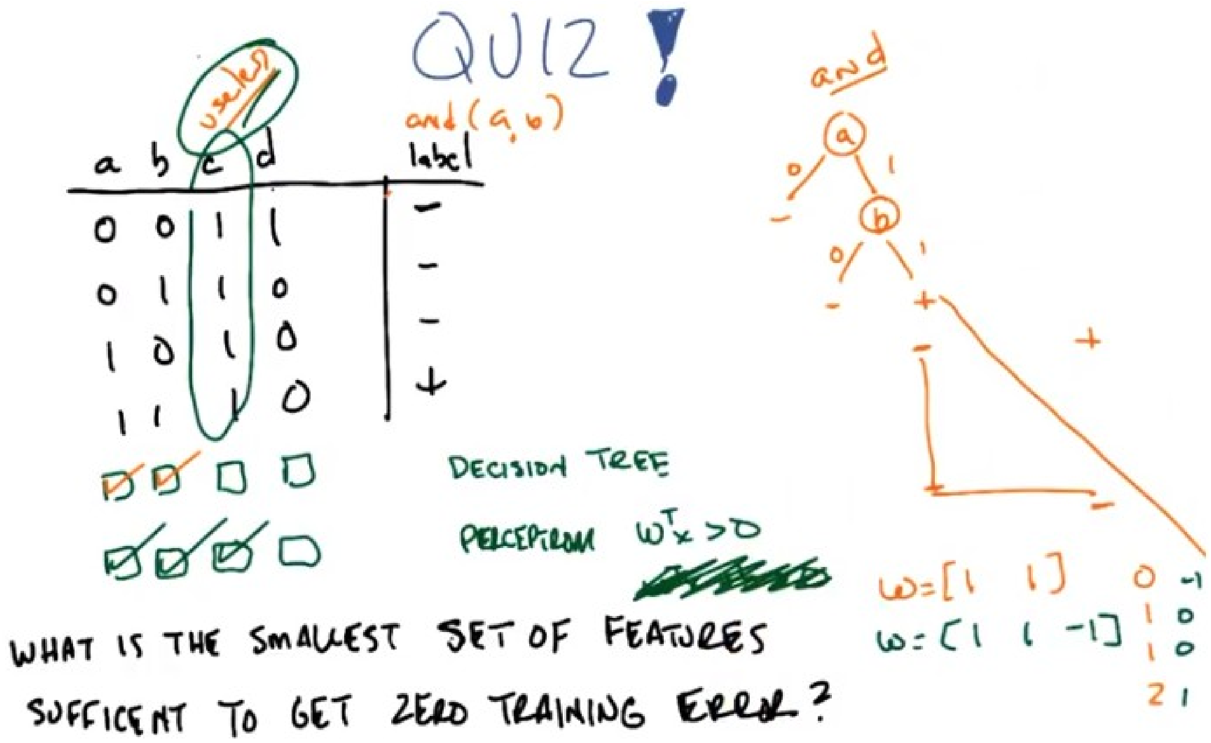
then maybe get rid of them. >> So you want independent features? >> Yeah. >> Or maybe we'll say non-redundant. >> Oh, but in this, in this kind of of linear algebraic sense. >> Right. So if I have feature. Let's say x2 which turns out to be equal to, you know, x1 plus x3 then I should probably get rid of x2. Is that what you're suggesting? >> Sure, yeah, that's that is what I'm saying. >> Okay, and that makes sense because I don't need x2. >> Well, in principle it gives, yeah as you said redundant information. But it also is the case that, for some learners like neural nets, it shouldn't really, you'll feel the difference. For things like decision trees, it might matter though because having it, having those sums together in one place could be more useful to split on. >> That's true, but I do think you said something important there, which, you said it shouldn't feel the difference. You're defining whether this is good or bad in terms of whether it helps you to learn. But if we can get rid of a feature, then we save, as we get rid of features, we save exponentially on the need for data. >> Yes. So as long as it doesn't make the learning problem harder, that's a good thing. >> Right. So long as it makes the learning problem easier, in terms of the amount of data you need, it's a win, up until the point where it makes the learning problem harder because you don't have enough features to do the actual learning. And there's a bunch of these. And I think all of these are probably good answers. You could think of all kinds of statistical tests or relevance. I mean basically are built in here these are all different ways of doing the same thing which is figuring out which features you need in order to do learning by some generic criteria. I do want to point out that something like Entropy doesn't even depend on the labels. Where something like Information Gain which is a kind of conditional Entropy, doe's depend upon knowing the labels. What about wrapping? How would you go about doing the wrapping? So we got a bunch of different ways we might do filtering. What are sort of the way in which you could imagine doing a wrapping? You got some learning, don't know what it is. You got a bunch of features you want to search over those, pass those features on to the learner and see how well it does. But you don't want to pay an exponential cost, by looking at all possible subsets. What might you imagine doing? >> Well, so, I mean we've already talked about a bunch of algorithms that can search, that aren't necessarily exponential. So various kinds of local search and "hill climbing" could be a way of doing the search part. >> Hmm, I like that. Let's write that down, "Hill Climbing". Now when you say Hill Climbing, you really mean like a gradient, a regular deterministic gradient search? >> Hmmmmmmmmm, yeah. I mean, I guess I was thinking of something where what does the search algorithm does is it tells the learning algorithm here's the features to try to use. The learning algorithm runs and it comes back with some kind of error. Presumably, on how that validate data otherwise, we run the risk of over fitting. >> Fair. >> And it gives a number back and then the, the search algorithm uses that to decide what subset to try next. Okay so you imagine that it's going to jump in some gradient directly, or is it going to have to do something like randomize optimization. >> I think it would have to, yeah I mean like, randomized optimization. >> Okay so, in general we can take almost everything we just did, and the last lesson on randomized optimization and apply it here, so then we might just say. Just do random. >> Oh, I see. So could be, could be mimic. It could be, genetic algorithms. It could be simulated annealing length. >> Right. So you can think of, you can do randomized optimization. And that's going to be important precisely because we want to solve, this exponential problem. We don't want that to be an issue. Okay that makes sense. Now we've already thrown out exhaustive search where we look at all possible features because, again, that's exponential. Is there anything else you can think of? I'm thinking of one more, so let's see if you can read my mind. >> [LAUGH] >> And really, when I say I'm thinking of one more, I'm thinking of two more. >> Oh. That's, that are not randomized optimization algorithms, but they're not going to somehow search. >> Mm-hm. >> A star? >> Yeah, but that's not what I was thinking. >> Oh. I give up. >> I'll give you a hint. You could go forward or you could go backward. >> EM. >> [LAUGH] No. No. No. No. Although I like that. >> ME? >> [LAUGH] That wouldn't be forward or backward, would there? No, I'm thinking of exactly a forward sequential selection and backward elimination. >> I don't think I know what that means. >> Oh. So let's talk about that then.



8. So, forward search simply means you start with a feature. Let's see if I can draw a picture. So in forward search, you simply start with a feature, of your end features. You look at all of your features in isolation. Let's imagine we have five features. And I say well, I'm just going to pass you the first feature. Pass the first feature to the learning algorithm, I get a score. Then the second feature, then the third feature, then the fourth feature to the fifth feature. And which ever one of those features is best, I keep. So let's say it's the first one over here. So now I keep the first feature. And then I say, well now that I'm given the first feature, let's look at each of the remaining features and see if I add them, which one does best. So, I end up with one, two, three, four, five. I pick the best one. And let's say that turns out to be feature one. And then I look at feature one in combination with two and combination with three and in combination with four and in combination with five. Whichever one has the highest score that's the one I move forward. And so that would be let's say in this case worked out to be feature five and now that I have these two features, I pick a third feature of the ones that are remaining, two, three and four. And whichever one of those in combination with one and five I add next, and let's just say that turns out to be feature three. But it turns out the error, or the score that I get for adding the third feature to one and five, isn't significantly better than just having one and five, and so I stop here and say, that's the subset of features that I would take. Does that make sense? >> Yeah, that seems like a really neat idea and I could see that as being a kind of efficient polynomial way of searching this space. Though couldn't, could I think of it as a kind of hill climbing, though? Where, where my neighborhood relation is basically adding one more feature? >> Sure. Absolutely. And if it's only adding one more feature, that's exactly what you're doing. You are doing a kind of, a kind of hill climb.



So, in fact, the, alternative, or the sort of complement to forward search is backward search. And backward search is, I think, by the way, you describe it as exactly another kind of hill climbing. We're here we start out with all of the features, and you say which one can I eliminate, so I try one, two, three, four (eliminated five, 下同). I try one, two, three, five. I try one, two, four, five. I try two, three, four, five, and whichever four work out best are the ones that I keep in the second round, at let's just say that turns out to be one, two, three, and five. And then I see, well, of the remaining four, which one can I eliminate and still have the best score? And I keep doing that until I get down to a subset that still does pretty well. So let's say I get to this case, and it turns out, you know what? One, two, and five do bet, just, almost as well as one, two, three, and five. But once I go to just one and five, it does much worse and so I'm going to stick with that. So here you're right, I liked the way you described this, Michael, as hill climbing. Here I'm doing hill climbing, except my neighborhood is removing a, a single point instead of adding a single point. >> Yeah, that's cool. It's, for some reason, reminding me of like making it team, like you could, I could imagine tryouts where you put everybody on the field and you have them play, and whoever's helping the least, you kick them off the team, so that's kind of the backward case. We just keep kicking people out until we have the team we want. And then the forward case is kind of like the dream team model where you start off saying, I got nobody let's take the best person around, add that person to my team. And, keep doing that until we have a full set of players. >> Okay. So this all makes sense? >> Yeah. >> Okay. Just for, just to summarize, filtering algorithms don't necess, while they do, then can take advantage of a label (應該是 learner 弄出來的) they don't really care about the learner. And so you have to come up with criteria that all probably makes sense, like the ones you came up with. Information gain, entropy variance, and you can tell stories for why all of these make sense. This by the way is where your domain knowledge comes in. It's the choice of the criterion or the optimization function that, that you're using. But they have a problem in that they don't actually take advantage of the bias of the learner, where wrapping definitely takes advantage of the bias of the learner. And then you just hit, simply have a question of how to deal with the exponential search problem by doing, I think in all of these cases, what looks a lot like hill climbing. You just have different methods for thinking about how to do the hill climbing. Do you add variables, or remove variables, or do you do some kind of random search. But ultimately, in both cases, you have some function you're trying to optimize over.

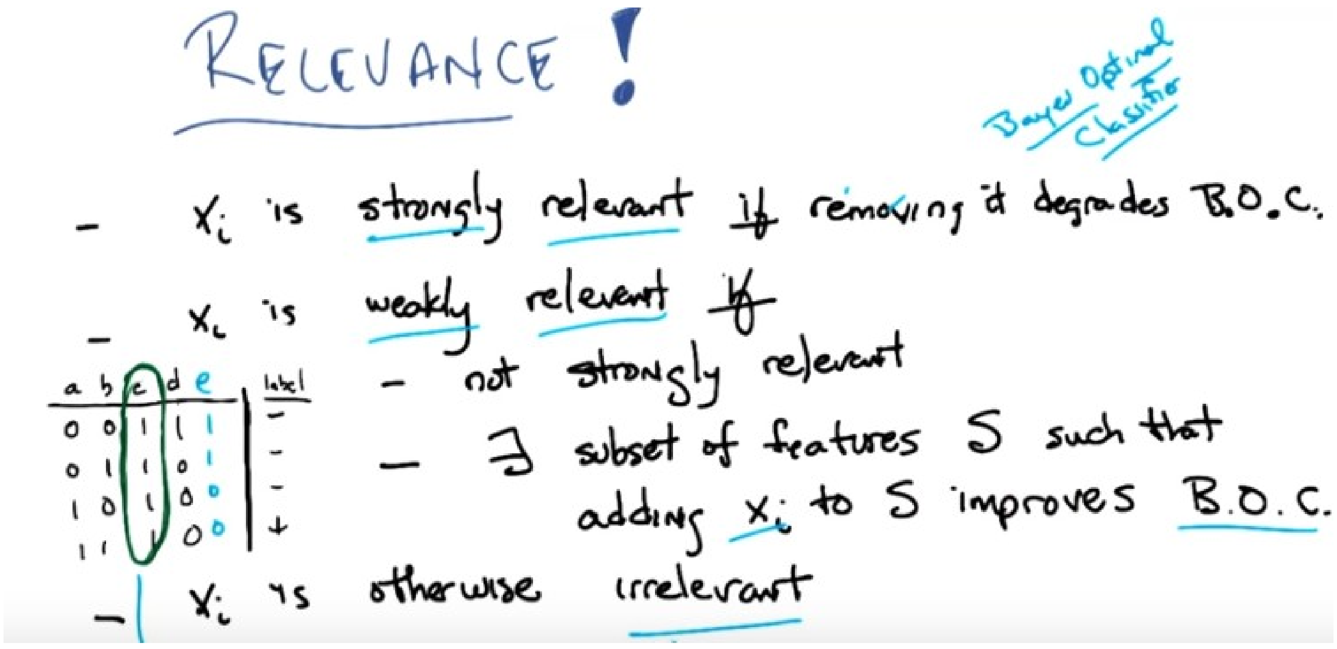


9. Okay Michael, so I would like to give you a quick quiz. >> Thanks. >> You're welcome, Michael. I'm about helping others. Okay, so here's the problem. Let me set it up for you. You let me know if the, if, if the question doesn't make any sense. Okay, so we have a set of fea, we have a problem, we have a learning problem that has four features in the input space, and I'm labeling them a, b, c, and d. And here's a table that has some training data. You'll notice that it's all binary, I'm representing the binary as zeroes and ones here, and that's the space that they're in. And I have a label, minus or plus, which you could also think of as zeroes and ones. Okay? >> Okay. >> And, here's what I want you to do. I want you to find the smallest subset of features of a, b, c, and d, that's sufficient to get zero training error, for these two different learning algorithms, decision trees, the first one, and the second is perceptron, but a very simple perceptron where you simply have w transpose x greater than zero. If it's greater than zero, you return positive. If it's zero or less, you return negative. Got it? >> I think so, yeah. >> Okay, and so what I have out here are a bunch of checkboxes and I want you to check off the variables that you need, or the features that you need, in order to allow decision tree to get zero training error or to allow the simple perceptron to get zero training error. >> And are we supposed to do this thinking of it as a filtering thing or a wrapping thing, or? >> Yes. >> I see, so you just want to ans, us to answer the question. >> yes. I want to answer the question and as a hint, you should be thinking about this as a filtering or wrapping problem. >> [LAUGH] I was kind of hoping you would tell me which of those two. >> Well let's see, you were really good at picking filtering, so think about it as filtering. >> Oh, okay. >> So, so, I'm not going to ask you, I'm not going to require as a part of the quiz, that you tell me what criterion you were using, information gain, or entropy, or variance, or anything like that, but when you give me the answer, I'm going to ask you to explain it to me, because I'm curious what you're thinking, okay? >> Okay, but I'm allowed to use the label to answer it too? >> Yes, you're allowed to

use the label if you want to, if you think it's necessary. >> Right, because filtering methods, I, well I guess you said that they, they can have access to the label. >> Yes. >> Okay. >> You got it? >> I think so, yeah, yeah. >> Alright, good. Okay, okay. Well so, go.

10. >> Okay, Michael. You got an answer for me? >> Yeah, I think so. I kind of like the method of dropping features that we don't need. So I'm going to see if I can apply that idea. >> OK. >> So, for the decision tree case, if I had all 4 features, could I even learn this thing? I think I could. Let's see, I could do A. And if A is true, then I can split on B. And if B is true, then I output A plus, otherwise I output minuses. >> So, hm. Is that a function? Do you know what function that is? >> It's a decision tree. >> Yes, but what, do you know if there's a Boolean function that decision tree represents? >> I don't know. Do you want to draw it and I can look at it? >> Yeah, sure. Let's do that. So first you split on A. Then you said A is either zero or 1. And then in the case where it was a 1, you split on B. And if that were a zero, you output no, and if that is a 1, you output yes. And over here, you output no. That's what you said, right? >> Yep. And that works for this example. >> Mm hm. >> Or this set of examples. And, let's see, so with 1 plus, oh I see. So it's it's AND right? So it, if A and B are true, then the output's true, otherwise not. >> Right. And in fact that is exactly the function that I was thinking of. >> Oh. >> So this, the true underlying function here is in fact AND of A and B. >> All right. And oh, I see, so in particular, now we've answered the first question. Because I, I need, well okay I, I, I only need A and B. And in fact, having just one of them alone isn't going to be enough. If I just had A then the decision tree can't learn it. And if I just had B, then the decision tree can't learn it. It needs more information. >> Right, and even in these examples you haven't, you can't really luck up and have picked C or D to learn this. Because C is always 1, and so you can't output the correct answer for all cases. And D is 1 in the wrong place, and zero in the wrong place, but zero in the right place a couple of times. So you can't do something like take the not of D and haven't gotten lucky. >> I see. Well yeah, and C in particular seems exceptionally useless. >> Ooh, I like that word. >> [LAUGH] >> I am going to say useless. Useless. Okay, fine. So, that's right. So, it, and maybe this helps you a little bit with the next one. I know that you now know what the true function is. It's AND A or B. And you've decided that D doesn't help a lot, and C is completely useless. Why'd you say it was useless, by the way? Because it's the same in all the cases. It doesn't discriminate between the positive and the negative labels. >> That's right. So it is useless in that regard. So, good. This is the answer I was looking for, for decision trees. What about for the simple perceptron? >> Okay, right. So I'm not great at doing this kind of thing in my head. But I guess the good news is, oh, how about this? How about this? Since we got it down to only these two features matter, we can draw it as a two dimensional plane, right? >> OK. >> So mark 0,0. Put a minus at 0, 0. >> OK. >> Put a minus at 0,1. >> Mmhm. >> Put a minus at 1,0. >> Mmhm. >> And put a plus at 1,1. So that's when we're using the A and the B features. >> Yep. >> And this is great, because it's linearly separable. So we just need to, yeah, we just need a set of weights that can do that. And let's see. So, how do we do that? So, if we add, let's see. If we do a weight of 1 on A, and a weight of 1 on B. >> Mm-hm. >> Then. >> Then. >> Let me, let me draw it. Maybe that will, that will help. So, you want to pick a weight vector which is, you said, 1 for A and 1 for B. >> Mm-hm. >> Okay, all right. Go. >> And that will give us for the first example, that gives us an output of zero. For the next one an output of 1, the next one an output of 1, and the last one an output of 2. So, we would like, if it was like, uh-oh. Wait. So, if it, if the weight vector was that, then what we really want it to be is w^T x greater than 1. But we can't. It says zero there (意思是題目中要求了的只能用 w^T x > 0). >> Mm hmm. In fact, wait, isn't it the case that if you don't have, if it's greater than or equal to zero, it's, we're really talking about lines that go through the origin. >> Yep that's right. This is an origin limited perceptron. >> So it seems like we're hosed, right? There isn't going to be a separator that goes to the origin that lops off the plus. >> That's right. >> So, I guess what we usually do in perceptrons is we include, was it like a bias unit, like a, like a, something that's all 1's. And then that lets us move the threshold around. >> Right. So normally this would look like this. W transpose X

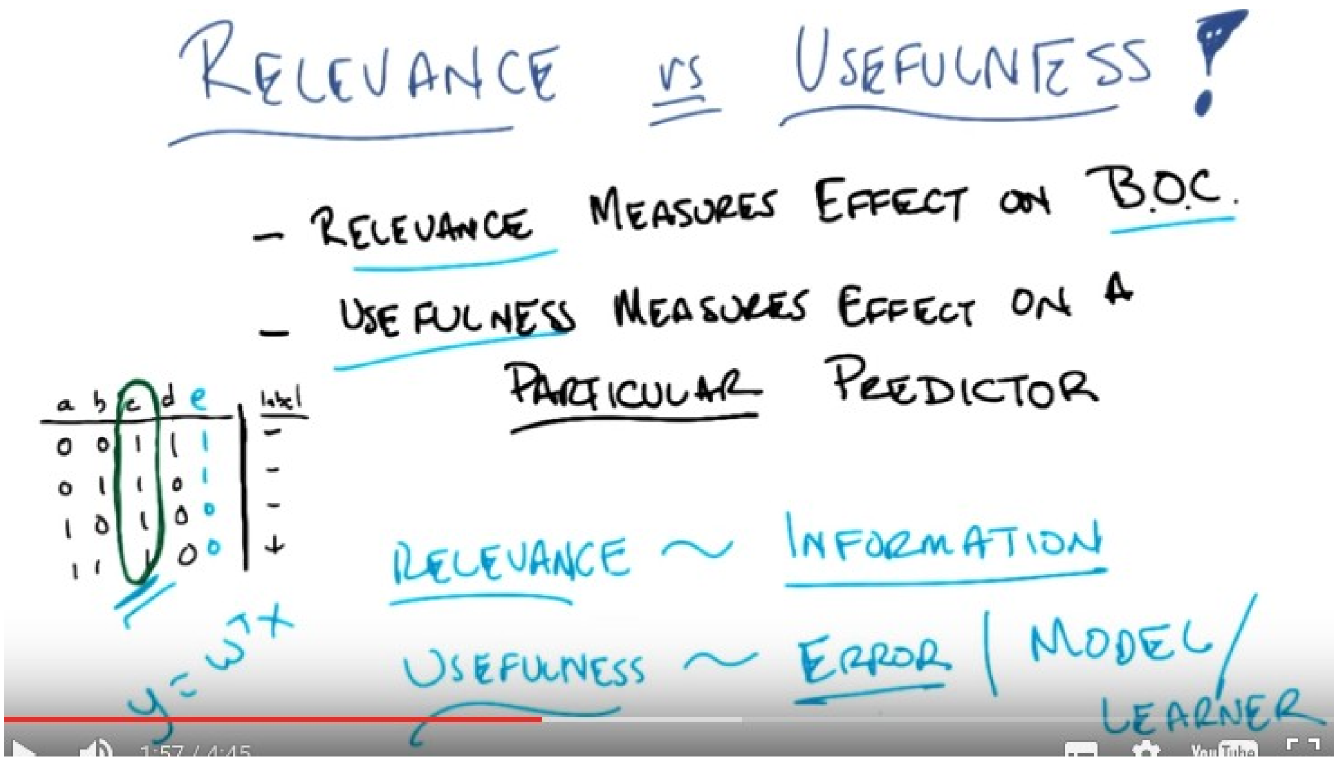
plus B is greater than zero. But I didn't let you have that, so you can't have that. >> But maybe I can. You say I can add B in. What if I could add C in? Because if you look at this >> Well, what would that give you? >> Well, look at the C column, it's all 1's. It's the same as if it was, it like, the thing that you called B a moment ago, the, this, this bias unit. So if we actually include A, B, and C as our features, then we can represent this with a, what did you call it, origin limited perceptron. >> Right. So then, and I'll even draw it in a different color, if we had WB, what did you have? 1, 1, and. >> I'm going to say negative 1. >> Negative 1, and that's A, B, and C. Then what would that give you, in the first case? >> So negative 1, zero, zero, 1. >> Yep. >> So it's only, the positive example is the only one that's greater than zero, strictly greater than zero. >> Yep, that's exactly right. So, you still think C is useless? >> I see. Well, when I said useless. >> Mm-hm. >> I meant it didn't give any information. >> And that's exactly right. It doesn't give any information. But that does not make it useless. >> Interesting. Okay, well, I now regret that word choice. >> C's everywhere across the world are thanking you. [LAUGH] >> So this actually brings me to the last point that I want to make about feature selection, which really gets down to noticing the difference between useless and what I'm going to call relevant. So let's dive into that, okay? >> Yeah, that, that would be really helpful. Is that, and that's the right answer? The first three? >> Yes, that's the right answer. >> Cool.


RELEVANCE!
Bayes Optimal Classifier
- x_i is strongly relevant if removing it degrades B.O.C.
- x_i is weakly relevant if
- not strongly relevant
- ∃ subset of features S such that adding x_i to S improves B.O.C.
- x_i is otherwise irrelevant
a b c d e label
0 0 1 1 1 -
0 1 1 0 1 -
1 0 1 0 0 -
1 1 1 0 0 +

文中有個 weakly relevant 的例子比較可以
From 最後一段: 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.

11. Okay Michael. So let's see if we can jump on that notion of useless and some other word you used. I'm going to define what it was you were trying to say [LAUGH] by calling it relevance, okay? >> Okay. >> So it turns out this thing that you were hanging onto where you were saying that well, C which you thought was useless and then it turned out to be useful and you were. You were coming up with reasons why doesn't change anything, it doesn't provide any information. We can actually make very precise informal. So here's a formal definition of the notion of relevance and i'm just going to go through this. Let me know if it makes sense. So we have a set of features and let's just say they're XI, so there is X1, X2, X3. The X end features and any particular feature X and I. Is going to be strongly relevant, exactly in the case if removing that feature degrades the Bayes Optimal Classifier, that is what

BOC stands for, does that make sense. >> Did we talk about what A BOC is? Yes, way back in the supervised learning. Lesson on bashe learning. We ended up talking about the Bayes Optimal Classifier, the one that takes the weighted average of all of the hypotheses, based on their probability of being the correct hypothesis. >> Okay. >> Remember that? >> Yeah. >> And we, and what we actually said is that the Bayes optimal classifier is the best that you could possibly do on average, if you could actually find it. >> Right. >> That coming back to you? >> Yeah i mean we need a notion of priors and stuff to be able to define strongly relevant. >> No it really just says that there is a Bayes optimal classifier which is to say the best you could do. That X_i is strongly relevant in the case if you didn't have that feature you couldn't do as well as the Bayes optimal classifier that had access to all the features. In the quiz that we did before we know that the actual function that we're looking for really was a and b. So if I remove a, I can't actually compute a and b. Similarly if I remove b, I can't actually compute a and b. So both a and b are strongly relevant. >> Okay, I mean in this case is uses that fact that not only that it's a and b but there is nothing else that has the same information as a and the same information as b. >> Right, exactly. And that little, that, that difference you just noted is the difference between being strongly relevant and being weakly relevant. >> Oh. >> So, a variable, a feature is called weakly relevant if it's the case that it's not strongly relevant (removing it doesn't actually hurt the base optimal classifier). So, the order of these definitions matter. And it turns out that there exists some subset of your features, let's call that subset s, such that if I added the feature to that subset s, it would in fact improve the Bayes Optimal Classifier. >> So in this case, we're talking about the Bayes Optimal Classifier on. Oh, I guess in both cases, we're talking about. The Bayes Optimal Classifier on the reduced set of features. >> Right. >> Comparing it to the Bayes Optimal Classifier on the full set of features in the first case. >> Right, or just on any subset of features. So in particular, in the strongly relevant case, that's the case. We're saying, well, what would the Bayes Optimal Classifier be on all of the features, versus the Bayes Optimal Classifier on all of the features except. X sub i, and if removing X sub i degrades the performance, then we say it's strongly relevant. On the other hand, if X sub i is not strongly relevant, that is, removing it doesn't actually hurt the base optimal classifier. It can still be weakly relevant in the case where there is some subset of the features. Such that if I added x of i to that subset it would improve the Bayes optimal classifier on just that subset. So I can make that a little more concrete: Imagine that we had another variable, let's call it e, ok? Which had these values. So, if you look carefully, you'll notice that e is in fact not a (即非 a). Mhm. >> So that means neither a nor e is strongly relevant, because I can remove a and still learn 'b and a' by basically making it 'b and not e', or I could move e and still learn 'a and b' by simply using 'a and b'. Agreed? >> Yea, so its so then b would be strongly relevant but the other two not. >> Right. However both a and e are weakly relevent. Because there exists a subset such that adding it back in gives you better performance. In particular, a is weakly relevant for any subset that doesn't include e. And e is weakly relevant for any subset that doesn't include a. >> Huh? And that includes the null set in this case? >> Yes, exactly. If you have a particular feature which is not strongly relevant, and is not weakly relevant, then we call it irrelevant. So, in this case, when, what you were calling c as being useless. What it actually is is irrelevant. Because it provides no information to the classifier. You might as well have just simply had the normal subset and just always output no and you would do just as well as having the value of c. >> And yet it somehow turned out to be helpful for the perceptron case. >> Right. And that's because there's another notion that we could think about. Which is not relevance but usefulness. So, let me define that for you, okay? So.



RELEVANCE vs USEFULNESS!

- RELEVANCE MEASURES EFFECT ON B.O.C.
- USEFULNESS MEASURES EFFECT ON A PARTICULAR PREDICTOR

a	b	c	d	e	label
0	0	1	1	1	-
0	1	1	0	1	-
1	0	1	0	0	-
1	1	1	0	0	+

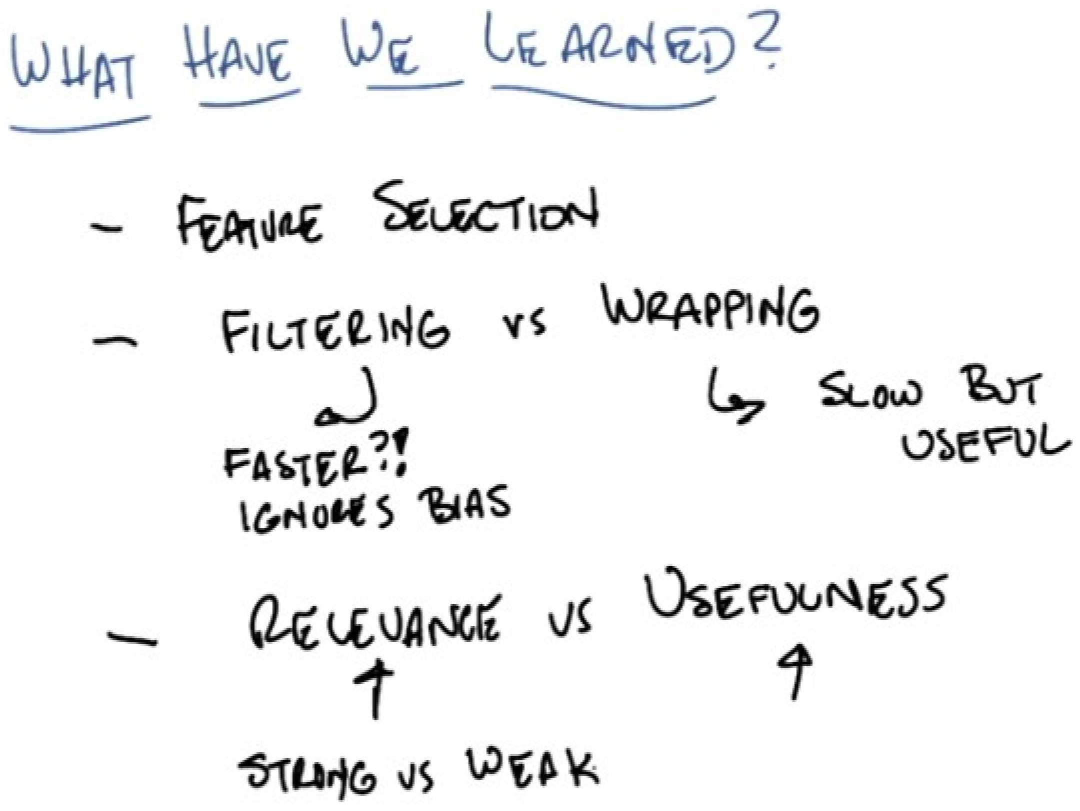
$y = w^T x$

RELEVANCE ~ INFORMATION

USEFULNESS ~ ERROR | MODEL/LEARNER

12. So let's see if we can be a bit more formal about this notion of usefulness. Okay? So, I erased all the stuff I had before but basically you could summarize that last side, as saying that relevance measures the effect on the bayes optimal classifier. Right? So a variable is relevant if it can make the bayes optimal classifiers performance better or worse. So, really relevance is about information. So, when you were talking earlier about things that you might use for filtering you said, well, I like things that have variance or things that have entropy or things that give me information gain. Those are all measures of the information that is present in a particular variable or a particular feature. Right? >> Yeah. >> So really from the bayes optimal classify point of view the only thing that matters is how much information a particular variable provides. You know, conditioned on some label or just in general how much information it provides so a variable like C here which doesn't change has zero entropy, provides no information, independent of the value of the label and therefore cannot be relevant to the bayes optimal classifier. That make sense? >> It does. Now I, and, and the, this this notion of help, usefulness. Not helpfulness, but usefulness. Is when we condition on particular predictor and that's why we can have C being useful in the context of a, perceptron. >> Right. So usefulness is exactly about effect on error given a particular classifier, or some specific model. So usefulness is exactly about minimizing error given some particular model or some particular learning algorithm, right? So, in this case although C is clearly not relevant, it is in fact useful at least for something like w^T x. Now, this is not useful for a decision tree, nor is it relevant. It is not relevant to this particular problem, but it is however useful for some algorithms. >> Now, just to clarify so, so it seems like the base optimal classifier has a privileged position in this definition right, so can we define relevant as measuring effect on, a perceptron. >> Relevance, no. >> No I'm saying, but it, you just, you plugged in, a, a kind of classifier there. Why can't we plug it some other kind of classifier? >> For the base optimal classifier? >> Yeah, why is that one special? >> Because the base optimal classifier captures this notion of the optimal thing that you could do. It's not a specific algorithm. I mean you could write

down an algorithm that would compute the base optimal classifier, except that it requires you looking at all possibly infinite number of hypotheses. Right? But it is, the, it the base optimal classifier computes the best label, given all the probabilities that you could, ostensibly compute over all the hypothesis space. It doesn't have to actually require specific algorithm to do so. It truly is a measure of information of variables. So, any other algorithm you have has a bias, in particular inductive bias. >> Okay. >> [LAUGH] Okay, that's a fine answer. So, at the very beginning of our discussion, Michael, you actually asked me this question of what the criteria was? When I said that features selection was about maximizing some criteria, removing features according to some criteria and I told you that eventually we'd kind of get to the point of what the criteria is. The notion of relevance versus usefulness gives us an idea of thinking about what that criteria is. Ultimately we've been talking about unsupervised learning, mostly in a kind of vacuum but presumably you know wheather some particular description compact or otherwise, some particular label is in fact a good one based on how it's used later on. So, one way of thinking about this is the labels that I come up with for a set of data are exactly good ones in so far as they help me to do something else like classification later. All that clustering that we did before, like with k means and E M. You could think of those as a kind of feature transformation algorithm which is what we'll be talking about next where you've taken a bunch of features and you've converted them into something simple like a label. And whether that label is a good label or a bad label, depends entirely upon whether you can then do some kind of classification or regression problem later. >> Label in this case meaning the cluster name? Right. >> Got it. >> Okay. >> That's interesting. >> So, you'll actually find if you go through the literature and you, you look at some algorithms people have predicted, that a lot of the measures like information gain or entropy or whatever, often end up being couched in terms of relevance. But ultimately, what we really care about is usefulness, or at least one could argue. >> Cool. >> Okay. So that's pretty much what I wanted to talk about for a feature selection. There's a lot of algorithms and, I mean, if you go and you look at the material we've made available to everyone you'll be able to see some of these algorithms discussed in more detail. But at a high level these are the, the key issues that I wanted you to see. So, with that Michael, let's wrap up.



WHAT HAVE WE LEARNED?

- FEATURE SELECTION
- FILTERING vs WRAPPING
  - FASTER?! IGNORES BIAS
  - SLOW BUT USEFUL
- RELEVANCE vs USEFULNESS
  - STRONG vs WEAK

13. >> 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 class fucker 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 (克利普頓石, 一種只存在於超人漫畫的虛構礦物, 可令普通人擁有不同的超能力), 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.