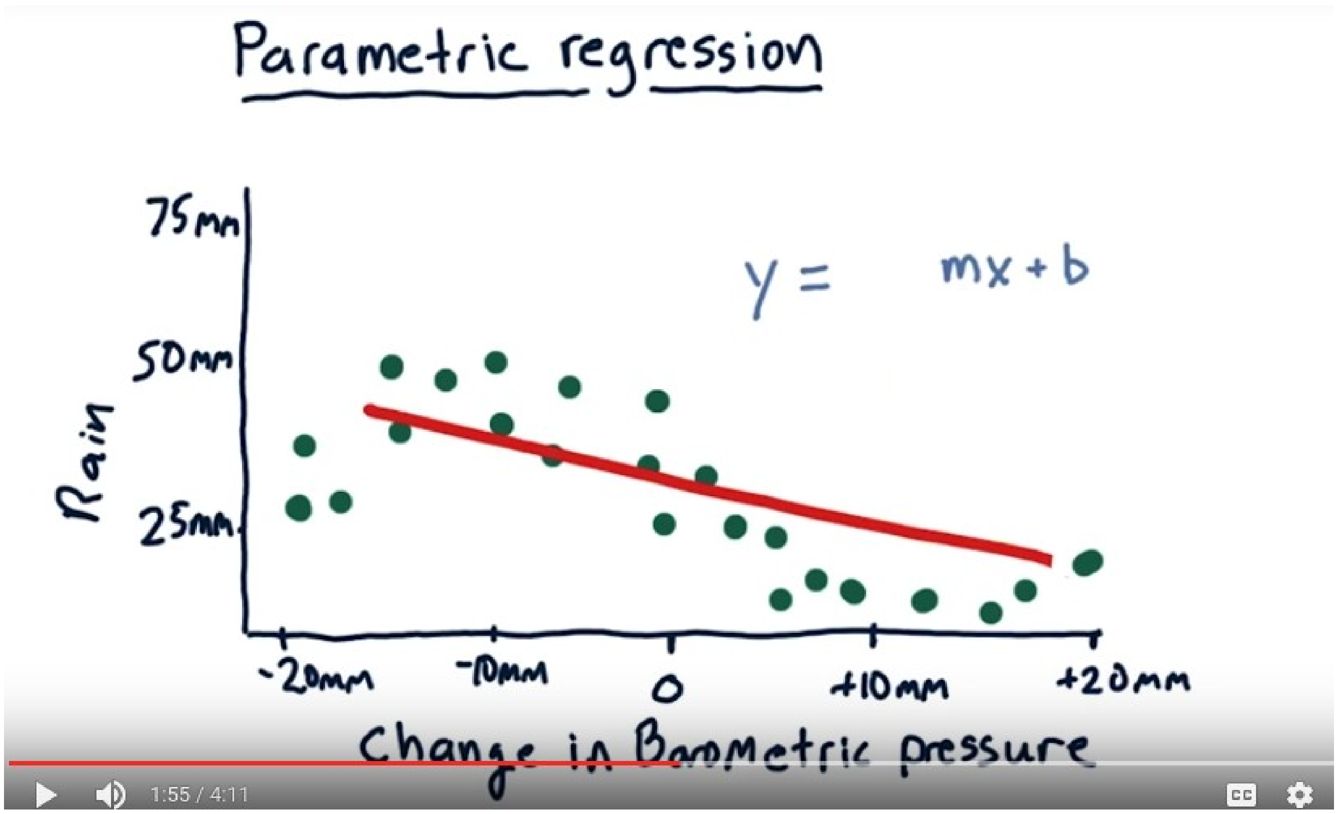
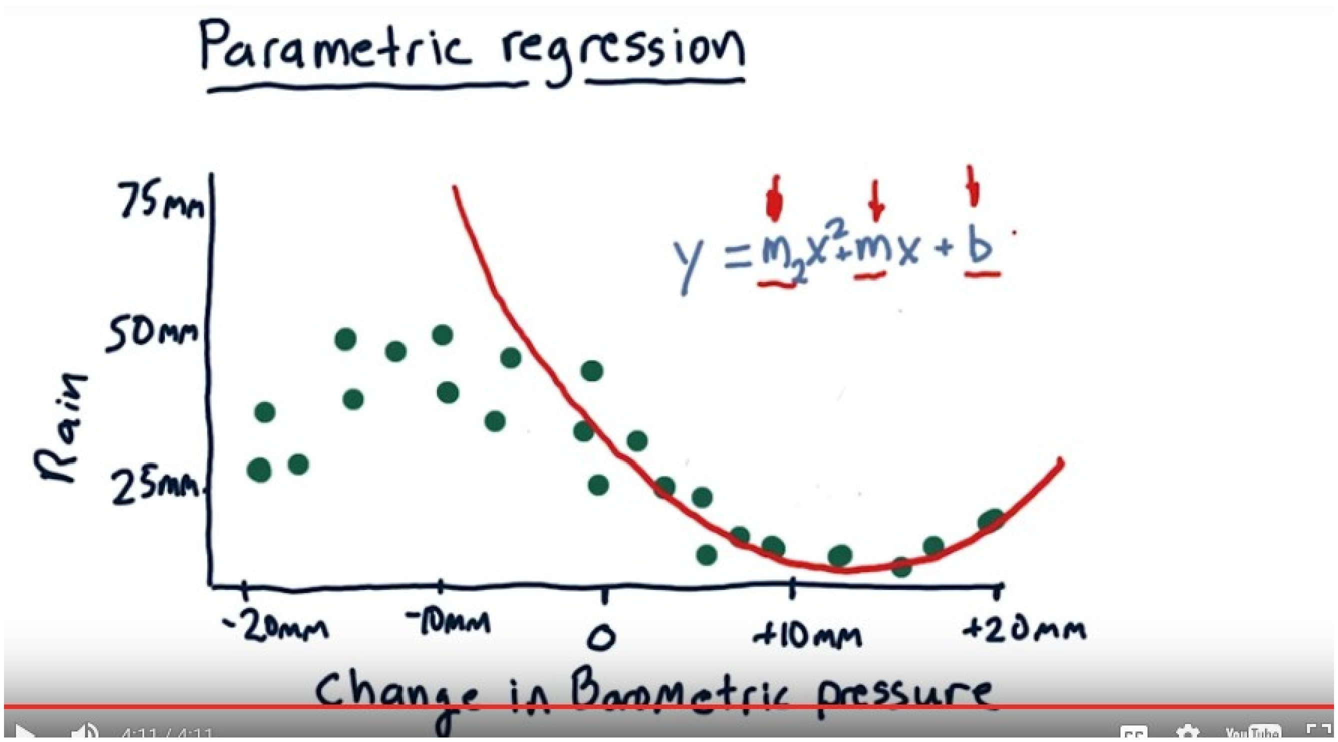
1. This lesson is about supervised regression learning. I've never much liked the name "regression", because it doesn't describe the activity very well. I prefer a name like numerical model. In any case, we're stuck with regression as the name for using data to build a model that predicts a numerical output based on a set of numerical inputs.

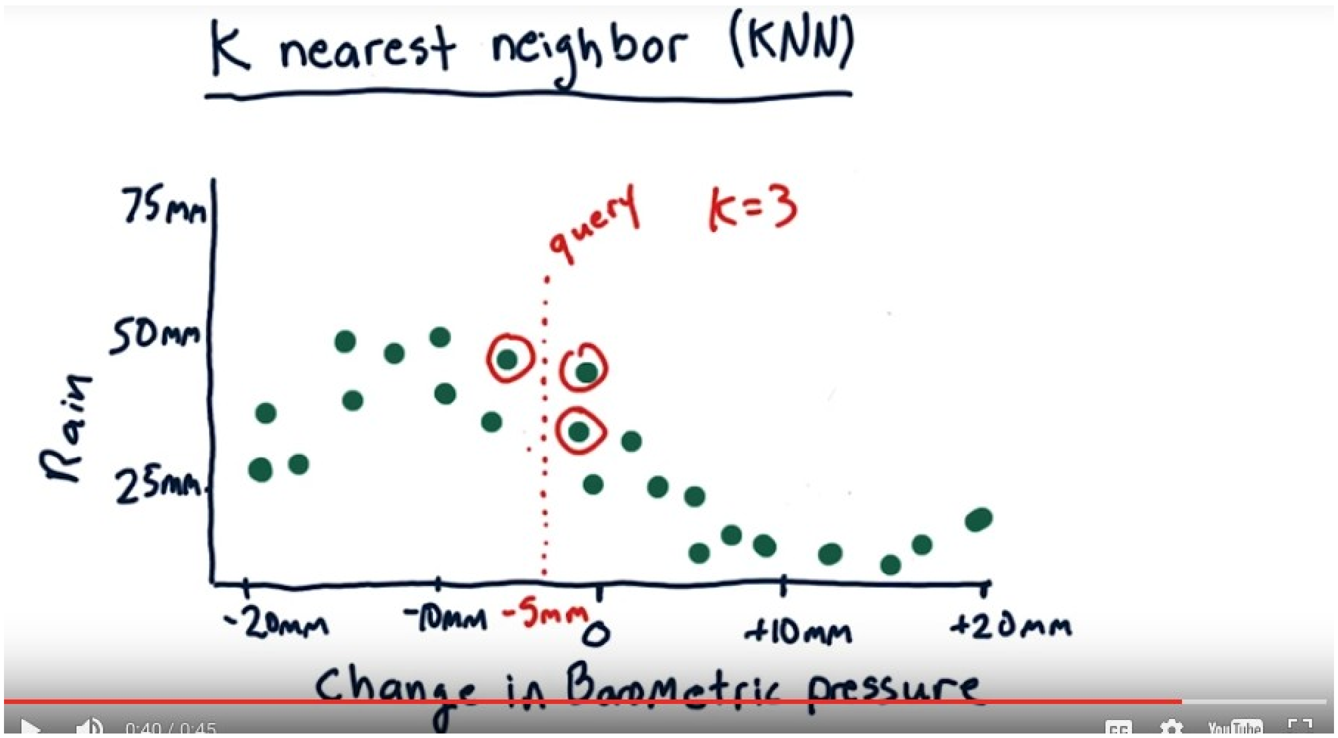


2. I'm going to start with parametric regression, which is a way of building a model where we represent the model with the number of parameters. Let's start with a simple example. Suppose we want to build a model that will predict how much it will rain today based on changes in barometric pressure. So as you may know, if barometric pressure declines, that usually means there's bad weather coming and it's going to rain. And when barometric pressure increases, it typically means that we've got good weather coming. So on this scatter plot each individual point represents one day. So let's consider a particular day, say here. And what this means is that on this day, the barometric pressure decreased by 10 millimeters and we had 50 millimeters of rain, about 2 inches. So let's consider that over time we collect data for many different days and that's what this looks like. So again, each one of these dots represents one day's worth of data and we have multiple day's worth of data here. And as you can see there's a general trend of as barometric pressure decreases, we typically have more rain and as it increases we have less rain. We'd like to create a model based on this data that when we query it at any particular point it'll give us a prediction of how much is going to rain, so we would measure barometric pressure or its change and then estimate how much is going to rain. The classic solution to this problem is to fit a line to the data. So let's give that a shot. As you probably know, this approach is called linear regression. And the model looks like this. So if you remember, from algebra in elementary school, or high school, wherever you got it. [LAUGH] The equation for line is simply y equals mx plus b. So x is our barometric pressure change variable here, and m and b are the parameters of our model. Our model now is fully described by these two parameters, and if we want to estimate or query how much it's going to rain at any particular point. We measure the barometric pressure. Let's say the barometric pressure today increased by 5. We would then plug that 5 into our model here and

multiply it by m and add b. And that's our estimate for that day of how much it's going to rain. And the linear regression approach is how we arrive at m and b. This model is decent, but it doesn't track the actual behavior of the data, for instance, in this region and in this region, so we can make a more complex model.

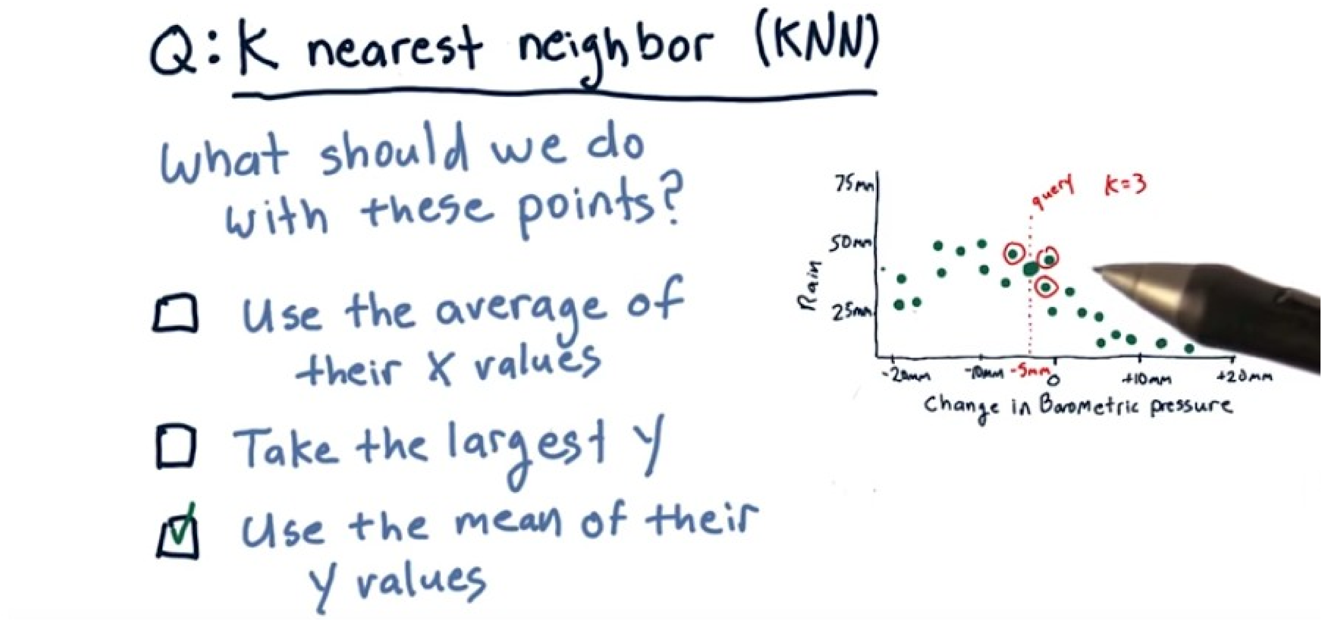


Instead of fitting a line we can fit a polynomial, we can add one more term x squared and now we've got to also find this additional parameter m2. So, when we find our model it's now represented by three parameters m, m2 and b. This polynomial model will look something like this. It fits the data pretty well over here, but not so well over here. Now, we can add more terms. We can fit an x cubed term and have another parameter and so on. In general, these parametric approaches come away with a number of parameters, and the more complex the model, the more parameters. Still, three parameters is a pretty simple model. All of these models, whether it's linear, with just mx plus b, or polynomial, with a cubed or squared term, are parametric models. In the end, after we learn these models, we have our parameters, in this case, m2, m, and b. And we throw away the data, and the model's represented just by these three parameters. But there's another way to approach it, and I'll show you that next.



3. There's another approach, it's a data-centric approach or instance based approach where we keep the data and we use it when we make a query. 這就是為何要叫 instance based approach. So here's an example, let's suppose our barometric pressure has gone down by 5 millimeters and we want to consult our model to see how much it's going to rain today. So our query here is at -5 millimeters. And let's suppose that K is equal to 3. So we will find the 3 nearest historical data points to this query. And those are 1, 2, 3. And we'll use them to estimate how much it's going to rain today

4. We've identified that we want to use these three historical data points in order to answer our query about how much rain we would expect if barometric pressure drops five millimeters. But what should we do with these data points to find that prediction? Here are three alternatives for you to consider. Take your time and check which one you think makes the most sense.

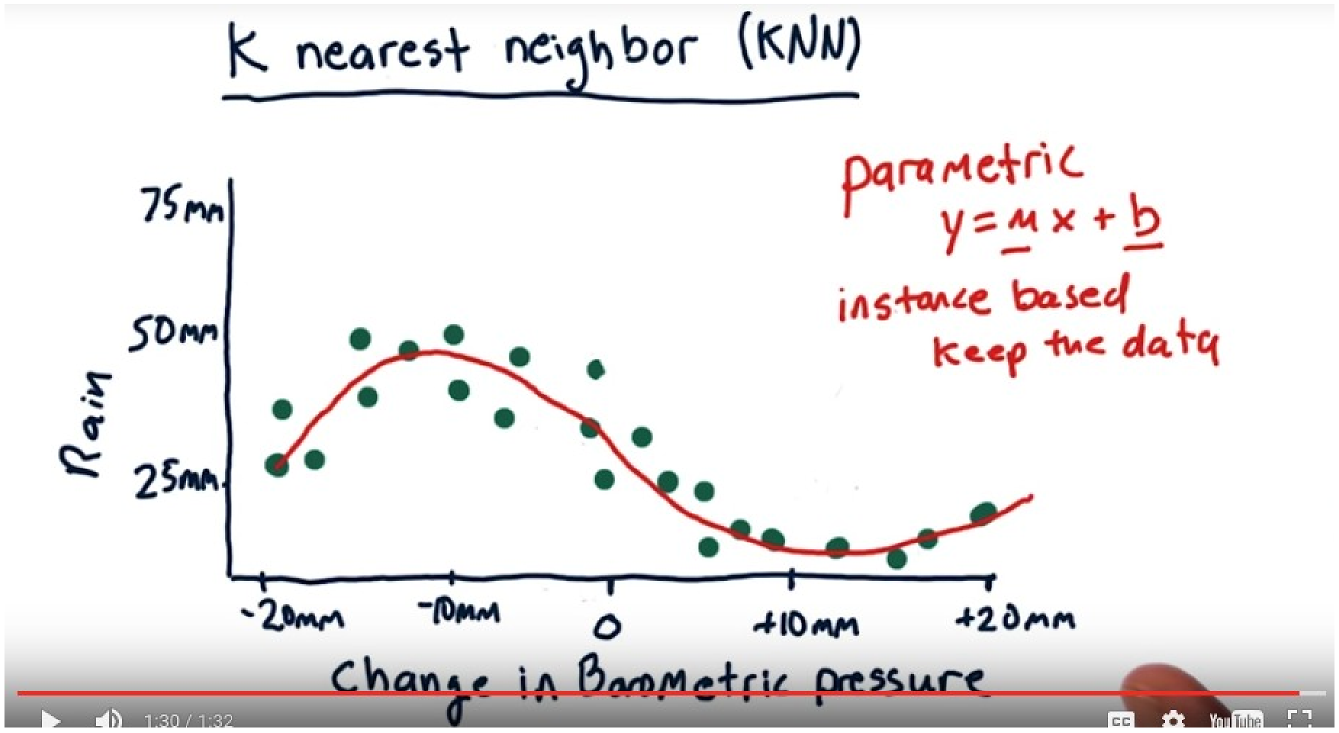


Q: K nearest neighbor (KNN)

What should we do with these points?

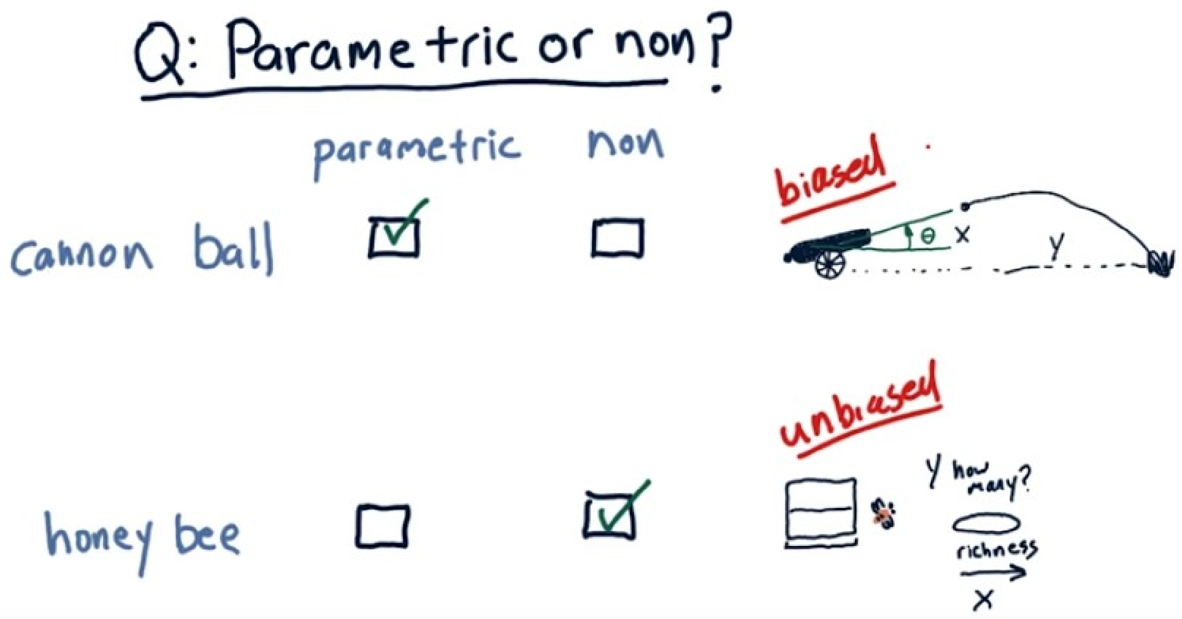
- ☐ Use the average of their X values
- ☐ Take the largest y
- [✓ Use the mean of their y values]

5. The correct answer is that we should take the mean of their y values. So that'll give us an answer somewhere around here. We don't want to take the average of their x values because that would be something like 5 millimeter, and in reality you know the correct answer is around 40 millimeters. We don't want to take the largest y, because we want to take advantage of all these votes that we have to get a closer, more correct answer.



6. So again, the correct response for this query is to take the mean of the K, where K is equal to 3, nearest neighbors. Take the mean of their Y-value, and that gives us a value somewhere about like there. If we were to repeat this process at many, many points along the X-axis, we end up with a model

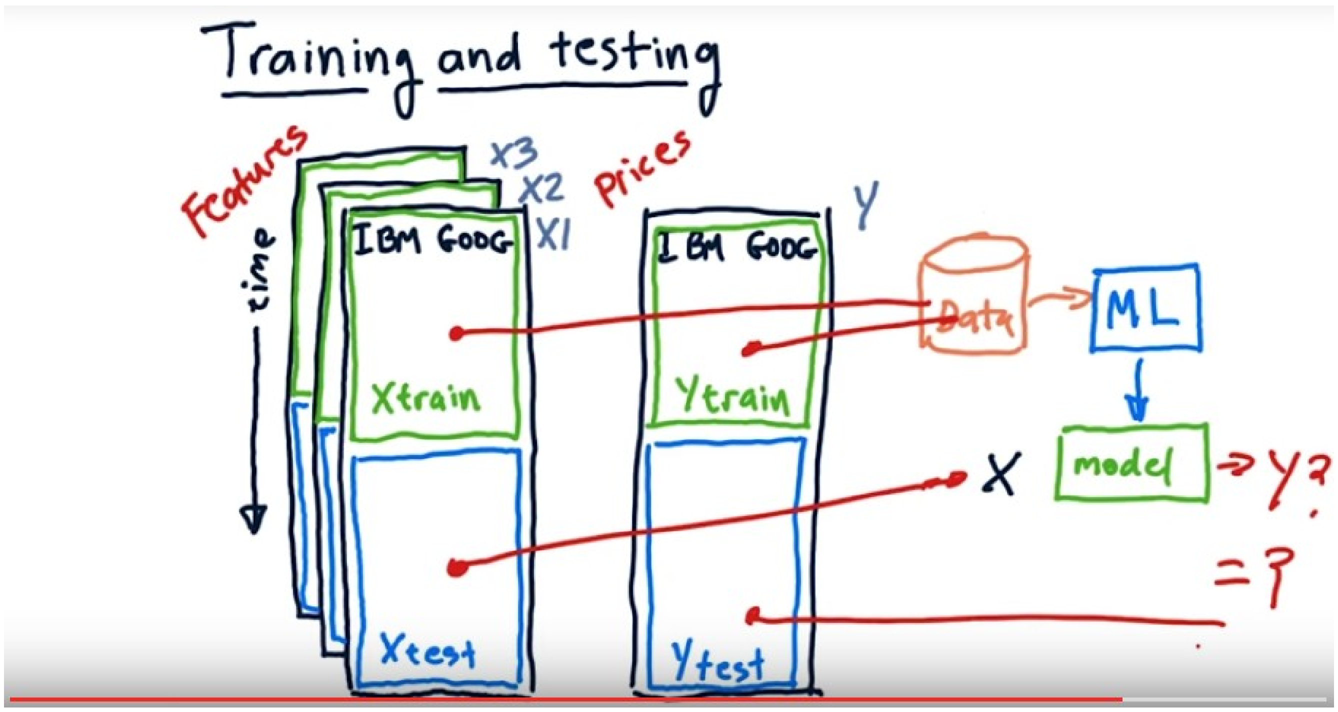
that looks like this. What's nice about it is it fits the data over here where it's curving this way. It fits the data over here where it's curving that way. And it interpolates nicely and smoothly between all the data points. There are number of methods like this that keep the data around and when they make a query, they consult the data to find the answer. The most famous of these methods is K nearest neighbor. But there are others, such as kernel regression. The main way that KNN differs from kernel regression is that in kernel regression we weight the contributions of each of the nearest data points according to how distant they are. Where as with KNN, each data point that we consider gets essentially an equal weight. We've covered now parametric models, where our goal is to find parameters like M and B. And we toss the data away, and then just use M and B later to make our queries and here I've shown you non-parametric or instance based methods where we keep the data and we consult it when we make a query.



7. So I'd like you to consider two data sets, and think about whether it would be best to use a parametric model or a non-parametric model. Our first problem is a cannon, and we orient the cannon up or down a certain number of degrees, and we want to estimate based on how many degrees up or down it is how far is the cannonball going to go. The cannon's oriented at a certain angle, boom, the cannonball comes out, and it follows some trajectory like this, and [SOUND] splats over here, so this angle theta is our x, and this distance, how far it went, is our y. And so we take the cannon, we orient it at many different thetas, and we measure how far the cannonball goes. And that's our data. Here's another problem from animal behavior. Let's suppose we have a beehive, and we want to estimate the behavior of the bees. In other words, how many of them will be attracted to this food source as richness of the food source increases? So in other words, our x is the richness of the food source, and our y is how many. So the reason this is a different sort of problem than, say, that cannonball problem, is it's not clear that as we increase the richness of the food, that necessarily more bees will come. Consider these two problems. The cannonball problem, we're trying to build a model for how far will the canon ball go depending on what is x, the angle at which the barrel is aimed. And the honeybee problem, how many honeybees will visit a food source as we change the richness of the food source from, say, poor to rich. Do you think this problem is best solved by a parametric model, or non-parametric model for each one? So go ahead and enter your guesses and I'll tell you what I think the answer is.

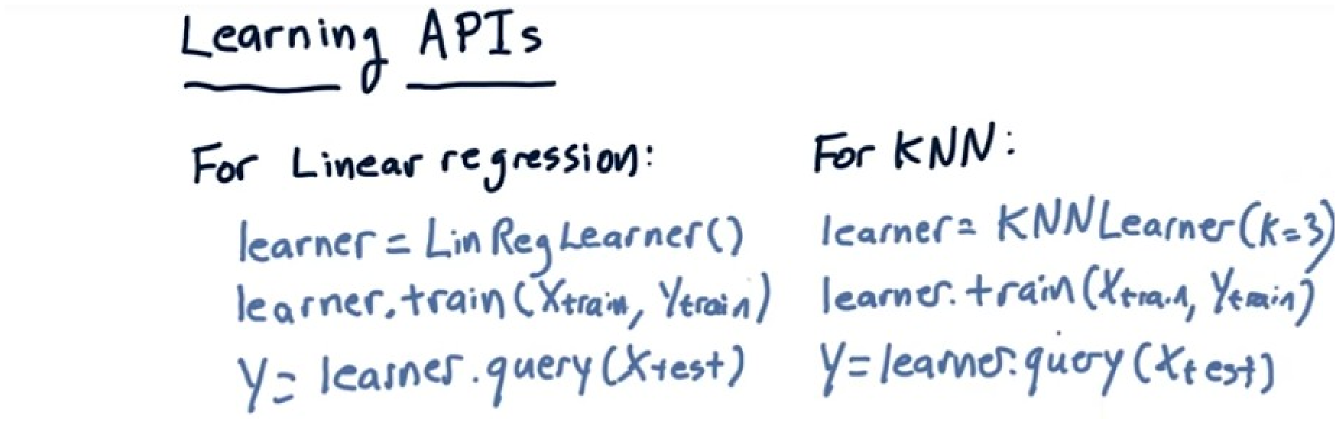
8. So in my view, this type of problem is best solved by a parametric learner, and this sort of problem is

best solved by a non-parametric learner. And here's why. For this problem, you can start with an equation, you can look it up on the net or whatever. That will express an estimate of how far a cannon ball will go given this angle. What's missing are some parameters of that equation that reflect the velocity of the cannon ball coming out and so on. But just by taking certain measurements of how far the ball goes at different angles, you can use a parametric learner to find those parameters. The key thing here is that you can start with an estimate of the underlying behavior of the system in terms of a mathematical equation that expresses how it behaves. Now, in the cases of the honey bees, we really don't know, we don't really have a guess of what that underlying mathematical equation might look like. And if you don't have a guess, it's better to use a non-parametric or instance-based model because it can fit any sort of shape. Another way to put that is that this model is biased, in the sense that we have an initial guess of what the form of the equation is. This solution is unbiased because we don't know. Now, we're raised to think biased is a bad thing, but if you go into the problem already knowing the form of the solution, it makes sense to take advantage of that bias, and aim your solution toward that bias. Let's reflect on the pros and cons of each approach. For a parametric approach, we don't have to store the original data, so it's very space efficient, but we can't easily update the model as more data is gathered. Usually we have to do a complete rerun of the learning algorithm to update the model, thus for parametric approaches, training is slow but querying is fast. For non-parametric approaches, or instance-based, we have to store all the data points. So it's hard to apply when we have a huge data set, but new evidence can be added easily since no perimeters need to be learned, adding data points doesn't consume additional time, thus training is fast, but querying is potentially slow. Most importantly, these nonparametric approaches avoid having to assume a certain type of model, whether it's linear or quadratic or so on. And therefore, they're suitable to fit complex patterns where we don't really know what the underlying model is like.



9. Consider now the data that we're going to use. We're going to have features that we've computed, these are things like Bollinger bands and momentum and price change and things like that. We're going to use these features to try and predict prices or price changes. So this is our X data, and if we've got multiple features, we've got multiple dimensions in X. So this might be X1, X2, X3, and so on. And this is our Y data, which we're trying to predict. In order to evaluate our learning algorithms in a

scientific manner we need to split this data into at least two sections. A training section and a testing section. If we trained over the same data that we tested over, the results would be suspicious because we should obviously be able to do very well if we test over the same data we trained on. This procedure of separating testing and training data from one another is called out of sample testing. This is a very important and essential technique. We'll call the X data that we use for training, Xtrain and the Y data that we use for training, Ytrain. Similarly, the data we'll test on will be split into X and Y sections, Xtest and Ytest. So the general idea here is that we'll take our Xtrain data and our Ytrain data, run that through our machine learning algorithm which might be linear regression or KNN to generate a model. We can then test the accuracy of that model using this data. So, the input to the model is Xtest, so we plug that X data into the model, and out comes something, some kind of Y. And the question is, is that Y equal to this Y which we know is ground truth. The more closely the model outputs a Y that reflects this Xtest data, the more accurate the model is. Something that I didn't mention, is that in this class, our data is time oriented. So, as you move downward, we're going forward in time. We typically split the data up according to time. We train our model on older data and test it on newer data. It's generally frowned upon to do the reverse. You might argue, well this data's different than that data. It's still out of sample. But there are certain look-ahead biases that can occur if you were to train on later data and test on earlier data.



Learning APIs

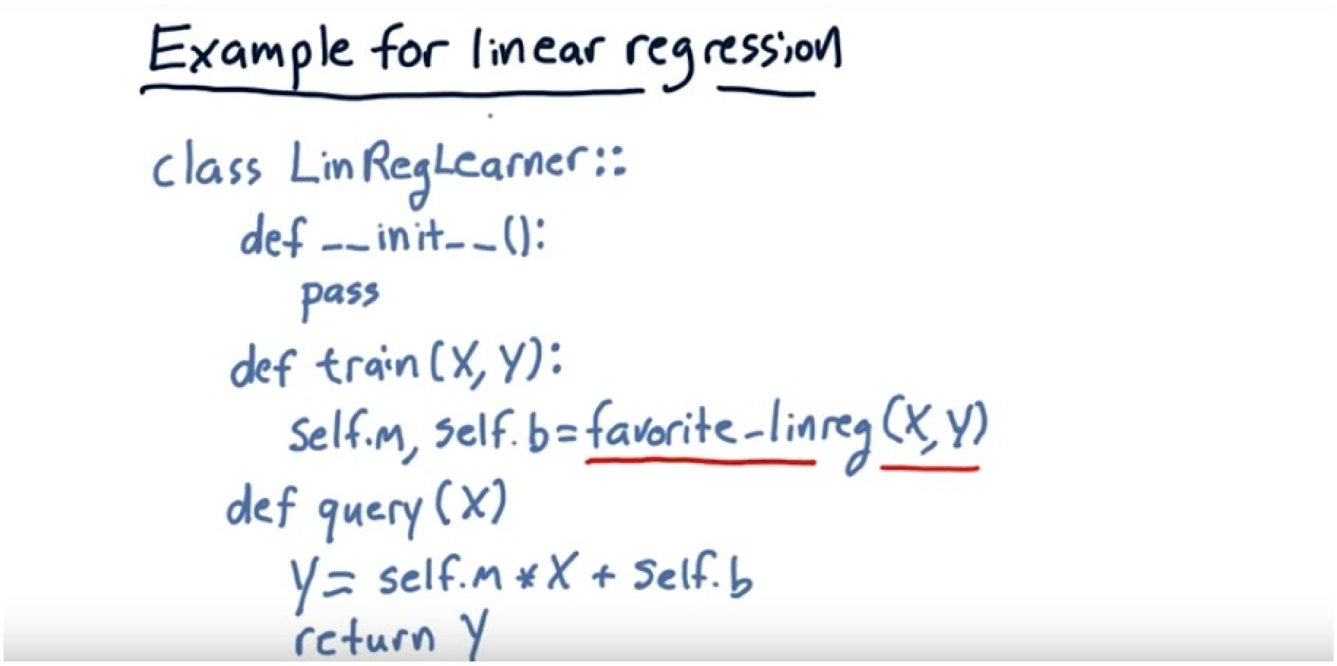
For Linear regression:

```
learner = LinRegLearner()
learner.train(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

For KNN:

```
learner = KNNLearner(k=3)
learner.train(Xtrain, Ytrain)
Y = learner.query(Xtest)
```

10. As part of this class you're going to have to write some software. In particular, you're going to have to write some machine learning algorithms. It's useful if we standardize on what the application programmer interface ought to look like (即自己寫的 API 應該是那樣的) for the code you're going to write. So here's what it looks like. For a linear regression learning algorithm, your API ought to implement something that works like this. A constructor that creates an instance of one of these learners, which is now learner. A method called train that can take our training data and train the model. And a query function that takes a list of X values that you want to test and returns a list of Y values according to what the model thinks they should be. These Y values, in turn, are the ones that we'll compare to Y test to see how well the algorithm works. Our KNNLearner ought to have the same methods. The only difference is the constructor here has this additional argument, K, which would allow you to set how large you want K to be. So if K is 3, that means you use the 3 nearest neighbors. But it also has a train and a query function, just like linear regression.


Example for linear regression

```
class LinRegLearner::
    def __init__():
        pass
    def train(X, Y):
        self.m, self.b = favorite-linreg(X, Y)
    def query(X)
        y = self.m * X + self.b
        return y
```

11. Now, I'm going to show you some pseudo code for how you might implement this API for a linear regression learner. So we define our class as LimRegLearner. And this first method init with underscores on either side is the constructor. The constructor is really easy. For the linear regression learner, we actually don't have to do anything when we instantiated an instance of the learner. So we just use pass, which means do nothing. Our training method takes an x and a y, and remember, x can be multidimensional. And what it's doing, it should take this x data and this y data and fit a line to it. So it's trying to find an m and a b. So the job of the train method is to find that linear equation or the parameters for that linear equation, m and b. So here on the left-hand side, we have self.m which means the m goes to the local instance, and b to the local instance, and you're allowed to use any number of linear regression algorithms at your disposal as part of SciPy and NumPy. So go Google and find which one you want to use here and just stuff it's output into m and b. Finally, query is passed in x and it's supposed to predicted y given that x. And remember, this is potentially a list of x's and it can also be multidimensional. Anyways, it's very simple. You just multiply that x times m and add b and return y, boom, that's it. In short, this is literally how short your LinRegLearner can be. Now, the k and n API is going to look exactly the same. And the reason for that is if we have the same API, we can easily mix and match, and try training and querying with the different algorithms and see how they compare. Okay, that is it for regression. I'll see you at the next lesson. Have a great day.