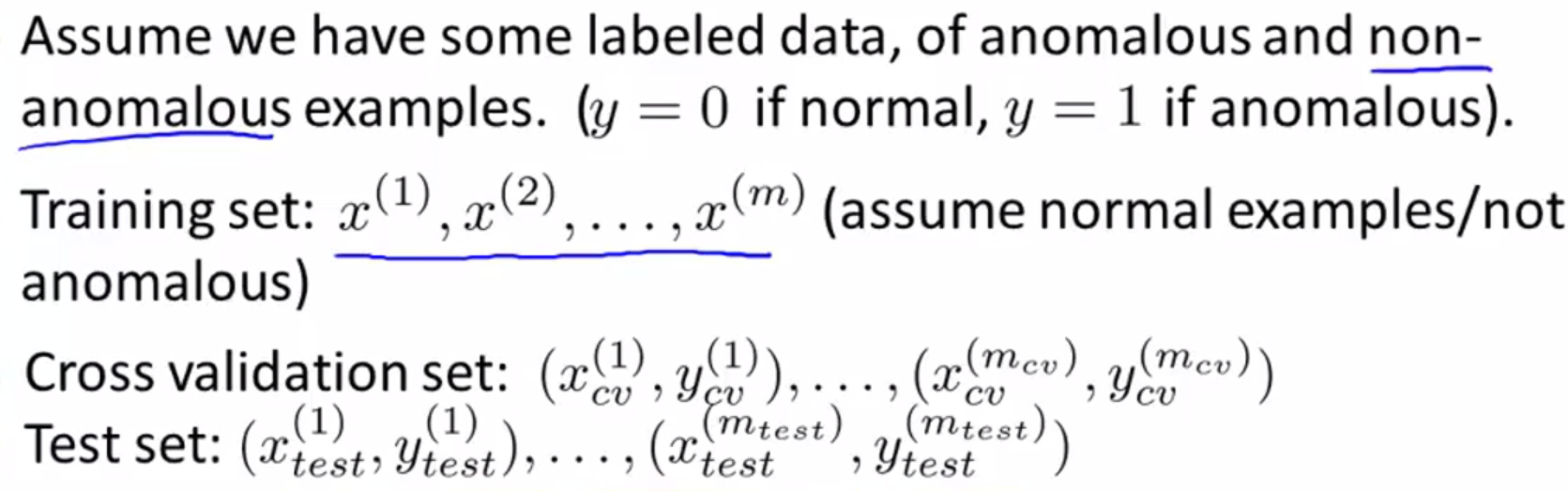
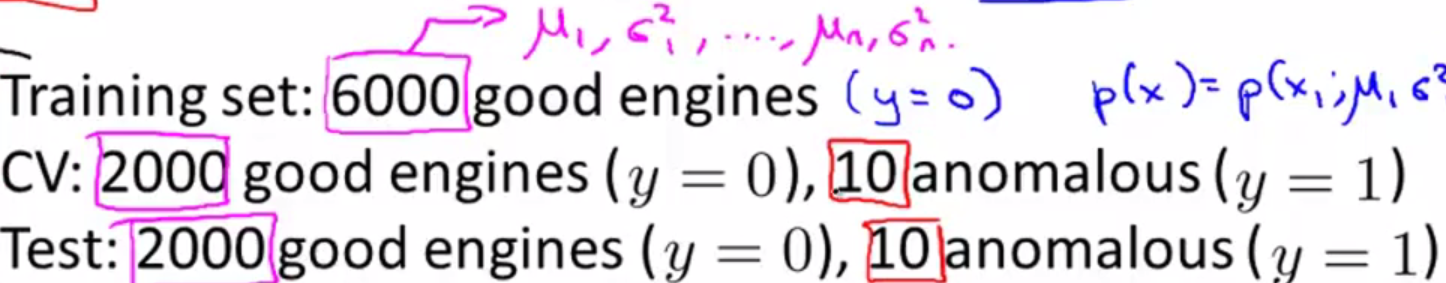
***Building an Anomaly Detection System***

**I. DEVELOPING AND EVALUATING AN ANOMALY DETECTION SYSTEM**

* The importance of **real number evaluation** 🡪 when trying to develop a learning algorithm for a specific application, often need to make a lot of choices (choosing features to use, etc.) + making decisions about all these choices is often much easier w/ a way to evaluate a learning algorithm that just *gives you back a number*.
* Have an idea for 1 extra feature? Run the algorithm w/ the feature + w/out the feature + see if it improved or worsened performance via a real number
* So in order to be able to develop an anomaly detection system quickly, it’d be a really helpful to have a way of *evaluating* an anomaly detection system.
* We're actually going to assume we have some labeled data
* So far, we've been treating anomaly detection as unsupervised (using unlabeled data)
* But if you have labeled data that specifies some anomalous + non-anomalous examples, this is the standard way of evaluating an anomaly detection algorithm.
* Some labeled data of a few anomalous examples of aircraft engines manufactured in the past + also have some non-anomalous examples
* y = 0 denotes normal/non-anomalous example + y = 1 denotes anomalous examples.
* The process of developing and evaluating an anomaly detection algorithm is as follows.
* Think of it as a training set as unlabeled = large collection of normal, non-anomalous examples.
* It's okay if a few anomalies slip into your unlabeled training set.
* Define a CV + a test set w/ which to evaluate a particular anomaly detection algorithm



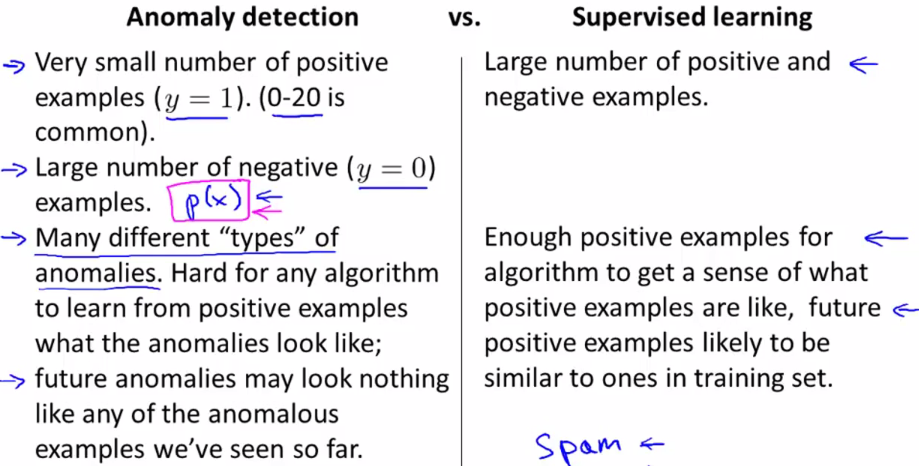
* Ex: 10k examples of normal engines (assume vast majority of these are non-anomalous engines) + 20 anomalous engines
* For a typical application of anomaly detection, non-anomalous examples may range from 20-50.
* Usually have a much larger number of good examples.
* A fairly typical way to split it into the training, CV, + test sets would be a 60/20/20 split:



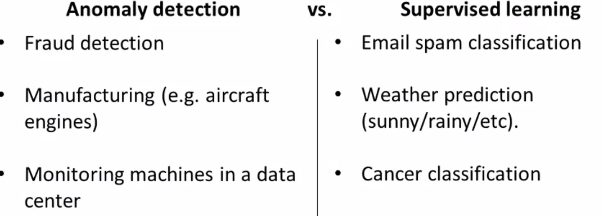
* 6k into unlabeled training set (unlabeled but all examples correspond to y = 0, as far as we know)
* We will use this to fit p(x(i) parametrized by μ(i), δ^2(i)) 🡪 to estimate parameters μ1, δ^2
* Halve remaining examples into CV + test sets w/ 2k examples
* Also have 20 flawed aircraft engines 🡪 split + put 10 in CV set + 10 in test set
* Like to think of the CV set + test set as being completely different data sets to each other, but, in anomaly detection, sometimes people use the same set of good examples in both the CV sets + the test sets, sometimes people use exactly the same sets of anomalous examples in these sets
* Certainly using the same data in the CV set + the test set, that is not considered a good ML practice.
* So, given the training, CV + test sets, here's how you develop and evaluate an algorithm.
* Take the training set + we fit the model p(x) 🡪 fit all those parameters for all the Gaussians to the m unlabeled examples
* Then imagine the anomaly detection algorithm is actually making predictions
* On the CV or test set, given example X, think of the algorithm as predicting that y = 1 when p(x) < epsilon + y = 0 if p(x) >= epsilon.
* Can evaluate it by seeing how often it gets these labels right, similar to evaluation metrics used in supervised learning.
* Labels will be very skewed b/c y = 0 are normal examples, usually much more common than y = 1 anomalous examples.
* B/c the data is very skewed (i.e. b/c y = 0 is much more common) classification accuracy would NOT be a good evaluation metric
* W/ a very skewed data set, predicting y = 0 all the time will give very high classification accuracy.
* Instead, use evaluation metrics like computing the **fraction of TP’s, FP’s, FN’s, TN’s, etc**.
* Or compute the **position** + **recall** of this algorithm or compute **the f1 score** (a single real number way of summarizing the position + recall values
* **Epsilon** is this threshold used to decide when to flag something as an anomaly.
* If you have a CV set, another way to + to choose this parameter epsilon would be to try many different values of epsilon + then pick the value that maximizes f1 score, or that otherwise does well on your CV set
* More generally, the way to use the training, testing, + CV sets, is when trying to make decisions (what features to include, tune the parameter epsilon), we’d continually evaluate the algorithm on the CV sets + make all those decisions for a set of features, or a value of epsilon we're happy w/
* We can then take the final model + do a final evaluation of the algorithm on the test sets.
* Being able to evaluate an algorithm w/ a single real number evaluation, like an F1 score, often allows you to much more efficient use of time when trying to develop an anomaly detection system

**II. ANOMALY DETECTION VS. SUPERVISED LEARNING**

* If we have labeled data, why don't we just use a supervised ML method like logistic regression or a NN to try to learn directly from the labeled data to predict whether Y = 1 or Y = 0.



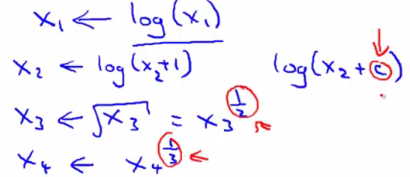
* If you have a problem w/ a very small number of positive (anomaly) examples, say 0-20, you might consider using an anomaly detection algorithm
* Usually w/ such a small set of positive examples, say 50, we're going to save the positive examples just for the CV set + the test set.
* In contrast, in a typical normal anomaly detection setting, we often have a relatively large number of negative (normal) examples.
* We can then use this very large number of negative examples to fit the model p(x).
* When doing the process of estimating p(x) + affecting all those Gaussian parameters, we only need *negative* examples to do that.
* So if you have little positive data + a lot of negative data, we can still fit p(x) pretty well.
* For supervised learning, more typically we’d have a reasonably large number of both positive + negative examples
* For anomaly detection applications, often there are very different types of anomalies.
* Many things could go wrong for an aircraft engine, for example
* If that's the case, + if you have a pretty small set of positive examples, it can be difficult for an algorithm to learn what the anomalies look like from the small set of positive examples.
* In particular, future anomalies may look nothing like ones you've seen so far.
* Ex: In your set of positive examples, you've seen 20 different ways an aircraft engine could go wrong, but tomorrow, you need to detect them in a totally new set w/ a totally new type of anomaly/way for an aircraft engine to be broken you've never seen before
* If that's the case, it might be more promising to just model negative examples w/ a Gaussian model p(x) instead of trying too hard to model positive examples b/c tomorrow's anomaly may be nothing like ones seen so far.
* In contrast, you have enough positive examples for an algorithm to get a sense of what they are like.
* In particular, if you think future positive examples are likely to be similar to ones in the training set, it might be more reasonable to have a supervised learning algorithm that looks at all positive examples + all negative examples, + uses that to try to distinguish between positives + negatives
* A key difference really is 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 them
* What we’d need to do instead is take a large set of negative examples + have it just learn p(x) from just the negative (normal) examples + reserve the small number of positive examples for evaluating our algorithms to use in the either the CV or test set.
* Side note: many different types of spam email (anomalies = positives), but we have enough examples of spam to know most types of spam email
* That's why we usually think of spam a supervised learning problem even though there are many different types of anomalies (negative examples)
* Applications of anomaly detection versus supervised learning:



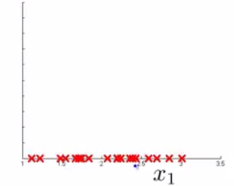
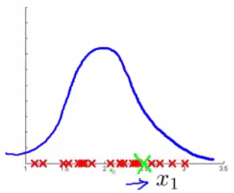
* If you have many different types of ways for people to try to commit fraud + a relatively small number of fraudulent users on a website, use an anomaly detection algorithm.
* A very major online retailer might have a lot of people commit fraud on their site, so they actually have a lot of examples of y = 1, so fraud detection could actually shift to supervised learning.
* But, if you haven't seen many examples of users doing strange things on your website, fraud detection is more frequently treated as an anomaly detection algorithm
* Hopefully, we see more examples are NOT anomalies in manufacturing, but if for some manufacturing processes w/ large volumes, you see a lot of bad examples + maybe manufacturing can shift to supervised learning
* Email spam classification, weather prediction, classifying cancers 🡺 W/ many or possibly equal numbers of positive + negative examples, tend to treat all these as supervised learning problems.
* 
* 
* 
* For many other problems faced by various tech companies, they actually very few or sometimes 0 positive training examples 🡺 so many different types of anomalies they’ve never seen before.
* For those sorts of problems, very often the algorithm that is used is an anomaly detection algorithm.

**III. CHOOSING WHAT FEATURES TO USE**

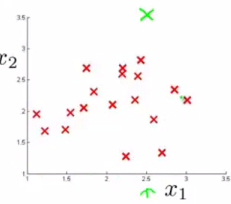
* When applying anomaly detection, 1 of the things that has a huge effect on how well it does is what features you choose to give the anomaly detection algorithm.
* In our anomaly detection algorithm, 1 thing we did was model the features using a Gaussian distribution w/ P(x(i); μ(i), δ^2(i))
* 1 thing to do is plot the histogram of the data to make sure it looks vaguely Gaussian (normal) before feeding it to an anomaly detection algorithm
* It'll usually work even if your data isn't Gaussian, but this is a nice check to run.
* If data is skewed, play w/ different transformations of the data to make it look more Gaussian.
* The algorithm will usually work okay even if you don't, but if you transform the data to be more Gaussian, it might work a bit better.



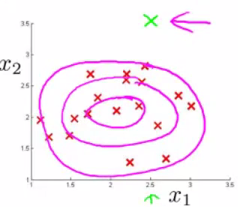
* How do you come up with features for an anomaly detection algorithm?
* 1 way is via an **error analysis procedure**, similar to the error analysis procedure for supervised learning (train a complete algorithm + run it on a CV set, look at examples it gets wrong, + see if we can come up w/ extra features to help the algorithm do better on those examples)
* Now for anomaly detection, we’re hoping p(x) will be large for normal examples + small for anomalous examples
* A pretty common problem would be if p(x) is **comparable** (maybe large for both normal + anomalous examples)
* Ex: Unlabeled data w/ just 1 feature, x1 + we try to fit a Gaussian to this

* Notice the green anomalous example w/ x = 2.5. So I plot my anomalous example there
* It's kind of buried in the middle of a bunch of normal examples, + so this anomalous example has a pretty high probability (height on the blue curve), + the algorithm fails to flag this as anomalous
* Can actually look at my training examples + look at what went wrong w/ that particular example +hope it helps come up w/ a new feature x2 that helps to distinguish between bad + good examples
* If I can create a new feature, X2, when I re-plot my data in 2D, hopefully the feature x2 takes on an unusual value for this anomaly for the same x1 value it had originally



* Now the anomaly detection algorithm gives high probability to data in the central regions (red), slightly lower probability as we move away from the center, and examples way out there have very low probability to



* This process involves looking at the mistakes the algorithm is making/looking at anomalies the algorithm is failing to flag + see if it helps to create some new feature to make it easier to distinguish anomalies from good examples
* Usually want to choose features that will take on either very, very large or very, very small values for examples we think might turn out to be anomalies.
* Ex: monitoring CPUs in a data center 🡪 lots of machines (maybe 10’s of 1000’s) + we want to know if 1 of the machines is acting up
* Features you might choose: memory used, # of disc accesses/sec, CPU load, network traffic.
* In my data set I think CPU load + network traffic tend to grow linearly w/ each other.
* Maybe I'm running a bunch of web servers + if 1 of my servers is serving a lot of users = have a very high CPU load + very high network traffic.
* But we have a suspicion that 1 of the failure cases is 1 of my CPUs has a job that gets stuck in some infinite loop
* So CPU load grows, but network traffic doesn't b/c it's just spinning its wheels 🡪 doing a lot of CPU work while stuck in some infinite loop.
* To detect that type of anomaly, create a new feature, X5 = CPU load/network traffic
* Here X5 will take on a unusually large value if 1 of the machines has a very large CPU load but not much network traffic + will help an anomaly detection capture that certain type of anomaly
* Can also get creative + come up w/ other features as well
* x6 = CPU load^2 / network traffic = another variant of a feature like x5 to try to capture anomalies where 1 machine has very high CPU load w/out commensurately large network traffic
* By creating features like these, you can start to capture anomalies that correspond to unusual combinations of values of the features