Machine Learning

Examples:

- Database Mining:
- Applications that can't be programmed
- Self-customizing programs
- Understanding human learning.

Supervised Learning

- Data set with right answers
- Regression problem - continous data output
- Classification problem - discrete valued output

Unsupervised Learning

- Data has no labels or right answers
  e.g. (clustering algorithms)
  (Associative)

Linear Regression with one Variable

- Also known as "univariate linear regression"

Hypothesis function:

- $h_\theta(x) = \theta_0 + \theta_1 x$

- Cost Function: accuracy of $h_\theta(x)$ related to data

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

$\frac{1}{2}$ the mean of the squares — $\bar{x}$: mean of the squares.

- Also known as: Squared error function
- Mean squered error.

- Gradient Descent
  - We take the derivative of our cost funtion, it will give us the direction to move towards.

    α: learning rate.
  - We make steps down by α

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

- For a Linear Regression

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right]$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[(h_\theta(x^{(i)}) - y^{(i)}) x^{(i)} \right]$$

- if uses all the data in the sum, it is also called "Batch" Gradient Descent.

SON MENTIRAS!!
TODO!!!

Multiple Features

- n: number of features
- $x^{(i)}$: input (features) of i^{th} training example. Vector of dimension n.
- $x_j^{(i)}$: value of feature j in i^{th} training example.
- Now linear Regression looks like

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$$

- For convenience we define $x_0^{(i)} = 1$

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \qquad \boxed{h_\theta(x) = \theta^T x}$$

- Cost Function

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

- Gradient Descent

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

- Feature scaling
  - Make sure features are on a similar scale
  - Devide given features by maximum value. $x^{(i)} \equiv \frac{x^{(i)}}{\text{abs}(\max(x_j^{(i)}))}$
- Mean normalization

$$x_i \to \frac{x_i - \mu_i}{x_{max} - x_{min} \text{ range}}$$

- How to choose learning rate α

Problem is solve by choosing smaller α.

Try α by 1 diff. of magnitudes:
1, .1, 0.01, 0.001

- Polynomial fitting

Σ NAAH.... ASÍ NO ES

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

take x, x^2, x^3 as features, then

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots$

③

- Exact Minimization of Cost Function:

X is the data with an added column of ones: $X = \begin{pmatrix} 1 & \text{data} \\ \vdots & \end{pmatrix}$ $y = \begin{pmatrix} \\ \\ \end{pmatrix}$

x_0	Size	# bedrooms	Price
1	—	—	—
1	—	—	—
⋮			

$\theta = (X^T X)^{-1} X^T y$ this are the values of θ that minimize $J(\theta)$.

$X = \begin{pmatrix} x^{(1)T} \\ x^{(i)T} \\ x^{(m)T} \end{pmatrix}$

- Gradient Descent vs Normal Equation

For n big # of features normal equation scales as $O(n^3)$ $n \geq 10{,}000$ somewhat the treshhold for implementing gradient descent.

- Normal Equation:

what if $X^T X$ is non-invertible?

- This can be because of redundant features. (linerly dependent)

eg. X_1 = size in feet

X_2 = size in m^2

- Also if there are more features than data point. $m \leq n$

• Delete some features or use regularization.

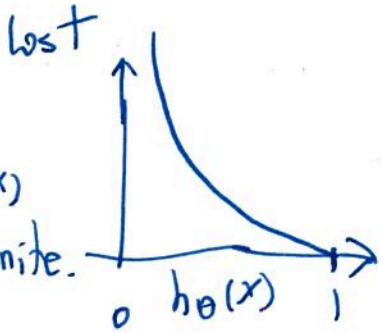
Classification

- $Y \in \{0, 1\}$ Two classes e.g. 0: Negative Class
  1: Positive Class
- Linear regression not a good idea.

$h_\theta(x)$ can be > 1 and < 0 !!

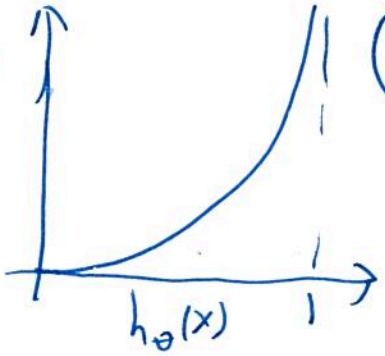
- Logistic Regression: $0 \leq h_\theta(x) \leq 1$

if $y=1$



and $h_\theta(x)$ then cost is infinite.

if $y=0$



- We write $cost(h_\theta(x), y)$ as

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

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

- We want to minimize $J(\theta)$

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

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

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

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

- Optimization Algorithms
  - Gradient Descent
  - Conjugate Gradient
  - BFGS
  - L-BFGS

} Pick α for you and converges faster.

- Function is fminunc(@function, initialguessTheta, options);

[jval, gradient] = function(theta)

function [jval, gradient] = function(theta)

jval = $J(\theta)$

gradient(1) = $\frac{\partial}{\partial \theta_0} J(\theta)$

⋮

gradient(n+1) = $\frac{\partial}{\partial \theta_n} J(\theta)$

Logistic Regression Model

$$0 \le h_\theta(x) \le 1$$

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

↑ Logistic function - sigmoid function

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

Interpretation: $h_\theta(x) = 0.7 \Rightarrow$ 70% chance

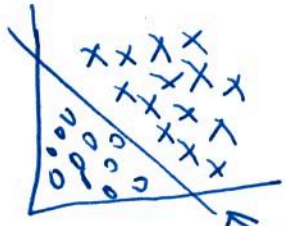
$h_\theta(x) = P(y=1 \mid x; \theta)$

$g(z) \ge 0.5$ when $z \ge 0$

$h_\theta(x) \ge 0.5$ when $\theta^T x \ge 0$

- Decision Boundary: $h_\theta(x) = 0.5$

$\theta^T x = 0$



- Non-linear decision boundaries

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

Adding polynomial terms can change shape of boundary.

- Higher order polynomial terms: can have more distribution boundaries.

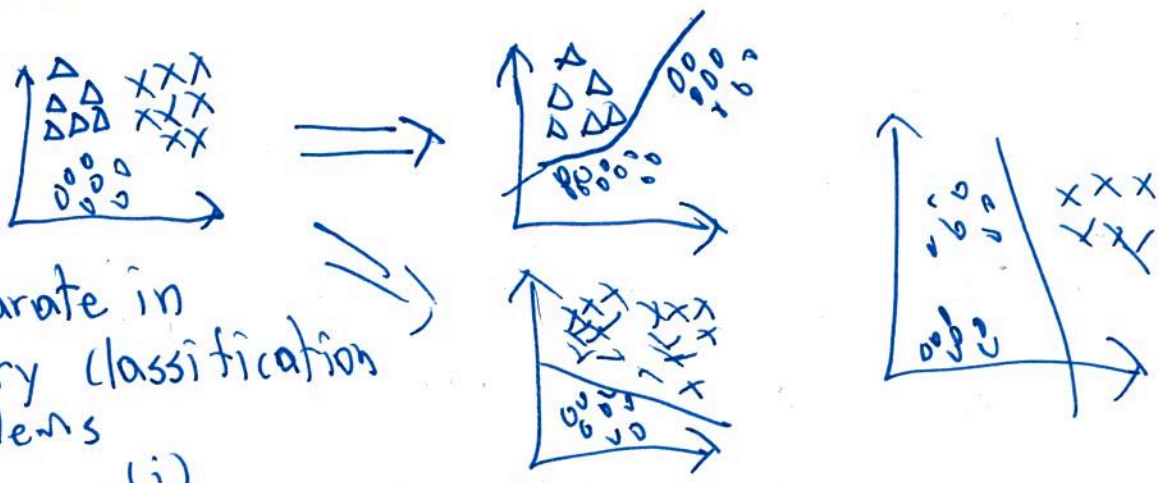
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \dots)$$

- cost function

$$J(\theta) = \frac{1}{m} \sum_{i=m} \text{cost}(h_\theta(x), y)$$

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

- Multi-class classification: One vs All



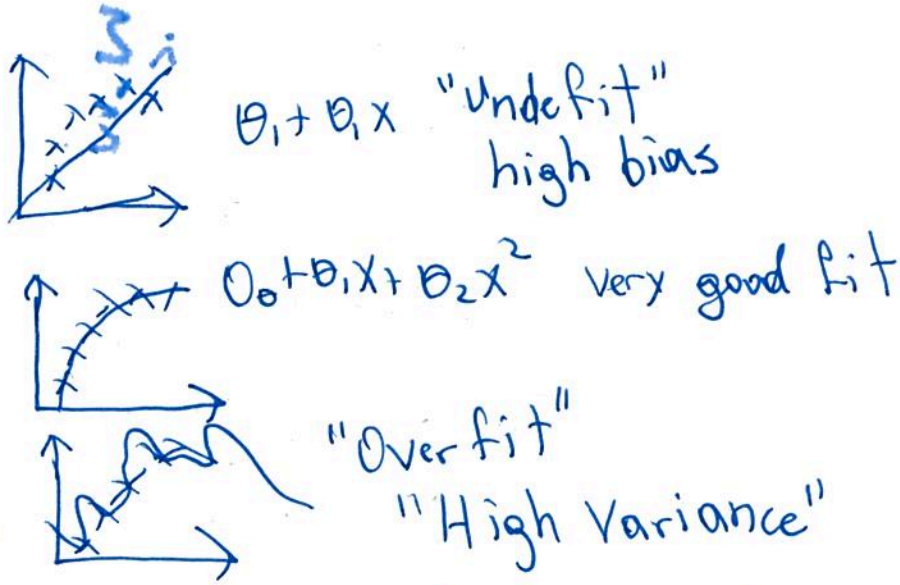
Separate in Binary classification problems

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

$\max_i h_\theta^{(i)}(x)$ To predict we choose the highest classification output.

- Overfitting

Linear Regression



- It can also happen for logistic Regression.
- Addressing Overfitting:
  - Reduce # of features
    - Manually select which features to keep.
    - Model selection algorithm
  - Regularization
    - Reduce magnitude/values of parameters θ_j.
    - Works well for a lot of features

- We penalize high order terms
  - "simpler" hypothesis
  - Less prone to overfitting

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

λ: regularization parameter

if λ is big "underfiting"

if λ is small "overfiting"

Regularized Linear Regresion

- Gradient Descent

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum[h_\theta(x^{(i)}) - y^{(i)}]x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum[h_\theta(x^{(i)}) - y^{(i)}]x_j^{(i)} + \frac{\lambda\alpha}{m}\theta_j$$

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

- Normal Equation

$$\theta = (X^TX + \lambda\begin{bmatrix}0 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1\end{bmatrix})^{-1}X^Ty$$

- Non-Invertible

if $m \leq n$ X^TX is singular

if $\lambda > 0$ the matrix $X^TX + \lambda[\ ]$ is invertible.

Regularized Logistic Regression

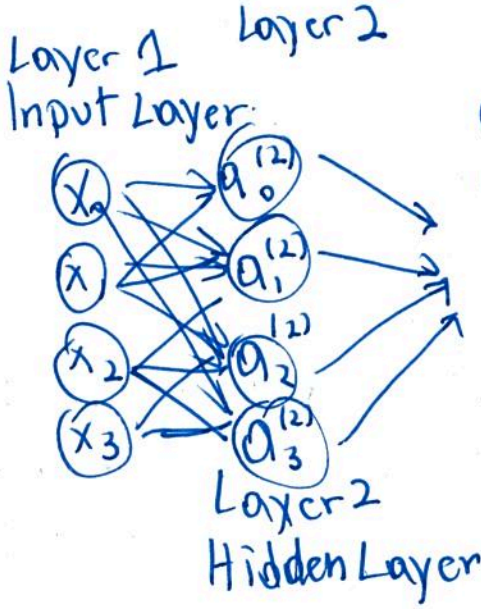
$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^{(i)}) - y^{(i)}]x_0^{(i)}$$

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

Neural Networks

- Non-linear hypotheses.
  - With n large just too many features
  e.g. Pixel examples, recognize object from image.

Neural network



$a_i^{(j)}$ = activation of unit i layer 2

$\Theta^{(j)}$ = matrix weights controlling function mapping from layer j to layer $j+1$.

3 input units
3 hidden units
then $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

- if network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)} \in \mathbb{R}^{s_{j+1}\times(s_j+1)}$

$$a_1^{(2)} = g\left[\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right]$$

$$h_\Theta(x) = a_1^{(3)} = g\left[\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right]$$

Forward Propagation: Vectorized Implementation

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

$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_3 \end{pmatrix}$ $\quad z^{(2)} = \begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{pmatrix}$ then $a^{(2)} = g(z^{(2)})$

$$z^{(2)} = \Theta^{(1)} x$$

Use $a^{(i)}$ to define activations then $z^{(2)} = \Theta^{(1)} a^{(1)}$

Extra bias unit $a_0^{(2)} \rightarrow a^{(2)} \in \mathbb{R}^4$

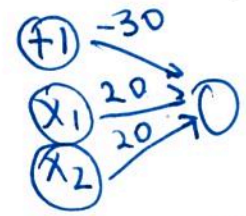
Add $a_0^{(2)} = 1$ $\quad z^{(3)} = \Theta^{(2)} a^{(2)}$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

- same as logistic regression but using updated a_i features, the a are the learned features from the initial features x_i.

Example:

AND $x_1, x_2 \in \{0, 1\}$ $\quad y = x_1$ AND x_2



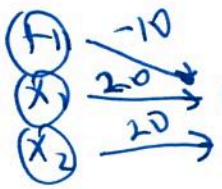
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

$\Theta_{10}^{(1)} = -30$, $\Theta_{11}^{(1)} = 20$ $\quad \Theta_{12}^{(1)} = 20$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

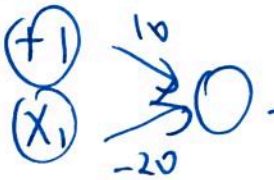
$h_\Theta(x) \approx x_1$ AND x_2

OR function



x_1	x_2	$h_\Theta(x)$
0	0	0
0	1	1
1	0	1
1	1	1

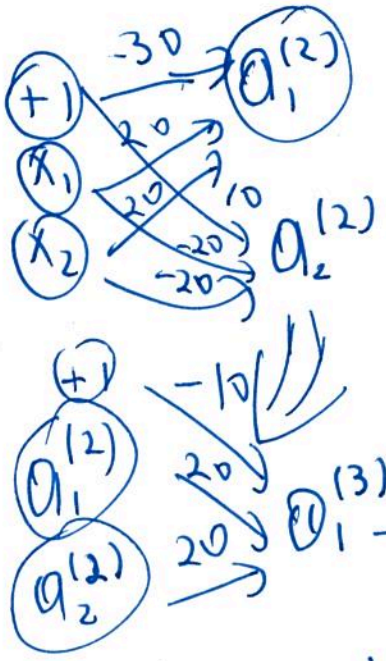
NOT



x	$h_\Theta(x) = g(10 - 20x_1)$
0	$g(10) \approx 1$
1	$g(-10) = 0$

XNOR

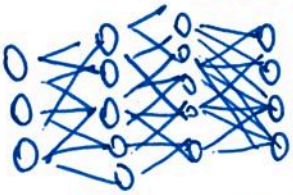
x_1	x_2	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Multi-Class classification

Example: Recognize a number 10 classes, each number a class.



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

4 Classes

$h_\Theta(x) \approx \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ first class

$h_\Theta(x) \approx \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ second class

Multiple output units: One vs all

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

$y^{(i)}$ one of $\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$

class 1, class 2, class 3, class 4

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

$x^{(i)}$: Image object

$y^{(i)}$: Vector representing a class.

$h_\Theta(x^{(i)}) \approx y^{(i)}$

Neural Network Classification

- Binary classification:

  $y = 0$ or 1

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

  $s_L = 1$ ← no. of layers

  $K = 1$ ← # of output units

Training set

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

- Multi-class classification (K classes)

  $y \in \mathbb{R}^K$ e.g. $\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}$

  K output units $h_\theta(x) \in \mathbb{R}^K$ $s_L = K$ $K \geq 3$

- Cost Function: generalization of logistic regression.

Logistic regression ⇓

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

Neural Network ⇓

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

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

Gradient Computation

- We want to minimize $J(\theta)$
- We need to compute
  - $J(\theta)$
  - $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

Backpropagation algorithm

$\delta_j^{(l)}$: "error" of node j in layer l.

activation $a_j^{(l)}$ e.g. $\delta_j^{(4)} = a_j^{(4)} - y_j$ $\quad (h_\theta(x))_j$

Vectorization: $\delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ $\quad g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})$

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

* No $\delta^{(1)}$ *

$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}$ (ignoring λ; if $\lambda = 0$)

- For many training sets (m)

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

  for $i = 1$ to m

  Set $a^{(1)} = x^{(i)}$

  Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$.

  Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

  Then back propagation $\delta^{(L-1)}, \delta^{(L-2)}, \ldots \delta^{(2)}$

  $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ $\quad \Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

  end

  $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}$ $\quad j \neq 0$

  $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ $\quad j = 0$

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

Formally $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i) \quad j \geq 0$

$cost(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log h_\theta(x^{(i)})$

Advanced Optimization

function [jval, gradient] = costFunction(theta) ← need $\mathbb{R}^{n+1}$

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

optTheta = fminunc(@costFunction, initialTheta, options)

Neural Network (L=4)

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ matrices

$D^{(1)}, D^{(2)}, D^{(3)}$ matrices

"Unroll into vectors"

thetaVec = [Theta1(:); Theta2(:); Theta3(:)];

DVec = [D1(:); D2(:); D3(:)];

Algorithm

- Initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
- Unroll to get initialTheta
- Use fminunc(@costfunction, initialTheta, options)
- function [jval, gradient] = costfunction(thetaVec)
  - from thetaVec reshape to get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
  - Use forward propagation/back propagation to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$
  - Unroll $D^{(i)}$ to get gradientVec.

- Gradient Checking

$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon} \quad \epsilon = 10^{-4}$

$\theta \in \mathbb{R}^n$ unrolled version of θ

$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{J(\theta_1, \ldots, \theta_i+\epsilon, \ldots) - J(\theta_1 \cdots, \theta_i-\epsilon)}{2\epsilon}$

for $i = 1:n$

$\theta_+ = \theta$

$\theta_+(i) = \theta_+ + \varepsilon;$

$\theta_- = \theta$

$\theta_- = \theta_- - \varepsilon;$

$grad(i) = \frac{J(\theta_+) - J(\theta_-)}{2\varepsilon}$

end

Check that grad ≈ Dvec (from back propagation)

*Once you check gradient turn off!! when training

- Initial value of θ:
  - if initial θ are zeros then don't work as ~~you can't calculate δ_1~~, Even after one gradient decent update. $\theta_{01}^{(1)} = \theta_{02}^{(1)}$

$a_1^{(2)} = a_2^{(2)} \quad \delta_1^{(2)} = \delta_2^{(2)}$

$\frac{\partial}{\partial \theta_{01}^{(1)}} J(\theta) = \frac{\partial}{\partial \theta_{02}^{(1)}} J(\theta)$

The hidden units are Identical,

- Use random initialization $\Theta_{ij}^{(l)} \in [-\varepsilon, \varepsilon]$

  Theta1 = rand(10,11) * 2ε - ε

  Symmetry Breaking

- How to choose architecture: Default 1 hidden layer. If more hidden same no. of hidden units in every layer.

- $J(\theta)$ non-convex

Debugging a learning algorithm: (16)

- Suppose the solution makes errors large in its predictions.
  - Get more training examples
  - Try smaller sets of features (prevent overfitting)
  - Try getting additional features.
  - Try polynomial features (x_1^2, x_2^2, x_1, x_2, etc.)
  - Try decreasing λ
  - Try increasing λ

Machine learning Diagnostic:

test to gain insight what is/isn't working and gain guidance as to how to improve.

- Evaluate your hypothesis
  - Split Data set and use 70% as training set and 30% as test set.

    m_{test} = # of test examples.

  - Compute test error $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x^i) - y^i)^2$ Linear Regression
  - For Logistic Regression $J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log(h_\theta(x_{test}^{(i)})$ ---

  Another way is to check the 0|1 misclassification error:

$$err(h_\theta(x)) = \begin{cases} 1 \text{ if } h_\theta(x) \geq .5, y=0 \\ \text{ if } h_\theta(x) < .5, y=1 \\ 0 \text{ otherwise} \end{cases}$$

- Model Selection

  d: degree of polynomial

  - check performance of each polynomial $J_{test}(\theta)$ and choose the minimum.
  - Separate Dataset even more. Training set 60%
    Cross Validation set (CV) 20%
    Test set 20%

Diagnostic Bias vs Variance

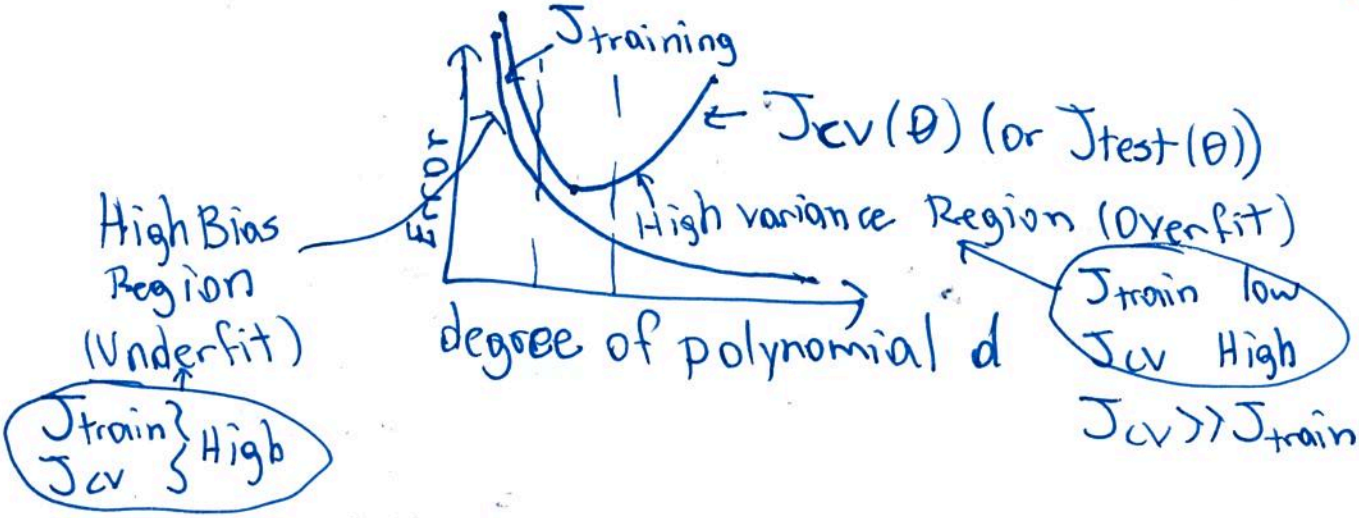
High Bias

Just Right

High Variance

Training Error: $J_{train}(\theta)$

Cross Validation Error: $J_{cv}(\theta)$ (or $J_{test}(\theta)$)



Regularization effect to High Bias and High Variance

- Large λ → High Bias (underfit)
- Small λ → High Variance (overfit)
- choosing λ : $J_{train}(\theta)$, $J_{cv}(\theta)$, $J_{test}(\theta)$ have no regularization parameter ($\lambda = 0$)
- Range of values for λ e.g. $[0, 10]$

$$\lambda_1 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^1 \rightarrow J_{cv}(\theta^1)$$
$$\lambda_2 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^2 \rightarrow J_{cv}(\theta^2)$$
$$\lambda_3 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^3 \rightarrow J_{cv}(\theta^3)$$

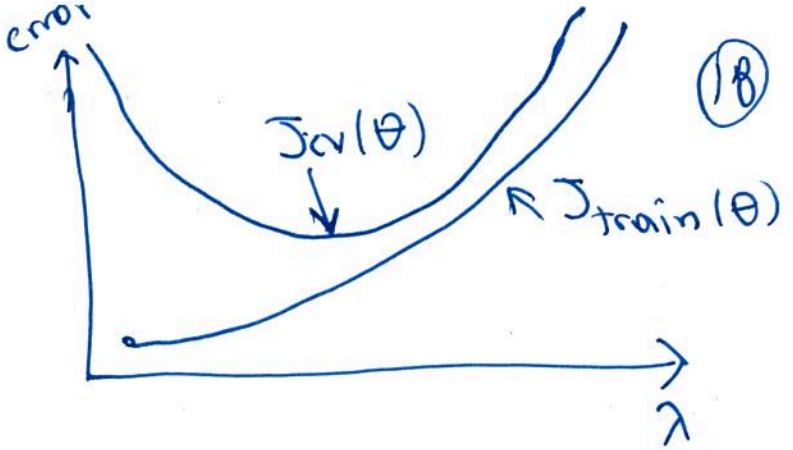
choose λ_i that gives lowest $J_{cv}(\theta^i)$

then. Test error: $J_{test}(\theta^{(i)})$

$J(\theta)$ ← regularization ↑

$J_{train}(\theta) \rightarrow \lambda = 0$ to get

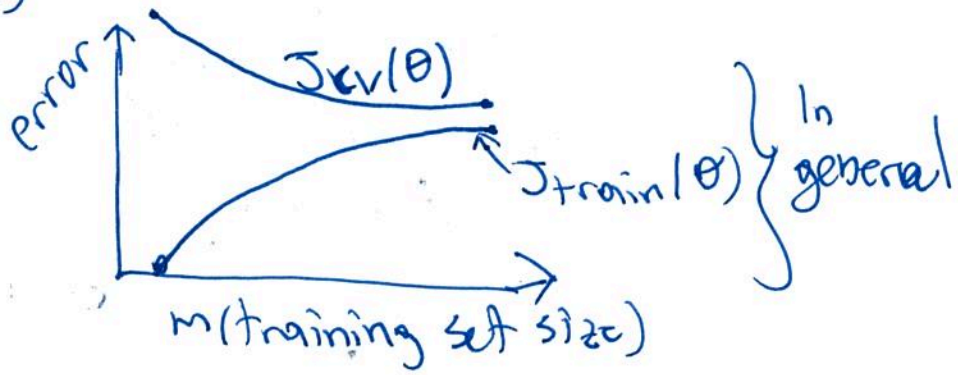
$J_{cv}(\theta) \rightarrow \lambda = 0$ to get



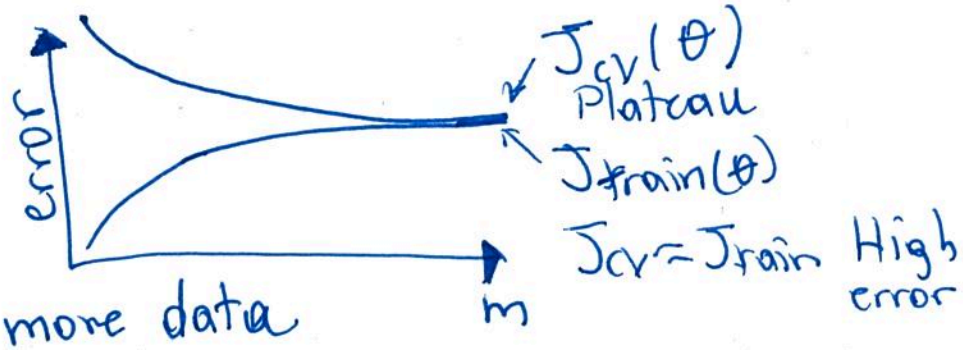
Learning Curves

J_{train}

J_{cv}

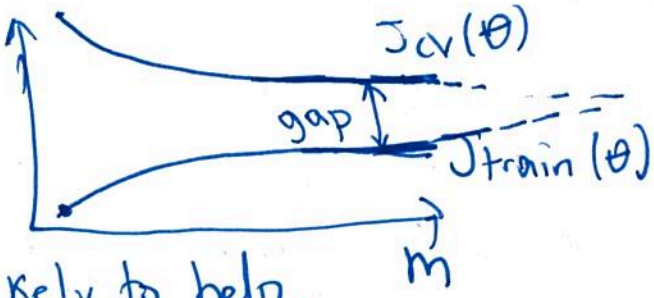


- High Bias



Not need more data

- High Variance



Getting more training data is likely to help.

- Get more training examples } fixes High Variance
- Try smaller set of features } fixes High Variance
- Try get additional features } fixes High Bias
- Try polynomial features
- Decreasin $\lambda \Rightarrow$ fixes High Bias
- Increasin $\lambda \Rightarrow$ fixes High Variance

Neural Networks and overfitting

- Small NN underfits
- Large NN overfits, (hange λ.

Example → Spam Classifier

Features are certain words e.g. buy, deal, discount, ..., now

Suppose email contains words w_1, w_2, w_4

$$x_j = \begin{cases} 1 & \text{if word } j \text{ appears} \\ 0 & \text{otherwise} \end{cases}$$

$$x = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

- What is the best way to spend time?
  - Collect lots of data
  - Develop sophisticated features (routing information)
  - Email message body and develop sophisticated features.
  - Develop sophisticated algorithm to detect misspellings.

Error Analysis

- Start with simple algorithm and test in cross validation data
- Plot learning Curves ← Error analysis
- Manually examine the examples (in cross validation set) that your algorithm made errors.

Error Analysis Example

$m_{cv} = 500$ cross validation set

Algorithm misclassifies 100 emails

Manually examine 100 errors and categorize them

i) what type of email is it? pharma, replicas, fishing

Pharma: 12

Replica: 4

fishing: 54 ⇒ Doing Poorly

Other: 31

ii) what cues (features) you think would have helped the algorithm classify them

Deliberate Misspelling: 5 → rare not worth the time

Unusual Routing: 16

Unusual Punctuation: 32 ⇒ strong indication for more features for punctuation.

ii) Numerical evaluation. single row number

e.g. Treat discount/discounts/discounted/... be treated as same word?

Can use "stemming" ~~prob~~ software

Negative: universe/university

- Look CV error for with and without stemming

e.g. Distinguish upper vs lower case (Mom/man)

Error Metrics for Skewed Classes

e.g. cancer classification

$h_\theta(x)$

Find 1% error but only 0.5% have cancer

function y = predict(cancer(x)) } this predicts better
return $y = 0$ } 0.5% error.

- when we have much more examples from one class than the other.

- Precision / Recall
  $y=1$ in presence of rare class.
  Compute:

Predicted Class \ Actual Class	1	0
1	True Positive	False Positive
0	False Negative	True Negative

  - Precision (of all patients where we predicted $y=1$ what fraction has cancer?.

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

  - Recall (of patients that have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True Positives}}{\#\text{ actual Positives}} = \frac{\text{True Positives}}{\text{True Positives-False Negative:}}$$

  - Then if algorithm predicts $y=0$ then recall $=0$ so not good.

Trading Off precision and recall

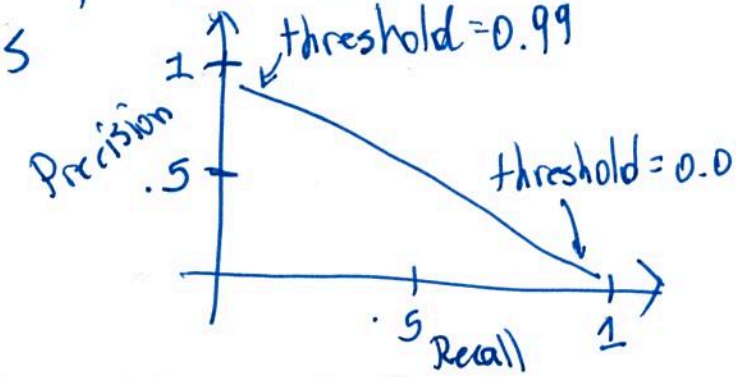
Logistic regression: $0 \le h_\theta(x) \le 1$

- Suppose we want to predict cancer only if very confident.
  - change treshold $h_\theta(x)$ $\begin{cases} 1 & h_\theta(x) \ge 0.7 \\ 0 & h_\theta(x) < 0.7 \end{cases}$
  - * Higher Precision, lower recall
- Avoid false negatives, missing too many cases of cancer.

$$h_\theta(x) \begin{cases} 1 & h_\theta(x) \ge .3 \\ 0 & h_\theta(x) < 0.3 \end{cases}$$

  - * Higher Recall, lower precision.

for most classifiers



- How to decide best threshold

	Precision	Recall
Alg 1	.5	0.4
Alg 2	.7	.1
Alg 3	.02	1

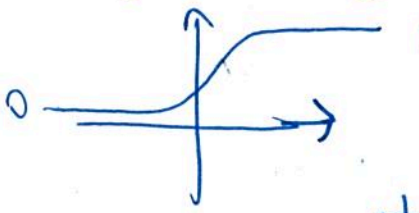
} Which is the best?

- Average {.45, .4, .51} *Not Good
- F_1 Score $\frac{2PR}{P+R}$ {0.444, 0.175, 0.0392}

(check in Cross Validation Set)

Support Vector Machine (SVM)

- Optimazation Objective
  - In logistic regression $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$



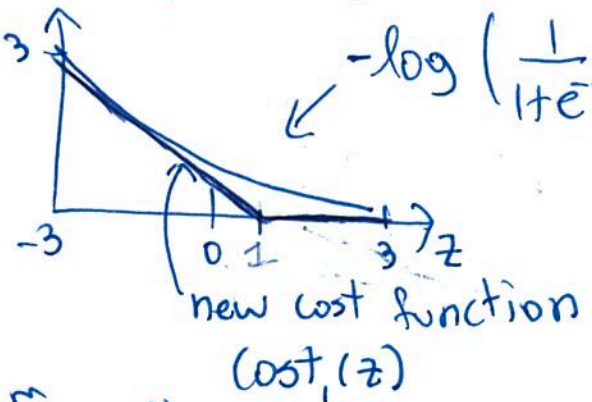
$z = \theta^T x$

if $y=1$, we want $h_\theta(x) \simeq 1$, $\theta^T x \gg 0$
if $y=0$, we want $h_\theta(x) \simeq 0$, $\theta^T x \ll 0$

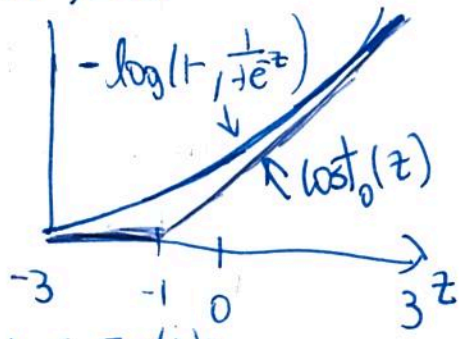
Cost of example (x, y)

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

if $y=1$



if $y=0$



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

By convention get rid of $1/m$

Normally $A + \lambda B$ (logistic regression) } same solution θ
now $CA + B$ for SVM } if $C = 1/\lambda$
↑weight

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

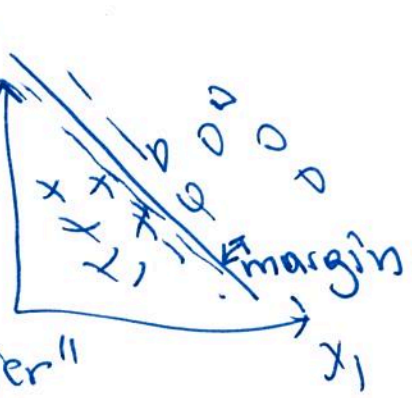
- Hypothesis:

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

if $y=1$ we want $\theta^T x \geq 1$ (not just ≥ 0)
if $y=0$ we want $\theta^T x \leq -1$ (not jus < 0)
adds safety margin factor

- C very large:

e.g. Linearly seperable case
Tries to separate fororm the data as big as posible
"Large margin classifier"



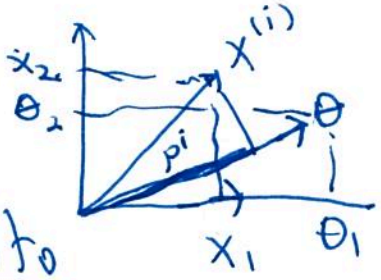
SVM Decision Boundary

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

$\theta^T x^i = ?$

$= p^i \|\theta\|$

and minimizing p^i



x^i and θ are perpendicular, or try to be. So SVM tries to minimize the projections of $x^{(i)}$ into θ

SVM tries that decision boundary is as perpendicular as posible to x^i vectors.
The separation margin are the projections so if projections are big $\|\theta\|$ can be smaller.

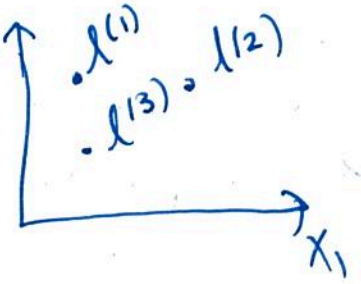
Kernels

Non-Linear Decision Boundary

$h_\theta(x)$ { 1 if $\theta^T x \geq 0$; 0 otherwise

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \dots$ Polynomial

Is there a better choice?



Given x:

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

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

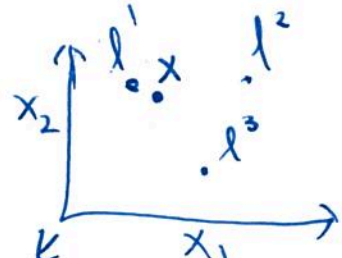
↑ Kernel (Gaussian kernel) function

if $x \approx l^{(1)}$: $f_1 \approx 1$

if x is far from $l^{(1)}$: $f_1 \approx 0$

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

e.g. $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$



$\theta_0 + \theta_1 \Rightarrow -0.5 \geq 0$ ← $f_1 \approx 1$ $f_2 \approx 0$ $f_3 \approx 0$

predict 1

- For point close to $l^{(1)}$ and $l^{(2)}$ predict "1" and far predict "0" σ defines how far or close.
- Choosing landmarks: where?

SVM with Kernels:

Location of landmarks where examples are.

(x^i, y^i): $x^i \rightarrow f^i = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$ $x^{(i)} \in \mathbb{R}^{n+1}$

$f^{(i)} = \begin{bmatrix} f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \in \mathbb{R}^{m+1}$

Given x compute $f \in \mathbb{R}^{m+1}$ (m landmarks)

Predict $y=1$ if $\theta^T f \geq 0$ $\quad \theta \in \mathbb{R}^{m+1}$

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

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

* Kernels with logistic regression very slow.
* Use packages.

- C Large C: Lower Bias, high variance
- C small C: High bias, low variance.
- Large σ^2 High bias features vary smoothly
- Small σ^2 High variance features less smoothly

Use SVM

Software package (e.g. liblinear, libsvm, ...) to solve for θ

→ choose parameter C.
→ choice kernel (similarity function)
e.g. no Kernel ("linear Kernel")
→ In case Gaussian Kernel need to choose σ^2
* Perform feature scaling before!!

- Similarity function needs to satisfy Mercer's Theorem to make sure SVM not diverges.
  - Polynomial Kernel: $K(x,l) = (x^T l + \text{constant})^{\text{degree}}$ use when data is always positive.
  - String Kernel, histogram Kernel

SVM for Multi-Class Classification

- Package
- One vs All

Logistic Regression vs SVM

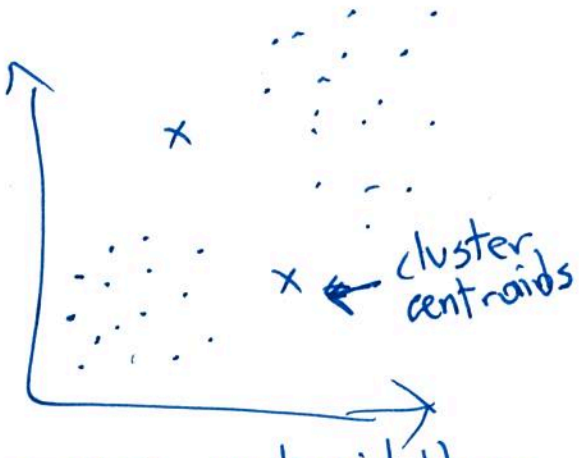
n = # of features
m = # of training examples

- if n is large (relative to m): $n \gg m$
  - ↳ Use logistic Regression or SVM with linear Kernel
- if n is small, m is intermediate
  - ↳ ~~SVM~~ SVM with Gaussian Kernel
- if n is small, m is large
  - SVM will struggle.
  - Create/add more features, then use logistic Regression or SVM without Kernel.
- Neural network will work well for most but may be slower to train.
- SVM (convex function) finds global minima.

Unsupervised Learning

- Clustering : Market segmentation
  e.g. Social Network analysis
  Astronomical data

K-means Algorithm



- Iterative algorithm
- Separates data to closest centroids.
- Calculates mean position of separated cluster and moves centroid there.

repeat

converges to a centroid point.

- Input:
  K (numbers of clusters)
  Training set (unlabeled)
  $x^{(i)} \in \mathbb{R}^n$

- Random K cluster centroids $\mu_1, \mu_2, \cdots \mu_K \in \mathbb{R}^n$

Repeat { for $i = 1$ to m } * Cluster assignment step

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

$\text{index}\left(\min_K \|x^{(i)} - \mu_K\|\right) = c^{(i)}$

Move centroid { for $K = 1$ to K

$\mu_K :=$ average of points assigned to cluster K.

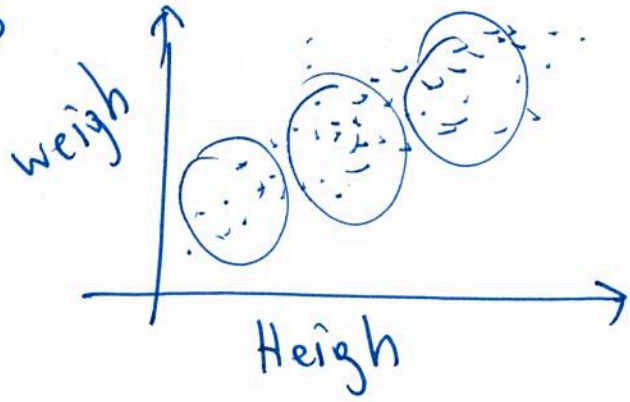
$\mu_2 = \frac{1}{4}[x^{(1)} + x^{(5)} + x^{(6)}] \in \mathbb{R}^n$

eg. if $x^{(1)}, x^{(5)}, x^{(6)}$ are in $K = 2$

* If no element was assigned to a cluster centroid eliminate or randomly assign.

K-means for non-separated clusters

e.g. shirt sizes



- will ~~kind of~~ try to separate data.
- Market segmentation example.

Optimization Objective

$c^{(i)}$ Index of cluster for example $x^{(i)}$

μ_k cluster centroid k

$\mu_{c^{(i)}}$ cluster centroid assign to example $x^{(i)}$

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

Distortion function

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

Random Initialization

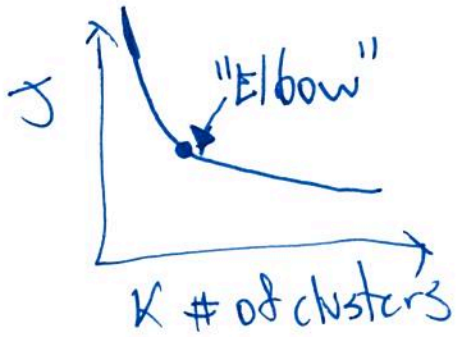
- should $K<m$
- Pick K random training examples
- Set $\mu_1\dots\mu_K$ to these K examples

Local Optima

- ~~Do~~ Many Random initializations
- ~~keep minimum valu~~
- Keep values that give minimum cost function (lowest distortion) for a few clusters.
- For many clusters makes little difference.

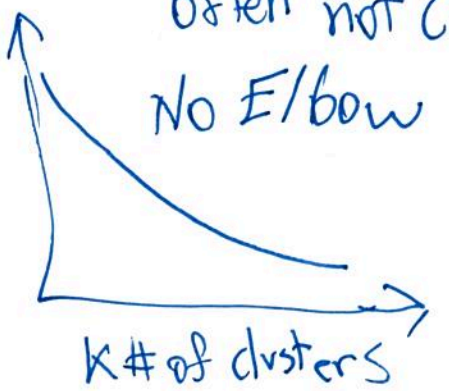
Choosing number of clusters

Elbow Method



often not clear

No Elbow



- Data Compression

  Reduce data from 2D to 1D e.g. inches and cm.

  Can also be in 3D to 2D

  e.g. if Data is in a plane.

- Data Visualization

  Reduce features.

Dimensionality Reduction

PCA: Principal Component Analysis.

Tries to find surface to minimize projections of Data.

- Preprocessing (feature scaling/mean normalization)

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

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

Reduce from n to K dimensions

"Covariance" matrix

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

Compute "eigenvectors" of matrix Σ:

$[U, S, V] = \text{svd}(\text{sigma})$

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

K vectors

$$\Rightarrow U_{reduce} = \begin{bmatrix} | & & | \\ u^{(1)} & \cdots & u^{(K)} \\ | & & | \end{bmatrix} \in n \times K$$

$$z = \underset{K \times n}{U_{reduce}^{t}} \underset{n \times 1}{x} = z \in \mathbb{R}^K \quad \text{if} \quad x = \begin{bmatrix} - & x^{(1)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix}$$

$$\Sigma = \frac{1}{m} X'X;$$

$U_{reduce} = U(:, 1:K)$

$z = U_{reduce}' x$

- Reconstruction from compressed Representation

given $z \in \mathbb{R}^K$ $x_{approx} = U_{reduce} \cdot z$

- How to choose K dimensional features

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

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

Choose K to be smallest value so that

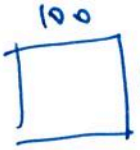
$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01 \ (1\%)$$

"99% of variance is retained"

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

$S = \begin{bmatrix} s_{11} & & & 0 \\ & s_{22} & & \\ & & \ddots & \\ 0 & & & s_{nn} \end{bmatrix}$ for given k check can be done by $1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \leq 0.01$

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

E.G. 

10,000 Pixels $x^{(i)} \in \mathbb{R}^{10,000}$

Unlabeled dataset $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10,000}$

↓ PCA

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

New training set $(z^1, y^1), (z^2, y^2), \ldots (z^m, y^m)$

- PCA only on training set, mapping $x^{(i)} \to z^{(i)}$ to find parameters. Then mapping $x_{cv}^{(i)}$ and $x_{test}^{(i)}$.

- Application of PCA
  - + Compression
  - + Visualization

* Bad Use of PCA: To prevent Overfitting
  Idea is ⇒ less features less likely to overfit.
  Use regularization!!!

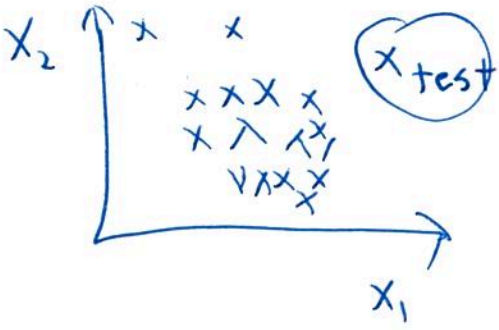
Anomaly Detection

E.g. Aircraft engine features

x_1 = heat generated.

x_2 = vibration intensity.

New engine: x_{test}



Is x_{test} anomalous?

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

- Model $p(x)$

if $p(x_{test}) < \varepsilon$ flag anomaly

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

E.g. Fraud Detection

- $x^{(i)}$ = features of user i's activities.
- Model $P(x)$ from data.
- Identify unusual users.

- Gaussian-Normal Distribution

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

- Density Estimation

- Trainin set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ $\qquad x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$

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

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

- Anomaly Detection algorithm

1- choose features x_i that might indicate anomalous examples.

2- Fit parameters $\mu_1, \dots \mu_n$, & $\sigma_1^2 \dots \sigma_n^2$

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

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

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

Anomaly if $p(x) < \varepsilon$

- Evaluating an anomaly detection system.

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

- Take training set and calculate μ and σ, $p(x)$ only non-anomalous
- CV set and test set use anomalous and non-anomalous.
- set will be very skewed
- Evaluation metrics:
  - True positive, false positive, false negative
  - Precision / Reall
  - F_1-score.
- Can also use CV to set ε.

Anomaly Detection vs Supervised Learning

Anomaly Detection	Supervised Learning
- Very small # of positive examples ($y=1$). (0-20 is common)	- Large # of positive and negative examples.
- Large # of negative ($y=0$) examples.	
- Many different "types" of anomalies.	- Enough positive examples for algorithm to get a sense of what positive examples look like.
- Future anomalies may look nothing like any of the anomalous examples.	- Future + examples likely to be similar to ones in training set.
- Fraud Detection	- Email Spam Classification
- Manufacturing	- Weather Prediction (sunny/rainy/etc)
- Monitoring machines in a data center	- Cancer Classification

Non-gaussian features

- Plot histogram of features.
- Play with transformations of data
  e.g. $\log(x)$, $\log(x+c)$, $\sqrt{}$

Multivariate Gaussian Distribution

$\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$

(covariance matrix)

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

↑ determinant

- Captures correlations between features.
- Original scales better, computationally cheaper.
- Original even if m (training size) is small
- Multi needs $m > n$ or Σ is non-invertible.
  - ↓ use if $m \gg n$
  - $\approx 10n$
- Σ may be non-invertible because of redundant features.

Recommender Systems

-Content-Based

E.g.

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)	
Love at last	5	5	0	0	.9	0	$\leftarrow x^{(1)}$
Romance forever	5			0	1	.01	$\leftarrow x^{(2)}$
Cute Puppies		4	0		.99	0	$\leftarrow x^{(3)}$
Non Stop Car	0	0	5	4	.1	1	$\leftarrow x^{(4)}$
Swords vs Karate	0	0	5		0	.9	

- Each movie represented by feature vector $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
- $n_u = 4$, $n_m = 5$
- $n = 2$ $\quad x^{(1)} = \begin{pmatrix} .9 \\ 0 \end{pmatrix}$

- For each user j, learn parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i $(\theta^{(j)})^T x^{(i)}$ stars

- Problem Formulation

  $r(i,j) = 1$ 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: $(\theta^{(j)})^T (x^{(i)})$

  $m^{(j)}$ = # of movies rated by user j

  To learn $\theta^{(j)}$: $\min_{\theta^{(j)}} \sum_{i:r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right]^2 \frac{1}{2m^{(j)}}$

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

  $+ \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \theta_k^{(j)}$

  For recommended system convention get rid of $m^{(j)}$

Optimitation Objective

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

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

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

Gradien Descend Update:

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

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

* (Collaborative Filtering / feature learning)

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

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

- Randomly Guess θ then x then θ
  - Iterate $\theta \to x \to \theta \to x \dots$

End at θ and x quite good, converge.

Minimize $x^{(1)} \dots x^{(n_m)}$ and $\theta^{(1)} \dots \theta^{(n_u)}$ simultaneously

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

we don't need x_0 and θ_0

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

1- ~~Choose~~ 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. For every $j = 1, \dots, n_u$ and $i = 1, \dots, n_m$:

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

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

3- Giving a user with θ and a movie with (learned) features x, predict a star rating of $\theta^T x$. $(\theta^{(j)})^T (x^{(i)})$.

Vectorization Implementation

Movie	A	B	C	D
Love last	5	5	0	0
Romance 4E	5	?	?	0
Cute puppies	?	4	0	?
Nonstop	0	0	5	4
Karate	0	0	5	?

$n_m = 5$
$n_u = 4$

$$y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}_{5 \times 4}$$

$\uparrow y^{(i,j)}$

Predicted ratings:

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

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

"Low rank matrix factorization"

Finding related movies

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

- How to find movies j related to movie i?

  movies j: $x^{(j)}$ — movie i: $x^{(i)}$

  if small $\|x^{(i)} - x^{(j)}\| \Rightarrow$ movie i and j are similar.

- Mean normalization

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

$$Y \to \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ \vdots & & & & \\ -1.25 & & & & \end{bmatrix} \quad Y = Y - \mu \text{ substract per row.}$$

Y is normalized

For user j non movie i predict

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

Then for a user that hasn't rated anything

$$\theta = \begin{pmatrix} 0 \\ \vdots \end{pmatrix} \quad (\theta^{(5)})^T (x^i) + \mu_i = \mu_i$$

Large Datasets

* Stochastic Gradient descent

Note: Batch gradient descent (all data needed)

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

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

1: Randomly shuffle dataset

2: Repeat { for $i = 1, \cdots, m$ {

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad (\text{for } j = 0, \cdots, n)\}\}$$

- Small steps for each example.
- Ends up around the global minimun.

* Mini-Batch Gradient Descent

~~Batch gradient~~

Use be examples in each iterations

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

Repeat { for $i = 1, 11, 21, 31, \cdots 991$ }

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

b: 10 - 100

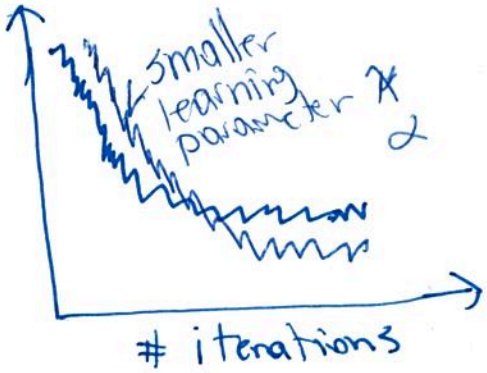
- Use vectorize way ↗ sometimes faster than both.

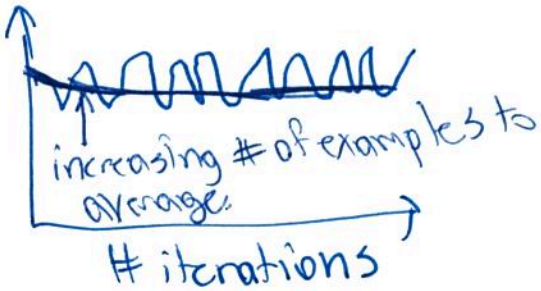
- Convergence

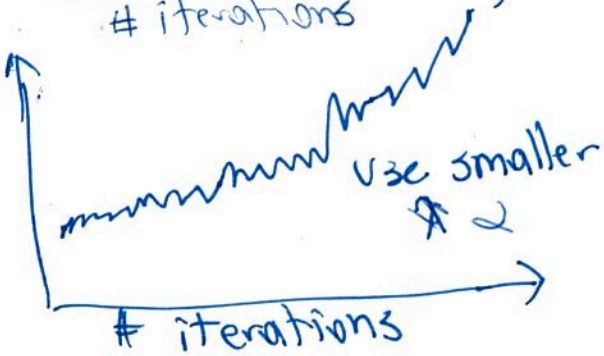
→ Compute $cost(\theta, (x^i, y^i))$ before updating θ using $(x^{(i)}, y^{(i)})$

→ Every 1000 iterations plot $cost(\theta, x^i, y^i))$ averaged over the last 1000 examples processed.

Examples:







- For stochastict to converge to global minima

$$\alpha = \frac{C_1}{\#\text{ iteration} + C_2}$$

Online Learning

- Continous stream of data.

e.g. shipping website, offer ship their ~~package~~ package and users use it ($y=1$) or not ($y=0$).

- features x, price, user properties etc.

Probability for use to optimize price.

$p(y=1|x;\theta)$

** Logistic Regression

Repeat forever{

Get (x,y) correspond to use

Update θ using (x,y): $\theta_j = \theta_j - \alpha(h_\theta(x)-y)x_j$ }

Can adapt to changing user preference.

Map-reduce and data paralellism

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

(x^{101},y^{101}) -- (x^{200},y^{200}) $temp_j^{(2)}$

(x^{201},y^{201}) -- (x^{300},y^{300}) $temp_j^{(3)}$

(x^{301},y^{301}) -- (x^{400},y^{400}) $temp_j^{(4)}$

Combine

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

Many algorithms can be expressed as computing sums of function over the training set.

e.g. logistic regression

- Can algorithm be express be sum over training set?

Photo OCR
(Optical character recognision)

1- text detection

2- character segmentation

3- character classification

* System like this is called a machine learning pipeline.

- Sliding Windows Classifier
  - Pedestrian Classifier: Supervised learning.
    Move window in image to check over the photo.
    Aspect ratio is ~~similar~~ equal always.

- Artificial Data Synthesis
  - In the case OCR use ~~a~~ fonts with background.
  - Use ~~different~~ examples computer and apply transformations or distortions to amplificd the data set.
  - Make sure you have a low bias before getting more data.
  - How much would it to get 10x as much data?.

Ways of getting more data:
- Artificial data
- Collect/label it yourself: calculate # of hours
- Crowd Source (e.g. Amazon Mechanical Turk)

Estimating the errors due to each component - Ceiling Analysis

Component	Accuracy	
Overall system	72%	↓17%
Text Detection	89%	↓ 1%
character Segmentation	90 %	↓10%
character Recognition	100%	

To check every part of the pipeline feed 100% accuracy results.

↖ Manually give exact results and see changes.

Summary:

Supervised:

- Linear regression
- Logistic regression
- Neural Networks
- SVMs

Unsupervised

- K-means
- PCA
- Anomaly Detection

Special ~~Algo~~ applications

- Recommender Systems
- Large Scale machine learning

* Advice on building a machine learning system
  - Bias/Variance
  - Regularization
  - Evaluation of Algorithms
  - Learning Curves
  - Error Analysis
  - Ceiling Analysis.