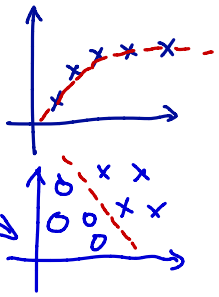
Machine Learning

Supervised Learning { Regression → price prediction
Classification → Benign or malignant



Unsupervised Learning → { google news
cocktail party problem
market segments

---

m: number of training examples $(x, y) \rightarrow$ 1 training example
x's: input / features
y's: output / target. $(x^{(i)}, y^{(i)})$

Cost Function Hypothesis: $h(x) = \theta_0 + \theta_1 x + \cdots$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$ Square Error Funct

Object: $\min_\theta J(\theta)$ → cost function

Gradient descent $\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$ $\alpha > 0$ Learing Rate

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)^2$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) \cdot \frac{\partial h_\theta(x^{(i)})}{\partial \theta_j}$$

Multiple features

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

Define $x_0^{(i)} = 1$

$$\Rightarrow \quad X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

$$= \theta^T X$$

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

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

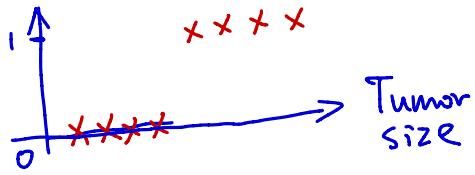
Featur Scaling (Normalization) : $x_i := \frac{x_i - \mu_i}{s_i}$

Normal Equation

$$\theta = (X^T X)^{-1} X^T y$$

code : $\theta = \text{pinv}(X' * X) * X' * y$

Classification



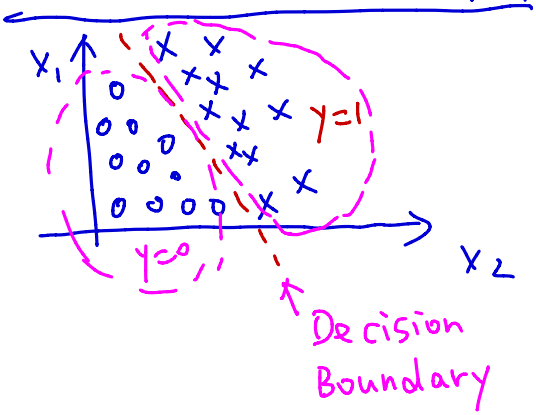
Logistic Regression: $0 \le h(\theta) \le 1$

↳ classification

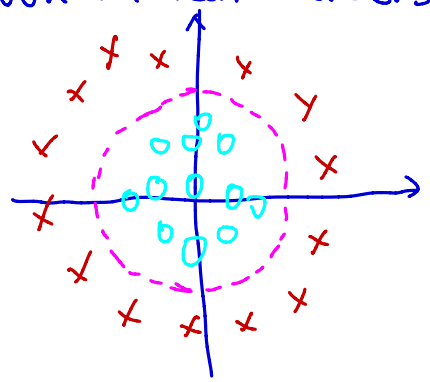
Def: $g(z) = \frac{1}{1+e^{-z}}$. $h(\theta) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

$h_\theta(x) = P(y=1 \mid x; \theta)$ → probability that $y=1$, given x & θ.

Decision Boundary



Non-linear decision boundary

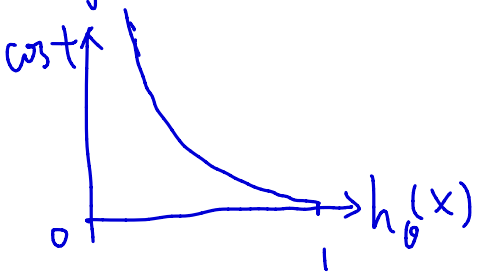


$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$.

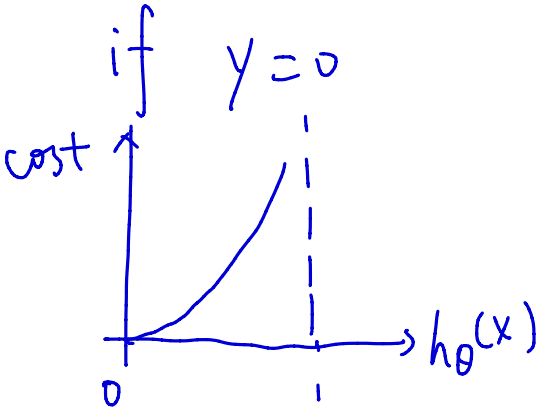
Cost Function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & y=1 \\ -\log(1-h_\theta(x)), & y=0 \end{cases}$$

if $y=1$



if $y=0$



Logistic Regression : $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta(x^{(i)}), y^{(i)}\right)$

$$cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & y=1 \\ -\log(1-h_\theta(x)), & y=0 \end{cases}$$

$y=0$ or $y=1$.

$\Rightarrow Cost(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$

$\Rightarrow J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))\right]$, $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

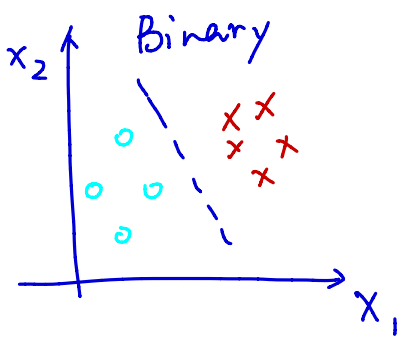
$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

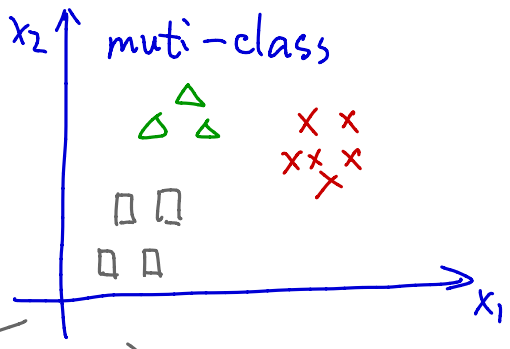
$$= \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left[\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}\right]$$

Advanced Optimization :

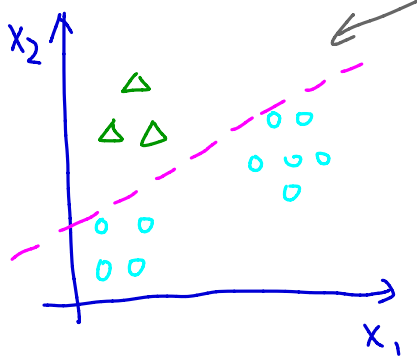
- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Multiclass Classification

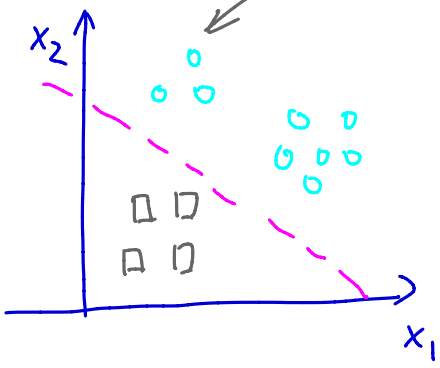




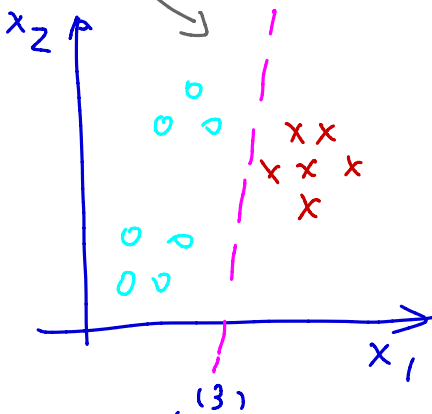
One-vs-all (one-vs-rest)



$h_\theta^{(1)}(x)$



$h_\theta^{(2)}(x)$



$h_\theta^{(3)}(x)$

$$h_\theta^{(i)}(x) = P(y=i \mid x; \theta)$$

$\Rightarrow$ Train a Logistic classifier $h_\theta^{(i)}(x)$ for each class i

For a new input x, pick $\max_i h_\theta^{(i)}(x)$

Overfitting

1. Reduce number of features
2. Regularization

Regularization Cost Function

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

Regularized linear Regression:

$$\begin{cases} \theta_0 = \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)} \\ \theta_j = \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \end{cases}$$

Normal Equation:

$$\theta = \left(X^TX + \lambda\underbrace{\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}}_{(n+1)\times(n+1)}\right)^{-1}X^Ty$$

Regularized logistic regression:

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

$$\begin{cases} \theta_0 = \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)} \\ \theta_j = \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \end{cases}$$

~~$\lambda = 0$~~ $\lambda = 1$

θ_j ↓.

Neural Network | Model:

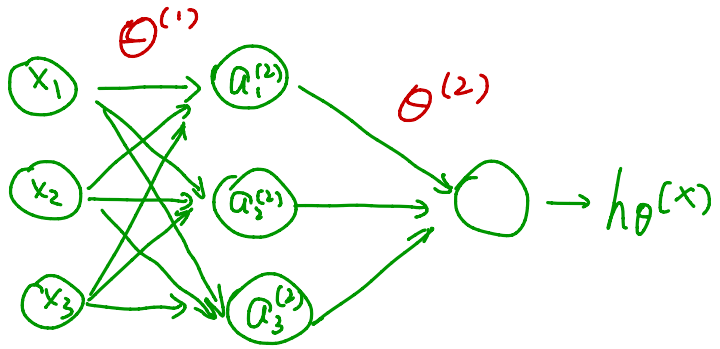


Fig. 1

Notation:

$a_i^{(j)}$: "activation" of unit i in layer j

$\theta^{(j)}$: matrix of weights controlling function mapping from j to $j+1$

In Fig. 1, there are 3 input units, 3 hidden units.

$$\Rightarrow \quad a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) \quad z_1^{(2)}$$

$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) \quad z_2^{(2)}$$

$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) \quad z_3^{(2)}$$

$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}) \quad z^{(3)}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_2^{(3)} \end{bmatrix} \quad \Rightarrow \quad z^{(2)} = \theta^{(1)} x = \theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

Then add $a_0^{(2)} = 1 \rightarrow a^{(2)} \in \mathbb{R}^4$

$$\Rightarrow \quad z^{(3)} = \theta^{(2)} a^{(2)}, \quad a^{(3)} = g(z^{(3)})$$

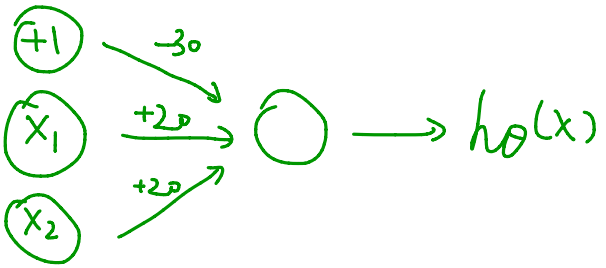
$$\Rightarrow \quad \boxed{\begin{aligned} z^{(j)} &= \theta^{(j-1)} a^{(j-1)} \\ a^{(j)} &= g(z^{(j)}) \end{aligned}}$$

Examples

AND:

$x_1, x_2 \in \{0, 1\}$

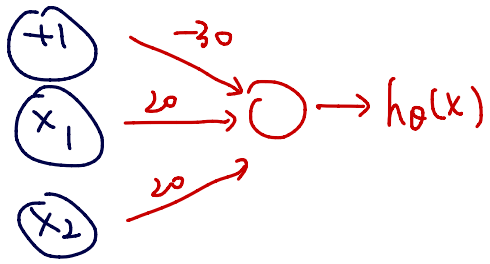
$y = x_1$ AND x_2



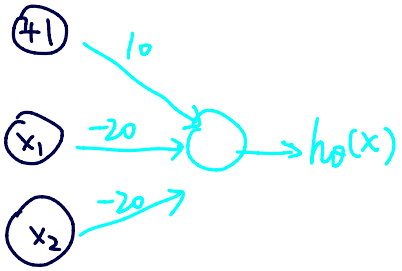
$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

$\theta_{10}^{(1)}$ $\theta_{11}^{(1)}$ $\theta_{12}^{(1)}$

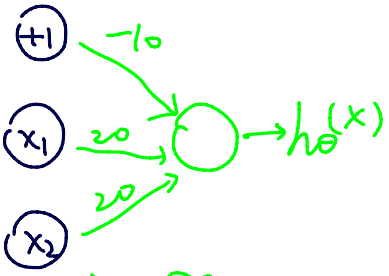
Multi-Layers Example



x_1 AND x_2

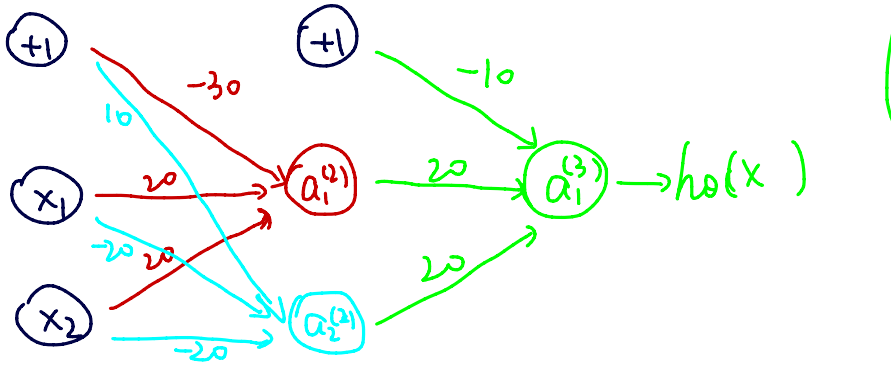


(NOT x_1) AND (NOT x_2)



x_1 OR x_2

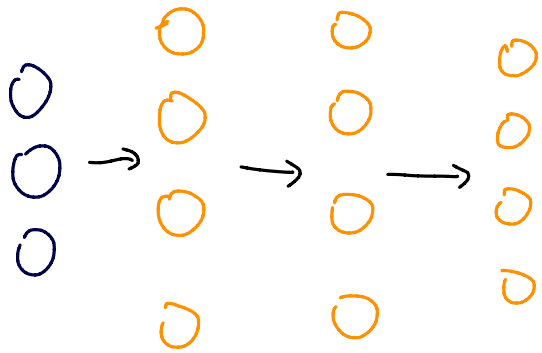
$\Rightarrow$



$\Rightarrow x_1$ XNOR x_2

Multiclass Classification | one-vs-all

Pedestrian. Car Motorcycle. Truck

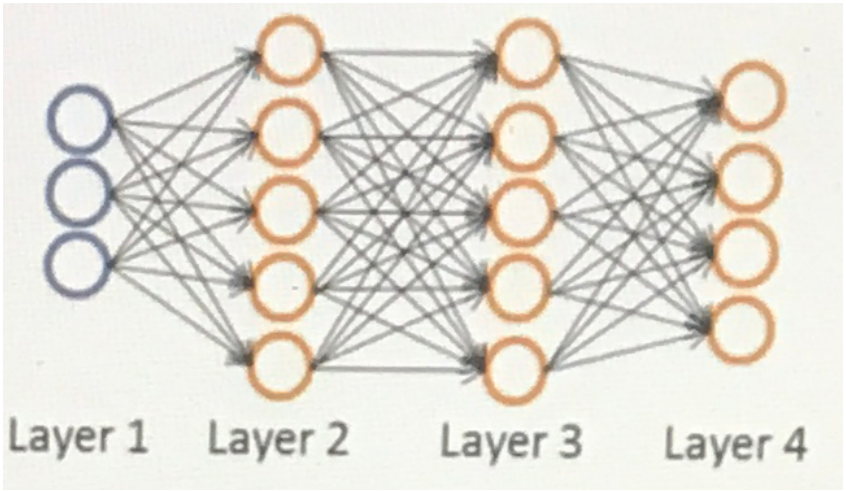


$h_\theta(x) \in \mathbb{R}^4$

$h_\theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $h_\theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ...

Pedestrian Car

Cost Function



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

L: total # of Layers

s_l: # of units (not counting bias unit) in Layer l

Logistic Regression:

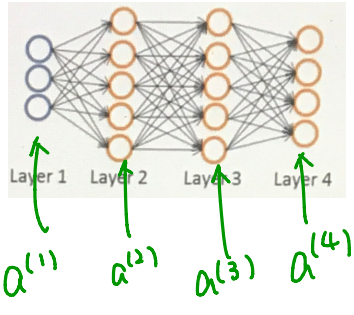
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural Network: $h_\theta(x) \in \mathbb{R}^K$ $(h_\theta(x))_i = i^{th}$ output

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} (\theta_{ji}^{(l)})^2$$

Backpropagation

Need: ① $J(\theta)$ ② : $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$



Given one training example: (x, y)

Forward Propagation:

$a^{(1)} = x$

$z^{(2)} = \theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ add $a_0^{(2)}$

$z^{(3)} = \theta^{(2)} a^{(2)}$

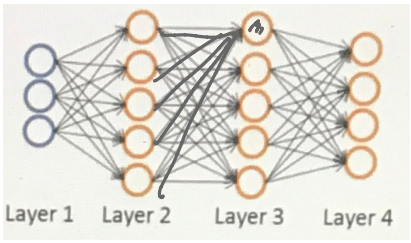
$a^{(3)} = g(z^{(3)})$ add $a_0^{(3)}$

$z^{(4)} = \theta^{(3)} a^{(3)}$

$a^{(4)} = g(z^{(4)}) = h_\theta(x)$

Back propagation:

Intuition: $\delta_j^{(l)}$ = "error" of node j in Layer l



$\delta_j^{(4)} = a_j^{(4)} - y_j$ vectorize: $\delta^{(4)} = a^{(4)} - y$

Proof:

$$\frac{\partial J}{\partial \theta_{ij}^{(l)}} = \frac{\partial J}{\partial a_i^{(l+1)}} \cdot \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \theta_{ij}^{(l)}}$$

$$\frac{\partial J}{\partial a_i^{(l+1)}} = \frac{\partial J}{\partial z_1^{(l+2)}} \cdot \frac{\partial z_1^{(l+2)}}{\partial a_i^{(l+1)}} + \frac{\partial J}{\partial z_2^{(l+2)}} \cdot \frac{\partial z_2^{(l+2)}}{\partial a_i^{(l+1)}} + \cdots + \frac{\partial J}{\partial z_{S_{l+2}}^{(l+2)}} \cdot \frac{\partial z_{S_{l+2}}^{(l+2)}}{\partial a_i^{(l+1)}}$$

$$= \sum_{j=1}^{S_{l+2}} \frac{\partial J}{\partial z_j^{(l+2)}} \cdot \frac{\partial z_j^{(l+2)}}{\partial a_i^{(l+1)}}$$

$$= \sum_{j=1}^{S_{l+2}} \delta_j^{(l+2)} \cdot \frac{\partial z_j^{(l+2)}}{\partial a_i^{(l+1)}}$$

$$= \sum_{j=1}^{S_{l+2}} \delta_j^{(l+2)} \cdot \frac{\partial}{\partial a_i^{(l+1)}} \left[\theta_{j0}^{(l+1)} a_0^{(l+1)} + \theta_{j1}^{(l+1)} a_1^{(l+1)} + \cdots + \theta_{j,S_{l+1}}^{(l+1)} a_{S_{l+1}}^{(l+1)} \right]$$

$$= \sum_{j=1}^{S_{l+2}} \delta_j^{(l+2)} \cdot \theta_{ji}^{(l+1)}$$

$$\frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} = \frac{\partial}{\partial z_i^{(l+1)}} \cdot g(z_i^{(l+1)})$$

$$= g(z_i^{(l+1)}) \cdot (1 - g(z_i^{(l+1)}))$$

$$= a_i^{(l+1)} \cdot (1 - a_i^{(l+1)})$$

$$\frac{\partial z_i^{(l+1)}}{\partial \theta_{ij}^{(l)}} = \frac{\partial}{\partial \theta_{ij}^{(l)}} \cdot \left[\theta_{i0}^{(l)} a_0^{(l)} + \theta_{i1}^{(l)} a_1^{(l)} + \cdots + \theta_{i,S_l}^{(l)} \cdot a_{S_l}^{(l)} \right]$$

$$= a_j^{(l)}$$

$$\Rightarrow \frac{\partial J}{\partial \theta_{ij}^{(l)}} = \sum_{k=1}^{S_{l+2}} \delta_k^{(l+2)} \theta_{ki}^{(l+1)} \cdot a_i^{(l+1)} (1 - a_i^{(l+1)}) \cdot a_j^{(l)}$$

$$\rightarrow \delta_i^{(l+1)}$$

def: $\delta_i^{(l)} = \frac{\partial}{\partial z_i^{(l)}} J(\theta)$

$$\Rightarrow \delta_i^{(l+1)} = \sum_{k=1}^{S_{l+2}} \delta_k^{(l+2)} \theta_{ki}^{(l+1)} \cdot a^{(l+1)} (1 - a_i^{(l+1)})$$

Vector: $\delta^{(l)} = [\theta^{(l)}]^T \cdot \delta^{(l+1)} .* \underbrace{a^{(l)} .* (1 - a^{(l)})}_{\rightarrow g'(z^{(l)})}$

$$\Rightarrow \frac{\partial J}{\partial \theta_{ij}^{(l)}} = \delta^{(l+1)} \cdot [a^{(l)}]^T$$

Additional: $J(\theta) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta(x))$

$$= -y \log(a^{(l)}) - (1-y) \log(1 - a^{(l)})$$

$$\Rightarrow \delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}}$$

$$= \frac{\partial}{\partial a_i^{(l)}} \left[-y_i \log(a_i^{(l)}) - (1-y_i) \log(1 - a_i^{(l)}) \right] \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}}$$

$$= a_i^{(l)} - y_i$$

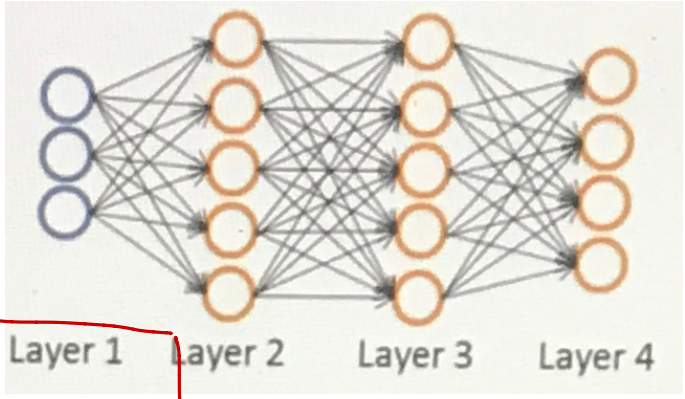
$$\Rightarrow \delta^{(L)} = a^{(l)} - y$$

Continue with the lecture.

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = [\theta^{(3)}]^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = [\theta^{(2)}]^T \delta^{(3)} .* g'(z^{(2)})$$



$$\delta^{(l)} = [\theta^{(l)}]^T \delta^{(l+1)} .* g'(z^{(l)})$$
$$= [\theta^{(l)}]^T \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)})$$

Back Propagation Algorithm:

Training Set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$

For $t = 1$ to m.

- Set $a^{(1)} = x^{(t)}$
- Compute $a^{(l)}$ by using forward propagation. $l = 2, 3, \cdots L$
- $\Rightarrow \delta^{(L)} = a^{(L)} - y$
- Compute $\delta^{(l)} = [\theta^{(l)}]^T \delta^{(l+1)} .* g'(z^{(l)})$, $l = 2, 3, \cdots L-1$

$\Rightarrow$ - $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$, $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} [a^{(l)}]^T$

Then update new Δ matrix

$$D_{ij}^{(l)} = \frac{1}{m} \left(\Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)} \right), \text{ if } j \neq 0$$

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}, \text{ if } j = 0$$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$$

Training a neural network

1. Random initialize weights
2. forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$
3. compute $J(\theta)$
4. back propagation to compute $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$
5. gradient checking to compare $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$
6. minimize $J(\theta)$

Evaluating a Hypothesis

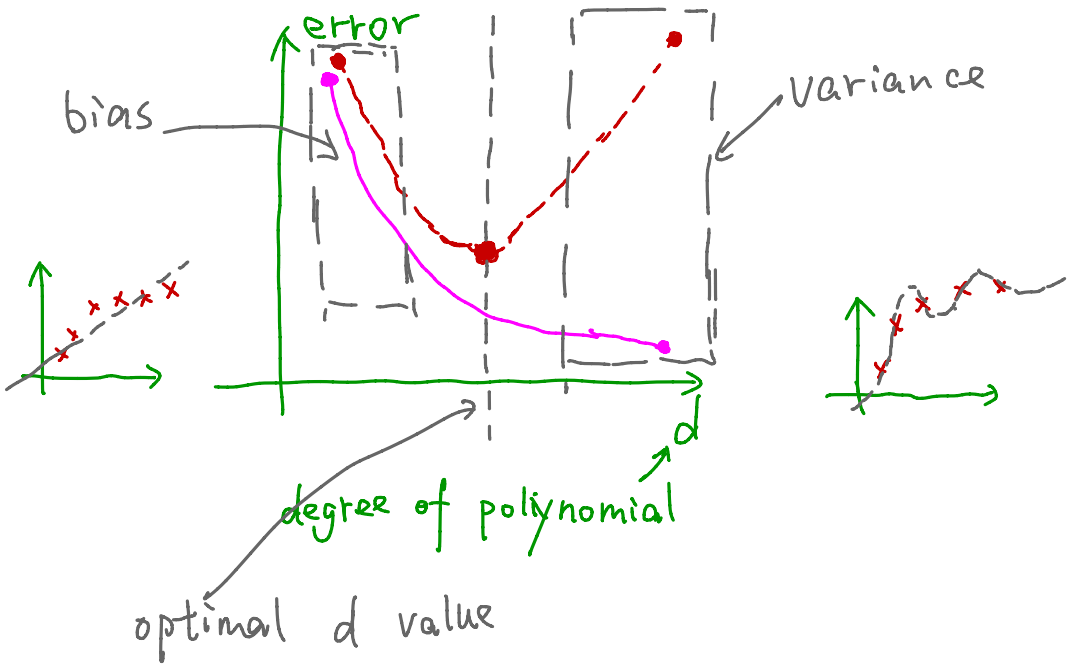
Data Set
- Training Set
- Cross Validation Set
- Test Set

Bias & Variance

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

Cross Validition Error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$

$\Rightarrow$



Bias (underfit): $J_{train}(\theta)$ high, $J_{cv}(\theta)$ high — $J_{cv} \approx J_{train}$

Variance (overfit): $J_{train}(\theta)$ low, $J_{cv}(\theta)$ high — $J_{cv} >> J_{train}$

Regularization (prevent overfitting)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

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

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)}\right)^2$$

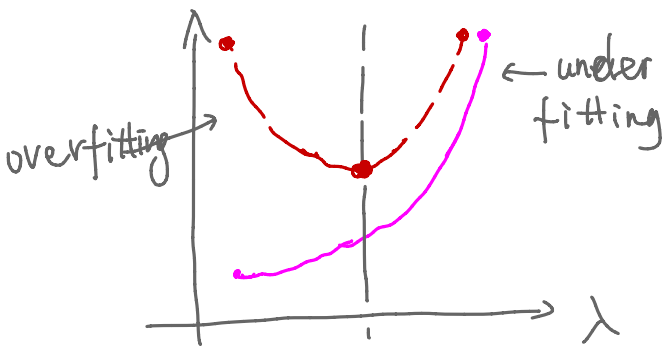
} no regularization

- Then how to choose regularization parameter λ.

1. Try $\lambda = 0 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01 \rightarrow \quad \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$

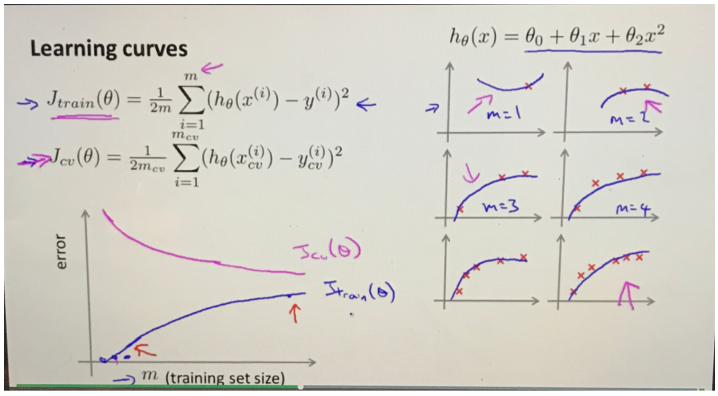
$\theta^{(5)} \rightarrow J_{cv}(\theta^{(5)})$ (minimum)

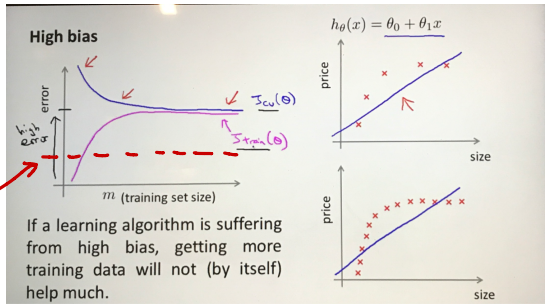
12. Try $\lambda = 10 \rightarrow \quad \rightarrow \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$



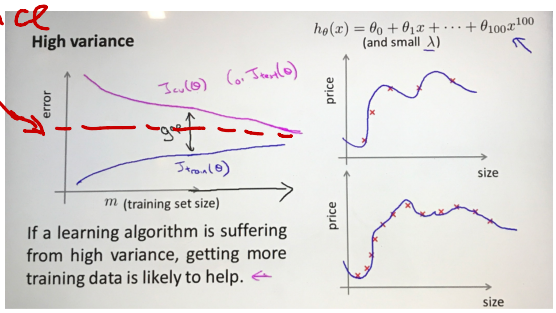
Bias/Variance as a function of the regularization parameter λ

Learning Curve





desired performance



Spam Classifier

x: features of email $y = \begin{cases} 1 & \text{spam} \\ 0 & \text{non-spam} \end{cases}$

- Collect lots of data ("honeypot" project)
- Develop sophisticated features
- Develop algorithms to process your inputs in different ways

Error Metrics for Skewed Classes

Precision / Recall

e.g. : cancer

$y=1$ in presence of rare class that we want to detect

Actual Class

predicted class	1	0
1	True pos	False pos
0	False neg	True neg

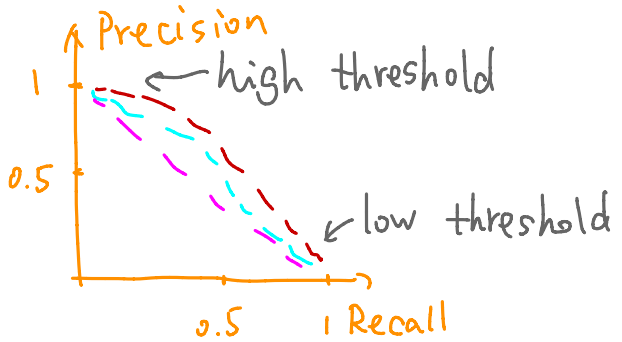
Precision:

$$\frac{\text{True Pos}}{\#\text{ predicted Pos}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False Pos}}$$

Recall:

$$\frac{\text{True Pos}}{\#\text{ actual Pos}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False neg}}$$

Trading off precison & Recall

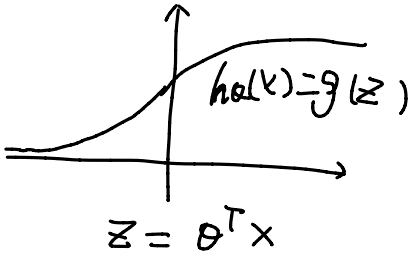


F1 Score

$$F_1 = 2\frac{PR}{P+R}$$

Support Vector Machines

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



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

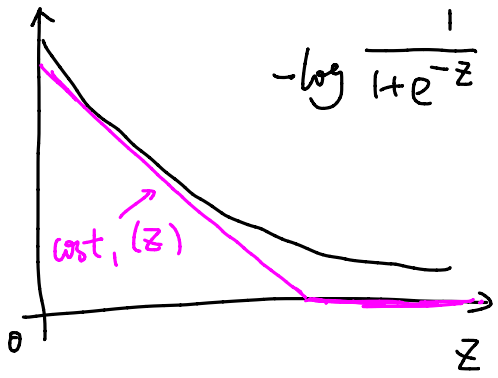
If $y=0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

Logistic Regression

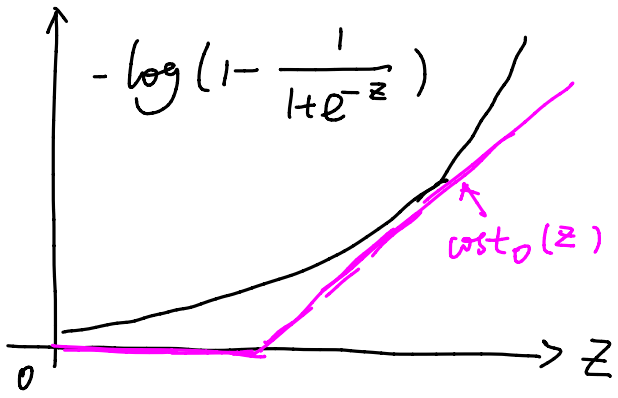
$$\text{Cost} = -\left[y \log h_\theta(x) + (1-y) \log (1 - h_\theta(x))\right]$$

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

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



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



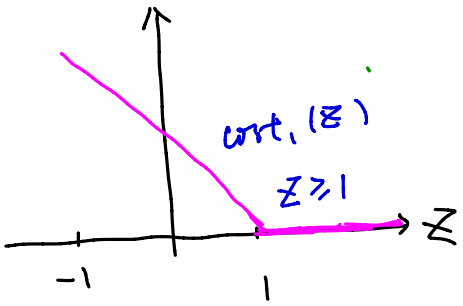
⇒ For SVM:

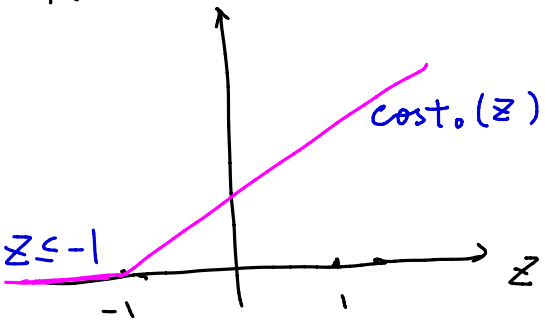
$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

Hypothesis:

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

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





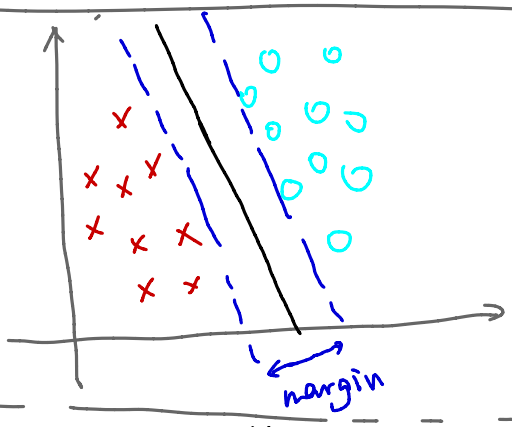
If $y=1$, we want $\theta^T x \geq 1$

If $y=0$, we want $\theta^T x \leq -1$

SVM: Large margin classifier

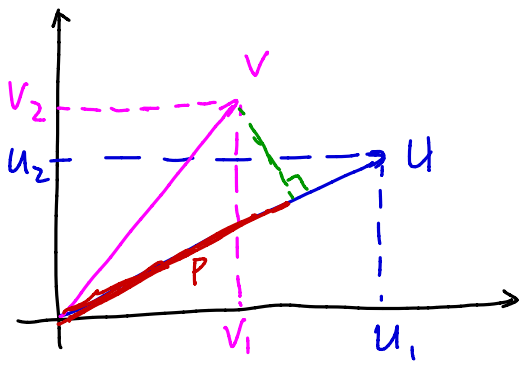
suppose C very large.

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



Math of SVM

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



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

$= u_1 v_1 + u_2 v_2$

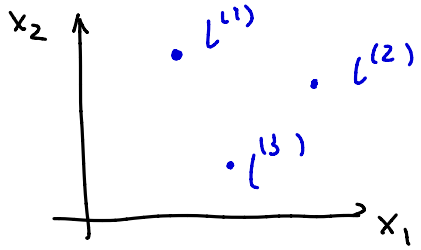
p: length of projection of v to u

$\|u\|$: length of vector u

$\|u\| = \sqrt{u_1^2 + u_2^2}$

kernels

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

SVM with kernels:

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

choose: $l^{(1)} = x^{(1)}, \cdots l^{(m)} = x^{(m)}$

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

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

$\vdots$

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

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

$x^{(i)} \rightarrow$

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

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

$\vdots$

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

SVM with kernels:

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

Predict $y=1$, if $\theta^T f \geq 0 \rightarrow \theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m \geq 0$

Training:

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

parametes:

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

- Large C: (small λ) Lower bias, high variance (overfitting)
- small C: (large λ) High bias, low variance (underfitting)

σ^2: Large σ^2: features f_i vary more smoothly

High bias, low variance

small σ^2: f_i vary less smoothly

Low bias, high variance.

Use SVM software packge

Need to specify:

→ parameter C

→ kernel (similarity function).

eg. ① No kernel (linear kernel)

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

② Gaussian kernel

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

need to choose σ^2.

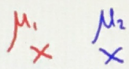
k-Means

Input: - k (# of clusters)

- Training set $\{x^{(1)}, x^{(2)} \dots x^{(m)}\}$

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

K-means algorithm



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

Repeat {

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

}

First for each of

---

k-means optimization objective.

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

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

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

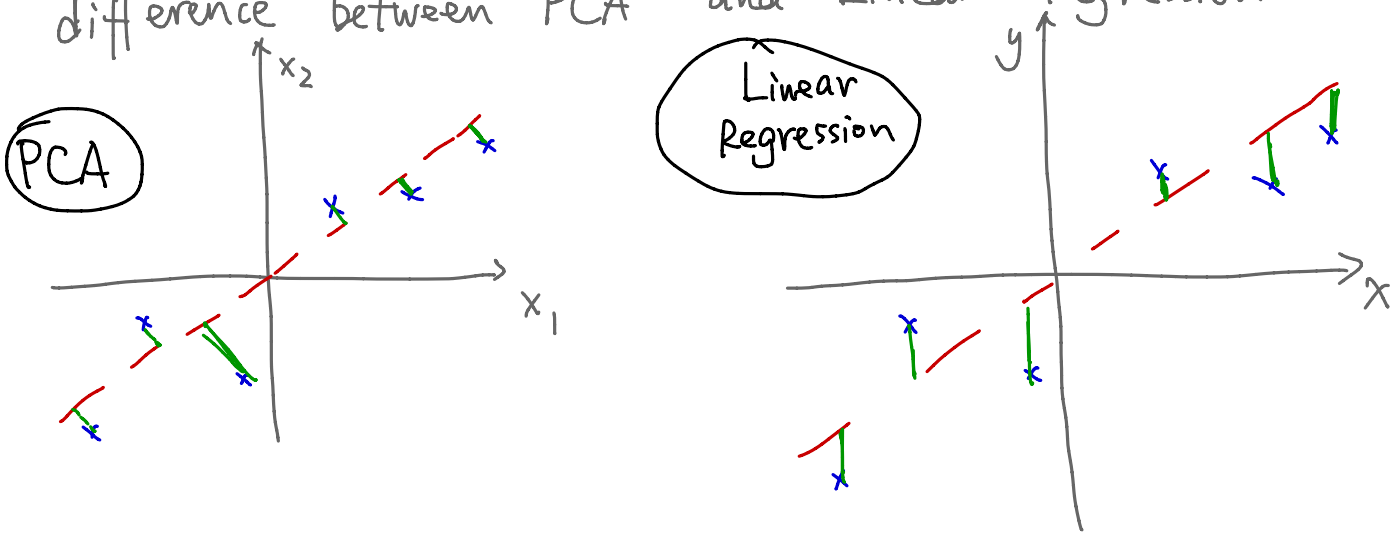
$$\Rightarrow J(c^{(1)} \dots c^{(m)}, \mu_1 \dots \mu_k) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$$

Dimensionality Reduction

1. Data compression
2. Data visualization

Principal Component Analysis (PCA)

difference between PCA and Linear Regression



PCA Algorithm:

Reduce data from n-dimensions to k-dimensions

- Compute "covariance matrix"

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

- Compute "eigenvectors" of matrix Σ

```
[U, S, V] = svd(sigma);   [eig(sigma)]
```

```
Ureduce = U(:, 1:k);
z = Ureduce' * x
```

Reconstruction

PCA: $z = U_{reduce}^T x$

$\Rightarrow x_{approx}^{(i)} = U_{reduce} \cdot z^{(i)}$

PCA: n-dimension $\rightarrow$ k-dimension

k is called the number of principal component

Q: How to choose k?

Choosing k (number of principal components)

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

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

Typically, choose k to be 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"

Choosing k (number of principal components)

Algorithm:

Try PCA with $k=1$, $k=2$, $k=3$, $k=4$, ...

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$

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

$k = 17$

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

$S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & S_{33} & \\ & & & \ddots \\ & & & & S_{nn} \end{bmatrix}$

For given k

$k=3$

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

$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$

Andrew N

summary: [U, S, V] = svd(sigma);

Pick smallest value of k for which:

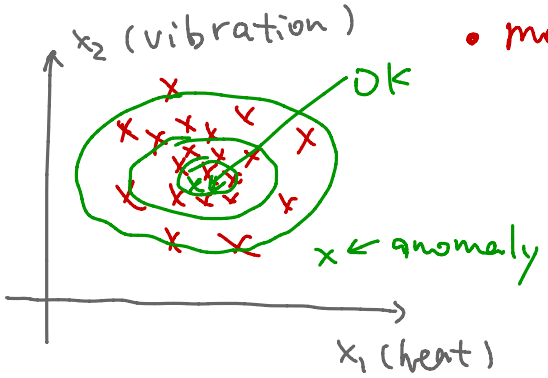
$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99.$$

Anomaly Detection

Density estimation:

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

Is $x^{(test)}$ anomalous?



- model $p(x)$

$p(x^{(test)}) < \varepsilon \rightarrow$ anomaly

$p(x^{(test)}) \geq \varepsilon \rightarrow$ ok

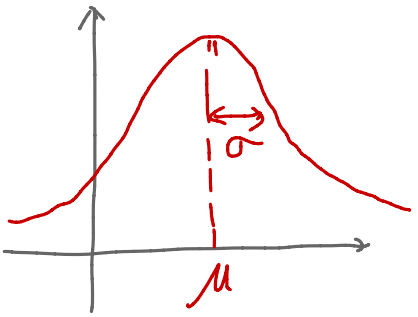
Examples:
- Fraud detection
- Manufacturing
- Monitoring computers in a data center

Gaussian Distribution (Normal Distribution)

$x \in \mathbb{R}$. mean: μ. variance: σ^2

Standard deviation: σ

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



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

Dataset $\{x^{(1)} \dots x^{(m)}\}$ $\quad x \in \mathbb{R}$

$$\Rightarrow \quad \mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

Density estimation

Training Set : $\{ x^{(1)} \cdots x^{(m)} \}$

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

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

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

$\vdots$

$$P(x) = P(x_1; \mu_1, \sigma_1^2) P(x_2; \mu_2, \sigma_2^2) \cdots P(x_n; \mu_n, \sigma_n^2)$$

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

Anomaly detection algorithm

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

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

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

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

- Very small number of positive examples ($y = 1$). 0~20 commonly
- Large number of negative examples
- Many different "types" of anomaly. Hard for any algorithm to learn from positive examples what the anomalies look like

Supervised Learning

- Large number 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 ones in the training set.

Multivariate Gaussian Distribution

$x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2)$ ...

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

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

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

Anomaly Detection with multiple Gaussian Distribution

Training Set: $\{x^{(1)} \cdots x^{(m)}\}$

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

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

anomaly if $p(x) < \varepsilon$

The original model:

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

corresponds to multivariable Gaussian model:

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

where: $\Sigma = \begin{bmatrix} \sigma_1^2 & & \\ & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_n^2 \end{bmatrix}$

→ Original model	vs.	→ Multivariate Gaussian
$p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$		$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} \lvert\Sigma\rvert^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$
Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values. → $x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$		→ Automatically captures correlations between features
→ Computationally cheaper (alternatively, scales better to large n) n=10,000, n=100,000		$\Sigma \in \mathbb{R}^{n \times n}$ Σ^{-1} Computationally more expensive → Σ ↑ $n^2/2$ → $x_1 = x_2$, $x_3 = x_4 + x_5$
OK even if m (training set size) is small		Must have $m > n$ or else Σ is non-invertible. $m \geq 10n$