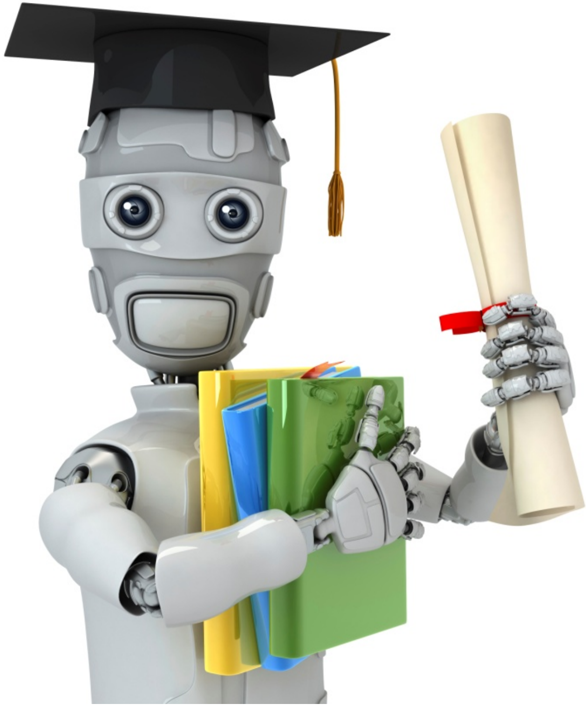


Machine Learning

Support Vector Machines

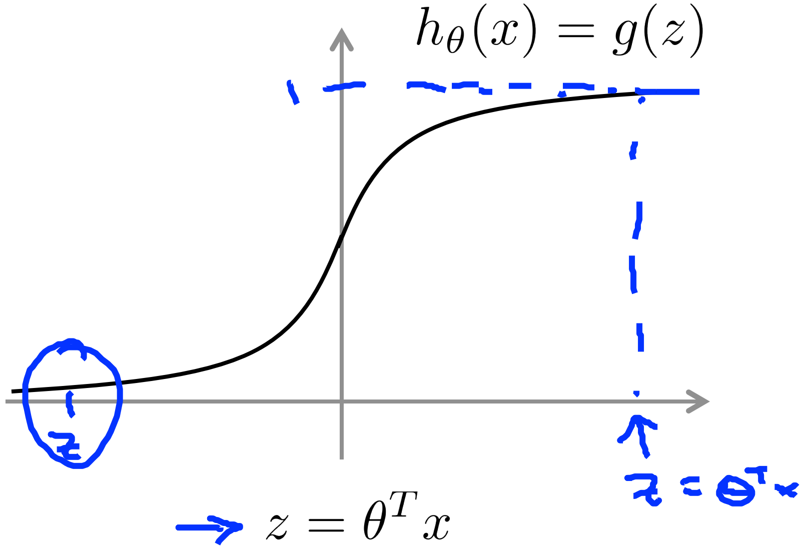
sometimes gives a cleaner and more powerful way of learning complex non-linear functions.

Optimization objective

Alternative view of logistic regression

sigmoid activation function

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



much much greater than

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

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

Alternative view of logistic regression

(x, y)

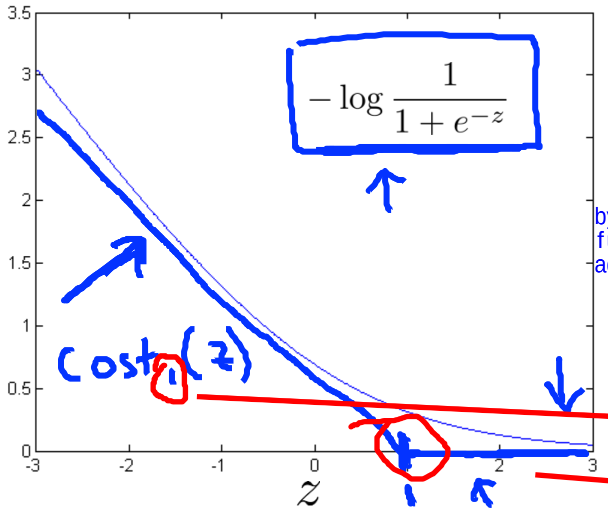
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

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

$z = \theta^T x$

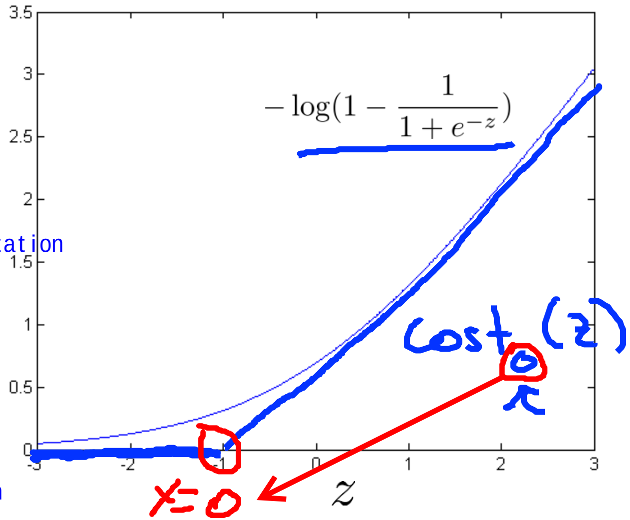
If $y = 0$ (want $\theta^T x \ll 0$):



by defining this new cost function, it gives us computation advantage

$x=1$

new cost function



$x=0$

Support vector machine

Logistic regression:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \underbrace{\left(-\log h_\theta(x^{(i)})\right)}_{\text{replace with } cost_1(\theta^T x^{(i)})} + (1-y^{(i)}) \underbrace{\left((-\log(1-h_\theta(x^{(i)}))\right)}_{cost_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Support vector machine:

C by convention, just a diff way of controlling the tradeoff in the regularization term!

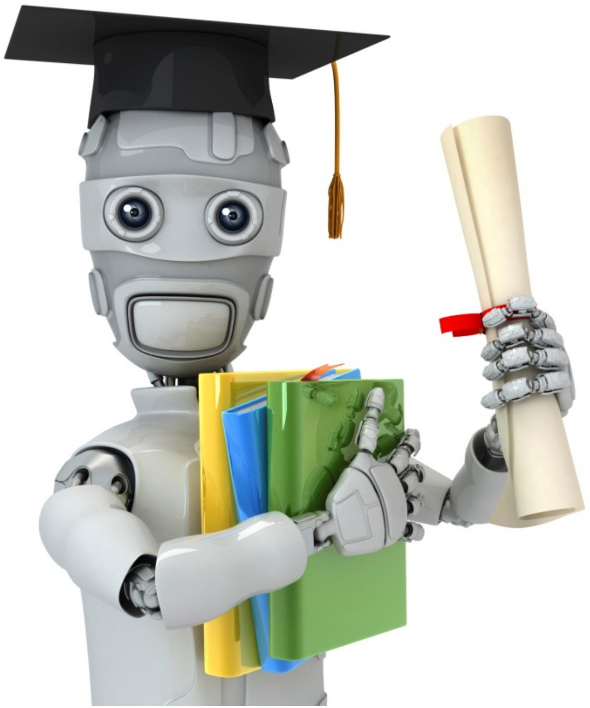
$$\boxed{\min_\theta C \sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2}$$

SVM hypothesis

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

Hypothesis:

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



Machine Learning

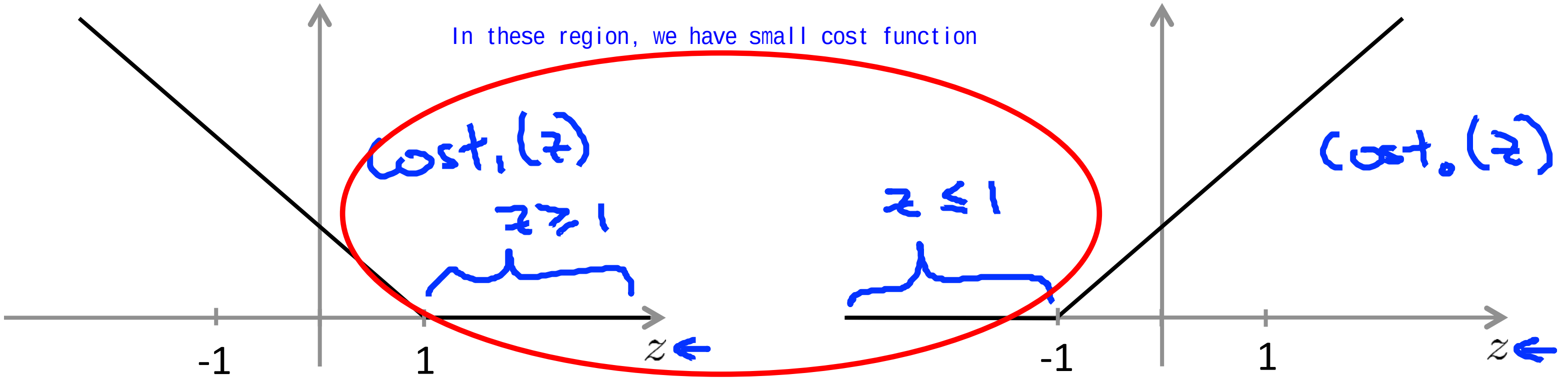
= large margin classifier

Support Vector Machines

Large Margin Intuition

Support Vector Machine

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



so that cost function small!!

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

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

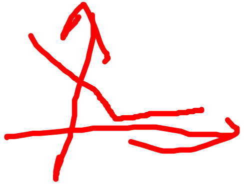
$C = 100{,}000$

SVM Decision Boundary

$$\min_\theta C \underbrace{\boxed{\sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right]}}_{=0} + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

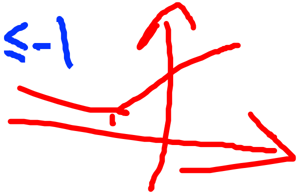
Whenever $y^{(i)} = 1$:

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



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

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



New optimization problem: much efficient!

$$\min_\theta \cancel{C \times 0} + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

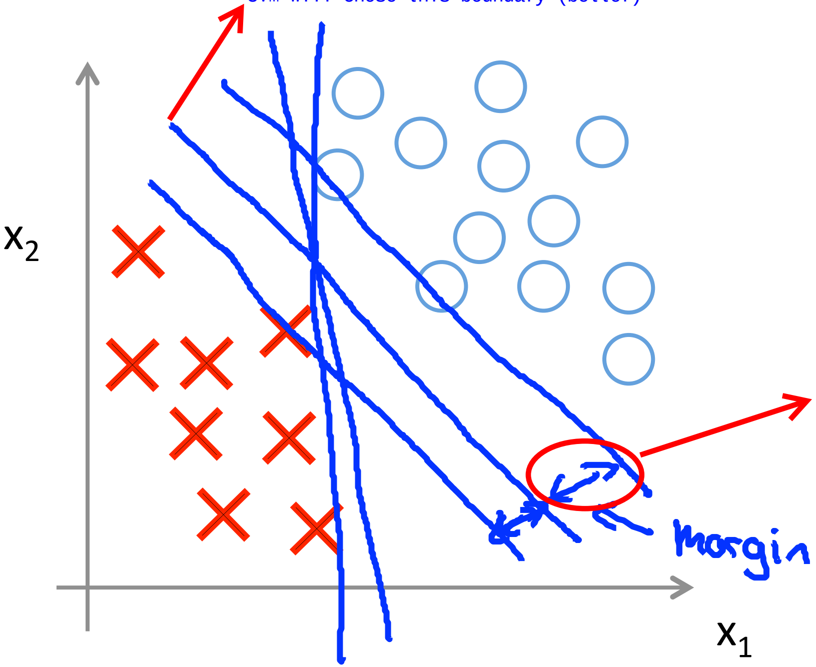
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

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

subject to these constraints

SVM Decision Boundary: Linearly separable case

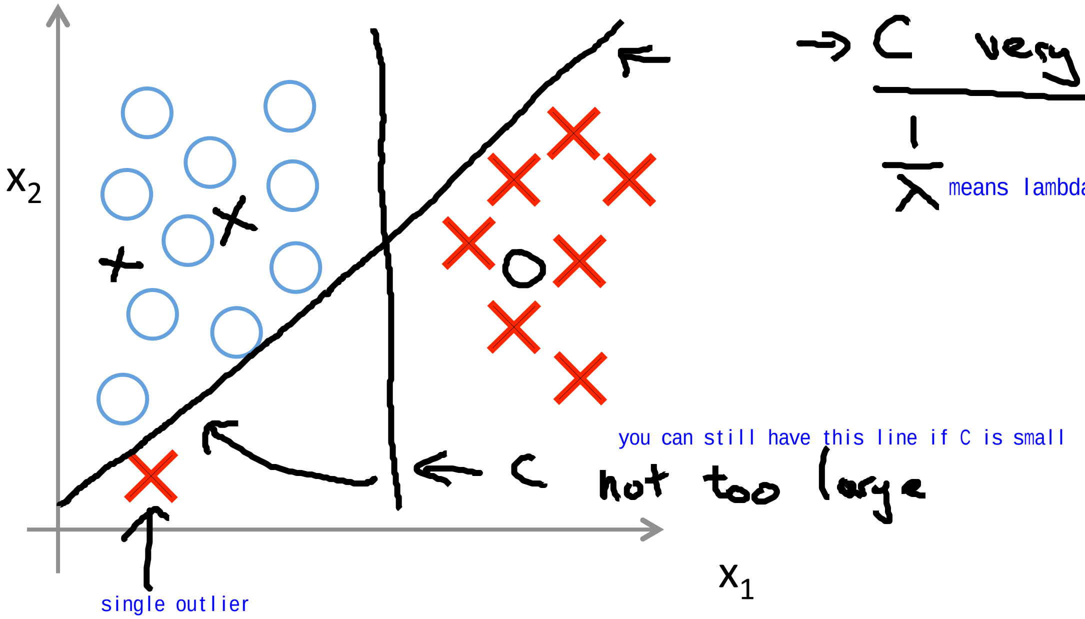
SVM will chose this boundary (better)



this distance is called the margin of SVC
this gives us certain robustness!
we want to separate data with the margin as large as possible!

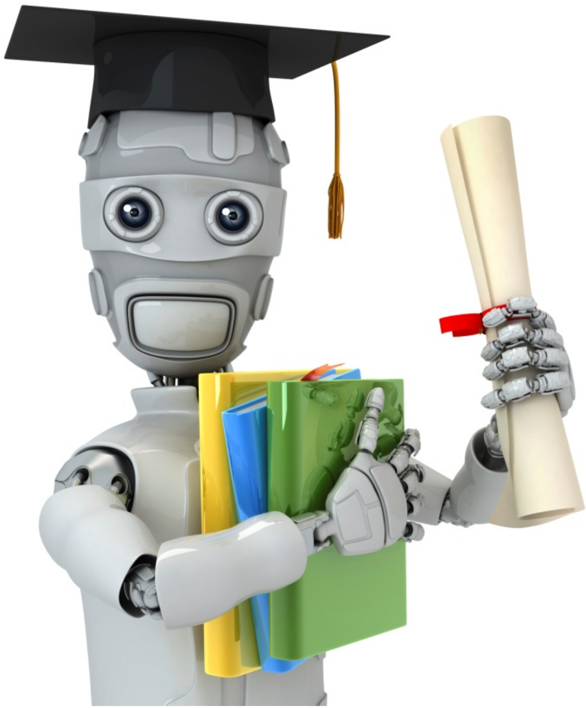
Large margin classifier = SVM

Large margin classifier in presence of outliers



$\rightarrow$ C very large

$\frac{1}{\lambda}$ means lambda is very small!



Machine Learning

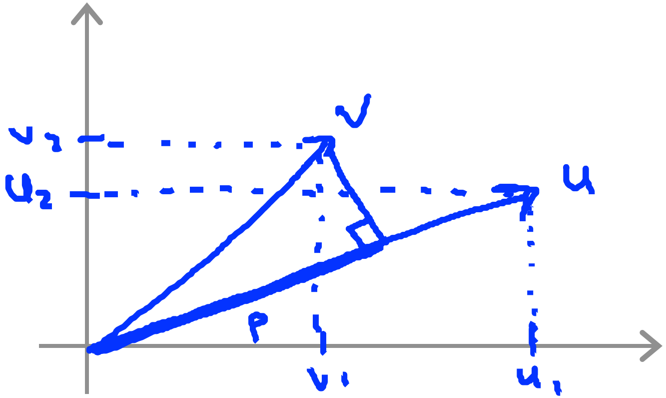
Gives some intuition about how the new optimization problem can result in a large margin classification problem.

Support Vector Machines

The mathematics behind large margin classification (optional)

Vector Inner Product

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



$u^T v = ?$ $\quad [u_1 \;\; u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

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

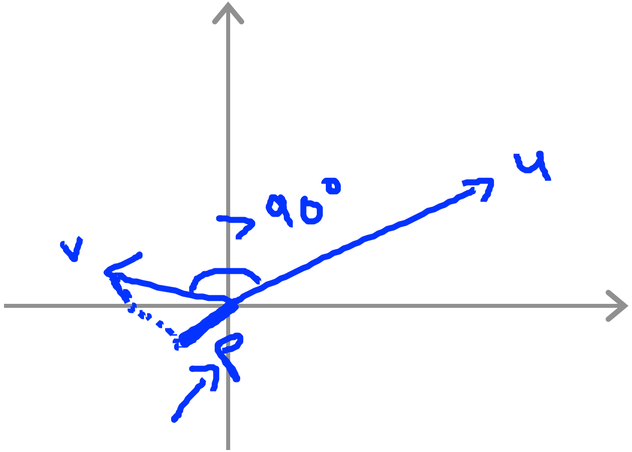
$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

p = length of projection of v onto u.

Signed

$u^T v = \underline{p} \cdot \underline{\|u\|} \quad = v^T u$

$= u_1 v_1 + u_2 v_2 \quad p \in \mathbb{R}$



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

$p < 0$

SVM Decision Boundary

optimization objective

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

$$\sqrt{\theta_1^2+\theta_2^2} = \|\theta\|$$

$$w = (\sqrt{w})^2$$

$$\text{s.t.}\quad \boxed{\theta^T x^{(i)} \geq 1} \quad \text{if } y^{(i)} = 1$$

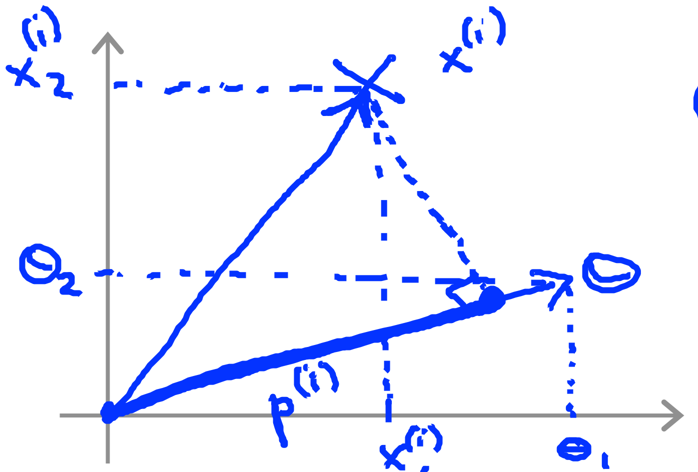
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

Simplication: $\theta_0 = 0$. $n=2$

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

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

$$u^T v$$



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

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

A geometric view/derivation/intuition/proof of large margin classifier

SVM Decision Boundary

equivalently, we have the following optimization objective!

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

$$\text{s.t.} \quad \boxed{p^{(i)}\cdot\|\theta\| \geq 1} \quad \text{if } y^{(i)} = 1$$

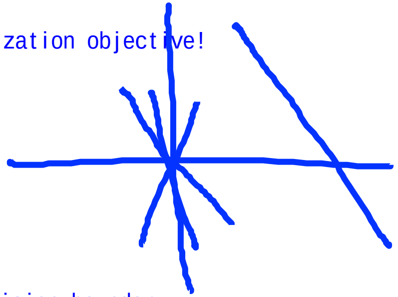
$$p^{(i)}\cdot\|\theta\| \leq -1 \quad \text{if } y^{(i)} = 1$$

C very large

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ.

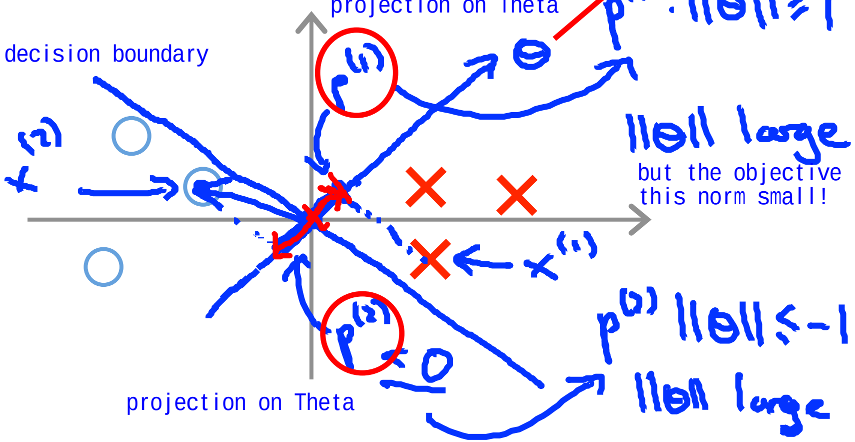
Simplification: $\theta_0 = 0$

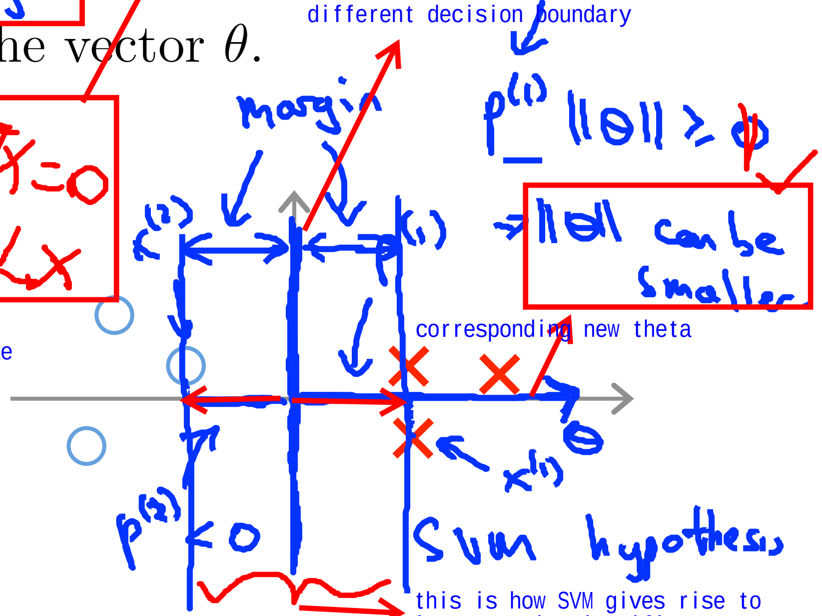
$\theta_0 = 0$



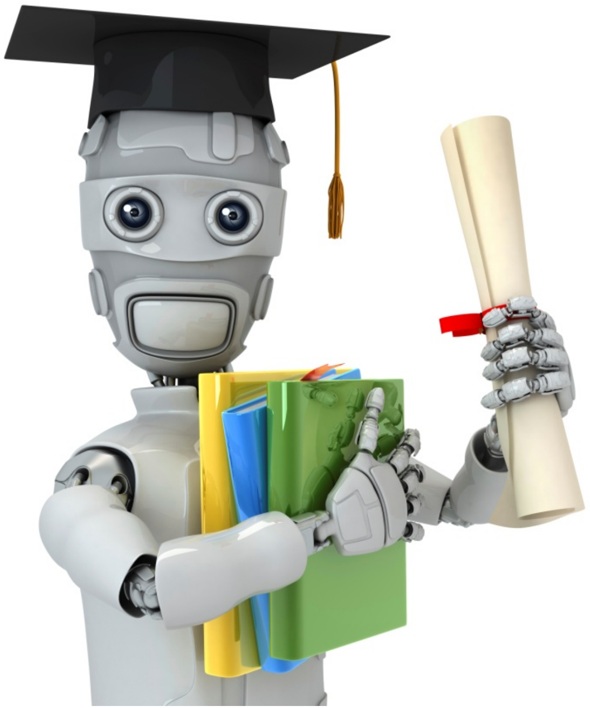
$\theta^T x = 0$

$\theta \perp x$





Andrew Ng



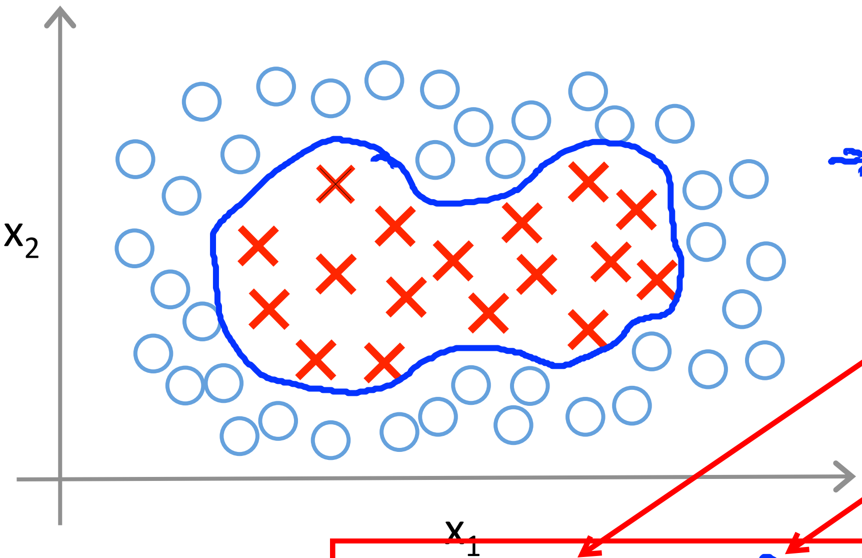
Machine Learning

Support Vector Machines

Kernels I

Main technique for adapting support vector machines in order to develop complex nonlinear classifiers.

Non-linear Decision Boundary



One way of doing this is to use polynomial fit

Predict $y = 1$ if

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

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

$$\boxed{\begin{array}{l} \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \ldots \\ f_1 = x_1,\quad f_2 = x_2,\quad f_3 = x_1 x_2,\quad f_4 = x_1^2,\quad f_5 = x_2^2, \ldots \end{array}}$$

linear regression

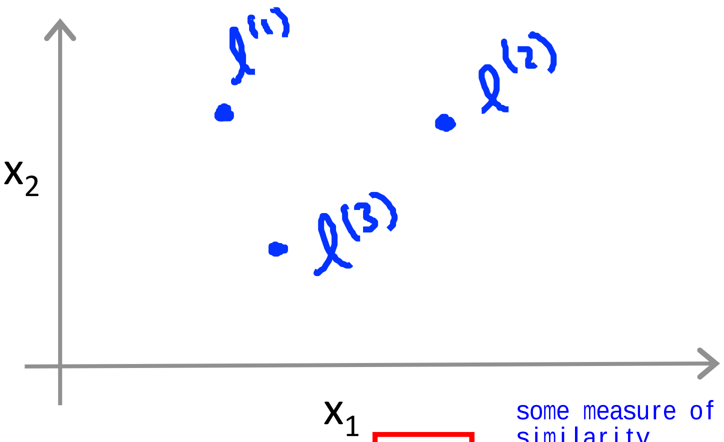
high order polynomials

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Andrew Ng

idea for developing new features

Kernel



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

measures similarity b/c x and l:
1. same, returns 1;
2. different, returns 0-1;

$\|w\|$

$\|x - l^{(1)}\|^2$

Given x:

some measure of similarity

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

$$f_2 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}(x, l^{(3)}) = \exp(\ldots)$$

new features

also denoted in this way

$$\boxed{k(x, l^{(i)})}$$

$\boxed{\text{kernel}}$ (Gaussian kernels)

The similarity function is called the kernel function

The specific kernel here is called the Gaussian Kernels

Andrew Ng

Kernels and Similarity

ignoring x0, which is always 1

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\boxed{\|x - l^{(1)}\|^2}}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

If x if far from $l^{(1)}$:

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

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

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

$l^{(3)} \rightarrow f_3$.

x

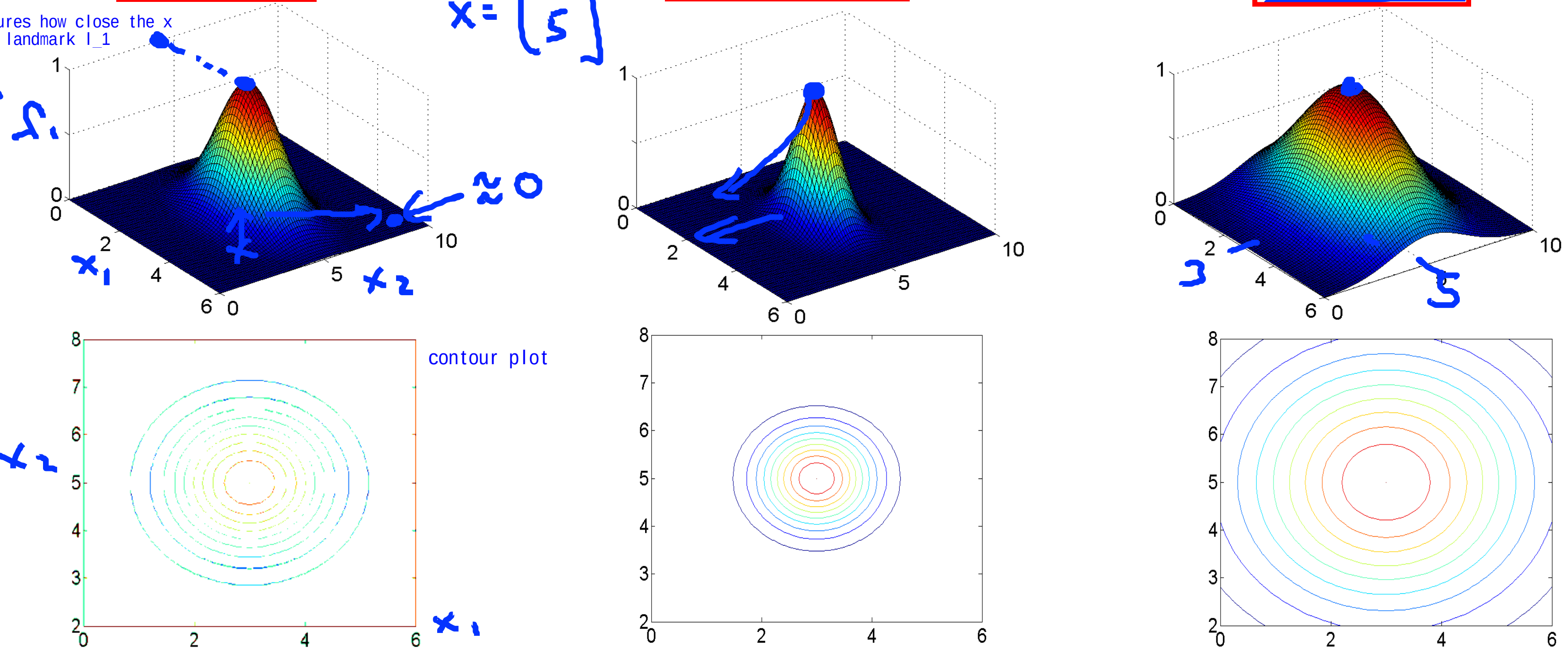
give a x, we can define three new features using ls

Example:

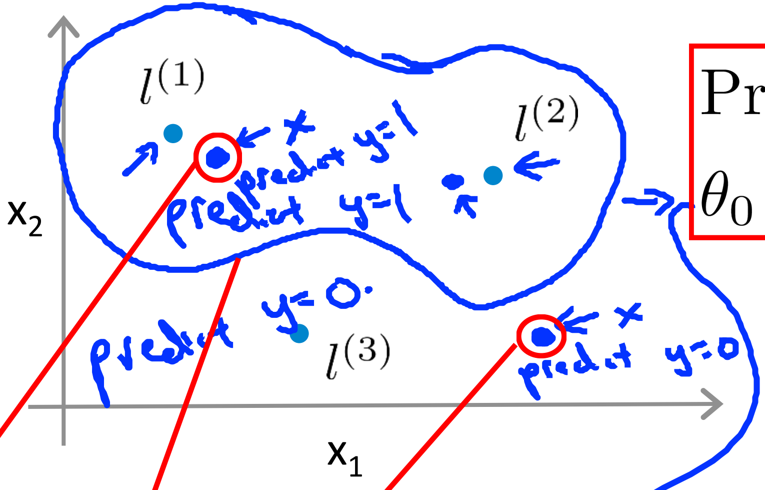
$$l^{(1)} = \boxed{\begin{bmatrix} 3 \\ 5 \end{bmatrix}}, \qquad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\boxed{\sigma^2}}\right)$$

$\boxed{\sigma^2 = 1}$ $\quad$ $x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$ $\quad$ $\boxed{\sigma^2 = 0.5}$ narrower $\quad$ $\boxed{\sigma^2 = 3}$ wider

this measures how close the x
is to the landmark l_1



Andrew Ng



a training example

another training example

for points close to l1 and l2, we predict positive
for points far away from l1 and l2, we predict negative
therefore, we will have a decision boundary shown above.
By using this, we can learn more complex classifiers!

$$\boxed{\text{Predict "1" when} \quad \theta_0+\theta_1 f_1+\theta_2 f_2+\theta_3 f_3 \geq 0}$$

x

$\theta_0 = -0.5, \quad \theta_1 = 1, \quad \theta_2 = 1, \quad \theta_3 = 0$

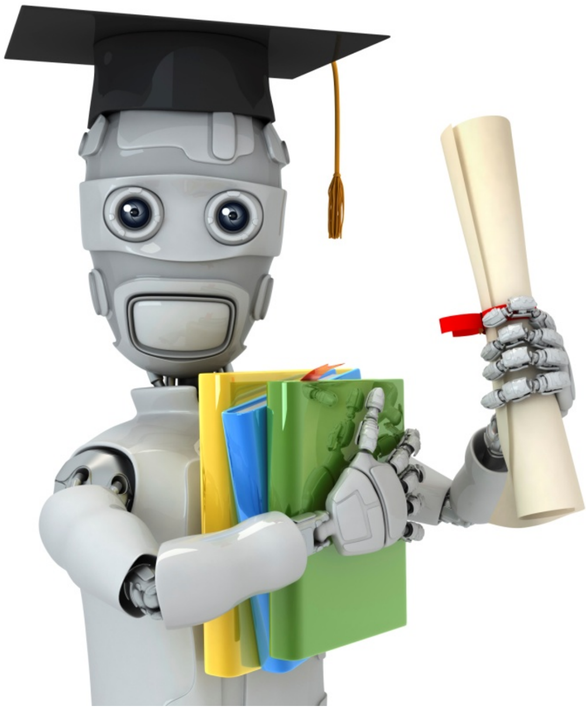
$f_1 \approx 1, \quad f_2 \approx 0, \quad f_3 \approx 0.$

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

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

$f_1, f_2, f_3 \approx 0$

$\rightarrow \theta_0 + \theta_1 f_1 + \dots \approx -0.5 < 0$

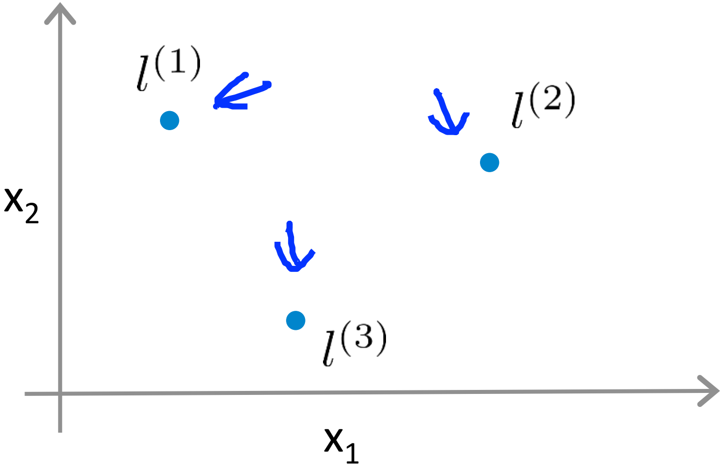


Machine Learning

Support Vector Machines

Kernels II

Choosing the landmarks



Given x:

$$f_i = \text{similarity}(x, l^{(i)})$$

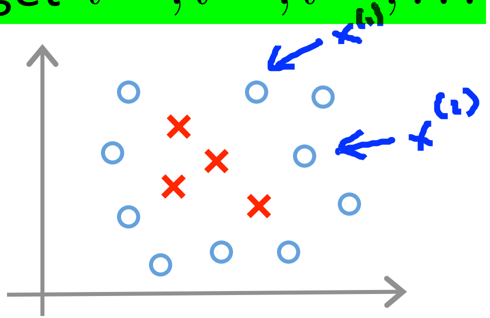
$$= \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$

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

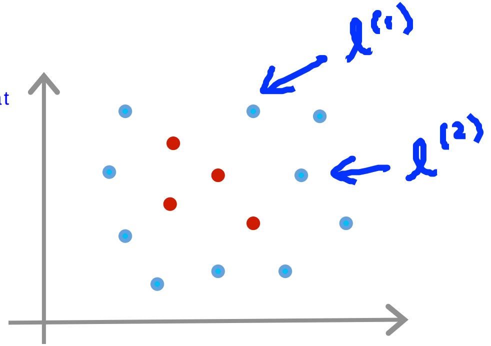
with 1 landmark per location
for each of my training
examples

$l^{(1)}$
$l^{(2)}$
$\vdots$
$l^{(m)}$

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



we first to put landmarks at
the exactly the location of
training examples



Andrew Ng

SVM with Kernels

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

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

feature vector

Given example x:

$x^{(i)}$

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

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

$\ldots$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \qquad f_0 = 1$$

we get a new feature vector that represent my training example xi

For training example $(x^{(i)}, y^{(i)})$:

$x^{(i)} \in \mathbb{R}^{n+1}$ (or $\mathbb{R}^n$)

$x^{(i)} \rightarrow$

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

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

$x^{(i)}$

$\vdots$

$f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp(-\frac{0}{2\sigma^2}) = 1$

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

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \qquad f_0^{(i)} = 1$$

Andrew Ng

SVM with Kernels

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

$\theta \in \mathbb{R}^{m+1}$

→ Predict "y=1" if $\theta^T f \geq 0$

$$\boxed{\theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m}$$

Training: we use the support vector machine algorithm to train the theta parameter

$n = m$

$\cancel{n} = m$

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

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

→ θ_0

we do not regularize the theta_0 term

$$\sum_j \theta_j^2 = \theta^T \theta \leftarrow \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix} \quad (\text{ignoring } \theta_0)$$

$\|\theta\|^2$

$m = 10{,}000$

$$\boxed{\theta^T M \theta}$$

we usually minimize this term, which allow the optimization software to run more efficiently.

SVM parameters:

C $\left(= \frac{1}{\lambda}\right)$. → Large C: Lower bias, high variance. (small λ)

more prone to overfitting

→ Small C: Higher bias, low variance. (large λ)

underfitting

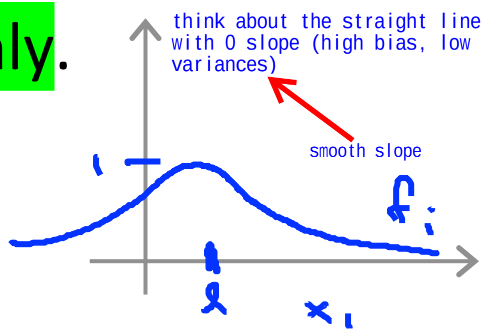
you will need to choose this Parameter in SVM

σ^2

Large σ^2: Features f_i vary more smoothly.

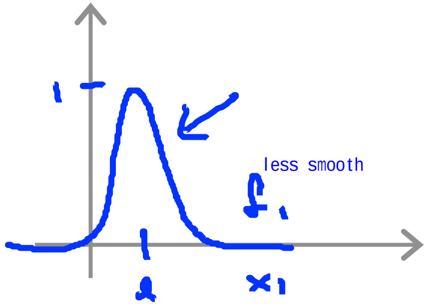
→ Higher bias, lower variance.

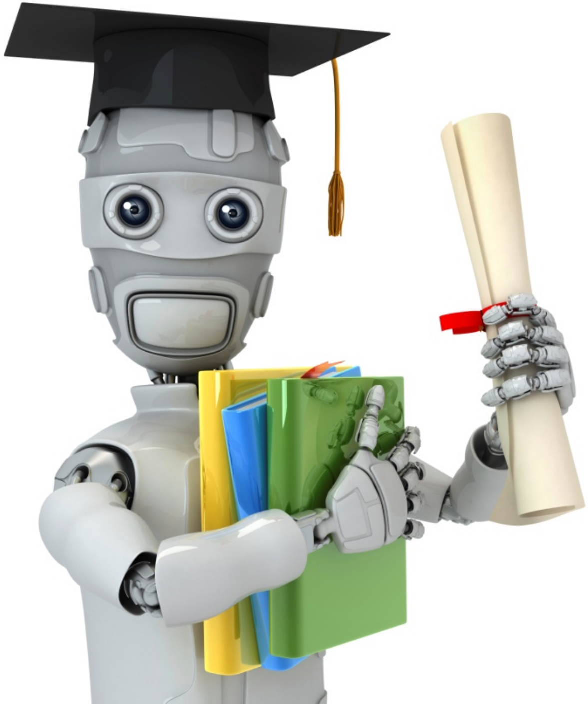
$$\exp\left(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2}\right)$$



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.





Machine Learning

Support Vector Machines

Using an SVM

Poses a new optimization Problem but we do not recommend to write a new software to solve you problem. Just use the available one.

Use SVM software package (e.g. $\boxed{\text{liblinear, libsvm, …}}$) to solve for parameters θ.

2 good optimization solver

Need to specify:

- Choice of parameter C.
- Choice of kernel (similarity function):

is also called the linear kernel

E.g. No kernel (“linear kernel”)

Predict “y = 1” if $\theta^T x \geq 0$

training samples

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

n large, m small

$x \in \mathbb{R}^{n+1}$

features

Gaussian kernel:

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

Need to choose σ^2.

$x \in \mathbb{R}^n$, n small

and/or m large

Andrew Ng

steps for implementing kernels

Kernel (similarity) functions:

f_i $x^{(i)}$ $l^{(j)} = x^{(j)}$

landmarks

```
function f = kernel(x1,x2)
```

$$f = \exp\left(-\frac{\|\mathbf{x1} - \mathbf{x2}\|^2}{2\sigma^2}\right)$$

```
return
```

generate all the features

$x \rightarrow$ f_1, f_2, $\vdots$, f_m

Note: Do perform feature scaling before using the Gaussian kernel.

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

$\boxed{\|x - l\|^2}$

$v = x - l$

$\|v\|^2 = v_1^2 + v_2^2 + \cdots + v_n^2$

$= \underbrace{(x_1 - l_1)^2}_{1000 \text{ feet}^2} + \underbrace{(x_2 - l_2)^2}_{1\text{-}5 \text{ bedrooms}} + \cdots + (x_n - l_n)^2$

perform feature scaling!!!

Andrew Ng

Other choices of kernel

linear and Guassian kernels are two most commonly used kernels

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.
(Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

main form

$(x^T l + \text{constant})^{\text{degree}}$

Many off-the-shelf kernels available:

the idea is that
if x and l are very close
the product tends to be large

- Polynomial kernel: $k(x, l) = (x^T l)^2 + 0$

people do not use it too much

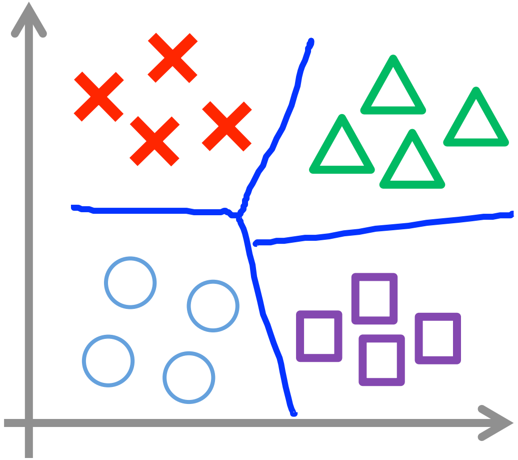
$(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$

string

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

$\text{sim}(x, l)$

Multi-class classification



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

K Classes

as in logistic classification

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 $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

y=1 y=2 .... θ=K

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

Logistic regression vs. SVMs

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

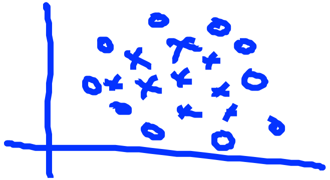
If n is large (relative to m): (e.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)

Use logistic regression, or SVM without a kernel ("linear kernel")

If n is small, m is intermediate: ($n = 1 - 1000$, $m = 10 - 10{,}000$)

Use SVM with Gaussian kernel

Gaussian kernel is slow!



If n is small, m is large: ($n = 1 - 1000$, $m = 50{,}000+$)

Create/add more features, then use logistic regression or SVM without a kernel

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