ERIC GRIMSON: OK. Welcome back. You know, it's that time a term when we're all kind of doing this. So let me see if I can get a few smiles by simply noting to you that two weeks from today is the last class. Should be worth at least a little bit of a smile, right? Professor Guttag is smiling. He likes that idea. You're almost there. What are we doing for the last couple of lectures? We're talking about linear regression.

And I just want to remind you, this was the idea of I have some experimental data. Case of a spring where I put different weights on measure displacements. And regression was giving us a way of deducing a model to fit that data. And In some cases it was easy. We knew, for example, it was going to be a linear model. We found the best line that would fit that data.

In some cases, we said we could use validation to actually let us explore to find the best model that would fit it, whether a linear, a quadratic, a cubic, some higher order thing. So we'll be using that to deduce something about a model.

That's a nice segue into the topic for the next three lectures, the last big topic of the class, which is machine learning. And I'm going to argue, you can debate whether that's actually an example of learning. But it has many of the elements that we want to talk about when we talk about machine learning. So as always, there's a reading assignment. Chapter 22 of the book gives you a good start on this, and it will follow up with other pieces.

And I want to start by basically outlining what we're going to do. And I'm going to begin by saying, as I'm sure you're aware, this is a huge topic. I've listed just five subjects in course six that all focus on machine learning. And that doesn't include other subjects where learning is a central part. So natural language processing, computational biology, computer vision robotics all rely today, heavily on machine learning. And you'll see those in those subjects as well.

So we're not going to compress five subjects into three lectures. But what we are going to do is give you the introduction. We're going to start by talking about the basic concepts of machine learning. The idea of having examples, and how do you talk about features representing those examples, how do you measure distances between them, and use the notion of distance to try and group similar things together as a way of doing machine learning. And we're going to look, as a consequence, of two different standard ways of doing learning.

One, we call classification methods. Example we're going to see, there is something called "k nearest neighbor" and the second class, called clustering methods. Classification works well when I have what we would call labeled data. I know labels on my examples, and I'm going to use that to try and define classes that I can learn, and clustering working well, when I don't have labeled data. And we'll see what that means in a couple of minutes. But we're going to give you an early view of this.

Unless Professor Guttag changes his mind, we're probably not going to show you the current really sophisticated machine learning methods like convolutional neural nets or deep learning, things you'll read about in the news. But you're going to get a sense of what's behind those, by looking at what we do when we talk about learning algorithms.

Before I do it, I want to point out to you just how prevalent this is. And I'm going to admit with my gray hair, I started working in AI in 1975 when machine learning was a pretty simple thing to do. And it's been fascinating to watch over 40 years, the change. And if you think about it, just think about where you see it. AlphaGo, machine learning based system from Google that beat a world-class level Go player. Chess has already been conquered by computers for a while. Go now belongs to computers. Best Go players in the world are computers.

I'm sure many of you use Netflix. Any recommendation system, Netflix, Amazon, pick your favorite, uses a machine learning algorithm to suggest things for you. And in fact, you've probably seen it on Google, right? The ads that pop up on Google are coming from a machine learning algorithm that's looking at your preferences. Scary thought. Drug discovery, character recognition-- the post office does character recognition of handwritten characters using a machine learning algorithm and a computer vision system behind it.

You probably don't know this company. It's actually an MIT spin-off called Two Sigma, it's a hedge fund in New York. They heavily use AI and machine learning techniques. And two years ago, their fund returned a 56% return. I wish I'd invested in the fund. I don't have the kinds of millions you need, but that's an impressive return. 56% return on your money in one year. Last year they didn't do quite as well, but they do extremely well using machine learning techniques.

Siri. Another great MIT company called Mobileye that does computer vision systems with a heavy machine learning component that is used in assistive driving and will be used in completely autonomous driving. It will do things like kick in your brakes if you're closing too fast on the car in front of you, which is going to be really bad for me because I drive like a Bostonian. And it would be kicking in constantly.

Face recognition. Facebook uses this, many other systems do to both detect and recognize faces. IBM Watson-- cancer diagnosis. These are all just examples of machine learning being used everywhere. And it really is. I've only picked nine. So what is it? I'm going to make an obnoxious statement. You're now used to that. I'm going to claim that you could argue that almost every computer program learns something. But the level of learning really varies a lot.

So if you think back to the first lecture in 60001, we showed you Newton's method for computing square roots. And you could argue, you'd have to stretch it, but you could argue that that method learns something about how to compute square roots. In fact, you could generalize it to roots of any order power. But it really didn't learn. I really had to program it. All right. Think about last week when we talked about linear regression. Now it starts to feel a little bit more like a learning algorithm. Because what did we do?

We gave you a set of data points, mass displacement data points. And then we showed you how the computer could essentially fit a curve to that data point. And it was, in some sense, learning a model for that data that it could then use to predict behavior. In other situations. And that's getting closer to what we would like when we think about a machine learning algorithm. We'd like to have program that can learn from experience, something that it can then use to deduce new facts.

Now it's been a problem in AI for a very long time. And I love this quote. It's from a gentleman named Art Samuel. 1959 is the quote in which he says, his definition of machine learning is the field of study that gives computers the ability to learn without being explicitly programmed. And I think many people would argue, he wrote the first such program. It learned from experience. In his case, it played checkers.

Kind of shows you how the field has progressed. But we started with checkers, we got to chess, we now do Go. But it played checkers. It beat national level players, most importantly, it learned to improve its methods by watching how it did in games and then inferring something to change what it thought about as it did that. Samuel did a bunch of other things. I just highlighted one. You may see in a follow on course, he invented what's called Alpha-Beta Pruning, which is a really useful technique for doing search.

But the idea is, how can we have the computer learn without being explicitly programmed? And one way to think about this is to think about the difference between how we would normally program and what we would like from a machine learning algorithm. Normal programming, I know you're not convinced there's such a thing as normal programming, but if you think of traditional programming, what's the process? I write a program that I input to the computer so that it can then take data and produce some appropriate output.

And the square root finder really sits there, right? I wrote code for using Newton method to find a square root, and then it gave me the process of given any number, I'll give you the square root. But if you think about what we did last time, it was a little different. And in fact, in a machine learning approach, the idea is that I'm going to give the computer output. I'm going to give it examples of what I want the program to do, labels on data, characterizations of different classes of things.

And what I want the computer to do is, given that characterization of output and data, I wanted that machine learning algorithm to actually produce for me a program, a program that I can then use to infer new information about things. And that creates, if you like, a really nice loop where I can have the machine learning algorithm learn the program which I can then use to solve some other problem. That would be really great if we could do it.

And as I suggested, that curve-fitting algorithm is a simple version of that. It learned a model for the data, which I could then use to label any other instances of the data or predict what I would see in terms of spring displacement as I changed the masses. So that's the kind of idea we're going to explore. If we want to learn things, we could also ask, so how do you learn? And how should a computer learn?

Well, for you as a human, there are a couple of possibilities. This is the boring one. This is the old style way of doing it, right? Memorize facts. Memorize as many facts as you can and hope that we ask you on the final exam instances of those facts, as opposed to some other facts you haven't memorized. This is, if you think way back to the first lecture, an example of declarative knowledge, statements of truth. Memorize as many as you can. Have Wikipedia in your back pocket.

Better way to learn is to be able to infer, to deduce new information from old. And if you think about this, this gets closer to what we called imperative knowledge-- ways to deduce new things. Now, in the first cases, we built that in when we wrote that program to do square roots. But what we'd like in a learning algorithm is to have much more like that generalization idea.

We're interested in extending our capabilities to write programs that can infer useful information from implicit patterns in the data. So not something explicitly built like that comparison of weights and displacements, but actually implicit patterns in the data, and have the algorithm figure out what those patterns are, and use those to generate a program you can use to infer new data about objects, about string displacements, whatever it is you're trying to do.

OK. So the idea then, the basic paradigm that we're going to see, is we're going to give the system some training data, some observations. We did that last time with just the spring displacements. We're going to then try and have a way to figure out, how do we write code, how do we write a program, a system that will infer something about the process that generated the data? And then from that, we want to be able to use that to make predictions about things we haven't seen before.

So again, I want to drive home this point. If you think about it, the spring example fit that model. I gave you a set of data, spatial deviations relative to mass displacements. For different masses, how far did the spring move? I then inferred something about the underlying process. In the first case, I said I know it's linear, but let me figure out what the actual linear equation is. What's the spring constant associated with it? And based on that result, I got a piece of code I could use to predict new displacements.

So it's got all of those elements, training data, an inference engine, and then the ability to use that to make new predictions. But that's a very simple kind of learning setting. So the more common one is one I'm going to use as an example, which is, when I give you a set of examples, those examples have some data associated with them, some features and some labels. For each example, I might say this is a particular kind of thing. This other one is another kind of thing. And what I want to do is figure out how to do inference on labeling new things.

So it's not just, what's the displacement of the mass, it's actually a label. And I'm going to use one of my favorite examples. I'm a big New England Patriots fan, if you're not, my apologies. But I'm going to use football players. So I'm going to show you in a second, I'm going to give you a set of examples of football players. The label is the position they play. And the data, well, it could be lots of things. We're going to use height and weight.

But what we want to do is then see how would we come up with a way of characterizing the implicit pattern of how does weight and height predict the kind of position this player could play. And then come up with an algorithm that will predict the position of new players. We'll do the draft for next year. Where do we want them to play? That's the paradigm. Set of observations, potentially labeled, potentially not. Think about how do we do inference to find a model. And then how do we use that model to make predictions.

What we're going to see, and we're going to see multiple examples today, is that that learning can be done in one of two very broad ways. The first one is called supervised learning. And in that case, for every new example I give you as part of the training data, I have a label on it. I know the kind of thing it is. And what I'm going to do is look for how do I find a rule that would predict the label associated with unseen input based on those examples. It's supervised because I know what the labeling is.

Second kind, if this is supervised, the obvious other one is called unsupervised. In that case, I'm just going to give you a bunch of examples. But I don't know the labels associated with them. I'm going to just try and find what are the natural ways to group those examples together into different models. And in some cases, I may know how many models are there. In some cases, I may want to just say what's the best grouping I can find.

OK. What I'm going to do today is not a lot of code. I was expecting cheers for that, John, but I didn't get them. Not a lot of code. What I'm going to do is show you basically, the intuitions behind doing this learning. And I"m going to start with my New England Patriots example. So here are some data points about current Patriots players. And I've got two kinds of positions. I've got receivers, and I have linemen. And each one is just labeled by the name, the height in inches, and the weight in pounds. OK? Five of each.

If I plot those on a two dimensional plot, this is what I get. OK? No big deal. What am I trying to do? I'm trying to learn, are their characteristics that distinguish the two classes from one another? And in the unlabeled case, all I have are just a set of examples. So what I want to do is decide what makes two players similar with the goal of seeing, can I separate this distribution into two or more natural groups.

Similar is a distance measure. It says how do I take two examples with values or features associated, and we're going to decide how far apart are they? And in the unlabeled case, the simple way to do it is to say, if I know that there are at least k groups there-- in this case, I'm going to tell you there are two different groups there-- how could I decide how best to cluster things together so that all the examples in one group are close to each other, all the examples in the other group are close to each other, and they're reasonably far apart.

There are many ways to do it. I'm going to show you one. It's a very standard way, and it works, basically, as follows. If all I know is that there are two groups there, I'm going to start by just picking two examples as my exemplars. Pick them at random. Actually at random is not great. I don't want to pick too closely to each other. I'm going to try and pick them far apart. But I pick two examples as my exemplars. And for all the other examples in the training data, I say which one is it closest to.

What I'm going to try and do is create clusters with the property that the distances between all of the examples of that cluster are small. The average distance is small. And see if I can find clusters that gets the average distance for both clusters as small as possible. This algorithm works by picking two examples, clustering all the other examples by simply saying put it in the group to which it's closest to that example.

Once I've got those clusters, I'm going to find the median element of that group. Not mean, but median, what's the one closest to the center? And treat those as exemplars and repeat the process. And I'll just do it either some number of times or until I don't get any change in the process. So it's clustering based on distance. And we'll come back to distance in a second.

So here's what would have my football players. If I just did this based on weight, there's the natural dividing line. And it kind of makes sense. All right? These three are obviously clustered, and again, it's just on this axis. They're all down here. These seven are at a different place. There's a natural dividing line there. If I were to do it based on height, not as clean. This is what my algorithm came up with as the best dividing line here, meaning that these four, again, just based on this axis are close together. These six are close together. But it's not nearly as clean.

And that's part of the issue we'll look at is how do I find the best clusters. If I use both height and weight, I get that, which was actually kind of nice, right? Those three cluster together. they're near each other, in terms of just distance in the plane. Those seven are near each other. There's a nice, natural dividing line through here. And in fact, that gives me a classifier.

This line is the equidistant line between the centers of those two clusters. Meaning, any point along this line is the same distance to the center of that group as it is to that group.

And so any new example, if it's above the line, I would say gets that label, if it's below the line, gets that label. In a second, we'll come back to look at how do we measure the distances, but the idea here is pretty simple. I want to find groupings near each other and far apart from the other group.

Now suppose I actually knew the labels on these players. These are the receivers. Those are the linemen. And for those of you who are football fans, you can figure it out, right? Those are the two tight ends. They are much bigger. I think that's Bennett and that's Gronk if you're really a big Patriots fan. But those are tight ends, those are wide receivers, and it's going to come back in a second, but there are the labels.

Now what I want to do is say, if I could take advantage of knowing the labels, how would I divide these groups up? And that's kind of easy to see. Basic idea, in this case, is if I've got labeled groups in that feature space, what I want to do is find a subsurface that naturally divides that space. Now subsurface is a fancy word. It says, in the two-dimensional case, I want to know what's the best line, if I can find a single line, that separates all the examples with one label from all the examples of the second label.

We'll see that, if the examples are well separated, this is easy to do, and it's great. But in some cases, it's going to be more complicated because some of the examples may be very close to one another. And that's going to raise a problem that you saw last lecture. I want to avoid overfitting. I don't want to create a really complicated surface to separate things. And so we may have to tolerate a few incorrectly labeled things, if we can't pull it out.

And as you already figured out, in this case, with the labeled data, there's the best fitting line right there. Anybody over 280 pounds is going to be a great lineman. Anybody under 280 pounds is more likely to be a receiver. OK. So I've got two different ways of trying to think about doing this labeling. I'm going to come back to both of them in a second.

Now suppose I add in some new data. I want to label new instances. Now these are actually players of a different position. These are running backs. But I say, all I know about is receivers and linemen. I get these two new data points. I'd like to know, are they more likely to be a receiver or a linemen? And there's the data for these two gentlemen. So if I go back to now plotting them, oh you notice one of the issues. So there are my linemen, the red ones are my receivers, the two black dots are the two running backs.

And notice right here. It's going to be really hard to separate those two examples from one another. They are so close to each other. And that's going to be one of the things we have to trade off. But if I think about using what I learned as a classifier with unlabeled data, there were my two clusters. Now you see, oh, I've got an interesting example. This new example I would say is clearly more like a receiver than a lineman. But that one there, unclear. Almost exactly lies along that dividing line between those two clusters.

And I would either say, I want to rethink the clustering or I want to say, you know what? As I know, maybe there aren't two clusters here. Maybe there are three. And I want to classify them a little differently. So I'll come back to that. On the other hand, if I had used the labeled data, there was my dividing line. This is really easy. Both of those new examples are clearly below the dividing line. They are clearly examples that I would categorize as being more like receivers than they are like linemen.

And I know it's a football example. If you don't like football, pick another example. But you get the sense of why I can use the data in a labeled case and the unlabeled case to come up with different ways of building the clusters. So what we're going to do over the next 2 and 1/2 lectures is look at how can we write code to learn that way of separating things out?

We're going to learn models based on unlabeled data. That's the case where I don't know what the labels are, by simply trying to find ways to cluster things together nearby, and then use the clusters to assign labels to new data. And we're going to learn models by looking at labeled data and seeing how do we best come up with a way of separating with a line or a plane or a collection of lines, examples from one group, from examples of the other group.

With the acknowledgment that we want to avoid overfitting, we don't want to create a really complicated system. And as a consequence, we're going to have to make some trade-offs between what we call false positives and false negatives. But the resulting classifier can then label any new data by just deciding where you are with respect to that separating line.

So here's what you're going to see over the next 2 and 1/2 lectures. Every machine learning method has five essential components. We need to decide what's the training data, and how are we going to evaluate the success of that system. We've already seen some examples of that. We need to decide how are we going to represent each instance that we're giving it.

I happened to choose height and weight for football players. But I might have been better off to pick average speed or, I don't know, arm length, something else. How do I figure out what are the right features. And associated with that, how do I measure distances between those features? How do I decide what's close and what's not close? Maybe it should be different, in terms of weight versus height, for example. I need to make that decision.

And those are the two things we're going to show you examples of today, how to go through that. Starting next week, Professor Guttag is going to show you how you take those and actually start building more detailed versions of measuring clustering, measuring similarities to find an objective function that you want to minimize to decide what is the best cluster to use. And then what is the best optimization method you want to use to learn that model.

So let's start talking about features. I've got a set of examples, labeled or not. I need to decide what is it about those examples that's useful to use when I want to decide what's close to another thing or not. And one of the problems is, if it was really easy, it would be really easy. Features don't always capture what you want. I'm going to belabor that football analogy, but why did I pick height and weight. Because it was easy to find.

You know, if you work for the New England Patriots, what is the thing that you really look for when you're asking, what's the right feature? It's probably some other combination of things. So you, as a designer, have to say what are the features I want to use. That quote, by the way, is from one of the great statisticians of the 20th century, which I think captures it well.

So feature engineering, as you, as a programmer, comes down to deciding both what are the features I want to measure in that vector that I'm going to put together, and how do I decide relative ways to weight it? So John, and Ana, and I could have made our job this term really easy if we had sat down at the beginning of the term and said, you know, we've taught this course many times. We've got data from, I don't know, John, thousands of students, probably over this time. Let's just build a little learning algorithm that takes a set of data and predicts your final grade.

You don't have to come to class, don't have to go through all the problems, because we'll just predict your final grade. Wouldn't that be nice? Make our job a little easier, and you may or may not like that idea. But I could think about predicting that grade? Now why am I telling this example. I was trying to see if I could get a few smiles. I saw a couple of them there.

But think about the features. What I measure? Actually, I'll put this on John because it's his idea. What would he measure? Well, GPA is probably not a bad predictor of performance. You do well in other classes, you're likely to do well in this class.

I'm going to use this one very carefully. Prior programming experience is at least a predictor, but it is not a perfect predictor. Those of you who haven't programmed before, in this class, you can still do really well in this class. But it's an indication that you've seen other programming languages.

On the other hand, I don't believe in astrology. So I don't think the month in which you're born, the astrological sign under which you were born has probably anything to do with how well you'd program. I doubt that eye color has anything to do with how well you'd program. You get the idea. Some features matter, others don't.

Now I could just throw all the features in and hope that the machine learning algorithm sorts out those it wants to keep from those it doesn't. But I remind you of that idea of overfitting. If I do that, there is the danger that it will find some correlation between birth month, eye color, and GPA.

And that's going to lead to a conclusion that we really don't like. By the way, in case you're worried, I can assure you that Stu Schmill in the dean of admissions department does not use machine learning to pick you. He actually looks at a whole bunch of things because it's not easy to replace him with a machine-- yet.

All right. So what this says is we need to think about how do we pick the features. And mostly, what we're trying to do is to maximize something called the signal to noise ratio. Maximize those features that carry the most information, and remove the ones that don't. So I want to show you an example of how you might think about this. I want to label reptiles. I want to come up with a way of labeling animals as, are they a reptile or not.

And I give you a single example. With a single example, you can't really do much. But from this example, I know that a cobra, it lays eggs, it has scales, it's poisonous, it's cold blooded, it has no legs, and it's a reptile. So I could say my model of a reptile is well, I'm not certain. I don't have enough data yet.

But if I give you a second example, and it also happens to be egg-laying, have scales, poisonous, cold blooded, no legs. There is my model, right? Perfectly reasonable model, whether I design it or a machine learning algorithm would do it says, if all of these are true, label it as a reptile. OK?

And now I give you a boa constrictor. Ah. It's a reptile. But it doesn't fit the model. And in particular, it's not egg-laying, and it's not poisonous. So I've got to refine the model. Or the algorithm has got to refine the model. And this, I want to remind you, is looking at the features. So I started out with five features. This doesn't fit.

So probably what I should do is reduce it. I'm going to look at scales. I'm going to look at cold blooded. I'm going to look at legs. That captures all three examples. Again, if you think about this in terms of clustering, all three of them would fit with that.

OK. Now I give you another example-- chicken. I don't think it's a reptile. In fact, I'm pretty sure it's not a reptile. And it nicely still fits this model, right? Because, while it has scales, which you may or not realize, it's not cold blooded, and it has legs. So it is a negative example that reinforces the model. Sounds good. And now I'll give you an alligator. It's a reptile. And oh fudge, right? It doesn't satisfy the model. Because while it does have scales and it is cold blooded, it has legs.

I'm almost done with the example. But you see the point. Again, I've got to think about how do I refine this. And I could by saying, all right. Let's make it a little more complicated-- has scales, cold blooded, 0 or four legs-- I'm going to say it's a reptile. I'll give you the dart frog. Not a reptile, it's an amphibian. And that's nice because it still satisfies this. So it's an example outside of the cluster that says no scales, not cold blooded, but happens to have four legs. It's not a reptile. That's good.

And then I give you-- I have to give you a python, right? I mean, there has to be a python in here. Oh come on. At least grown at me when I say that. There has to be a python here. And I give you that and a salmon. And now I am in trouble. Because look at scales, look at cold blooded, look at legs. I can't separate them. On those features, there's no way to come up with a way that will correctly say that the python is a reptile and the salmon is not. And so there's no easy way to add in that rule.

And probably my best thing is to simply go back to just two features, scales and cold blooded. And basically say, if something has scales and it's cold blooded, I'm going to call it a reptile. If it doesn't have both of those, I'm going to say it's not a reptile. It won't be perfect. It's going to incorrectly label the salmon. But I've made a design choice here that's important. And the design choice is that I will have no false negatives.

What that means is there's not going to be any instance of something that's not a reptile that I'm going to call a reptile. I may have some false positives. So I did that the wrong way. A false negative says, everything that's not a reptile I'm going to categorize that direction. I may have some false positives, in that, I may have a few things that I will incorrectly label as a reptile. And in particular, salmon is going to be an instance of that.

This trade off of false positives and false negatives is something that we worry about, as we think about it. Because there's no perfect way, in many cases, to separate out the data. And if you think back to my example of the New England Patriots, that running back and that wide receiver were so close together in height and weight, there was no way I'm going to be able to separate them apart. And I just have to be willing to decide how many false positives or false negatives do I want to tolerate.

Once I've figured out what features to use, which is good, then I have to decide about distance. How do I compare two feature vectors? I'm going to say vector because there could be multiple dimensions to it. How do I decide how to compare them? Because I want to use the distances to figure out either how to group things together or how to find a dividing line that separates things apart.

So one of the things I have to decide is which features. I also have to decide the distance. And finally, I may want to decide how to weigh relative importance of different dimensions in the feature vector. Some may be more valuable than others in making that decision. And I want to show you an example of that.

So let's go back to my animals. I started off with a feature vector that actually had five dimensions to it. It was egg-laying, cold blooded, has scales, I forget what the other one was, and number of legs. So one of the ways I could think about this is saying I've got four binary features and one integer feature associated with each animal. And one way to learn to separate out reptiles from non reptiles is to measure the distance between pairs of examples and use that distance to decide what's near each other and what's not.

And as we've said before, it will either be used to cluster things or to find a classifier surface that separates them. So here's a simple way to do it. For each of these examples, I'm going to just let true be 1, false be 0. So the first four are either 0s or 1s. And the last one is the number of legs. And now I could say, all right. How do I measure distances between animals or anything else, but these kinds of feature vectors?

[illegible]

Here, we're going to use something called the Minkowski Metric or the Minkowski difference. Given two vectors and a power, p, we basically take the absolute value of the difference between each of the components of the vector, raise it to the p-th power, take the sum, and take the p-th route of that. So let's do the two obvious examples. If p is equal to 1, I just measure the absolute distance between each component, add them up, and that's my distance. It's called the Manhattan metric.

The one you've seen more, the one we saw last time, if p is equal to 2, this is Euclidean distance, right? It's the sum of the squares of the differences of the components. Take the square root. Take the square root because it makes it have certain properties of a distance. That's the Euclidean distance. So now if I want to measure difference between these two, here's the question. Is this circle closer to the star or closer to the cross?

Unfortunately, I put the answer up here. But it differs, depending on the metric I use. Right? Euclidean distance, well, that's square root of 2 times 2, so it's about 2.8. And that's three. So in terms of just standard distance in the plane, we would say that these two are closer than those two are. Manhattan distance, why is it called that? Because you can only walk along the avenues and the streets. Manhattan distance would basically say this is one, two, three, four units away. This is one, two, three units away.

And under Manhattan distance, this is closer, this pairing is closer than that pairing is. Now you're used to thinking Euclidean. We're going to use that. But this is going to be important when we think about how are we comparing distances between these different pieces. So typically, we'll use Euclidean. We're going to see Manhattan actually has some value. So if I go back to my three examples-- boy, that's a gross slide, isn't it? But there we go-- rattlesnake, boa constrictor, and dart frog. There is the representation.

I can ask, what's the distance between them? In the handout for today, we've given you a little piece of code that would do that. And if I actually run through it, I get, actually, a nice little result. Here are the distances between those vectors using Euclidean metric. I'm going to come back to them. But you can see the two snakes, nicely, are reasonably close to each other. Whereas, the dart frog is a fair distance away from that. Nice, right? That's a nice separation that says there's a difference between these two.

OK. Now I throw in the alligator. Sounds like a Dungeons & Dragons game. I throw in the alligator, and I want to do the same comparison. And I don't get nearly as nice a result. Because now it says, as before, the two snakes are close to each other. But it says that the dart frog and the alligator are much closer, under this measurement, than either of them is to the other. And to remind you, right, the alligator and the two snakes I would like to be close to one another and a distance away from the frog. Because I'm trying to classify reptiles versus not.

So what happened here? Well, this is a place where the feature engineering is going to be important. Because in fact, the alligator differs from the frog in three features. And only in two features from, say, the boa constrictor. But one of those features is the number of legs. And there, while on the binary axes, the difference is between a 0 and 1, here it can be between 0 and 4. So that is weighing the distance a lot more than we would like. The legs dimension is too large, if you like.

How would I fix this? This is actually, I would argue, a natural place to use Manhattan distance. Why should I think that the difference in the number of legs or the number of legs difference is more important than whether it has scales or not? Why should I think that measuring that distance Euclidean-wise makes sense? They are really completely different measurements. And in fact, I'm not going to do it, but if I ran Manhattan metric on this, it would get the alligator much closer to the snakes, exactly because it differs only in two features, not three.

The other way I could fix it would be to say I'm letting too much weight be associated with the difference in the number of legs. So let's just make it a binary feature. Either it doesn't have legs or it does have legs. Run the same classification. And now you see the snakes and the alligator are all close to each other. Whereas the dart frog, not as far away as it was before, but there's a pretty natural separation, especially using that number between them.

What's my point? Choice of features matters. Throwing too many features in may, in fact, give us some overfitting. And in particular, deciding the weights that I want on those features has a real impact. And you, as a designer or a programmer, have a lot of influence in how you think about using those. So feature engineering really matters. How you pick the features, what you use is going to be important. OK.

The last piece of this then is we're going to look at some examples where we give you data, got features associated with them. We're going to, in some cases have them labeled, in other cases not. And we know how now to think about how do we measure distances between them.

John.

JOHN GUTTAG: You probably didn't intend to say weights of features. You intended to say how they're scaled.

ERIC GRIMSON: Sorry. The scales and not the-- thank you, John. No, I did. I take that back. I did not mean to say weights of features. I meant to say the scale of the dimension is going to be important here. Thank you, for the amplification and correction. You're absolutely right.

JOHN GUTTAG: Weights, we use in a different way, as we'll see next time.

ERIC GRIMSON: And we're going to see next time why we're going to use weights in different ways. So rephrase it. Block that out of your mind. We're going to talk about scales and the scale on the axes as being important here. And we already said we're going to look at two different kinds of learning, labeled and unlabeled, clustering and classifying. And I want to just finish up by showing you two examples of that. How we would think about them algorithmically, and we'll look at them in more detail next time.

As we look at it, I want to remind you the things that are going to be important to you. How do I measure distance between examples? What's the right way to design that? What is the right set of features to use in that vector? And then, what constraints do I want to put on the model? In the case of unlabelled data, how do I decide how many clusters I want to have? Because I can give you a really easy way to do clustering. If I give you 100 examples, I say build 100 clusters. Every example is its own cluster.

Distance is really good. It's really close to itself, but it does a lousy job of labeling things on it. So I have to think about, how do I decide how many clusters, what's the complexity of that separating service? How do I basically avoid the overfitting problem, which I don't want to have? So just to remind you, we've already seen a little version of this, the clustering method. This is a standard way to do it, simply repeating what we had on an earlier slide.

If I want to cluster it into groups, I start by saying how many clusters am I looking for? Pick an example I take as my early representation. For every other example in the training data, put it to the closest cluster. Once I've got those, find the median, repeat the process. And that led to that separation. Now once I've got it, I like to validate it. And in fact, I should have said this better. Those two clusters came without looking at the two black dots.

Once I put the black dots in, I'd like to validate, how well does this really work? And that example there is really not very encouraging. It's too close. So that's a natural place to say, OK, what if I did this with three clusters? That's what I get. I like the that. All right? That has a really nice cluster up here. The fact that the algorithm didn't know the labeling is irrelevant. There's a nice grouping of five. There's a nice grouping of four. And there's a nice grouping of three in between.

And in fact, if I looked at the average distance between examples in each of these clusters, it is much tighter than in that example. And so that leads to, then, the question of should I look for four clusters? Question, please.

AUDIENCE: Is that overlap between the two clusters not an issue?

ERIC GRIMSON: Yes. The question is, is the overlap between the two clusters a problem? No. I just drew it here so I could let you see where those pieces are. But in fact, if you like, the center is there. Those three points are all closer to that center than they are to that center. So the fact that they overlap is a good question. It's just the way I happened to draw them. I should really draw these, not as circles, but as some little bit more convoluted surface. OK? Having done three, I could say should I look for four?

Well, those points down there, as I've already said, are an example where it's going to be hard to separate them out. And I don't want to overfit. Because the only way to separate those out is going to be to come up with a really convoluted cluster, which I don't like. All right? Let me finish with showing you one other example from the other direction. Which is, suppose I give you labeled examples.

So again, the goal is I've got features associated with each example. They're going to have multiple dimensions on it. But I also know the label associated with them. And I want to learn what is the best way to come up with a rule that will let me take new examples and assign them to the right group. A number of ways to do this. You can simply say I'm looking for the simplest surface that will separate those examples. In my football case that were in the plane, what's the best line that separates them, which turns out to be easy.

I might look for a more complicated surface. And we're going to see an example in a second where maybe it's a sequence of line segments that separates them out. Because there's not just one line that does the separation. As before, I want to be careful. If I make it too complicated, I may get a really good separator, but I overfit to the data. And you're going to see next time. I'm going to just highlight it here. There's a third way, which will lead to almost the same kind of result called k nearest neighbors.

And the idea here is I've got a set of labeled data. And what I'm going to do is, for every new example, say find the k, say the five closest labeled examples. And take a vote. If 3 out of 5 or 4 out of 5 or 5 out of 5 of those labels are the same, I'm going to say it's part of that group. And if I have less than that, I'm going to leave it as unclassified. And that's a nice way of actually thinking about how to learn them.

And let me just finish by showing you an example. Now I won't use football players on this one. I'll use a different example. I'm going to give you some voting data. I think this is actually simulated data. But these are a set of voters in the United States with their preference. They tend to vote Republican. They tend to vote Democrat. And the two categories are their age and how far away they live from Boston. Whether those are relevant or not, I don't know, but they are just two things I'm going to use to classify them. And I'd like to say, how would I fit a curve to separate those two classes?

I'm going to keep half the data to test. I'm going to use half the data to train. So if this is my training data, I can say what's the best line that separates these? I don't know about best, but here are two examples. This solid line has the property that all the Democrats are on one side. Everything on the other side is a Republican, but there are some Republicans on this side of the line. I can't find a line that completely separates these, as I did with the football players. But there is a decent line to separate them.

Here's another candidate. That dash line has the property that on the right side you've got-- boy, I don't think this is deliberate, John, right-- but on the right side, you've got almost all Republicans. It seems perfectly appropriate. One Democrat, but there's a pretty good separation there. And on the left side, you've got a mix of things. But most of the Democrats are on the left side of that line. All right? The fact that left and right correlates with distance from Boston is completely irrelevant here. But it has a nice punch to it.

JOHN GUTTAG: Relevant, but not accidental.

ERIC GRIMSON: But not accidental. Thank you. All right. So now the question is, how would I evaluate these? How do I decide which one is better? And I'm simply going to show you, very quickly, some examples. First one is to look at what's called the confusion matrix. What does that mean? It says for this, one of these classifiers for example, the solid line. Here are the predictions, based on the solid line of whether they would be more likely to be Democrat or Republican.

And here is the actual label. Same thing for the dashed line.

And that diagonal is important because those are the correctly labeled results. Right? It correctly, in the solid line case, gets all of the correct labelings of the Democrats. It gets half of the Republicans right. But it has some where it's actually Republican, but it labels it as a Democrat. That, we'd like to be really large. And in fact, it leads to a natural measure called the accuracy. Which is, just to go back to that, we say that these are true positives. Meaning, I labeled it as being an instance, and it really is.

These are true negatives. I label it as not being an instance, and it really isn't. And then these are the false positives. I labeled it as being an instance and it's not, and these are the false negatives. I labeled it as not being an instance, and it is. And an easy way to measure it is to look at the correct labels over all of the labels. The true positives and the true negatives, the ones I got right. And in that case, both models come up with a value of 0.7.

So which one is better? Well, I should validate that. And I'm going to do that in a second by looking at other data. We could also ask, could we find something with less training error? This is only getting 70% right. Not great. Well, here is a more complicated model. And this is where you start getting worried about overfitting. Now what I've done, is I've come up with a sequence of lines that separate them. So everything above this line, I'm going to say is a Republican. Everything below this line, I'm going to say is a Democrat.

So I'm avoiding that one. I'm avoiding that one. I'm still capturing many of the same things. And in this case, I get 12 true positives, 13 true negatives, and only 5 false positives. And that's kind of nice. You can see the 5. It's those five red ones down there. It's accuracy is 0.833. And now, if I apply that to the test data, I get an OK result. It has an accuracy of about 0.6.

I could use this idea to try and generalize to say could I come up with a better model. And you're going to see that next time. There could be other ways in which I measure this. And I want to use this as the last example. Another good measure we use is called PPV, Positive Predictive Value which is how many true positives do I come up with out of all the things I labeled positively. And in this solid model, in the dashed line, I can get values about 0.57. The complex model on the training data is better. And then the testing data is even stronger.

And finally, two other examples are called sensitivity and specificity. Sensitivity basically tells you what percentage did I correctly find. And specificity said what percentage did I correctly reject. And I show you this because this is where the trade-off comes in. If sensitivity is how many did I correctly label out of those that I both correctly labeled and incorrectly labeled as being negative, how many them did I correctly label as being the kind that I want? I can make sensitivity 1.

Label everything is the thing I'm looking for. Great. Everything is correct. But the specificity will be 0. Because I'll have a bunch of things incorrectly labeled. I could make the specificity 1, reject everything. Say nothing as an instance. True negatives goes to 1, and I'm in a great place there, but my sensitivity goes to 0. I've got a trade-off. As I think about the machine learning algorithm I'm using and my choice of that classifier, I'm going to see a trade off where I can increase specificity at the cost of sensitivity or vice versa.

And you'll see a nice technique called ROC or Receiver Operator Curve that gives you a sense of how you want to deal with that. And with that, we'll see you next time. We'll take your question off line if you don't mind, because I've run over time. But we'll see you next time where Professor Guttag will show you examples of this.