Coruse 3

Clustering and Anomaly Detection

Recommenders (one of the most commercially ML technologies, worth even billions)

Reinforcement Learning

We no longer have labels so it's not supervised.

Week 1

What is clustering?

A clustering algorithm looks at data points and points out which ones are related or similar. Do to this, we only have x's, not y's.

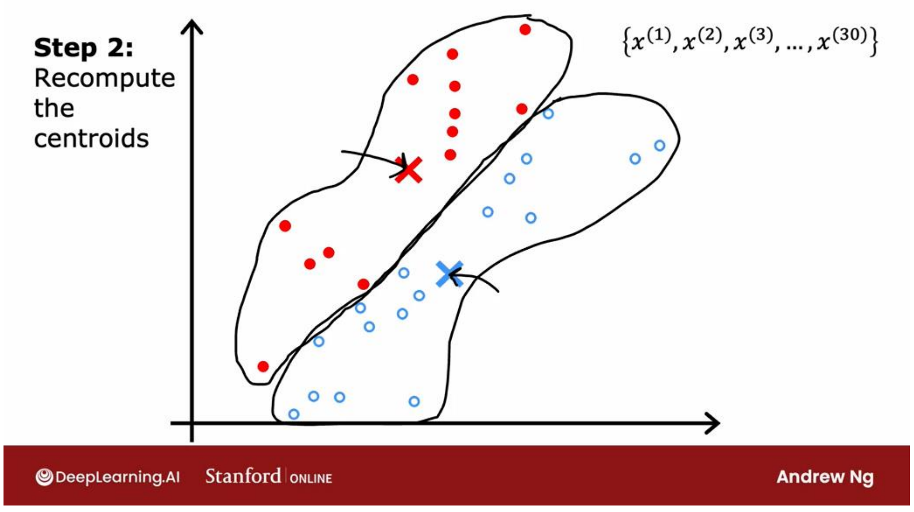
Goal: Find sth interesting about data, find the way it is structured. Clustering, for example, will try to group similar data points into packages.

Clustering has also been used to analyze DNA data. Look at DNA and detect people with similar features. One interesting application is astronomical data analysis.

K-Means Intuition

K-means redeatedly executes 2 steps.

- **Determine centroids, assign data points to centroids.**
  - First, it makes a random guess about centers of data clusters. Number of clusters is a hyperparameter (how to choose will come). Then it moves these **cluster centroids**.
  - Then it checks every data point and determines which cluster centroid that data point is closest to. Assign that point to closest centroid.
- **Recompute the centroids.**
  - Get average of a cluster, say red, and move centroid to that average location. To this for every centroid. In every iteration, you get average of entire training set.

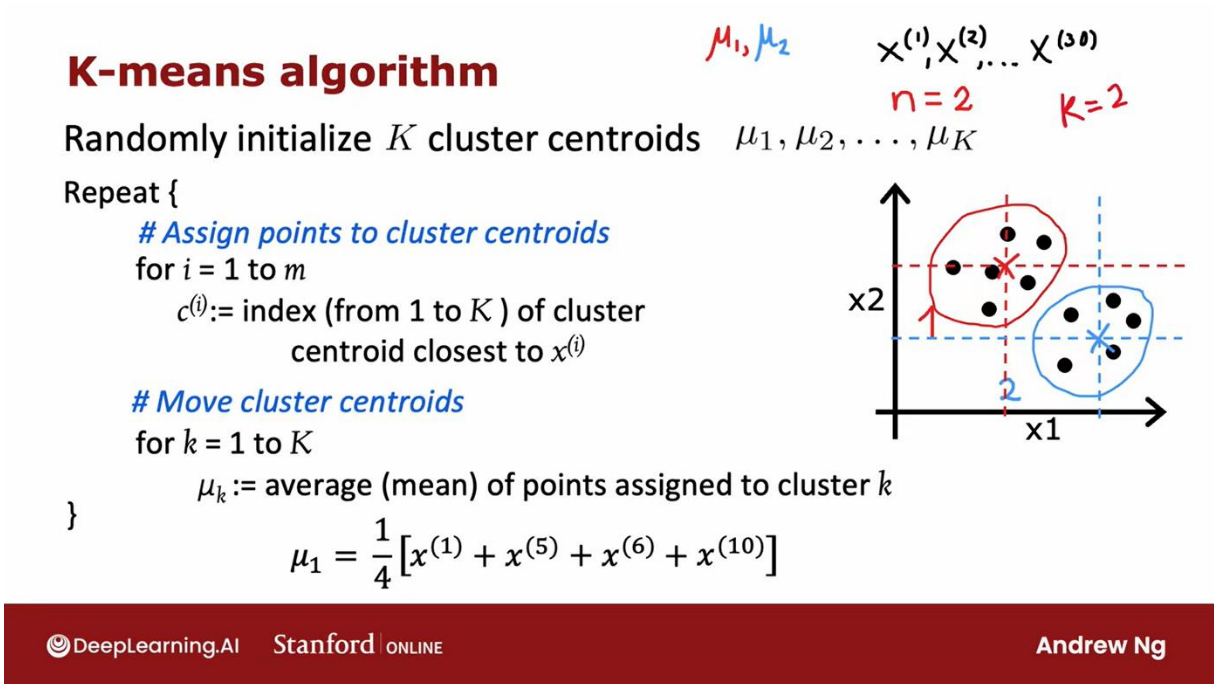

Step 2: Recompute the centroids
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$
DeepLearning.AI
Stanford ONLINE
Andrew Ng

In a point, you'll see that centoids are not moving anymore and points stay the same.

This means algorithm has converged.

K-Means Algorithm

Note: Centroids are vectors that have the same dimension as your data points.



K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

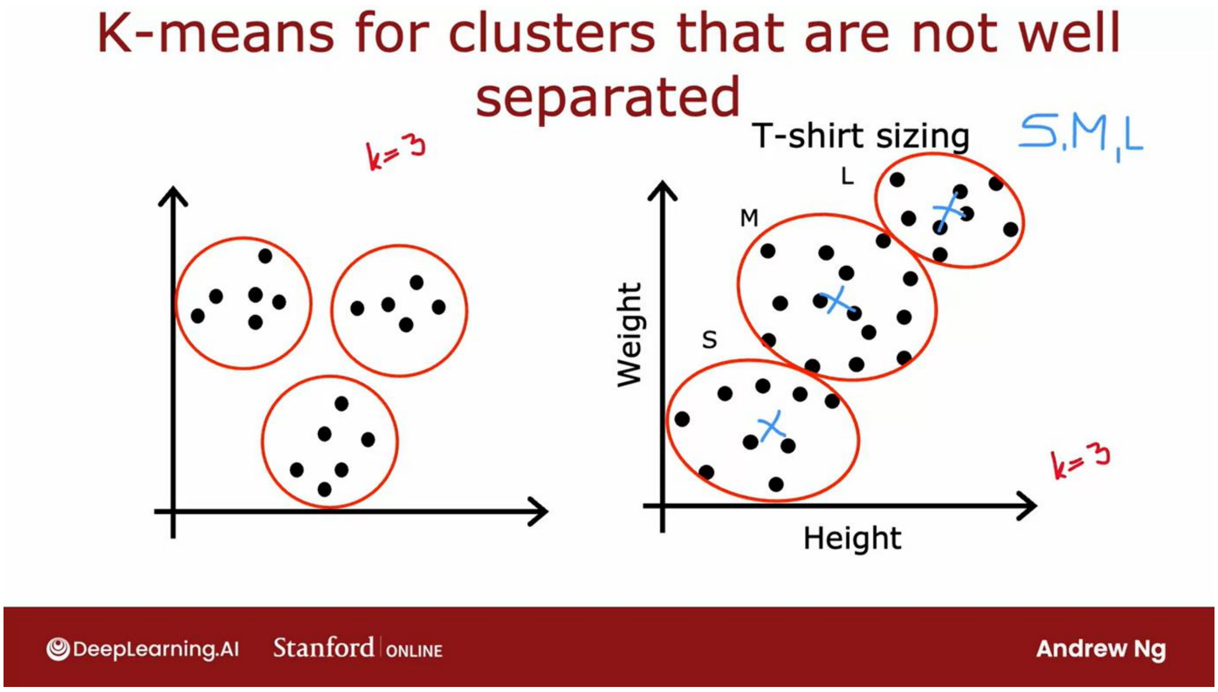
```
Repeat {
    # Assign points to cluster centroids
    for $i$ = 1 to $m$
        $c^{(i)}$ := index (from 1 to $K$) of cluster
                     centroid closest to $x^{(i)}$

    # Move cluster centroids
    for $k$ = 1 to $K$
        $\mu_k$ := average (mean) of points assigned to cluster $k$
}
```

$$\mu_1 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right]$$

****There is one corner case (an exceptional situation) of this algorithm:**

What if a cluster has 0 points assigned to it? Situation is not wanted and computing average is not well-defined. Solution:

1) Eliminate that cluster and continue with K-1 number of clusters.
2) Randomly reassign that cluster's centroid.
   First approach is more commonly used.



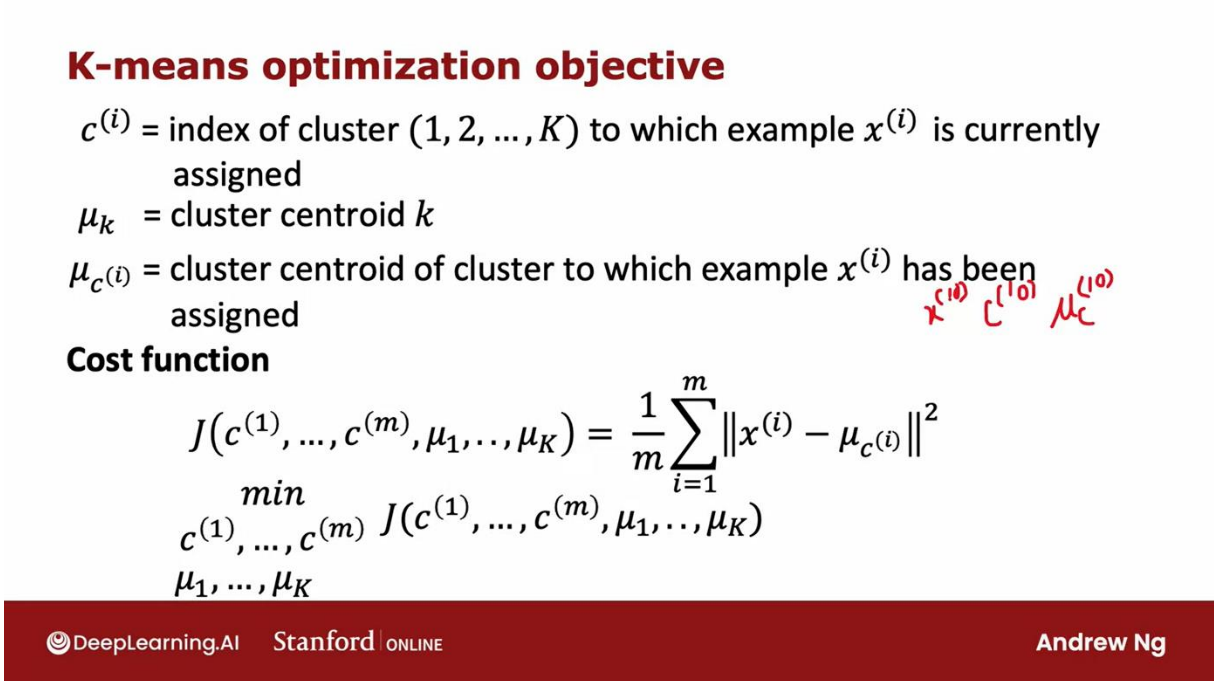
K-means for clusters that are not well separated

Optimization objective: What K-Means is trying to do?

In supervised learning, we had a cost function based on labels and our goal was to minimize it. K-Means is also trying to minimize a specific cost function.

Cost function will be a function of c1 to cm and mu_1 to mu_k:

Minimize: Average of squared distances of every data point to cluster it has been assigned to.



K-means optimization objective

$c^{(i)}$ = index of cluster $(1, 2, \ldots, K)$ to which example $x^{(i)}$ is currently assigned

μ_k = cluster centroid k

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

$x^{(10)}$ $c^{(10)}$ $\mu_{c^{(10)}}$

Cost function

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu_{c^{(i)}}\right\|^2$$

$$\min_{\substack{c^{(1)}, \ldots, c^{(m)} \\ \mu_1, \ldots, \mu_K}} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$$

DeepLearning.AI Stanford ONLINE Andrew Ng

This cost function has a name. It's called.

DISTORTION FUNCTION

*If you think about the previous page;

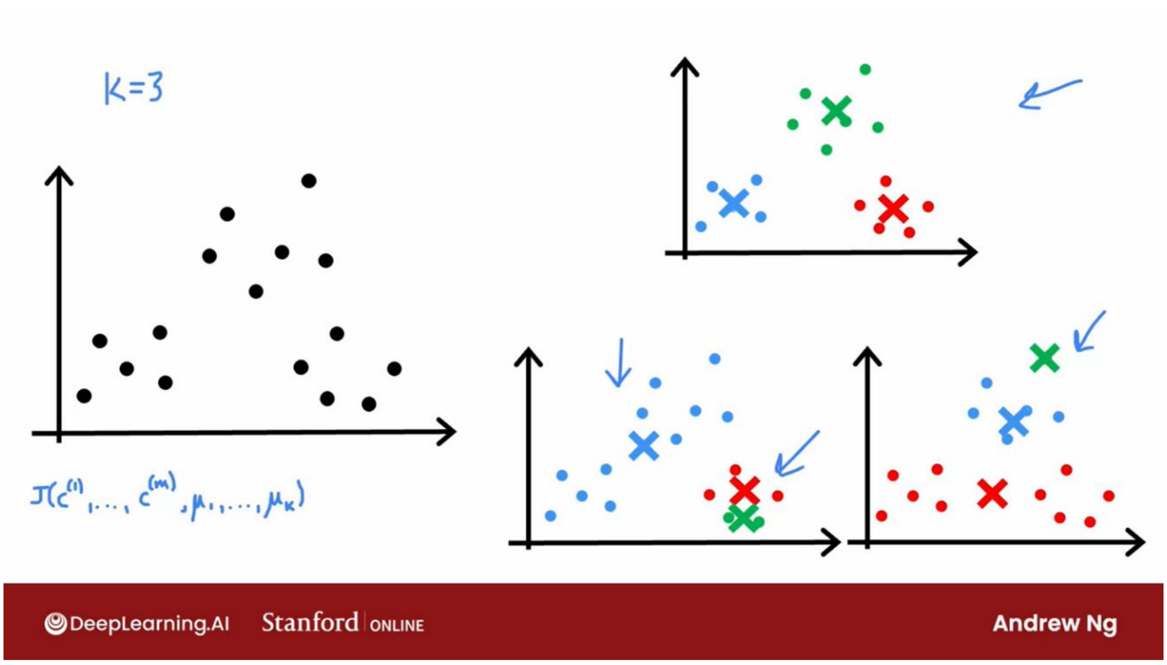
> Step 1 keeps mu_1 to mu_k fix and minimizes J with changing c1 to cm. Step 2 keeps c1 to cm fix and adjusts mu_1 to mu_k to minimize J.

***DISTORTION COST CAN NEVER GO UP, IT CAN EITHER DECREASE OR STAY THE SAME. IF IT GOES UP, YOU DID SOMETHING WRONG.**

Initializing K-Means

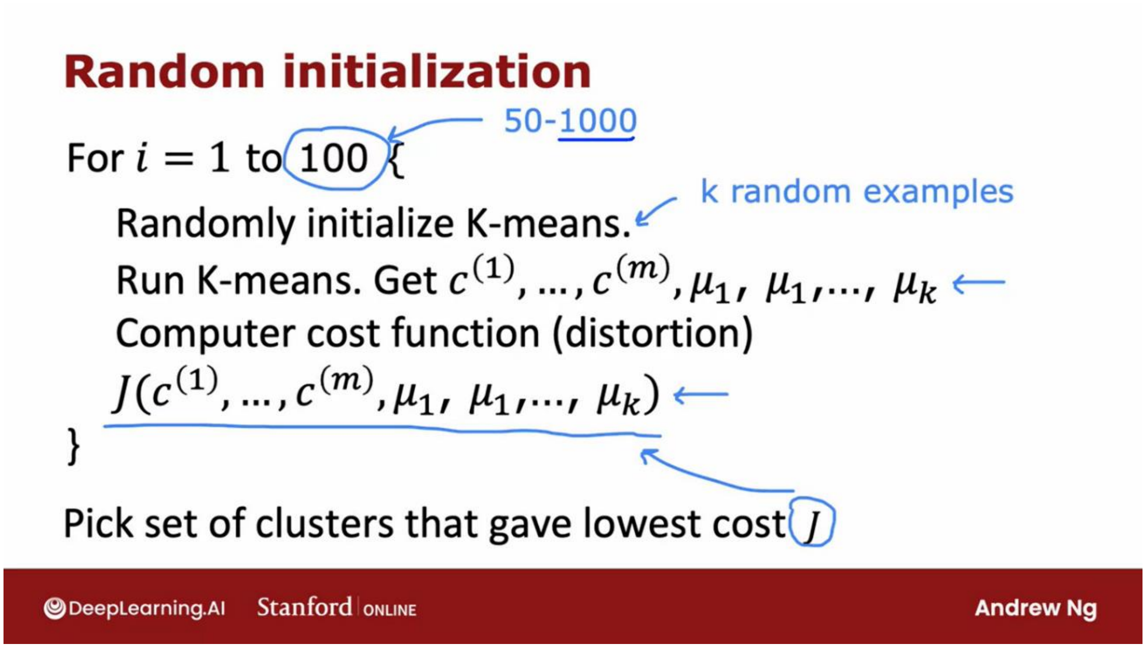
Step 1 was to choose random locations for centroids.

- You can pick K number of random training examples and set mu_1 to mu_k equal to these K examples. (Consider that in this approach, the way we comprehended cost function changes a bit. In other approach we just choose random initialization points but this way makes that thought process a bit "spicy")
  This was is much more commonly used.



Above, we initialized randomly and got different clusters. **This happens because cost function finds a local minimum and gets stuck in it.**

What you can do is to do multiple initializations and compare cost function of those situations at the end.



Random initialization

```
For $i = 1$ to $\boxed{100}$ {          50-1000
    Randomly initialize K-means.          k random examples
    Run K-means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \mu_1, \dots, \mu_k$ ←
    Computer cost function (distortion)
    $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \mu_1, \dots, \mu_k)$ ←
}
```

Pick set of clusters that gave lowest cost $\boxed{J}$

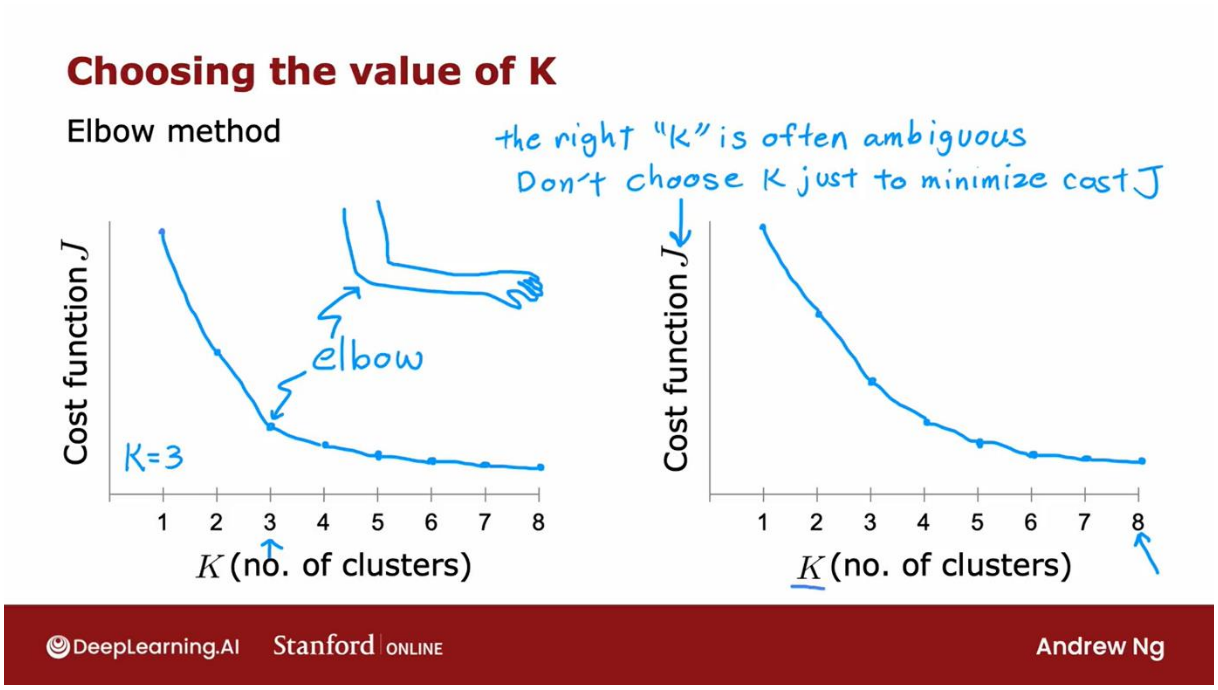
DeepLearning.AI
Stanford ONLINE
Andrew Ng

How do you choose the value of K?

To start with, for a lot of clustering problems it’s ambiguous.

Andrew NG: “I hardly ever use this technique. Right value of K is mostly ambiguous and when you plot the cost function, usually you don’t see an elbow, it just decreases smoothly.”

Method: Plot cost value with K. Then, if you can, choose the most meaningful value where J was decreasing rapidly before, but not so rapidly after. It looks like an elbow so this method is called the **"Elbow Method".**



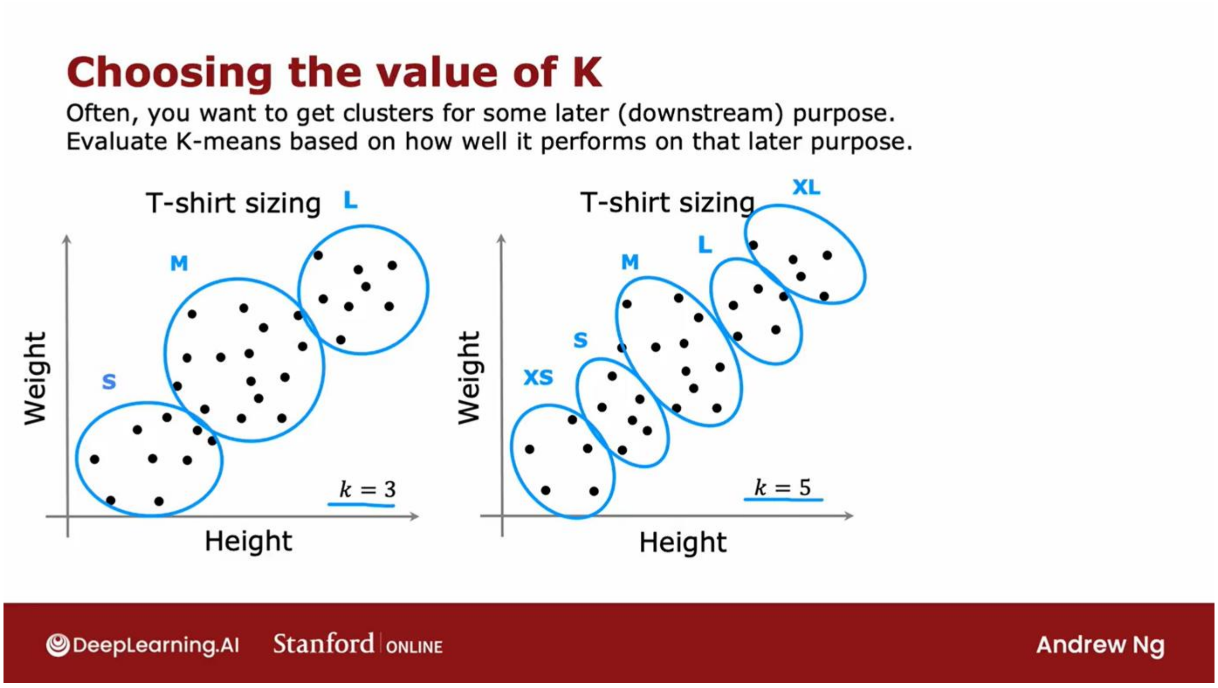
Choosing the value of K

Elbow method

DON'T: Don't just try to minimize value of J. As K gets bigger, J will always decrease.

So how do you choose a value?

If you're doing clustering, you have a purpose. You're going to use clusters for that purpose. So:



Choosing the value of K

Often, you want to get clusters for some later (downstream) purpose.
Evaluate K-means based on how well it performs on that later purpose.

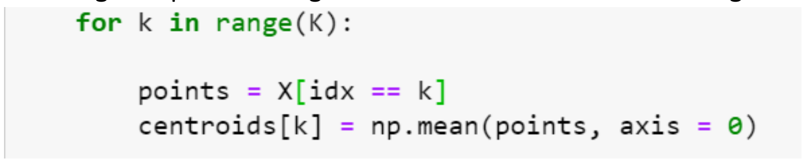
If you're having 5 clusters, customers can find T-shirt that fit them more easily, but it'll increase your costs. Consider this trade-off and then make a decision.

To sum up; choice of K depends on you and your purposes. Consider what you'll do with those clusters.

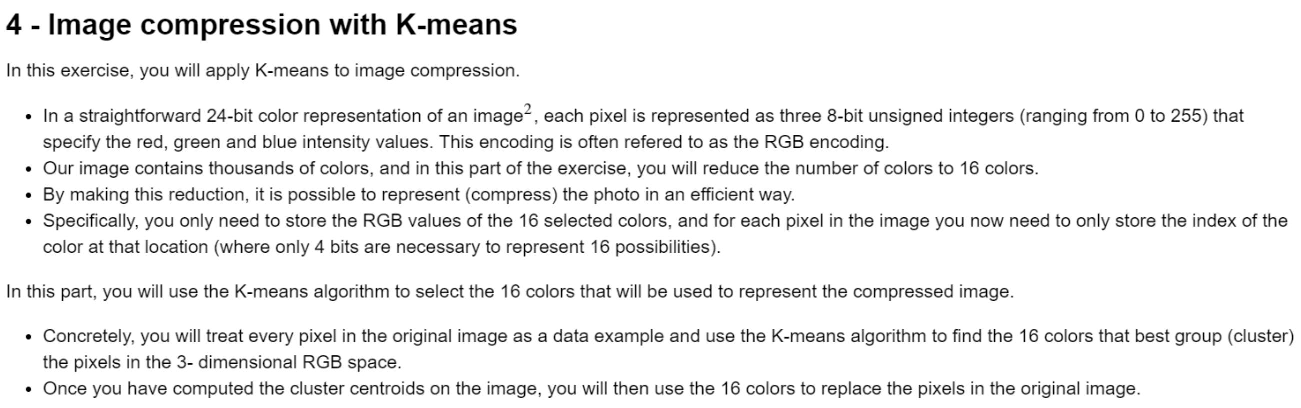
Another Example: An application of K-Means to image compression: There will be a trade-off between quality of an image and how much you can compress, to save space.

NOTES FROM PRACTICE LAB

- To compare distances to cluster centoids, you need to keep in mind that both data points and centroids are high dimensional vectors. To get distance between them, you can use this code:
  np.linalg.norm(X[i] - centroids[j])
- Use **np.argmin** to get min value.
- Below is the process of updating centroids. First find the data points where idx is equal to k, meaning that point is assigned to our current centroid. Then get their mean.



```
    for k in range(K):

        points = X[idx == k]
        centroids[k] = np.mean(points, axis = 0)
```



4 - Image compression with K-means

In this exercise, you will apply K-means to image compression.

- In a straightforward 24-bit color representation of an image[2], each pixel is represented as three 8-bit unsigned integers (ranging from 0 to 255) that specify the red, green and blue intensity values. This encoding is often refered to as the RGB encoding.
- Our image contains thousands of colors, and in this part of the exercise, you will reduce the number of colors to 16 colors.
- By making this reduction, it is possible to represent (compress) the photo in an efficient way.
- Specifically, you only need to store the RGB values of the 16 selected colors, and for each pixel in the image you now need to only store the index of the color at that location (where only 4 bits are necessary to represent 16 possibilities).

In this part, you will use the K-means algorithm to select the 16 colors that will be used to represent the compressed image.

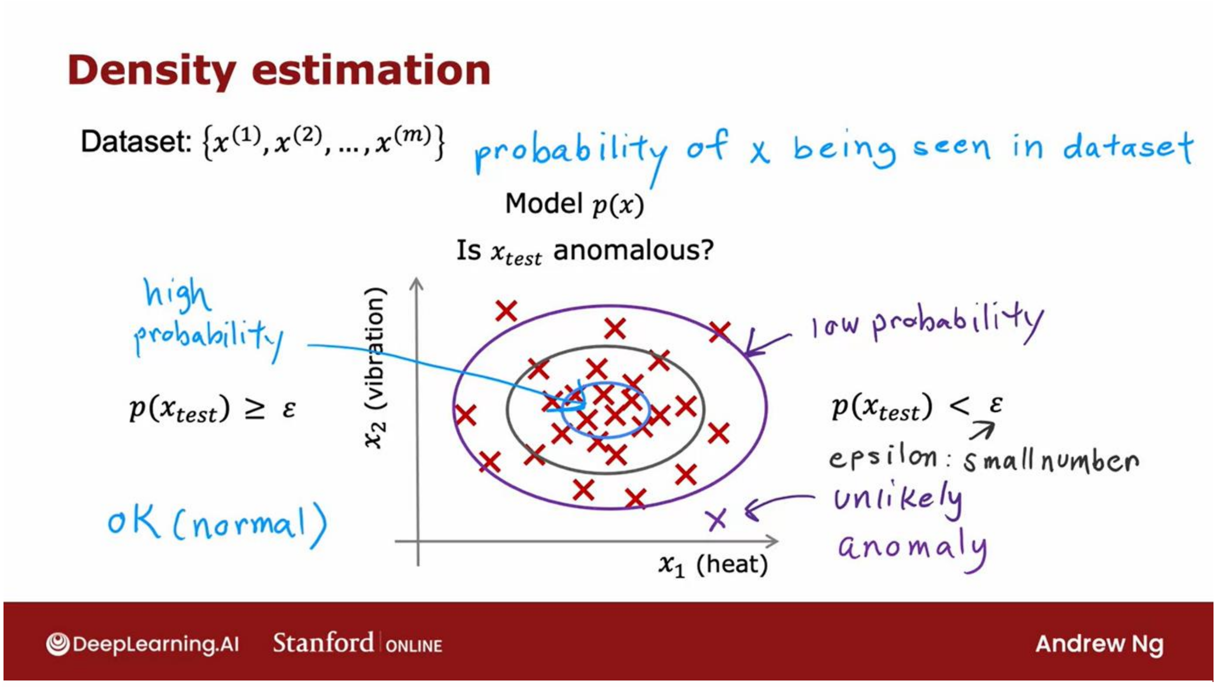
- Concretely, you will treat every pixel in the original image as a data example and use the K-means algorithm to find the 16 colors that best group (cluster) the pixels in the 3- dimensional RGB space.
- Once you have computed the cluster centroids on the image, you will then use the 16 colors to replace the pixels in the original image.

Anomaly Detection

Means detection possible problems. We expect most of samples to be normal, undefected. You'll have a dataset of unlabeled features, look at those features and decide whether a new example is normal or not. **"Is this okay? Is there something weird?"**

Example Usage: Defect detection, fraud detection (with user activities: how often do they log in, what pages do they visit, how many transactions or posts…)

Most common way to perform anomaly detection is to use **Density Estimation. It finds out which values have high probability, and which values have low for each feature.



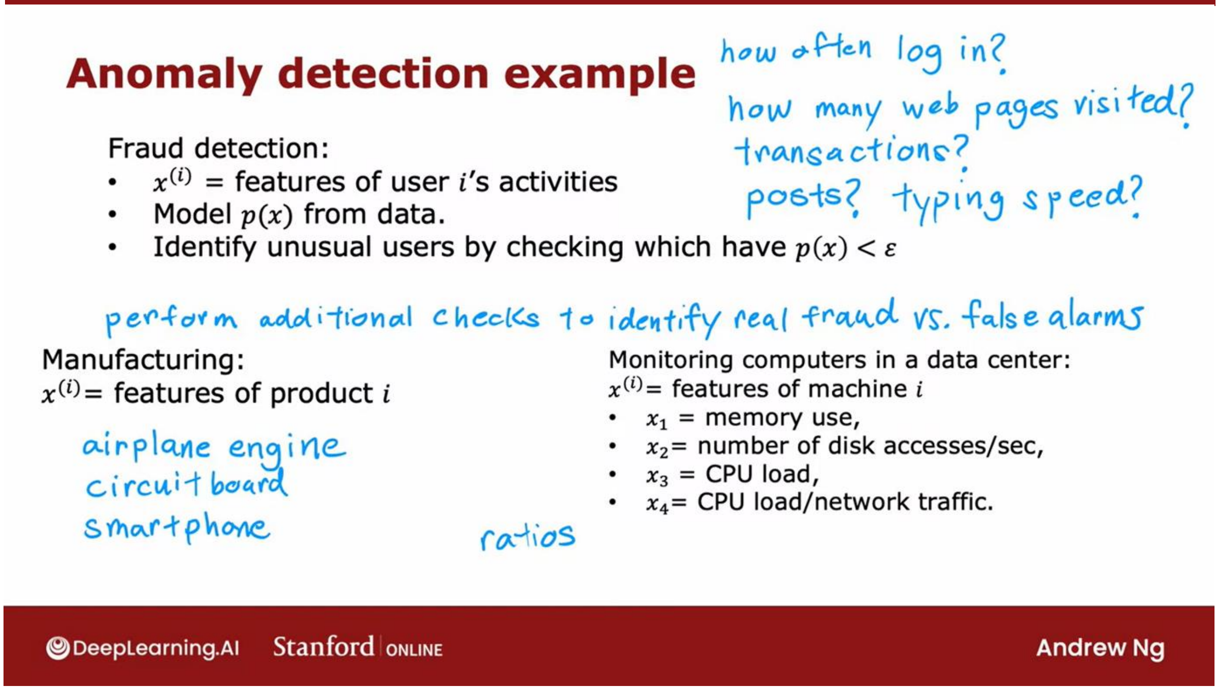
Density estimation

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ probability of x being seen in dataset

Model $p(x)$

Is x_{test} anomalous?

DeepLearning.AI Stanford ONLINE Andrew Ng



Anomaly detection example

how often log in?
how many web pages visited?
transactions?
posts? typing speed?

Fraud detection:

- $x^{(i)}$ = features of user i's activities
- Model $p(x)$ from data.
- Identify unusual users by checking which have $p(x) < \varepsilon$

perform additional checks to identify real fraud vs. false alarms

Manufacturing:
$x^{(i)}$ = features of product i

airplane engine
circuit board
smartphone

Monitoring computers in a data center:
$x^{(i)}$ = features of machine i

- x_1 = memory use,
- x_2 = number of disk accesses/sec,
- x_3 = CPU load,
- x_4 = CPU load/network traffic.

ratios

DeepLearning.AI Stanford ONLINE Andrew Ng

Gaussian (Normal) Distribution (bell-shaped is used too)

In order to apply anomaly detection, we'll need it. In a histogram, while bin width goes to 0 (meaning number of examples goes to infinity), distribution becomes gaussian, normal.

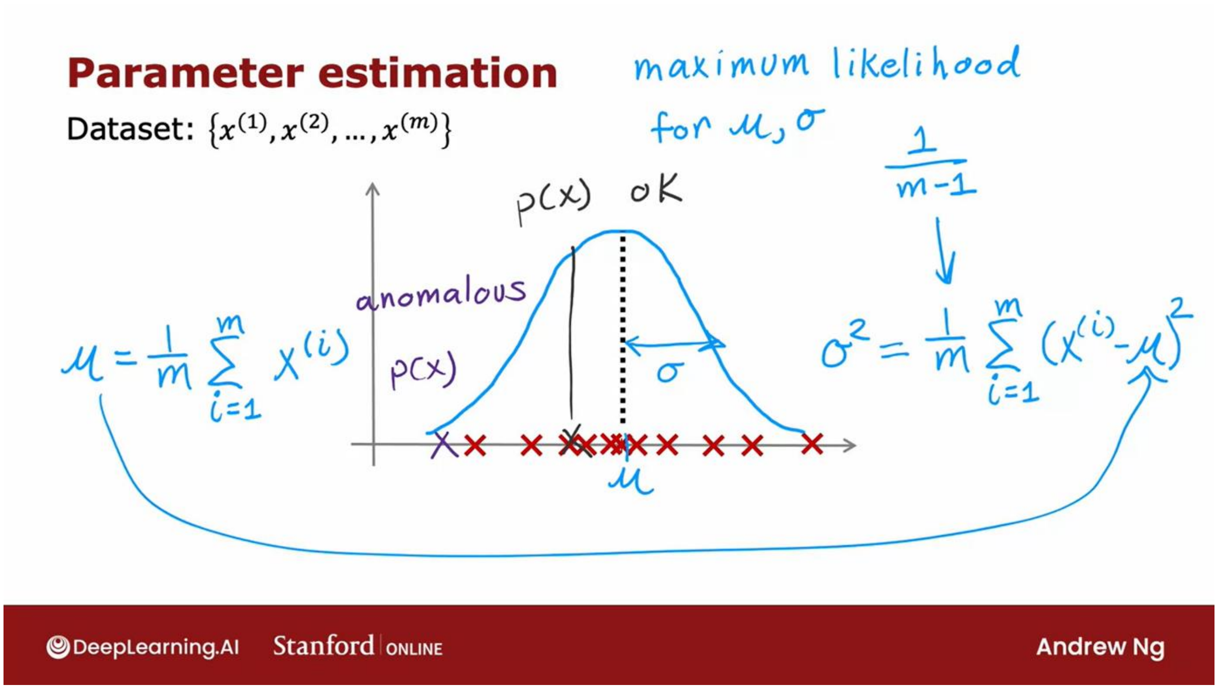
FORMULA:

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

Mu shows you the center. Std (sigma) means how thin your curve is. As you decrease std dev, data centers around the mean.

***Probability always adds up to 1, you area under the graph must be 1. That's why as curve gets thinner, it gets taller.**

While fitting data to normal distribution, what you have to do is to find good choices for mean and std dev. **You find these two values with mathematical operations**.



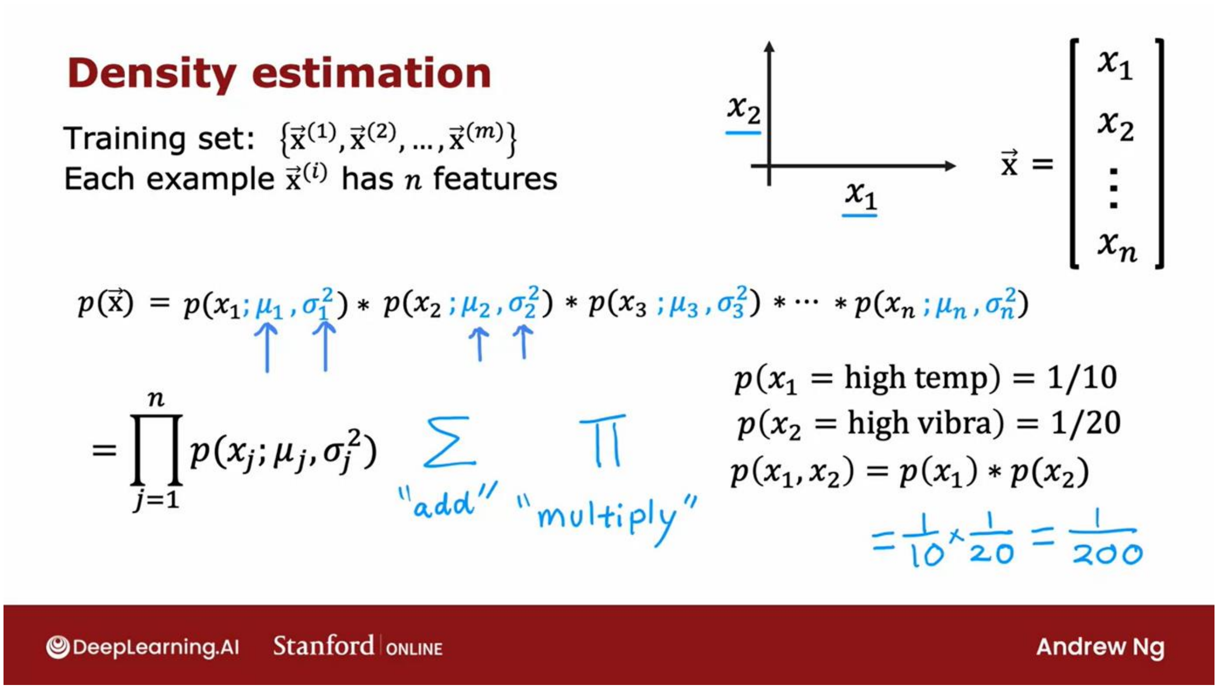
Parameter estimation
Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
maximum likelihood for μ, σ
$\frac{1}{m-1}$
p(x) ok
anomalous
p(x)
σ
$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)^2$
μ
DeepLearning.AI
Stanford ONLINE
Andrew Ng

Note: In statistics, these formulae give you **maximum likelyhood estimates** of mu and sigma. Statisticians prefer 1/(n-1). Andrew: "I always use 1/m."

*In the procedure you just saw, x was just a number, meaning you had only 1 feature. Now we'll apply it in a situation with multiple features.

Anomaly detection algorithm

Model will compute a probability, which is: product of each feature's possibility. To compute these possibilities, we first need to estimate **mean** and **std dev** of every feature. Then we can carry out calculations.



Density estimation

Training set: $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$
Each example $\vec{x}^{(i)}$ has n features

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \cdots * p(x_n; \mu_n, \sigma_n^2)$$

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

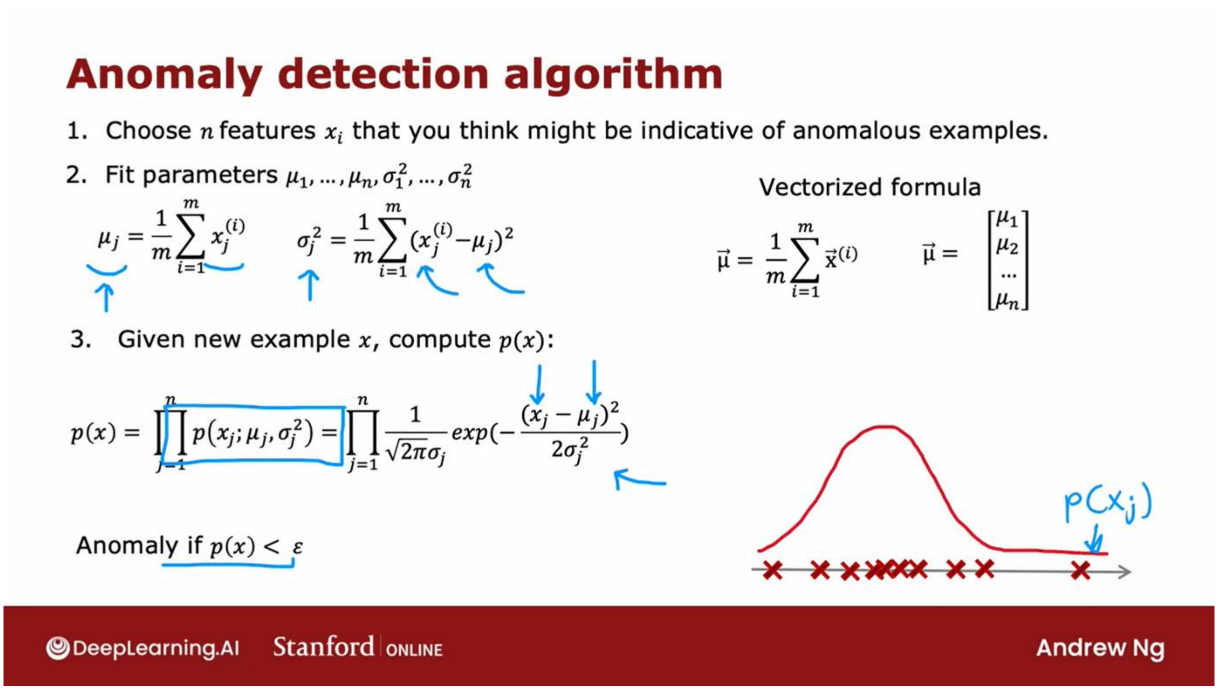
$\sum$ "add" $\prod$ "multiply"

$p(x_1 = \text{high temp}) = 1/10$
$p(x_2 = \text{high vibra}) = 1/20$
$p(x_1, x_2) = p(x_1) * p(x_2)$
$= \frac{1}{10} \times \frac{1}{20} = \frac{1}{200}$

x_1
x_2
DeepLearning.AI
Stanford ONLINE
Andrew Ng

Procedure:

1) Choose n features that might indicate anomalous examples.
2) Fit parameters mu_1 to mu_n and sigma_1 to sigma_n.
3) When given new example, just compute P(x) (which is product of features' probabilities).
4) Decide with a threshold. (**What should that threshold be?)**



Anomaly detection algorithm

1. Choose n features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

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

Vectorized formula

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^{m} \vec{x}^{(i)} \qquad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_n \end{bmatrix}$$

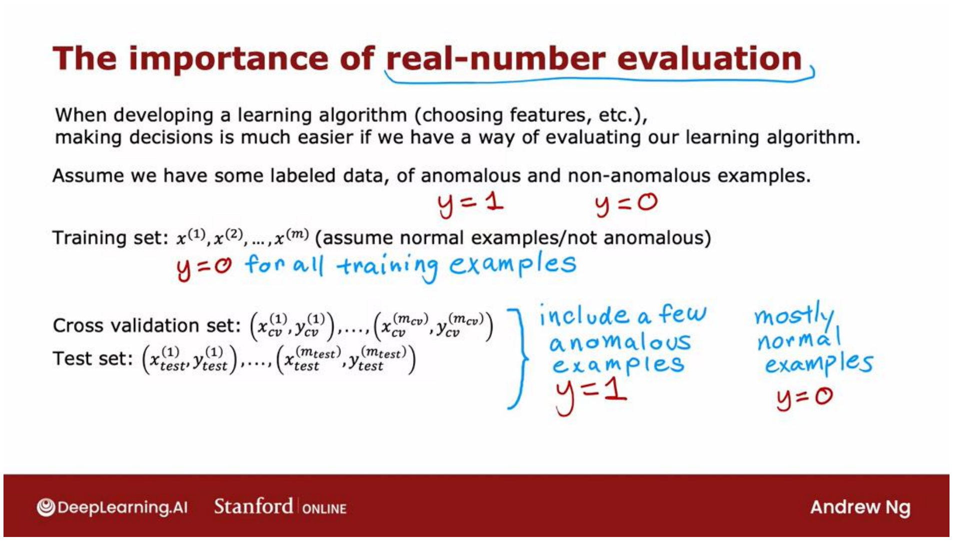
3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} \boxed{p(x_j; \mu_j, \sigma_j^2) =} \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $\underline{p(x) < \varepsilon}$

Developing and Evaluating

If you can evaluate, making decisions becomes much easier. (remember J_train - J_cv situations)



The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labeled data, of anomalous and non-anomalous examples.

$y=1$ $y=0$

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ (assume normal examples/not anomalous)

$y=0$ for all training examples

Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

include a few anomalous examples $y=1$

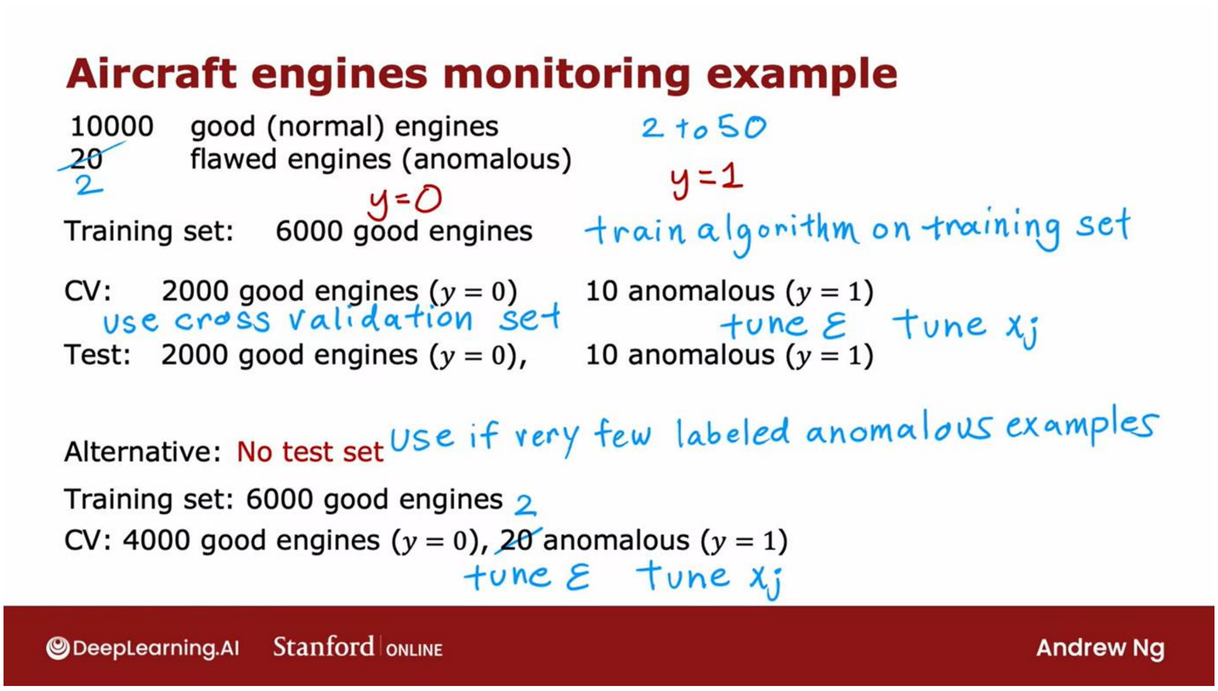
mostly normal examples $y=0$

DeepLearning.AI Stanford ONLINE Andrew Ng

Assume we have some labeled data, maybe acquired through time with production, for cv and test.

- **We always assume that number of normal examples are much more than anomalous ones.**

- **Training set includes normal examples. Then you try to label new ones whether they're normal. Some anomalous examples can splip into training set, it's okay.**



Aircraft engines monitoring example

10000 good (normal) engines

~~20~~ 2 flawed engines (anomalous) 2 to 50 $y=1$

$y=0$

Training set: 6000 good engines train algorithm on training set

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

use cross validation set tune ε tune x_j

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

Alternative: No test set Use if very few labeled anomalous examples

Training set: 6000 good engines

CV: 4000 good engines ($y=0$), ~~20~~ 2 anomalous ($y=1$)

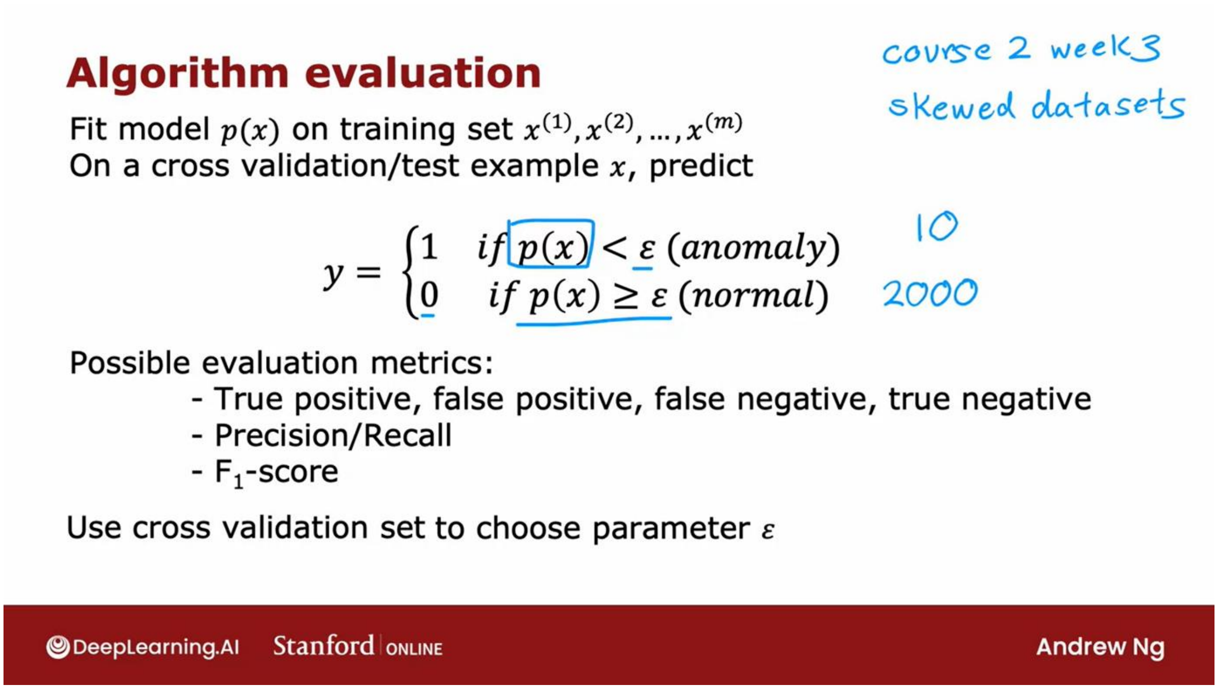
tune ε tune x_j

DeepLearning.AI Stanford ONLINE Andrew Ng

- Objective is to detect anomalous examples in CV set. Depending on how you flag anomalous and normal examples, you can tune value of epsilon (threshold).
- Then test it with test set.
- **About Alternative:** Downside is that after you tune, you don't have a fair way to say if it's accurate. But sometimes it's the best alternative. **THIS APPROACH HAS HIGHER RISK OF OVERFITTING.**

STEPS OF EVALUATION

- Fit the model first.
- On cv set, compute y=0 or y=1 depending on p(x).



Algorithm evaluation

course 2 week3
skewed datasets

Fit model $p(x)$ on training set $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
On a cross validation/test example x, predict

$$y = \begin{cases} 1 & if\ \boxed{p(x)} < \underline{\varepsilon}\ (anomaly) \\ \underline{0} & if\ \underline{p(x) \geq \varepsilon}\ (normal) \end{cases} \quad \begin{matrix} 10 \\ 2000 \end{matrix}$$

Possible evaluation metrics:
- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1-score

Use cross validation set to choose parameter ε

DeepLearning.AI Stanford ONLINE Andrew Ng

- **REMEMBER "SKEWED DATASETS" OPTIONAL VIDEOS. There we used precision-recall and F1 score. A similar approach can be used in terms of evaluation metrics.**

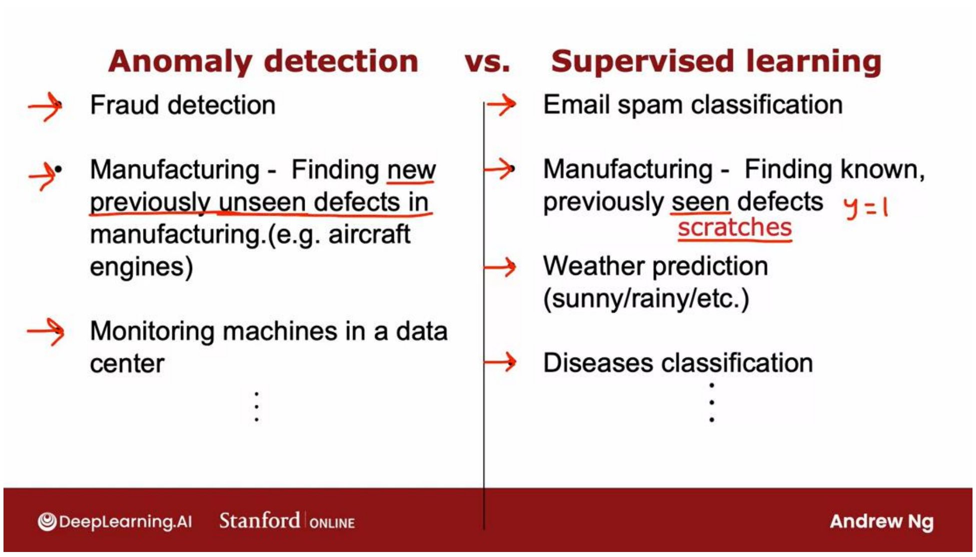
NEXT: If we're labeling a few examples for cv and test, why not use supervised learning entirely?

Anomaly detection vs. supervised learning

Main difference: The way these two algorithms look at the data set is quite different. In supervised learning, you learn from y=1 examples what should be labeled as wrong. So if you've never seen a type of error before, you can't handle it good enough. But anomaly detection, from normal examples (which you have tons of) you learn what is normal. Then you can label anything that's wrong, even if you've never seen it again.

Idea is, future anomalies may look nothing like current anomalies. Can you still identify them?

Examples:



Anomaly detection	vs. Supervised learning
→ Fraud detection	→ Email spam classification
→ Manufacturing - Finding <u>new previously unseen defects in</u> manufacturing.(e.g. aircraft engines)	→ Manufacturing - Finding known, previously <u>seen</u> defects y=1 <u>scratches</u>
→ Monitoring machines in a data center	→ Weather prediction (sunny/rainy/etc.)
⋮	→ Diseases classification
	⋮

DeepLearning.AI Stanford ONLINE Andrew Ng

Choosing what features to use: Tuning the features

While carrying out supervised learning, having some unrelated features is fine, because model sees enough labeled examples to figure out which features matter, which should be rescaled etc.

But for anomaly detection, same does not apply. So you have to be more careful when it comes to feature selection.

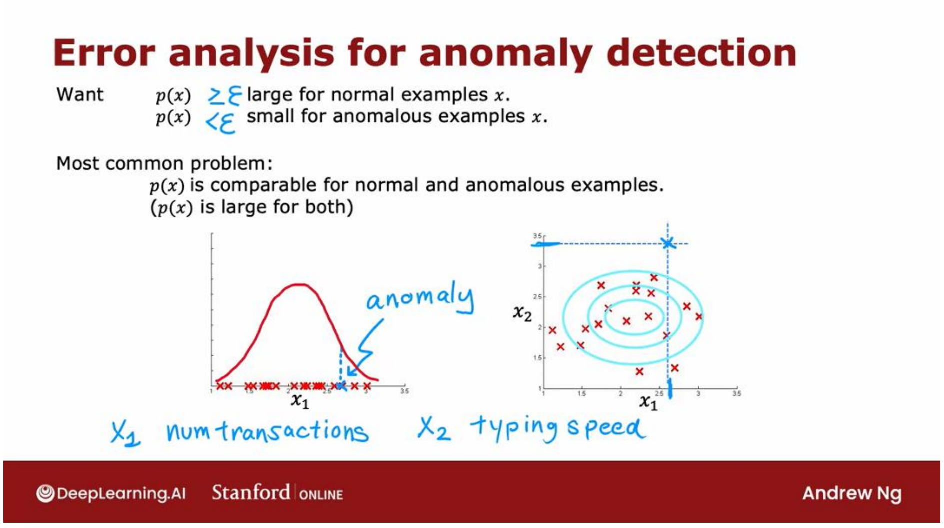
Soooo, tips!

- Have gaussian features or make them be gaussion.
  - You can plot a histogram for the data using **"plt.hist(X, bins=..., color=...,)**
  - If feature is not gaussian, transform it. You could use mathematical operations like log. Then replace that x_j with log(x_j).

Remember: Whatever transform you're applying to training set, you need to apply it to dev and test sets as well.

- You can also carry out "Error analysis for anomaly detection", meaning examining what algorithm fails to do and trying to improve it.
- **Most common problem is:** p(x) can be large for both normal and anomalous examples. A scenario that shows how to handle this problem is shown below.

Consider that you have a distribution. An anomaly is located in a high probability zone. Ask yourself "What made this an anomaly?" Maybe it had normal transactions but it's typing speed was insanely fast. In this case you can add typing speed as a feature, and now that example is considered an anomaly.



Error analysis for anomaly detection

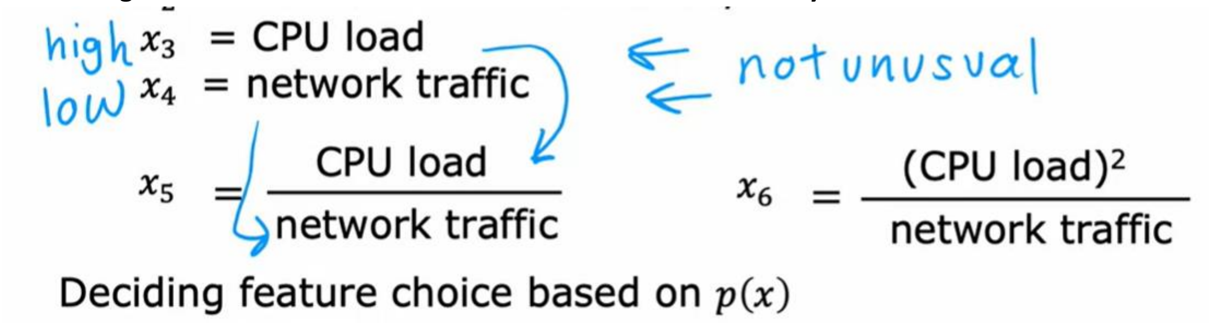
Want $p(x)$ $\geq \varepsilon$ large for normal examples x.
$p(x)$ $< \varepsilon$ small for anomalous examples x.

Most common problem:
$p(x)$ is comparable for normal and anomalous examples.
($p(x)$ is large for both)

DeepLearning.AI Stanford ONLINE Andrew Ng

This procedure is kinda standart. Look at dev set to see what you missed and consider adding new features for these purposes.

- **Combining features to find new ones is also a smart and useful way**.



x_3 = CPU load

x_4 = network traffic

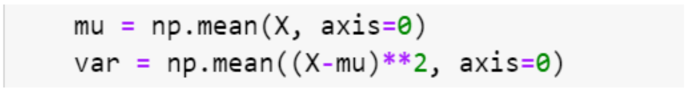
$$x_5 = \frac{\text{CPU load}}{\text{network traffic}} \qquad x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$

Deciding feature choice based on $p(x)$

Objective: P becomes large for normal examples and small for anomalies.

Notes From Practice Lab

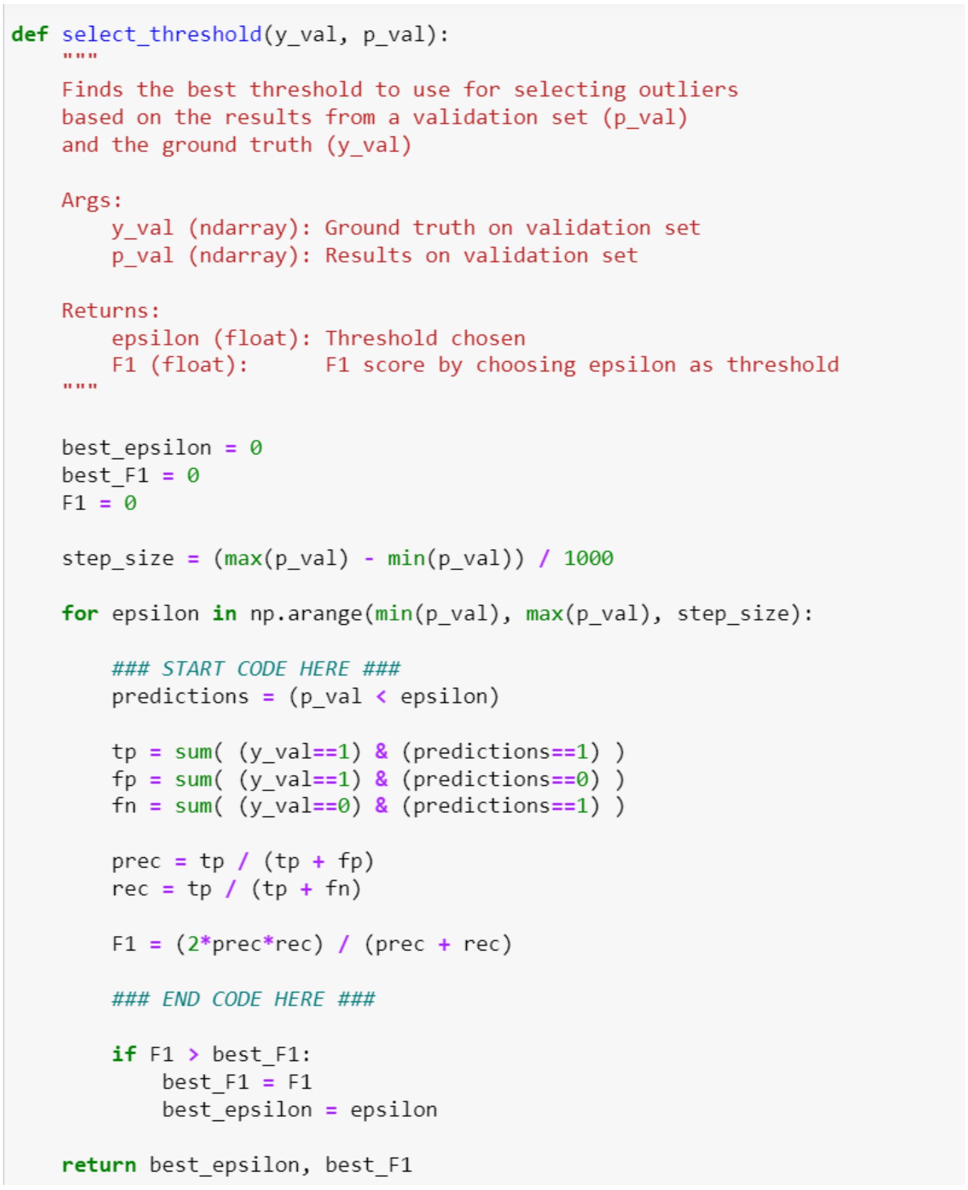
- Way to compute mean and std dev:



```
    mu = np.mean(X, axis=0)
    var = np.mean((X-mu)**2, axis=0)
```

- **REMEMBER:** axis=0 means row wise operations. Axis=1 means column wise.

- Code to select best threshold:



```python
def select_threshold(y_val, p_val):
    """
    Finds the best threshold to use for selecting outliers
    based on the results from a validation set (p_val)
    and the ground truth (y_val)

    Args:
        y_val (ndarray): Ground truth on validation set
        p_val (ndarray): Results on validation set

    Returns:
        epsilon (float): Threshold chosen
        F1 (float):      F1 score by choosing epsilon as threshold
    """

    best_epsilon = 0
    best_F1 = 0
    F1 = 0

    step_size = (max(p_val) - min(p_val)) / 1000

    for epsilon in np.arange(min(p_val), max(p_val), step_size):

        ### START CODE HERE ###
        predictions = (p_val < epsilon)

        tp = sum( (y_val==1) & (predictions==1) )
        fp = sum( (y_val==1) & (predictions==0) )
        fn = sum( (y_val==0) & (predictions==1) )

        prec = tp / (tp + fp)
        rec = tp / (tp + fn)

        F1 = (2*prec*rec) / (prec + rec)

        ### END CODE HERE ###

        if F1 > best_F1:
            best_F1 = F1
            best_epsilon = epsilon

    return best_epsilon, best_F1
```