WEEK 9

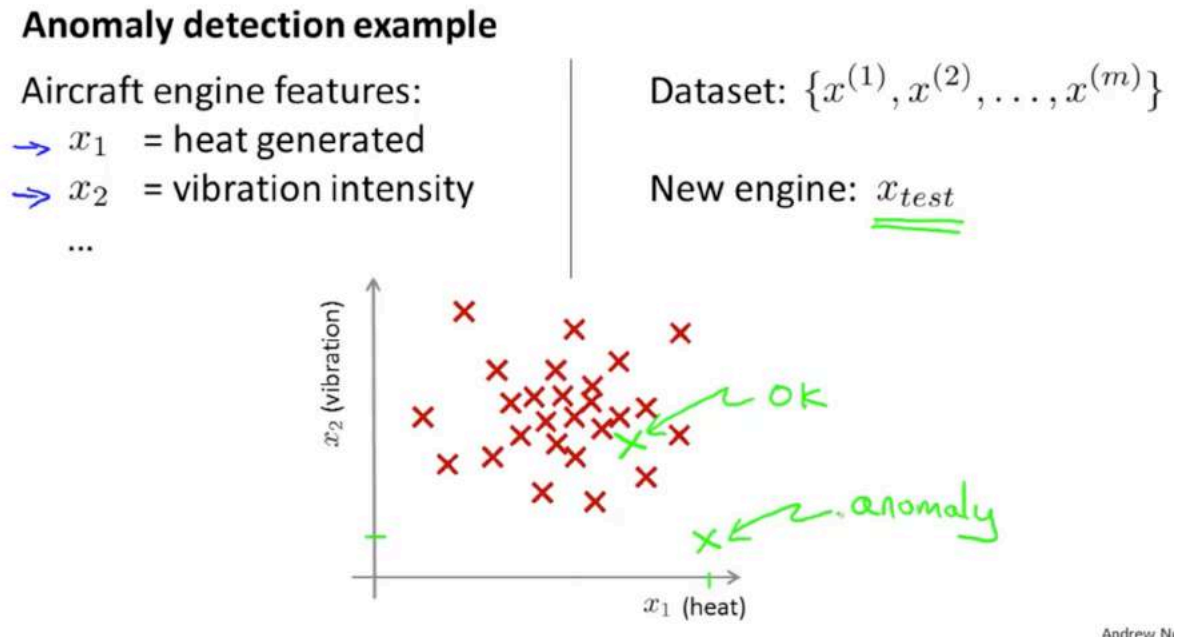
Anomaly Detection

Density Estimation

Problem Motivation

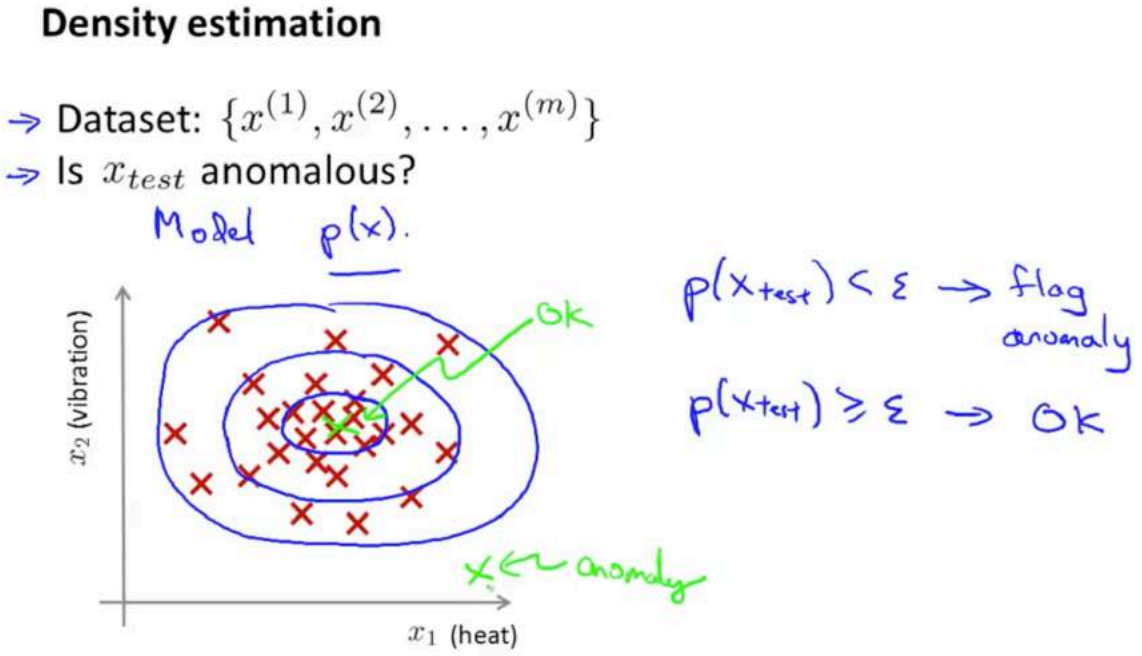
In this next set of videos, I'd like to tell you about a problem called Anomaly Detection. This is a reasonably commonly use you type machine learning. And one of the interesting aspects is that it's mainly for unsupervised problem, that there's some aspects of it that are also very similar to sort of the supervised learning problem. So, what is anomaly detection? To explain it. Let me use the motivating example of: Imagine that you're a manufacturer of aircraft engines, and let's say that as your aircraft engines roll off the assembly line, you're doing, you know, QA or quality assurance testing, and as part of that testing you measure features of your aircraft engine, like maybe, you measure the heat generated, things like the vibrations and so on. I share some friends that worked on this problem a long time ago, and these were actually the sorts of features that they were collecting off actual aircraft engines so you now have a data set of X1 through Xm, if you have manufactured m aircraft engines, and

if you plot your data, maybe it looks like this. So, each point here, each cross here as one of your unlabeled examples. So, the anomaly detection problem is the following. Let's say that on, you know, the next day, you have a new aircraft engine that rolls off the assembly line and your new aircraft engine has some set of features x-test. What the anomaly detection problem is, we want to know if this aircraft engine is anomalous in any way, in other words, we want to know if, maybe, this engine should undergo further testing because, or if it looks like an okay engine, and so it's okay to just ship it to a customer without further testing. So, if your new aircraft engine looks like a point over there, well, you know, that looks a lot like the aircraft engines we've seen before, and so maybe we'll say that it looks okay. Whereas, if your new aircraft engine, if x-test, you know, were a point that were out here, so that if X1 and X2 are the features of this new example. If x-tests were all the way out there, then we would call that an anomaly. and maybe send that aircraft engine for further testing before we ship it to a customer, since it looks very different than the rest of the aircraft engines we've seen before.



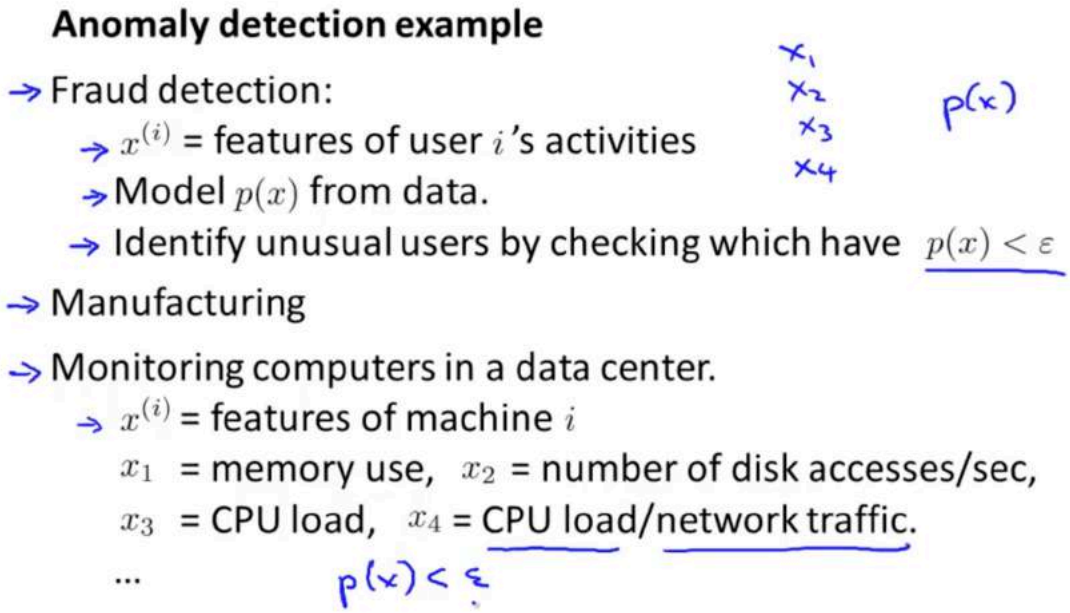
More formally in the anomaly detection problem, we're give some data sets, x1 through Xm of examples, and we usually assume that these end examples are normal or non-anomalous examples, and we want an algorithm to tell us if some new example x-test is anomalous. The approach that we're going to take is that given this training set, given the unlabeled training set, we're going to build a model for p of x. In other words, we're going to build a model for the probability of x, where x are these features of, say, aircraft engines. And so, having built a model of the probability of x we're then going to say that for the new aircraft engine, if p of x-test is less than some epsilon then we flag this as an anomaly. So we see a new engine that, you know, has very low probability under a model p of x that we estimate from the data, then we flag this anomaly, whereas if p of x-test is, say, greater than or equal to some small

threshold. Then we say that, you know, okay, it looks okay. And so, given the training set, like that plotted here, if you build a model, hopefully you will find that aircraft engines, or hopefully the model p of x will say that points that lie, you know, somewhere in the middle, that's pretty high probability, whereas points a little bit further out have lower probability. Points that are even further out have somewhat lower probability, and the point that's way out here, the point that's way out there, would be an anomaly. Whereas the point that's way in there, right in the middle, this would be okay because p of x right in the middle of that would be very high cause we've seen a lot of points in that region.



Here are some examples of applications of anomaly detection. Perhaps the most common application of anomaly detection is actually for detection if you have many users, and if each of your users take different activities, you know maybe on your website or in the physical plant or something, you can compute features of the different users activities. And what you can do is build a model to say, you know, what is the probability of different users behaving different ways. What is the probability of a particular vector of features of a users behavior so you know examples of features of a users activity may be on the website it'd be things like, maybe x1 is how often does this user log in, x2, you know, maybe the number of what pages visited, or the number of transactions, maybe x3 is, you know, the number of posts of the users on the forum, feature x4 could be what is the typing speed of the user and some websites can actually track that was the typing speed of this user in characters per second. And so you can model p of x based on this sort of data. And finally having your model p of x, you can try to identify users that are behaving very strangely on

your website by checking which ones have probably effects less than epsilon and maybe send the profiles of those users for further review. Or demand additional identification from those users, or some such to guard against you know, strange behavior or fraudulent behavior on your website. This sort of technique will tend of flag the users that are behaving unusually, not just users that maybe behaving fraudulently. So not just constantly having stolen or users that are trying to do funny things, or just find unusual users. But this is actually the technique that is used by many online websites that sell things to try identify users behaving strangely that might be indicative of either fraudulent behavior or of computer accounts that have been stolen. Another example of anomaly detection is manufacturing. So, already talked about the aircraft engine thing where you can find unusual, say, aircraft engines and send those for further review. A third application would be monitoring computers in a data center. I actually have some friends who work on this too. So if you have a lot of machines in a computer cluster or in a data center, we can do things like compute features at each machine. So maybe some features capturing you know, how much memory used, number of disc accesses, CPU load. As well as more complex features like what is the CPU load on this machine divided by the amount of network traffic on this machine? Then given the dataset of how your computers in your data center usually behave, you can model the probability of x, so you can model the probability of these machines having different amounts of memory use or probability of these machines having different numbers of disc accesses or different CPU loads and so on. And if you ever have a machine whose probability of x, p of x, is very small then you know that machine is behaving unusually and maybe that machine is about to go down, and you can flag that for review by a system administrator. And this is actually being used today by various data centers to watch out for unusual things happening on their machines.



Question

Your anomaly detection system flags x as anomalous whenever $p(x) \leq \epsilon$. Suppose your system is flagging too many things as anomalous that are not actually so (similar to supervised learning, these mistakes are called false positives). What should you do?

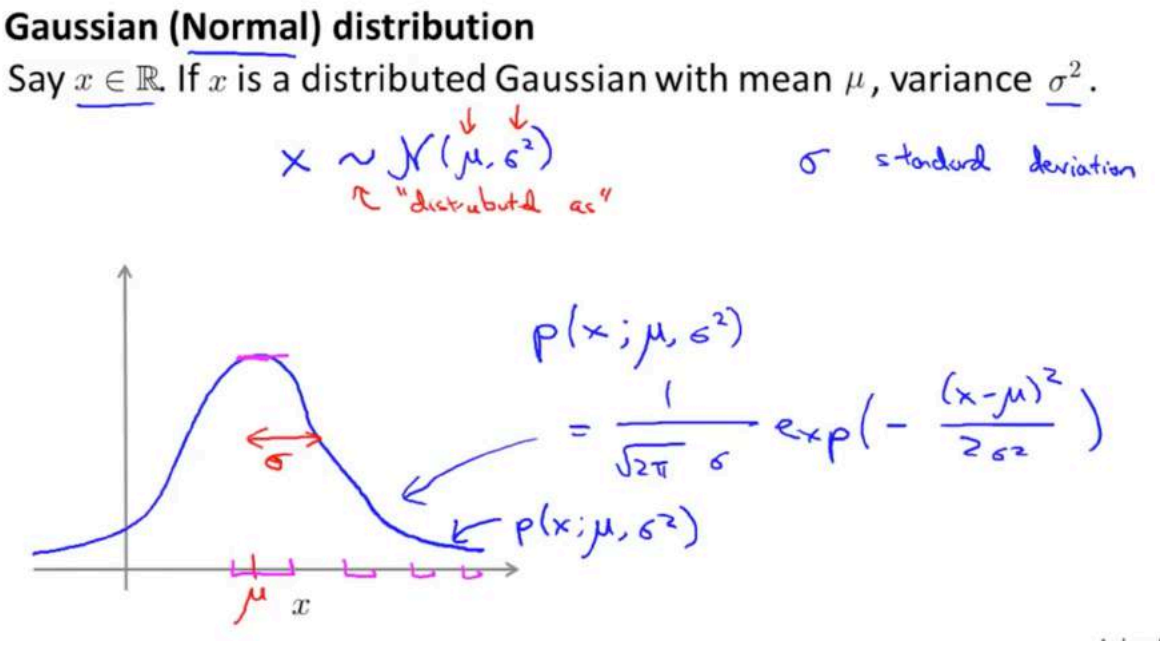
- ○ Try increasing ϵ.
- ● Try decreasing ϵ.

So, that's anomaly detection. In the next video, I'll talk a bit about the Gaussian distribution and review properties of the Gaussian probability distribution, and in videos after that, we will apply it to develop an anomaly detection algorithm.

Gaussian Distribution

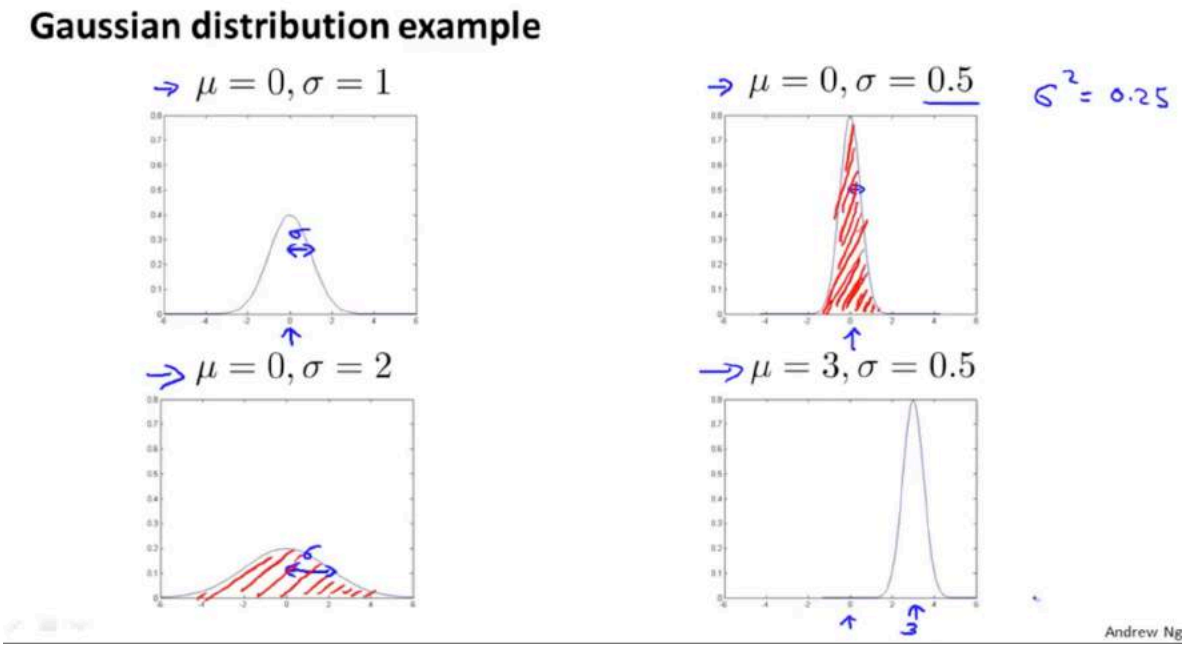
In this video, I'd like to talk about the Gaussian distribution which is also called the normal distribution. In case you're already intimately familiar with the Gaussian distribution, it's probably okay to skip this video, but if you're not sure or if it has been a while since you've worked with the Gaussian distribution or normal distribution then please do watch this video all the way to the end. And in the video after this we'll start applying the Gaussian distribution to developing an anomaly detection algorithm. Let's say x is a row value's random variable, so x is a row number. If the probability distribution of x is Gaussian with mean mu and variance sigma squared. Then, we'll write this as x, the random variable. Tilde, this little tilde, this is distributed as. And then to denote a Gaussian distribution, sometimes I'm going to write script N parentheses mu comma sigma script. So this script N stands for normal since Gaussian and normal they mean the thing are synonyms. And the Gaussian distribution is parametarized by two parameters, by a mean parameter which we denote mu and a variance parameter which we denote via sigma squared. If we plot the Gaussian distribution or Gaussian probability density. It'll look like the bell shaped curve which you may have seen before. And so this bell shaped curve is paramafied by those two parameters, mu and sequel. And the location of the center of this bell shaped curve is the mean mu. And the width of this bell shaped curve, roughly that, is this parameter, sigma, is also called one standard deviation, and so this specifies the probability of x taking on different values. So, x taking on values here in the middle here it's pretty high, since the Gaussian density here is pretty high, whereas x taking on values further, and

further away will be diminishing in probability. Finally just for completeness let me write out the formula for the Gaussian distribution. So the probability of x, and I'll sometimes write this as the p (x) when we write this as P (x ; mu, sigma squared), and so this denotes that the probability of X is parameterized by the two parameters mu and sigma squared. And the formula for the Gaussian density is this 1/ root 2 pi, sigma e (-(x-mu/g) squared/2 sigma squared. So there's no need to memorize this formula. This is just the formula for the bell-shaped curve over here on the left. There's no need to memorize it, and if you ever need to use this, you can always look this up. And so that figure on the left, that is what you get if you take a fixed value of mu and take a fixed value of sigma, and you plot P(x) so this curve here. This is really p(x) plotted as a function of X for a fixed value of Mu and of sigma squared. And by the way sometimes it's easier to think in terms of sigma squared that's called the variance. And sometimes is easier to think in terms of sigma. So sigma is called the standard deviation, and so it specifies the width of this Gaussian probability density, where as the square sigma, or sigma squared, is called the variance.



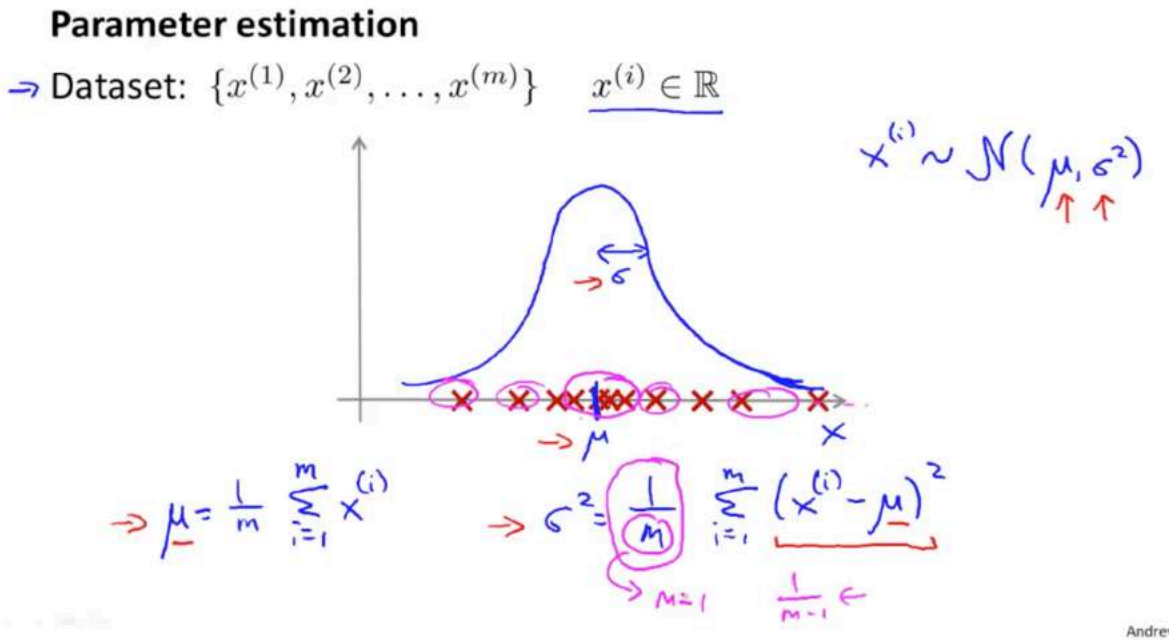
Let's look at some examples of what the Gaussian distribution looks like. If mu equals zero, sigma equals one. Then we have a Gaussian distribution that's centered around zero, because that's mu and the width of this Gaussian, so that's one standard deviation is sigma over there. Let's look at some examples of Gaussians. If mu is equal to zero and sigma equals one, then that corresponds to a Gaussian distribution that is centered at zero, since mu is zero, and the width of this Gaussian is is controlled by sigma by that variance parameter sigma. Here's another example. That same mu is equal to 0 and sigma is equal to .5 so the standard deviation is .5 and the variance sigma squared would therefore be the square of 0.5 would be 0.25 and in that case the Gaussian distribution, the Gaussian probability density goes like this. Is

also sent as zero. But now the width of this is much smaller because the smaller the area is, the width of this Gaussian density is roughly half as wide. But because this is a probability distribution, the area under the curve, that's the shaded area there. That area must integrate to one this is a property of probability distributing. So this is a much taller Gaussian density because this half is Y but half the standard deviation but it twice as tall. Another example is sigma is equal to 2 then you get a much fatter a much wider Gaussian density and so here the sigma parameter controls that Gaussian distribution has a wider width. And once again, the area under the curve, that is the shaded area, will always integrate to one, that's the property of probability distributions and because it's wider it's also half as tall in order to still integrate to the same thing. And finally one last example would be if we now change the mu parameters as well. Then instead of being centered at 0 we now have a Gaussian distribution that's centered at 3 because this shifts over the entire Gaussian distribution.



Next, let's talk about the Parameter estimation problem. So what's the parameter estimation problem? Let's say we have a dataset of m examples so exponents x m and lets say each of this example is a row number. Here in the figure I've plotted an example of the dataset so the horizontal axis is the x axis and either will have a range of examples of x, and I've just plotted them on this figure here. And the parameter estimation problem is, let's say I suspect that these examples came from a Gaussian distribution. So let's say I suspect that each of my examples, x i, was distributed. That's what this tilde thing means. Let's not suspect that each of these examples were distributed according to a normal distribution, or Gaussian distribution, with some parameter mu and some parameter sigma square. But I don't know what the values of these

parameters are. The problem of parameter estimation is, given my data set, I want to try to figure out, well I want to estimate what are the values of mu and sigma squared. So if you're given a data set like this, it looks like maybe if I estimate what Gaussian distribution the data came from, maybe that might be roughly the Gaussian distribution it came from. With mu being the center of the distribution, sigma standing for the deviation controlling the width of this Gaussian distribution. Seems like a reasonable fit to the data. Because, you know, looks like the data has a very high probability of being in the central region, and a low probability of being further out, even though probability of being further out, and so on. So maybe this is a reasonable estimate of mu and sigma squared. That is, if it corresponds to a Gaussian distribution function that looks like this. So what I'm going to do is just write out the formula the standard formulas for estimating the parameters Mu and sigma squared. Our estimate or the way we're going to estimate mu is going to be just the average of my example. So mu is the mean parameter. Just take my training set, take my m examples and average them. And that just means the center of this distribution. How about sigma squared? Well, the variance, I'll just write out the standard formula again, I'm going to estimate as sum over one through m of x i minus mu squared. And so this mu here is actually the mu that I compute over here using this formula. And what the variance is, or one interpretation of the variance is that if you look at this term, that's the square difference between the value I got in my example minus the mean. Minus the center, minus the mean of the distribution. And so in the variance I'm gonna estimate as just the average of the square differences between my examples, minus the mean. And as a side comment, only for those of you that are experts in statistics. If you're an expert in statistics, and if you've heard of maximum likelihood estimation, then these parameters, these estimates, are actually the maximum likelihood estimates of the primes of mu and sigma squared but if you haven't heard of that before don't worry about it, all you need to know is that these are the two standard formulas for how to figure out what are mu and Sigma squared given the data set. Finally one last side comment again only for those of you that have maybe taken the statistics class before but if you've taken statistics This class before. Some of you may have seen the formula here where this is M-1 instead of M so this first term becomes 1/M-1 instead of 1/M. In machine learning people tend to learn 1/M formula but in practice whether it is 1/M or 1/M-1 it makes essentially no difference assuming M is reasonably large. a reasonably large training set size. So just in case you've seen this other version before. In either version it works just about equally well but in machine learning most people tend to use 1/M in this formula.And the two versions have slightly different theoretical properties like these are different math properties. Bit of practice it really makes makes very little difference, if any.

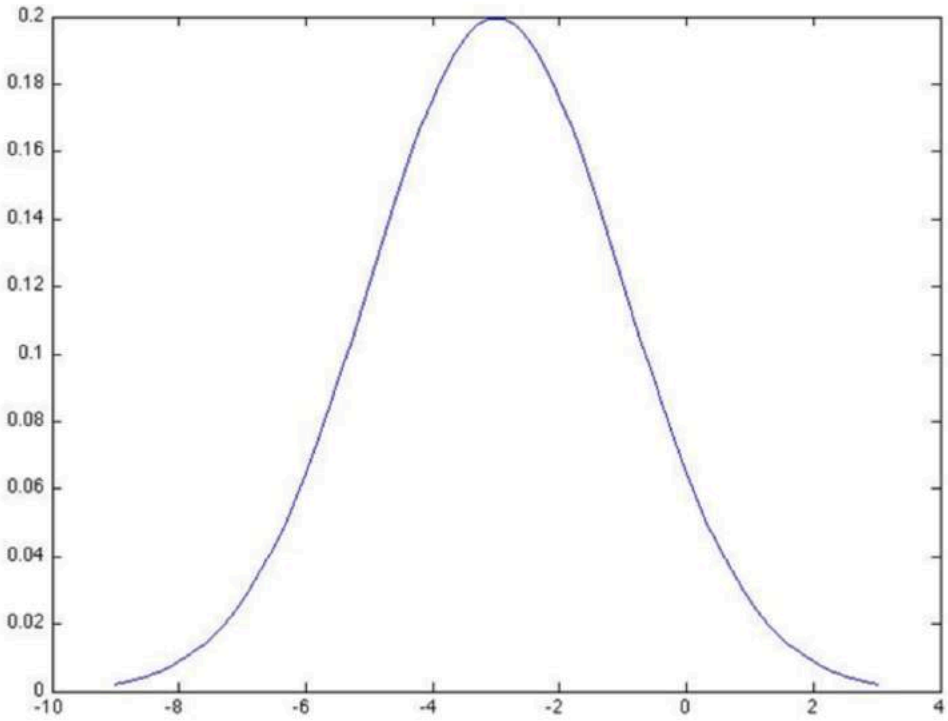

Parameter estimation
⇒ Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $x^{(i)} \in \mathbb{R}$
$x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$
σ
μ
x
$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$
$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$
$m-1$
$\frac{1}{m-1}$

Question

The formula for the Gaussian density is:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Which of the following is the formula for the density to the right?



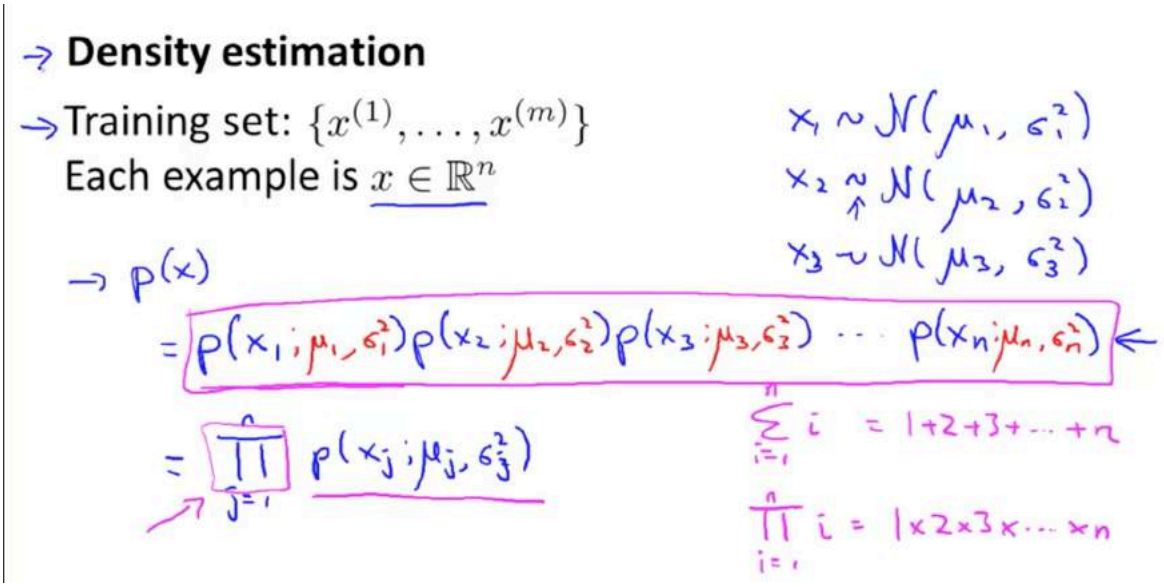
- $p(x) = \frac{1}{\sqrt{2\pi \times 2}} \exp\left(-\frac{(x-3)^2}{2\times 4}\right)$
- $p(x) = \frac{1}{\sqrt{2\pi \times 4}} \exp\left(-\frac{(x-3)^2}{2\times 2}\right)$
- **(selected)** $p(x) = \frac{1}{\sqrt{2\pi \times 2}} \exp\left(-\frac{(x+3)^2}{2\times 4}\right)$
- $p(x) = \frac{1}{\sqrt{2\pi \times 4}} \exp\left(-\frac{(x+3)^2}{2\times 2}\right)$

So, hopefully you now have a good sense of what the Gaussian distribution looks like, as well as how to estimate the parameters mu and sigma squared of Gaussian distribution if you're given a training set, that is if you're given a set of data that you suspect comes from a Gaussian distribution with unknown parameters, mu and sigma squared. In the next video, we'll start to take this and apply it to develop an anomaly detection algorithm.

Algorithm

In the last video, we talked about the Gaussian distribution. In this video lets apply that to develop an anomaly detection algorithm. Let's say that we have an unlabeled training set of M examples, and each of these examples is going to be a feature in Rn so your training set could be, feature vectors from the last M aircraft engines being manufactured. Or it could be features from m users or something else. The way we are going to address anomaly detection, is we are going to model p of x from the data sets. We're going to try to figure out what are high probability features, what are lower probability types of features. So, x is a vector and what we are going to do is model p of x, as probability of x1, that is of the first component of x, times the probability of x2, that is the probability of the second feature, times the probability of the third feature, and so on up to the probability of the final feature of Xn. Now I'm leaving space here cause I'll fill in something in a minute. So, how do we model each of these terms, p of X1, p of X2, and so on. What we're going to do, is assume that the feature, X1, is distributed according to a Gaussian distribution, with some mean, which you want to write as mu1 and some variance, which I'm going to write as sigma squared 1, and so p of X1 is going to be a Gaussian probability distribution, with mean mu1 and variance sigma squared 1. And similarly I'm going to assume that X2 is distributed, Gaussian, that's what this little tilda stands for, that means distributed Gaussian with mean mu2 and Sigma squared 2, so it's distributed according to a different Gaussian, which has a different set of parameters, mu2 sigma square 2. And similarly, you know, X3 is yet another Gaussian, so this can have a different mean and a different standard deviation than the other features, and so on, up to XN. And so that's my model. Just as a side comment for those of you that are experts in statistics, it turns out that this equation that I just wrote out actually corresponds to an independence assumption on the values of the features x1 through xn. But in practice it turns out that the algorithm of this fragment, it works just fine, whether or not these features are anywhere close to independent and even if independence assumption doesn't hold true this algorithm works just fine. But in case you don't know those terms I just used independence assumptions and so on, don't worry about it. You'll be able to understand it and implement this algorithm just fine and that comment was really meant only for the experts in statistics. Finally, in order to wrap this up, let me take this expression and write it a little bit more compactly. So, we're going to write this is a product from J equals one through N, of P of XJ parameterized by mu j comma sigma squared j. So this funny symbol here, there is capital Greek alphabet pi, that funny symbol there corresponds to taking the product of a set of values. And so, you're familiar with the summation notation, so the sum from i equals one through n, of i. This means 1 + 2 + 3 plus dot dot dot, up to n. Where as this funny symbol here, this product

symbol, right product from i equals 1 through n of i. Then this means that, it's just like summation except that we're now multiplying. This becomes 1 times 2 times 3 times up to N. And so using this product notation, this product from j equals 1 through n of this expression. It's just more compact, it's just shorter way for writing out this product of of all of these terms up there. Since we're are taking these p of x j given mu j comma sigma squared j terms and multiplying them together. And, by the way the problem of estimating this distribution p of x, they're sometimes called the problem of density estimation. Hence the title of the slide.

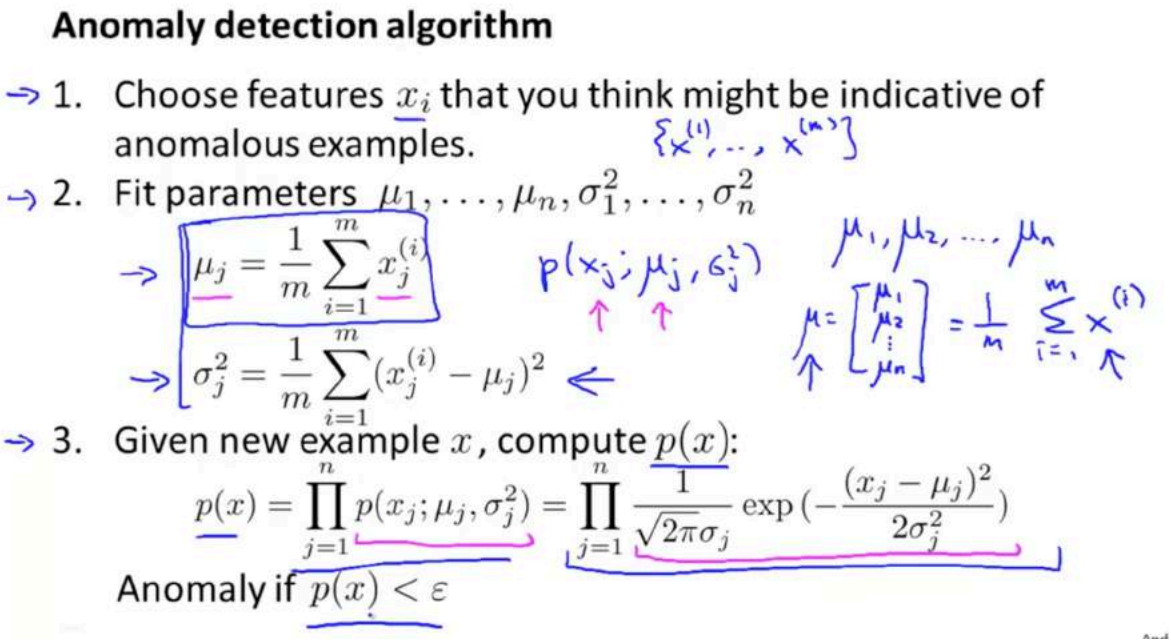


Question

Given a training set $\{x^{(1)}, \ldots, x^{(m)}\}$, how would you estimate each μ_j and σ_j^2 (Note $\mu_j \in \mathbb{R}, \sigma_j^2 \in \mathbb{R}$.)

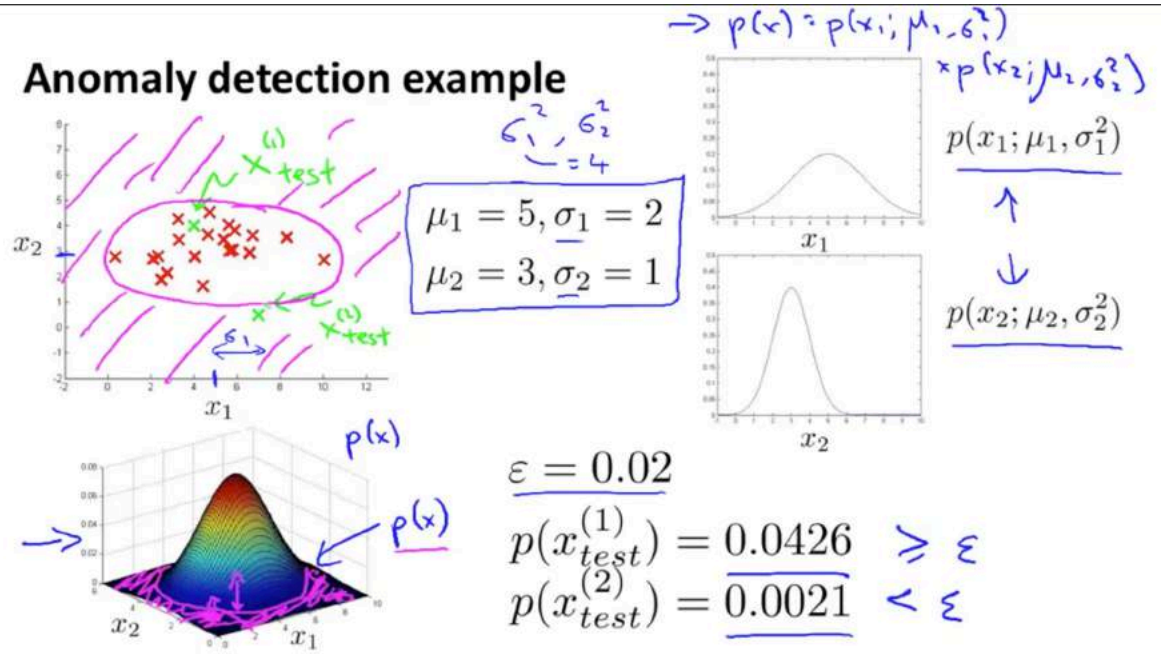
- $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x^{(i)},\ \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$
- $\mu_j = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)})^2,\ \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$
- $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)},\ \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$
- $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)},\ \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$

So putting everything together, here is our anomaly detection algorithm. The first step is to choose features, or come up with features xi that we think might be indicative of anomalous examples. So what I mean by that, is, try to come up with features, so that when there's an unusual user in your system that may be doing fraudulent things, or when the aircraft engine examples, you know there's something funny, something strange about one of the aircraft engines. Choose features X I, that you think might take on unusually large values, or unusually small values, for what an anomalous example might look like. But more generally, just try to choose features that describe general properties of the things that you're collecting data on. Next, given a training set, of M, unlabled examples, X1 through X M, we then fit the parameters, mu 1 through mu n, and sigma squared 1 through sigma squared n, and so these were the formulas similar to the formulas we have in the previous video, that we're going to use the estimate each of these parameters, and just to give some interpretation, mu J, that's my average value of the j feature. Mu j goes in this term p of xj. which is parametrized by mu J and sigma squared J. And so this says for the mu J just take the mean over my training set of the values of the j feature. And, just to mention, that you do this, you compute these formulas for j equals one through n. So use these formulas to estimate mu 1, to estimate mu 2, and so on up to mu n, and similarly for sigma squared, and it's also possible to come up with vectorized versions of these. So if you think of mu as a vector, so mu if is a vector there's mu 1, mu 2, down to mu n, then a vectorized version of that set of parameters can be written like so sum from 1 equals one through n xi. So, this formula that I just wrote out estimates this xi as the feature vectors that estimates mu for all the values of n simultaneously. And it's also possible to come up with a vectorized formula for estimating sigma squared j. Finally, when you're given a new example, so when you have a new aircraft engine and you want to know is this aircraft engine anomalous. What we need to do is then compute p of x, what's the probability of this new example? So, p of x is equal to this product, and what you implement, what you compute, is this formula and where over here, this thing here this is just the formula for the Gaussian probability, so you compute this thing, and finally if this probability is very small, then you flag this thing as an anomaly.



Here's an example of an application of this method. Let's say we have this data set plotted on the upper left of this slide. if you look at this, well, lets look the feature of x1. If you look at this data set, it looks like on average, the features x1 has a mean of about 5 and the standard deviation, if you only look at just the x1 values of this data set has the standard deviation of maybe 2. So that sigma 1 and looks like x2 the values of the features as measured on the vertical axis, looks like it has an average value of about 3, and a standard deviation of about 1. So if you take this data set and if you estimate mu1, mu2, sigma1, sigma2, this is what you get. And again, I'm writing sigma here, I'm think about standard deviations, but the formula on the previous 5 actually gave the estimates of the squares of theses things, so sigma squared 1 and sigma squared 2. So, just be careful whether you are using sigma 1, sigma 2, or sigma squared 1 or sigma squared 2. So, sigma squared 1 of course would be equal to 4, for example, as the square of 2. And in pictures what p of x1 parametrized by mu1 and sigma squared 1 and p of x2, parametrized by mu 2 and sigma squared 2, that would look like these two distributions over here. And, turns out that if were to plot of p of x, right, which is the product of these two things, you can actually get a surface plot that looks like this. This is a plot of p of x, where the height above of this, where the height of this surface at a particular point, so given a particular x1 x2 values of x2 if x1 equals 2, x equal 2, that's this point. And the height of this 3-D surface here, that's p of x. So p of x, that is the height of this plot, is literally just p of x1 parametrized by mu 1 sigma squared 1, times p of x2 parametrized by mu 2 sigma squared 2. Now, so this is how we fit the parameters to this data. Let's see if we have a couple of new examples. Maybe I have a new example there. Is this an anomaly or not? Or, maybe I have a different example, maybe I have a different second example over there. So, is that an anomaly or not? They way we do that is, we would set

some value for Epsilon, let's say I've chosen Epsilon equals 0.02. I'll say later how we choose Epsilon. But let's take this first example, let me call this example X1 test. And let me call the second example X2 test. What we do is, we then compute p of X1 test, so we use this formula to compute it and this looks like a pretty large value. In particular, this is greater than, or greater than or equal to epsilon. And so this is a pretty high probability at least bigger than epsilon, so we'll say that X1 test is not an anomaly. Whereas, if you compute p of X2 test, well that is just a much smaller value. So this is less than epsilon and so we'll say that that is indeed an anomaly, because it is much smaller than that epsilon that we then chose. And in fact, I'd improve it here. What this is really saying is that, you look through the 3d surface plot. It's saying that all the values of x1 and x2 that have a high height above the surface, corresponds to an a non-anomalous example of an OK or normal example. Whereas all the points far out here, all the points out here, all of those points have very low probability, so we are going to flag those points as anomalous, and so it's gonna define some region, that maybe looks like this, so that everything outside this, it flags as anomalous, whereas the things inside this ellipse I just drew, if it considers okay, or non-anomalous, not anomalous examples. And so this example x2 test lies outside that region, and so it has very small probability, and so we consider it an anomalous example.



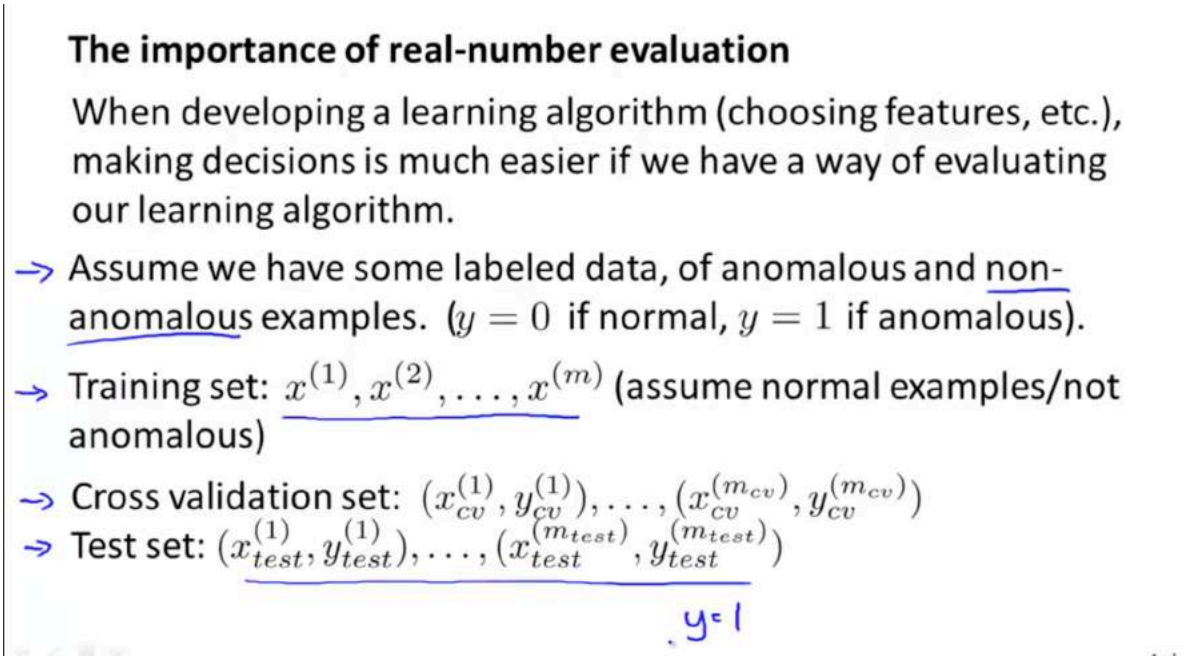
In this video we talked about how to estimate p of x, the probability of x, for the purpose of developing an anomaly detection algorithm. And in this video, we also stepped through an entire process of giving data set, we have, fitting the parameters, doing parameter estimations. We get mu and sigma parameters, and then taking new examples and deciding if the new examples are anomalous or not. In the next few videos we will delve deeper into this algorithm, and talk a bit more about how to actually get this to work well.

Building an Anomaly Detection System

Developing and Evaluating an Anomaly Detection System

In the last video, we developed an anomaly detection algorithm. In this video, I like to talk about the process of how to go about developing a specific application of anomaly detection to a problem and in particular this will focus on the problem of how to evaluate an anomaly detection algorithm. In previous videos, we've already talked about the importance of real number evaluation and this captures the idea that when you're trying to develop a learning algorithm for a specific application, you need to often make a lot of choices like, you know, choosing what features to use and then so on. And making decisions about all of these choices is often much easier, and if you have a way to evaluate your learning algorithm that just gives you back a number. So if you're trying to decide, you know, I have an idea for one extra feature, do I include this feature or not. If you can run the algorithm with the feature, and run the algorithm without the feature, and just get back a number that tells you, you know, did it improve or worsen performance to add this feature? Then it gives you a much better way, a much simpler way, with which to decide whether or not to include that feature. So in order to be able to develop an anomaly detection system quickly, it would be a really helpful to have a way of evaluating an anomaly detection system. In order to do this, in order to evaluate an anomaly detection system, we're actually going to assume have some labeled data. So, so far, we'll be treating anomaly detection as an unsupervised learning problem, using unlabeled data. But if you have some labeled data that specifies what are some anomalous examples, and what are some non-anomalous examples, then this is how we actually think of as the standard way of evaluating an anomaly detection algorithm. So taking the

aircraft engine example again. Let's say that, you know, we have some label data of just a few anomalous examples of some aircraft engines that were manufactured in the past that turns out to be anomalous. Turned out to be flawed or strange in some way. Let's say we use we also have some non-anomalous examples, so some perfectly okay examples. I'm going to use y equals 0 to denote the normal or the non-anomalous example and y equals 1 to denote the anomalous examples. The process of developing and evaluating an anomaly detection algorithm is as follows. We're going to think of it as a training set and talk about the cross validation in test sets later, but the training set we usually think of this as still the unlabeled training set. And so this is our large collection of normal, non-anomalous or not anomalous examples. And usually we think of this as being as non-anomalous, but it's actually okay even if a few anomalies slip into your unlabeled training set. And next we are going to define a cross validation set and a test set, with which to evaluate a particular anomaly detection algorithm. So, specifically, for both the cross validation test sets we're going to assume that, you know, we can include a few examples in the cross validation set and the test set that contain examples that are known to be anomalous. So the test sets say we have a few examples with y equals 1 that correspond to anomalous aircraft engines.



So here's a specific example. Let's say that, altogether, this is the data that we have. We have manufactured 10,000 examples of engines that, as far as we know we're perfectly normal, perfectly good aircraft engines. And again, it turns out to be okay even if a few flawed engine slips into the set of 10,000 is actually okay, but we kind of assumed that the vast majority of these 10,000 examples are, you know, good and normal non-anomalous engines. And let's say that, you know, historically, however long we've been running on manufacturing plant, let's say that we end up getting features, getting 24 to 28 anomalous engines as well. And for a pretty typical application of anomaly

detection, you know, the number non-anomalous examples, that is with y equals 1, we may have anywhere from, you know, 20 to 50. It would be a pretty typical range of examples, number of examples that we have with y equals 1. And usually we will have a much larger number of good examples. So, given this data set, a fairly typical way to split it into the training set, cross validation set and test set would be as follows. Let's take 10,000 good aircraft engines and put 6,000 of that into the unlabeled training set. So, I'm calling this an unlabeled training set but all of these examples are really ones that correspond to y equals 0, as far as we know. And so, we will use this to fit p of x, right. So, we will use these 6000 engines to fit p of x, which is that p of x one parametrized by Mu 1, sigma squared 1, up to p of Xn parametrized by Mu N sigma squared n. And so it would be these 6,000 examples that we would use to estimate the parameters Mu 1, sigma squared 1, up to Mu N, sigma squared N. And so that's our training set of all, you know, good, or the vast majority of good examples. Next we will take our good aircraft engines and put some number of them in a cross validation set plus some number of them in the test sets. So 6,000 plus 2,000 plus 2,000, that's how we split up our 10,000 good aircraft engines. And then we also have 20 flawed aircraft engines, and we'll take that and maybe split it up, you know, put ten of them in the cross validation set and put ten of them in the test sets. And in the next slide we will talk about how to actually use this to evaluate the anomaly detection algorithm. So what I have just described here is a you know probably the recommend a good way of splitting the labeled and unlabeled example. The good and the flawed aircraft engines. Where we use like a 60, 20, 20% split for the good engines and we take the flawed engines, and we put them just in the cross validation set, and just in the test set, then we'll see in the next slide why that's the case. Just as an aside, if you look at how people apply anomaly detection algorithms, sometimes you see other peoples' split the data differently as well. So, another alternative, this is really not a recommended alternative, but some people want to take off your 10,000 good engines, maybe put 6000 of them in your training set and then put the same 4000 in the cross validation set and the test set. And so, you know, we like to think of the cross validation set and the test set as being completely different data sets to each other. But you know, in anomaly detection, you know, for sometimes you see people, sort of, use the same set of good engines in the cross validation sets, and the test sets, and sometimes you see people use exactly the same sets of anomalous engines in the cross validation set and the test set. And so, all of these are considered, you know, less good practices and definitely less recommended. Certainly using the same data in the cross validation set and the test set, that is not considered a good machine learning practice. But, sometimes you see people do this too.

Aircraft engines motivating example

- 10000 good (normal) engines
- 20 flawed engines (anomalous)

Training set: 6000 good engines ($y = 0$)

CV: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Alternative:

Training set: 6000 good engines

CV: 4000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 4000 good engines ($y = 0$), 10 anomalous ($y = 1$)

So, given the training cross validation and test sets, here's how you evaluate or here is how you develop and evaluate an algorithm. First, we take the training sets and we fit the model p of x. So, we fit, you know, all these Gaussians to my m unlabeled examples of aircraft engines, and these, I am calling them unlabeled examples, but these are really examples that we're assuming our goods are the normal aircraft engines. Then imagine that your anomaly detection algorithm is actually making prediction. So, on the cross validation of the test set, given that, say, test example X, think of the algorithm as predicting that y is equal to 1, p of x is less than epsilon, we must be taking zero, if p of x is greater than or equal to epsilon. So, given x, it's trying to predict, what is the label, given y equals 1 corresponding to an anomaly or is it y equals 0 corresponding to a normal example

Algorithm evaluation

- Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$
- On a cross validation/test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

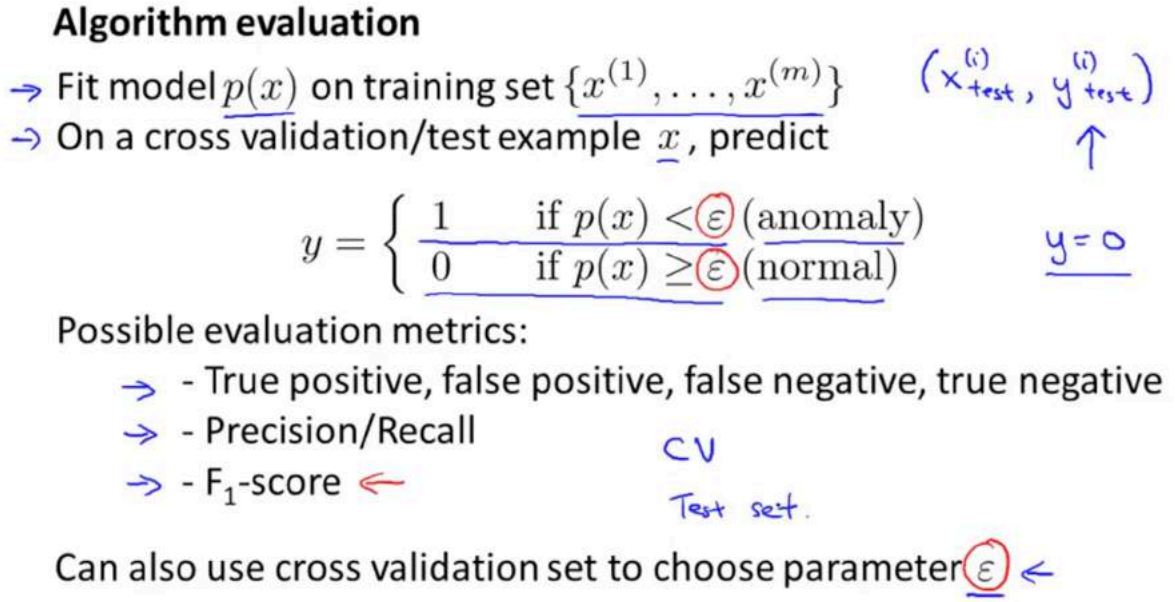
Possible evaluation metrics:

- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1-score

Can also use cross validation set to choose parameter ε

So given the training, cross validation, and test sets. How do you develop an algorithm? And more specifically, how do you evaluate an anomaly detection algorithm? Well, to this whole, the first step is to take the unlabeled training set, and to fit the model p of x lead training data. So you take this, you know on I'm coming, unlabeled training set, but really, these are examples that we are assuming, vast majority of which are normal aircraft engines, not because they're not anomalies and it will fit the model p of x. It will fit all those parameters for all the Gaussians on this data. Next on the cross validation of the test set, we're going to think of the anomaly detention algorithm as trying to predict the value of y. So in each of like say test examples. We have these X-I tests, Y-I test, where y is going to be equal to 1 or 0 depending on whether this was an anomalous example. So given input x in my test set, my anomaly detection algorithm think of it as predicting the y as 1 if p of x is less than epsilon. So predicting that it is an anomaly, it is probably is very low. And we think of the algorithm is predicting that y is equal to 0. If p of x is greater then or equals epsilon. So predicting those normal example if the p of x is reasonably large. And so we can now think of the anomaly detection algorithm as making predictions for what are the values of these y labels in the test sets or on the cross validation set. And this puts us somewhat more similar to the supervised learning setting, right? Where we have label test set and our algorithm is making predictions on these labels and so we can evaluate it you know by seeing how often it gets these labels right. Of course these labels are will be very skewed because y equals zero, that is normal examples, usually be much more common than y equals 1 than anomalous examples. But, you know, this is much closer to the source of evaluation metrics we can use in supervised learning. So what's a good evaluation metric to use. Well, because the data is very skewed, because y equals 0 is much more common, classification accuracy would not be a good the evaluation metrics. So, we talked about this in the earlier video. So, if you have a very skewed data set, then predicting y equals 0 all the time, will have very high classification accuracy. Instead, we should use evaluation metrics, like computing the fraction of true positives, false positives, false negatives, true negatives or compute the position of the v curve of this algorithm or do things like compute the f1 score, right, which is a single real number way of summarizing the position and the recall numbers. And so these would be ways to evaluate an anomaly detection algorithm on your cross validation set or on your test set. Finally, earlier in the anomaly detection algorithm, we also had this parameter epsilon, right? So, epsilon is this threshold that we would use to decide when to flag something as an anomaly. And so, if you have a cross validation set, another way to and to choose this parameter epsilon, would be to try a different, try many different values of epsilon, and then pick the value of epsilon that, let's say, maximizes f1 score, or that otherwise does well on your cross validation set. And more generally, the way to reduce the training, testing, and cross validation sets, is that when we are trying to make decisions,

like what features to include, or trying to, you know, tune the parameter epsilon, we would then continually evaluate the algorithm on the cross validation sets and make all those decisions like what features did you use, you know, how to set epsilon, use that, evaluate the algorithm on the cross validation set, and then when we've picked the set of features, when we've found the value of epsilon that we're happy with, we can then take the final model and evaluate it, you know, do the final evaluation of the algorithm on the test sets.



Question

Suppose you have fit a model p(x). When evaluating on the cross validation set or test set, your algorithm predicts:

$$y = \begin{cases} 1 & \text{if } p(x) \leq \epsilon \\ 0 & \text{if } p(x) > \epsilon \end{cases}$$

Is classification accuracy a good way to measure the algorithm's performance?

- Yes, because we have labels in the cross validation / test sets.
- No, because we do not have labels in the cross validation / test sets.
- No, because of skewed classes (so an algorithm that always predicts y = 0 will have high accuracy).

Correct

- No for the cross validation set; yes for the test set.

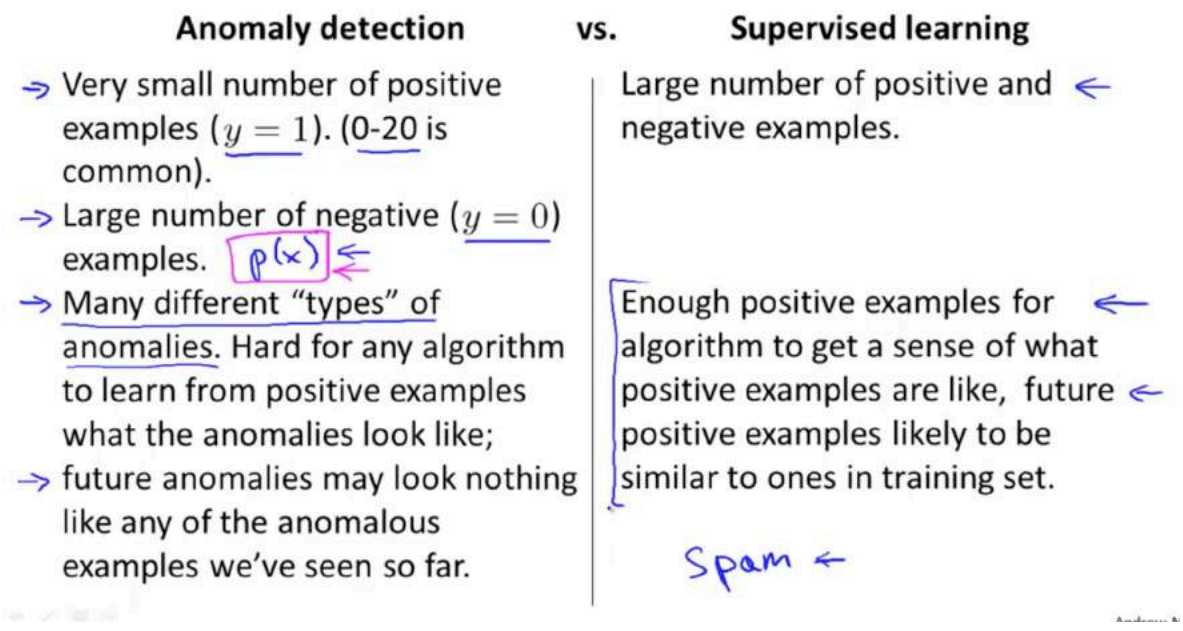
So, in this video, we talked about the process of how to evaluate an anomaly detection algorithm, and again, having being able to evaluate an algorithm, you know, with a single real number evaluation, with a number like an F1 score that often allows you to much more efficient use of your time when you are trying to develop an anomaly detection system. And we try to make these sorts of decisions. I have to chose epsilon, what features to include, and so on. In this video, we started to use a bit of labeled data in order to evaluate the anomaly detection algorithm and this takes us a little bit closer to a supervised learning setting. In the next video, I'm going to say a bit more about that. And in particular we'll talk about when should you be using an anomaly detection algorithm and when should we be thinking about using supervised learning instead, and what are the differences between these two formalisms.

Anomaly Detection vs. Supervised Learning

In the last video we talked about the process of evaluating an anomaly detection algorithm. And there we started to use some label data with

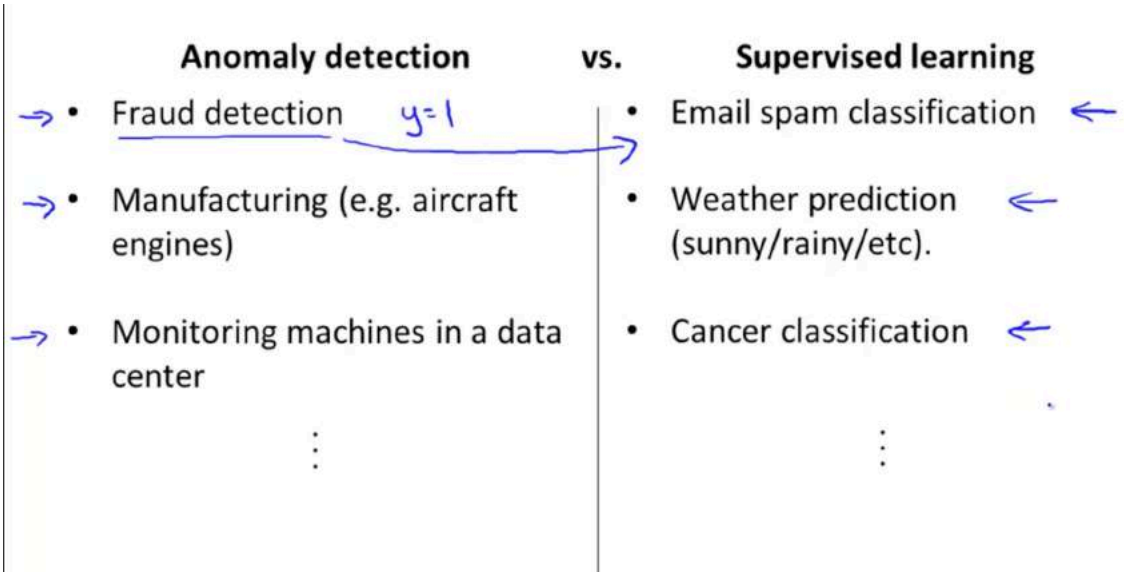
examples that we knew were either anomalous or not anomalous with Y equals one, or Y equals 0. And so, the question then arises of, and if we have the label data, that we have some examples and know the anomalies, and some of them will not be anomalies. Why don't we just use a supervisor on half of them? So why don't we just use logistic regression, or a neuro network to try to learn directly from our labeled data to predict whether Y equals one or Y equals 0. In this video, I'll try to share with you some of the thinking and some guidelines for when you should probably use an anomaly detection algorithm, and whether it might be more fruitful instead of using a supervisor in the algorithm. This slide shows what are the settings under which you should maybe use anomaly detection versus when supervised learning might be more fruitful. If you have a problem with a very small number of positive examples, and remember the examples of y equals one are the anomaly examples. Then you might consider using an anomaly detection algorithm instead. So, having 0 to 20, it may be up to 50 positive examples, might be pretty typical. And usually we have such a small positive, set of positive examples, we're going to save the positive examples just for the cross validation set in the test set. And in contrast, in a typical normal anomaly detection setting, we will often have a relatively large number of negative examples of the normal examples of normal aircraft engines. And we can then use this very large number of negative examples With which to fit the model p(x). And so there's this idea that in many anomaly detection applications, you have very few positive examples and lots of negative examples. And when we're doing the process of estimating p(x), affecting all those Gaussian parameters, we need only negative examples to do that. So if you have a lot negative data, we can still fit p(x) pretty well. In contrast, for supervised learning, more typically we would have a reasonably large number of both positive and negative examples. And so this is one way to look at your problem and decide if you should use an anomaly detection algorithm or a supervised. Here's another way that people often think about anomaly detection. So for anomaly detection applications, often there are very different types of anomalies. So think about so many different ways for go wrong. There are so many things that could go wrong that could the aircraft engine. And so if that's the case, and if you have a pretty small set of positive examples, then it can be hard for an algorithm, difficult for an algorithm to learn from your small set of positive examples what the anomalies look like. And in particular, you know future anomalies may look nothing like the ones you've seen so far. So maybe in your set of positive examples, maybe you've seen 5 or 10 or 20 different ways that an aircraft engine could go wrong. But maybe tomorrow, you need to detect a totally new set, a totally new type of anomaly. A totally new way for an aircraft engine to be broken, that you've just never seen before. And if that's the case, it might be more promising to just model the negative examples with this sort of calcium model p of x instead of try to hard to model the positive examples. Because tomorrow's anomaly may be nothing like the ones you've seen so far. In contrast, in some other problems, you have enough positive examples for an algorithm to get a sense of what the positive examples are like. In particular, if you think that future

positive examples are likely to be similar to ones in the training set; then in that setting, it might be more reasonable to have a supervisor in the algorithm that looks at all of the positive examples, looks at all of the negative examples, and uses that to try to distinguish between positives and negatives. Hopefully, this gives you a sense of if you have a specific problem, should you think about using an anomaly detection algorithm, or a supervised learning algorithm. And a key difference really is that in anomaly detection, often we have such a small number of positive examples that it is not possible for a learning algorithm to learn that much from the positive examples. And so what we do instead is take a large set of negative examples and have it just learn a lot, learn p(x) from just the negative examples. Of the normal [INAUDIBLE] and we've reserved the small number of positive examples for evaluating our algorithms to use in the either the transvalidation set or the test set. And just as a side comment about this many different types of easier. In some earlier videos we talked about the email spam examples. In those examples, there are actually many different types of spam email, right? There's spam email that's trying to sell you things. Spam email trying to steal your passwords, this is called phishing emails and many different types of spam emails. But for the spam problem we usually have enough examples of spam email to see most of these different types of spam email because we have a large set of examples of spam. And that's why we usually think of spam as a supervised learning setting even though there are many different types of.



If we look at some applications of anomaly detection versus supervised learning we'll find fraud detection. If you have many different types of ways for people to try to commit fraud and a relatively small number of fraudulent users on your website, then I use an anomaly detection algorithm. I should say, if you have, if you're a very major online retailer and if you actually have had a lot of

people commit fraud on your website, so you actually have a lot of examples of y=1, then sometimes fraud detection could actually shift over to the supervised learning column. But, if you haven't seen that many examples of users doing strange things on your website, then more frequently fraud detection is actually treated as an anomaly detection algorithm rather than a supervised learning algorithm. Other examples, we've talked about manufacturing already. Hopefully, you see more and more examples are not that many anomalies but if again for some manufacturing processes, if you manufacture in very large volumes and you see a lot of bad examples, maybe manufacturing can shift to the supervised learning column as well. But if you haven't seen that many bad examples of so to do the anomaly detection monitoring machines in a data center [INAUDIBLE] similar source of apply. Whereas, you must have classification, weather prediction, and classifying cancers. If you have equal numbers of positive and negative examples. Your positive and your negative examples, then we would tend to treat all of these as supervisor problems.



Question

Which of the following problems would you approach with an anomaly detection algorithm (rather than a supervised learning algorithm)? Check all that apply.

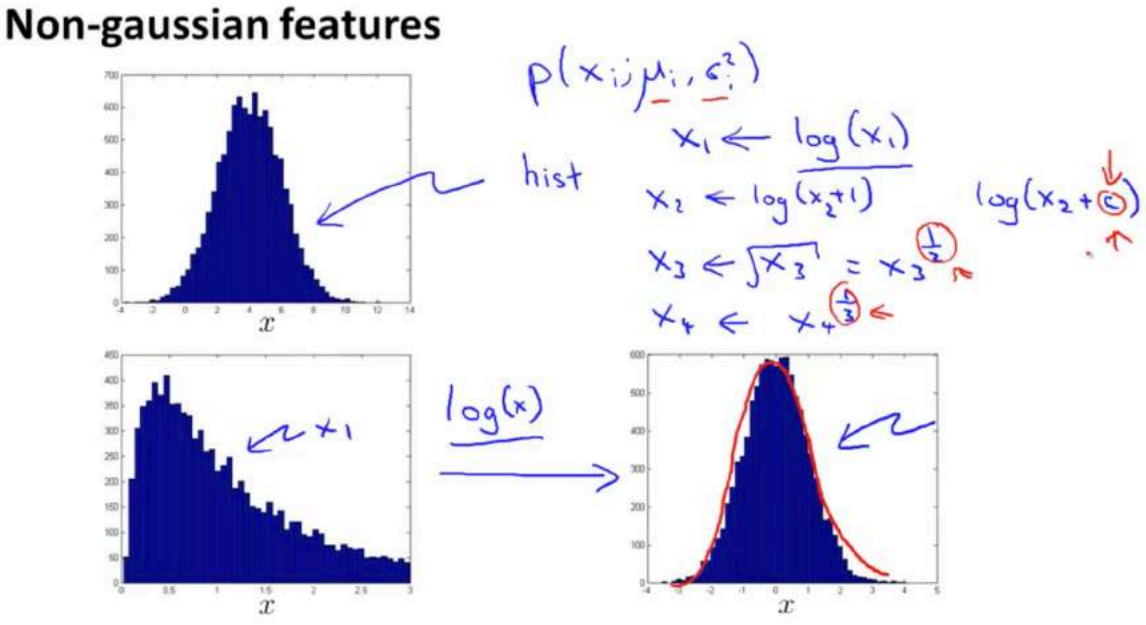
- [x] You run a power utility (supplying electricity to customers) and want to monitor your electric plants to see if any one of them might be behaving strangely.
- [ ] You run a power utility and want to predict tomorrow's expected demand for electricity (so that you can plan to ramp up an appropriate amount of generation capacity).
- [x] A computer vision / security application, where you examine video images to see if anyone in your company's parking lot is acting in an unusual way.
- [ ] A computer vision application, where you examine an image of a person entering your retail store to determine if the person is male or female.

So hopefully, that gives you a sense of one of the properties of a learning problem that would cause you to treat it as an anomaly detection problem versus a supervisory problem. And for many other problems that are faced by various technology companies and so on, we actually are in the settings where we have very few or sometimes zero positive training examples. There's just so many different types of anomalies that we've never seen them before. And for those sorts of problems, very often the algorithm that is used is an anomaly detection algorithm.

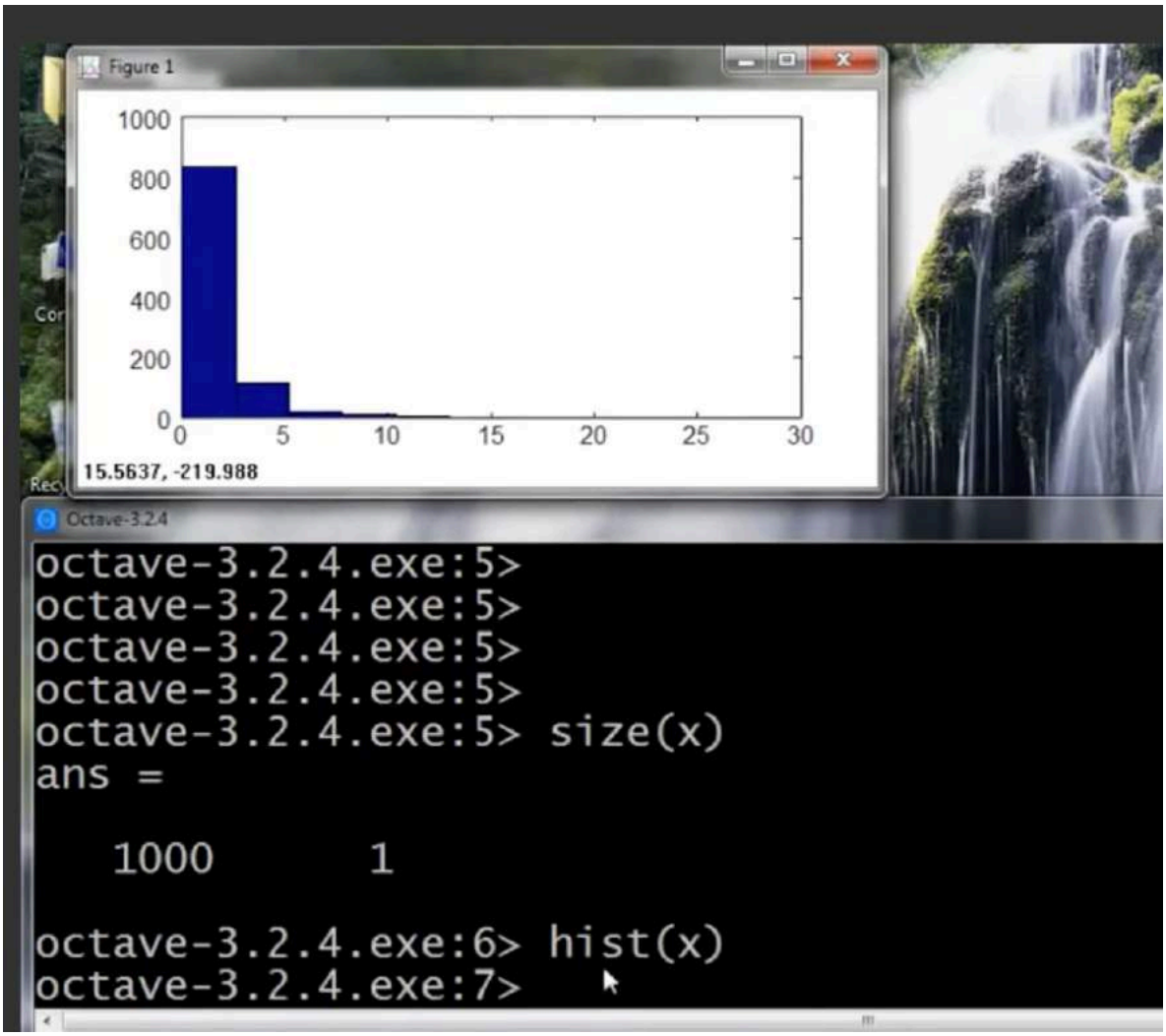
Choosing What Features to Use

By now you've seen the anomaly detection algorithm and we've also talked about how to evaluate an anomaly detection algorithm. It turns out, that when you're applying anomaly detection, one of the things that has a huge effect on how well it does, is what features you use, and what features you choose, to give the anomaly detection algorithm. So in this video, what I'd like to do is say a few words, give some suggestions and guidelines for how to go about designing or selecting features give to an anomaly detection algorithm. In our anomaly detection algorithm, one of the things we did was model the features using this sort of Gaussian distribution. With xi to mu i, sigma squared i, lets say. And so one thing that I often do would be to plot the data or the histogram of the data, to make sure that the data looks vaguely Gaussian before feeding it to my anomaly detection algorithm. And, it'll usually work okay, even if your data isn't Gaussian, but this is sort of a nice sanitary check to run. And by the way, in case your data looks non-Gaussian, the algorithms will often work just

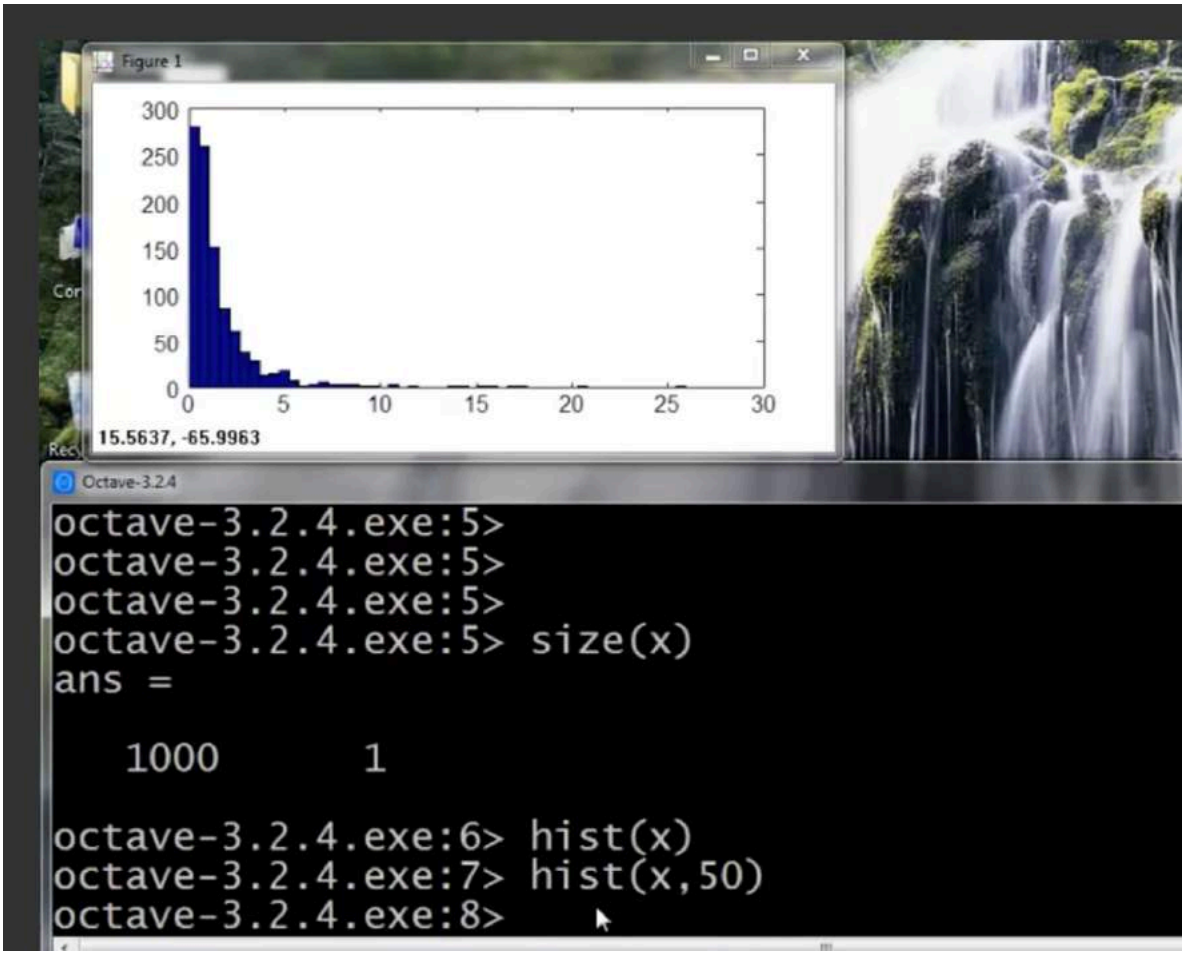
find. But, concretely if I plot the data like this, and if it looks like a histogram like this, and the way to plot a histogram is to use the HIST, or the HIST command in Octave, but it looks like this, this looks vaguely Gaussian, so if my features look like this, I would be pretty happy feeding into my algorithm. But if i were to plot a histogram of my data, and it were to look like this well, this doesn't look at all like a bell shaped curve, this is a very asymmetric distribution, it has a peak way off to one side. If this is what my data looks like, what I'll often do is play with different transformations of the data in order to make it look more Gaussian. And again the algorithm will usually work okay, even if you don't. But if you use these transformations to make your data more gaussian, it might work a bit better. So given the data set that looks like this, what I might do is take a log transformation of the data and if i do that and re-plot the histogram, what I end up with in this particular example, is a histogram that looks like this. And this looks much more Gaussian, right? This looks much more like the classic bell shaped curve, that we can fit with some mean and variance paramater sigma. So what I mean by taking a log transform, is really that if I have some feature x1 and then the histogram of x1 looks like this then I might take my feature x1 and replace it with log of x1 and this is my new x1 that I'll plot to the histogram over on the right, and this looks much more Guassian. Rather than just a log transform some other things you can do, might be, let's say I have a different feature x2, maybe I'll replace that will log x plus 1, or more generally with log x with x2 and some constant c and this constant could be something that I play with, to try to make it look as Gaussian as possible. Or for a different feature x3, maybe I'll replace it with x3, I might take the square root. The square root is just x3 to the power of one half, right? And this one half is another example of a parameter I can play with. So, I might have x4 and maybe I might instead replace that with x4 to the power of something else, maybe to the power of 1/3. And these, all of these, this one, this exponent parameter, or the C parameter, all of these are examples of parameters that you can play with in order to make your data look a little bit more Gaussian.



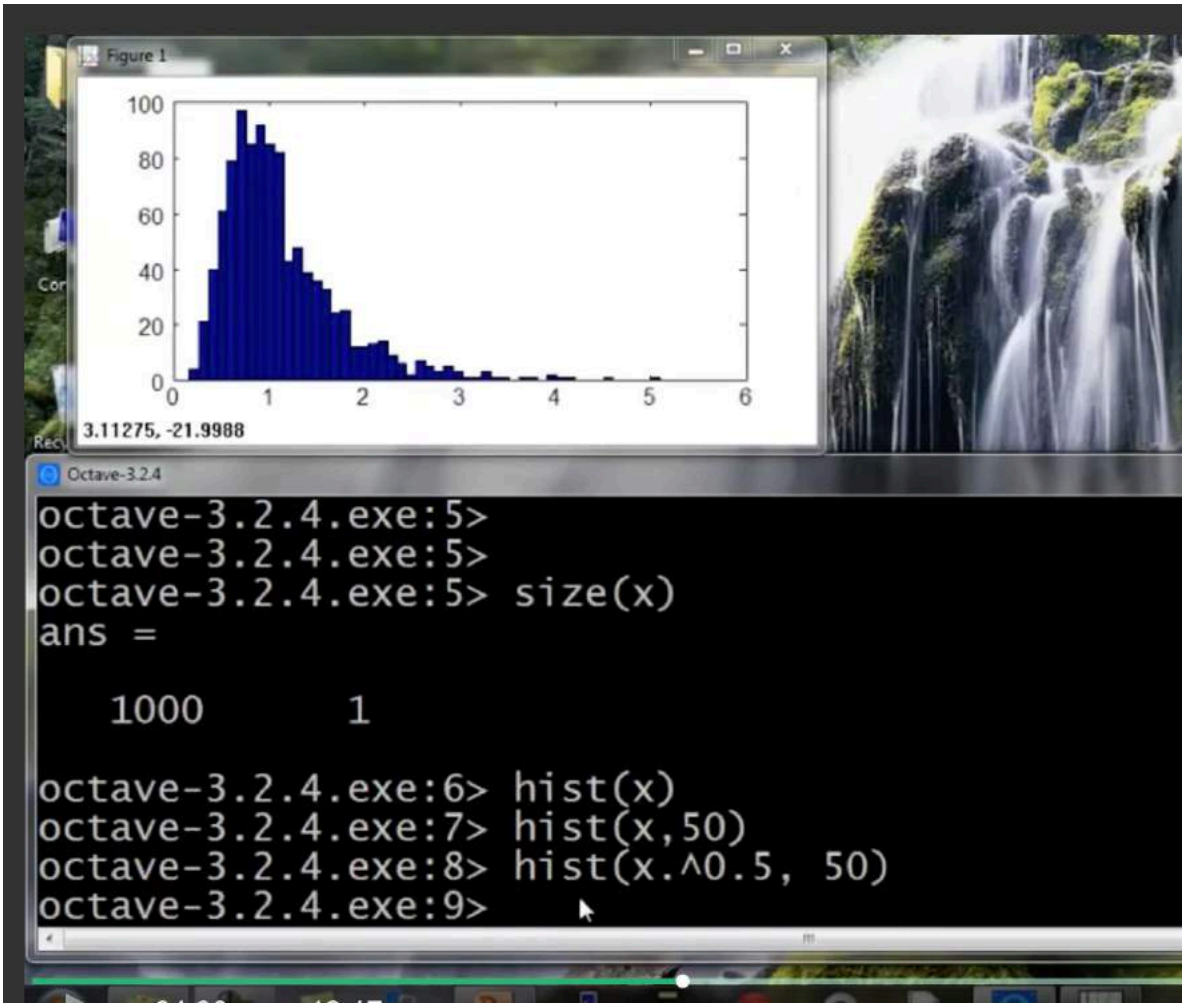
So, let me show you a live demo of how I actually go about playing with my data to make it look more Gaussian. So, I have already loaded in to octave here a set of features x I have a thousand examples loaded over there. So let's pull up the histogram of my data. Use the hist x command. So there's my histogram.



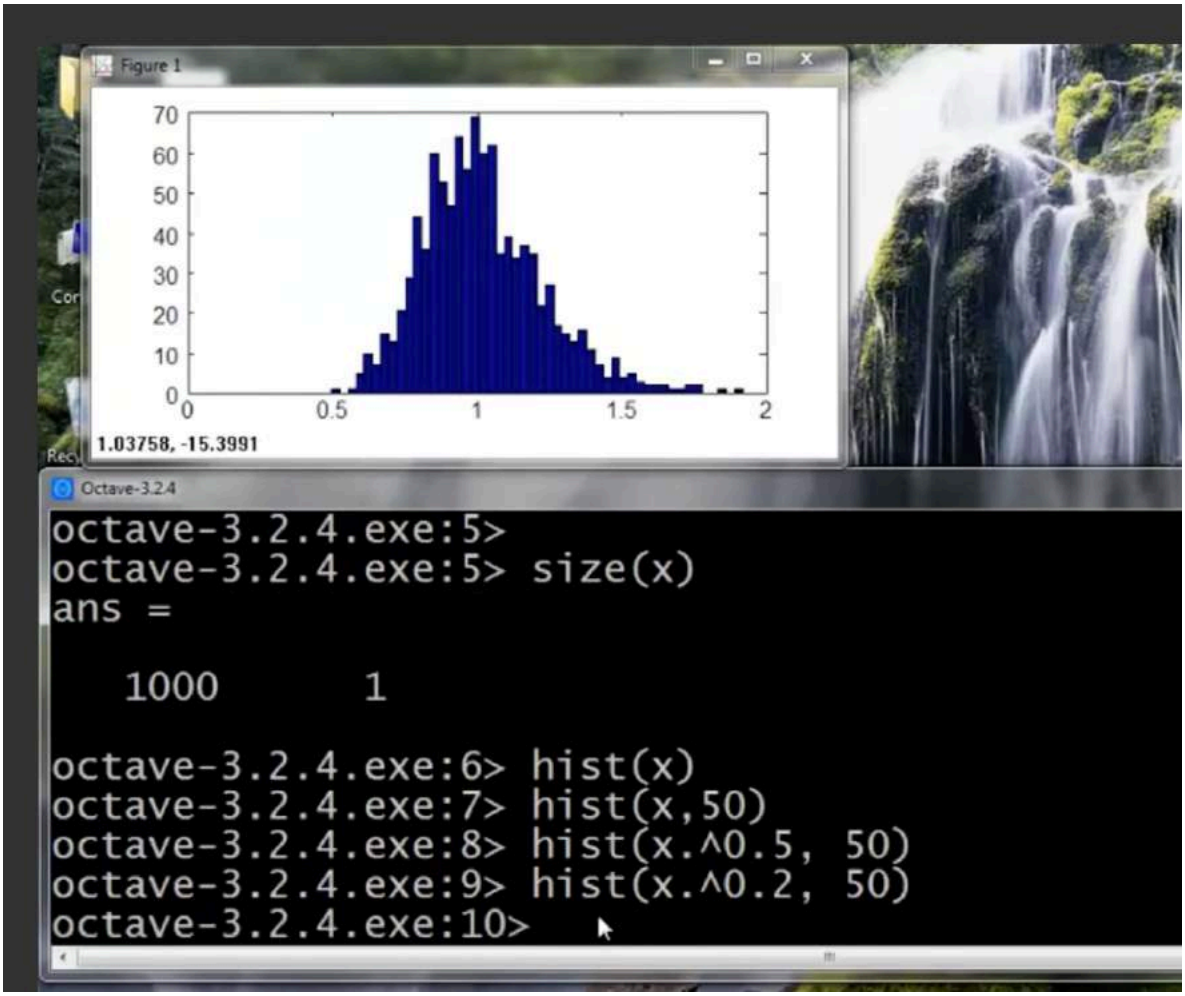
By default, I think this uses 10 bins of histograms, but I want to see a more fine grid histogram. So we do hist to the x, 50, so, this plots it in 50 different bins. Okay, that looks better. Now, this doesn't look very Gaussian, does it? So, lets start playing around with the data.



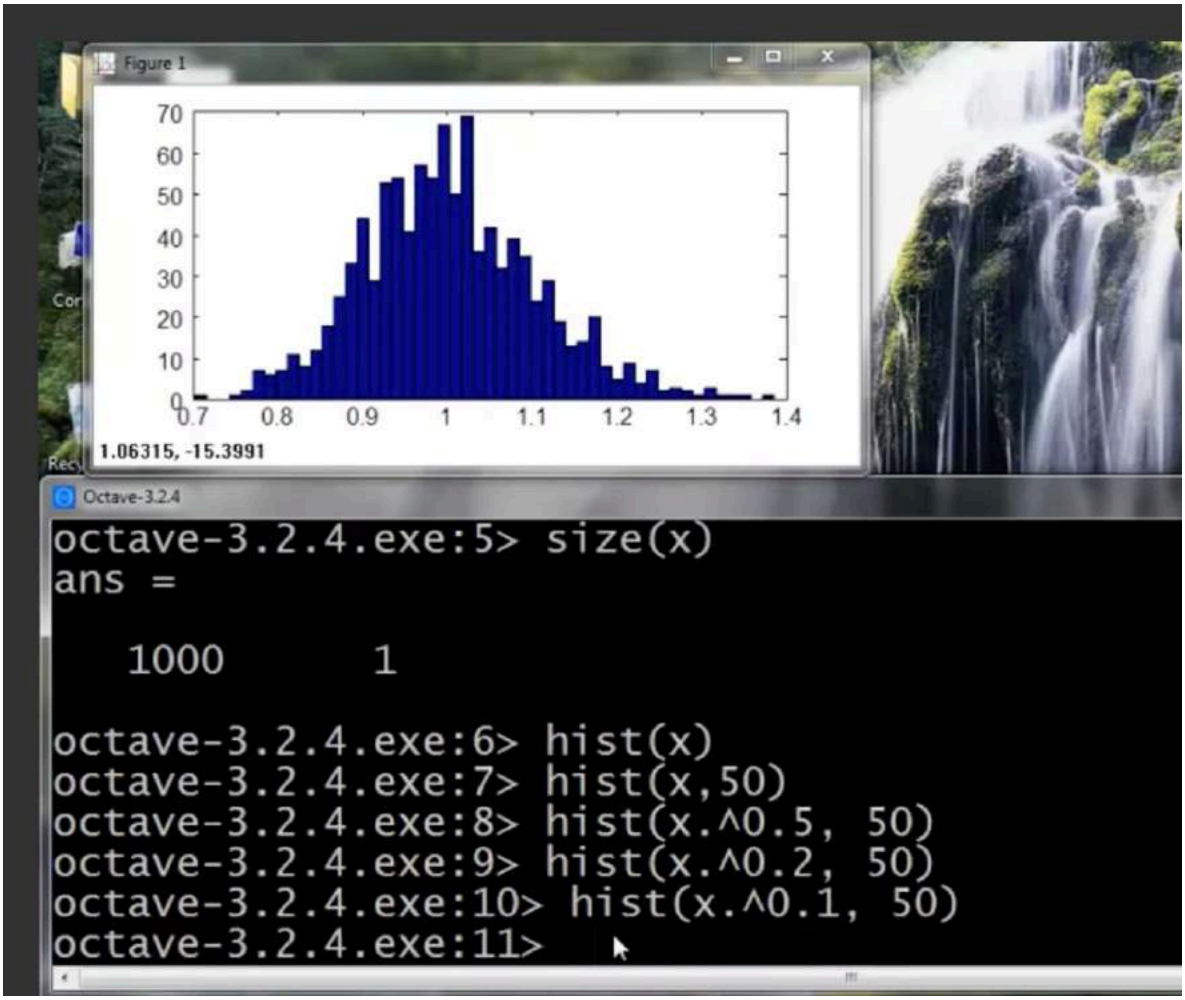
Lets try a hist of x to the 0.5. So we take the square root of the data, and plot that histogram. And, okay, it looks a little bit more Gaussian, but not quite there, so let's play at the 0.5 parameter. Let's see.



Set this to 0.2. Looks a little bit more Gaussian.



Let's reduce a little bit more 0.1. Yeah, that looks pretty good. I could actually just use 0.1.



Well, let's reduce it to 0.05. And, you know? Okay, this looks pretty Gaussian, so I can define a new feature which is x mu equals x to the 0.05, and now my new feature x Mu looks more Gaussian than my previous one and then I might instead use this new feature to feed into my anomaly detection algorithm.

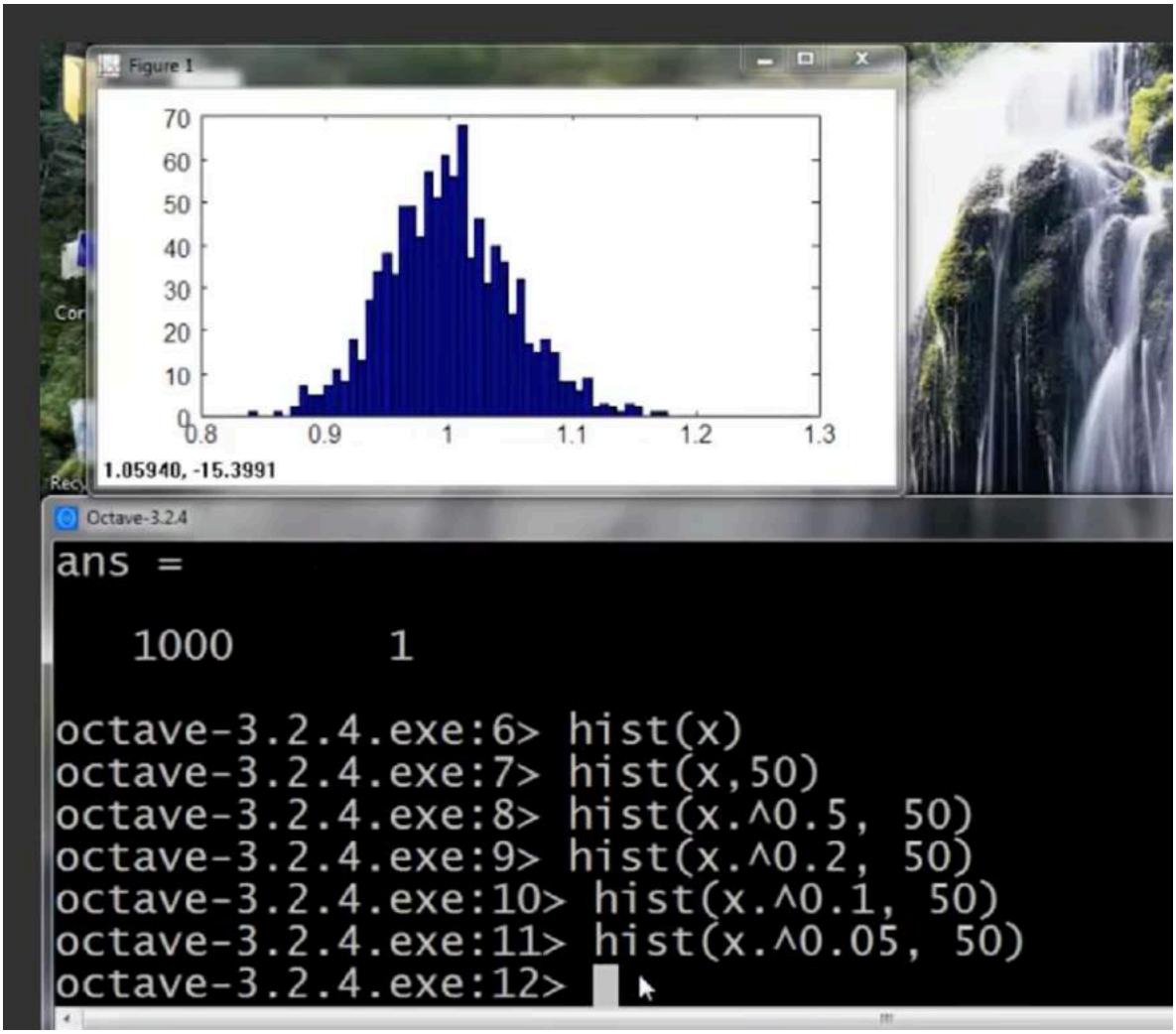
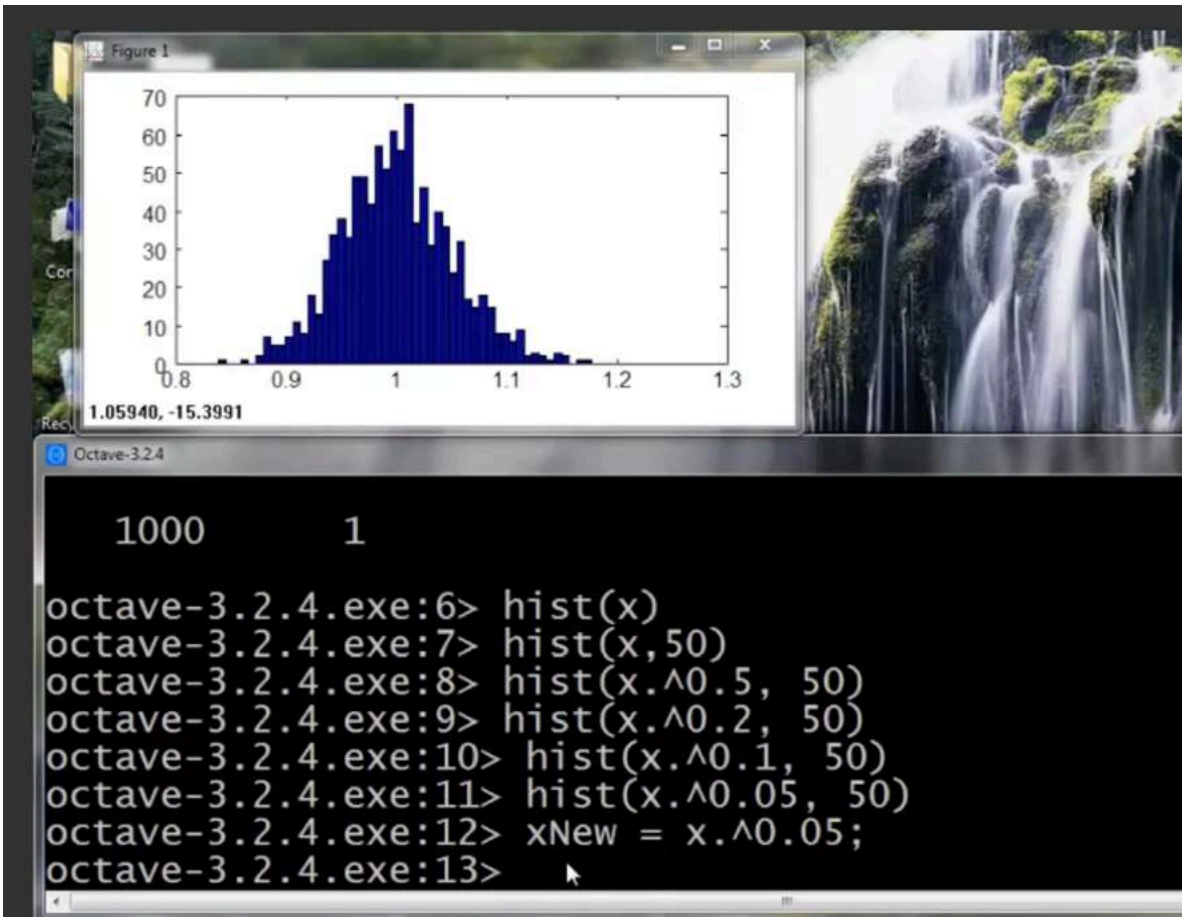
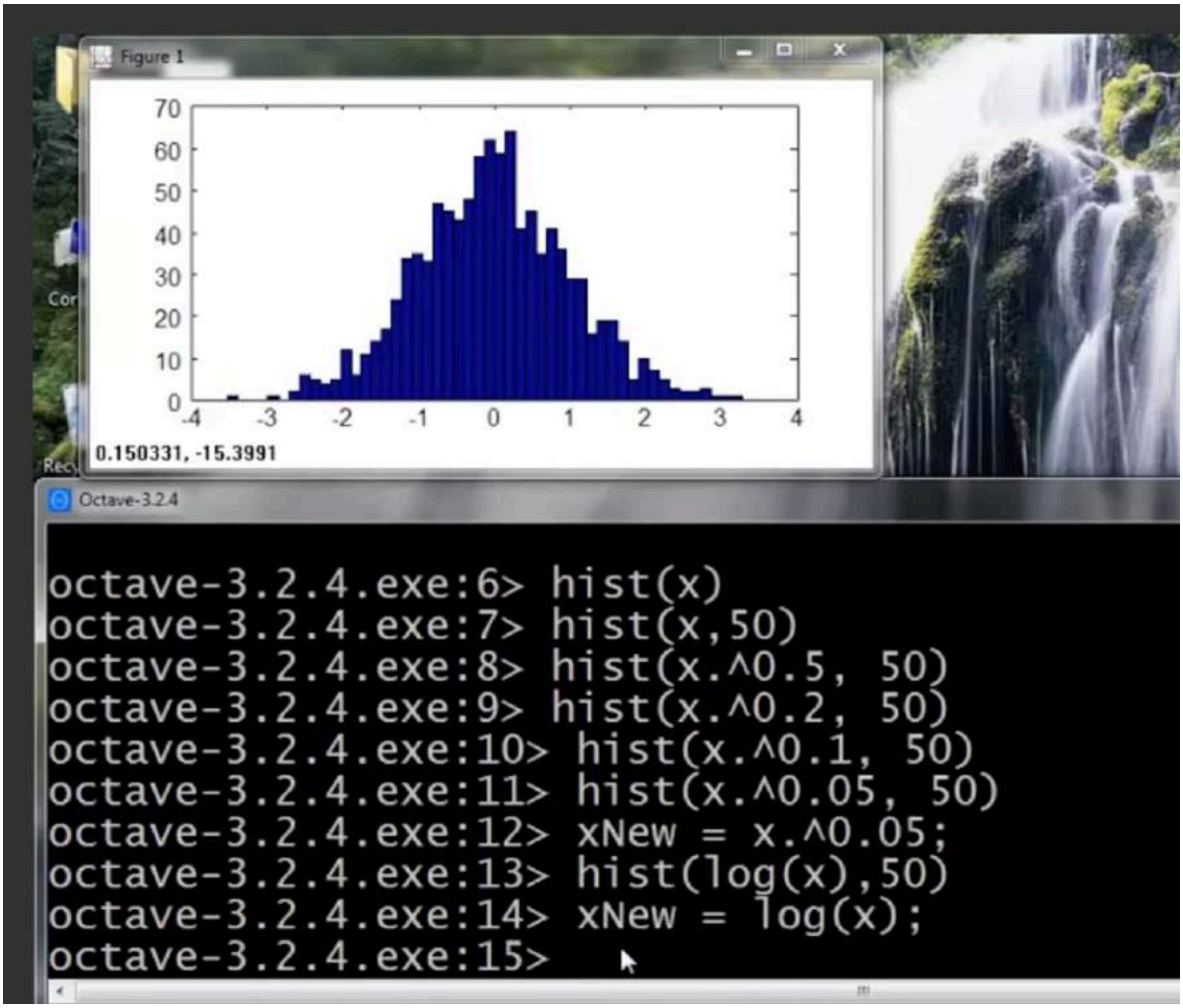


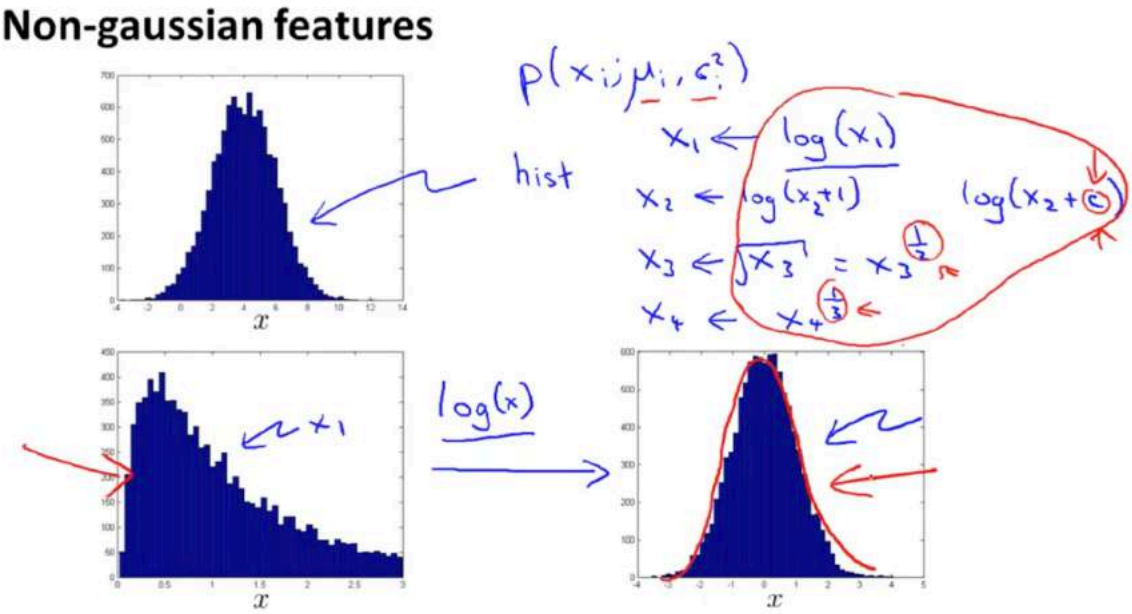
Figure 1
70
60
50
40
30
20
10
0
0.8
0.9
1
1.1
1.2
1.3
1.05940, -15.3991
Octave-3.2.4
ans =
1000 1
octave-3.2.4.exe:6> hist(x)
octave-3.2.4.exe:7> hist(x,50)
octave-3.2.4.exe:8> hist(x.^0.5, 50)
octave-3.2.4.exe:9> hist(x.^0.2, 50)
octave-3.2.4.exe:10> hist(x.^0.1, 50)
octave-3.2.4.exe:11> hist(x.^0.05, 50)
octave-3.2.4.exe:12>



And of course, there is more than one way to do this. You could also have hist of log of x, that's another example of a transformation you can use. And, you know, that also look pretty Gaussian. So, I can also define x mu equals log of x. and that would be another pretty good choice of a feature to use.

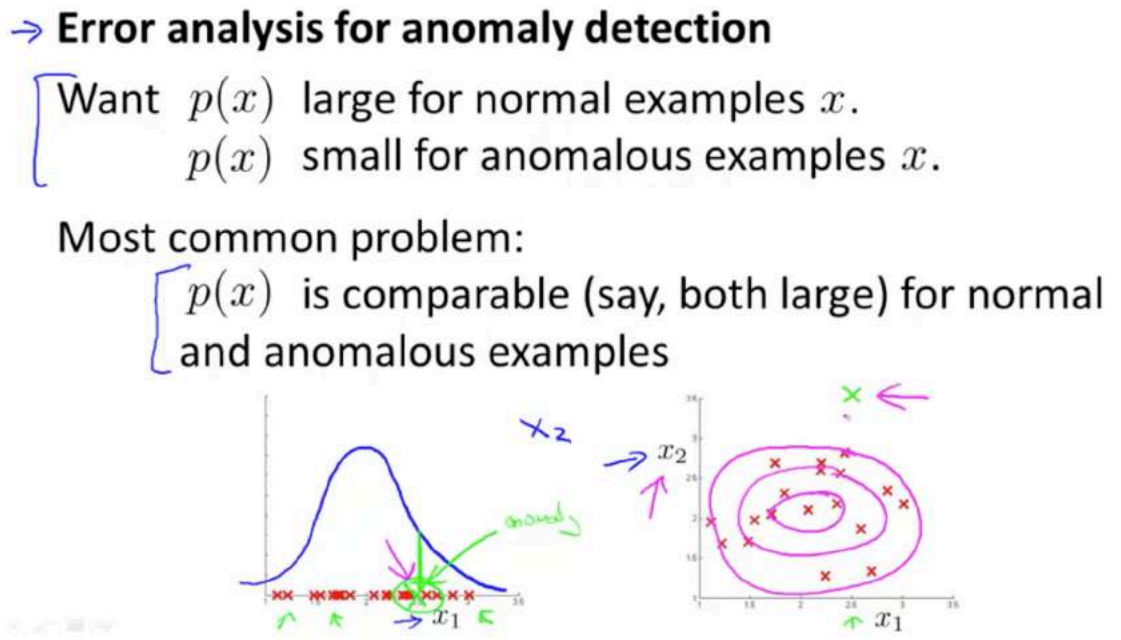


So to summarize, if you plot a histogram with the data, and find that it looks pretty non-Gaussian, it's worth playing around a little bit with different transformations like these, to see if you can make your data look a little bit more Gaussian, before you feed it to your learning algorithm, although even if you don't, it might work okay. But I usually do take this step.



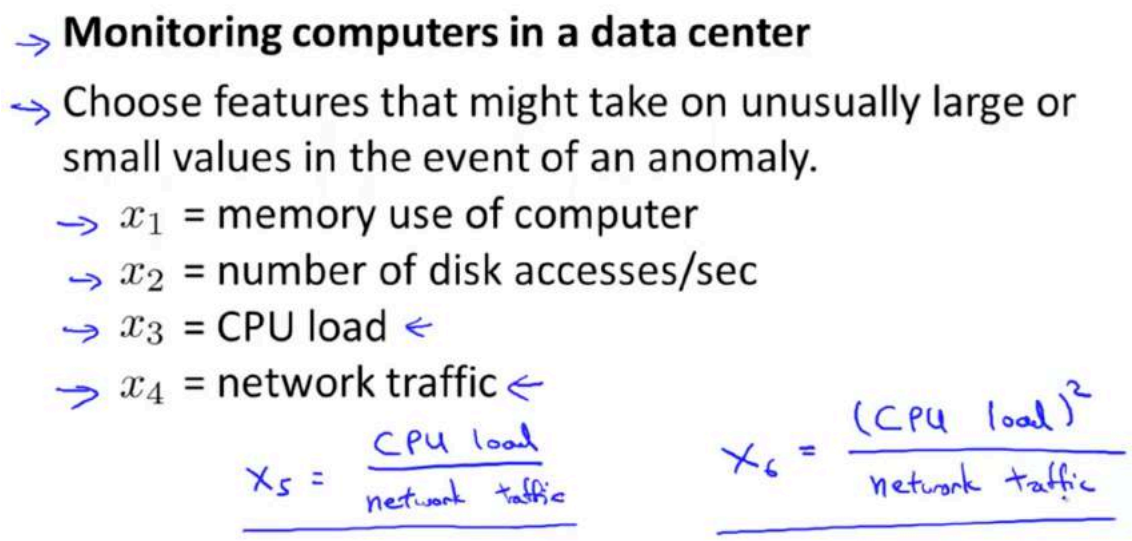
Now, the second thing I want to talk about is, how do you come up with features for an anomaly detection algorithm. And the way I often do so, is via an error analysis procedure. So what I mean by that, is that this is really similar to the error analysis procedure that we have for supervised learning, where we would train a complete algorithm, and run the algorithm on a cross validation set, and look at the examples it gets wrong, and see if we can come up with extra features to help the algorithm do better on the examples that it got wrong in the cross-validation set. So lets try to reason through an example of this process. In anomaly detection, we are hoping that p of x will be large for the normal examples and it will be small for the anomalous examples. And so a pretty common problem would be if p of x is comparable, maybe both are large for both the normal and the anomalous examples. Lets look at a specific example of that. Let's say that this is my unlabeled data. So, here I have just one feature, x1 and so I'm gonna fit a Gaussian to this. And maybe my Gaussian that I fit to my data looks like that. And now let's say I have an anomalous example, and let's say that my anomalous example takes on an x value of 2.5. So I plot my anomalous example there. And you know, it's kind of buried in the middle of a bunch of normal examples, and so, just this anomalous example that I've drawn in green, it gets a pretty high probability, where it's the height of the blue curve, and the algorithm fails to flag this as an anomalous example. Now, if this were maybe aircraft engine manufacturing or something, what I would do is, I would actually look at my training examples and look at what went wrong with that particular aircraft engine, and see, if looking at that example can inspire me to come up with a new feature x2, that helps to distinguish between this bad example, compared to the rest of my red examples, compared to all of my normal aircraft engines. And if I managed to do so, the hope would be then, that, if I can create a new feature, X2, so that

when I re-plot my data, if I take all my normal examples of my training set, hopefully I find that all my training examples are these red crosses here. And hopefully, if I find that for my anomalous example, the feature x2 takes on the the unusual value. So for my green example here, this anomaly, right, my X1 value, is still 2.5. Then maybe my X2 value, hopefully it takes on a very large value like 3.5 over there, or a very small value. But now, if I model my data, I'll find that my anomaly detection algorithm gives high probability to data in the central regions, slightly lower probability to that, sightly lower probability to that. An example that's all the way out there, my algorithm will now give very low probability to. And so, the process of this is, really look at the mistakes that it is making. Look at the anomaly that the algorithm is failing to flag, and see if that inspires you to create some new feature. So find something unusual about that aircraft engine and use that to create a new feature, so that with this new feature it becomes easier to distinguish the anomalies from your good examples. And so that's the process of error analysis and using that to create new features for anomaly detection.



Finally, let me share with you my thinking on how I usually go about choosing features for anomaly detection. So, usually, the way I think about choosing features is I want to choose features that will take on either very, very large values, or very, very small values, for examples that I think might turn out to be anomalies. So let's use our example again of monitoring the computers in a data center. And so you have lots of machines, maybe thousands, or tens of thousands of machines in a data center. And we want to know if one of the machines, one of our computers is acting up, so doing something strange. So here are examples of features you may choose, maybe memory used, number of disc accesses, CPU load, network traffic. But now, lets say that I suspect

one of the failure cases, let's say that in my data set I think that CPU load the network traffic tend to grow linearly with each other. Maybe I'm running a bunch of web servers, and so, here if one of my servers is serving a lot of users, I have a very high CPU load, and have a very high network traffic. But let's say, I think, let's say I have a suspicion, that one of the failure cases is if one of my computers has a job that gets stuck in some infinite loop. So if I think one of the failure cases, is one of my machines, one of my web servers-- server code-- gets stuck in some infinite loop, and so the CPU load grows, but the network traffic doesn't because it's just spinning it's wheels and doing a lot of CPU work, you know, stuck in some infinite loop. In that case, to detect that type of anomaly, I might create a new feature, X5, which might be CPU load divided by network traffic. And so here X5 will take on a unusually large value if one of the machines has a very large CPU load but not that much network traffic and so this will be a feature that will help your anomaly detection capture, a certain type of anomaly. And you can also get creative and come up with other features as well. Like maybe I have a feature x6 thats CPU load squared divided by network traffic. And this would be another variant of a feature like x5 to try to capture anomalies where one of your machines has a very high CPU load, that maybe doesn't have a commensurately large network traffic. And by creating features like these, you can start to capture anomalies that correspond to unusual combinations of values of the features.



Question

Suppose your anomaly detection algorithm is performing poorly and outputs a large value of p(x) for many normal examples and for many anomalous examples in your cross validation dataset. Which of the following changes to your algorithm is most likely to help?

- ○ Try using fewer features.
- ● Try coming up with more features to distinguish between the normal and the anomalous examples.
- ○ Get a larger training set (of normal examples) with which to fit p(x).
- ○ Try changing ϵ.

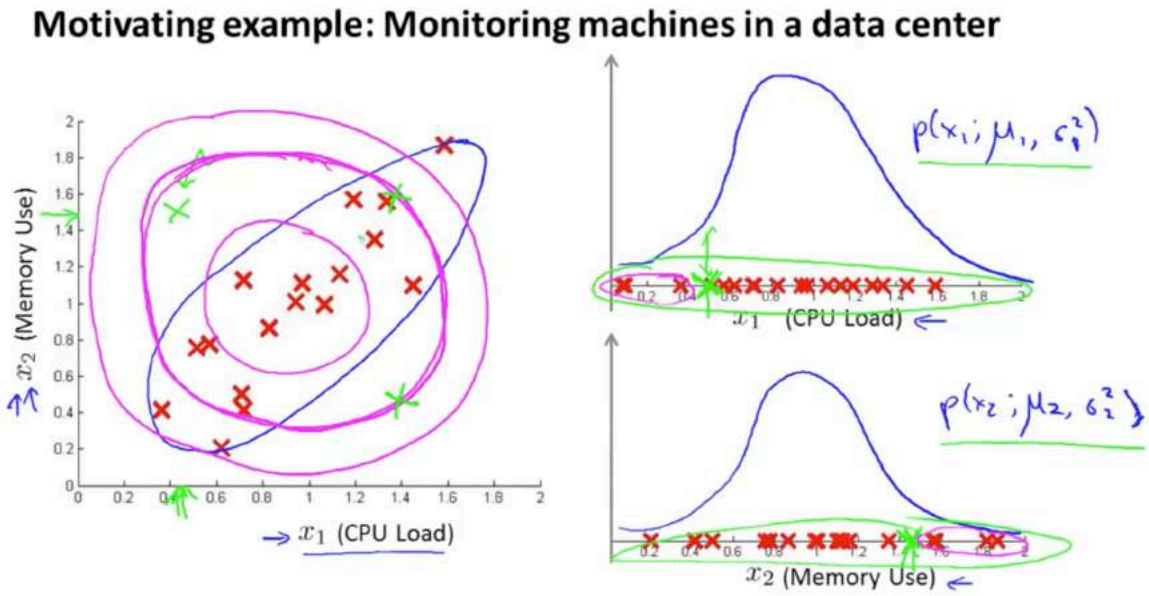
So in this video we talked about how to and take a feature, and maybe transform it a little bit, so that it becomes a bit more Gaussian, before feeding into an anomaly detection algorithm. And also the error analysis in this process of creating features to try to capture different types of anomalies. And with these sorts of guidelines hopefully that will help you to choose good features, to give to your anomaly detection algorithm, to help it capture all sorts of anomalies.

Multivariate Gaussian Distribution (Optional)

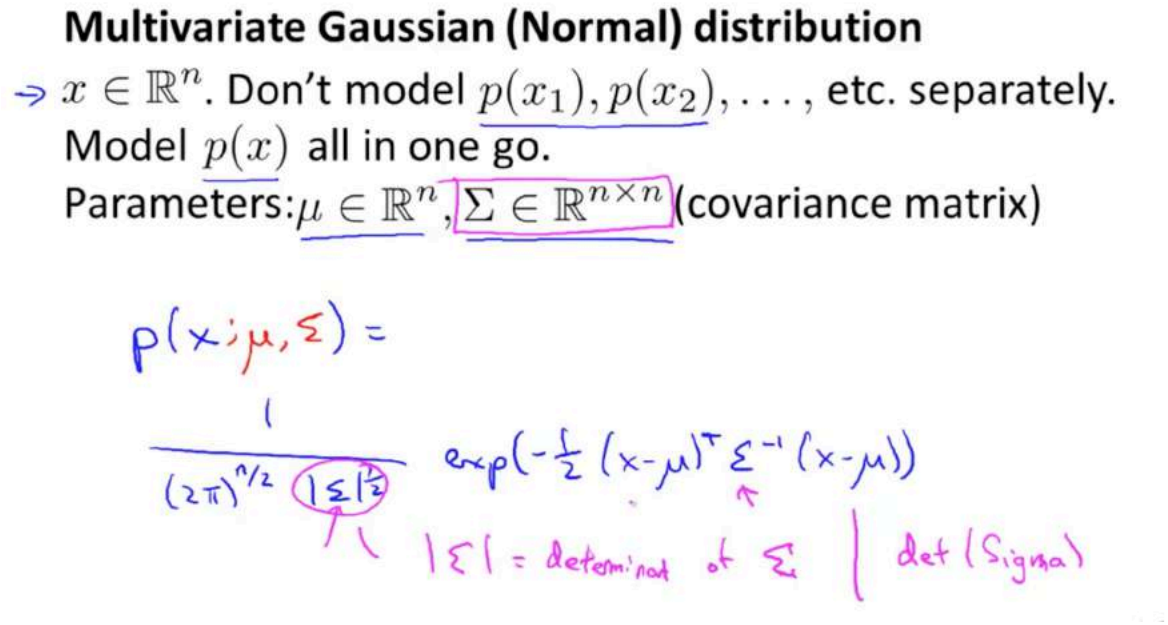
Multivariate Gaussian Distribution

In this and the next video, I'd like to tell you about one possible extension to the anomaly detection algorithm that we've developed so far. This extension uses something called the multivariate Gaussian distribution, and it has some advantages, and some disadvantages, and it can sometimes catch some anomalies that the earlier algorithm didn't. To motivate this, let's start with an example. Let's say that so our unlabeled data looks like what I have plotted

here. And I'm going to use the example of monitoring machines in the data center, monitoring computers in the data center. So my two features are x1 which is the CPU load and x2 which is maybe the memory use. So if I take my two features, x1 and x2, and I model them as Gaussians then here's a plot of my X1 features, here's a plot of my X2 features, and so if I fit a Gaussian to that, maybe I'll get a Gaussian like this, so here's P of X 1, which depends on the parameters mu 1, and sigma squared 1, and here's my memory used, and, you know, maybe I'll get a Gaussian that looks like this, and this is my P of X 2, which depends on mu 2 and sigma squared 2. And so this is how the anomaly detection algorithm models X1 and X2. Now let's say that in the test sets I have an example that looks like this. The location of that green cross, so the value of X 1 is about 0.4, and the value of X 2 is about 1.5. Now, if you look at the data, it looks like, yeah, most of the data data lies in this region, and so that green cross is pretty far away from any of the data I've seen. It looks like that should be raised as an anomaly. So, in my data, in my, in the data of my good examples, it looks like, you know, the CPU load, and the memory use, they sort of grow linearly with each other. So if I have a machine using lots of CPU, you know memory use will also be high, whereas this example, this green example it looks like here, the CPU load is very low, but the memory use is very high, and I just have not seen that before in my training set. It looks like that should be an anomaly. But let's see what the anomaly detection algorithm will do. Well, for the CPU load, it puts it at around there 0.5 and this reasonably high probability is not that far from other examples we've seen, maybe, whereas, for the memory use, this appointment, 0.5, whereas for the memory use, it's about 1.5, which is there. Again, you know, it's all to us, it's not terribly Gaussian, but the value here and the value here is not that different from many other examples we've seen, and so P of X 1, will be pretty high, reasonably high. P of X 2 reasonably high. I mean, if you look at this plot right, this point here, it doesn't look that bad, and if you look at this plot, you know across here, doesn't look that bad. I mean, I have had examples with even greater memory used, or with even less CPU use, and so this example doesn't look that anomalous. And so, an anomaly detection algorithm will fail to flag this point as an anomaly. And it turns out what our anomaly detection algorithm is doing is that it is not realizing that this blue ellipse shows the high probability region, is that, one of the thing is that, examples here, a high probability, and the examples, the next circle of from a lower probably, and examples here are even lower probability, and somehow, here are things that are, green cross there, it's pretty high probability, and in particular, it tends to think that, you know, everything in this region, everything on the line that I'm circling over, has, you know, about equal probability, and it doesn't realize that something out here actually has much lower probability than something over there.

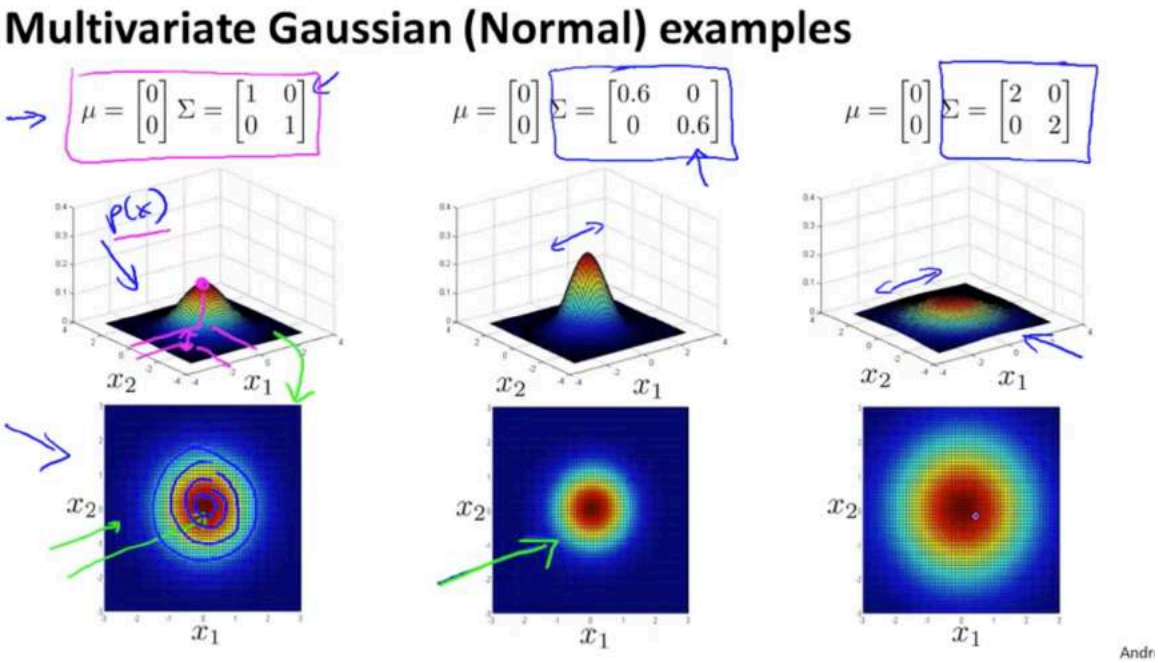


So, in order to fix this, we can, we're going to develop a modified version of the anomaly detection algorithm, using something called the multivariate Gaussian distribution also called the multivariate normal distribution. So here's what we're going to do. We have features x which are in Rn and instead of P of X 1, P of X 2, separately, we're going to model P of X, all in one go, so model P of X, you know, all at the same time. So the parameters of the multivariate Gaussian distribution are mu, which is a vector, and sigma, which is an n by n matrix, called a covariance matrix, and this is similar to the covariance matrix that we saw when we were working with the PCA, with the principal components analysis algorithm. For the second complete is, let me just write out the formula for the multivariate Gaussian distribution. So we say that probability of X, and this is parameterized by my parameters mu and sigma that the probability of x is equal to once again there's absolutely no need to memorize this formula. You know, you can look it up whenever you need to use it, but this is what the probability of X looks like. Transverse, 2nd inverse, X minus mu. And this thing here, the absolute value of sigma, this thing here when you write this symbol, this is called the determent of sigma and this is a mathematical function of a matrix and you really don't need to know what the determinant of a matrix is, but really all you need to know is that you can compute it in octave by using the octave command DET of sigma. Okay, and again, just be clear, alright? In this expression, these sigmas here, these are just n by n matrix. This is not a summation and you know, the sigma there is an n by n matrix. So that's the formula for P of X

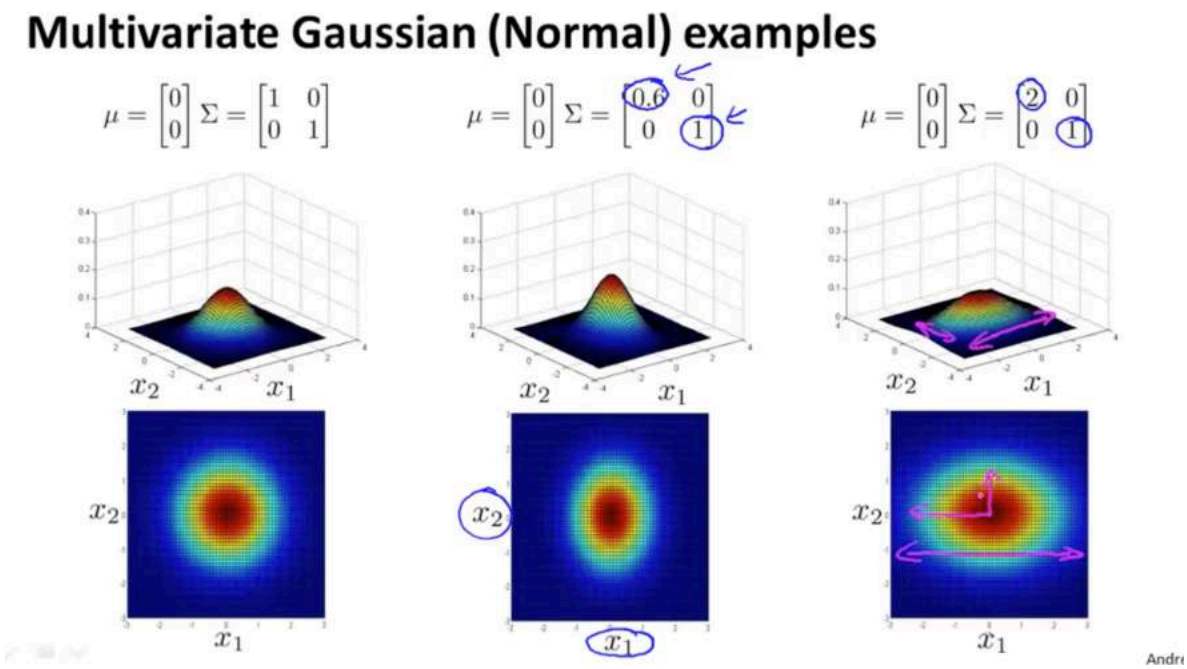


But it's more interestingly, or more importantly, what does P of X actually looks like? Lets look at some examples of multivariate Gaussian distributions. So let's take a two dimensional example, say if I have N equals 2, I have two features, X 1 and X 2. Lets say I set MU to be equal to 0 and sigma to be equal to this matrix here. With 1s on the diagonals and 0s on the off-diagonals, this matrix is sometimes also called the identity matrix. In that case, p of x will look like this, and what I'm showing in this figure is, you know, for a specific value of X1 and for a specific value of X2, the height of this surface the value of p of x. And so with this setting the parameters p of x is highest when X1 and X2 equal zero 0, so that's the peak of this Gaussian distribution, and the probability falls off with this sort of two dimensional Gaussian or this bell shaped two dimensional bell-shaped surface. Down below is the same thing but plotted using a contour plot instead, or using different colors, and so this heavy intense red in the middle, corresponds to the highest values, and then the values decrease with the yellow being slightly lower values the cyan being lower values and this deep blue being the lowest values so this is really the same figure but plotted viewed from the top instead, using colors instead. And so, with this distribution, you see that it faces most of the probability near 0,0 and then as you go out from 0,0 the probability of X1 and X2 goes down. Now lets try varying some of the parameters and see what happens. So let's take sigma and change it so let's say sigma shrinks a little bit. Sigma is a covariance matrix and so it measures the variance or the variability of the features X1 X2. So if the shrink sigma then what you get is what you get is that the width of this bump diminishes and the height also increases a bit, because the area under the surface is equal to 1. So the integral of the volume under the surface is equal to 1, because probability distribution must integrate to one. But, if you shrink the variance, it's kinda like shrinking sigma squared, you end up with a narrower distribution, and one that's a little bit taller. And so you

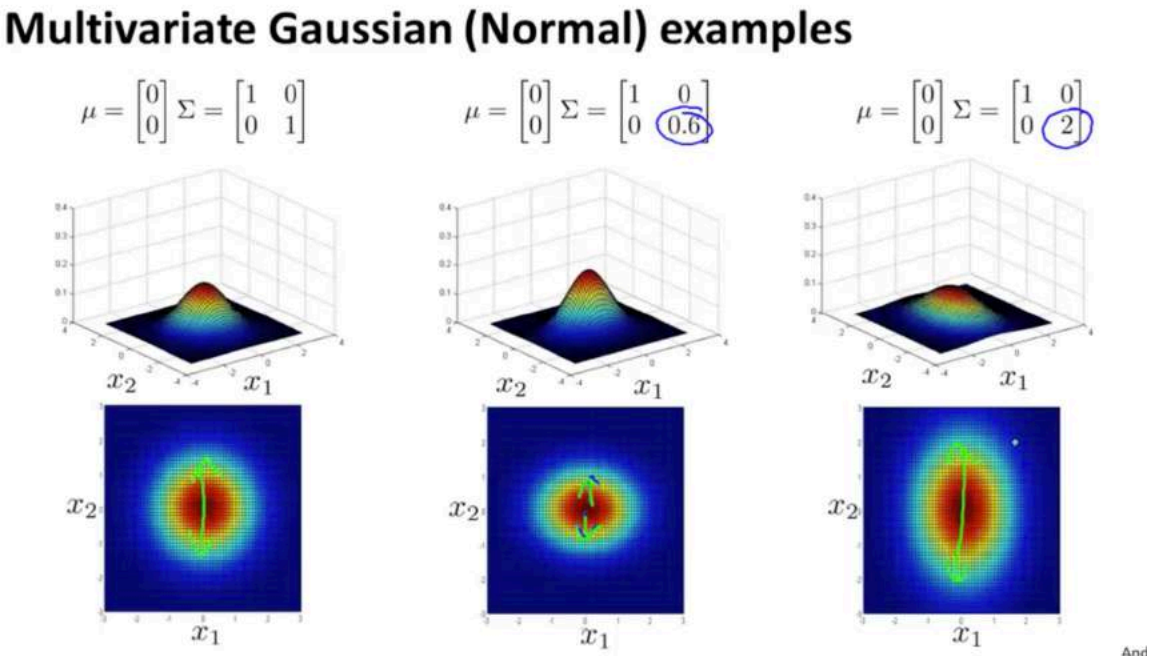
see here also the concentric ellipsis has shrunk a little bit. Whereas in contrast if you were to increase sigma to 2 2 on the diagonals, so it is now two times the identity then you end up with a much wider and much flatter Gaussian. And so the width of this is much wider. This is hard to see but this is still a bell shaped bump, it's just flattened down a lot, it has become much wider and so the variance or the variability of X1 and X2 just becomes wider.



Here are a few more examples. Now lets try varying one of the elements of sigma at the time. Let's say I send sigma to 0.6 there, and 1 over there. What this does, is this reduces the variance of the first feature, X 1, while keeping the variance of the second feature X 2, the same. And so with this setting of parameters, you can model things like that. X 1 has smaller variance, and X 2 has larger variance. Whereas if I do this, if I set this matrix to 2, 1 then you can also model examples where you know here we'll say X1 can have take on a large range of values whereas X2 takes on a relatively narrower range of values. And that's reflected in this figure as well, you know where, the distribution falls off more slowly as X 1 moves away from 0, and falls off very rapidly as X 2 moves away from 0.

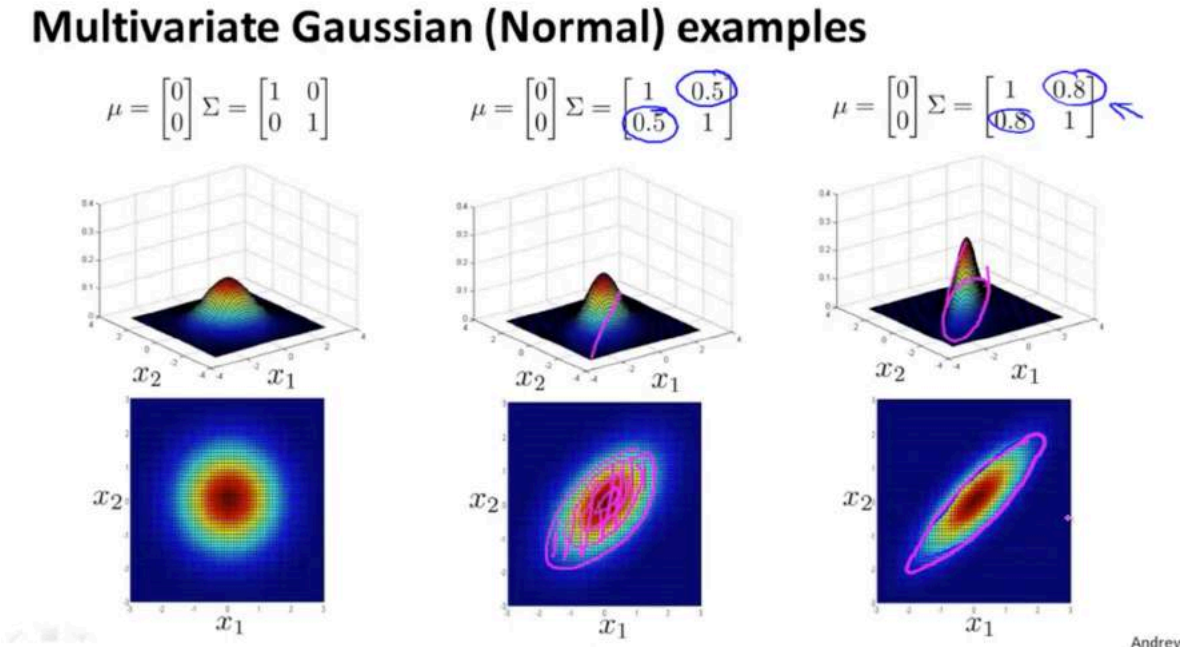


And similarly if we were to modify this element of the matrix instead, then similar to the previous slide, except that here where you know playing around here saying that X2 can take on a very small range of values and so here if this is 0.6, we notice now X2 tends to take on a much smaller range of values than the original example, whereas if we were to set sigma to be equal to 2 then that's like saying X2 you know, has a much larger range of values.

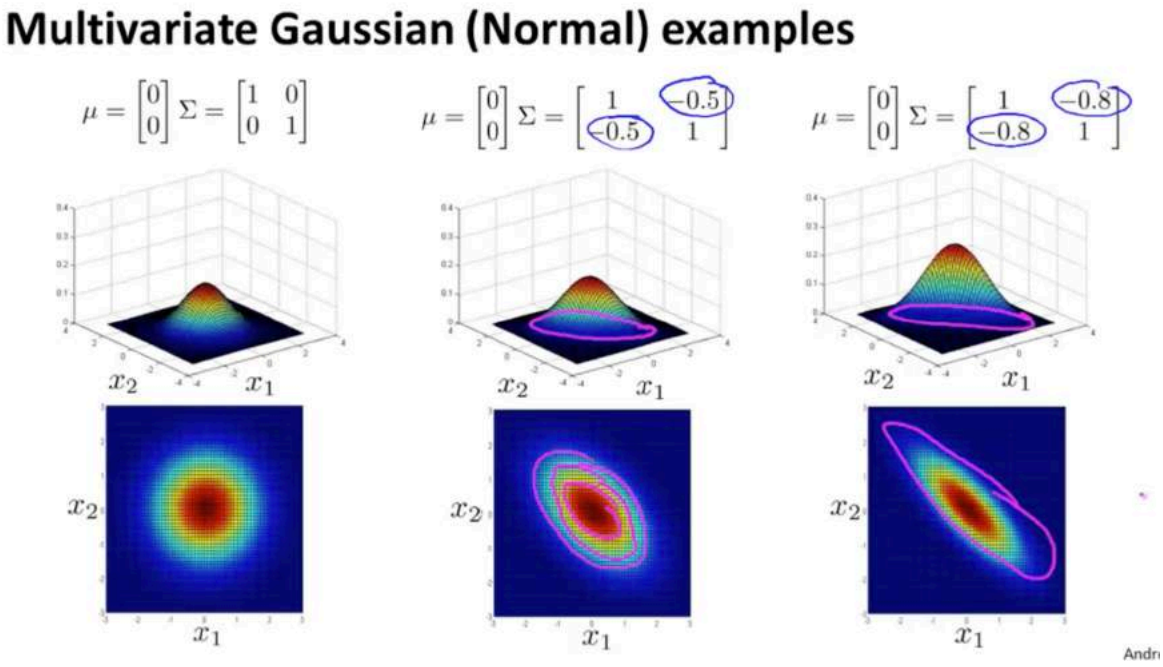


Now, one of the cool things about the multivariate Gaussian distribution is that

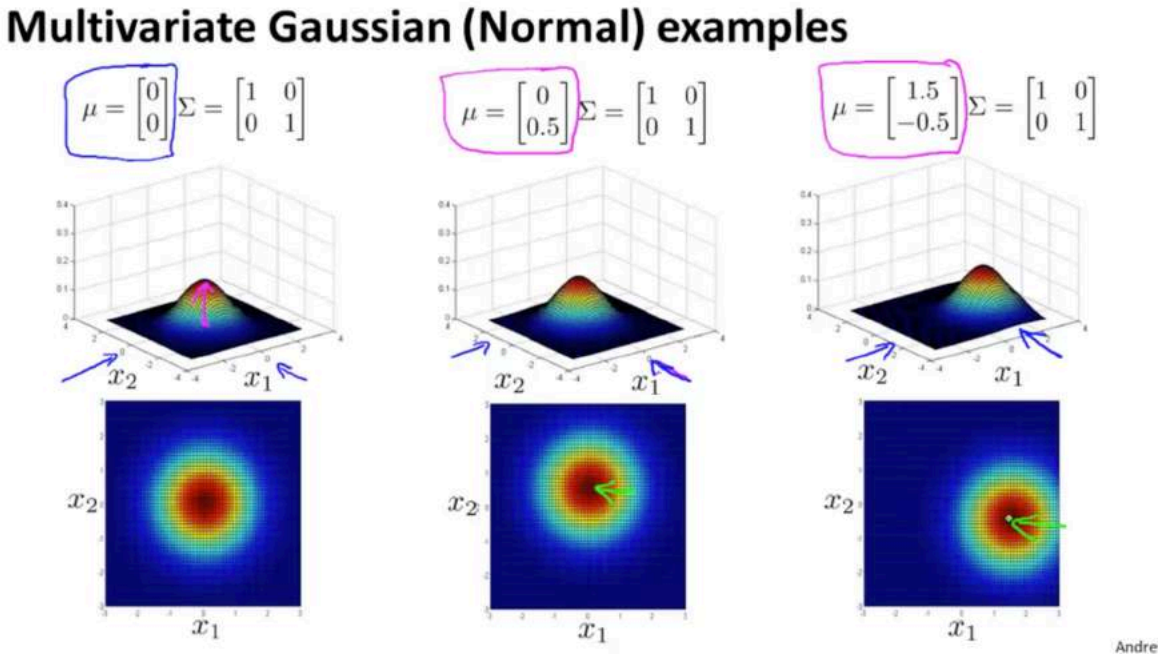
you can also use it to model correlations between the data. That is we can use it to model the fact that X1 and X2 tend to be highly correlated with each other for example. So specifically if you start to change the off diagonal entries of this covariance matrix you can get a different type of Gaussian distribution. And so as I increase the off-diagonal entries from .5 to .8, what I get is this distribution that is more and more thinly peaked along this sort of x equals y line. And so here the contour says that x and y tend to grow together and the things that are with large probability are if either X1 is large and Y2 is large or X1 is small and Y2 is small. Or somewhere in between. And as this entry, 0.8 gets large, you get a Gaussian distribution, that's sort of where all the probability lies on this sort of narrow region, where x is approximately equal to y. This is a very tall, thin distribution you know line mostly along this line central region where x is close to y. So this is if we set these entries to be positive entries.



In contrast if we set these to negative values, as I decreases it to -.5 down to -.8, then what we get is a model where we put most of the probability in this sort of negative X one in the next 2 correlation region, and so, most of the probability now lies in this region, where X 1 is about equal to -X 2, rather than X 1 equals X 2. And so this captures a sort of negative correlation between x1 and x2. And so this is a hopefully this gives you a sense of the different distributions that the multivariate Gaussian distribution can capture.



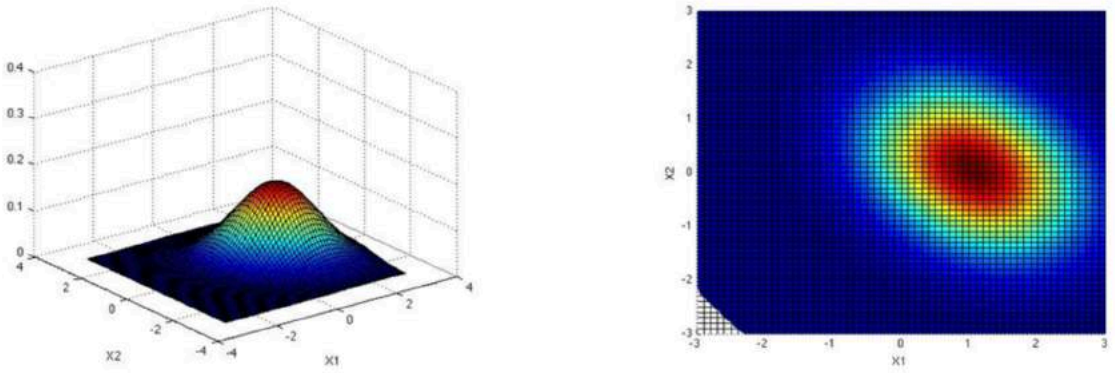
So follow up in varying, the covariance matrix sigma, the other thing you can do is also, vary the mean parameter mu, and so operationally, we have mu equal 0 0, and so the distribution was centered around X 1 equals 0, X2 equals 0, so the peak of the distribution is here, whereas, if we vary the values of mu, then that varies the peak of the distribution and so, if mu equals 0, 0.5, the peak is at, you know, X1 equals zero, and X2 equals 0.5, and so the peak or the center of this distribution has shifted, and if mu was 1.5 minus 0.5 then OK, and similarly the peak of the distribution has now shifted to a different location, corresponding to where, you know, X1 is 1.5 and X2 is -0.5, and so varying the mu parameter, just shifts around the center of this whole distribution.



Andre

Question

Consider the following multivariate Gaussian:



Which of the following are the μ and Σ for this distribution?

- $\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}$
- $\mu = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}$
- $\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & -0.3 \\ -0.3 & 1 \end{bmatrix}$
- $\mu = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & -0.3 \\ -0.3 & 1 \end{bmatrix}$

So, hopefully, looking at all these different pictures gives you a sense of the sort of probability distributions that the Multivariate Gaussian Distribution

allows you to capture. And the key advantage of it is it allows you to capture, when you'd expect two different features to be positively correlated, or maybe negatively correlated. In the next video, we'll take this multivariate Gaussian distribution and apply it to anomaly detection.

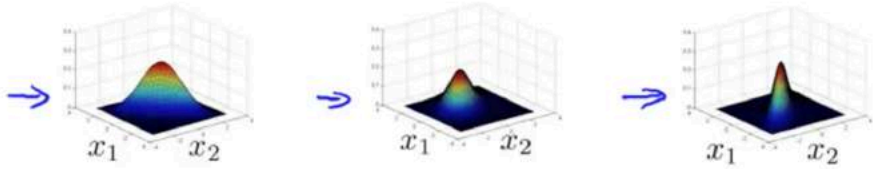
Anomaly Detection using the Multivariate Gaussian Distribution

In the last video we talked about the Multivariate Gaussian Distribution and saw some examples of the sorts of distributions you can model, as you vary the parameters, mu and sigma. In this video, let's take those ideas, and apply them to develop a different anomaly detection algorithm. To recap the multivariate Gaussian distribution and the multivariate normal distribution has two parameters, mu and sigma. Where mu this an n dimensional vector and sigma, the covariance matrix, is an n by n matrix. And here's the formula for the probability of X, as parameterized by mu and sigma, and as you vary mu and sigma, you can get a range of different distributions, like, you know, these are three examples of the ones that we saw in the previous video. So let's talk about the parameter fitting or the parameter estimation problem. The question, as usual, is if I have a set of examples X1 through XM and here each of these examples is an n dimensional vector and I think my examples come from a multivariate Gaussian distribution. How do I try to estimate my parameters mu and sigma? Well the standard formulas for estimating them is you set mu to be just the average of your training examples. And you set sigma to be equal to this. And this is actually just like the sigma that we had written out, when we were using the PCA or the Principal Components Analysis algorithm. So you just plug in these two formulas and this would give you your estimated parameter mu and your estimated parameter sigma.

Multivariate Gaussian (Normal) distribution

Parameters μ, Σ $\quad \mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$



Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $\quad x \in \mathbb{R}^n$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

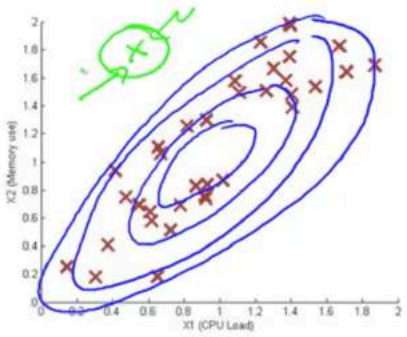
So given the data set here is how you estimate mu and sigma. Let's take this method and just plug it into an anomaly detection algorithm. So how do we put all of this together to develop an anomaly detection algorithm? Here 's what we do. First we take our training set, and we fit the model, we fit P of X, by, you know, setting mu and sigma as described on the previous slide. Next when you are given a new example X. So if you are given a test example, lets take an earlier example to have a new example out here. And that is my test example. Given the new example X, what we are going to do is compute P of X, using this formula for the multivariate Gaussian distribution. And then, if P of X is very small, then we flagged it as an anomaly, whereas, if P of X is greater than that parameter epsilon, then we don't flag it as an anomaly. So it turns out, if we were to fit a multivariate Gaussian distribution to this data set, so just the red crosses, not the green example, you end up with a Gaussian distribution that places lots of probability in the central region, slightly less probability here, slightly less probability here, slightly less probability here, and very low probability at the point that is way out here. And so, if you apply the multivariate Gaussian distribution to this example, it will actually correctly flag that example. as an anomaly.

Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$



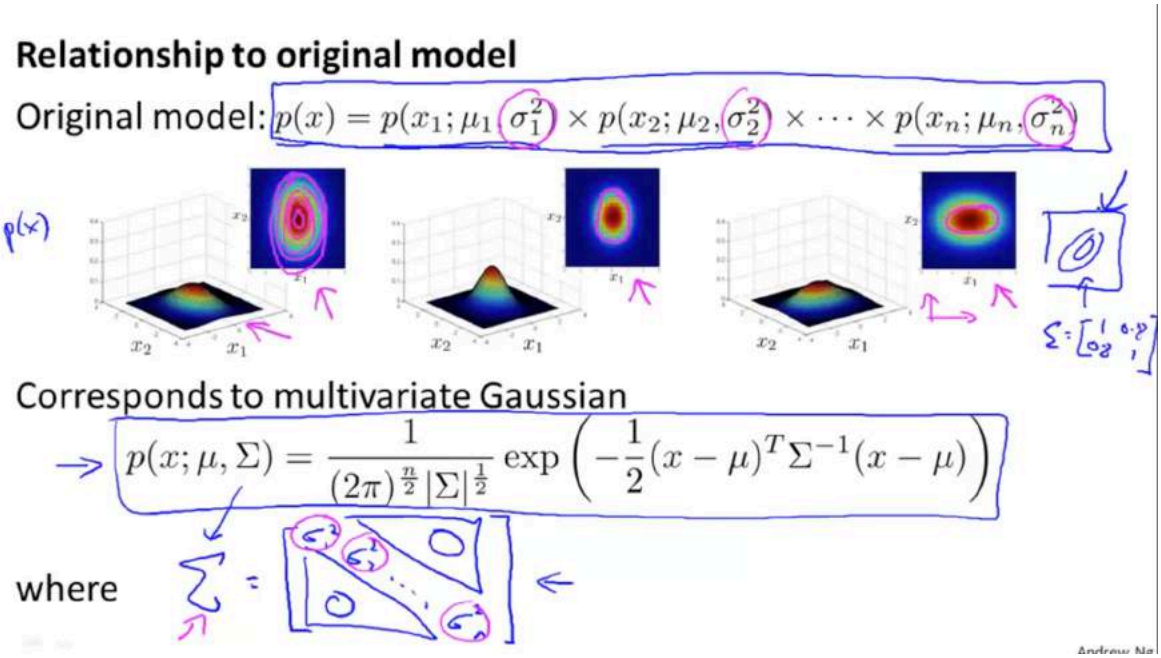
2. Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Flag an anomaly if $p(x) < \varepsilon$

Finally it's worth saying a few words about what is the relationship between the multivariate Gaussian distribution model, and the original model, where we were modeling P of X as a product of this P of X1, P of X2, up to P of Xn. It turns out that you can prove mathematically, I'm not going to do the proof here, but you can prove mathematically that this relationship, between the multivariate Gaussian model and this original one. And in particular, it turns out that the original model corresponds to multivariate Gaussians, where the contours of the Gaussian are always axis aligned. So all three of these are examples of Gaussian distributions that you can fit using the original model. It turns out that that corresponds to multivariate Gaussian, where, you know, the ellipsis here, the contours of this distribution--it turns out that this model actually corresponds to a special case of a multivariate Gaussian distribution. And in particular, this special case is defined by constraining the distribution of p of x, the multivariate a Gaussian distribution of p of x, so that the contours of the probability density function, of the probability distribution function, are axis aligned. And so you can get a p of x with a multivariate Gaussian that looks like this, or like this, or like this. And you notice, that in all 3 of these examples, these ellipses, or these ovals that I'm drawing, have their axes aligned with the X1 X2 axes. And what we do not have, is a set of contours that are at an angle, right? And this corresponded to examples where sigma is equal to 1 1, 0.8, 0.8. Let's say, with non-0 elements on the off diagonals. So, it turns out that it's possible to show mathematically that this model actually is the same as a multivariate Gaussian distribution but with a constraint. And the constraint is that the covariance matrix sigma must have 0's on the off diagonal elements. In particular, the covariance matrix sigma, this thing here, it would be sigma squared 1, sigma squared 2, down to sigma squared n, and then everything on

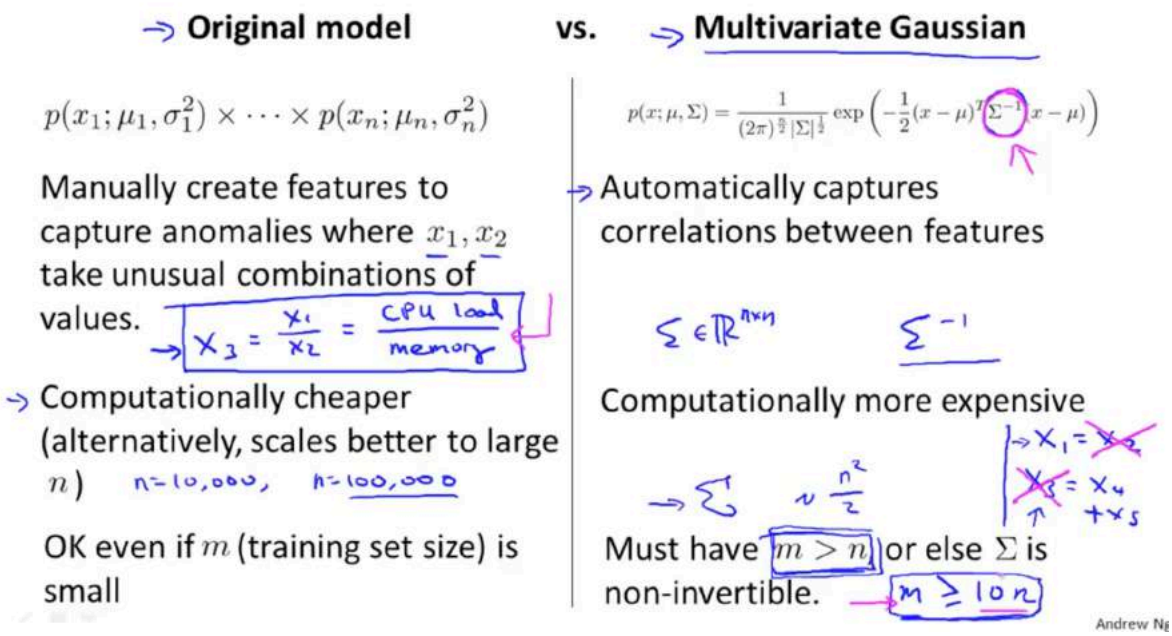
the off diagonal entries, all of these elements above and below the diagonal of the matrix, all of those are going to be zero. And in fact if you take these values of sigma, sigma squared 1, sigma squared 2, down to sigma squared n, and plug them into here, and you know, plug them into this covariance matrix, then the two models are actually identical. That is, this new model, using a multivariate Gaussian distribution, corresponds exactly to the old model, if the covariance matrix sigma, has only 0 elements off the diagonals, and in pictures that corresponds to having Gaussian distributions, where the contours of this distribution function are axis aligned. So you aren't allowed to model the correlations between the different features. So in that sense the original model is actually a special case of this multivariate Gaussian model.



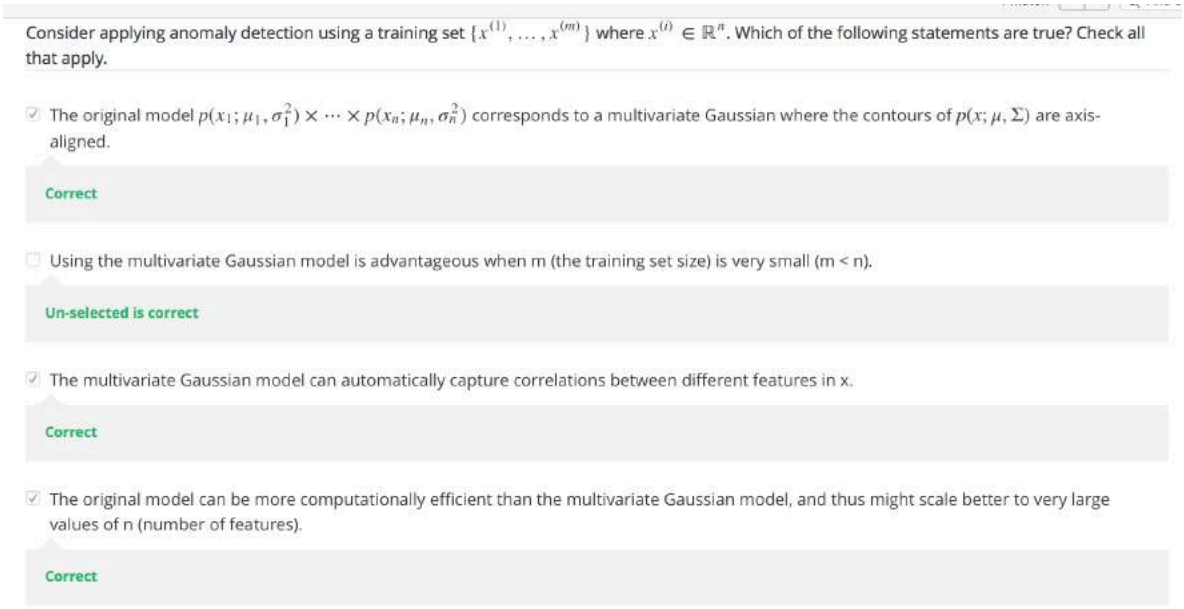
So when would you use each of these two models? So when would you the original model and when would you use the multivariate Gaussian model? The original model is probably used somewhat more often, and whereas the multivariate Gaussian distribution is used somewhat less but it has the advantage of being able to capture correlations between features. So suppose you want to capture anomalies where you have different features say where features x1, x2 take on unusual combinations of values so in the earlier example, we had that example where the anomaly was with the CPU load and the memory use taking on unusual combinations of values, if you want to use the original model to capture that, then what you need to do is create an extra feature, such as X3 equals X1/X2, you know equals maybe the CPU load divided by the memory used, or something, and you need to create extra features if there's unusual combinations of values where X1 and X2 take on an unusual combination of values even though X1 by itself and X2 by itself looks like it's taking a perfectly normal value. But if you're willing to spend the time to manually create an extra feature like this, then the original model will work fine.

Whereas in contrast, the multivariate Gaussian model can automatically capture correlations between different features. But the original model has some other more significant advantages, too, and one huge advantage of the original model is that it is computationally cheaper, and another view on this is that is scales better to very large values of n and very large numbers of features, and so even if n were ten thousand, or even if n were equal to a hundred thousand, the original model will usually work just fine. Whereas in contrast for the multivariate Gaussian model notice here, for example, that we need to compute the inverse of the matrix sigma where sigma is an n by n matrix and so computing sigma if sigma is a hundred thousand by a hundred thousand matrix that is going to be very computationally expensive. And so the multivariate Gaussian model scales less well to large values of N. And finally for the original model, it turns out to work out ok even if you have a relatively small training set this is the small unlabeled examples that we use to model p of x of course, and this works fine, even if M is, you know, maybe 50, 100, works fine. Whereas for the multivariate Gaussian, it is sort of a mathematical property of the algorithm that you must have m greater than n, so that the number of examples is greater than the number of features you have. And there's a mathematical property of the way we estimate the parameters that if this is not true, so if m is less than or equal to n, then this matrix isn't even invertible, that is this matrix is singular, and so you can't even use the multivariate Gaussian model unless you make some changes to it. But a typical rule of thumb that I use is, I will use the multivariate Gaussian model only if m is much greater than n, so this is sort of the narrow mathematical requirement, but in practice, I would use the multivariate Gaussian model, only if m were quite a bit bigger than n. So if m were greater than or equal to 10 times n, let's say, might be a reasonable rule of thumb, and if it doesn't satisfy this, then the multivariate Gaussian model has a lot of parameters, right, so this covariance matrix sigma is an n by n matrix, so it has, you know, roughly n squared parameters, because it's a symmetric matrix, it's actually closer to n squared over 2 parameters, but this is a lot of parameters, so you need make sure you have a fairly large value for m, make sure you have enough data to fit all these parameters. And m greater than or equal to 10 n would be a reasonable rule of thumb to make sure that you can estimate this covariance matrix sigma reasonably well. So in practice the original model shown on the left that is used more often. And if you suspect that you need to capture correlations between features what people will often do is just manually design extra features like these to capture specific unusual combinations of values. But in problems where you have a very large training set or m is very large and n is not too large, then the multivariate Gaussian model is well worth considering and may work better as well, and can save you from having to spend your time to manually create extra features in case the anomalies turn out to be captured by unusual combinations of values of the features. Finally I just want to briefly mention one somewhat technical property, but if you're fitting multivariate Gaussian model, and if you find that the covariance matrix sigma is singular, or you find it's non-invertible, they're usually 2 cases for this. One is if it's failing

to satisfy this m greater than n condition, and the second case is if you have redundant features. So by redundant features, I mean, if you have 2 features that are the same. Somehow you accidentally made two copies of the feature, so your x1 is just equal to x2. Or if you have redundant features like maybe your features X3 is equal to feature X4, plus feature X5. Okay, so if you have highly redundant features like these, you know, where if X3 is equal to X4 plus X5, well X3 doesn't contain any extra information, right? You just take these 2 other features, and add them together. And if you have this sort of redundant features, duplicated features, or this sort of features, than sigma may be non-invertible. And so there's a debugging set-- this should very rarely happen, so you probably won't run into this, it is very unlikely that you have to worry about this-- but in case you implement a multivariate Gaussian model you find that sigma is non-invertible. What I would do is first make sure that M is quite a bit bigger than N, and if it is then, the second thing I do, is just check for redundant features. And so if there are 2 features that are equal, just get rid of one of them, or if you have redundant if these , X3 equals X4 plus X5, just get rid of the redundant feature, and then it should work fine again. As an aside for those of you who are experts in linear algebra, by redundant features, what I mean is the formal term is features that are linearly dependent. But in practice what that really means is one of these problems tripping up the algorithm if you just make you features non-redundant., that should solve the problem of sigma being non-invertable. But once again the odds of your running into this at all are pretty low so chances are, you can just apply the multivariate Gaussian model, without having to worry about sigma being non-invertible, so long as m is greater than or equal to n.



Question



So that's it for anomaly detection, with the multivariate Gaussian distribution. And if you apply this method you would be able to have an anomaly detection algorithm that automatically captures positive and negative correlations between your different features and flags an anomaly if it sees is unusual combination of the values of the features.

Review

Quiz

1. For which of the following problems would anomaly detection be a suitable algorithm?

- [x] In a computer chip fabrication plant, identify microchips that might be defective.
- [x] From a large set of primary care patient records, identify individuals who might have unusual health conditions.
- [ ] Given data from credit card transactions, classify each transaction according to type of purchase (for example: food, transportation, clothing).
- [ ] From a large set of hospital patient records, predict which patients have a particular disease (say, the flu).

2. Suppose you have trained an anomaly detection system for fraud detection, and your system that flags anomalies when $p(x)$ is less than ε, and you find on the cross-validation set that it is missing many fradulent transactions (i.e., failing to flag them as anomalies). What should you do?

- ( ) Decrease ε
- (•) Increase ε

3. Suppose you are developing an anomaly detection system to catch manufacturing defects in airplane engines. You model uses

$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$.

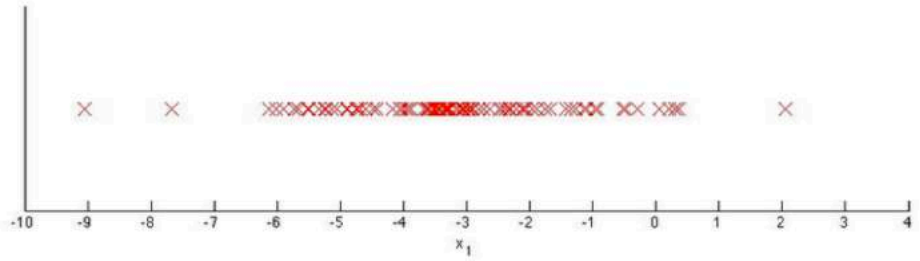
You have two features x_1 = vibration intensity, and x_2 = heat generated. Both x_1 and x_2 take on values between 0 and 1 (and are strictly greater than 0), and for most "normal" engines you expect that $x_1 \approx x_2$. One of the suspected anomalies is that a flawed engine may vibrate very intensely even without generating much heat (large x_1, small x_2), even though the particular values of x_1 and x_2 may not fall outside their typical ranges of values. What additional feature x_3 should you create to capture these types of anomalies:

- $x_3 = x_1 \times x_2^2$
- $x_3 = (x_1 + x_2)^2$
- $x_3 = \frac{x_1}{x_2}$
- $x_3 = x_1^2 \times x_2^2$

4. Which of the following are true? Check all that apply.

- [ ] If you have a large labeled training set with many positive examples and many negative examples, the anomaly detection algorithm will likely perform just as well as a supervised learning algorithm such as an SVM.
- [ ] If you are developing an anomaly detection system, there is no way to make use of labeled data to improve your system.
- [x] When choosing features for an anomaly detection system, it is a good idea to look for features that take on unusually large or small values for (mainly the) anomalous examples.
- [x] If you do not have any labeled data (or if all your data has label $y = 0$), then is is still possible to learn $p(x)$, but it may be harder to evaluate the system or choose a good value of ϵ.

5. You have a 1-D dataset $\{x^{(1)}, \ldots, x^{(m)}\}$ and you want to detect outliers in the dataset. You first plot the dataset and it looks like this:



Suppose you fit the gaussian distribution parameters μ_1 and σ_1^2 to this dataset. Which of the following values for μ_1 and σ_1^2 might you get?

- (●) $\mu_1 = -3, \sigma_1^2 = 4$
- ( ) $\mu_1 = -6, \sigma_1^2 = 4$
- ( ) $\mu_1 = -3, \sigma_1^2 = 2$
- ( ) $\mu_1 = -6, \sigma_1^2 = 2$

Recommender Systems

Predicting Movie

Ratings

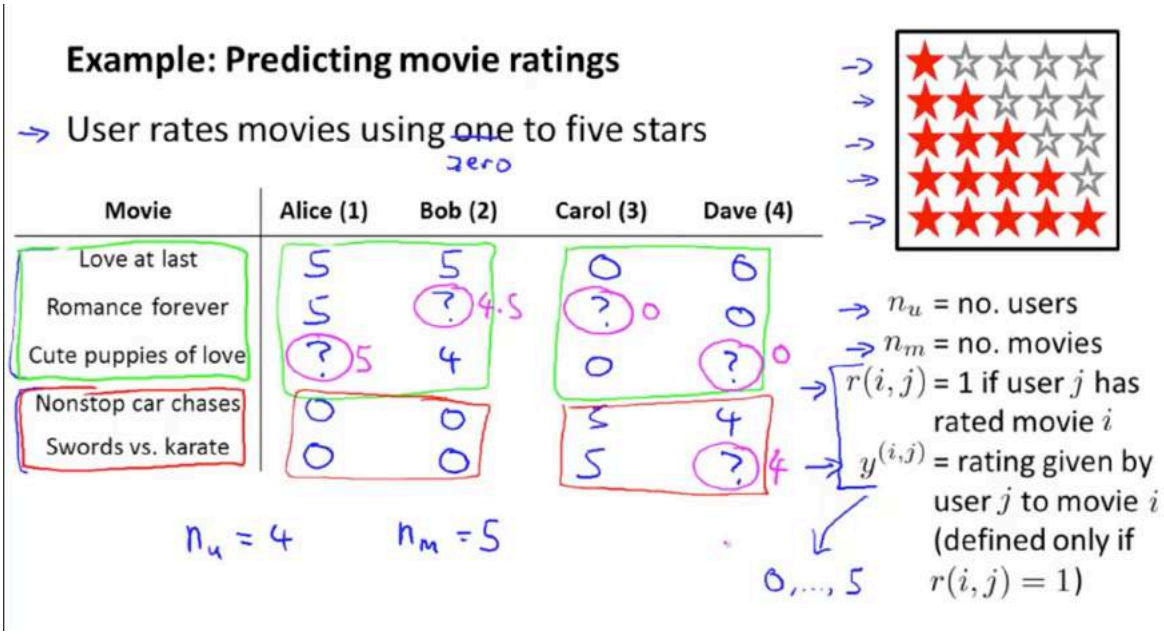
Problem Formulation

In this next set of videos, I would like to tell you about recommender systems. There are two reasons, I had two motivations for why I wanted to talk about recommender systems. The first is just that it is an important application of machine learning. Over the last few years, occasionally I visit different, you know, technology companies here in Silicon Valley and I often talk to people working on machine learning applications there and so I've asked people what are the most important applications of machine learning or what are the machine learning applications that you would most like to get an improvement in the performance of. And one of the most frequent answers I heard was that there are many groups out in Silicon Valley now, trying to build better recommender systems. So, if you think about what the websites are like Amazon, or what Netflix or what eBay, or what iTunes Genius, made by Apple does, there are many websites or systems that try to recommend new products to use. So, Amazon recommends new books to you, Netflix try to recommend new movies to you, and so on. And these sorts of recommender systems, that look at what books you may have purchased in the past, or what movies you have rated in the past, but these are the systems that are responsible for today, a substantial fraction of Amazon's revenue and for a company like Netflix, the recommendations that they make to the users is also responsible for a substantial fraction of the movies watched by their users. And so an improvement in performance of a recommender system can have a substantial and immediate impact on the bottom line of many of these companies. Recommender systems is kind of a funny problem, within academic machine learning so that we could go to an academic machine learning conference, the problem of recommender systems, actually receives relatively little attention, or at least it's sort of a smaller fraction of what goes on within Academia. But if you look at what's happening, many technology companies, the ability to build these systems seems to be a high priority for many companies. And that's one of the reasons why I want to talk about them in this class. The second reason that I want to talk about recommender systems is that as we approach the last few sets of videos of this class I wanted to talk about a few of the big ideas in machine learning and share with you, you know, some of the big ideas in machine learning. And we've already seen in this class that features are important for machine learning, the features you choose will have a big effect on the performance of your learning algorithm. So there's this big idea in machine learning, which is that for some problems, maybe not all problems,

but some problems, there are algorithms that can try to automatically learn a good set of features for you. So rather than trying to hand design, or hand code the features, which is mostly what we've been doing so far, there are a few settings where you might be able to have an algorithm, just to learn what feature to use, and the recommender systems is just one example of that sort of setting. There are many others, but engraved through recommender systems, will be able to go a little bit into this idea of learning the features and you'll be able to see at least one example of this, I think, big idea in machine learning as well. So, without further ado, let's get started, and talk about the recommender system problem formulation.

As my running example, I'm going to use the modern problem of predicting movie ratings. So, here's a problem. Imagine that you're a website or a company that sells or rents out movies, or what have you. And so, you know, Amazon, and Netflix, and I think iTunes are all examples of companies that do this, and let's say you let your users rate different movies, using a 1 to 5 star rating. So, users may, you know, something one, two, three, four or five stars. In order to make this example just a little bit nicer, I'm going to allow 0 to 5 stars as well, because that just makes some of the math come out just nicer. Although most of these websites use the 1 to 5 star scale. So here, I have 5 movies. You know, Love That Lasts, Romance Forever, Cute Puppies of Love, Nonstop Car Chases, and Swords vs. Karate. And we have 4 users, which, calling, you know, Alice, Bob, Carol, and Dave, with initials A, B, C, and D, we'll call them users 1, 2, 3, and 4. So, let's say Alice really likes Love That Lasts and rates that 5 stars, likes Romance Forever, rates it 5 stars. She did not watch Cute Puppies of Love, and did rate it, so we don't have a rating for that, and Alice really did not like Nonstop Car Chases or Swords vs. Karate. And a different user Bob, user two, maybe rated a different set of movies, maybe she likes to Love at Last, did not to watch Romance Forever, just have a rating of 4, a 0, a 0, and maybe our 3rd user, rates this 0, did not watch that one, 0, 5, 5, and, you know, let's just fill in some of the numbers. And so just to introduce a bit of notation, this notation that we'll be using throughout, I'm going to use NU to denote the number of users. So in this example, NU will be equal to 4. So the u-subscript stands for users and Nm, going to use to denote the number of movies, so here I have five movies so Nm equals equals 5. And you know for this example, I have for this example, I have loosely 3 maybe romantic or romantic comedy movies and 2 action movies and you know, if you look at this small example, it looks like Alice and Bob are giving high ratings to these romantic comedies or movies about love, and giving very low ratings about the action movies, and for Carol and Dave, it's the opposite, right? Carol and Dave, users three and four, really like the action movies and give them high ratings, but don't like the romance and love- type movies as much. Specifically, in the recommender system problem, we are given the following data. Our data comprises the following: we have these values r(i, j), and r(i, j) is 1 if user J has

rated movie I. So our users rate only some of the movies, and so, you know, we don't have ratings for those movies. And whenever r(i, j) is equal to 1, whenever user j has rated movie i, we also get this number y(i, j), which is the rating given by user j to movie i. And so, y(i, j) would be a number from zero to five, depending on the star rating, zero to five stars that user gave that particular movie. So, the recommender system problem is given this data that has give these r(i, j)'s and the y(i, j)'s to look through the data and look at all the movie ratings that are missing and to try to predict what these values of the question marks should be. In the particular example, I have a very small number of movies and a very small number of users and so most users have rated most movies but in the realistic settings your users each of your users may have rated only a minuscule fraction of your movies but looking at this data, you know, if Alice and Bob both like the romantic movies maybe we think that Alice would have given this a five. Maybe we think Bob would have given this a 4.5 or some high value, as we think maybe Carol and Dave were doing these very low ratings. And Dave, well, if Dave really likes action movies, maybe he would have given Swords and Karate a 4 rating or maybe a 5 rating, okay? And so, our job in developing a recommender system is to come up with a learning algorithm that can automatically go fill in these missing values for us so that we can look at, say, the movies that the user has not yet watched, and recommend new movies to that user to watch. You try to predict what else might be interesting to a user.



Question

In our notation, $r(i,j) = 1$ if user j has rated movie i, and $y^{(i,j)}$ is his rating on that movie.
Consider the following example (no. of movies $n_m = 2$, no. of users $n_u = 3$):

.	User 1	User 2	User 3
Movie 1	0	1	?
Movie 2	?	5	5

What is $r(2, 1)$? How about $y^{(2,1)}$?

- ○ $r(2, 1) = 0,\ y^{(2,1)} = 1$
- ○ $r(2, 1) = 1,\ y^{(2,1)} = 1$
- ● $r(2, 1) = 0,\ y^{(2,1)} = \text{undefined}$
- ○ $r(2, 1) = 1,\ y^{(2,1)} = \text{undefined}$

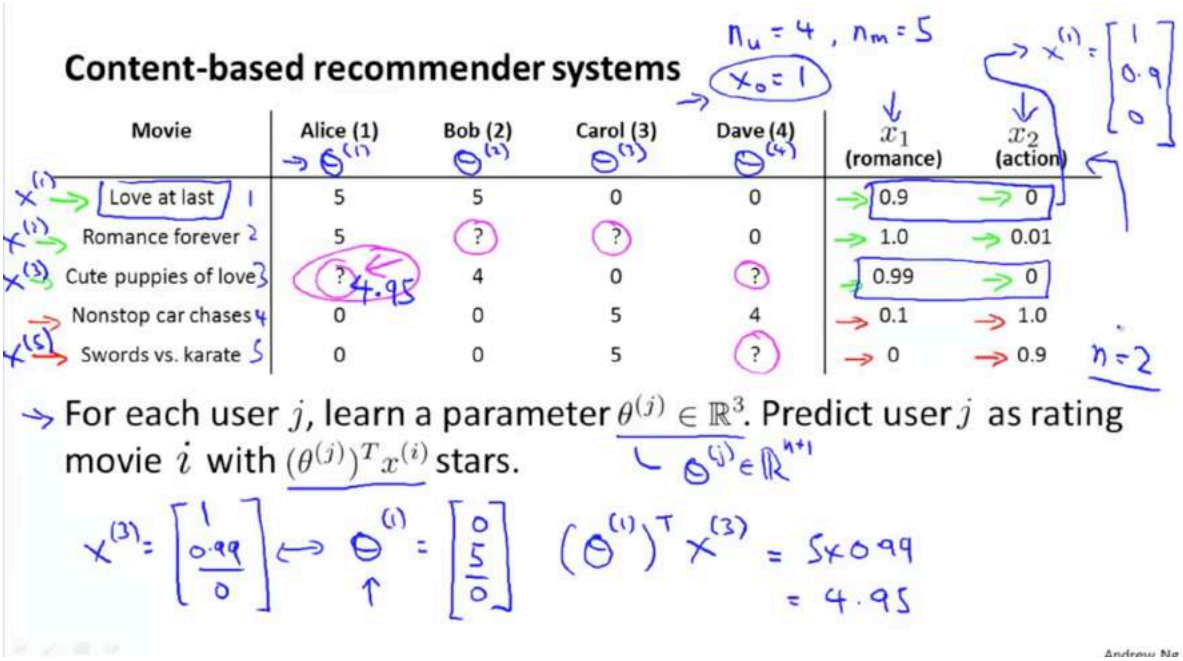
So that's the formalism of the recommender system problem. In the next video we'll start to develop a learning algorithm to address this problem.

Content Based Recommendations

In the last video, we talked about the recommender systems problem where for example you might have a set of movies and you may have a set of users, each who have rated some subset of the movies. They've rated the movies one to five stars or zero to five stars. And what we would like to do is look at these users and predict how they would have rated other movies that they have not yet rated. In this video I'd like to talk about our first approach to building a recommender system. This approach is called content based recommendations. Here's our data set from before and just to remind you of a bit of notation, I was using nu to denote the number of users and so that's equal to 4, and nm to denote the number of movies, I have 5 movies. So, how do I predict what these missing values would be? Let's suppose that for each of these movies I have a set of features for them. In particular, let's say that for each of the movies have two features which I'm going to denote x1 and x2. Where x1 measures the degree to which a movie is a romantic movie and x2 measures the degree to which a movie is an action movie. So, if you take a

movie, Love at last, you know it's 0.9 rating on the romance scale. This is a highly romantic movie, but zero on the action scale. So, almost no action in that movie. Romance forever is a 1.0, lot of romance and 0.01 action. I don't know, maybe there's a minor car crash in that movie or something. So there's a little bit of action. Skipping one, let's do Swords vs karate, maybe that has a 0 romance rating and no romance at all in that but plenty of action. And Nonstop car chases, maybe again there's a tiny bit of romance in that movie but mainly action. And Cute puppies of love mainly a romance movie with no action at all. So if we have features like these, then each movie can be represented with a feature vector. Let's take movie one. So let's call these movies 1, 2, 3, 4, and 5. But my first movie, Love at last, I have my two features, 0.9 and 0. And so these are features x1 and x2. And let's add an extra feature as usual, which is my interceptor feature x0 = 1. And so putting these together I would then have a feature x1. The superscript 1 denotes it's the feature vector for my first movie, and this feature vector is equal to 1. The first 1 there is this interceptor. And then my two feature is 0.90 like so. So for Love at last I would have a feature vector x1, for the movie Romance forever I may have a software feature of vector x2, and so on, and for Swords vs karate I would have a different feature vector x superscript 5. Also, consistence with our earlier node notation that we were using, we're going to set n to be the number of features not counting this x0 interceptor. So n is equal to 2 because it's we have two features x1 and x2 capturing the degree of romance and the degree of action in each movie. Now in order to make predictions here's one thing that we do which is that we could treat predicting the ratings of each user as a separate linear regression problem. So specifically, let's say that for each user j, we're going to learn the parameter vector theta j, which would be an R3 in this case. More generally, theta (j) would be an R (n+1), where n is the number of features not counting the set term. And we're going to predict user j as rating movie i with just the inner product between parameters vectors theta and the features xi. So let's take a specific example. Let's take user 1, so that would be Alice. And associated with Alice would be some parameter vector theta 1. And our second user, Bob, will be associated a different parameter vector theta 2. Carol will be associated with a different parameter vector theta 3 and Dave a different parameter vector theta 4. So let's say you want to make a prediction for what Alice will think of the movie Cute puppies of love. Well that movie is going to have some parameter vector x3 where we have that x3 is going to be equal to 1, which is my intercept term and then 0.99 and then 0. And let's say, for this example, let's say that we've somehow already gotten a parameter vector theta 1 for Alice. We'll say it later exactly how we come up with this parameter vector. But let's just say for now that some unspecified learning algorithm has learned the parameter vector theta 1 and is equal to this 0,5,0. So our prediction for this entry is going to be equal to theta 1, that is Alice's parameter vector, transpose x3, that is the feature vector for the Cute puppies of love movie, number 3. And so the inner product between these two vectors is gonna be 5 times 0.99, which is equal to 4.95. And so my prediction for this value over here is going to be 4.95. And maybe that seems like a reasonable

value if indeed this is my parameter vector theta 1. So, all we're doing here is we're applying a different copy of this linear regression for each user, and we're saying that what Alice does is Alice has some parameter vector theta 1 that she uses, that we use to predict her ratings as a function of how romantic and how action packed a movie is. And Bob and Carol and Dave, each of them have a different linear function of the romanticness and actionness, or degree of romance and degree of action in a movie and that that's how we're gonna predict that their star ratings.



Question

Consider the following set of movie ratings:

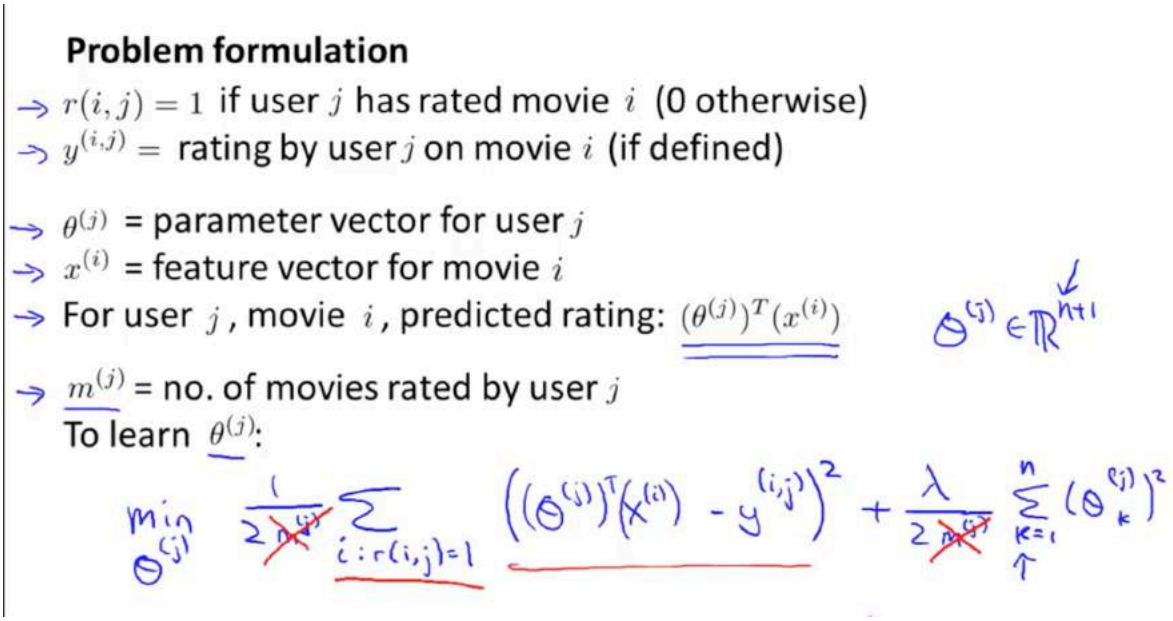
Movie	Alice (1)	Bob (2)	Carol (3)	David (4)	(romance)	(action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

Which of the following is a reasonable value for $\theta^{(3)}$? Recall that $x_0 = 1$.

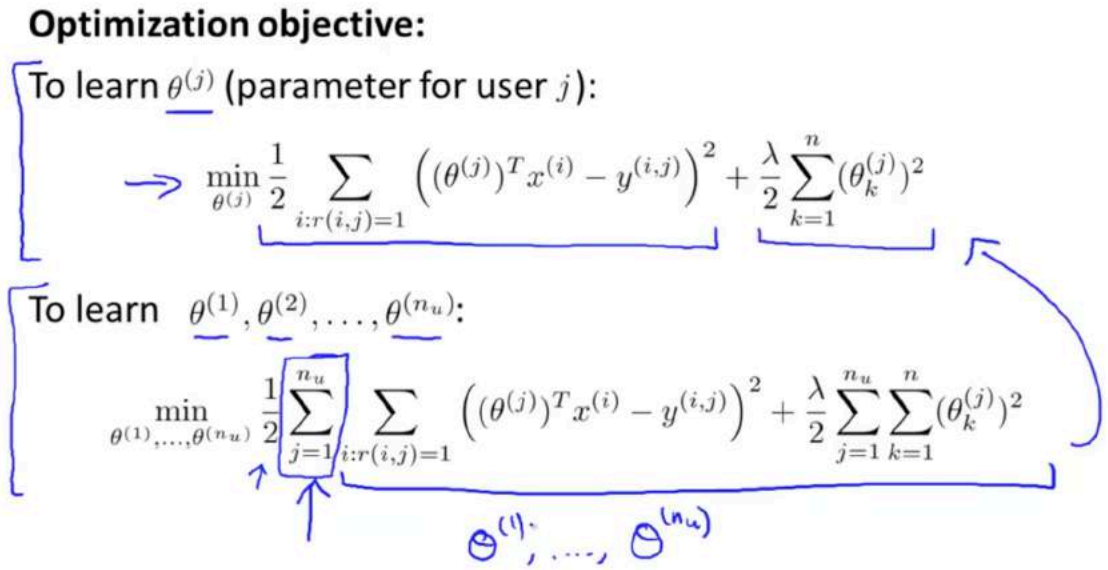
- $\theta^{(3)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$
- $\theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
- $\theta^{(3)} = \begin{bmatrix} 1 \\ 0 \\ 4 \end{bmatrix}$
- $\theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

More formally, here's how we can write down the problem. Our notation is that r(i,j) is equal to 1 if user j has rated movie i and y(i,j) is the rating of that movie, if that rating exists. That is, if that user has actually rated that movie. And, on the previous slide we also defined these, theta j, which is a parameter for the user xi, which is a feature vector for a specific movie. And for each user and each movie, we predict that rating as follows. So let me introduce just temporarily introduce one extra bit of notation mj. We're gonna use mj to denote the number of users rated by movie j. We don't need this notation only for this line. Now in order to learn the parameter vector for theta j, well how do we do so. This is basically a linear regression problem. So what we can do is just choose a parameter vector theta j so that the predicted values here are as close as possible to the values that we observed in our training sets and the values we observed in our data. So let's write that down. In order to learn the parameter vector theta j, let's minimize over the parameter vector theta j of sum, and I want to sum over all movies that user j has rated. So we write it as sum over all values of i. That's a :r(i,j) equals 1. So the way to read this summation syntax is this is summation over all the values of i, so the r(i.j) is equal to 1. So you'll be summing over all the movies that user j has rated. And then I'm going to compute theta j, transpose x i. So that's the prediction of using j's rating on movie i,- y (i,j). So that's the actual observed rating squared. And then, let me just divide by the number of movies that user j has actually rated. So let's just divide by 1 over 2m j. And so this is just like the least squares regressions. It's just like linear regression, where we want to choose the parameter vector theta j to minimize this type of squared error term. And if you want, you can also add in irregularization terms so plus lambda over 2m and this is really 2mj because we have mj examples. User j has rated that many movies, it's not like we have that many data points with which to fit the parameters of theta j. And then let me add in my usual regularization term here of theta j k squared. As usual, this sum is from k equals 1 through n, so here, theta j is going to be an n plus 1 dimensional vector, where in our early example n was equal to 2. But more broadly, more generally n is the number of features

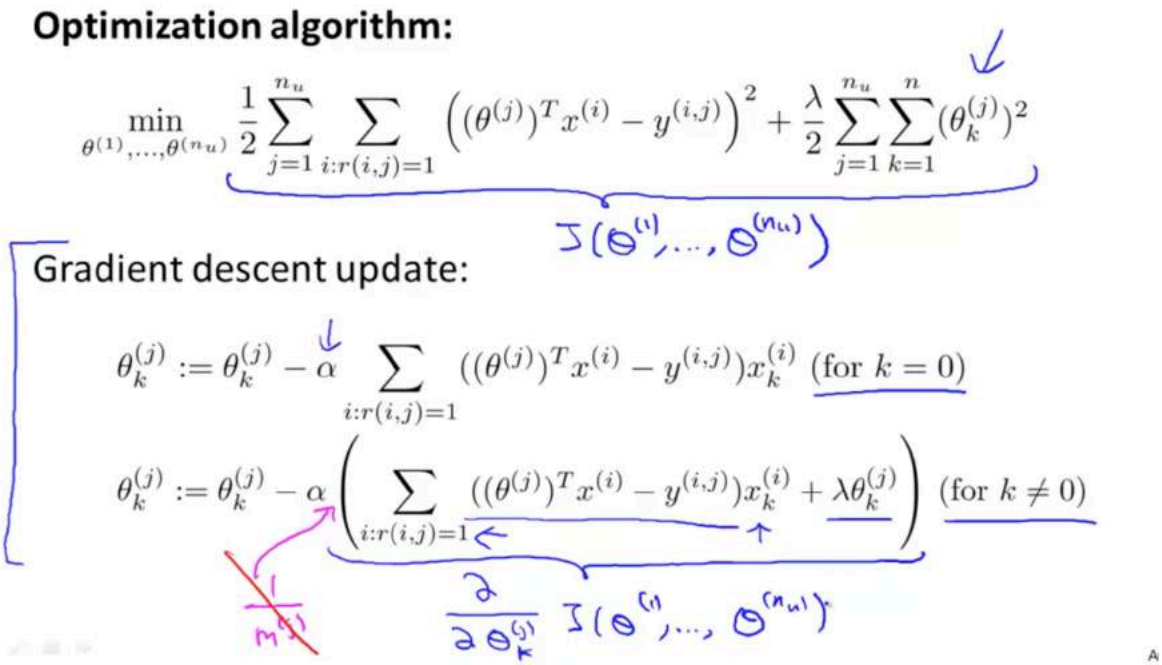
we have per movie. And so as usual we don't regularize over theta 0. We don't regularize over the bias terms. The sum is from k equals 1 through n. So if you minimize this as a function of theta j you get a good solution, you get a pretty good estimate of a parameter vector theta j with which to make predictions for user j's movie ratings. For recommender systems, I'm gonna change this notation a little bit. So to simplify the subsequent math, I with to get rid of this term mj. So that's just a constant, right? So I can delete it without changing the value of theta j that I get out of this optimization. So if you imagine taking this whole equation, taking this whole expression and multiplying it by mj, get rid of that constant. And when I minimize this, I should still get the same value of theta j as before.



So just to repeat what we wrote on the previous slide, here's our optimization objective. In order to learn theta j which is the parameter for user j, we're going to minimize over theta j of this optimization objectives. So this is our usual squared error term and then this is our regularizations term. Now of course in building a recommender system, we don't just want to learn parameters for a single user. We want to learn parameters for all of our users. I have n subscript u users, so I want to learn all of these parameters. And so, what I'm going to do is take this optimization objective and just add the mixture summation there. So this expression here with the one half on top of this is exactly the same as what we had on top. Except that now instead of just doing this for a specific user theta j, I'm going to sum my objective over all of my users and then minimize this overall optimization objective, minimize this overall cost on. And when I minimize this as a function of theta 1, theta 2, up to theta nu, I will get a separate parameter vector for each user. And I can then use that to make predictions for all of my users, for all of my n subscript users.



So putting everything together, this was our optimization objective on top. And to give this thing a name, I'll just call this J(theta1, ..., theta nu). So j as usual is my optimization objective, which I'm trying to minimize. Next, in order to actually do the minimization, if you were to derive the gradient descent update, these are the equations that you would get. So you take theta j, k, and subtract from an alpha, which is the learning rate, times these terms over here on the right. So there's slightly different cases when k equals 0 and when k does not equal 0. Because our regularization term here regularizes only the values of theta jk for k not equal to 0, so we don't regularize theta 0, so with slightly different updates when k equals 0 and k is not equal to 0. And this term over here, for example, is just the partial derivative with respect to your parameter, that of your optimization objective. Right and so this is just gradient descent and I've already computed the derivatives and plugged them into here. And if this gradient descent update look a lot like what we have here for linear regression. That's because these are essentially the same as linear regression. The only minor difference is that for linear regression we have these 1 over m terms, this really would've been 1 over mj. But because earlier when we are deriving the optimization objective, we got rid of this, that's why we don't have this 1 over m term. But otherwise, it's really some of my training examples of the ever times xk plus that regularization term, plus that term of regularization contributes to the derivative. And so if you're using gradient descent here's how you can minimize the cost function j to learn all the parameters. And using these formulas for the derivative if you want, you can also plug them into a more advanced optimization algorithm, like conjugate gradient or LBFGS or what have you. And use that to try to minimize the cost function j as well.



So hopefully you now know how you can apply essentially a deviation on linear regression in order to predict different movie ratings by different users. This particular algorithm is called a content based recommendations, or a content based approach, because we assume that we have available to us features for the different movies. And so where features that capture what is the content of these movies, of how romantic is this movie, how much action is in this movie. And we're really using features of a content of the movies to make our predictions. But for many movies, we don't actually have such features. Or maybe very difficult to get such features for all of our movies, for all of whatever items we're trying to sell. And so, in the next video, we'll start to talk about an approach to recommender systems that isn't content based and does not assume that we have someone else giving us all of these features for all of the movies in our data set.

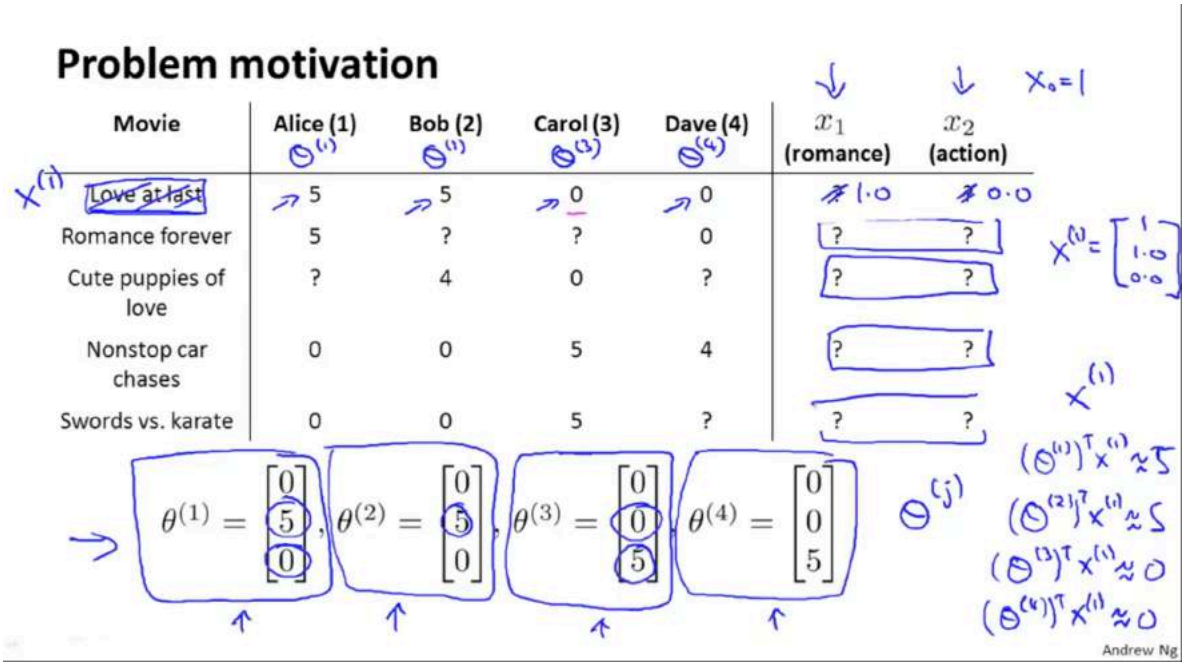
Collaborative Filtering

Collaborative Filtering

In this video we'll talk about an approach to building a recommender system that's called collaborative filtering. The algorithm that we're talking about has a very interesting property that it does what is called feature learning and by that I mean that this will be an algorithm that can start to learn for itself what features to use.

Here was the data set that we had and we had assumed that for each movie, someone had come and told us how romantic that movie was and how much action there was in that movie. But as you can imagine it can be very difficult and time consuming and expensive to actually try to get someone to, you know, watch each movie and tell you how romantic each movie and how action packed is each movie, and often you'll want even more features than just these two. So where do you get these features from? So let's change the problem a bit and suppose that we have a data set where we do not know the values of these features. So we're given the data set of movies and of how the users rated them, but we have no idea how romantic each movie is and we have no idea how action packed each movie is so I've replaced all of these things with question marks. But now let's make a slightly different assumption. Let's say we've gone to each of our users, and each of our users has told has told us how much they like the romantic movies and how much they like action packed movies. So Alice has associated a current of theta 1. Bob theta 2. Carol theta 3. Dave theta 4. And let's say we also use this and that Alice tells us that she really likes romantic movies and so there's a five there which is the multiplier associated with X1 and lets say that Alice tells us she really doesn't like action movies and so there's a 0 there. And Bob tells us something similar so we have theta 2 over here. Whereas Carol tells us that she really likes action movies which is why there's a 5 there, that's the multiplier associated with X2, and remember there's also X0 equals 1 and let's say that Carol tells us she doesn't like romantic movies and so on, similarly for Dave. So let's assume that somehow we can go to users and each user J just tells us what is the value of theta J for them. And so basically specifies to us of how much they like different types of movies. If we can get these parameters theta from our users then it turns out that it becomes possible to try to infer what are the values of x1 and x2 for each movie. Let's look at an example. Let's look at movie 1. So that movie 1 has associated with it a feature vector x1. And you know this movie is called Love at last but let's ignore that. Let's pretend we don't know what this movie is about, so let's ignore the title of this movie. All we know is that Alice loved this move. Bob loved this movie. Carol and Dave hated this movie. So what can we infer? Well, we know from the feature vectors that Alice and Bob love romantic movies because they told us that there's a 5 here.

Whereas Carol and Dave, we know that they hate romantic movies and that they love action movies. So because those are the parameter vectors that you know, uses 3 and 4, Carol and Dave, gave us. And so based on the fact that movie 1 is loved by Alice and Bob and hated by Carol and Dave, we might reasonably conclude that this is probably a romantic movie, it is probably not much of an action movie. this example is a little bit mathematically simplified but what we're really asking is what feature vector should X1 be so that theta 1 transpose x1 is approximately equal to 5, that's Alice's rating, and theta 2 transpose x1 is also approximately equal to 5, and theta 3 transpose x1 is approximately equal to 0, so this would be Carol's rating, and theta 4 transpose X1 is approximately equal to 0. And from this it looks like, you know, X1 equals one that's the intercept term, and then 1.0, 0.0, that makes sense given what we know of Alice, Bob, Carol, and Dave's preferences for movies and the way they rated this movie. And so more generally, we can go down this list and try to figure out what might be reasonable features for these other movies as well.



Question

Consider the following movie ratings:

.	User 1	User 2	User 3	(romance)
Movie 1	0	1.5	2.5	?

Note that there is only one feature x_1. Suppose that:

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$$

What would be a reasonable value for $x_1^{(1)}$ (the value denoted "?" in the table above)?

- 0.5
- 1
- 2
- Any of these values would be equally reasonable.

Consider the following movie ratings:

.	User 1	User 2	User 3	(romance)
Movie 1	0	1.5	2.5	?

Note that there is only one feature x_1. Suppose that:

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$$

What would be a reasonable value for $x_1^{(1)}$ (the value denoted "?" in the table above)?

- 0.5

Correct

- 1
- 2
- Any of these values would be equally reasonable.

Let's formalize this problem of learning the features XI. Let's say that our users have given us their preferences. So let's say that our users have come and, you know, told us these values for theta 1 through theta of NU and we want to learn the feature vector XI for movie number I. What we can do is therefore pose the following optimization problem. So we want to sum over all the indices J for which we have a rating for movie I because we're trying to learn the features for movie I that is this feature vector XI. So and then what we want to do is minimize this squared error, so we want to choose features XI, so that, you know, the predictive value of how user J rates movie I will be similar, will be not too far in the squared error sense of the actual value YIJ that we actually

observe in the rating of user j on movie I. So, just to summarize what this term does is it tries to choose features XI so that for all the users J that have rated that movie, the algorithm also predicts a value for how that user would have rated that movie that is not too far, in the squared error sense, from the actual value that the user had rated that movie. So that's the squared error term. As usual, we can also add this sort of regularization term to prevent the features from becoming too big. So this is how we would learn the features for one specific movie but what we want to do is learn all the features for all the movies and so what I'm going to do is add this extra summation here so I'm going to sum over all Nm movies, N subscript m movies, and minimize this objective on top that sums of all movies. And if you do that, you end up with the following optimization problem. And if you minimize this, you have hopefully a reasonable set of features for all of your movies.

Optimization algorithm

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

$$\min_{x^{(1)},\ldots,x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Question

Suppose you use gradient descent to minimize:

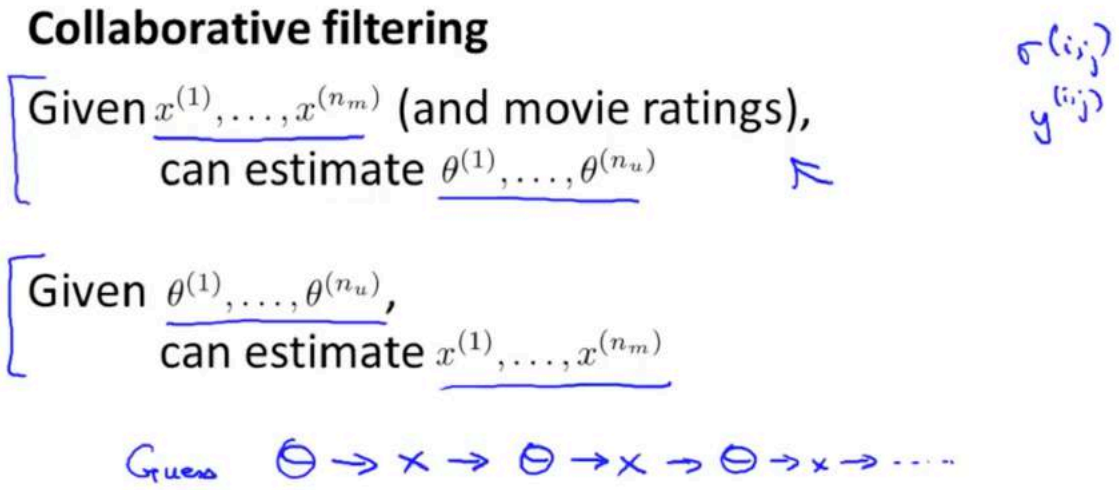
$$\min_{x^{(1)},\ldots,x^{(n_m)}} \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$$

Which of the following is a correct gradient descent update rule for $i \neq 0$?

- $x_k^{(i)} := x_k^{(i)} + \alpha\left(\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)\theta_k^{(j)}\right)$
- $x_k^{(i)} := x_k^{(i)} - \alpha\left(\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)\theta_k^{(j)}\right)$
- $x_k^{(i)} := x_k^{(i)} + \alpha\left(\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)\theta_k^{(j)} + \lambda x_k^{(i)}\right)$
- **(selected)** $x_k^{(i)} := x_k^{(i)} - \alpha\left(\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)\theta_k^{(j)} + \lambda x_k^{(i)}\right)$

So putting everything together, what we, the algorithm we talked about in the previous video and the algorithm that we just talked about in this video. In the previous video, what we showed was that you know, if you have a set of movie ratings, so if you have the data the rij's and then you have the yij's that will be the movie ratings. Then given features for your different movies we can learn these parameters theta. So if you knew the features, you can learn the parameters theta for your different users. And what we showed earlier in this video is that if your users are willing to give you parameters, then you can estimate features for the different movies. So this is kind of a chicken and egg problem. Which comes first? You know, do we want if we can get the thetas, we can know the Xs. If we have the Xs, we can learn the thetas. And what you can do is, and then this actually works, what you can do is in fact randomly guess some value of the thetas. Now based on your initial random guess for the thetas, you can then go ahead and use the procedure that we just talked about in order to learn features for your different movies. Now given some initial set of features for your movies you can then use this first method that we talked about in the previous video to try to get an even better estimate for your parameters theta. Now that you have a better setting of the parameters theta for your users, we can use that to maybe even get a better set of features and so on. We can sort of keep iterating, going back and forth and optimizing theta, x theta, x theta, nd this actually works and if you do this, this will actually cause your album to converge to a reasonable set of features for you movies and a reasonable set of parameters for your different users. So this is a basic collaborative filtering algorithm. This isn't actually the final algorithm that we're going to use. In the next video we are going to be able to improve on this algorithm and make it quite a bit more computationally efficient. But, hopefully this gives you a sense of how you can formulate a problem where you can simultaneously learn the parameters and simultaneously learn the features

from the different movies. And for this problem, for the recommender system problem, this is possible only because each user rates multiple movies and hopefully each movie is rated by multiple users. And so you can do this back and forth process to estimate theta and x.



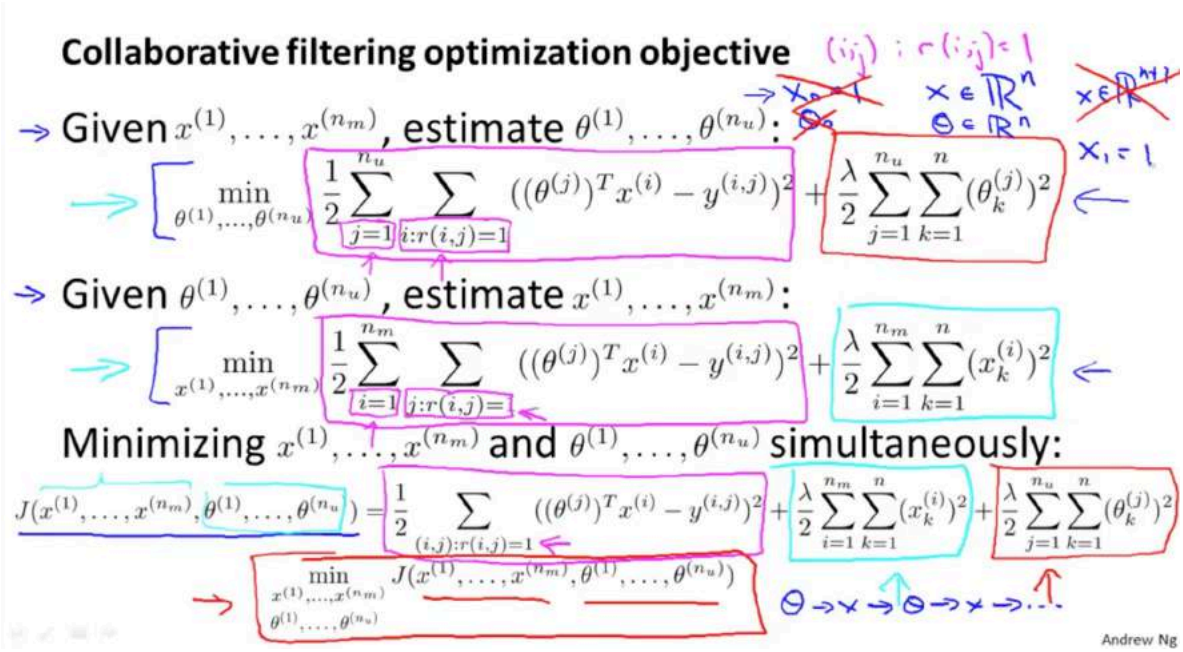
So to summarize, in this video we've seen an initial collaborative filtering algorithm. The term collaborative filtering refers to the observation that when you run this algorithm with a large set of users, what all of these users are effectively doing are sort of collaboratively--or collaborating to get better movie ratings for everyone because with every user rating some subset with the movies, every user is helping the algorithm a little bit to learn better features, and then by helping-- by rating a few movies myself, I will be helping the system learn better features and then these features can be used by the system to make better movie predictions for everyone else. And so there is a sense of collaboration where every user is helping the system learn better features for the common good. This is this collaborative filtering. And, in the next video what we going to do is take the ideas that have worked out, and try to develop a better an even better algorithm, a slightly better technique for collaborative filtering.

Collaborative Filtering Algorithm

In the last couple videos, we talked about the ideas of how, first, if you're given features for movies, you can use that to learn parameters data for users. And second, if you're given parameters for the users, you can use that to learn features for the movies. In this video we're going to take those ideas and put

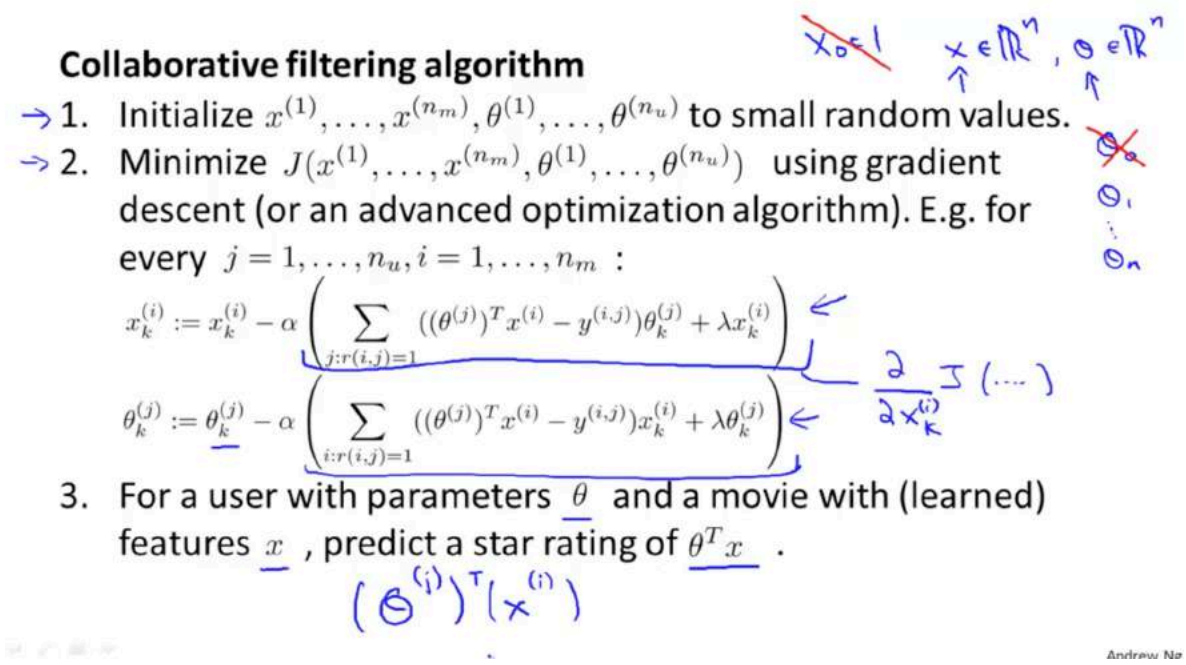
them together to come up with a collaborative filtering algorithm. So one of the things we worked out earlier is that if you have features for the movies then you can solve this minimization problem to find the parameters theta for your users. And then we also worked that out, if you are given the parameters theta, you can also use that to estimate the features x, and you can do that by solving this minimization problem. So one thing you could do is actually go back and forth. Maybe randomly initialize the parameters and then solve for theta, solve for x, solve for theta, solve for x. But, it turns out that there is a more efficient algorithm that doesn't need to go back and forth between the x's and the thetas, but that can solve for theta and x simultaneously. And here it is. What we are going to do, is basically take both of these optimization objectives, and put them into the same objective. So I'm going to define the new optimization objective j, which is a cost function, that is a function of my features x and a function of my parameters theta. And, it's basically the two optimization objectives I had on top, but I put together. So, in order to explain this, first, I want to point out that this term over here, this squared error term, is the same as this squared error term and the summations look a little bit different, but let's see what the summations are really doing. The first summation is sum over all users J and then sum over all movies rated by that user. So, this is really summing over all pairs IJ, that correspond to a movie that was rated by a user. Sum over J says, for every user, the sum of all the movies rated by that user. This summation down here, just does things in the opposite order. This says for every movie I, sum over all the users J that have rated that movie and so, you know these summations, both of these are just summations over all pairs ij for which r of i J is equal to 1. It's just something over all the user movie pairs for which you have a rating. and so those two terms up there is just exactly this first term, and I've just written the summation here explicitly, where I'm just saying the sum of all pairs IJ, such that RIJ is equal to 1. So what we're going to do is define a combined optimization objective that we want to minimize in order to solve simultaneously for x and theta. And then the other terms in the optimization objective are this, which is a regularization in terms of theta. So that came down here and the final piece is this term which is my optimization objective for the x's and that became this. And this optimization objective j actually has an interesting property that if you were to hold the x's constant and just minimize with respect to the thetas then you'd be solving exactly this problem, whereas if you were to do the opposite, if you were to hold the thetas constant, and minimize j only with respect to the x's, then it becomes equivalent to this. Because either this term or this term is constant if you're minimizing only the respective x's or only respective thetas. So here's an optimization objective that puts together my cost functions in terms of x and in terms of theta. And in order to come up with just one optimization problem, what we're going to do, is treat this cost function, as a function of my features x and of my user pro user parameters data and just minimize this whole thing, as a function of both the Xs and a function of the thetas. And really the only difference between this and the older algorithm is that, instead of going back and forth, previously we talked about minimizing

with respect to theta then minimizing with respect to x, whereas minimizing with respect to theta, minimizing with respect to x and so on. In this new version instead of sequentially going between the 2 sets of parameters x and theta, what we are going to do is just minimize with respect to both sets of parameters simultaneously. Finally one last detail is that when we're learning the features this way. Previously we have been using this convention that we have a feature x0 equals one that corresponds to an interceptor. When we are using this sort of formalism where we're are actually learning the features, we are actually going to do away with this convention. And so the features we are going to learn x, will be in Rn. Whereas previously we had features x and Rn + 1 including the intercept term. By getting rid of x0 we now just have x in Rn. And so similarly, because the parameters theta is in the same dimension, we now also have theta in RN because if there's no x0, then there's no need parameter theta 0 as well. And the reason we do away with this convention is because we're now learning all the features, right? So there is no need to hard code the feature that is always equal to one. Because if the algorithm really wants a feature that is always equal to 1, it can choose to learn one for itself. So if the algorithm chooses, it can set the feature X1 equals 1. So there's no need to hard code the feature of 001, the algorithm now has the flexibility to just learn it by itself.



So, putting everything together, here is our collaborative filtering algorithm. first we are going to initialize x and theta to small random values. And this is a little bit like neural network training, where there we were also initializing all the parameters of a neural network to small random values. Next we're then going to minimize the cost function using great intercepts or one of the advance optimization algorithms. So, if you take derivatives you find that the great intercept like these and so this term here is the partial derivative of the cost

function, I'm not going to write that out, with respect to the feature value Xik and similarly this term here is also a partial derivative value of the cost function with respect to the parameter theta that we're minimizing. And just as a reminder, in this formula that we no longer have this X0 equals 1 and so we have that x is in Rn and theta is a Rn. In this new formalism, we're regularizing every one of our perimeters theta, you know, every one of our parameters Xn. There's no longer the special case theta zero, which was regularized differently, or which was not regularized compared to the parameters theta 1 down to theta. So there is now no longer a theta 0, which is why in these updates, I did not break out a special case for k equals 0. So we then use gradient descent to minimize the cost function j with respect to the features x and with respect to the parameters theta. And finally, given a user, if a user has some parameters, theta, and if there's a movie with some sort of learned features x, we would then predict that that movie would be given a star rating by that user of theta transpose j. Or just to fill those in, then we're saying that if user J has not yet rated movie I, then what we do is predict that user J is going to rate movie I according to theta J transpose Xi.



Question

In the algorithm we described, we initialized $x^{(1)}, \ldots, x^{(n_m)}$ and $\theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values. Why is this?

- This step is optional. Initializing to all 0's would work just as well.
- Random initialization is always necessary when using gradient descent on any problem.
- This ensures that $x^{(i)} \neq \theta^{(j)}$ for any i, j.
- This serves as symmetry breaking (similar to the random initialization of a neural network's parameters) and ensures the algorithm learns features $x^{(1)}, \ldots, x^{(n_m)}$ that are different from each other.

Correct

So that's the collaborative filtering algorithm and if you implement this algorithm you actually get a pretty decent algorithm that will simultaneously learn good features for hopefully all the movies as well as learn parameters for all the users and hopefully give pretty good predictions for how different users will rate different movies that they have not yet rated

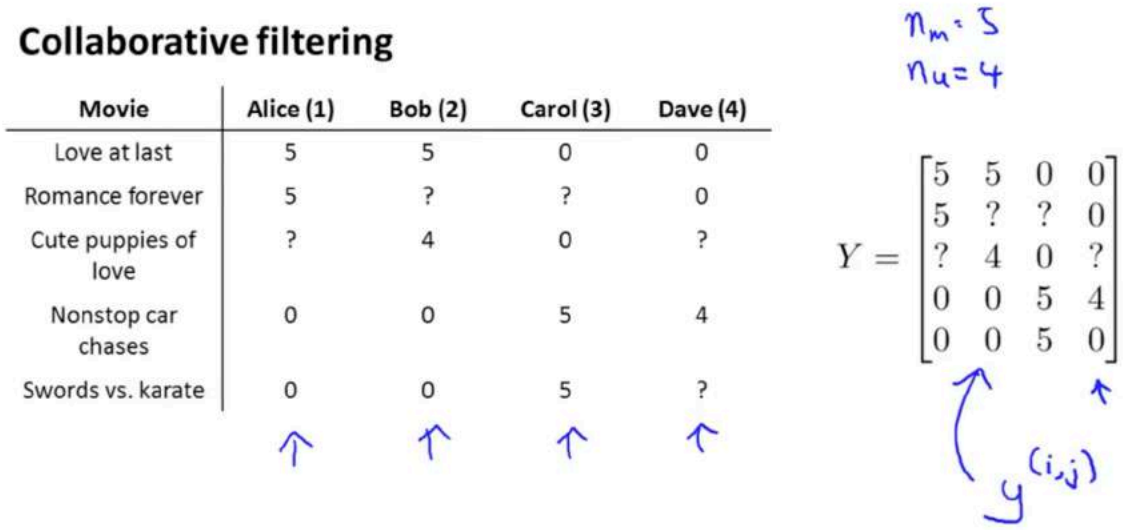
Low Rank Matrix Factorization

Vectorization: Low Rank Matrix Factorization

In the last few videos, we talked about a collaborative filtering algorithm. In this video I'm going to say a little bit about the vectorization implementation of this algorithm. And also talk a little bit about other things you can do with this algorithm. For example, one of the things you can do is, given one product can

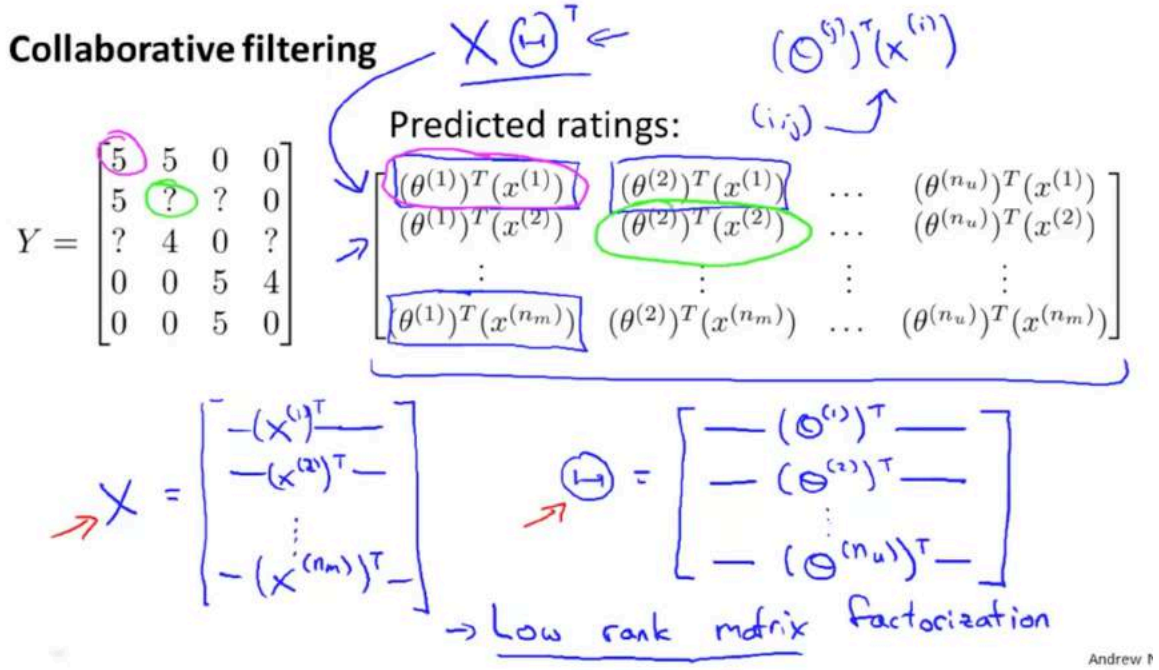
you find other products that are related to this so that for example, a user has recently been looking at one product. Are there other related products that you could recommend to this user? So let's see what we could do about that.

What I'd like to do is work out an alternative way of writing out the predictions of the collaborative filtering algorithm. To start, here is our data set with our five movies and what I'm going to do is take all the ratings by all the users and group them into a matrix. So, here we have five movies and four users, and so this matrix y is going to be a 5 by 4 matrix. It's just you know, taking all of the elements, all of this data. Including question marks, and grouping them into this matrix. And of course the elements of this matrix of the (i, j) element of this matrix is really what we were previously writing as y superscript i, j. It's the rating given to movie i by user j.



Given this matrix y of all the ratings that we have, there's an alternative way of writing out all the predictive ratings of the algorithm. And, in particular if you look at what a certain user predicts on a certain movie, what user j predicts on movie i is given by this formula. And so, if you have a matrix of the predicted ratings, what you would have is the following matrix where the i, j entry. So this corresponds to the rating that we predict using j will give to movie i is exactly equal to that theta j transpose XI, and so, you know, this is a matrix where this first element the one-one element is a predictive rating of user one or movie one and this element, this is the one-two element is the predicted rating of user two on movie one, and so on, and this is the predicted rating of user one on the last movie and if you want, you know, this rating is what we would have predicted for this value and this rating is what we would have predicted for that value, and so on. Now, given this matrix of predictive ratings there is then a simpler or vectorized way of writing these out. In particular if I define the matrix x, and this is going to be just like the matrix we had earlier for linear regression to be sort of x1 transpose x2 transpose down to x of nm transpose.

So I'm take all the features for my movies and stack them in rows. So if you think of each movie as one example and stack all of the features of the different movies and rows. And if we also to find a matrix capital theta, and what I'm going to do is take each of the per user parameter vectors, and stack them in rows, like so. So that's theta 1, which is the parameter vector for the first user. And, you know, theta 2, and so, you must stack them in rows like this to define a matrix capital theta and so I have nu parameter vectors all stacked in rows like this. Now given this definition for the matrix x and this definition for the matrix theta in order to have a vectorized way of computing the matrix of all the predictions you can just compute x times the matrix theta transpose, and that gives you a vectorized way of computing this matrix over here. To give the collaborative filtering algorithm that you've been using another name. The algorithm that we're using is also called low rank matrix factorization. And so if you hear people talk about low rank matrix factorization that's essentially exactly the algorithm that we have been talking about. And this term comes from the property that this matrix x times theta transpose has a mathematical property in linear algebra called that this is a low rank matrix and so that's what gives rise to this name low rank matrix factorization for these algorithms, because of this low rank property of this matrix x theta transpose. In case you don't know what low rank means or in case you don't know what a low rank matrix is, don't worry about it. You really don't need to know that in order to use this algorithm. But if you're an expert in linear algebra, that's what gives this algorithm, this other name of low rank matrix factorization.



Question

Let $X = \begin{bmatrix} - & (x^{(1)})^T & - \\ & \vdots & \\ - & (x^{(n_m)} & - \end{bmatrix}$, $\Theta = \begin{bmatrix} - & (\theta^{(1)})^T & - \\ & \vdots & \\ - & (\theta^{(n_u)} & - \end{bmatrix}$.

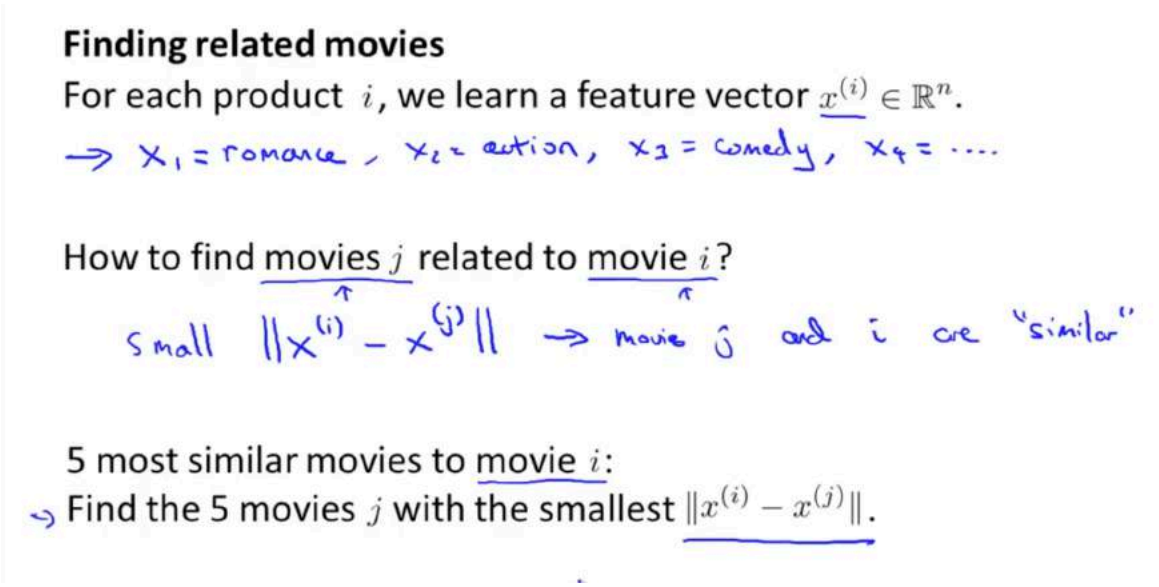
What is another way of writing the following:

$$\begin{bmatrix} (x^{(1)})^T(\theta^{(1)}) & \dots & (x^{(1)})^T(\theta^{(n_u)}) \\ \vdots & \ddots & \vdots \\ (x^{(n_m)})^T(\theta^{(1)}) & \dots & (x^{(n_m)})^T(\theta^{(n_u)}) \end{bmatrix}$$

- $X\Theta$
- $X^T\Theta$
- $X\Theta^T$ (selected)
- $\Theta^T X^T$

Finally, having run the collaborative filtering algorithm here's something else that you can do which is use the learned features in order to find related movies. Specifically for each product i really for each movie i, we've learned a feature vector xi. So, you know, when you learn a certain features without really know that can the advance what the different features are going to be, but if you run the algorithm and perfectly the features will tend to capture what are the important aspects of these different movies or different products or what have you. What are the important aspects that cause some users to like certain movies and cause some users to like different sets of movies. So maybe you end up learning a feature, you know, where x1 equals romance, x2 equals action similar to an earlier video and maybe you learned a different feature x3 which is a degree to which this is a comedy. Then some feature x4 which is, you know, some other thing. And you have N features all together and after you have learned features it's actually often pretty difficult to go in to the learned features and come up with a human understandable interpretation of what these features really are. But in practice, you know, the features even though these features can be hard to visualize. It can be hard to figure out just what these features are. Usually, it will learn features that are very meaningful for capturing whatever are the most important or the most salient properties of a movie that causes you to like or dislike it. And so now let's say we want to address the following problem. Say you have some specific movie i and you want to find other movies j that are related to that movie. And so well, why would you want to do this? Right, maybe you have a user that's browsing movies, and they're currently watching movie j, than what's a reasonable

movie to recommend to them to watch after they're done with movie j? Or if someone's recently purchased movie j, well, what's a different movie that would be reasonable to recommend to them for them to consider purchasing. So, now that you have learned these feature vectors, this gives us a very convenient way to measure how similar two movies are. In particular, movie i has a feature vector xi. and so if you can find a different movie, j, so that the distance between xi and xj is small, then this is a pretty strong indication that, you know, movies j and i are somehow similar. At least in the sense that some of them likes movie i, maybe more likely to like movie j as well. So, just to recap, if your user is looking at some movie i and if you want to find the 5 most similar movies to that movie in order to recommend 5 new movies to them, what you do is find the five movies j, with the smallest distance between the features between these different movies. And this could give you a few different movies to recommend to your user.

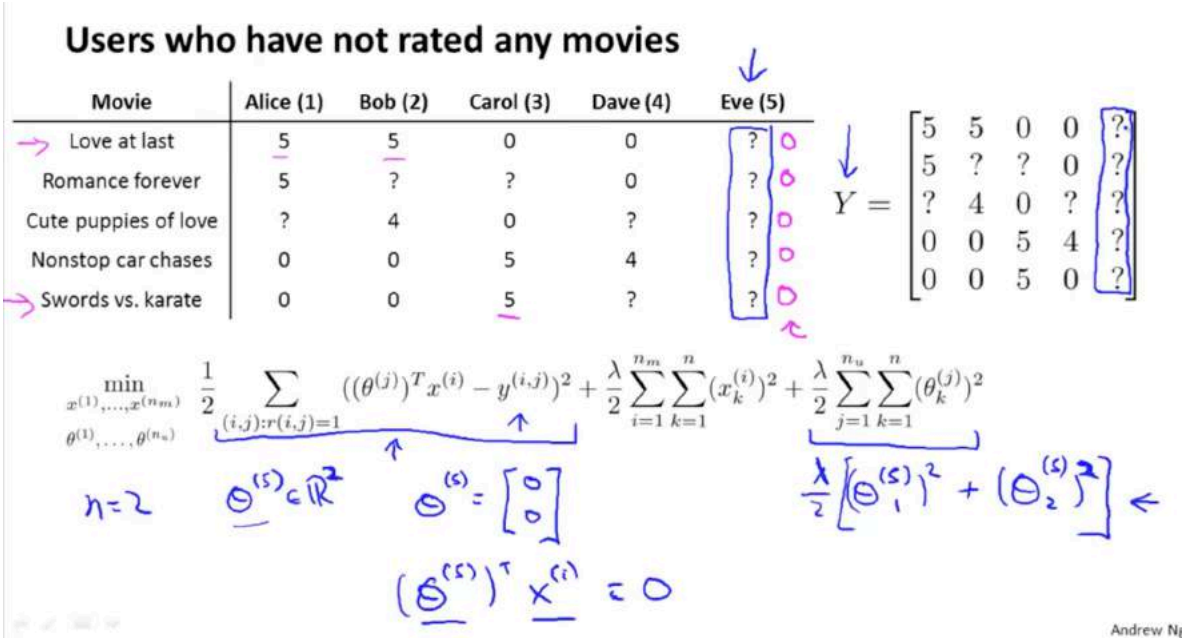


So with that, hopefully, you now know how to use a vectorized implementation to compute all the predicted ratings of all the users and all the movies, and also how to do things like use learned features to find what might be movies and what might be products that aren't related to each other.

Implementational Detail: Mean Normalization

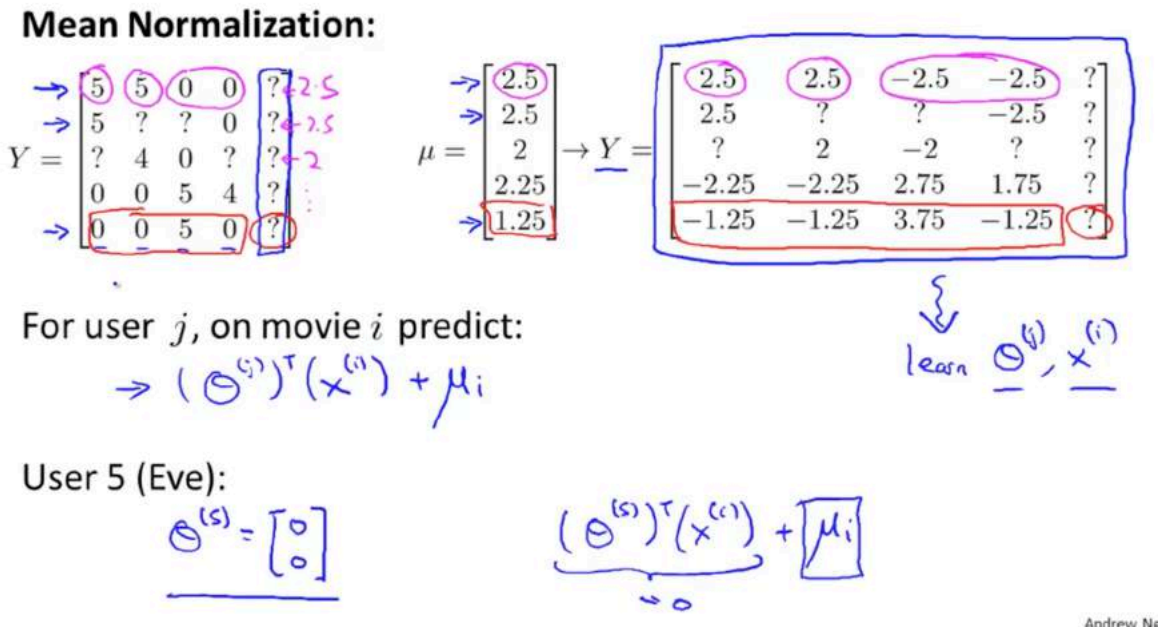
By now you've seen all of the main pieces of the recommender system

algorithm or the collaborative filtering algorithm. In this video I want to just share one last implementational detail, namely mean normalization, which can sometimes just make the algorithm work a little bit better. To motivate the idea of mean normalization, let's consider an example of where there's a user that has not rated any movies. So, in addition to our four users, Alice, Bob, Carol, and Dave, I've added a fifth user, Eve, who hasn't rated any movies. Let's see what our collaborative filtering algorithm will do on this user. Let's say that n is equal to 2 and so we're going to learn two features and we are going to have to learn a parameter vector theta 5, which is going to be in R2, remember this is now vectors in Rn not Rn+1, we'll learn the parameter vector theta 5 for our user number 5, Eve. So if we look in the first term in this optimization objective, well the user Eve hasn't rated any movies, so there are no movies for which Rij is equal to one for the user Eve and so this first term plays no role at all in determining theta 5 because there are no movies that Eve has rated. And so the only term that effects theta 5 is this term. And so we're saying that we want to choose vector theta 5 so that the last regularization term is as small as possible. In other words we want to minimize this lambda over 2 theta 5 subscript 1 squared plus theta 5 subscript 2 squared so that's the component of the regularization term that corresponds to user 5, and of course if your goal is to minimize this term, then what you're going to end up with is just theta 5 equals 0 0. Because a regularization term is encouraging us to set parameters close to 0 and if there is no data to try to pull the parameters away from 0, because this first term doesn't effect theta 5, we just end up with theta 5 equals the vector of all zeros. And so when we go to predict how user 5 would rate any movie, we have that theta 5 transpose xi, for any i, that's just going to be equal to zero. Because theta 5 is 0 for any value of x, this inner product is going to be equal to 0. And what we're going to have therefore, is that we're going to predict that Eve is going to rate every single movie with zero stars. But this doesn't seem very useful does it? I mean if you look at the different movies, Love at Last, this first movie, a couple people rated it 5 stars. And for even the Swords vs. Karate, someone rated it 5 stars. So some people do like some movies. It seems not useful to just predict that Eve is going to rate everything 0 stars. And in fact if we're predicting that eve is going to rate everything 0 stars, we also don't have any good way of recommending any movies to her, because you know all of these movies are getting exactly the same predicted rating for Eve so there's no one movie with a higher predicted rating that we could recommend to her, so, that's not very good. The idea of mean normalization will let us fix this problem. So here's how it works. As before let me group all of my movie ratings into this matrix Y, so just take all of these ratings and group them into matrix Y. And this column over here of all question marks corresponds to Eve's not having rated any movies.



Now to perform mean normalization what I'm going to do is compute the average rating that each movie obtained. And I'm going to store that in a vector that we'll call mu. So the first movie got two 5-star and two 0-star ratings, so the average of that is a 2.5-star rating. The second movie had an average of 2.5-stars and so on. And the final movie that has 0, 0, 5, 0. And the average of 0, 0, 5, 0, that averages out to an average of 1.25 rating. And what I'm going to do is look at all the movie ratings and I'm going to subtract off the mean rating. So this first element 5 I'm going to subtract off 2.5 and that gives me 2.5. And the second element 5 subtract off of 2.5, get a 2.5. And then the 0, 0, subtract off 2.5 and you get -2.5, -2.5. In other words, what I'm going to do is take my matrix of movie ratings, take this wide matrix, and subtract form each row the average rating for that movie. So, what I'm doing is just normalizing each movie to have an average rating of zero. And so just one last example. If you look at this last row, 0 0 5 0. We're going to subtract 1.25, and so I end up with these values over here. So now and of course the question marks stay a question mark. So each movie in this new matrix Y has an average rating of 0. What I'm going to do then, is take this set of ratings and use it with my collaborative filtering algorithm. So I'm going to pretend that this was the data that I had gotten from my users, or pretend that these are the actual ratings I had gotten from the users, and I'm going to use this as my data set with which to learn my parameters theta J and my features XI - from these mean normalized movie ratings. When I want to make predictions of movie ratings, what I'm going to do is the following: for user J on movie I, I'm gonna predict theta J transpose XI, where X and theta are the parameters that I've learned from this mean normalized data set. But, because on the data set, I had subtracted off the means in order to make a prediction on movie i, I'm going to need to add back in the mean, and so i'm going to add back in mu i.

And so that's going to be my prediction where in my training data subtracted off all the means and so when we make predictions and we need to add back in these means mu i for movie i. And so specifically if you user 5 which is Eve, the same argument as the previous slide still applies in the sense that Eve had not rated any movies and so the learned parameter for user 5 is still going to be equal to 0, 0. And so what we're going to get then is that on a particular movie i we're going to predict for Eve theta 5, transpose xi plus add back in mu i and so this first component is going to be equal to zero, if theta five is equal to zero. And so on movie i, we are going to end a predicting mu i. And, this actually makes sense. It means that on movie 1 we're going to predict Eve rates it 2.5. On movie 2 we're gonna predict Eve rates it 2.5. On movie 3 we're gonna predict Eve rates it at 2 and so on. This actually makes sense, because it says that if Eve hasn't rated any movies and we just don't know anything about this new user Eve, what we're going to do is just predict for each of the movies, what are the average rating that those movies got. Finally, as an aside, in this video we talked about mean normalization, where we normalized each row of the matrix y, to have mean 0. In case you have some movies with no ratings, so it is analogous to a user who hasn't rated anything, but in case you have some movies with no ratings, you can also play with versions of the algorithm, where you normalize the different columns to have means zero, instead of normalizing the rows to have mean zero, although that's maybe less important, because if you really have a movie with no rating, maybe you just shouldn't recommend that movie to anyone, anyway. And so, taking care of the case of a user who hasn't rated anything might be more important than taking care of the case of a movie that hasn't gotten a single rating.



Question

We talked about mean normalization. However, unlike some other applications of feature scaling, we did not scale the movie ratings by dividing by the range (max – min value). This is because:

- ○ This sort of scaling is not useful when the value being predicted is real-valued.
- ● All the movie ratings are already comparable (e.g., 0 to 5 stars), so they are already on similar scales.
- ○ Subtracting the mean is mathematically equivalent to dividing by the range.
- ○ This makes the overall algorithm significantly more computationally efficient.

So to summarize, that's how you can do mean normalization as a sort of pre-processing step for collaborative filtering. Depending on your data set, this might some times make your implementation work just a little bit better.

Review

Quiz

1. Suppose you run a bookstore, and have ratings (1 to 5 stars) of books. Your collaborative filtering algorithm has learned a parameter vector $\theta^{(j)}$ for user j, and a feature vector $x^{(i)}$ for each book. You would like to compute the "training error", meaning the average squared error of your system's predictions on all the ratings that you have gotten from your users. Which of these are correct ways of doing so (check all that apply)? For this problem, let m be the total number of ratings you have gotten from your users. (Another way of saying this is that $m = \sum_{i=1}^{n_m} \sum_{j=1}^{n_u} r(i,j)$). [Hint: Two of the four options below are correct.]

- [x] $\frac{1}{m} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (\sum_{k=1}^{n} (\theta^{(j)})_k x_k^{(i)} - y^{(i,j)})^2$
- [ ] $\frac{1}{m} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} (\sum_{k=1}^{n} (\theta^{(k)})_j x_i^{(k)} - y^{(i,j)})^2$
- [x] $\frac{1}{m} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2$
- [ ] $\frac{1}{m} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - r(i,j))^2$

2. In which of the following situations will a collaborative filtering system be the most appropriate learning algorithm (compared to linear or logistic regression)?

- [x] You own a clothing store that sells many styles and brands of jeans. You have collected reviews of the different styles and brands from frequent shoppers, and you want to use these reviews to offer those shoppers discounts on the jeans you think they are most likely to purchase
- [ ] You manage an online bookstore and you have the book ratings from many users. You want to learn to predict the expected sales volume (number of books sold) as a function of the average rating of a book.
- [ ] You're an artist and hand-paint portraits for your clients. Each client gets a different portrait (of themselves) and gives you 1-5 star rating feedback, and each client purchases at most 1 portrait. You'd like to predict what rating your next customer will give you.
- [x] You run an online bookstore and collect the ratings of many users. You want to use this to identify what books are "similar" to each other (i.e., if one user likes a certain book, what are other books that she might also like?)

3. You run a movie empire, and want to build a movie recommendation system based on collaborative filtering. There were three popular review websites (which we'll call A, B and C) which users to go to rate movies, and you have just acquired all three companies that run these websites. You'd like to merge the three companies' datasets together to build a single/unified system. On website A, users rank a movie as having 1 through 5 stars. On website B, users rank on a scale of 1 - 10, and decimal values (e.g., 7.5) are allowed. On website C, the ratings are from 1 to 100. You also have enough information to identify users/movies on one website with users/movies on a different website. Which of the following statements is true?

- ( ) Assuming that there is at least one movie/user in one database that doesn't also appear in a second database, there is no sound way to merge the datasets, because of the missing data.
- (•) You can merge the three datasets into one, but you should first normalize each dataset's ratings (say rescale each dataset's ratings to a 1-100 range).
- ( ) You can combine all three training sets into one without any modification and expect high performance from a recommendation system.
- ( ) It is not possible to combine these websites' data. You must build three separate recommendation systems.

4. Which of the following are true of collaborative filtering systems? Check all that apply.

- [ ] When using gradient descent to train a collaborative filtering system, it is okay to initialize all the parameters ($x^{(i)}$ and $\theta^{(j)}$) to zero.

- [x] Recall that the cost function for the content-based recommendation system is $J(\theta) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$. Suppose there is only one user and he has rated every movie in the training set. This implies that $n_u = 1$ and $r(i,j) = 1$ for every i, j. In this case, the cost function $J(\theta)$ is equivalent to the one used for regularized linear regression.

- [x] If you have a dataset of users ratings' on some products, you can use these to predict one user's preferences on products he has not rated.

- [ ] To use collaborative filtering, you need to manually design a feature vector for every item (e.g., movie) in your dataset, that describes that item's most important properties.

5. Suppose you have two matrices A and B, where A is 5x3 and B is 3x5. Their product is $C = AB$, a 5x5 matrix. Furthermore, you have a 5x5 matrix R where every entry is 0 or 1. You want to find the sum of all elements $C(i,j)$ for which the corresponding $R(i,j)$ is 1, and ignore all elements $C(i,j)$ where $R(i,j) = 0$. One way to do so is the following code:

```
C = A * B;
total = 0;
for i = 1:5
  for j = 1:5
    if (R(i,j) == 1)
      total = total + C(i,j);
    end
  end
end
```

Which of the following pieces of Octave code will also correctly compute this total? Check all that apply. Assume all options are in code.

- [x] total = sum(sum((A * B) .* R))

- [x] C = (A * B) .* R; total = sum(C(:));

- [ ] total = sum(sum((A * B) * R));

- [ ] C = (A * B) * R; total = sum(C(:));