Week 7

Support Vector Machine (SVM): optimisation objective

an alternative view of logistic regression, recall

$$\rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

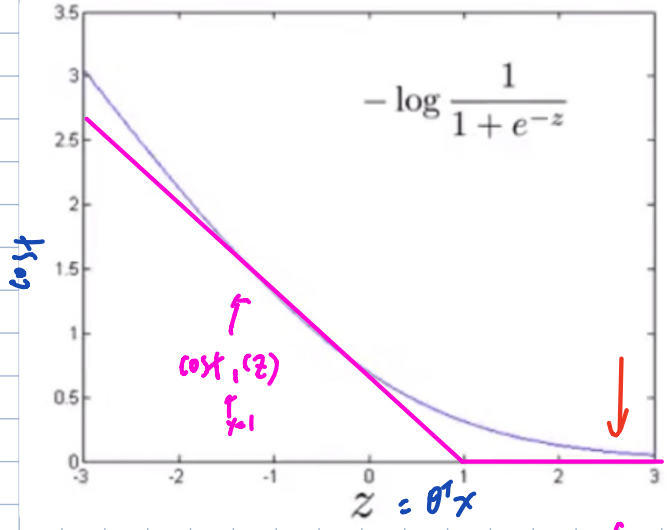
If $y=1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

If $y=0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

→ cost of example: $-(y \log(h_\theta(x)) + (1-y) \log(1-h_\theta(x)))$ ← for one training example $(x^{(i)}, y^{(i)})$

$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y) \log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

If $y=1$ (want $\theta^T x \gg 0$):

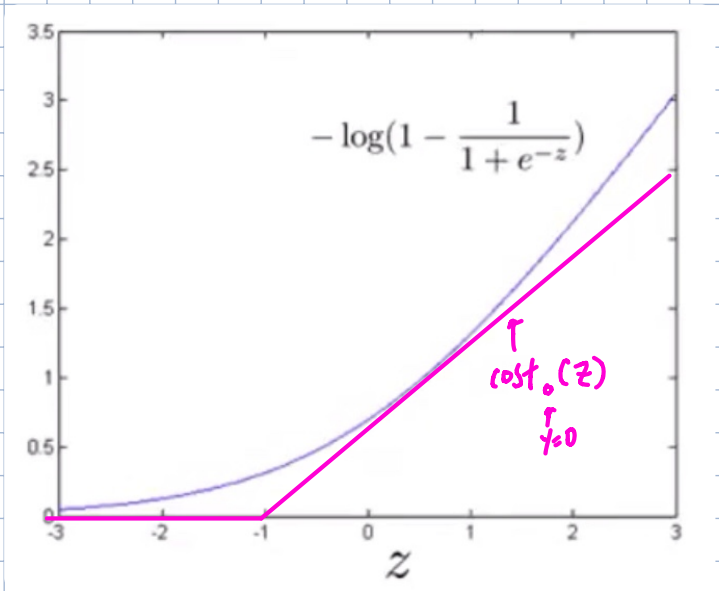


when logistic regression sees a positive example, $(y=1)$, it tries to set $\theta^T x$ to be very large

→ give us some computational advantage

cost function of SVM

If $y=0$ (want $\theta^T x \ll 0$):



logistic regression cost function:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\underbrace{(-\log h_\theta(x^{(i)}))}_{\text{for SVM: } \text{cost}_1(\theta^T x^{(i)})} + (1-y^{(i)})\underbrace{(-\log(1-h_\theta(x^{(i)})))}_{\text{for SVM: } \text{cost}_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

A B

SVM cost function

$$\min_\theta \frac{1}{m} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

convention

$\min_u (u-5)^2+1 \rightarrow u=5$

$\min_u 10\times(u-5)^2+1 \rightarrow u=5$

logistic regression: $A+\lambda B$ — if λ is big, B is high weight

SVM: $CA+B$ — if C is big, B is low weight

$\lambda = 1/C$

SVM cost function

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

unlike logistic regression, SVM doesn't output the probability. SVM just makes a prediction of y being equal to 1 or 0 directly.

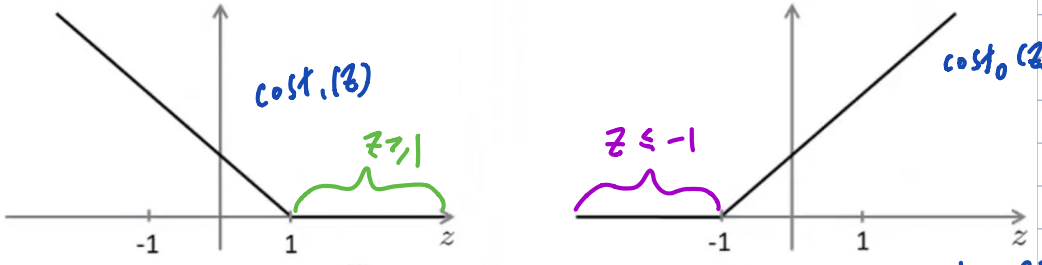
hypothesis:

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

SVM: Large Margin intuition

Support Vector Machine

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$



→ If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

→ If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

previously $\theta^T x \geq 0$ } barely
$\theta^T x < 0$ } right

↓

extra safety factor in SVM

SVM Decision Boundary

consider C = 100,000 (a very large value)

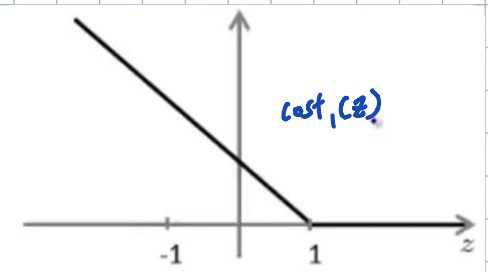
$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

↳ highly motivated = 0
(if C is very large)

whenever $y^{(i)} = 1$:

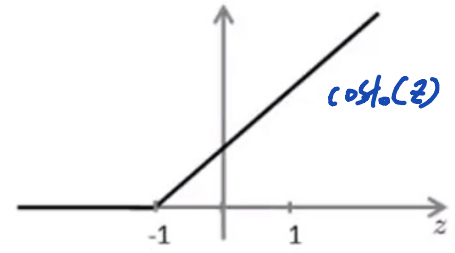
if we want 1st term above $= 0$, we need

$\theta^T x^{(i)} \geq 1$



whenever $y^{(i)} = 0$:

$\theta^T x^{(i)} \leq -1$



therefore, assuming we try to make first term $= 0$, what we left is the optimisation problem:

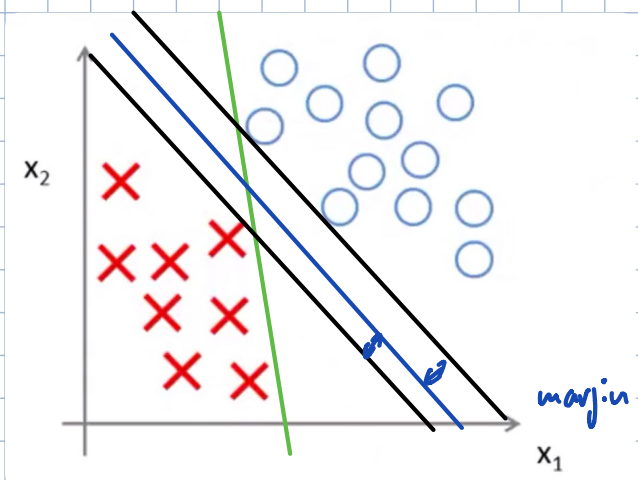
$$\min \quad \cancel{C \times 0} + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

this will be subject to the constraint that

$\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

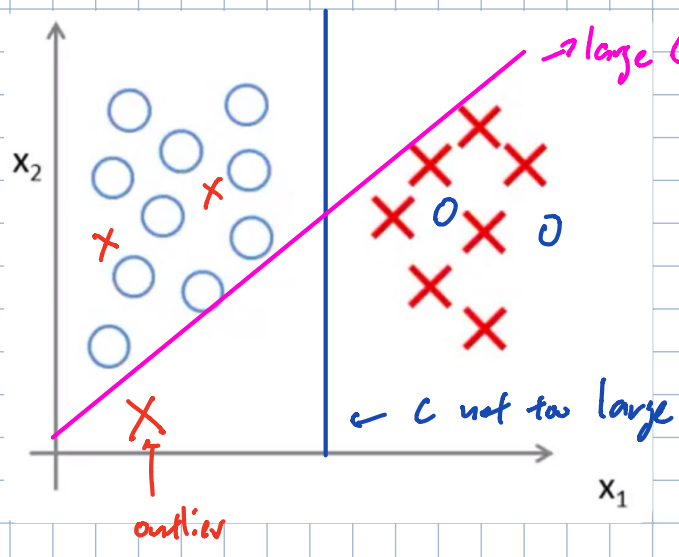
$\theta^T x^{(i)} \leq 1$ if $y^{(i)} = 0$

SVM. Linearly separable case



$\Rightarrow$ large margin classifier

Large margin classifier in presence of outliers



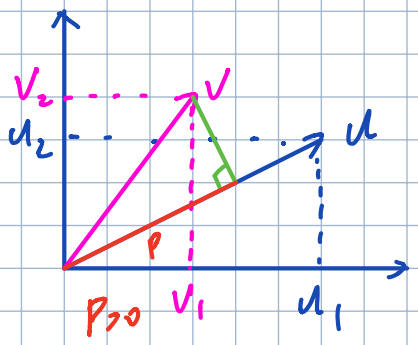
SVM: the maths

Vector Inner Product

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$\|u\|$ = length/norm of vector u

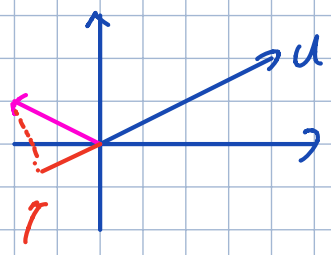
$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$



signed → p = length of projection of v onto u $\in \mathbb{R}$

$$u^T v = p \cdot \|u\| = v^T u$$

$$= u_1 v_1 + u_2 v_2$$



$u^T v = p \cdot \|u\|$

$p < 0$ angle between u and $v > 90°$

SVM Decision Boundary

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2+\theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2+\theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

$$= \|\theta\|$$

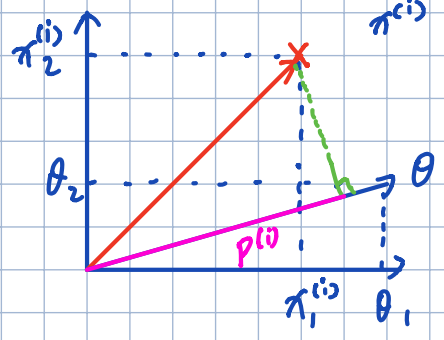
$$\Leftrightarrow \begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\end{bmatrix} \quad \theta_0 = 0$$

subject to $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

$\Rightarrow$ simplification here: $\theta_0 = 0$, $n = 2$ (features)

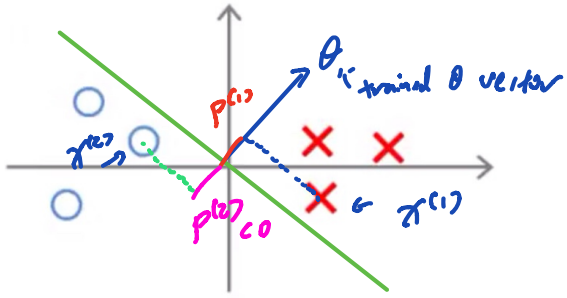
$\theta^T x^{(i)}$

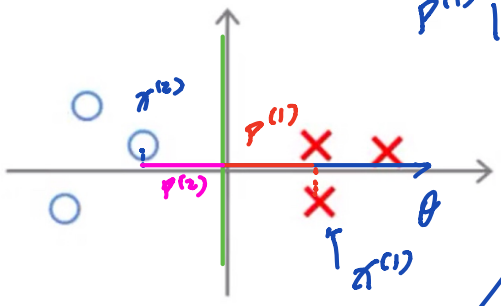


$$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\|$$

$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$

$\theta_0 = 0$ means that decision boundary pass through $(0,0)$





$p^{(1)} \|\theta\| \geq 1$

$\|\theta\|$ can be small

$p^{(i)}$ is small number

but we want $p^{(i)} \cdot \|\theta\| \geq 1$ for $y^{(i)} = 1$

if $p^{(i)}$ is small, then $\|\theta\|$ has to be large

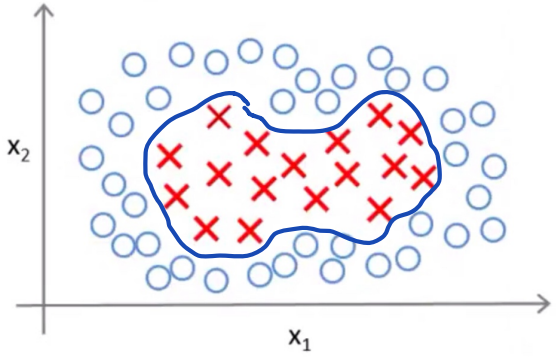
same for

$p^{(2)} \cdot \|\theta\| \leq -1$ for $y^{(i)} = 0$

objective of $\min_\theta$

SVM: Kernels

Non-linear Decision Boundary



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

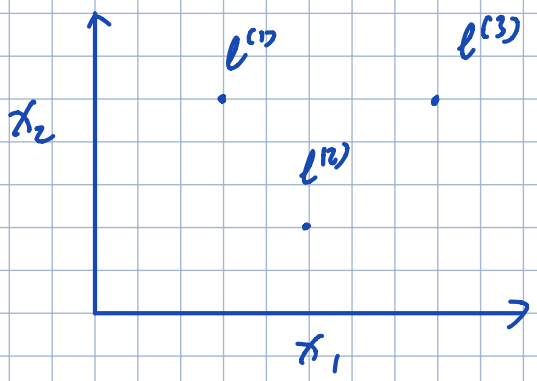
we define $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

where $f_1 = x_1$, $f_2 = x_2$, $f_3 = x_1 x_2$, $f_4 = x_1^2$, $f_5 = x_2^2$

⇒ is there a different/better choice of the features $f_1, f_2, \ldots$?

↳ computationally expensive

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

Given x: define

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}(x, l^{(3)}) = \exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right)$$

Kernel (Gaussian)

↳ $k(x, l^{(i)})$

Kernels and Similarity

$$f_1 = \text{similarity}(x, \ell^{(1)}) = \exp\left(-\frac{\|x - \ell^{(1)}\|^2}{2\sigma^2}\right)$$

$\|x - \ell^{(1)}\|^2 = \sum_{j=1}^{n} (x_j - \ell_j^{(1)})^2$

→ If $x \approx \ell^{(1)}$: $f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$

→ If x is far from $\ell^{(1)}$:

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$$

Each landmark defines a new feature (Given a training example $x^{(i)}$)

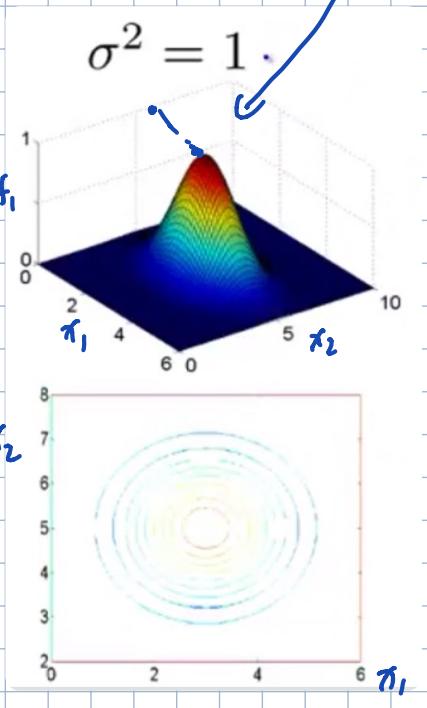
$\ell^{(1)} \rightarrow f_1$

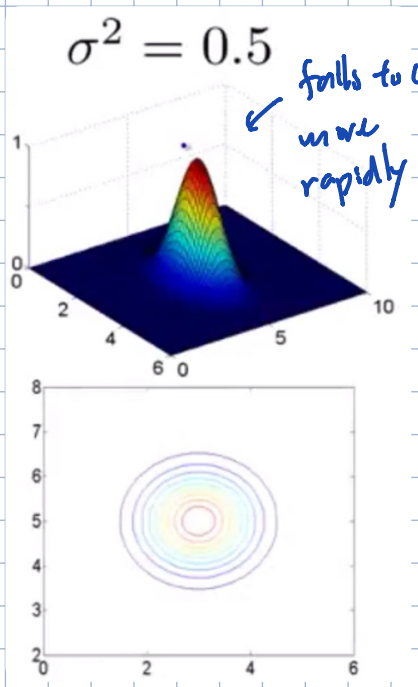
$\ell^{(2)} \rightarrow f_2$

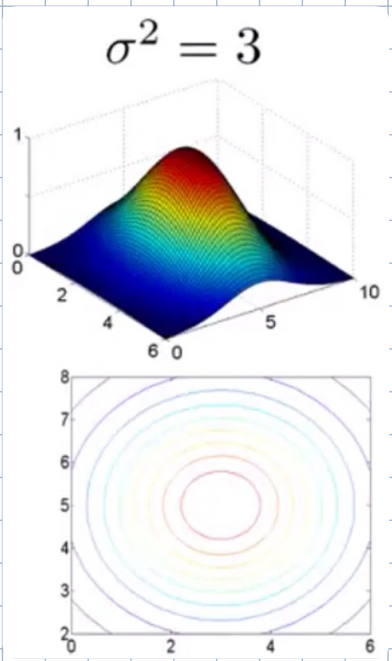
$\ell^{(3)} \rightarrow f_3$

e.g. $\ell^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, $f_1 = \exp\left(-\frac{\|x - \ell^{(1)}\|^2}{2\sigma^2}\right)$

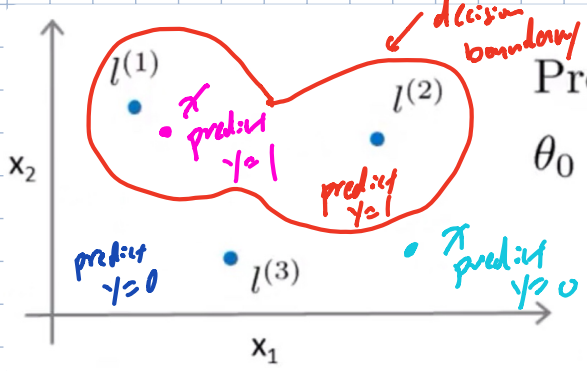
when $x = \begin{bmatrix} 3 \\ 5 \end{bmatrix} \Rightarrow f_1 = 1$







Given the training example x, we are going to compute f_1, f_2, f_3



Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

Say after training we get $\theta_0 = -0.5,\ \theta_1 = 1,\ \theta_2 = 1,\ \theta_3 = 0$

$f_1 \approx 1,\ f_2 \approx 0,\ f_3 \approx 0$

$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0$

$= -0.5 + 1 = 0.5 \geq 0$

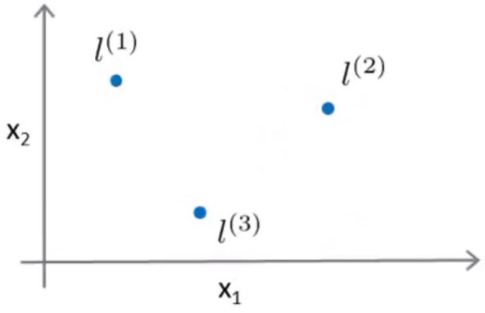
$f_1, f_2, f_3 \approx 0 \Rightarrow \theta_0 + \theta_1 f_1 + \cdots \approx \theta_0 = -0.5 < 0$

Here, if x is near $l^{(1)}$ and $l^{(2)}$ $\Rightarrow$ predict $y=1$ ($\because \theta_3 = 0$)

is far from... $\Rightarrow$ predict $y=0$

Choosing landmarks

Choosing the landmarks



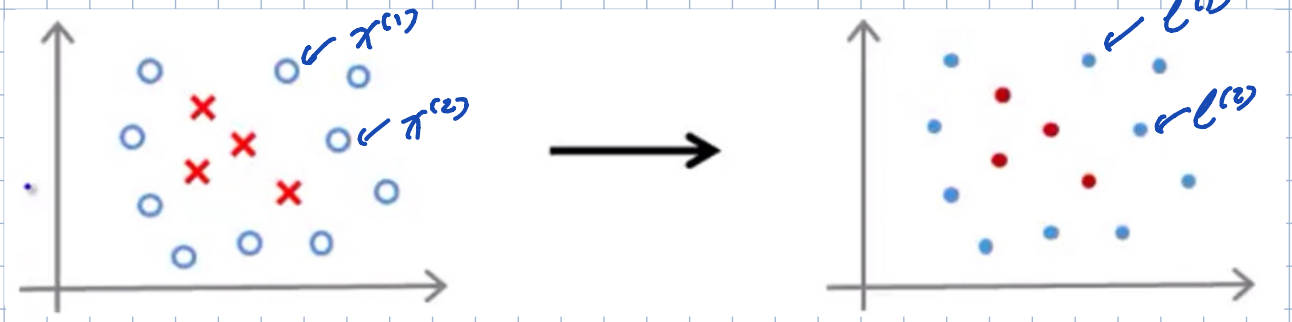
Given x:

$$f_i = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?

In practice, for every training example we have, we're putting landmarks at the same locations as the training examples.



↳ $l^{(1)}$ ⇒ **SVM with kernels**

$\vdots$

$l^{(m)}$

→ Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$,
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$

→ Given example x, (x can be in training/test/CV set)

$f_1 = \text{similarity}(x, l^{(1)})$ ← $x^{(1)}$

$f_2 = \text{similarity}(x, l^{(2)})$

$\dots$

$f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix}$ ($f_0 = 1$) — feature vector

e.g. for training example $(x^{(i)}, y^{(i)})$:

$x^{(i)} \xrightarrow{\text{map}}$ $f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$

$\in \mathbb{R}^{n+1}$ $f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$

$\vdots$ ← $f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = 1$ (for Gaussian kernel) ($x^{(i)}$)

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

$\downarrow$ map

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} = 1 \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \in \mathbb{R}^{m+1}$$

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

predict "$y=1$" if $\theta^T f \geq 0$ (i.e. $\theta_0 f_0 + \theta_1 f_1 + \dots + \theta_m f_m \geq 0$)

(assume we already have a learning set of θ) $\theta \in \mathbb{R}^{m+1}$

training for θ:

$$\min_\theta \; C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

(n = m)

in practice

$$\sum_{j=1}^{n} \theta_j^2 = \theta^T \theta \leftarrow \|\theta\|^2 \qquad \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix} \quad \text{ignore } \theta_0$$

↳ $\theta^T M \theta$ ← rescaled

↳ able to scaled to much bigger training set

why not kernels idea to other algorithms (logistic regression)?

we can, but computational tricks that apply for SVM don't generalise well to other algorithms. (run very slow!)

SVM parameters

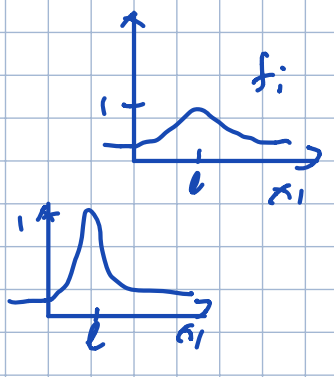
C ($= 1/\lambda$): large C: low bias, high variance (small λ), overfit

small C: high bias, low variance (large λ), underfit

σ^2: large σ^2: feature f_i vary more smoothly. high bias, low variance

(Gaussian kernel)

small σ^2: feature f_i vary less smoothly low bias, high variance



SVM in practice: using an SVM

→ use SVM library to solve for θ

→ hyper parameter:

↳ C

↳ Kernel (similarity function)

e.g. No Kernel ("linear kernel")

↳ predict "y=1" if $\theta^T x \geq 0$

↳ use when n is large, m is small

e.g. Gaussian Kernel:

$$f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right), \text{ where } l^{(i)} = x^{(i)}$$

Need to choose σ^2

↳ if n is small, m is large

↳ implement Kernel (Similarity) functions:

function $f = $ kernel(x_1, x_2)

$$f = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

return

* Note: Do perform feature scaling before using Gaussian Kernel.

Other choice of Kernel

Not all similarity functions similarity(x, l) make valid kernels. (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM package's optimisation runs correctly, and do not diverge.)

→ polynomial Kernel: $K(x, l) = (x^T l)^2$ or $(x^T l)^3$ ← dot product

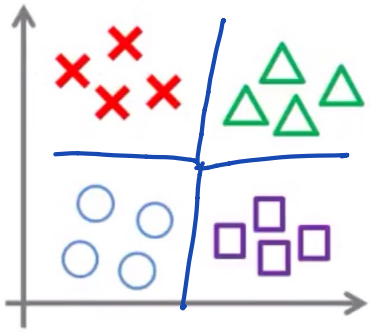
$(x^T l + 5)^2 \Rightarrow (x^T l + \text{const})^{\text{degree}}$

↳ perform worse than Gaussian most of the time

↳ x, l strictly not negative

→ string kernel, chi-square kernel, histogram intersection kernel, ...

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

Pick class i with largest $(\theta^{(i)})^T x$

↑ ↑
y=1 y=2

Logistic regression vs SVMs

Say n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples.

if n is large (relative to m): $n \geq m$

→ use logistic regression, or SVM w/o kernel ("linear kernel")

if n is small, m is intermediate:

→ use SVM with Gaussian kernel

if n is small, m is large

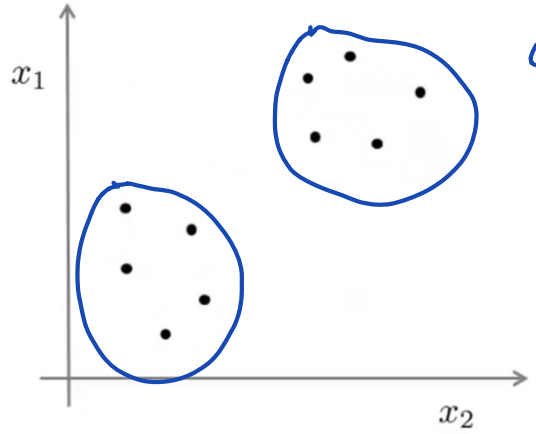
→ create/add more feature, then use logistic regression / SVM w/o kernel.

→ Neural Network likely to work well for most of these settings, but may be slower to train.

SVM optimisation problem is a convex optimisation problem
↳ can always find global minimum.

Week 8

Unsupervised Learning



clustering algorithm

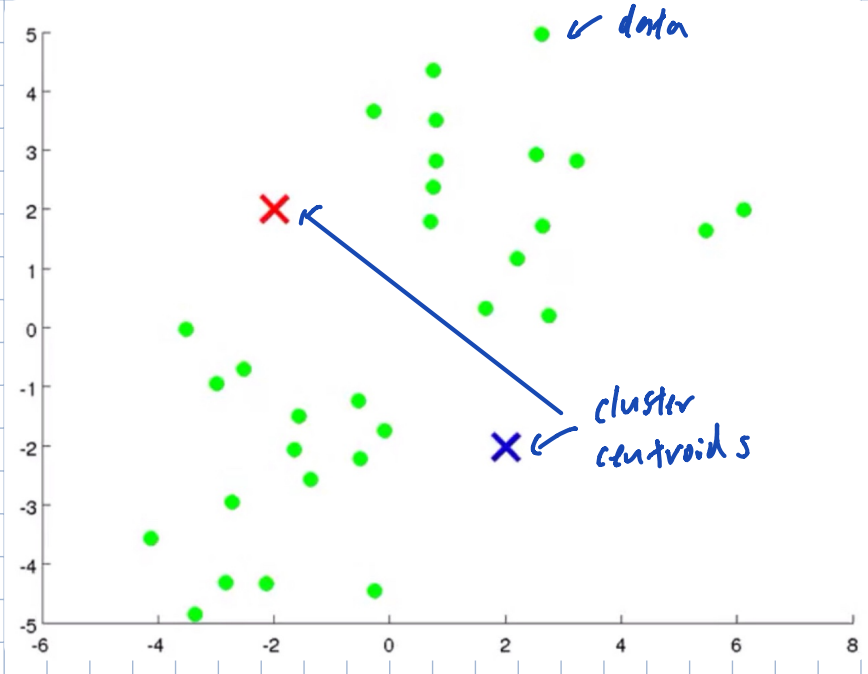
Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$ no $y^{(i)}$ in training set

No labels

⇒ algorithm suppose to find some structure for us

Unsupervised Learning Problem: Clustering

Clustering: K-means algorithm



e.g.

→ randomly initialise 2 points called cluster centroid

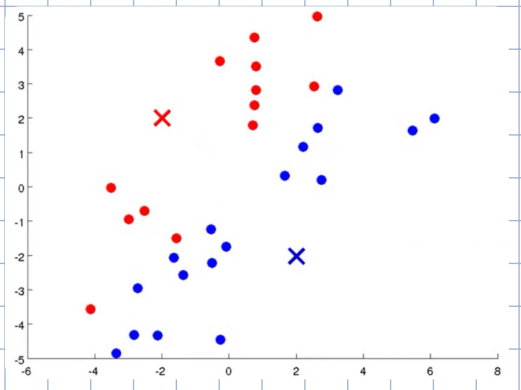
K-mean is an iterative algorithm and it does 2 things:

→ cluster assignment

→ move centroid

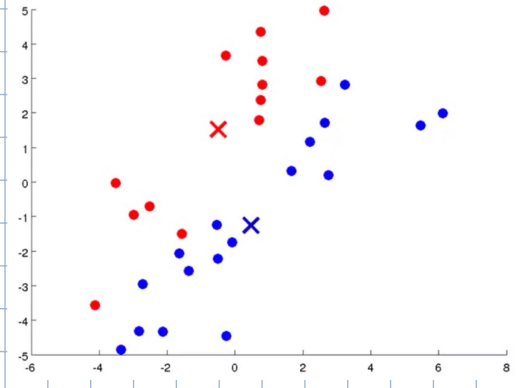
Cluster assignment

The algorithm will go through each of examples, depending on whether it is closer to the red or the blue cluster centroid, it is going to assign each of the data points to one of the cluster centroids.

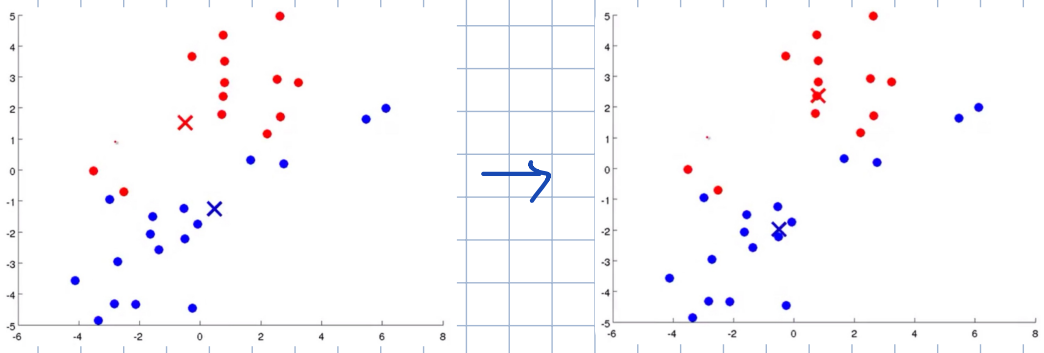


Move centroids

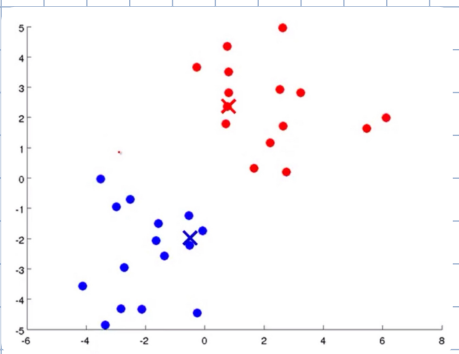
The 2 cluster centroid will be moved to the average of the points colored the same color.



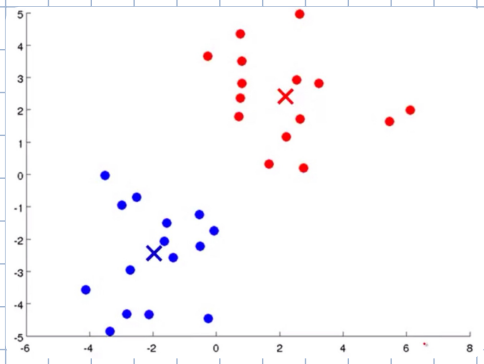
These 2 step repeats:



cluster assignment → move centroids



cluster assignment



move centroids

K-means algorithm

→ takes 2 inputs:

1. K (number of clusters)
2. training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

→ Randomly initialise K cluster centroids $\mu_1, \mu_2, \ldots \mu_K \in \mathbb{R}^n$

Repeat {

cluster assignment {

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$ i.e. $\min_k \|x^{(i)} - \mu_k\|^2$ (find k) $\Rightarrow c^{(i)}$

($c^{(i)}$ is a number from 1 to K)

}

move centroid {

for $k = 1$ to K

μ_k := average (mean) of points assigned to cluster K

}

}

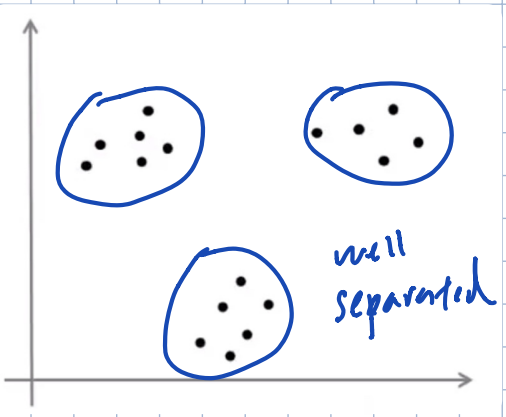
If there is no point in cluster, eliminate the cluster centroid

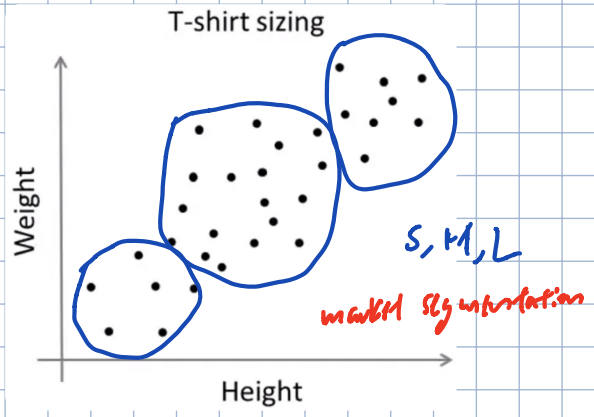
e.g. cluster 2 has training example $x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)}$

↳ $c^{(1)} = 2, c^{(5)} = 2, c^{(6)} = 2, c^{(10)} = 2$

↳ $\mu_2 = \left[\frac{x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}}{4}\right] \in \mathbb{R}^n$

K-means for non-separated clusters





not so well separated

K-means: optimisation objective

For supervised learning, they all have an optimisation objective, or some cost function that algorithm was trying to minimise.

Same for K-means.

While K-means is running, it keeps track of 2 sets of variables.

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

→ μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

one more notation:

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned. e.g. if $x^{(i)} \to 5$ (i.e. assigned to cluster 5)

$c^{(i)} = 5$

↳ $\mu_{c^{(i)}} = \mu_5$

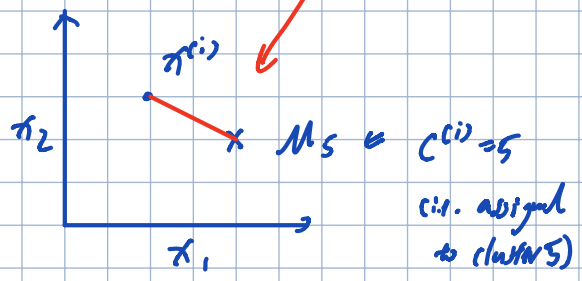
Optimisation objective:

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

$\rightarrow \min_{c^{(1)}, \dots, c^{(m)}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots \mu_K)$

$\rightarrow \mu_1, \dots, \mu_K$

try to find $c^{(i)}$, μ_k that minimise J.



This cost function J is sometimes called Distortion.

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

cluster assignment steps: minimise $J(\dots)$ wrt $c^{(1)}, c^{(2)}, \dots, c^{(m)}$ (holding $\mu_1, \dots, \mu_K$ fixed)

for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}

move centroid steps: minimise $J(\dots)$ wrt $\mu_1, \dots \mu_K$

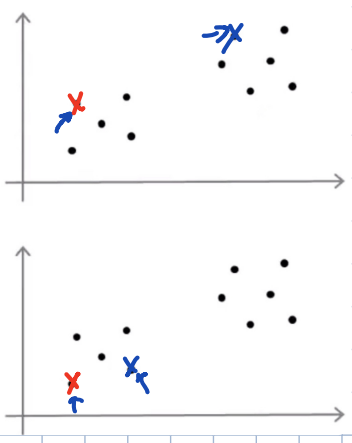
K-means: Random Initialisation

Random initialization

Should have $K < m$

Randomly pick K training examples.

Set $\mu_1, \dots, \mu_K$ equal to these K examples.



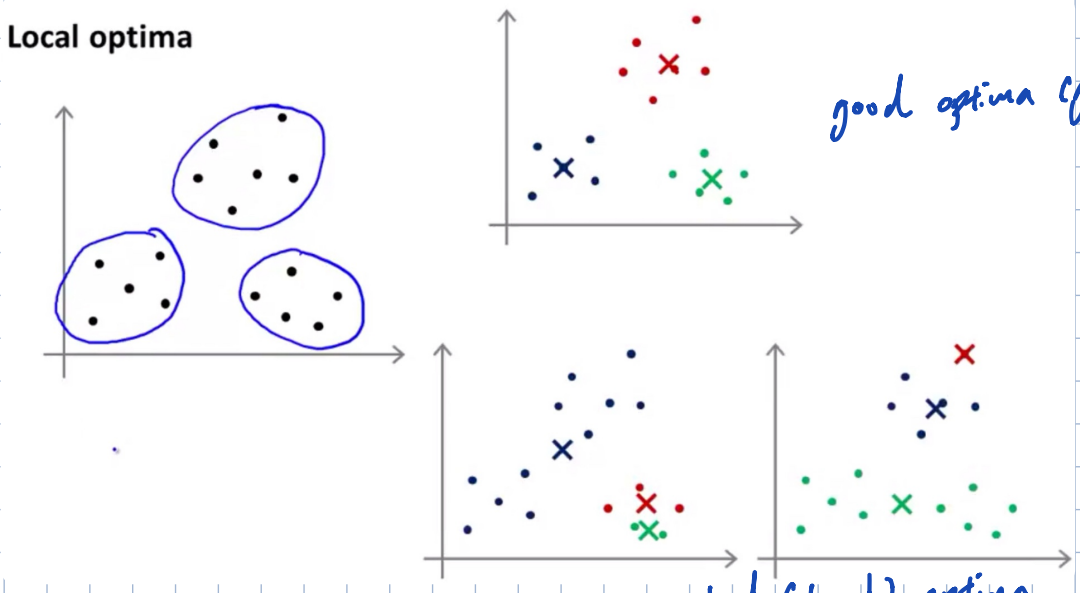
e.g. say $K=2$, pick 2 examples

$\Rightarrow \mu_1 = x^{(i)}$

$\mu_2 = x^{(j)}$

$\Rightarrow$ K-means can end up in different solutions depend on how initialisation goes.

Local optima



⇒ we can try multiple random initialisation

Random initialisation

For $i = 1$ to 100 {

Randomly initialise K-means.

Run K-means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$.

Compute cost function (distortion)

$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

}

Pick clustering that gave the lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

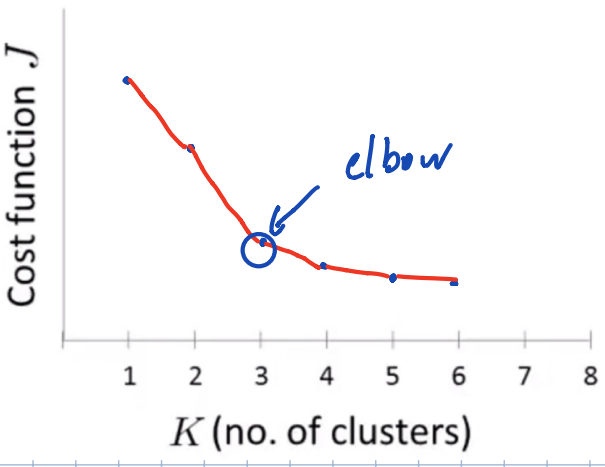
⇒ Usually for small K (e.g. 2 – 10), random initialisation a few times might help. If large K, then not so much.

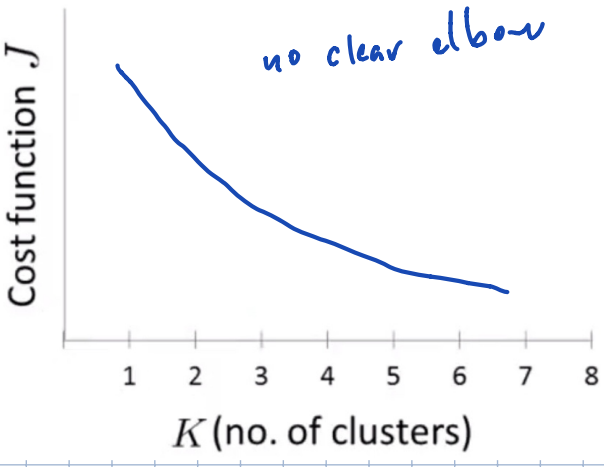
K-means: choosing the number of clusters

Commonly choose K by hand (through visualisation of data etc)

It is often ambiguous how many clusters there are in the data

Elbow Method





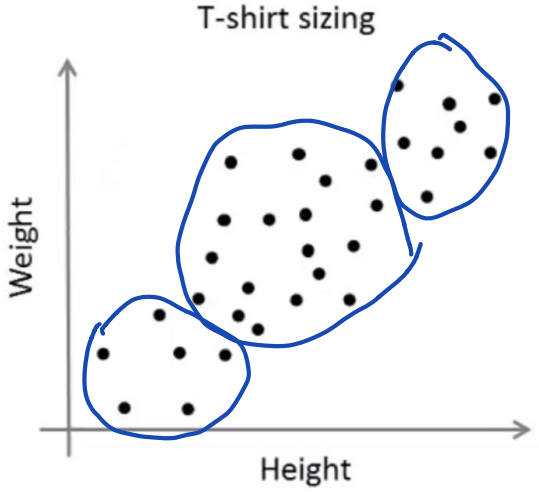
Choosing the value of K

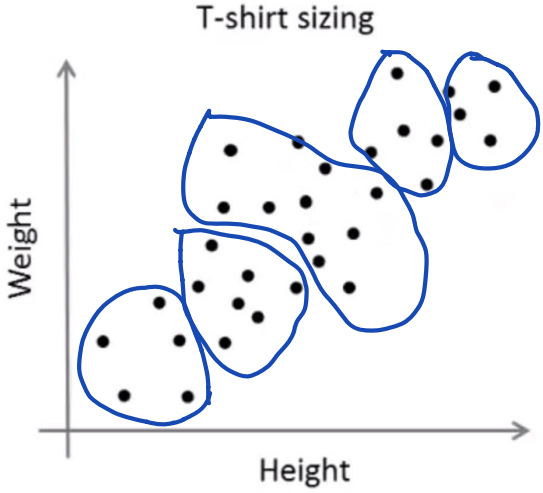
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

K=3 S, M, L

K=5 XS, S, M, L, XL

E.g.

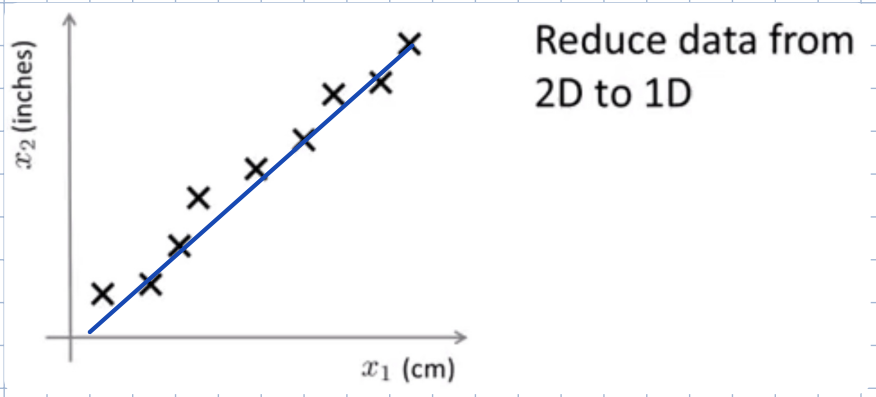




Unsupervised Learning Problem: Dimensionality Reduction

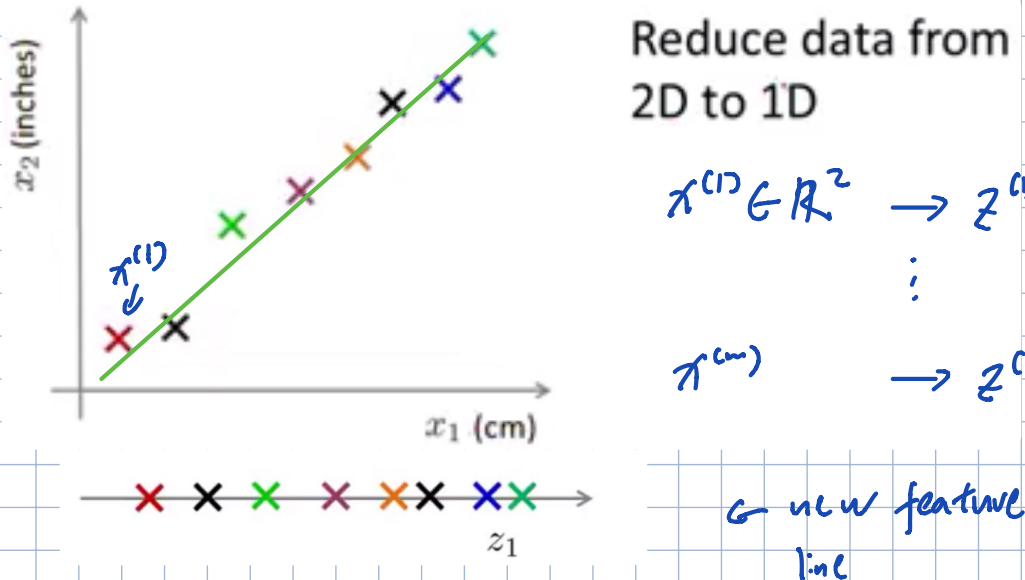
Motivation: Data Compression

2D $\longrightarrow$ 1D



Lets say a dataset with many features, unknown to us, 2 of the features are length in different unit. This gives a highly redundant representation $\Rightarrow$ reduce the data (plotted above) to 1 D.

(Reasons that these 2 features are not on a perfectly line is due to round-off during data collection/processing)



$x^{(1)} \in \mathbb{R}^2 \longrightarrow z^{(1)} \in \mathbb{R}^1$

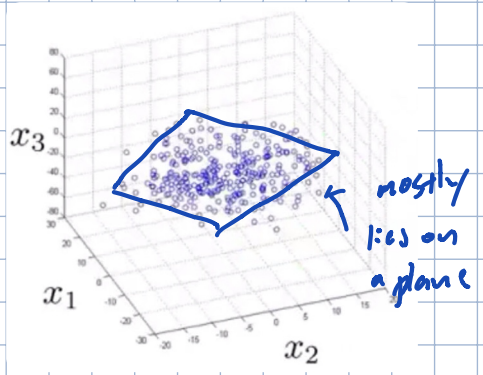
$\vdots$

$x^{(m)} \longrightarrow z^{(m)}$

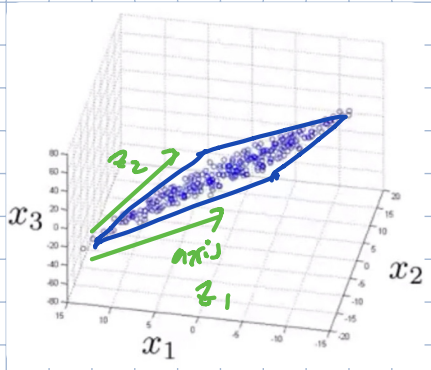
$\leftarrow$ new feature z_1 on the green line

more specifically here, by dimensionality reduction, we would find the direction where the data seems to lie (green line), and project all the data on that line.

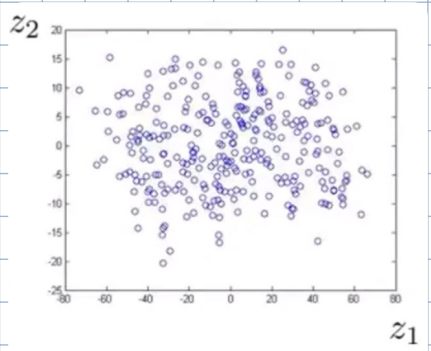
3D ⟶ 2D



$x^{(i)} \in \mathbb{R}^3$



project all the data on a plane



$z^{(i)} \in \mathbb{R}^2$

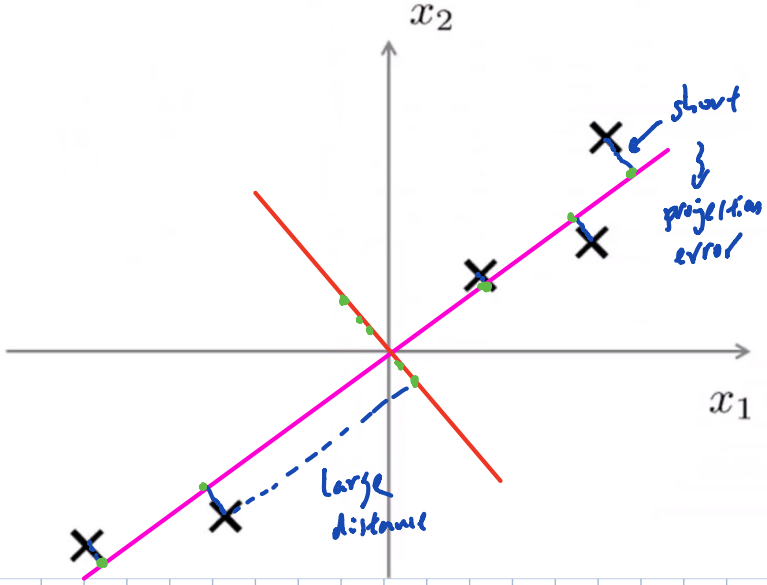
$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \Rightarrow z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$

Motivation: Data Visualisation

Instead of $x^{(i)} \in \mathbb{R}^{50}$, we can have a different feature representation we can plot them out in $\mathbb{R}^2$. (new feature: $z \in \mathbb{R}^2$)

Dimensionality Reduction: Principal Component Analysis (PCA)

Principal Component Analysis (PCA) problem formulation

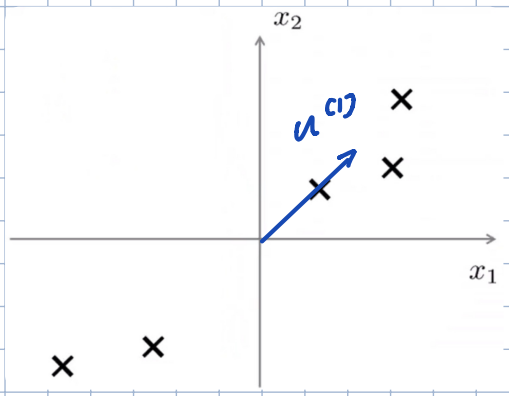


Say we have data set $x \in \mathbb{R}^2$ and would like 2D → 1D

PCA will try to find the projection plane that minimize projection error

Before Applying PCA, it is standard practice to first perform mean normalisation at feature scaling so that the feature x_1 and x_2 should have 0 mean, and have comparable range of values.

Goal of PCA: Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimise the projection error.

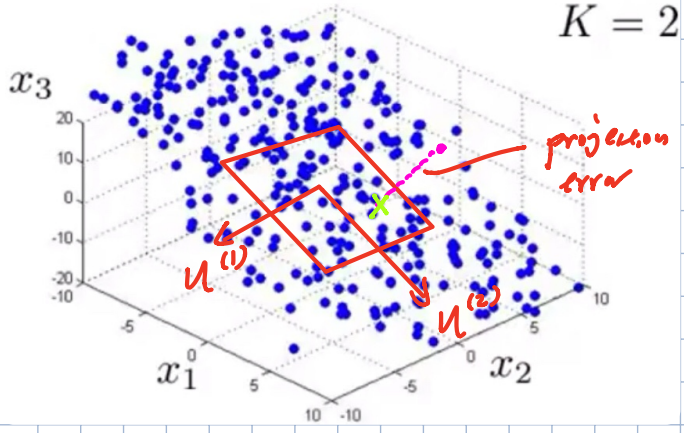


whether $u^{(1)}$ or $-u^{(1)}$ found by PCA does not matter.

More generally:

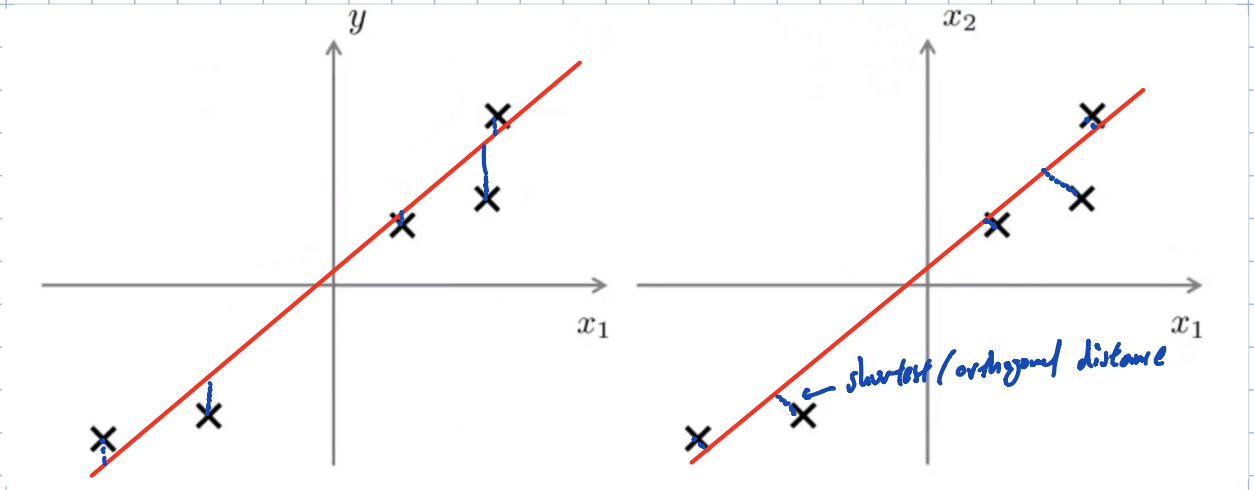
Reduce from n-dimension to K-dimension: Find K vectors $u^{(1)}, u^{(2)}, \ldots, u^{(K)}$ onto which to project the data, so as to minimise the projection error.

$3D \to 2D$
$K = 2$



i.e. project data onto the linear subspace spanned by this set of $u^{(K)}$ vectors.

PCA is NOT linear regression:



linear regression

$x^{(i)} \longrightarrow y^{(i)}$

PCA

no $y^{(i)}$

PCA: algorithm

Before applying PCA, there is a data pre-processing step we should always do.

Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

→ pre processing (feature scaling / mean normalisation):

↳ $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$ (get mean of feature x_j, across all m samples)

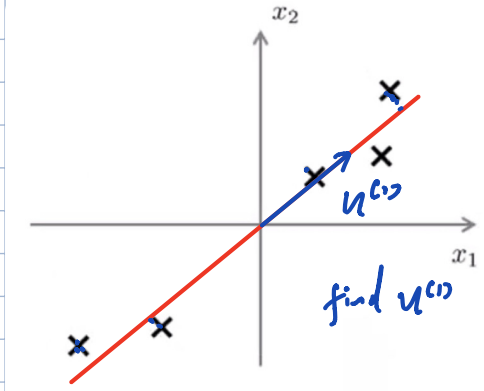
↳ replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$ (all samples's x_j feature − mean)
(each feature is zero mean)

↳ If different features on different scales (e.g. x_1 = size of house, x_2 = number of bedrooms), scale feature to have comparable range of values.

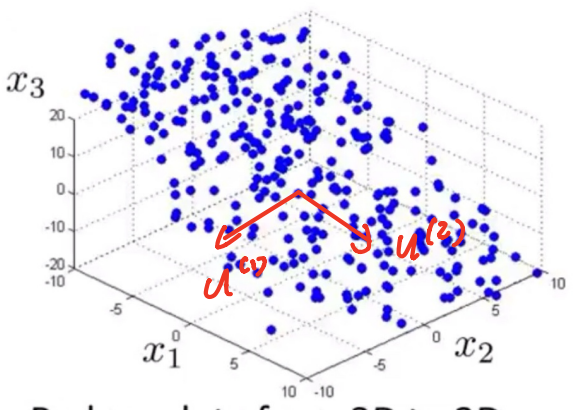
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

s_j = std deviation of x_j

Recall PCA does:



Reduce data from 2D to 1D



Reduce data from 3D to 2D

$x^{(i)} \in \mathbb{R}^2 \longrightarrow z^{(i)} \in \mathbb{R}$

$x^{(i)} \in \mathbb{R}^3 \longrightarrow z^{(i)} \in \mathbb{R}^2$

$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

PCA needs to compute:

① $u^{(1)}, \ldots, u^{(k)}$

② $z^{(1)}, \ldots, z^{(m)}$

PCA algorithm

Reduce data from n-dimensions to k-dimensions

↳ Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \underbrace{(x^{(i)})}_{n \times 1} \underbrace{(x^{(i)})^T}_{1 \times n} \qquad \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \; [x_1 \cdots x_n]$$

↳ compute "eigenvector" of matrix Σ:

$$[U, S, V] = svd(Sigma)$$

singular value decomposition

Σ: $n \times n$ matrix

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

n columns, k rows

↳ select the first k vectors $\Rightarrow u^{(1)}, \ldots u^{(k)}$

Next we need find the z: $x \in \mathbb{R}^n \longrightarrow z \in \mathbb{R}^k$

↳ construct a matrix use the 1st K columns of U.

$$U_{reduced} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}_{n} \quad n \times K \text{ matrix}$$

$$z^{(i)} = U_{reduced}^{T} x^{(i)} = \begin{bmatrix} -(u^{(1)})^T- \\ -(u^{(2)})^T- \\ \vdots \\ -(u^{(k)})^T- \end{bmatrix} \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$$

for 1 example

$K \times n$ $\quad$ $n \times 1$

$K \times 1$

$z \in \mathbb{R}^K$

$z_j = (u^{(j)})^T x$

Summary

→ After mean normalisation (ensure every feature has zero mean) and optionally feature scaling!

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

[U, S, V] = svd(Sigma);

Ureduce = U(:, 1:K);

z = Ureduce' * x; $x \in \mathbb{R}^n$ ~~$x_0 = 1$~~

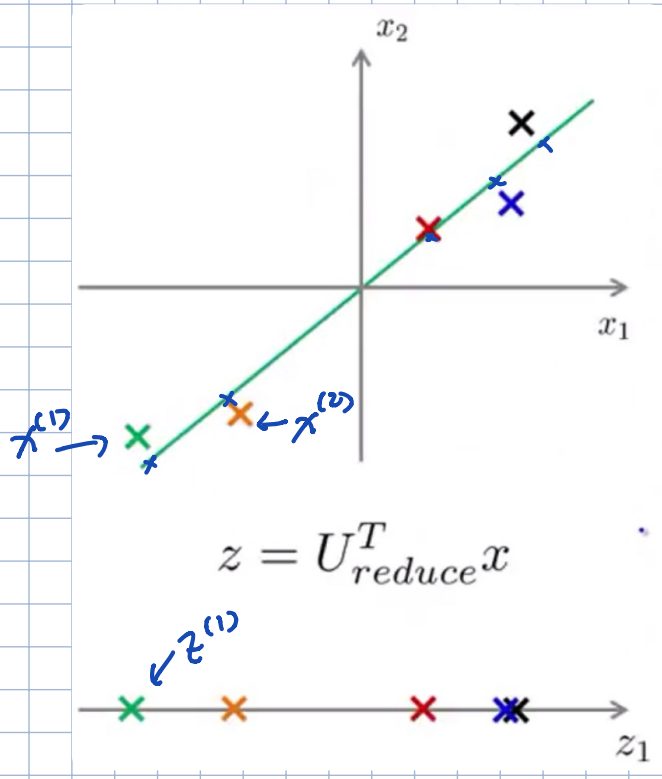
if $x = \begin{bmatrix} -x^{(1)T}- \\ -x^{(m)T}- \end{bmatrix}$ $\quad x' = \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix}$

vectorised

sigma = (1/m) * X' * X;

* This procedure gives the projection of data onto K-dimension subspace that actually minimise the squared projection error.

PCA: Reconstruction from Compressed Representation



Given points on z, we would like to recover points in $\mathbb{R}^2$.

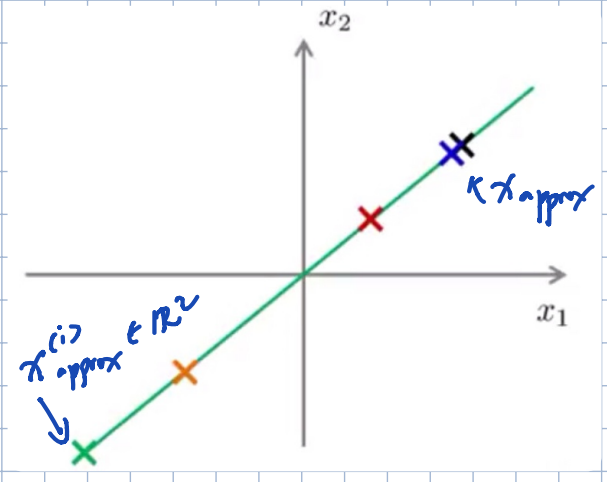
i.e. $z \in \mathbb{R} \longrightarrow x \in \mathbb{R}^2$

$x_{approx} = U_{reduce} \cdot z$

$\mathbb{R}^n$ $\quad$ $n \times k$ $\quad$ $k \times 1$

$n \times 1$

$\approx x$



PCA: Choosing the number of principle components

In PCA, we take n-dimensional features and reduce them to some K-dimensional feature representation.

K: a hyperparameter that specify no. of principle components that we've retained.

What PCA is trying to do is that it tries to minimise the average squared projection error.

→ Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$

→ Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2$ ← avg length of the training examples

Typical understanding: On average, how far are my training examples from the vectors that are all zeros?

Typically, choose k to be the smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \quad (1\%)$$

↳ 99% of variance is retained

Implementation

Algorithm:

try PCA with $k=1$

compute $U_{reduced}, z^{(1)}, z^{(2)} \dots z^{(m)}$,

$x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \ ?$$

if no good, try $k=2$

:

till ≤ 0.01

not efficient!

$[U, S, V] = svd(Sigma)$

S = $n \times n$ diag matrix $= \begin{bmatrix} S_{11} & & 0 \\ & S_{22} & \\ & & \ddots \\ 0 & & S_{nn} \end{bmatrix}$

For given k:

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

Set k from $1 \to n$, to ensure this ≤ 0.01

Call svd only once.

Advice for applying PCA

Use PCA to speed up a learning algorithm

→ Supervised learning speed up

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

Say $x^{(i)} \in \mathbb{R}^{10,000}$ i.e. high dimensional feature

e.g. in computer vision 

→ 10,000 features, 1 for each pixel

we first take our labelled training set and extract just the inputs:

↳ unlabeled dataset: $x^{(1)}, x^{(2)}, \dots, x^{(m)} \in \mathbb{R}^{10000}$

↓ PCA

$z^{(1)}, z^{(2)}, \dots z^{(m)} \in \mathbb{R}^{1000}$

↳ New training set:

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \dots, (z^{(m)}, y^{(m)})$

proceed with supervised learning.

e.g. logistic regression: $h_\theta(z) = \frac{1}{1+e^{-\theta^T z}}$

↳ If new test example $x^{(i)}$, then use the same mapping to get the corresponding $z^{(i)}$, then use the trained hypothesis $h_\theta(z)$ to predict.

* Mapping $x^{(i)} \to z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to $x^{(i)}_{cv}$ and $x^{(i)}_{test}$ in the cross validation and test sets.

i.e. $U_{reduced}$ is a parameter learned by PCA through training set.

Application of PCA

→ Compression
- ↳ reduce memory/disk needed to store data
- ↳ speed up learning algorithm

→ choose k by % of variance retained

→ Visualisation

→ choose $k=2$ or $k=3$

Bad use of PCA: To prevent overfitting

Reasoning: use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to k ($<n$). Thus - fewer features, less likely to overfit.

This might work ok, but is <u>NOT</u> a good way to address overfitting.

Use regularisation instead.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

PCA throw away info on x, w/o knowing y (unsupervised!)

Regularisation knows y value during minimisation, it is better.

Other times PCA is used where it shouldn't be

Design of ML system: (the plan)

→ Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

→ Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$

→ Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$

→ Test on test set:

Map $x^{(i)}_{test}$ to $z^{(i)}_{test}$.

Run $h_\theta(z)$ on $\{(z^{(1)}_{test}, y^{(1)}_{test}), \ldots, (z^{(m)}_{test}, y^{(m)}_{test})\}$

How about doing the whole thing w/o PCA?

Before implementing PCA, first try the original data $x^{(i)}$. Only if that doesn't do what we want, then use PCA and consider using $z^{(i)}$.

Week 9

Anomaly Detection: motivation

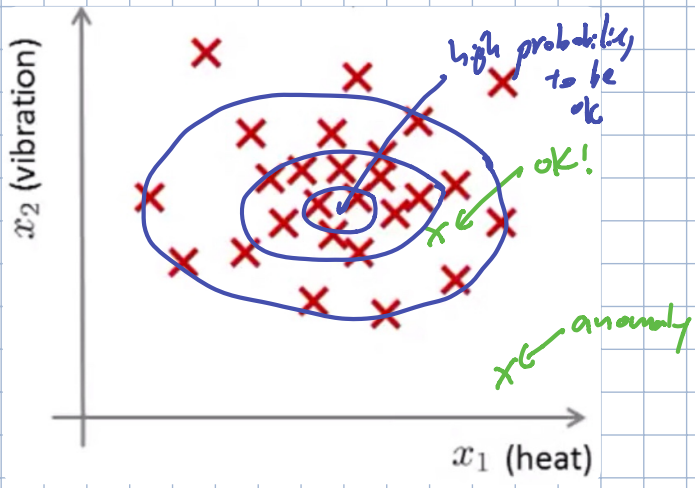
e.g. Aircraft engine features:

x_1 = heat generated

x_2 = vibration intensity

...

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$



Say now we have a new aircraft engine, and has some set of features x_{test}.

→ We want to know if this engine is anomalous i.e. if engine should go further checking

More formally, in the anomaly detection problem, we are given some data set: $\{x^{(1)}, x^{(2)}, \dots x^{(m)}\}$ (non-anomalous examples)

Question: is x_{test} anomalous?

Approach: given the unlabelled training set, build a model: $p(x)$, i.e. probability of x (where x are the features of aircraft engines)

↳ then if $p(x_{test}) < \varepsilon \longrightarrow$ flag as anomaly

$p(x_{test}) \geq \varepsilon \longrightarrow$ OK

* $p(x)$ is actually the unnormalised probability density as parameterised by the feature vector, x.

ε is a threshold condition on the probability density.

↳ Determination of the actual probability need integration of this density over the appropriate extent of phase space.

Anomaly Detection example:

→ Fraud detection:

$x^{(i)}$ = features of user i's activities.

model $p(x)$ from data. (what is the probability that different users behave in different ways?)

Identify unusual users by checking which have $p(x) < \varepsilon$

e.g. x_1 : how often this user log in

x_2: pages visited

x_3: no. of transactions

⋮

→ Manufacturing

→ Monitoring computers in a data center:

$x^{(i)}$ = features of machine i

x_1 = memory use, x_2: no. of disk access

x_3 = CPU load, x_4 = CPU load/network traffic

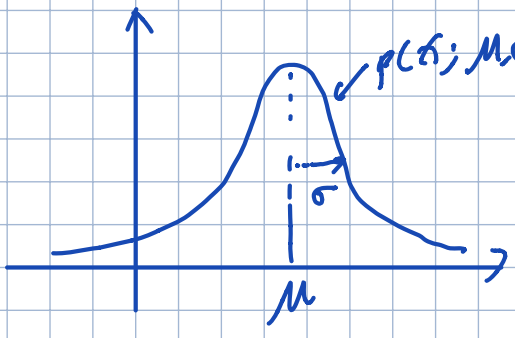
Anomaly Detection: Gaussian distribution

Also called Normal Distribution.

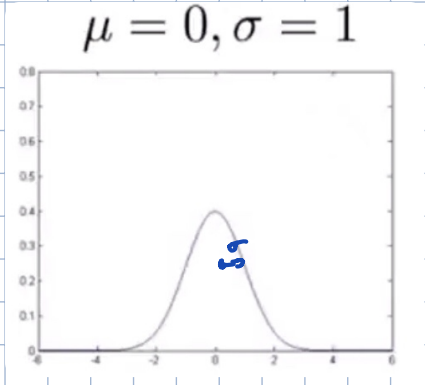
Say $x \in \mathbb{R}$ (i.e. a row number). If x is a distributed Gaussian with mean μ, variance σ^2. (σ: standard deviation)

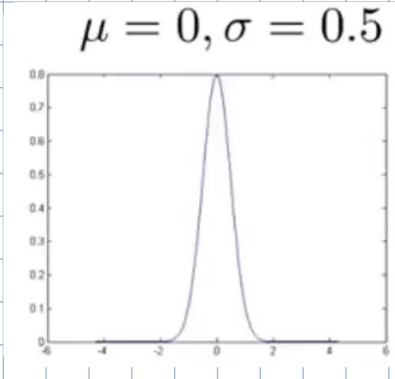
$$x \sim \mathcal{N}(\mu, \sigma^2)$$

"distributed as"

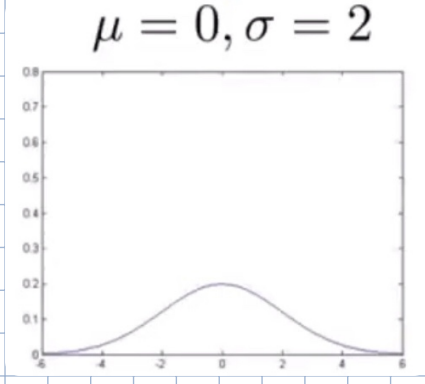


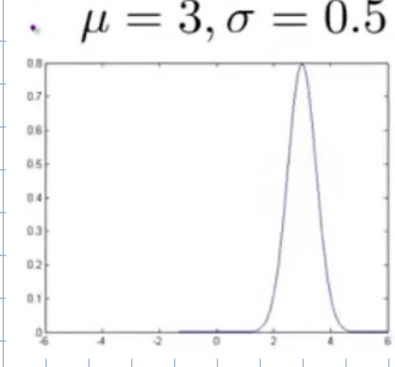
$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$





$\sigma^2 = 0.25$





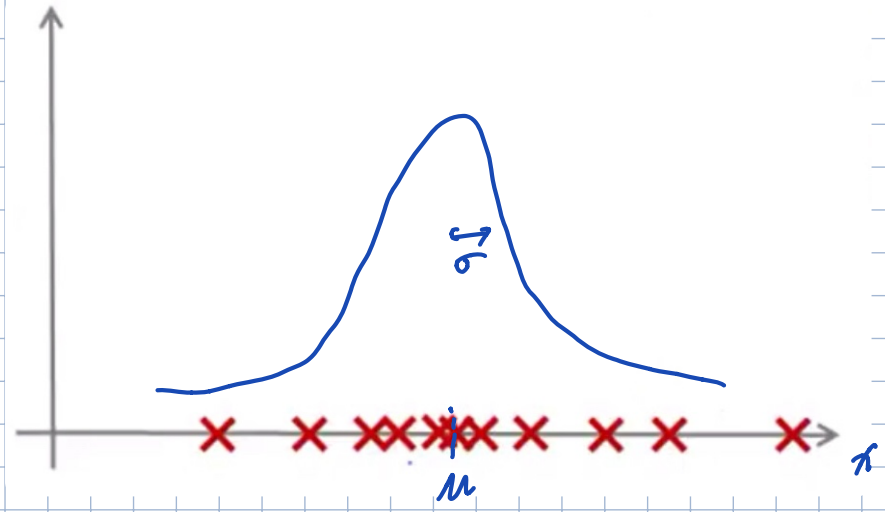
Parameter estimation problem

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ $x^{(i)} \in \mathbb{R}$

Let's say we suspect these examples form a Gaussian distribution.

i.e. $x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$

↳ Given the dataset, try to estimate what are the values of μ and σ^2.



$$\begin{cases} \mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \\ \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2 \end{cases}$$

↳ sometimes use $m-1$, not much difference in practice.

they are also the maximum likelihood estimates of the prime if μ and σ^2.

Anomaly Detection: Algorithm

Let's say we have an unlabeled training set of m examples:

training set: $\{x^{(1)}, \dots, x^{(m)}\}$,

each example is $x \in \mathbb{R}^n$

The way we are going to address anomaly detection, is going to model $p(x)$ from the dataset.

↳ $p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2) \dots p(x_n; \mu_n, \sigma_n^2)$

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

↳ we have to assume the feature x_i is distributed according to a Gaussian distribution, with some mean and variance:

$$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

$$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

* features by right should be independent when using this equation, but in practice it works just fine.

The Anomaly Detection Algorithm

1. Choose features x_i that we think might be indicative of anomalous examples

2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$ (Given training set $\{x^{(1)}, \dots, x^{(m)}\}$)

$p(x_j; \mu_j, \sigma_j^2)$ $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$ i.e. for each feature across all m training samples

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

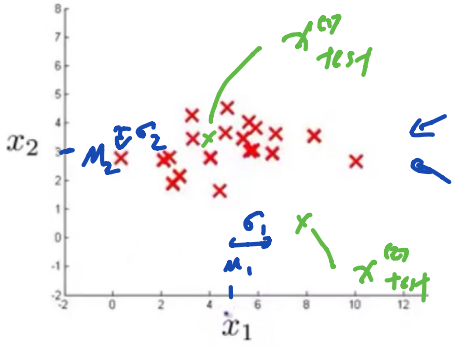
if $\mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$ (vectorize)

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

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

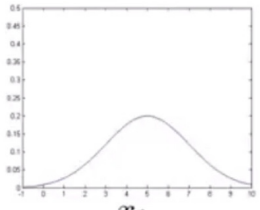
Anomaly if $p(x) < \varepsilon$

Anomaly detection example

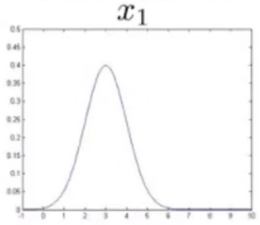


$\mu_1 = 5, \sigma_1 = 2$

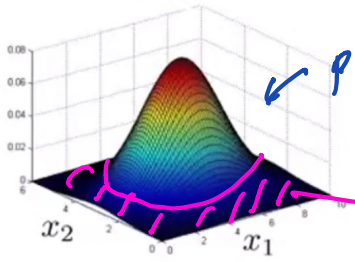
$\mu_2 = 3, \sigma_2 = 1$



$p(x_1; \mu_1, \sigma_1^2)$



$p(x_2; \mu_2, \sigma_2^2)$



$p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2)$

set $\varepsilon = 0.02$

low probability → anomaly to occur assuming Gaussian Distribution

↳ $p(x^{(1)}_{test}) = 0.0426 > \varepsilon$

$p(x^{(2)}_{test}) = 0.0021 < \varepsilon$ (anomaly)

Anomaly Detection: developing / evaluating an anomaly detection system

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

Assume we have some labeled data, of anomalous and non-anomalous examples ($y=0$ if normal, $y=1$ if anomalous)

(previously we treat anomaly detection as an unsupervised problem)

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

↳ cross validation set: $(x^{(1)}_{cv}, y^{(1)}_{cv}), \dots, (x^{(m_{cv})}_{cv}, y^{(m_{cv})}_{cv})$

↳ test set: $(x^{(1)}_{test}, y^{(1)}_{test}), \dots, (x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

e.g. aircraft engine

10000 good (normal) engines ($y=0$)

20 flawed engines (anomalous) ($y=1$)

A typical way of splitting:

training set: 6000 good engines → $\mu_1, \sigma_1^2, \dots, \mu_n, \sigma_n^2$ → $p(x) = p(x_1; \mu_1, \sigma_1^2) \cdots p(x_n; \mu_n, \sigma_n^2)$

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

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

Alternatively, some people do this:

train set: 6000 good engines

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

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

Algorithm Evaluation

→ Fit model $p(x)$ on training set $\{x^{(1)}, \dots, x^{(m)}\}$

→ On a cross validation/test example x, predict

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

↳ Possible evaluation metrics

- ↳ confusion matrix
- ↳ Precision / recall
- ↳ F_1-score

classification accuracy is not a good metrics
↳ very skewed data

→ can also use CV set to choose ε.

Anomaly Detection vs Supervised Learning

Anomaly Detection

→ Very small no. of positive examples ($y=1$). (0-20 is common)

Large no. of negative examples ($y=0$)

→ Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what anomaly looks like; future anomaly may look very different.

e.g. Fraud detection
manufacturing
monitor machines in data center

Supervised learning

→ Large no. of positive and negative examples

→ Enough positive examples for algorithm to get a sense of what positive examples are like; future positive examples likely to be similar to the ones in training set.

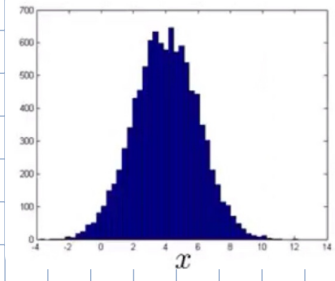
e.g. Spam email
weather prediction
Cancer classification

Anomaly Detection: Choosing features to use

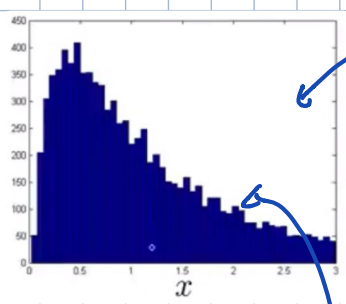
Previously we mention that the features are modeled using Gaussian distribution: $p(x_i; \mu_i, \sigma_i^2)$

We can plot a histogram of the data to make sure they are (sort of) Gaussian. (Even if the data is non-Gaussian, model usually works fine)

good

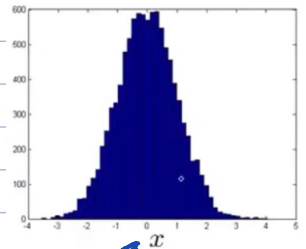


bad



tried some transformation

$\log(x)$



or $x_3 \leftarrow \sqrt{x_3}$

$x_4 \leftarrow x_4^{1/3}$

i.e. $x_1 \leftarrow \log(x_1)$

How we come up with features for an anomaly detection?

→ via error analysis procedure

Goal: want $p(x)$ large for normal examples x.

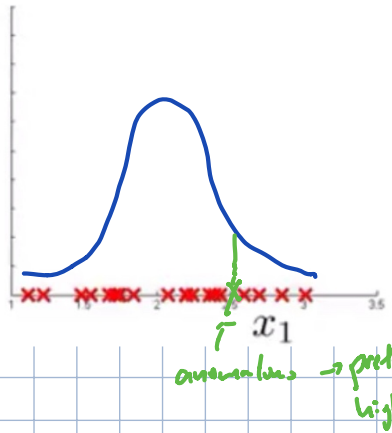
$p(x)$ small for anomalous examples x.

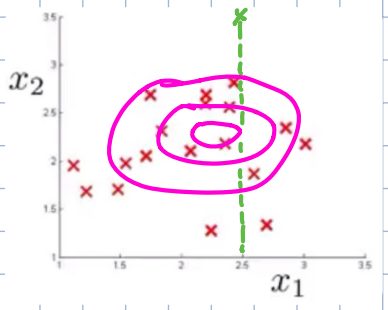
Most common problem:

$p(x)$ is comparable (say, both large) for normal and anomalous examples.

e.g.

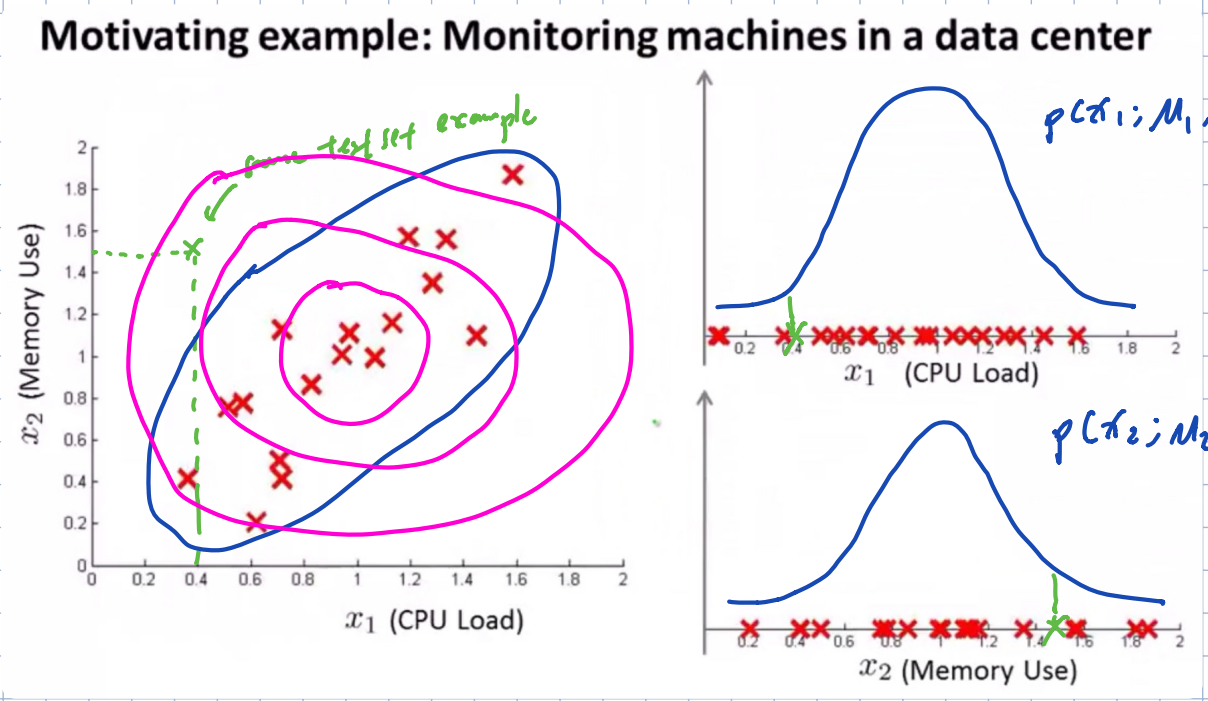
⇒ look at this anomalous example, construct a new feature (say x_2) to identify this anomalous example





* Choose features that might take on unusually large or small in the event of anomaly.

Multivariate Gaussian Distribution



this test set example does not look that bad.

Multivariate Gaussian (Normal) distribution

$x \in \mathbb{R}^n$.

Don't model $p(x_1)$, $p(x_2)$, ..., etc. separately.

Model $p(x)$ all in one go.

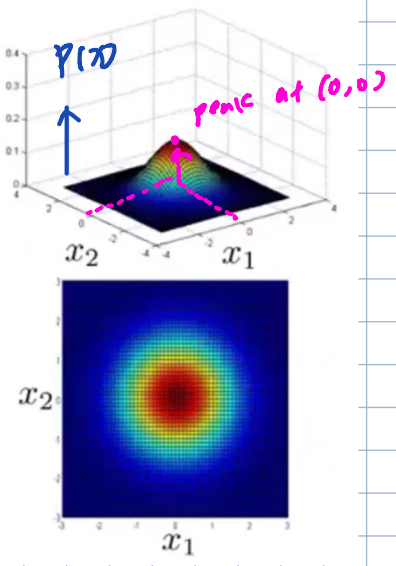
Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

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

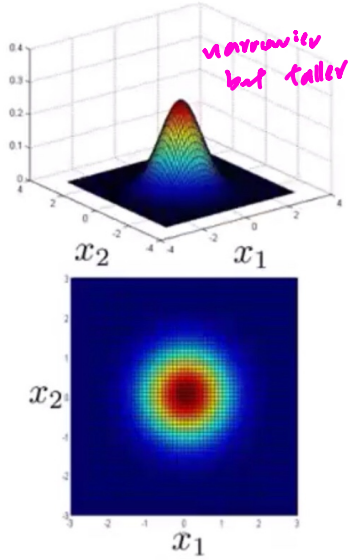
$|\Sigma|$: determinant of Σ

Multivariate Gaussian (Normal) examples

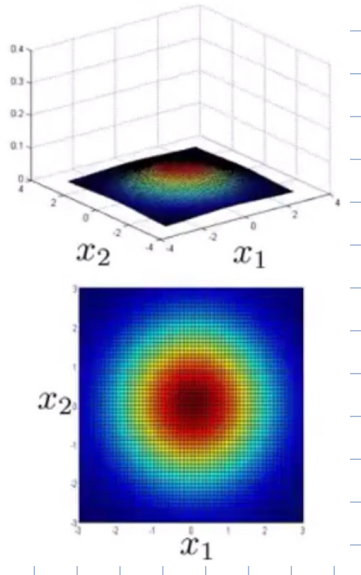
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$ ← less "variance"

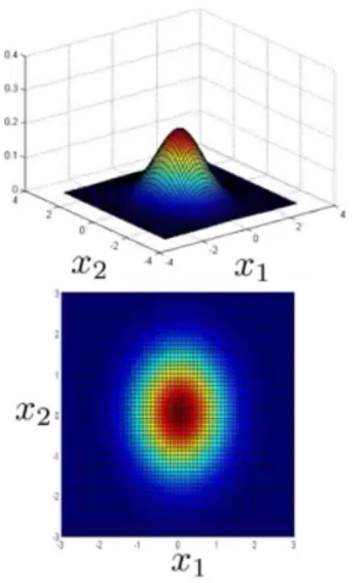


$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

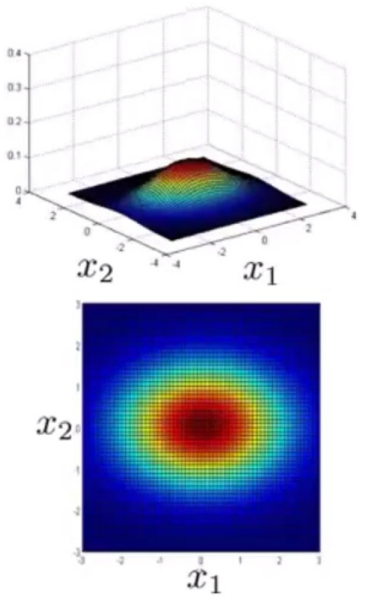


x_1 x_2

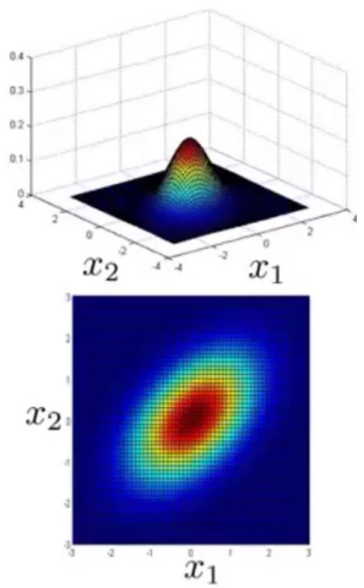
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$



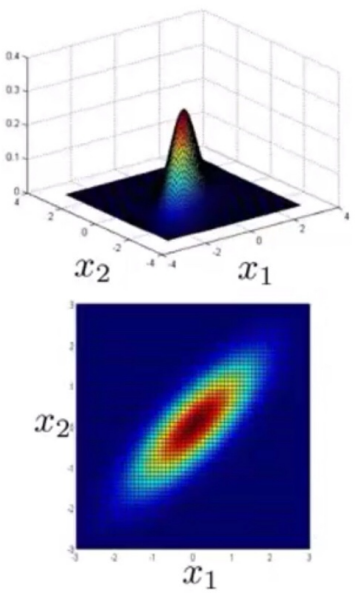
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$

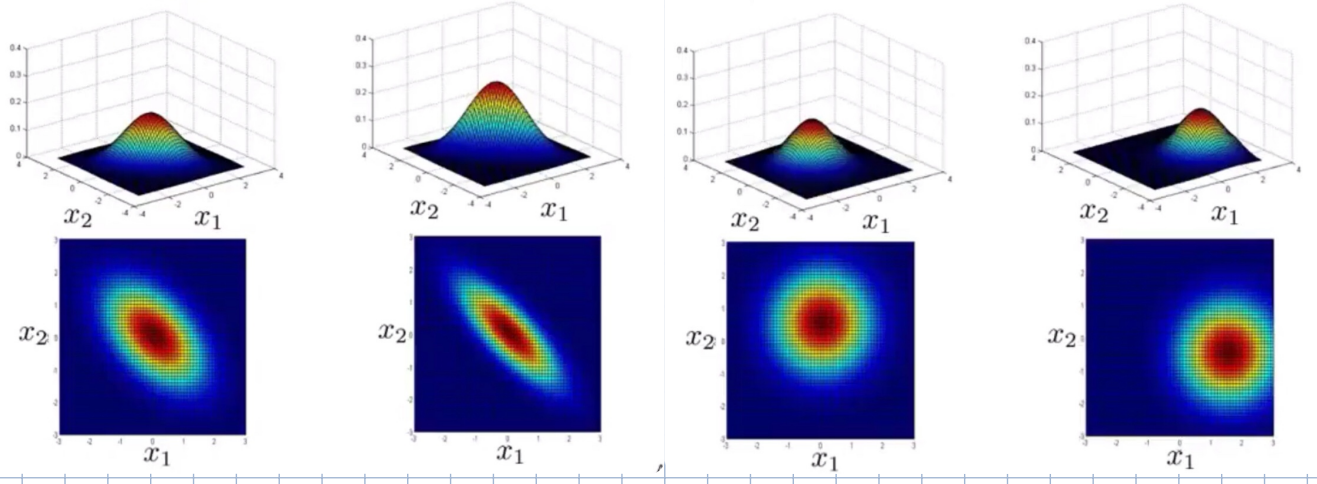


$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$



can be modeled for correlation between x_1 and x_2

$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

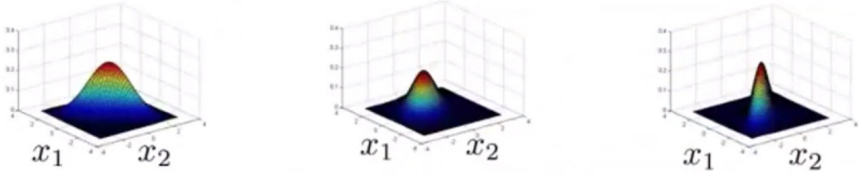


Anomaly Detection: multivariate Gaussian Distribution

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

If I think my examples comes from a multivariate Gaussian Distribution, how do we estimate μ and Σ

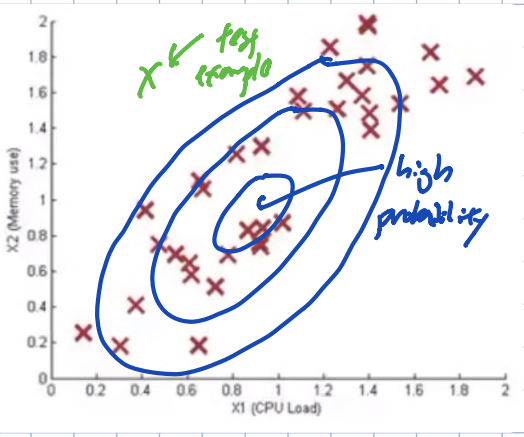
$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

Anomaly detection with the multivariate Gaussian distribution Algorithm:

① Fit model $p(x)$ by setting

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

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



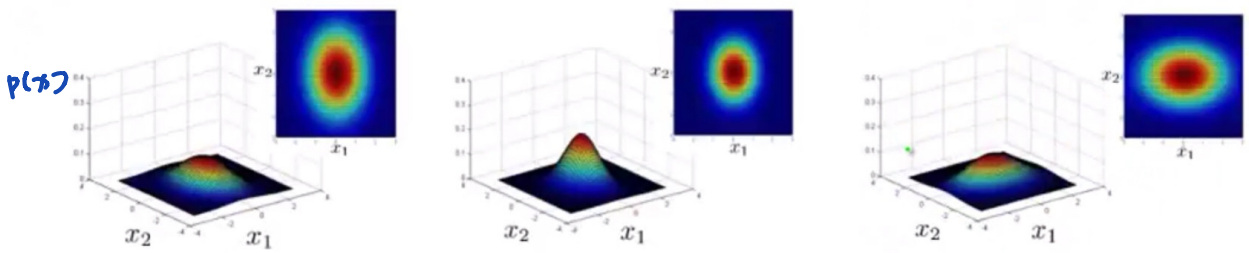
② Given a new example x, compute

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

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

Relationship to original model

Original model: $p(x) = p(x_1;\mu_1,\sigma_1^2) \times p(x_2;\mu_2,\sigma_2^2) \times \cdots \times p(x_n;\mu_n,\sigma_n^2)$



corresponds to a special case of multivariate Gaussian distribution.

by constraining $p(x)$ so that the contour of $p(x)$ are axis aligned.

i.e. no 

i.e. corresponds to multivariate Gaussian

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

where $\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \sigma_2^2 & \\ 0 & & \sigma_n^2 \end{bmatrix}$ (i.e. no correlations between features)

original (Gaussian) model

$p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$

→ Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values.

→ computationally cheaper (scales better to large n).

→ OK even if m (training set size) is small.

Multivariate Gaussian

$p(x; \mu, \Sigma)$

→ Automatically captures correlations between features.

→ computationally more expensive

(e.g. Σ^{-1} is expensive for large n)

→ must have $m > n$, or else Σ is non-invertible.

↳ or redundant features).

(e.g. $x_1 = x_2$ or $x_3 = x_4 + x_5$)

Recommender System: Problem formulation

Example: Predicting movie ratings

User rates movies using ~~one~~ zero to five stars

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

n_u = no. of users

n_m = no. of movies

So, $n_u = 4$

$n_m = 5$ here

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j) = 1$)

Given $r(i,j)$ and $y^{(i,j)}$, can we predict the "?" rating?

Recommender System: Content Based Recommendation

	Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
1	Love at last $x^{(1)}$	5	5	0	0	0.9	0
2	Romance forever	5	?	?	0	1.0	0.01
3	Cute puppies of love	?	4	0	?	0.99	0
4	Nonstop car chases	0	0	5	4	0.1	1.0
5	Swords vs. karate	0	0	5	?	0	0.9

x_1: a feature that measures the degree to which a movie is a romantic movie.

x_2: a feature that measures the degree to which a movie is an action movie.

$x_0 = 1$

then we can have a movie vector

i.e. $x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$ for the first movie

In order to make predictions, we can treat for each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. (i.e. separate linear regression problem) $\theta^{(j)} \in \mathbb{R}^{n+1}$

Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

e.g. $x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$ for "Cute puppies of love"

we want to predict what Alice would rate this movie

Say we have $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ after training, then our prediction will be

$(\theta^{(1)})^T x^{(3)} = 4.95$

Problem formulation

More formally:

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)

$y^{(i,j)}$ = rating by user j on movie i (if defined)

$\theta^{(j)}$ = parameter vector for user j

$x^{(i)}$ = feature vector for movie i

For user j, movie i, predicted rating: $(\theta^{(j)})^T (x^{(i)})$

$m^{(j)}$ = no. of movie rated by user j

To learn $\theta^{(j)}$:

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

$\theta^{(j)} \in \mathbb{R}^{n+1}$

sum over i that $r(i,j)=1$ i.e. j movies has rated

* For easier implementation on Recommender

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

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

↓

obtain separate θ for each user

Optimization objective:

To learn $\theta^{(j)}$ (parameter for user j):

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

To learn $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$:

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

Optimization algorithm:

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

$\rightarrow J(\theta^{(1)}, \ldots \theta^{(n_u)})$

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

$$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \ldots, \theta^{(n_u)})$$

This content base approach assumes that we have available to us features for the different movie (i.e. is this action movie?)

Recommender System: Collaborative filtering

Problem motivation:

sometimes we do not have feature about movie content

we change the problem a bit and suppose we do not know of these value about movie content.

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$ Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

Lets say we've gone to each of our users, and each of our users has told us how much they like romantic / action movies ($x_0 = 1$)

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \quad \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

i.e. each user tells us $\theta^{(j)}$

↳ It is possible to infer x_1 and x_2 from each movie

Mathematically:

what feature vector should $x^{(1)}$ be, so that $(\theta^{(1)})^T x^{(1)} \approx 5$

$$x^{(1)} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$(\theta^{(2)})^T x^{(1)} \approx 5$

$(\theta^{(3)})^T x^{(1)} \approx 0$

$(\theta^{(4)})^T x^{(1)} \approx 0$

Optimisation algorithm

→ Given $\theta^{(1)}, \dots, \theta^{(n_m)}$, to learn $x^{(i)}$:

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

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

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

Collaborative Filtering

Given $x^{(1)}, \dots, x^{(n_m)}$ (and movie rating)

can estimate $\theta^{(1)}, \dots \theta^{(n_u)}$

Given $\theta^{(1)}, \dots \theta^{(n_u)}$,

can estimate $x^{(1)}, \dots, x^{(n_m)}$

We can randomly guess some θ, then get x, then estimate better θ.

The collaborative filtering algorithm

Collaborative filtering optimization objective

→ Given $x^{(1)}, \dots, x^{(n_m)}$, estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$:

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

→ Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, estimate $x^{(1)}, \dots, x^{(n_m)}$:

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

Minimizing $x^{(1)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \dots, \theta^{(n_u)}$ simultaneously:

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

$$\min_{\substack{x^{(1)},\dots,x^{(n_m)} \\ \theta^{(1)},\dots,\theta^{(n_u)}}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$$

Andrew Ng

instead of iteratively go from $\theta \to x \to \theta \to x \dots$

more efficient algorithm

Collaborative filtering algorithm

$x_0 = 1$ ~~$x_0 = 1$~~ $x \in \mathbb{R}^n$, $\theta \in \mathbb{R}^n$

→ 1. Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.

→ 2. Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm). E.g. for every $j = 1, \dots, n_u, i = 1, \dots, n_m$:

θ_0 ~~θ_0~~

θ_1

$\vdots$

θ_n

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

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

$\frac{\partial}{\partial x_k^{(i)}} J(\dots)$

3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

$(\theta^{(j)})^T x^{(i)}$

user j on movie i

Recommender System Vectorisation: Low Rank Matrix Factorisation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

$y^{(i,j)}$

$n_m = 5$ ↑ 5×4

$n_u = 4$

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

↑ $(\theta^{(j)})^T(x^{(i)})$

↑ $X(\Theta)^T$

Vectorisation:

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(n_m)})^T — \end{bmatrix} \qquad \Theta = \begin{bmatrix} — (\theta^{(1)})^T — \\ — (\theta^{(2)})^T — \\ \vdots \\ — (\theta^{(n_u)})^T — \end{bmatrix}$$

Also called Low Rank Matrix factorisation algorithm.

Additionally, we can use the learned features in order to find related movies.

For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$

e.g. x_1 = romance, x_2 = action, ...

How to find movies j related to movie i?

Small $||x^{(i)} - x^{(j)}|| \longrightarrow$ movies i and j are similar

e.g. 5 most similar movies to movie i:

↳ 5 movie j with the smallest $||x^{(i)} - x^{(j)}||$

Implementation Details: Mean Normalisation

Users who have not rated any movies

(Eve (5): not rated any movie)

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

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

say $n=2$, (i.e. 2 features: romance, action)

$\theta^{(5)} \in \mathbb{R}^2$, since Eve has not rated any movie, so no movies that $r(i,j)=1$.

↳ only last term has effect in determine $\theta^{(5)}$, i.e.

$$\frac{\lambda}{2}\left[(\theta_1^{(5)})^2 + (\theta_2^{(5)})^2\right] \longrightarrow \theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

then $(\theta^{(5)})^T x^{(i)} = 0$ NOT useful

Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \qquad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

average rating for each movie

Substract the means

learn $\theta^{(j)}, x^{(i)}$

For user j, on movie i, predict:

$$(\theta^{(j)})^T (x^{(i)}) + \mu_i$$

↑ compensate back

user 5 (eve):

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \underbrace{(\theta^{(5)})^T (x^{(i)})}_{0} + \mu_i$$

i.e. if eve never rates any movies - just use the means.

week 10

Learning with large datasets

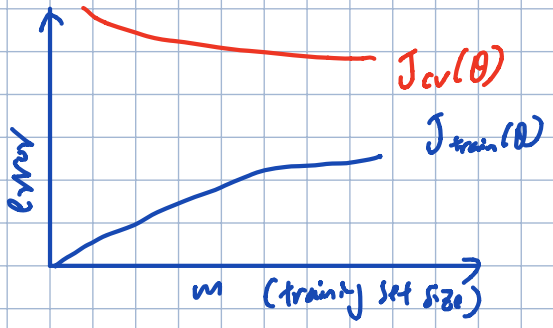
Says, $m = 100,000,000$ (training samples)

Gradient descent for linear / logistic regression:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

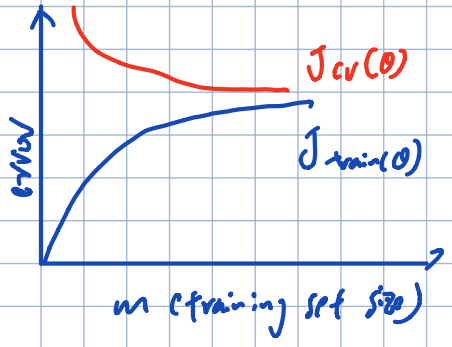
$\uparrow$ $m = 100$ mil

For sanity check, if m is small is also good, plot learning curve



high variance learning problem.

increase m might help



high bias problem

increase m doesn't help much

should add extra feature, or extra hidden layer.

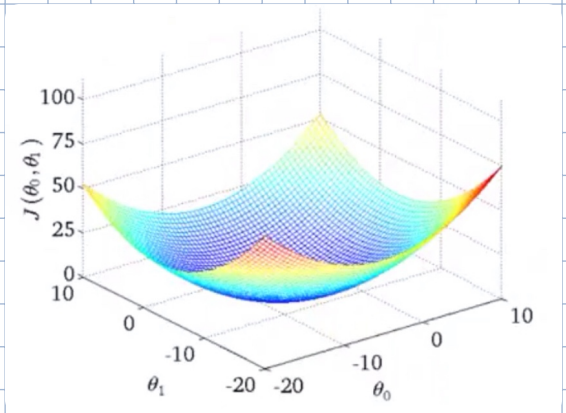
Stochastic Gradient Descent

Gradient descent becomes computationally expensive when data set is big

Linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$



Gradient Descent:

Repeat {

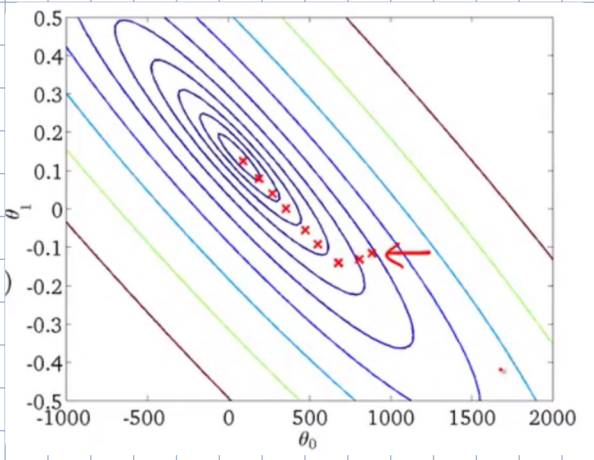
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for every $j = 0, \ldots, n$)

}

expensive if m is large

$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$



↳ this is called batch gradient descent.

↳ looking at all of the training examples.

Stochastic gradient descent

$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ on a single example

$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$

1. Randomly shuffle the dataset
2. Repeat {

   for $i = 1, \dots, m$ {

   $\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

   (for $j = 0, \dots, n$)

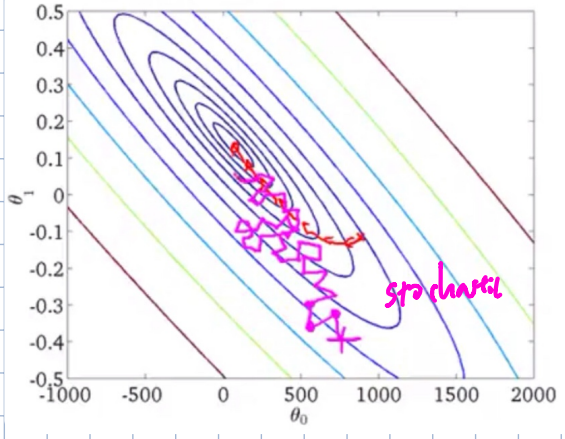
   } ← maybe 1-10 times

   } ← $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

i.e. the algorithm will only look at $(x^{(i)}, y^{(i)})$, and the cost of just this training example.

then fit the 2nd training example → then continue on for the entire training set.

this approach modifies the parameter bit by bit, rather than in one go.



Stochastic approach might not reach the global minimum in the same sense as batch.

It will also wandering around the minimum instead of converge to minimum.

Mini-batch Gradient Descent

So far:

Batch gradient descent: Use all m examples in each iteration.

Stochastic gradient descent: use 1 example in each iteration.

mini-batch gradient descent: use b examples in each iteration.

b = mini-batch size (e.g. $b = 10$)

e.g.

Get $b = 10$ examples $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$$

$$i := i + 10$$

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \dots, n$)

}

}

mini-batch is likely to outperform stochastic only if we have good vectorisation implementation

Stochastic Gradient Descent Convergence

Check for convergence:

→ Batch gradient descent:

Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

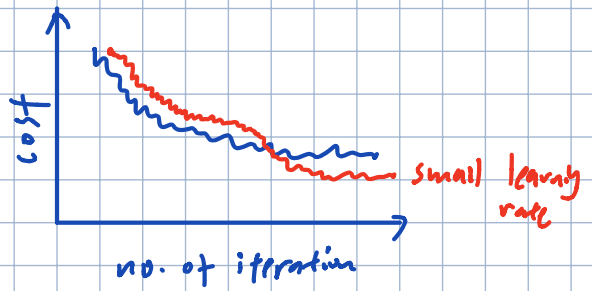
if m is small $J_{train}(\theta)$ can be computed rather efficiently.

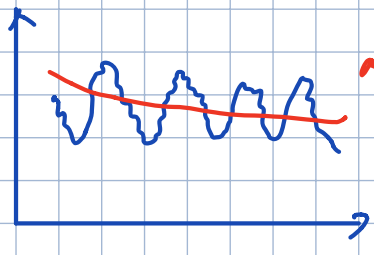
→ Stochastic gradient descent:

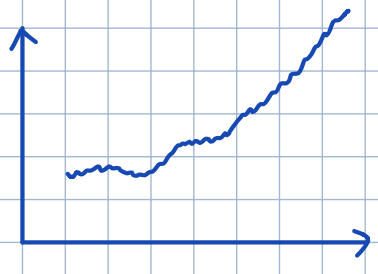
$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$

Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.







the learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (Eg. $\alpha = \frac{const\ 1}{iteration\ Number + const\ 2}$)

↳ ∵ for stochastic, θ won't converge, just wander around global minimum

Online Learning

Suppose we run a shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y=1$), sometime not ($y=0$)

Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y=1 \mid x;\theta)$ to optimise price.

↑ Captive price.

For online learning and logistic regression:

Repeat forever {
  Get (x,y) corresponding to user. ← origin/destination/price etc. 0 or 1
  Update θ using $(x,y) \leftarrow (x^{(i)},y^{(i)})$, i.e. just this example
  $\theta_j := \theta_j - \alpha(h_\theta(x)-y)\cdot x_j \quad (j=0,\dots,n)$
}

Look at 1 example a time, and discards after using.

Can adapt to changing user preferance.

other online learning example:

Product search (learning to search)

User searches for "Android phone 1080p camera"

Have 100 phones in store. will return 10 results.

For each phone and given a specific user query, we can construct a feature vector x.

x = features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc

$y=1$ if user click on link, $y=0$ otherwise

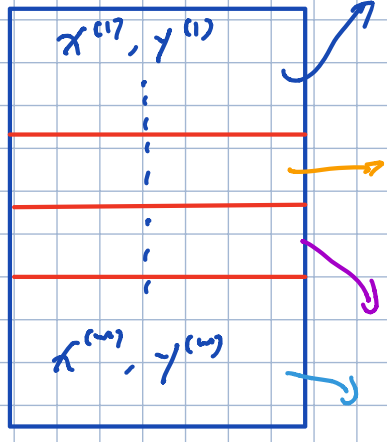
learn $p(y=1 \mid x;\theta)$

use to show user 10 phones they're most likely to click on.

Map-Reduce and Data Parallelism

Say we want to fit a linear regression model or logistic regression model.

Batch gradient descent: $\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$



Machine 1: use $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$

$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

machine 2: use $(x^{(101)}, y^{(101)}), \ldots (x^{(200)}, y^{(200)})$

$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

machine 3

machine 4

same

↳ Combine: $\theta_j := \theta_j - \alpha \frac{1}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$

$(j=0, \ldots, n)$

→ Many learning algorithms can be expressed as computing sums of functions over the training set.

E.g. for advanced optimization, with logistic regression, need:

$$J_{train}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$$

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

week 11

Photo OCR: problem description and pipeline

Photo OCR: photo Optical Character Recognition

Photo OCR pipeline

1. Text detection

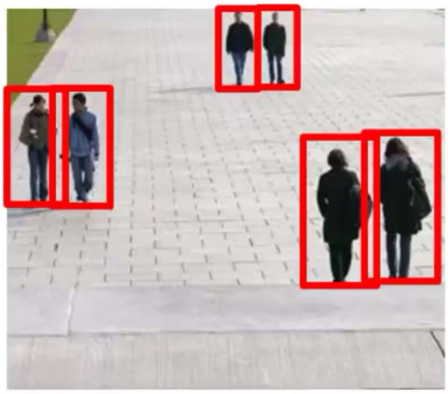
2. Character segmentation

3. Character classification

Photo OCR: sliding windows

Text detection

Pedestrian detection



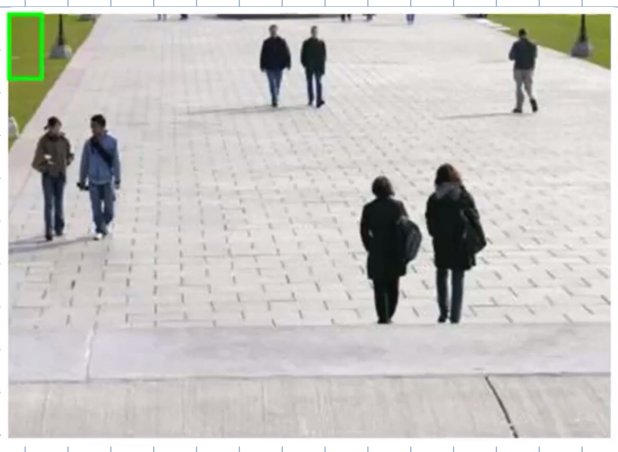
Supervised learning for pedestrian detection

x = pixels in 82x36 image patches → aspect ratio

Positive examples $(y = 1)$

Negative examples $(y = 0)$

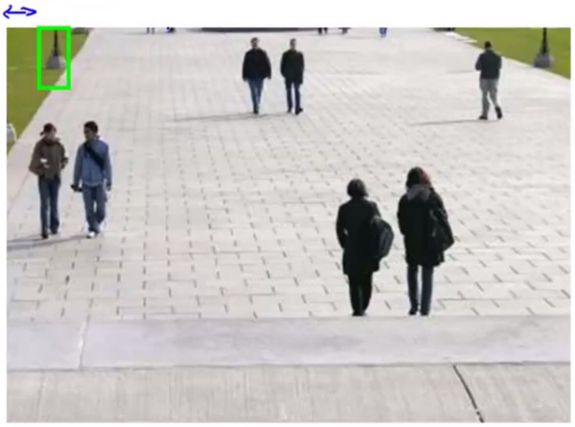
↳ collect large set of positive / negative examples



→

step-size / stride

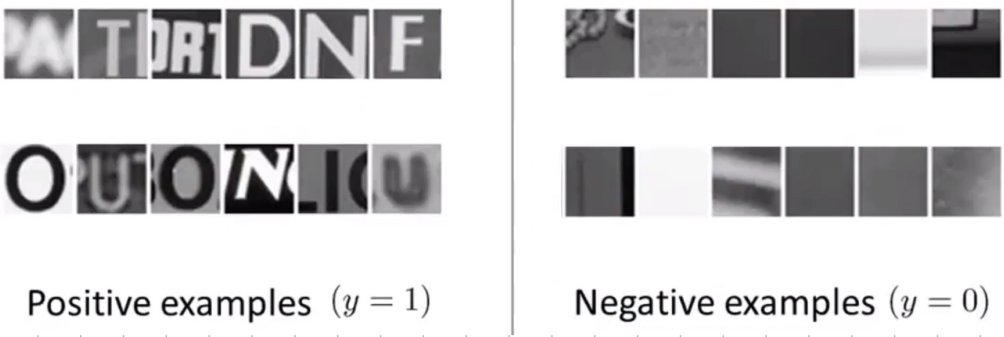
↔



Run the first patch (82×36 img) through classifier. (Hopefully returns y=0).

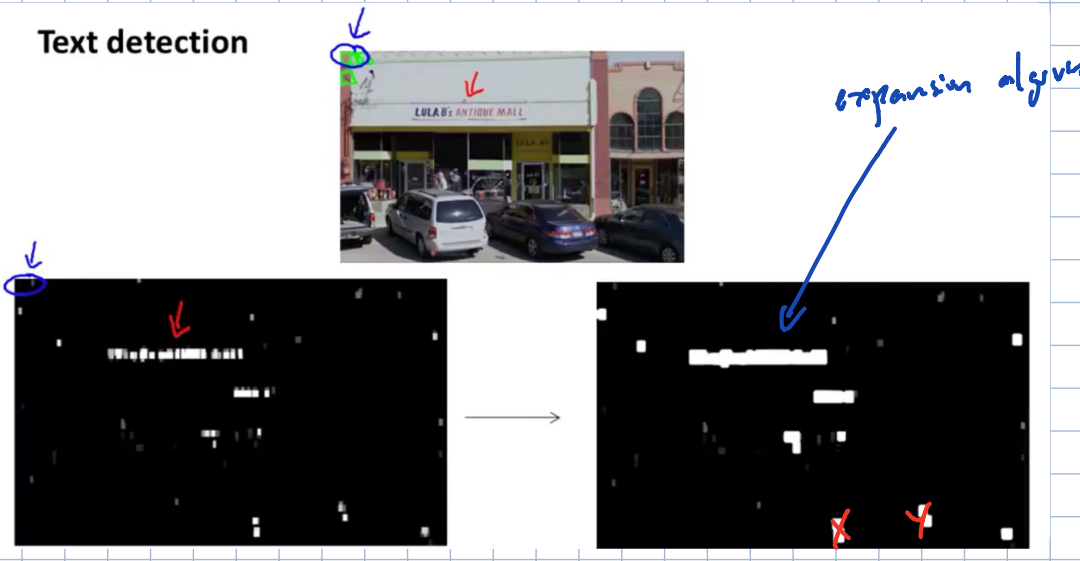
Then slide the green box, and run that patch to the classifier again.

Text detection



Positive examples $(y = 1)$ | Negative examples $(y = 0)$

Text detection



expansion algorithm

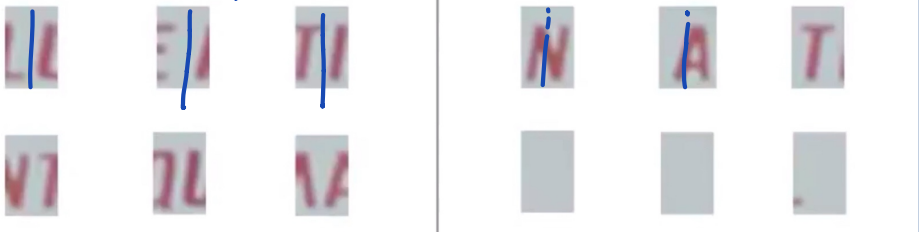
too thin box

1D Sliding window for character segmentation

train classifier to split text

split

not split



Positive examples $(y = 1)$ | Negative examples $(y = 0)$

Artificial Data Synthesis

Artificial data synthesis for photo OCR

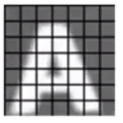
Real data

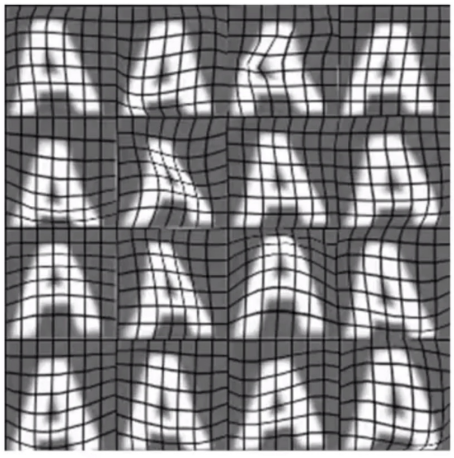
we can take characters of different fonts, and paste them against random backgrounds, to increase training examples.

Real data

Synthetic data

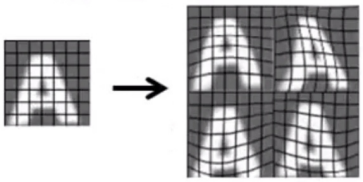
Synthesizing data by introducing distortions





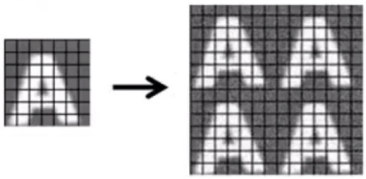
Synthesizing data by introducing distortions

Distortion introduced should be representation of the type of noise/distortions in the test set.



Audio:
Background noise,
bad cellphone connection

Usually does not help to add purely random/meaningless noise to your data.



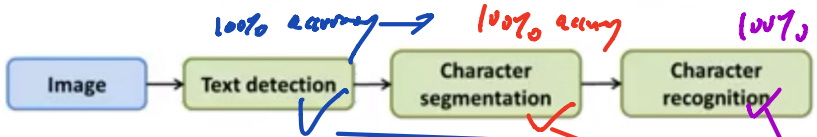
x_i = intensity (brightness) of pixel i
$x_i \leftarrow x_i +$ random noise

Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
   - Artificial data synthesis
   - Collect/label it yourself
   - "Crowd source" (E.g. Amazon Mechanical Turk)

Ceiling analysis

Estimating the errors due to each component (ceiling analysis)



What part of the pipeline should you spend the most time trying to improve?

component	accuracy
overall system	72 %
Text detection	89 %
char seg	90 %
char rec	100 %

Summary: Main topics

$(x^{(i)}, y^{(i)})$

→ Supervised Learning
- Linear regression, logistic regression, neural networks, SVMs

$x^{(i)}$

→ Unsupervised Learning
- K-means, PCA, Anomaly detection

→ Special applications/special topics
- Recommender systems, large scale machine learning.

→ Advice on building a machine learning system
- Bias/variance, regularization; deciding what to work on next: evaluation of learning algorithms, learning curves, error analysis, ceiling analysis.