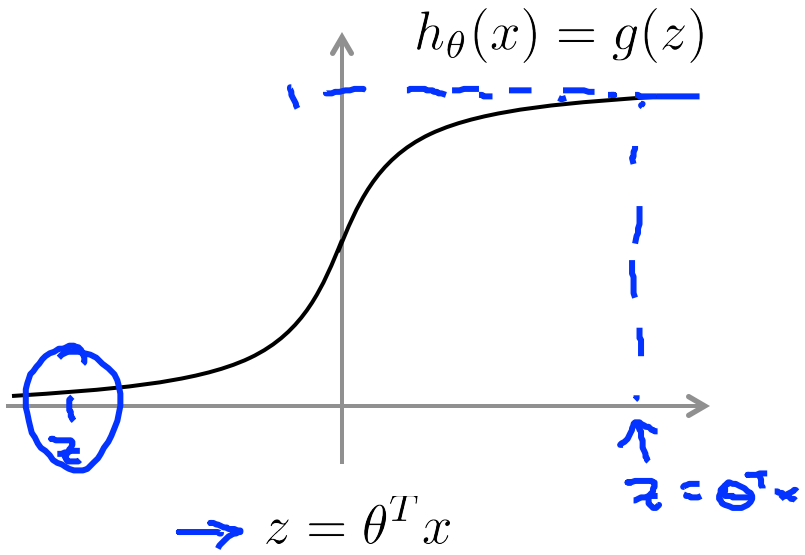
With supervised learning, the performance of many supervised learning algorithms will be pretty similar, and what matters less often will be whether you use learning algorithm a or learning algorithm b, but what matters more will often be things like the amount of data you create these algorithms on, as well as your skill in applying these algorithms. Things like your choice of the features you design to give to the learning algorithms, and how you choose the regularization parameter, and things like that.
But, there's one more algorithm that is very powerful and is very widely used both within industry and academia, and that's called the
support vector machine.
And compared to both logistic regression and neural networks, the Support Vector Machine, or SVM sometimes gives a cleaner, and sometimes more powerful way of learning complex non-linear functions.

Support Vector Machines

Optimization objective

Alternative view of logistic regression

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



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

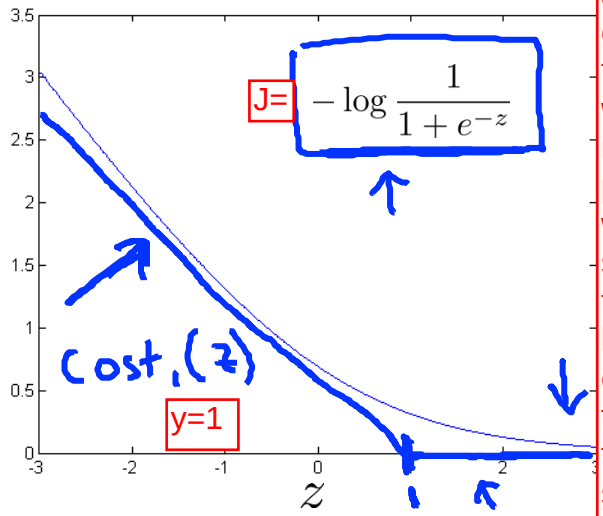
If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

Alternative view of logistic regression

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

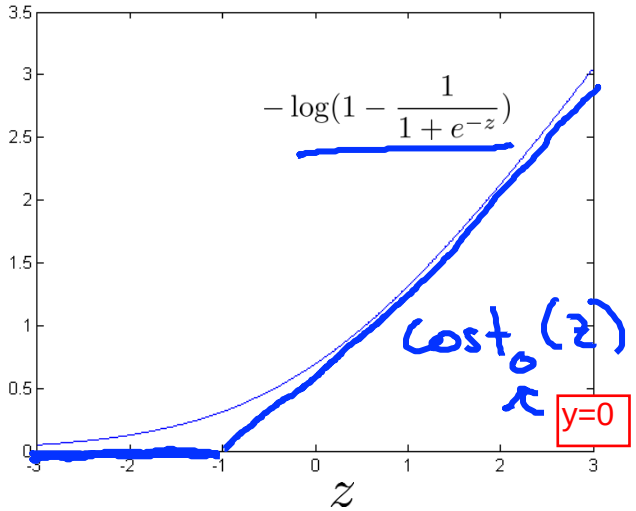
$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

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



when z is large, that corresponds to a small value of cost function. And this explains why, when logistic regression sees a positive example, with y=1, it tries to set theta transport x to be very large because that corresponds to this term, in the cost function, being small.

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



Support vector machine

Logistic regression:

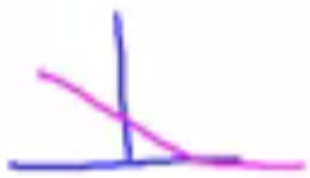
$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Support vector machine:

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

Support vector machine

Logistic regression:



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

A B

Support vector machine:

$$\min_\theta \; C\sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2}\sum_{i=0}^{n}\theta_j^2$$

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

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

$A + \lambda B$

$C\,A + B$

$$C = \frac{1}{\lambda}$$

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

SVM hypothesis

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

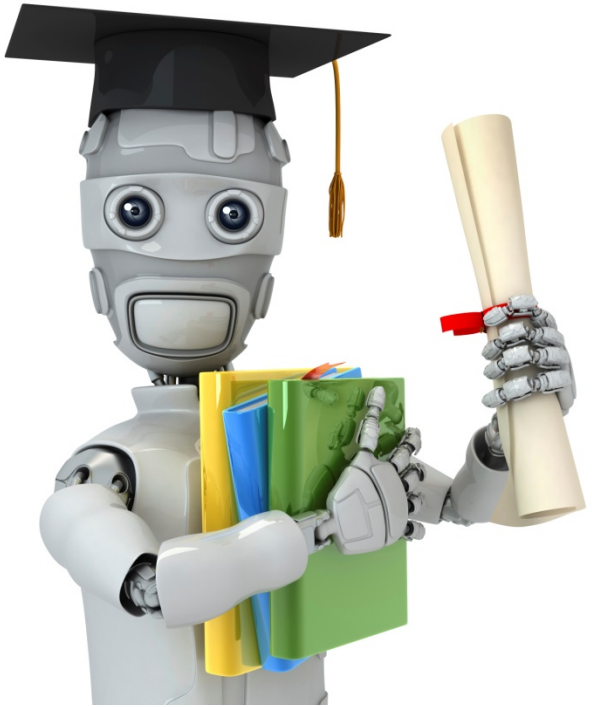
Hypothesis:

SVM hypothesis

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

Hypothesis:

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



Machine Learning

Support Vector Machines

Large Margin Intuition

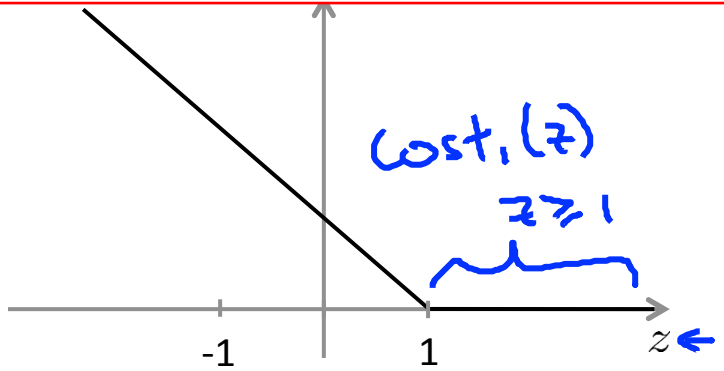
Sometimes people talk about support vector machines, as large margin classifiers

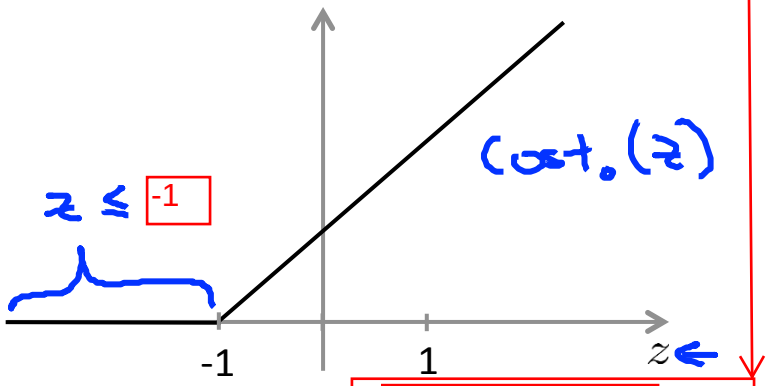
Support Vector Machine

all we need to get the classification right is >= or < 0, but svm wants more than that (use 1) ! so this builds in an extra safety factor or safety margin factor into SVM,logistic regression can do it too

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

what it takes to make the cost function small ?





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)

$\theta^T x \geqslant 1$

$\theta^T x \leqslant -1$

$C = 100,000$

C is very large, what SVM will do ?

SVM Decision Boundary

If C is very large, then when minimizing this optimization objective, we're going to be highly motivated to choose a value, so that this first term =0

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

if C is very large

what does it take to make this first term = 0 (when C huge)

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

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

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

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

now when we choose parameters and sure that this first term =0, then what left is

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

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

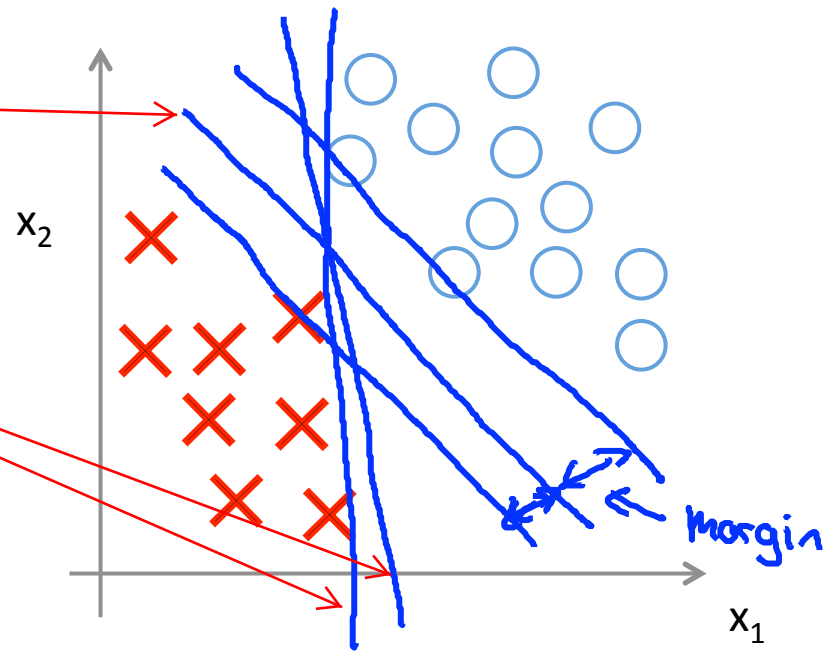
$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

subject to the constrain that

linearly separable : there exists a straight line, although there is many a different straight lines, they can separate the positive and negative examples perfectly.

SVM Decision Boundary: Linearly separable case

The Support Vector Machines will instead choose this decision boundary,and that seems like a much better decision boundary than others



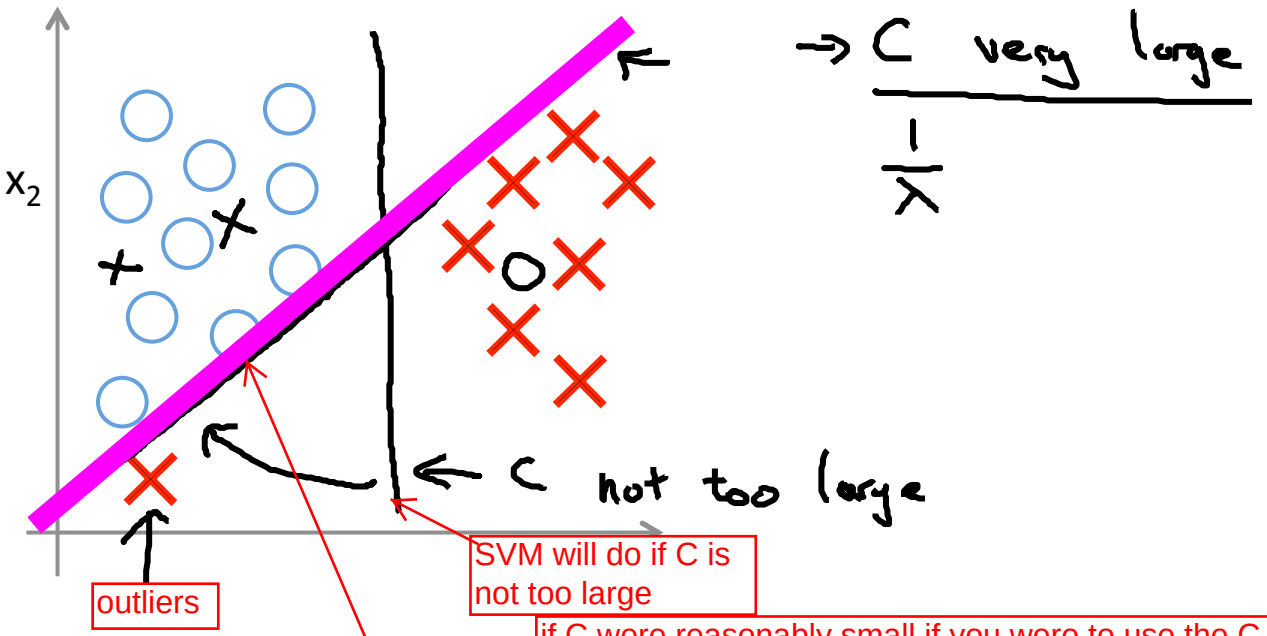
Large margin classifier

this distance is called "margin" of the SVM and this gives the SVM a certain robustness, because it tries to separate the data with a large a margin,so SVM is also called a "large margin classifier" and this is a consequence of the optimization problem we wrote down on the previous slide.

C parameter is a positive value that controls the penalty for misclassied training examples. A large C tells the SVM to try to classify all the examples correctly.

Large margin classifier in presence of outliers

use large margin classifier it can be sensitive to outsiers

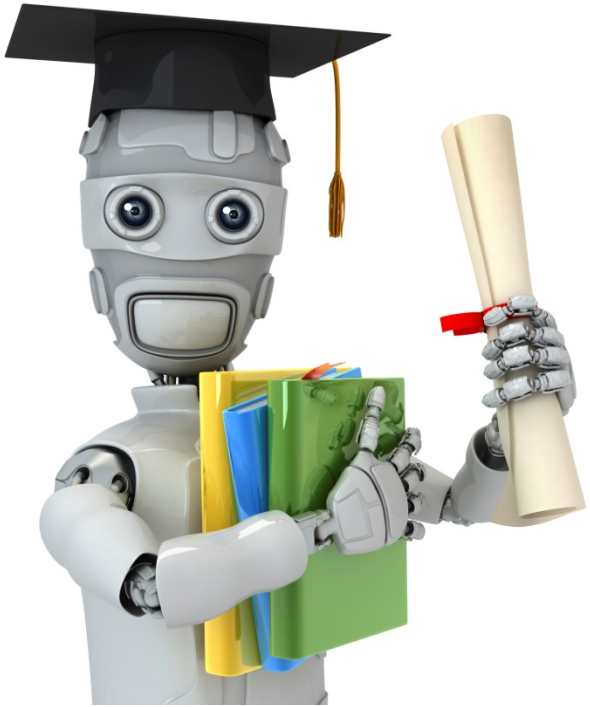


if you have a outlier like this, it's really not clear that based on the single outlier it's actually a good idea to change my decision boundary from the black one over to the magenta one,if the regularization parameter C were very large, then this is actually what SVM will do, it will change the boundary from black to magenta

SVM will do if C is not too large

my guess: large C (small lambda), then no regularization is used => algorithm is overfitting => sensitive to outliers.

if C were reasonably small if you were to use the C, then you still end up with this black decision boundary, the outliers might be ignored.



Machine Learning

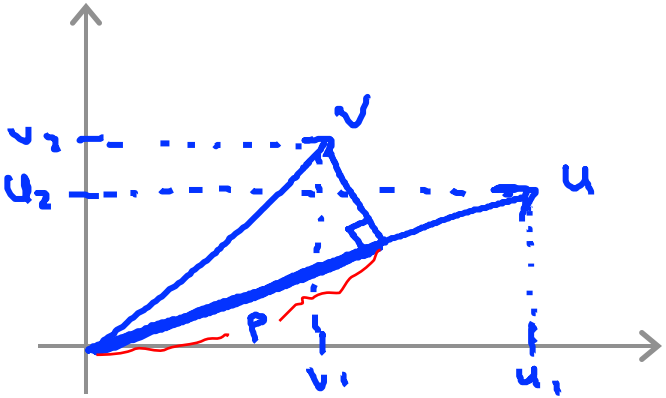
Support Vector Machines

The mathematics behind large margin classification (optional)

why SVM is a large margin classifier :

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

norm of u

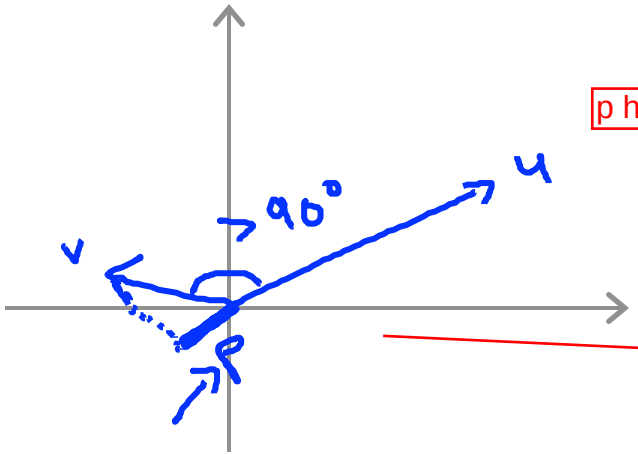
$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

p = length of projection of v onto u.

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

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

p has sign — Signed



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

$p < 0$

SVM Decision Boundary

svm in the optimization objective is minimizing the squared norm of the square length of the parameter vector theta.

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

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

$$= \|\theta\|$$

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

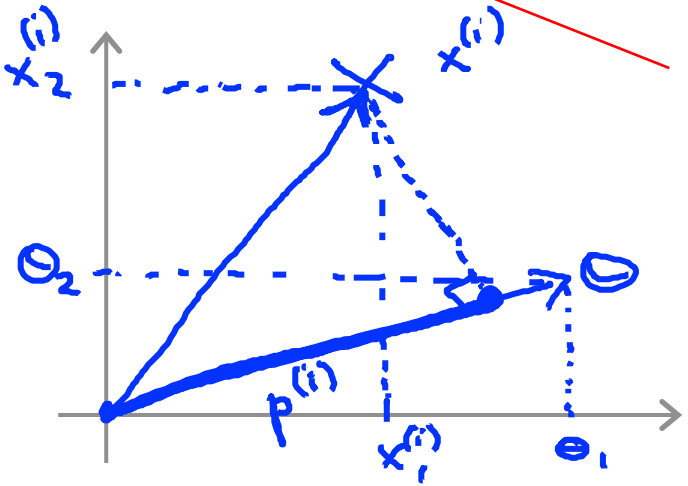
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

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

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

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

$u^T v$



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

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

x 在 theta 上的投影 * |theta|

by making the margin large(by these P1, P2, P3 large) the SVM can end up with a smaller value for the norm of theta which is what it is trying to do in the objective.

SVM Decision Boundary

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

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

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

C very large

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

Simplification: $\theta_0 = 0$

$\theta_0 = 0$

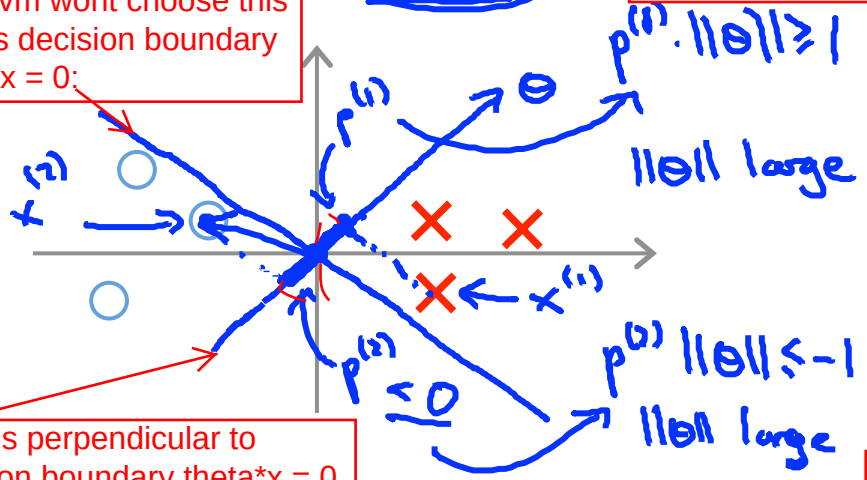
decision boundary not pass origins

decision boundary only pass origins

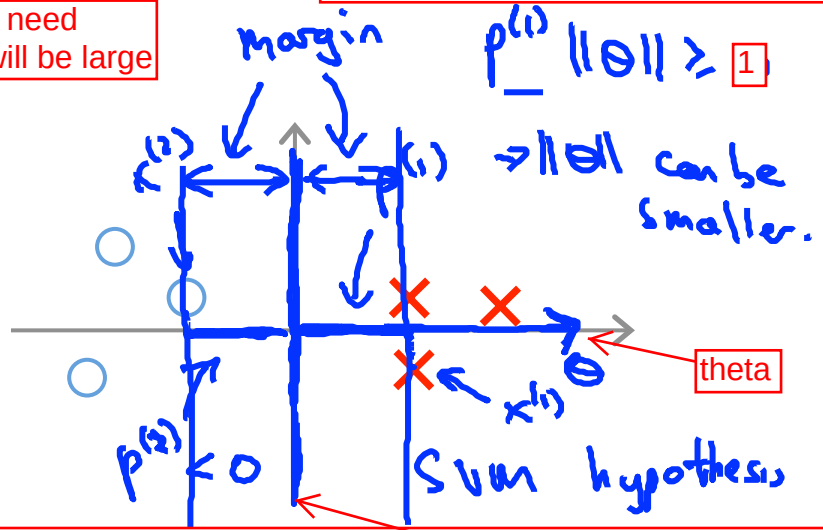
p1is small, because we need p*theta>=1,then theta will be large

p1is large, because we need p*theta>=1,then theta will be smaller

why svm wont choose this line as decision boundary theta*x = 0:



theta is perpendicular to decision boundary theta*x = 0



why svm choose this line as decision boundary theta*x = 0:

SVM Decision Boundary

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

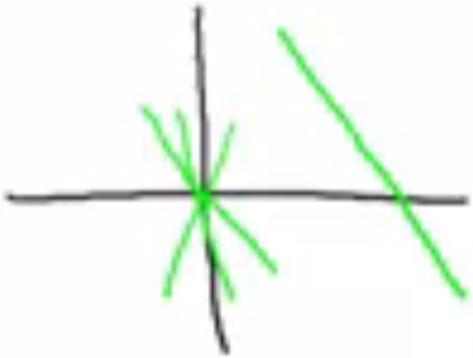
s.t. $p^{(i)} \cdot \|\theta\| \geq 1$ if $y^{(i)} = 1$

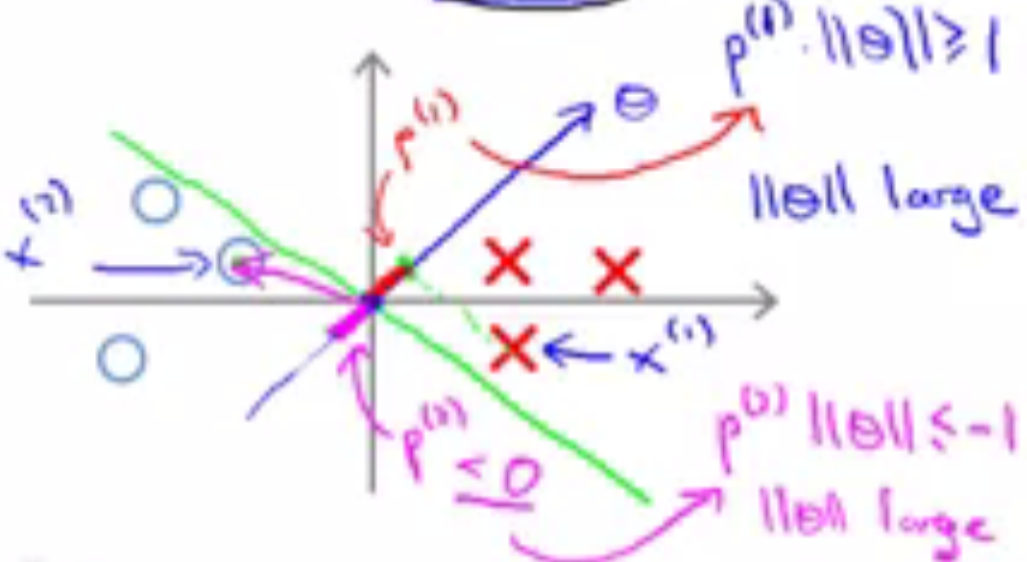
$p^{(i)} \cdot \|\theta\| \leq -1$ if $y^{(i)} = 1$

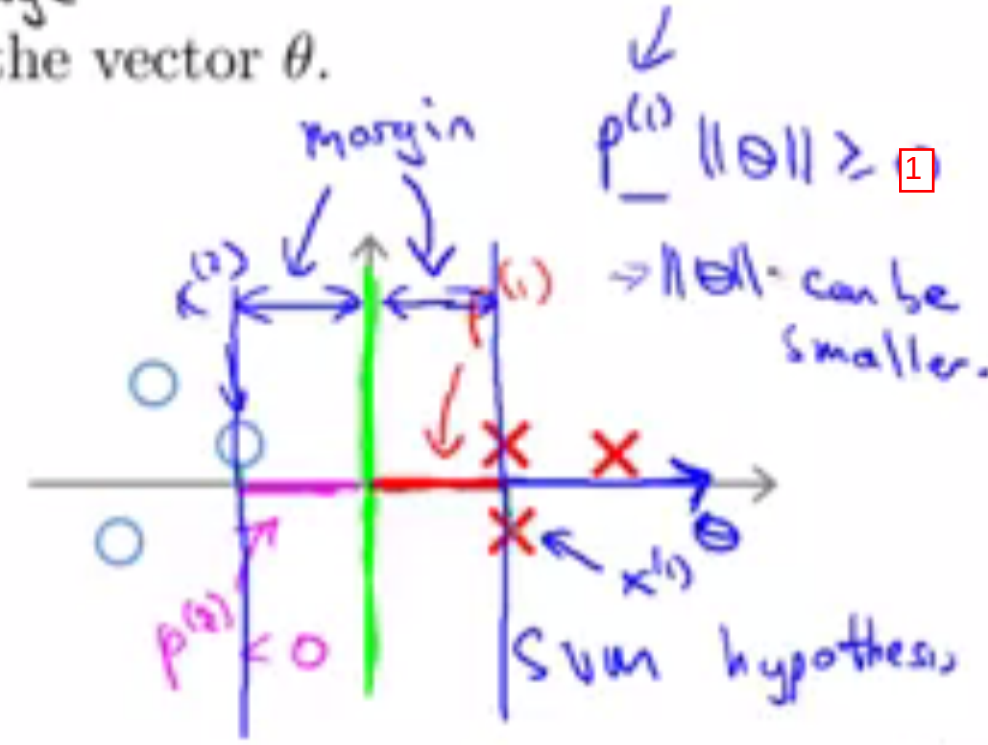
C very large

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

Simplification: $\theta_0 = 0$



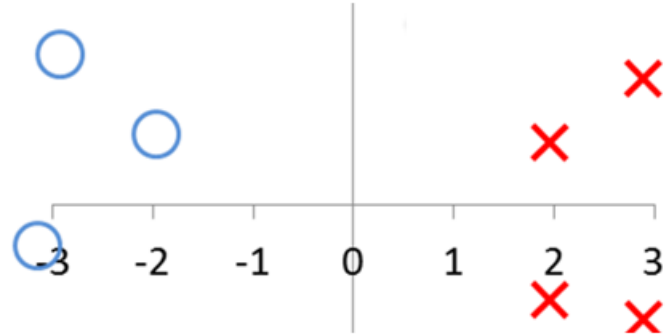




The SVM optimization problem we used is:

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

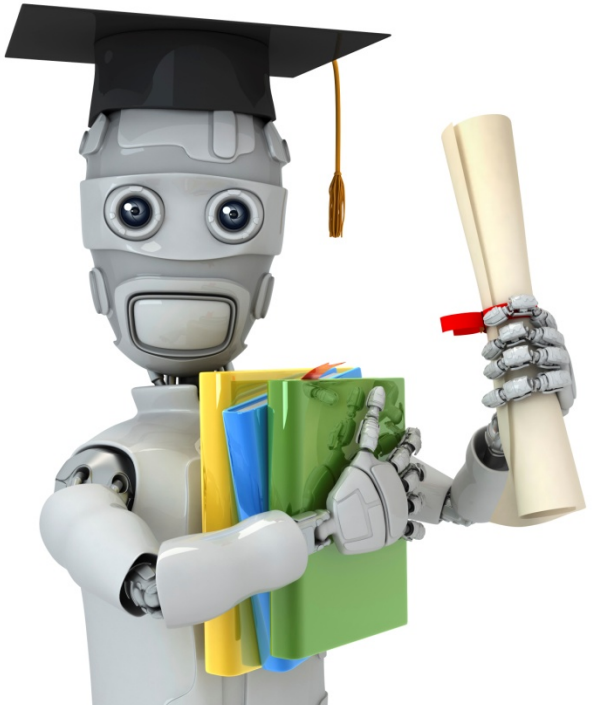
$$\text{s.t. } \|\theta\| \cdot p^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$
$$\|\theta\| \cdot p^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$



where $p^{(i)}$ is the (signed - positive or negative) projection of $x^{(i)}$ onto θ. Consider the training set above. At the optimal value of θ, what is $\|\theta\|$?

- 1/4
- 1/2

Correct

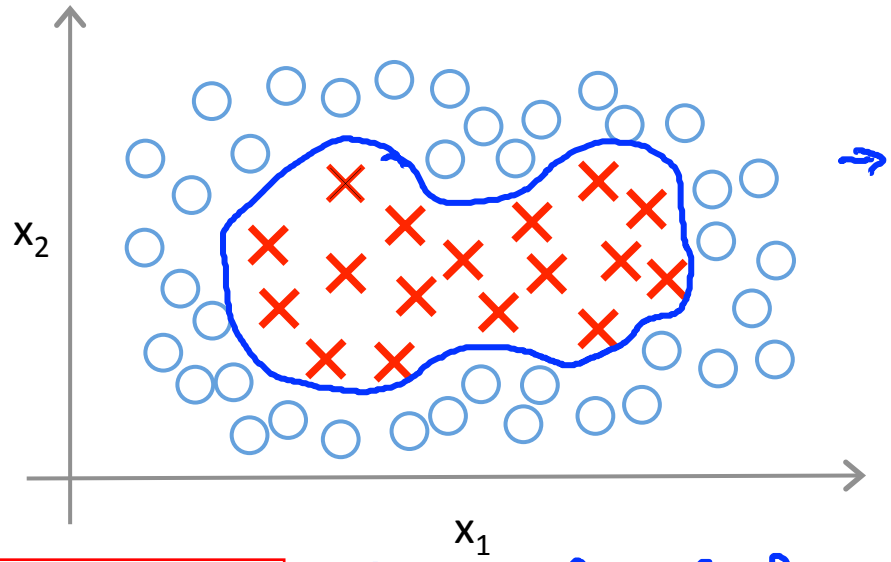


Machine Learning

Support Vector Machines

Kernels I

Non-linear Decision Boundary



you can do complex polynomial features :

Predict $y = 1$ if

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

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

another way to write it : $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

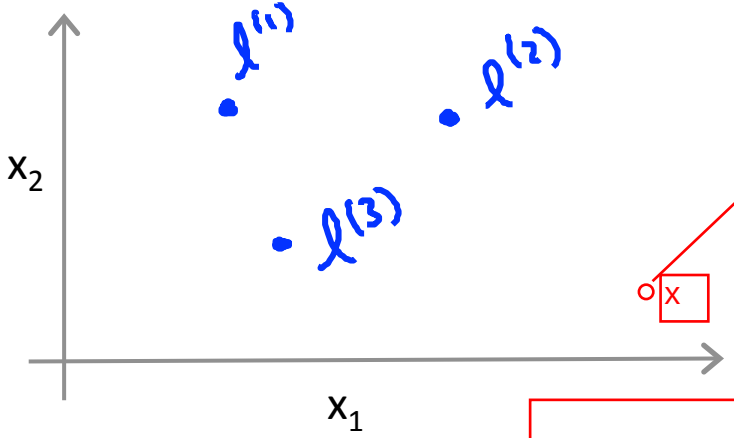
$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_1^2, \ldots$

is there a different/better choice of features than this high order polynomials because it's not clear that this high order polynomial is what we want, and for computer vision the input is an image with lots of pixels, we saw how using high order polynomials becomes very computationally expensive because there are a lot of these higher order polynomial terms.

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

its one idea to find out features f1, f2, f3:

Kernel



given an example x

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

Given x:

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

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

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

$\|w\|$

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

kernel (Gaussian kernels)

one example of similarity function is Gaussian kernel

Kernels and Similarity

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

if x close to l1

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

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

if x far from l1

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

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

given the the training example X, we can now compute three new features: f1, f2, and f3, given three landmarks

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

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

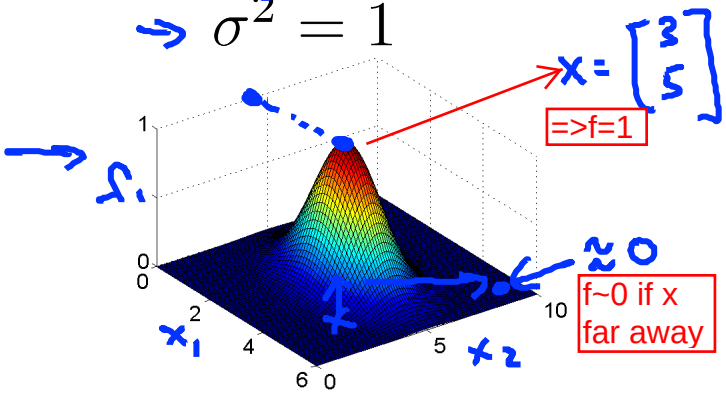
$l^{(3)} \rightarrow f_3$.

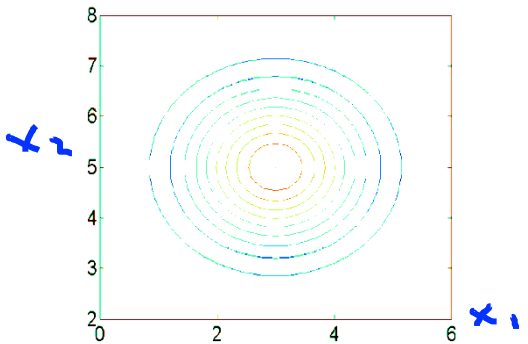
this is a feature, f1 measures how close X is to the first landmark and if varies between 0 and 1 one depending on how close X is to the first landmark l1.

Example:

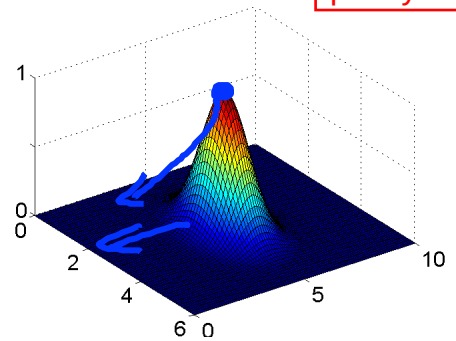
$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, $f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

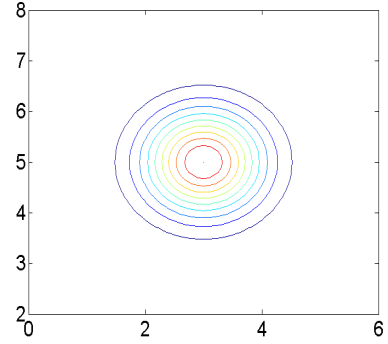
$\sigma^2 = 1$



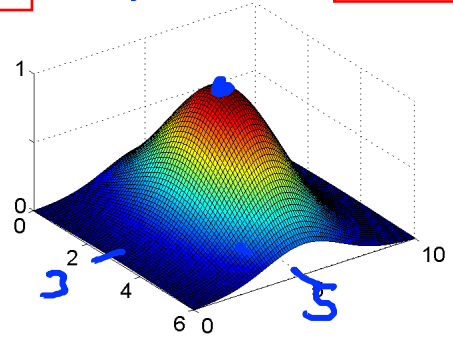


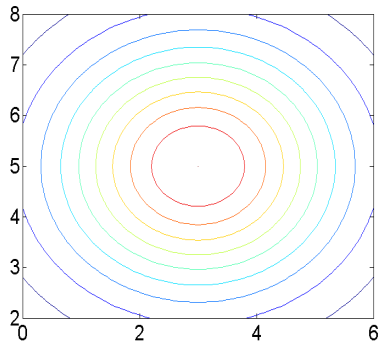
$\sigma^2 = 0.5$ smaller => f1 falls quickly



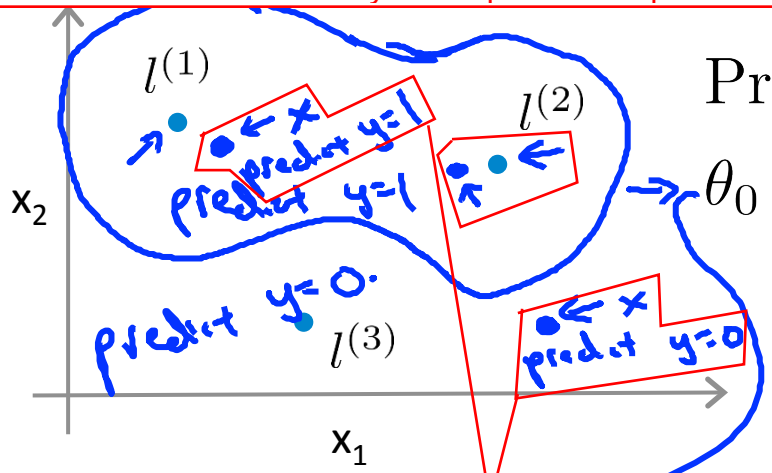


$\sigma^2 = 3$ larger => f1 falls slowly





for points near l1 and l2 we end up predicting positive. And for points far away from l1 and l2, we end up predicting that the class is equal to 0. so What we end up doing is that the decision boundary of this hypothesis would end up looking something like this where inside this red decision boundary would predict Y equals 1 and outside we predict Y equals 0.



Predict "1" when

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

assume we already train the model and get theta

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

if i have x here => $f_1 \approx 1, \quad f_2 \approx 0, \quad f_3 \approx 0.$

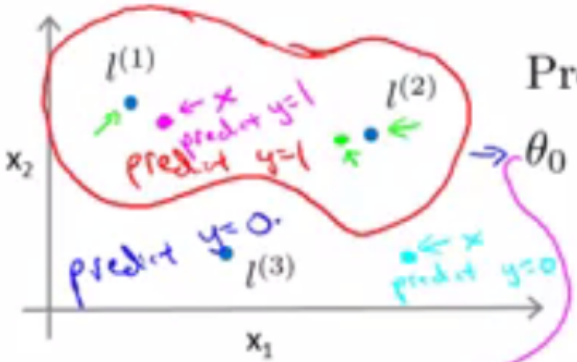
$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0$

$= -0.5 + 1 = 0.5 \geq 0$ => predict y=1

if i have x here => $f_1, f_2, f_3 \approx 0$

$\theta_0 + \theta_1 f_1 + \ldots \approx -0.5 < 0$

=> predict y=0



Predict "1" when

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

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

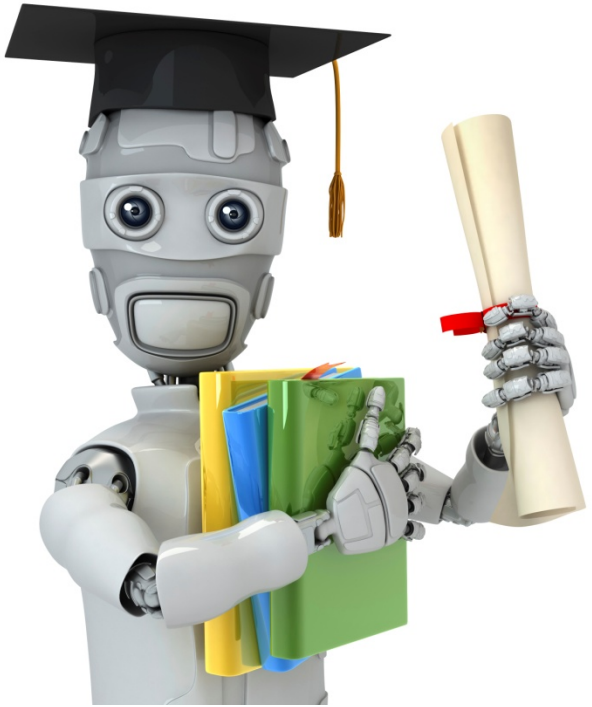
$f_1 \approx 1, \quad f_2 \approx 0, \quad f_3 \approx 0.$

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

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

$f_1, f_2, f_3 \approx 0$

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

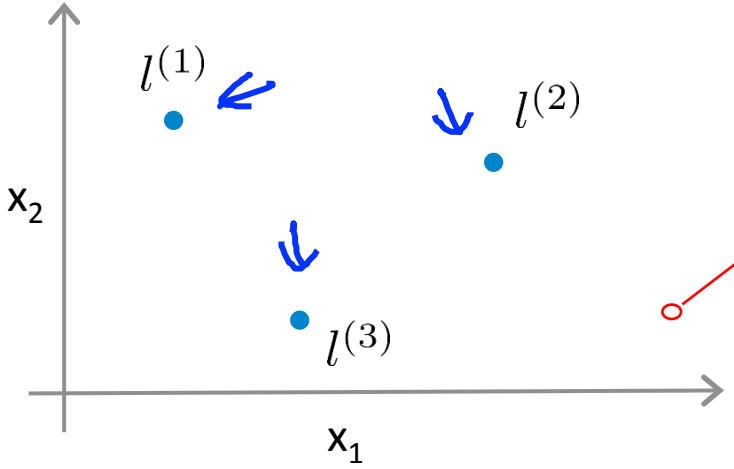


Machine Learning

Support Vector Machines

Kernels II

Choosing the landmarks



Given x:

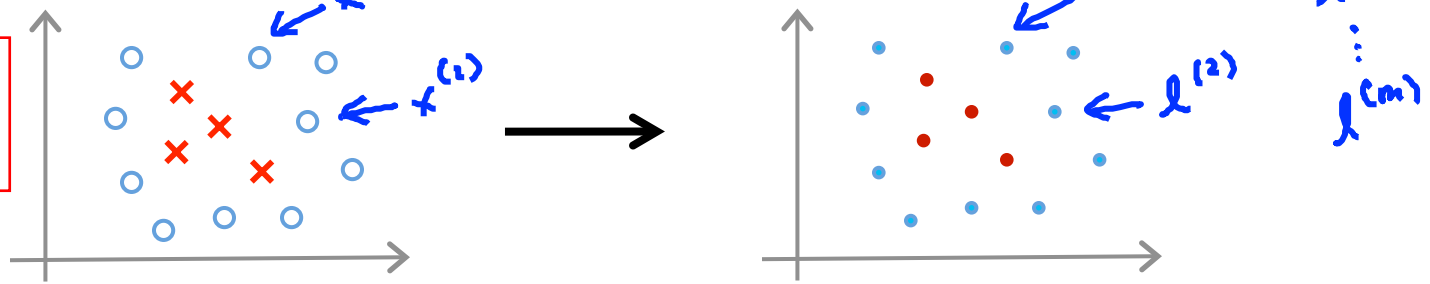
$$f_i = \text{similarity}(x, l^{(i)})$$

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

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

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

We're just going to put landmarks as exactly the same locations as the training examples.



SVM with Kernels

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

Given example x:

$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$\ldots$$

$x^{(i)}$

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

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

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

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

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

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

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

$f_0^{(i)} = 1$

第 i 個點與其他點有多近? (0~1表示)

If you already have a learning set of theta, then if you given a value of x and you want to make a prediction:

SVM with Kernels

m training examples (m landmarks)

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

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

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

$\theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m$

$n = m$

Training:

how do you get theta:sole this min problem

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

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

dont do regularization of theta 0

in svm this usually done differently

$\sum_{j=1}^{n=m} \theta_j^2 = \theta^T \theta \leftarrow \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ (ignore θ_0)

$\|\theta\|^2$

$\theta^T M \theta$

$m = 10{,}000$

support vector machines and kernels tend go particularly well together. Whereas, logistic regression and kernels, you know, you can do it, but this would run very slowly.

SVM parameters:

cause

C ($= \frac{1}{\lambda}$).

→ Large C: Lower bias, high variance.

→ Small C: Higher bias, low variance.

(small λ) =not using regularization

(large λ) using regularization

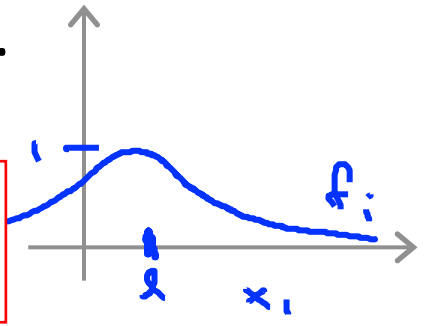
σ^2

Large σ^2: Features f_i vary more smoothly.

cause Higher bias, lower variance.

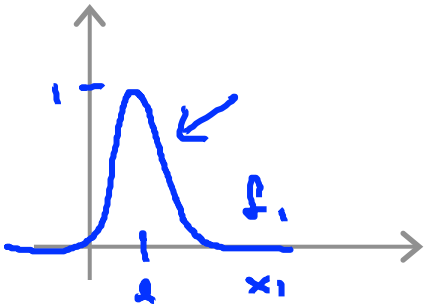
$\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$

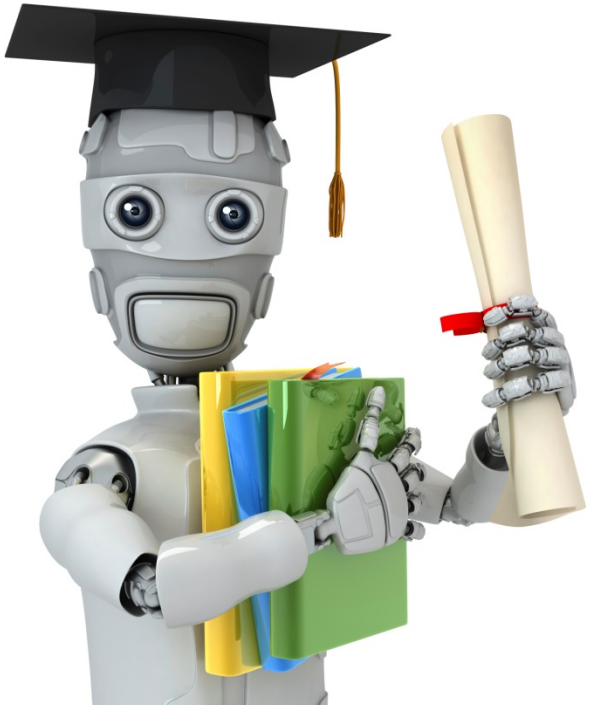
you tend to get a hypothesis that varies slowly, or varies smoothly as you change the input x => not too sensitive to noise



Small σ^2: Features f_i vary less smoothly.

cause Lower bias, higher variance.





Machine Learning

Support Vector Machines

Using an SVM

Use SVM software package (e.g. liblinear, libsvm, …) to solve for parameters θ.

Need to specify:

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

you want to fit a linear decision boundary and not try to fit a very complicated nonlinear function, because might not have enough data. And you might risk overfitting, if you're trying to fit a very complicated function

standard linear classifier

E.g. No kernel ("linear kernel")

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

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

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

→ n large, m small

- 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.

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

and/or n large

Kernel (similarity) functions:

第 i 個點與其他點有多近? (0~1表示)

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

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

```
return
```

$x \rightarrow \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$

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

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

$\|x - l\|^2$

$v = x - l$

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

$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$

1000 feet^2 1-5 bedrooms

those distances will be almost essentially dominated by the sizes of the houses and the number of bathrooms would be largely ignored. so to avoid this in order to make a machine work well, do perform future scaling.

Other choices of kernel

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).

Many off-the-shelf kernels available:

- Polynomial kernel:

$k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$

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

Usually it is used only for data where X and l are all strictly non negative, and so that ensures that these inner products are never negative.

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

used if your input data is text strings or other types of strings.

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

There are sort of more esoteric kernels that you can use to measure similarity between different objects.

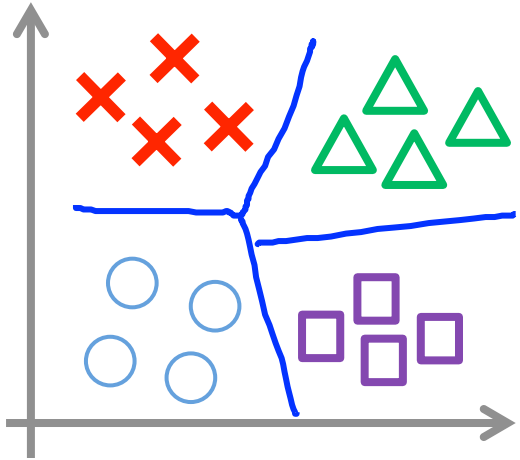
Suppose you are trying to decide among a few different choices of kernel and are also choosing parameters such as C, σ^2, etc. How should you make the choice?

- Choose whatever performs best on the training data.
- Choose whatever performs best on the cross-validation data.

Correct

- Choose whatever performs best on the test data.
- Choose whatever gives the largest SVM margin.

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

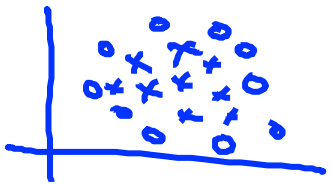
y=1 y=2 ... class y=K

Logistic regression vs. SVMs

if you have many features with smaller training sets, a linear function will probably do fine, and you don't have really enough data to fit a very complicated nonlinear function.

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



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

logistic regression and SVM without the kernel, are similar algorithms and they will usually do pretty similar things and give similar performance, but depending on your implementational details, one may be more efficient than the other

optimization problem that the SVM has is a convex optimization problem and so the good SVM optimization software packages will always find the global minimum or something close to it.