**Anomaly Detection**

A solution to an unsupervised learning problem, But, has aspects of supervised learning.

First, using our training dataset we build a model

* We can access this model using **p(x)**

This asks, "What is the probability that example x is normal" .

Having built a model

* if p(xtest) < ε --> flag this as an anomaly
* if p(xtest) >= ε --> this is OK
* ε is some threshold probability value which we define, depending on how sure we need/want to be

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.

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.

By **decreasing** epsilon, you allow more data to be passed as OK.

**Applications**

Fraud detection

Manufacturing

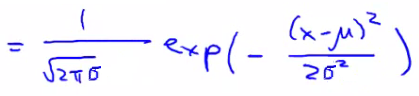
Monitoring computers in data center

**The Gaussian distribution (optional),** Also called the **normal distribution**

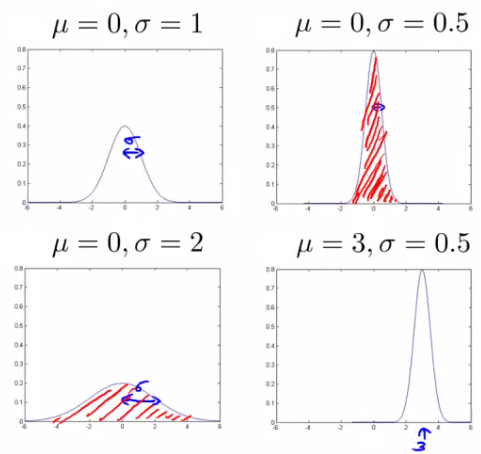
Example

* Say x (data set) is made up of real numbers
  + Mean is μ
  + Variance is σ2
    - σ is also called the **standard deviation** - specifies the width of the Gaussian probability
  + The data has a Gaussian distribution
* Then we can write this ~ *N(*μ,σ2 )
  + ~ means = is distributed as
  + *N* (should really be "script" N (even curlier!) -> means normal distribution
  + μ, σ2 represent the mean and variance, respectively
    - These are the two parameters a Gaussian means

 Gaussian equation is

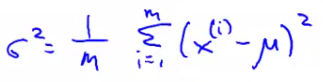
*  P(x : μ , σ2) (probability of x, parameterized by the mean and squared variance)  
  

Area is always the same (must = 1), But width changes as standard deviation changes



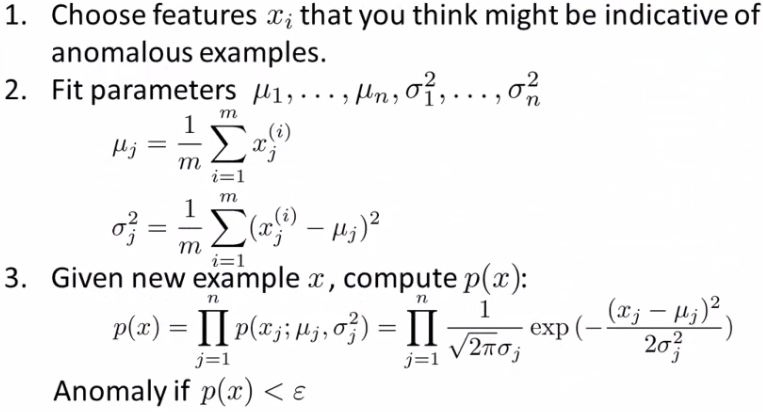
**Parameter estimation problem**

 Estimating μ and σ2

*  μ = average of examples
* σ2 = standard deviation squared   
  
* As a side comment
  + These parameters are the maximum likelihood estimation values for μ and σ2
  + You can also do 1/(m) or 1/(m-1) doesn't make too much difference
    - Slightly different mathematical problems, but in practice it makes little difference

**Anomaly detection algorithm**

**Algorithm**

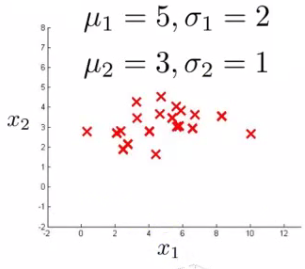
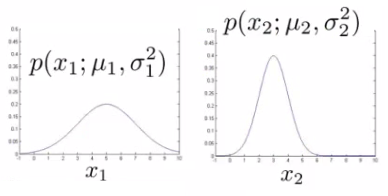
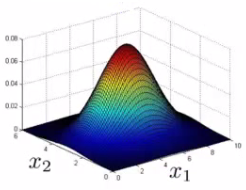


* **1 - Chose features**
  + Try to come up with features which might help identify something anomalous - may be unusually large or small values
  + More generally, chose features which describe the general properties
  + This is nothing unique to anomaly detection - it's just the idea of building a sensible feature vector
* **2 - Fit parameters**
  + Determine parameters for each of your examples μi and σi2
    - Fit is a bit misleading, really should just be "Calculate parameters for 1 to n"
  + So you're calculating standard deviation and mean for each feature
  + You should of course used some vectorized implementation rather than a loop probably
* **3 - compute p(x)**
  + You compute the formula shown (i.e. the formula for the Gaussian probability)
  + If the number is very small, very low chance of it being "normal"

The height of the surface is the probability - p(x)

What this is saying is if you look at the surface plot, all values above a certain height are normal, all the values below that threshold are probably anomalous

**Anomaly detection example**

* x1
  + Mean is about 5
  + Standard deviation looks to be about 2
* x2
  + Mean is about 3
  + Standard deviation about 1
* So we have the following system  
  
* If we plot the Gaussian for x1 and x2 we get something like this  
  
* If you plot the product of these things you get a surface plot like this  
  
  + With this surface plot, the height of the surface is the probability - p(x)
  + We can't always do surface plots, but for this example it's quite a nice way to show the probability of a 2D feature vector
* Check if a value is anomalous
  + Set epsilon as some value
  + Say we have two new data points new data-point has the values
    - x1test
    - x2test
  + We compute
    - p(x1test) = 0.436 >= epsilon (~40% chance it's normal)
      * Normal
    - p(x2test) = 0.0021 < epsilon (~0.2% chance it's normal)
      * Anomalous
  + What this is saying is if you look at the surface plot, all values above a certain height are normal, all the values below that threshold are probably anomalous

**Developing and evaluating and anomaly detection system**

* Often need to make a lot of choices (e.g. features to use)
  + Easier to evaluate your algorithm if it returns a **single number** to show if changes you made improved or worsened an algorithm's performance
* To develop an anomaly detection system quickly, would be helpful to have a way to evaluate your algorithm

What's a good metric to use for evaluation

* y = 0 is very common
  + So classification would be bad
* Compute fraction of true positives/false positive/false negative/true negative
* Compute precision/recall
* Compute F1-score

 Earlier, also had **epsilon** (the threshold value)

*  Threshold to show when something is anomalous
* If you have CV set you can see how varying epsilon effects various evaluation metrics
  + Then pick the value of epsilon which maximizes the score on your CV set
* Evaluate algorithm using cross validation
* Do final algorithm evaluation on the test set

**Anomaly detection vs. supervised learning**

**Anomaly detection**

 **Very small number of positive examples**

* Save positive examples just for CV and test set
* Consider using an anomaly detection algorithm
* Not enough data to "learn" positive examples

 **Have a very large number of negative examples**

* Use these negative examples for p(x) fitting
* Only need negative examples for this

**Many "types" of anomalies**

So anomaly detection doesn't know what they look like, but knows what they *don't* look like

Application and why they're anomaly detection

**Fraud detection, Manufacturing, Monitoring machines in data**

**Supervised learning**

**Reasonably large number of positive and negative examples**

Application

* Email/SPAM classification
* Weather prediction
* Cancer classification

**Choosing features to use**

**Non-Gaussian features**

Plot a histogram of data to check it has a Gaussian description - nice sanity check

Can play with different transformations of the data to make it look more Gaussian

**Error analysis for anomaly detection**

 Good way of coming up with features

 Like supervised learning error analysis procedure

* Run algorithm on CV set
* See which one it got wrong
* Develop new features based on trying to understand *why* the algorithm got those examples wrong

**Multivariate Gaussian distribution**

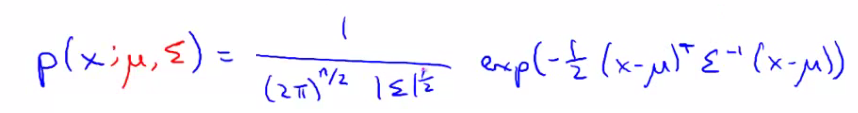
Is a slightly different technique which can sometimes catch some anomalies which non-multivariate Gaussian distribution anomaly detection fails to.

Our function makes probability prediction in concentric circles around the means of both

**Multivariate Gaussian distribution model**

To get around this we develop the **multivariate Gaussian distribution**

* Model p(x) all in one go, instead of each feature separately
  + What are the parameters for this new model?
    - μ - which is an *n* dimensional vector (where n is number of features)
    - Σ - which is an [n x n] matrix - the **covariance matrix**

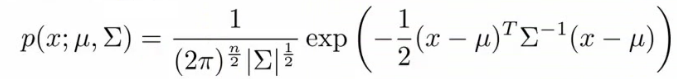
For the sake of completeness, the formula for the multivariate Gaussian distribution is as follows  


* C:\Books\Machine Learning\Machine_learning_complete\Machine_learning_complete\15_Anomaly_Detection_files\Image [22].png = absolute value of Σ (determinant of sigma)
  + This is a mathematic function of a matrix
  + You can compute it in MATLAB using **det(sigma)**

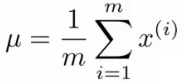
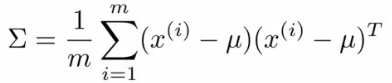
Sigma is sometimes call the identity matrix

For inputs of x1 and x2 the height of the surface gives the value of p(x)

**Applying multivariate Gaussian distribution to anomaly detection**

multivariate Gaussian modeling uses the following equation;  


Which comes with the parameters μ and Σ

* Where
  + μ - the mean (n-dimenisonal vector)
  + Σ - covariance matrix ([nxn] matrix)
*  The formula for estimating the parameters is  
    
  

 Using these two formulas you get the parameters

**Anomaly detection algorithm with multivariate Gaussian distribution**

* **1)** Fit model - take data set and calculate μ and Σ using the formula above
* **2)** We're next given a new example (xtest)
*  Original model corresponds to multivariate Gaussian where the Gaussians' contours are axis aligned
*  i.e. the normal Gaussian model is a special case of multivariate Gaussian distribution

**Original model vs. Multivariate Gaussian**

Original Gaussian model

* Probably used more often
* There is a need to manually create features to capture anomalies where x1 and x2take unusual combinations of values
  + So **need to make extra features**
  + Might not be obvious what they should be
    - This is always a risk - where you're using your own expectation of a problem to "predict" future anomalies
    - Typically, the things that catch you out aren't going to be the things you though of
      * If you thought of them they'd probably be avoided in the first place
    - Obviously this is a bigger issue, and one which may or may not be relevant depending on your problem space
* Much **cheaper computationally**
* **Scales much better** to very large feature vectors
  + Even if n = 100 000 the original model works fine
* **Works well even with a small training set**
  + e.g. 50, 100
* Because of these factors it's used more often because it really represents a optimized but axis-symmetric specialization of the general model

**Multivariate Gaussian model**

* Used less frequently
* **Can capture feature correlation**
  + So no need to create extra values
* **Less computationally efficient**
  + Must compute inverse of matrix which is [n x n]
  + So lots of features is bad - makes this calculation very expensive
  + So if n = 100 000 not very good
* **Needs for m > n**
  + i.e. number of examples must be greater than number of features
  + If this is not true then we have a singular matrix (non-invertible)
  + So should be used only in m >> n
* If you find the matrix is non-invertible, could be for one of two main reasons
  + m < n
    - So use original simple model
  + Redundant features (i.e. linearly dependent)
    - i.e. two features that are the same
    - If this is the case you could use PCA or sanity check your data