WEEK #1

1) What is ML.
2) Introductn
3) Quiz #1
4) Other materials
5) Model and Cost Fxn
6) Paramtr Learning
7) Review Quiz
8) Linear Algebra review
9) Review Quiz

WEEK #2

1) Environment Setup Instr.
2) Multivariate Linear Regr.
3) Computng Paramtrs Analtc.
4) Submitting Progr. Assignts.
5) Review Quiz
6) Octave/MATLAB Tutorial
7) Review Quiz

Programming Assignment

WEEK #3

1) Classificatn & Representn.
2) Logistic Regression Model
3) Multiclass Classificatn
4) Review Quiz
5) Solving the Prob. Overfitting
6) Review Quiz

Programming Assignment.

WEEK #4

1) Motivatns
2) NN
3) Applicatns
4) Review (Quiz+PA)

WEEK #5

1) Cost fxn & Backpropagatn
2) Backpropagatn in practice
3) Applicatn of NN
4) Review (QUIZ + P.A.)

WEEK #6

1) Evaluating a Learn. Algor.
2) Bias vs. Variance
3) Review (QUIZ + P.A.)
4) Building a Spam Classifier
5) Handling Skewed Data
6) Using Large Data Sets
7) QUIZ

WEEK #7

1) Large Margin Class'n.
2) Kernels
3) SVM's in Practice
4) Review (QUIZ+ P.A.)

WEEK #8

1) Clustering
2) Review (QUIZ)
3) Motivatn
4) PCA Analysis
5) Applying PCA
6) Review (QUIZ+ P.A.)

WEEK #9

1) Density Estimatn
2) Building a Anomaly Det.S.
3) Multivariate Gaus. Dist.
4) Review (QUIZ)
5) Predicting Movie Ratings
6) Collaborate Filtering
7) Low Rank Mtx Factorztn
8) Review (QUIZ+ P.A.)

WEEK #10

1) Grad. Desc. with Large Datasets
2) Advanced Topics
3) Review (QUIZ)

WEEK #11

1) Photo OCR
2) Review (QUIZ)
3) Conclusion.

WEEK #1

1. What is Machine Learning

- Science of getting computers to learn without being explicitly programmed.
- NN mimics the human brain.

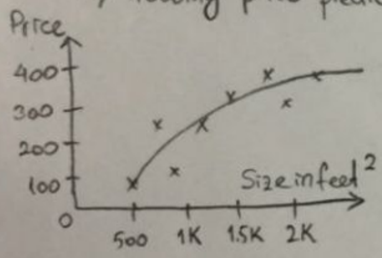
2. INTRODUCTION

2.1. Welcome

- Web search engine: Google, Bing
- Facebook, Apple face recognitn
- Spam filters
- Autonomous robots
- Computatnl biology
- Handwriting recognitn.
- NLP: Natural Language Processing
- Computer Vision
- Self-customizing programs
- In ∀ fields of engineering there is a huge amount of data sets, that we are trying to understand using learning algorithms.

2.2. Supervised Learning.

ex/ Housing price predictn.



ex/ Breast cancer (malignant, benign)

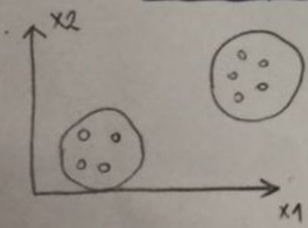
⇒ Supervised Learning: "right answers" given

⇒ Regression: Predict cont^{s} valued outputs

⇒ Classificatn: Descrete valued outputs (0 or 1) / (0, 1, 2, 3)

⇒ # of features may vary by problem

2.3. Unsupervised Learning:



ex/ Google News, Social network analysis, Astronomical data analysis, Market segmentatn.

⇒ Unsuperviced Learning: we are given a data that does not have any labels / that ∀ has the same label.

Cocktail Party Problem Algorithm:

```
[W,s,v] = svd((repmat(sum(x.*x,1), size(x,1),1).*x)*x');
```

3. QUIZ #1 (100% ✓)

4. OTHER MATERIALS:

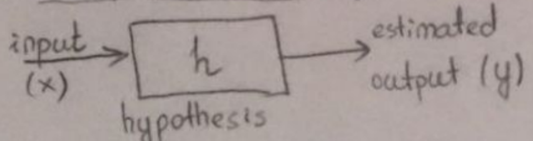
The Course Wiki

5. MODEL & COST FUNCTION

5.1. Model Representatn:

m = # of training examples
x's = "input" variable/features
y's = "output"/"target" variable.

h maps from x's to y's:



$h_\theta(x) = \theta_0 + \theta_1 x$ (univariant linear regression)

5.2. Cost Functn:

θ_0, θ_1: parameters.

→ Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples

$$\boxed{\underset{\theta_0,\theta_1}{\text{minimize}} \sum_{i=1}^{M} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \cdot \frac{1}{2m} = J(\theta_0, \theta_1)}$$

Cost func.

(squared error func.n)

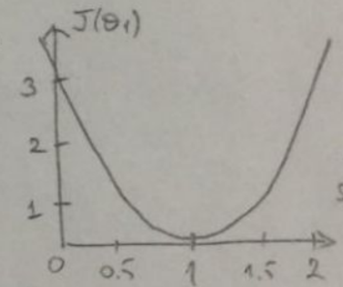
5.3. Cost Func'n - Intuitn 1:

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Paramtrs: θ_0, θ_1

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

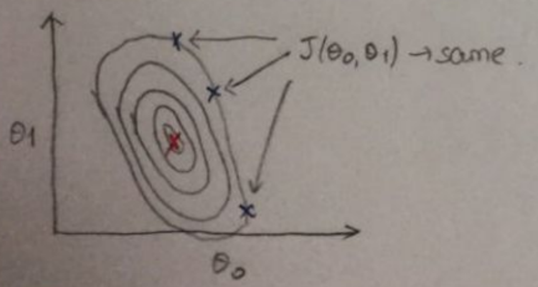
Goal: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0,\theta_1)$



⇒ Each value of θ_1 corresponds to a different hypothesis and we can get different $J(\theta_1)$

for $x^{(i)} = y^{(i)} \longrightarrow J(\theta_1) = 0$.

5.4 Cost Func'n - Intuitn 2:



6. PARAMETER LEARNING

6.1. Gradient Descend:

- for minimizing some arbitrary fxn J. ✓ Have some fxn $J(\theta_0,\theta_1)$ ✓ Want $\min_{\theta_0,\theta_1} J(\theta_0,\theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

⇒ If you are standing at the pnt on the hill, look around and find that the best drn is to take a little step downhill is roughly that drn.

Gradient Descend Algorithm:

repeat until convergence {

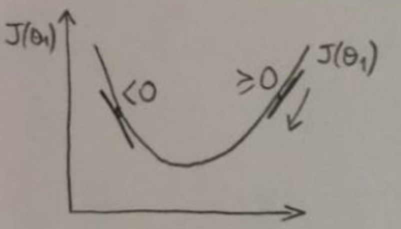
$$\theta_j := \theta_j - \underset{\uparrow}{\alpha} \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j=0 \;\&\; j=1)$$

} learning rate (step size)

⇒ simultaneous update:

$$\boxed{\begin{aligned} temp0 &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ temp1 &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_0 &:= temp0 \\ \theta_1 &:= temp1. \end{aligned}}$$

6.2. Gradient descend Intuitn:



if derive ≥ 0

$\theta_1 := \theta_1 - \alpha$ (pos've #) → decrease θ_1

if derive < 0

$\theta_1 := \theta_1 - \alpha$ (neg've #) → increase θ_1

⇒ if α is too small, gradient descend can be slow

⇒ if α is too large, gradient descend can overshoot the minimum. It may fail to converge or even diverge.

⇒ Gradient descend can converge to a local minimum, even with the learning rate α fixed

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descend will automatically take smaller steps. So, no need to decrease α over time.

6.3. Gradient descend for Linear Regression:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \quad \underline{j=0}$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)} \quad \underline{j=1}$$

"Batch" Gradient descend: Each step of gradient descend uses ∀ the training examples.

⇒ Normal eq'n method can solve this minimizatn problem numerically, but Gradient descend can scale better to larger data sets than that normal eqn method.

WEEK #2.

2 MULTIVARIATE LINEAR REGRESSION

2.1. Multiple features:

ex/ 1) size x_1
2) # of bedrooms x_2
3) # of floors x_3
4) Age of home x_4

} Price y

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$x^{(i)}$ = input (features) of i^{th} training example.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

hypothesis

$$h_\theta(x) = \theta^T x = [\theta_0 \ \theta_1 \ \dots \ \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

2.2. Gradient descend for multiple variables: (n variables/features)

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

Parameters: θ $(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$ "n+1" dimentional vector.

Cost func'n: $J(\theta) = J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat {

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

}

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

for $n=2$:

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

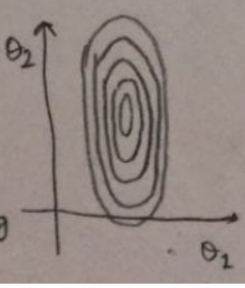
2.3. Gradient descent in Practice 1 - Feature Scaling:

⇒ Make sure that the features are on a similar scale, then gradient descent can converge more quickly!

ex/ x_1 = size (0 – 2000 feet^2)

x_2 = # of bedrooms (1 – 5)

If you ran grad. descent, on this cost fxn, gradients can take a long time to reach global minimum.

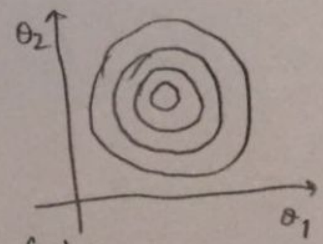


$x_1 = \frac{\text{size (feet}^2)}{2000}$

$x_2 = \frac{\text{\# bedrooms}}{5}$

$0 \leq x_1, x_2 \leq 1$

Can converge much faster.



⇒ Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

$0 \leq x_1 \leq 3$ ✓ $\quad -2 \leq x_2 \leq 0.5$ ✓ $\quad -100 \leq x_3 \leq 100$ ✗ $\quad -0.0001 \leq x_4 \leq 0.0001$ ✗

Mean Normalizatn:

Replace x_i with $x_i - \mu_i$ to make features have ~ zero mean (Do not apply to $x_0 = 1$)

ex/ $x_1 = \frac{size - 1000}{2000}$ $\quad -0.5 \leq x_1 \leq 0.5$

$x_2 = \frac{\#bedrooms - 2}{5}$ $\quad -0.5 \leq x_2 \leq 0.5$

$\boxed{x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}}$ ← avg. value of x_1 in training set

range (or standard deviation)
(max-min)

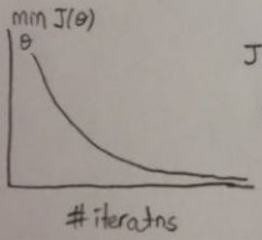
$\boxed{x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}}$

2.4. Gradient Descent in Practice 2 - Learning Rate:

Gradient descent

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

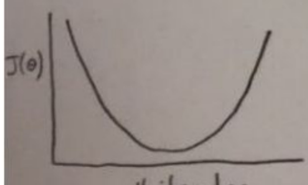
- "Debugging": how to make sure grad.desc. is working correctly
- How to choose learning rate α.



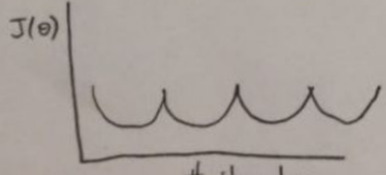
$J(\theta)$ should decrease after every iteratn

Example automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} (ε) in one iter'n.



if grad.desc. does not work ⇒ use smaller α.



⇒ For sufficiently small α, $J(\theta)$ should decrease on every iteratn.

⇒ But if α is too small, grad.desc. can be slow to converge

Summary: - if α is too small $\rightarrow$ slow convergence
- if α is too large $\rightarrow J(\theta)$ may not decr. on every itrn, may not converge

Try to choose α: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...

2.5. Features and Polynomial Regression:

$$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2}$$

$\Rightarrow$ You can create new features by yourself.

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

land area $(x_1 \cdot x_2)$

$$h_\theta(x) = \theta_0 + \theta_1 (\text{size}) + \theta_2 (\text{size})^2 + \theta_3 (\text{size})^3$$

3. COMPUTING PARAMETERS ANALYTICALLY.

3.1. Normal Equatn: method to solve for θ analytically.

$$\left.\begin{aligned} & J(\theta) = a\theta^2 + b\theta + c \\ & \frac{d}{d\theta} J(\theta) = 0 \\ & \text{solve for } \theta \end{aligned}\right\} \theta \in \mathbb{R}$$

$$\left.\begin{aligned} & J(\theta_0, \theta_1, \dots \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \\ & \frac{\partial}{\partial \theta_j} J(\theta) = 0. \quad \text{(for every } j\text{)} \\ & \text{solve for } \theta_0, \theta_1, \dots, \theta_n \end{aligned}\right\} \theta \in \mathbb{R}^{n+1}$$

$$X = \underbrace{\begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ -(x^{(m)})^T- \end{bmatrix}}_{(m \times (n+1))} \qquad y = \underbrace{\begin{bmatrix} \\ \\ \end{bmatrix}}_{(m \times 1)}$$

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

$$\boxed{\text{pinv } (X' * X) * X' * y}$$

$\Rightarrow$ if you are using Normal Eqn method $\Rightarrow$ feature scaling is not actually necessary

m training example, n features:

Gradient descent:	Normal Eqn:
- Need to choose α	- No need to choose α
- Needs many iteratns.	- Don't need to iterate.
- Works well even when n is large (10K)	- Need to compute $(X^T X)^{-1}$
	- Slow if n is very large ($\geq$ 10K)

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

- what if $X^T X$ is non-invertible? (singular/degenerate)

pinv — pseudo inverse

inv — inverse

→ you may have redundant features (linearly dependent)

→ you may have too many features ($m \leq n$)

- delete some features, or use regularizatn
- use more training examples.

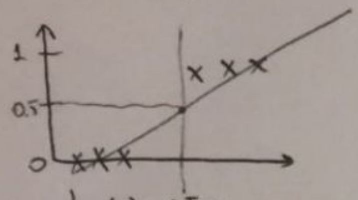
4. Programming Assignment (✓)

WEEK #3

1 CLASSIFICATION & REPRESENTATION.

1.1. Classificatn:

$y \in \{0,1\}$ $\quad y \in \{0,1,2,3\}$



$h_\theta(x) = \theta^T x$

Threshold classifier output $h_\theta(x)$ at 0.5

if $h_\theta(x) \geq 0.5$, predict "$y=1$"

if $h_\theta(x) < 0.5$, predict "$y=0$"

⇒ Applying linear regression to a classificatn problem often is not a great idea

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

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

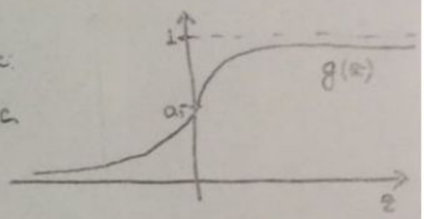
1.2. Hypothesis Representatn:

Logistic Regression Model

Want $0 \leq h_\theta(x) \leq 1$

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

sigmoid func.
logistic func.



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

Interpretatn of hypothesis output:

$h_\theta(x)$ = estimated prob. that $\boxed{y=1}$ on input x

ex/ $h_\theta(x) = 0.7$ ⇒ 70% chance of tumor being malignant.

$h_\theta(x) = P(y=1 \mid x;\theta)$ "Prob. that $y=1$, given x, parameterized by θ".

$$P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1$$

$$P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta)$$

1.3. Decision Boundary:

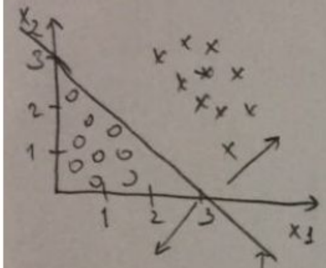
$h_\theta(x) = g(\theta^T x)$ → Try to understand better when this hypothesis will make predictns that $y=1$ vs. when it might make predictns that $y=0$.

$g(z) = \frac{1}{1+e^{-z}}$ → Understand better what hypothesis fxn looks like particularly when we have more than one feature.

$g(z) \geq 0.5$ when $z \geq 0$ $(y=1) \longrightarrow \theta^T x \geq 0$

$g(z) < 0.5$ when $z < 0$ $(y=0) \longrightarrow \theta^T x < 0$

Decision Boundary: (2 features)



$x_1 + x_2 = 3$
(decision boundary)

$h_\theta(x) = g(\overset{-3}{\theta_0} + \overset{1}{\theta_1} x_1 + \overset{1}{\theta_2} x_2)$

Predict "$y=1$" if $-3 + x_1 + x_2 \geq 0$

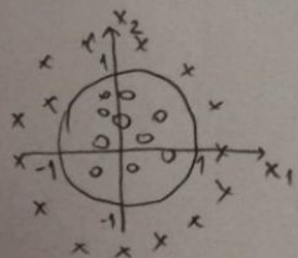
$x_1 + x_2 \geq 3 \longrightarrow y=1$

$x_1 + x_2 < 3 \longrightarrow y=0$

$x_1 + x_2 = 3 \rightarrow h_\theta(x) = 0.5$

⇒ Once we have particular values for the paramtrs $\theta_0, \theta_1, \theta_2$ ⇒ that completely defines the decision boundary and we do not need to plot a training set i.o.t. plot the decision boundary.

Non-Linear decision boundaries:



$h_\theta(x) = g(\overset{-1}{\theta_0} + \overset{0}{\theta_1} x_1 + \overset{0}{\theta_2} x_2 + \overset{1}{\theta_3} x_1^2 + \overset{1}{\theta_4} x_2^2)$

Predict "$y=1$" if $-1 + x_1^2 + x_2^2 \geq 0$

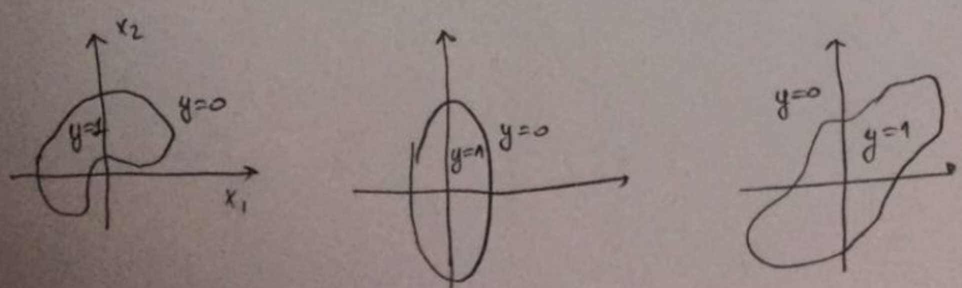
$x_1^2 + x_2^2 \geq 1$

$\boxed{x_1^2 + x_2^2 = 1}$

circle eqn.
decision boundary.

⇒ Add higher order features like in polynomial regression.

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$



2. LOGISTIC REGRESSION MODEL

2.1. Cost Function

Training set – m examples

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad x_0 = 1 \quad y \in \{0,1\} \qquad h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

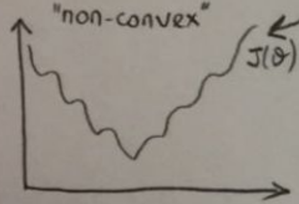
How to choose paramtrs θ?

Cost fxn:

Linear Regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2}_{cost(h_\theta(x^{(i)}),\, y^{(i)})}$

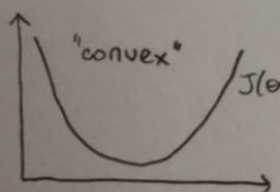
$cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$

– for logistic regression, $\frac{1}{1+e^{-\theta^T x}}$, pretty complicated, nonlinear fxn



→ if you run grad. desc. on this fxn, it is not guaranteed to converge to the global minimum.

"LOGISTIC REGRESSION" — $J(\theta)$ – non convex fxn if we define a square cost fxn.



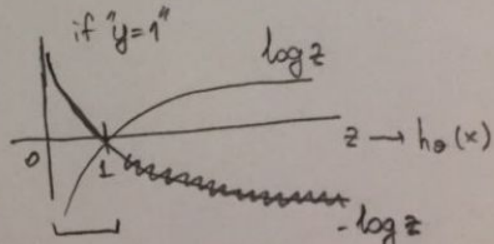
→ we would be guaranteed that grad. desc. would converge to the global minimum.

"LINEAR REGRESSION"

⇒ We want $J(\theta)$ to be convex fxn (for logistic regression), so we can apply a great algorithm (grad. desc.)

Logistic Regression Cost fxn:

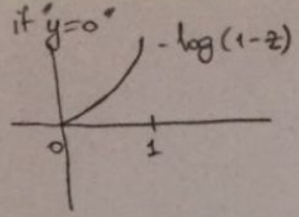
$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$$



Cost = 0 if $y=1$, $h_\theta(x) = 1$

as $h_\theta(x) \to 0$, Cost $\to \infty$

if $h_\theta(x) = 0$ $(P(y=1|x;\theta) = 0)$, but $y=1$, we will penalize learning algorithm by a very large cost.



2.2. Simplified Cost Fxn & Gradient descent.

Logistic Regression cost fxn:

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

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

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

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

→ This cost fxn can be derived from statistics using the principle of max. likelihood estimatn. Which is an idea in statistics for how to efficiently find paramtrs' data for different models.

To fit paramtrs θ: $\min_\theta J(\theta)$

To make a predictn given new x: Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

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

Repeat {

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

} [simultaneously update $\forall\ \theta_j$]!

looks like identical to linear regression!

linear regression: $h_\theta(x) = \theta^T x$

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

⇒ feature scaling can help grad. desc. converge faster for both linear regr. & logis. regr.

2.3. Advanced Optimizatn:

Optimizatn algorithm:

Cost fxn $J(\theta)$. Want $\min_\theta J(\theta)$

Given θ, we have code that can compute
- $J(\theta)$
- $\frac{\partial}{\partial\theta_j} J(\theta)$

Grad. descent:

Repeat {

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

}

Optimizatn algorithms:

	Pros:	Cons:
→ Gradient Descent		
→ Conjugate Gradient → BFGS → L-BFGS	- No need to manually pick α - has clever inner-loop: line-search algorithm automatically tries out different α & pics α. - Often faster than grad.desc. (converge much faster).	- more complex

ex/ $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5) \qquad \theta_1 = 5$

$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5) \qquad \theta_2 = 5$

```
function [jVal, gradient] = costFunction (theta)
    jVal = (θ1-5)^2 + (θ2-5)^2          → J(θ)
    gradient = zeros (2, 1)              →
    gradient (1) = 2(θ1-5)               → ∂/∂θ1 J(θ)
    gradient (2) = 2(θ2-5)               → ∂/∂θ2 J(θ)

options = optimset ('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros (2,1);
[optTheta, functionVal, exitFlag, ....] =
        = fminunc (@costFunction, initialTheta, options);
```

[fminunc = func. minimizatn unconstrained]

3. MULTICLASS CLASSIFICATION.

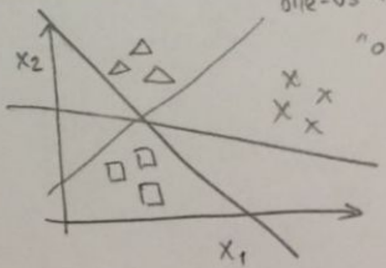
3.1. Multiclass Classificatn: One-vs-All:

"one-vs-rest"

"one-vs-all"

Email foldering: Work, Friends, Family, Hobby
$y=1, \quad y=2, \quad y=3, \quad y=4$

Weather: Sunny, Cloudy, Rain, Snow



⇓

3 seperate classificatn problem.

$h_\theta^{(i)}(x) = P(y=i \mid x; \theta) \quad (i = 1, 2, 3)$

⇒ Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$

⇒ On a new input x, to make prediction, pick the class i that maximizes

$\max_i h_\theta^{(i)}(x)$

4. QUIZ (100% ✓)

Scanned by CamScanner

WEEK # 4

1. MOTIVATIONS.

1.1. Non-linear hypothesis:

Non-linear Classificatn:

$$g(\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1x_2+\theta_4x_1^2x_2+\theta_5x_1^3x_2+\theta_6x_1\cdot x_2^2+\ldots)$$

this method works well when we have only 2 features (x_1, x_2)

But for many ML problems would have a lot more features!

if we have $n=100$ features, you end up with ≈ 5000 features (quadratic features!) $\approx \frac{n^2}{2}$. Because of many features, it may overfit the training set.

→ if you add qubic polynomials, there would be $\approx 170\,000$ features. $(O(n^3))$

ex/ if we have 50x50 pixel img, there are $n=2500$ (7500 if RGB) features.

θ quadratic features $\approx$ 3 million features.

if you try to classify imgs only by looking the pxl values, probably you will overfit! can not classify imgs.

1.2. Neurons and the Brain:

NNs: Algorithms that try to mimic the brain.

The "one learning algorithm" hypothesis. Audotory cortex learns to see. Somatosensory cortex learns to see. } neural-rewiring experiments.

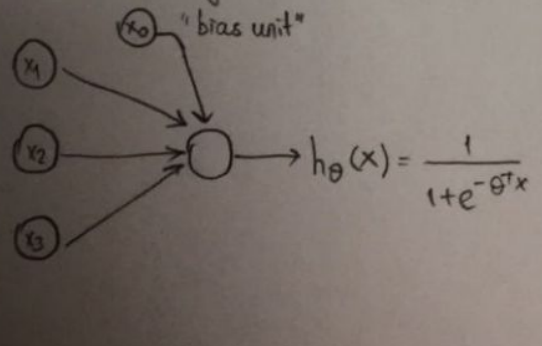
2. NEURAL NETWORKS

2.1. Model Representation -1:

Dendrite — "input wires"

Axon — "output wire"

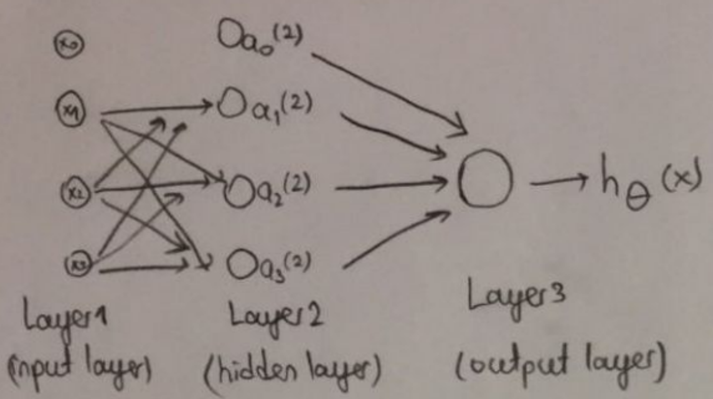
Neural model: Logistic unit



Sigmoid (logistic) activation fxn

Weights $= \theta$'s. $= \begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \vdots\\ \theta_n\end{bmatrix}$ paramtrs.

NN:



$a_i^{(j)}$ = "activatn" of unit i in layer j

$\Theta^{(j)}$ = mtx of weights controlling fcn mapping from layer j to layer $j+1$

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

$\vdots$

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

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ dmn will be $s_{j+1} \times (s_j + 1)$

2.2. Model Representatn -2:

Forward Propagatn: Vectorized implementatn.

$a_3^{(2)} = g(z_3^{(2)})$ $\quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$ $\quad z^{(2)} = \Theta^{(1)} \cdot x$

$$\underbrace{a^{(2)}}_{\mathbb{R}^3} = g(\underbrace{z^{(2)}}_{\mathbb{R}^3})$$

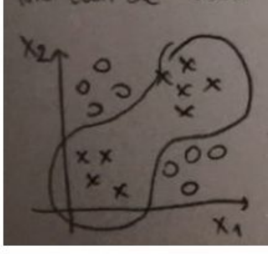
⇓

Instead of being contrained to feed the features (x_1, x_2, x_3) to logistic regression, it gets to learn its own features (a_1, a_2, a_3) to feed into logistic regression.

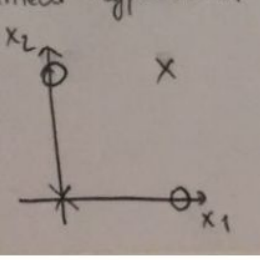
⇒ it can learn some pretty interesting and complex features & can end up with better hypothes

③ APPLICATIONS.

NN can be used to learn complex non-linear hypothesis.



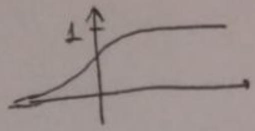
we'd like to learn a non-linear decision boundary

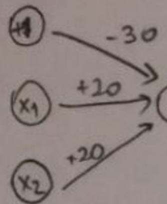


$y = x_1$ XOR x_2

x_1 XNOR x_2

AND $y = x_1 \text{ AND } x_2$





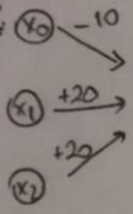
$\rightarrow h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

$\Theta_{10}^{(1)} \quad \Theta_{11}^{(1)} \quad \Theta_{12}^{(1)}$

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	$g(10) \approx 1$

$h_\Theta(x) \approx x_1 \text{ AND } x_2$

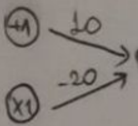
OR:



$\rightarrow h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$

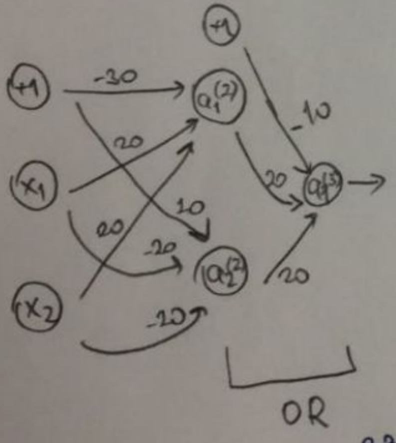
3.2. Examples & Intuitns - 2:

NOT:



$\rightarrow h_\Theta(x) = g(10 - 20x_1)$

XNOR:



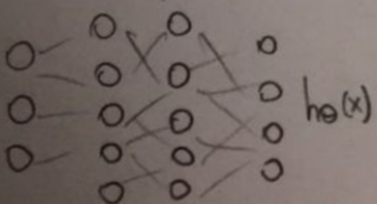
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

3.3. Multiclass Classificatn:

multiple output units: one-vs-all (pedestrian, car, motorcycle, truck)

Want $h_\Theta(x) = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $h_\Theta(x) = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

when pedestrian, when car, when motorcycle, when truck.



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

WEEK #5

1. COST FUNCTION AND BACKPROPAGATION.

1.1. Cost Functn:

NN (Classificatn)

L = Total # of layers in network.

s_l = # of units in layer l (excluding bias unit)

Binary classificatn: 1 output unit. $h_\theta(x) \in \mathbb{R}$

Multi-Class classfctn: K output units. $h_\theta(x) \in \mathbb{R}^K$

Cost Func:

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

NN:

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

1.2. Backpropagatn Algorithm:

$\min_\theta J(\theta)$

Need code to compute: $\rightarrow J(\theta)$

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

Gradient Computatn:

$(x,y) \longleftarrow$ training example

$a^{(1)} = x$

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

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

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

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

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

$a^{(4)} = g(z^{(4)}) = h_\theta(x)$ (add $a_0^{(4)}$)

Backpropagatn Algorithm:

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

For each output unit (layer $L=4$)

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

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

$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) \rightarrow a^{(2)} .* (1-a^{(2)})$

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

For the huge # of training set.

$\Delta_{ij}^{(l)} = 0$ (for $\forall$ i,j,l) (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

for i=1 to m

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

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

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

compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ $\cancel{\delta^{(1)}}$

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

$$\left.\begin{aligned} D_{ij}^{(l)} &= \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0 \\ D_{ij}^{(l)} &= \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0. \end{aligned}\right\} \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

end.

1.3. Backpropagatn Intuitn:

$z_1^{(3)} = \Theta_{10}^{(2)} + \Theta_{11}^{(2)} \cdot a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_1^{(2)}$

Backpropagatn: (running the feedforward algorithm, but doing it backwards.)

$cost(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$ [Think of $cost(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$]

how well is the network doing on example i?

$\delta_2^{(2)} = \Theta_{12}^{(2)} \cdot \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$

$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$.

2. BACKPROPAGATION IN PRACTICE.

2.1. Implementatn Note: Unrolling Paramtrs:

NN: (L=4)

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

$D^{(1)}, D^{(2)}, D^{(3)}$ — matrices (D1, D2, D3)

"Unroll" into vectors.

ex/ $S_1 = 10$, $S_2 = 10$, $S_3 = 1$

$\Theta^{(1)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(2)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(3)} \in \mathbb{R}^{1 \times 11}$

$D^{(1)} \in \mathbb{R}^{10 \times 11}$, $D^{(2)} \in \mathbb{R}^{10 \times 11}$, $D^{(3)} \in \mathbb{R}^{1 \times 11}$

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];
Theta1 = reshape(thetaVec(1:110), 10, 11);
```

Learning Algorithm:

- have initial paramtrs $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
- unroll to get initialTheta to pass to

fminunc (@costFunction, initialTheta, options).

function [jval, gradientVec]= costFunction (thetaVec)

- from thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
- use forward prop/back prop. to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$
- unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec.

2.2. Gradient Checking:

It may look $J(\Theta)$ is decreasing, but you might end up with a NN that has a higher level of error than you would with a bug free implementatn.

gradApprox = (J(theta + ε) − J(theta − ε)) / (2ε)

```
for i=1 to n
    θp = θ;
    θp(i) = θp(i)+ε;
    θm = θ;
    θm(i) = θm(i)−ε;
    gradApprox(i) = (J(θp) − J(θm)) / (2ε);
end
```

Check that

$\underbrace{\text{gradApprox}}_{\text{from numerical}} \approx \underbrace{\text{DVec}}_{\text{from back.prop}}$

1) Implement back prop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$)
2) Implement numerical gradient check to compute gradApprox
3) Make sure they give similar values.
4) Turn off gradient check
5) Use back prop code for learning

2.3. Random Initializatn:

→ Zero initializatn: $\Theta_{ij}^{(l)} = 0$ for $\forall$ i, j, l.

for every training examples $a_1^{(2)} = a_2^{(2)}$. $\delta_1^{(2)} = \delta_2^{(2)}$

$$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$$

$$\downarrow$$

updated weights would be equal to each other ← $\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

⇒ After each update, paramtrs corresponding to inputs going into each of two hidden units are identical. $a_1^{(2)} = a_2^{(2)}$

⇒ $a_1^{(2)} = a_2^{(2)} = a_3^{(2)} = \ldots = a_n^{(2)}$ means that ∀ of our hidden units are computing the exact same func. of the input! (highly redundant = equivalent with a unique feature, one unit on that layer).

INSTEAD!

→ Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

Theta1 = rand(10,11) * (2ε) − ε; ⟶ $\boxed{-\varepsilon < \text{Theta1} < \varepsilon}$

Theta2 = rand(1,11) * (2ε) − ε; ⟶ $\boxed{-\varepsilon < \text{Theta2} < \varepsilon}$

2.4. Putting It Together:

→ The more hidden units → the better.

Training a NN.

1. Randomly initialize weights
2. FP ($h_\Theta(x^{(i)})$)
3. Compute $J(\Theta)$
4. BP ($\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$)

for $i = 1$ to m
   FP & BP using $(x^{(i)}, y^{(i)})$
   (Get activatns $a^{(l)}$ & deltas $\delta^{(l)}$ for $l = 2, 3, \ldots, L$)
   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
end

compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

5. use grad. checking.
   then disable checking code.
6. Use grad. desc. or advanced optimizatn method with backprop to try to minimize $J(\Theta)$ as a fcn of Θ.

$J(\Theta)$ — "non-convex", global minimum is not guaranteed, but in practice this is not a huge problem.

! What will if we try to maximize $J(\Theta)$ at first, then initialize the weights with those (maximized $J(\Theta)$), in order to, hopefully, end up with global minimum, or at least, with better local minimum?!

Scanned by CamScanner

3. APPLICATION OF NN.

! What will it we apply BRIEF based logic into NNs, rather than CNNs.

→ Automous driving.

Once every 2 seconds, ALVINN digitizes a video img of the road ahead, and records the person's steering drn.

(3 layered Network)

→ This same procedure is repeated for other road types

WEEK #6
Advice for Applying M.L.

1. EVALUATING A LEARNING ALGORITHM.

1.1. Deciding What to Try Next.

→ If you are developing ML, stms, that you know how to choose one of the most promising avenues to spend your time pursuing.

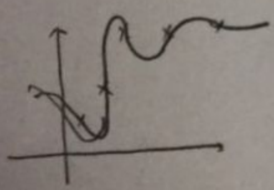
⇒ Debugging a learning algorithm.

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

→ Get more training examples
→ Try smaller sets of features (to prevent overfitting)
→ Try getting additional features (may be current features aren't informative enough & u want to collect more data in the sense of getting more features).
→ Try adding polynomial features (x_1^2, x_2^2, x_1x_2, etc.)
→ Try decreasing λ
→ Try increasing λ

⇒ M.L. diagnostic: test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

1.2. Evaluating a Hypothesis:



Fails to generalize to new examples not in training set
How to be sure about overfitting?
Divide the training set into 2 groups: → training set (70%)
↘ test set (30%)

→ Learn paramtr θ from training data
→ Compute test error:

1.3. Model Selectn and Train/Validatn/Test Sets:

⇒ Model Selection:
d-degree of polynomial.
d=1 ⟶ Training error ($\theta^{(1)}$) ⟶ Test error ($\theta^{(1)}$)
⋮
d=5 ⟶ Training error ($\theta^{(5)}$) ⟶ Test error ($\theta^{(5)}$)
$J_{test}(\theta^{(5)})$
⋮

→ $J_{test}(\theta^{(5)})$ is like to be an optimistic estimate of generalizatn error, i.e. our extra paramtr (d) is fit to test set.

→ Divide the dataset into 3 groups: training set (60%) → $J_{train}(\theta)$
Cross-Validatn set (20%) → $J_{cv}(\theta)$
test set (20%) → $J_{test}(\theta)$

Use CV set to select the d-paramtr (rather than using test set).

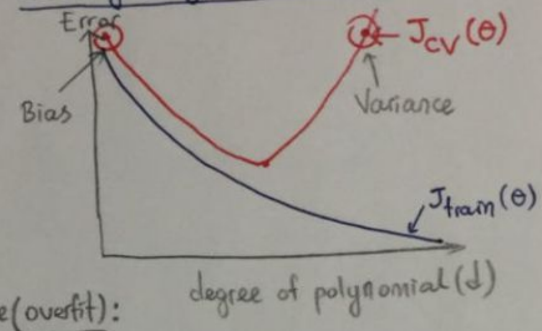
Pick the optimum hypothesis (ex/ $\theta_0+\theta_1 x+\theta_2 x^2+\theta_3 x^3+\theta_4 x^4$)

Estimate generalizatn error for test set $J_{test}(\theta^{(4)})$

(2.) BIAS Vs. VARIANCE

2.1. Diagnosing bias Vs. variance:

bias → underfitting (d=1)
variance → overfitting (d=10)
just right (d=5)



Bias (underfit):
$J_{train}(\theta)$ → high
$J_{cv}(\theta)$ → high

Variance (overfit):
$J_{train}(\theta)$ → low
$J_{cv}(\theta) \gg J_{train}(\theta)$

2.2. Regularizatn and Bias/Variance:

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

Large λ:
High bias, underfit
$\lambda=10K$, $\theta_1\approx 0$, $\theta_2\approx 0$
$h_\theta(x)\approx 0$

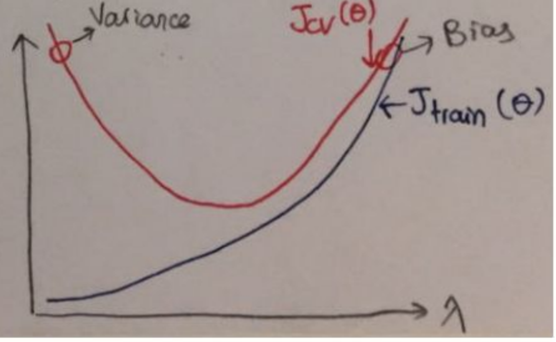
Intermediate λ:
Just Right

Small λ:
High Variance, Overfit
$\lambda=0$.

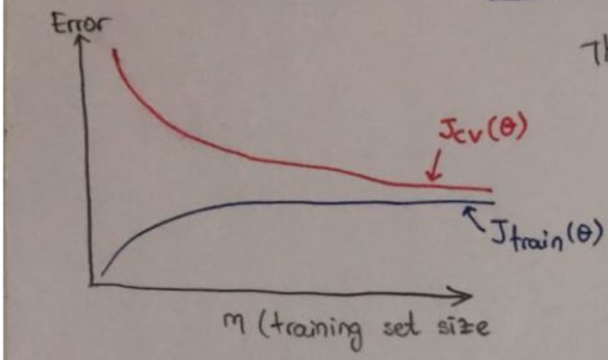
⇒ Choosing the regularizatn error:

1) $\lambda=0 \longrightarrow \min_\theta J(\theta) \longrightarrow \theta^{(1)} \longrightarrow J_{cv}(\theta^{(1)})$
2) $\lambda=0.01$
3) $\lambda=0.02$
4) $\lambda=0.04 \longrightarrow \min_\theta J(\theta) \longrightarrow \theta^{(4)} \longrightarrow J_{cv}(\theta^{(4)})$
⋮
12) $\lambda=10 \longrightarrow \theta^{(12)} \longrightarrow J_{cv}(\theta^{(12)})$

Pick (say) $\theta^{(5)}$ → Test Error: $J_{test}(\theta^{(5)})$

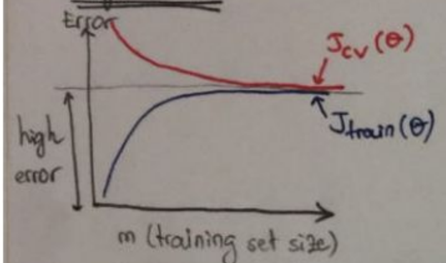


2.3. Learning Curves:



The more data you have ⇒ the better the hypothesis you fit.

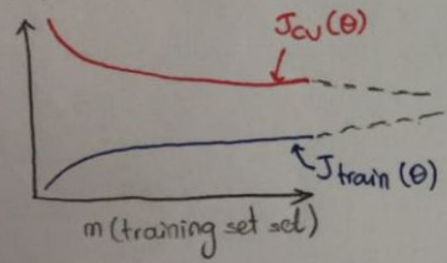
High Bias:



J_{cv} & J_{train} will be very similar

If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

High Variance:



→ Large gap btw training error & cv-error

If a learning algorithm is suffering from high variance, getting more training is likely to help.

2.4. Deciding What to do Next (Revisited)

→ Get more training examples → fixes high variance
→ Try smaller sets of features → fixes high variance
→ Try getting additnl features → fixes high bias
→ Try adding polynomial features → fixes high bias
→ Try decreasing λ → fixes high bias
→ Try increasing λ → fixes high variance

Small NN
- computatnly cheaper
- prone to underfitting

Large NN
- more expensive
- prone to overfitting
- Use regulatn ↗
- Select # of hidden layers.

3. REVIEW: QUIZ + P.A.

4. BUILDING A SPAM CLASSIFIER.

4.1. Prioritizing What to Work On:

1) x=features of email , y= 1 (spam) / 0 (non-spam)

features x: choose 100 words indicative of spam/non-spam.

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

2) collect lots of data
3) develop sophisticated features based on email routing info.
4) " " " for message body ("discounts"...)
5) " " algorithm to detect misspellings (med1cine)

4.2. Error Analysis:

1) start with simple algorithm
2) plot learning curves
3) error analysis: manually examine the examples

ex/ 100 errors on cathogorizatn of spam classifier:

i) what type of mail it is
ii) what cues (features) you think would have helped the algorithm to classify.

4.3. Error metrics for skewed classes:

Find that you got 1% error on test set.

→ Only 0.50% of patients have cancer.
↳ skewed classes.

```
function y=predictCancer(x)
    y=0;      → 0.5% error gives
end
```

recall=0

Precision /Recall. (y=1 in presence of rare class)

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

		Actual class 1	Actual class 0
Predicted Class	1	True positive	False positive
Predicted Class	0	False negative	True negative

Precision: $\frac{\text{true positive}}{\#\text{ predicted positive}} = \frac{\text{True positive}}{\text{True pos} + \text{False pos}}$

↑ good

Recall: $\frac{\text{True pos}}{\#\text{actual pos}} = \frac{\text{True pos}}{\text{True pos} + \text{False neg.}}$

↑ good

5. HANDLING SKEWED DATA.

5.2. Trading Off Precision and Recall.

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

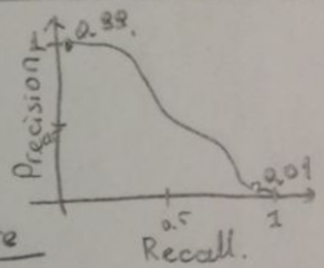
Suppose we want to predict $y=1$ (cancer) only if very confident.

$$\left.\begin{array}{l}\text{Predict } 1 \text{ if } h_\theta(x) \geq 0.7 \\ \text{Predict } 0 \text{ if } h_\theta(x) < 0.7\end{array}\right\} \begin{array}{l}\rightarrow \text{Higher precision.} \\ \rightarrow \text{Lower recall.}\end{array}$$

Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

$$\left.\begin{array}{lll}1 & \text{if} & h_\theta(x) \geq 0.3 \\ 0 & \text{if} & h_\theta(x) < 0.3\end{array}\right\} \begin{array}{l}\rightarrow \text{Higher recall} \\ \rightarrow \text{Lower precision.}\end{array}$$

More generally: Predict 1 if $h_\theta(x) \geq$ threshold



F_1 score (F score)

	Precision (P)	Recall (R)	~~Average~~	F_1 score
Algor. 1	0.5	0.4	~~0.45~~	0.444
Algor. 2	0.7	0.1	~~0.4~~	0.175
Algor. 3	0.02	1.0	~~0.51~~	0.0392

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

6. USING LARGE DATA SETS.

6.1. Data for M.L.

Designing a high accuracy learning stm:

Algorithms:
- Perceptron (Logistic Regression)
- Winnow
- Memory-based
- Naive Bayes.

Large data rationale: Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient info to predict y.

$\Rightarrow$ Use a learn. alg. with many paramtrs (hidden units)
 $\rightarrow$ low bias algorithms. $J_{train}(\theta)$ will be small.

$\Rightarrow$ Use a very large training set
 $\rightarrow$ low variance

$J_{train}(\theta) \approx J_{test}(\theta)$ will be small

WEEK # 7
Support Vector Machines.

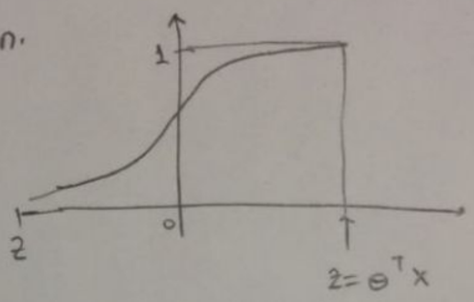
1. LARGE MARGIN CLASSIFICATN.

1.1. Obtimizatn Objective

SVM gives a cleaner & powerful way of learning complex non-linear fxns.

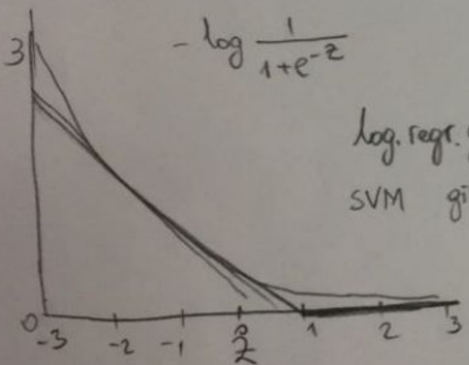
Alternative view of logistic regression.

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



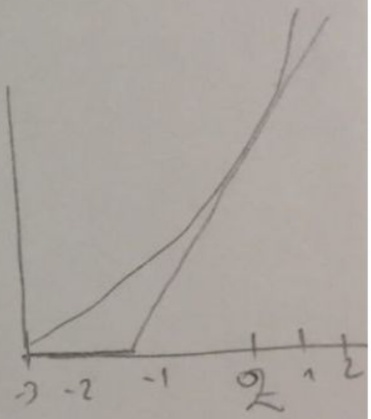
Cost of example: $-\left(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x))\right)$

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



log. regr. gives cost of zero when $z \geq 3$

SVM gives cost of zero when $z \geq 1$



Logistic Regression:

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

SVM:

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

Log. Regr: $A + \lambda B$

SVM: $CA + B$: $C = \frac{1}{\lambda}$

$\downarrow$ small

$$\min_\theta C \sum_{i=1}^{m}\left[y^{(i)} \; \dots \; \right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

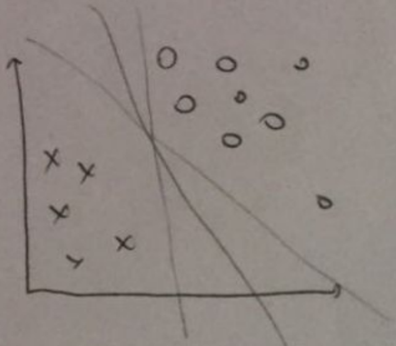
Scanned by CamScanner

1.2. Large Margin Classifiers:

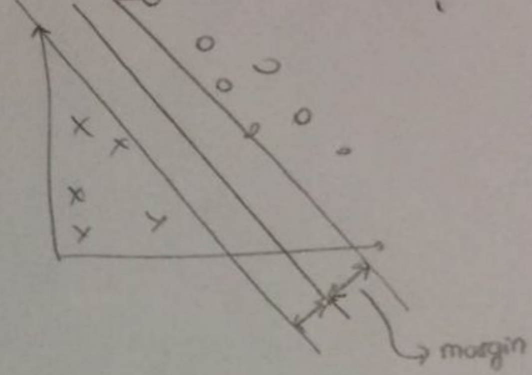
if $y=1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

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

when $C = 1000$ (very large) $\qquad \min_\theta C \sum_{i=1}^{m} \underbrace{[\ \ldots\ldots\ ]}_{=0} + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$



several decision boundaries exist



if C is very large SVM will be sensitive to outliers.

if C is not too large, then SVM will be robust against outliers.

$\Rightarrow$ if the data are not linearly separable $\Rightarrow$ SVM also do the right thing.

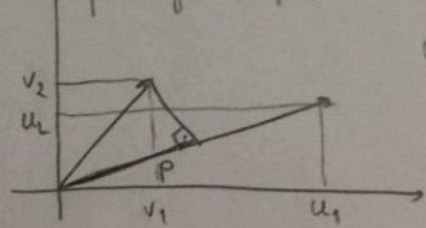
1.3. Mathematical Behind Large Margin Classificatn:

Vector Inner Product

$u^T v$

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

p= length of projectn of v onto u.



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

$= u_1 \cdot v_1 + u_2 \cdot v_2$

$\min_\theta \frac{1}{2} \sum \theta_j^2 = \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\Big(\underbrace{\sqrt{\theta_1^2 + \theta_2^2}}_{\|\theta\|}\Big)^2 = \frac{1}{2}\|\theta\|^2 \longrightarrow$ we want minimize $\|\theta\|$, thus we are going to maximize $p^{(i)}$

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

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

$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\| = \theta_1 \cdot x_1^{(i)} + \theta_2 x_2^{(i)}$

$u^T v$

stands for margin.

② KERNELS

2.1. Kernel 1:

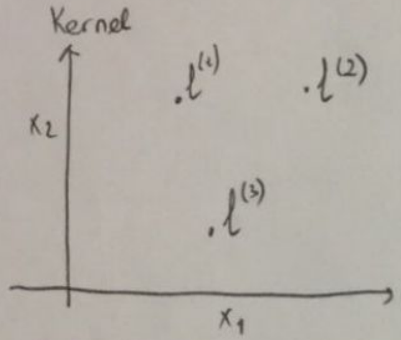
Non-linear decision boundary.

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

$$\theta_1 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \theta_5 f_5 + \dots$$

Is there a better choice of features $f_1, f_2, f_3 \dots$?

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



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

Gaussian Kernel.

Kernels & Similarity:

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

if x is far from $l^{(1)}$: $f_1 = \exp\left(-\frac{(\text{large } \#)^2}{2\sigma^2}\right) \approx 0$

2.2. Kernel 2:

Choosing the landmarks:
→ put the m-landmarks to the places where the training examples exactly are.

SVM with Kernels:

$$\sum \theta_j^2 = \theta^T \theta = \|\theta\|^2$$

$C = \left(\frac{1}{\lambda}\right)$ Large C: Lower Bias, high variance
Small C: Higher Bias, lower variance

σ^2: Large σ^2: features f_i vary more smoothly → higher bias, lower variance
Small σ^2: " " " less " → lower bias, higher variance

3. SVMs in Practice

3.1. Using an SVM: [liblinear, libsvm, ...]

→ Choice of paramtr C

→ Choice of kernel (similarity fxn) ex/ no kernel ("linear kernel")

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots \geq 0$

n large, m small, do not try nonlinear decision boundary.

→ Choice of σ^2

Other choices of kernel:

- Polynomial kernel $k(x, l) = (x^T l)^2$
- More esoteric: string kernel, chi-square kernel, histogram intersectn kernel

---

if n is large (relative to m) → use logist. regr., or SVM without a kernel ("linear kernel")

if n is small, m is intermediate → use SVM with Gaussian kernel

if n is small, m is large → create/add more features, then use logist. regr. or SVM without kernel.

WEEK #8
Unsupervised Learning

1. CLUSTERING

1.1. Unsupervised Learning: Introduction:

Algorithm finds some structure in the data for us.

ex/ market segmentatn
social network analysis
organize computing clusters
Astronomical data analysis

1.2. K-means Algorithm:

In clustering problem we are given an unlabeled dataset.

K-means algorithm:
Input: - K (# of clusters)
- Training set

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots \mu_K$
Repeat {
for $i=1$ to m $\quad c^{(i)} = \min_k \|x^{(i)} - \mu_k\|^2$
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
for $k=1$ to K
μ_k := average (mean) of pnts assigned to cluster
}

1.3. Optimizatn Objective:

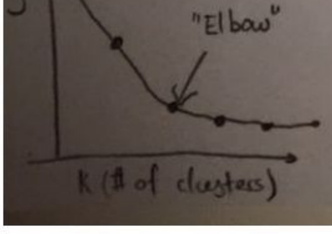
$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

distortn cost fxn.

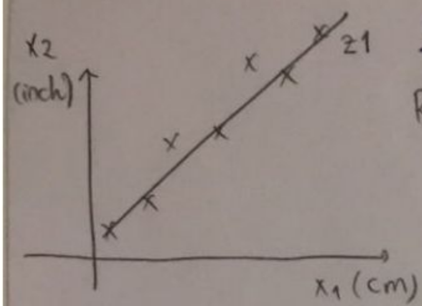
1.4. Random Initializatn:

Depending on the random initializatn K-means can end up at different solutns.
Initialize K-means lots of time, ⇒ Pick clustering that gave lowest cost $J(c^{(1)}, \dots c^{(m)}, \mu_1, \dots \mu_K)$
valid for $K = 2-10$; if $K \geq 100$ the 1st initializatn gives similar (optimal) solutn.

1.5. Choosing # of Clusters:

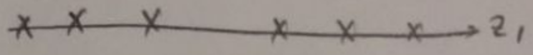


2. MOTIVATION.



2.1. Data Compression:

Reduce data from 2D to 1D.



2.2. Visualizatn:

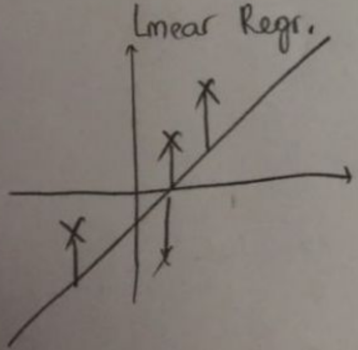
3. PCA - Principal Component Analysis.

3.1. PCA Problem formulatn:

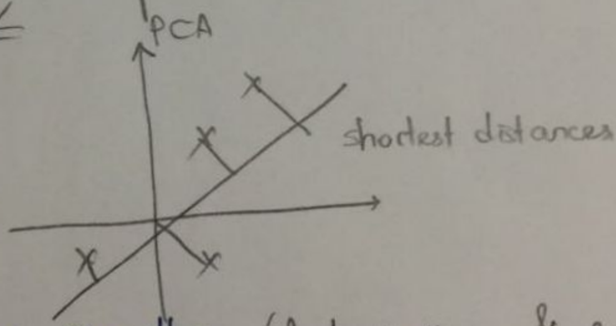
PCA tries to find lower dimensional surface onto which to project data so that sum of squares of distances btw the pnts and surface are minimized. (projectn error).

→ mean normalizatn feature scaling.
→ Apply PCA

project onto linear subspace spanned by this set of k vectors $(u(1), u(2) \ldots u(k))$



$\neq$



3.2. PCA Algorithm: (find $u_1, u_2 \ldots$ & $z_1, z_2 \ldots$)

Data preprocessing (feature scaling / mean normalizatn).

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

Reduce data from n-dimensions to k-dimensions:

1) Compute Covariance mtx. $\Sigma = \frac{1}{m}\sum_{i=1}^{n} (x^{(i)}) \cdot (x^{(i)})^T$

2) Compute "Eigenvectors" of mtx Σ. [U,S,V] = svd (Sigma);
Ureduce = U(:, 1:k);
z = Ureduce '* x;

4. APPLYING PCA.

4.1. Reconstructn from Compressed Representatn:

$$\underbrace{X_{approx}}_{n\times 1} = \underbrace{U_{reduce}}_{n\times k} \cdot \underbrace{z}_{k\times 1}$$

4.2. Choosing the # of Principal Components:

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

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

$\frac{A}{B} \leq 0.01$ (1%) "99% of variance is retained".

0.05 (5%) "95% " " " "

Try PCA with $k=1$

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

Check if $\frac{A}{B} \leq 0.01$

else

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

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

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

iterate until satisfies.

4.3. Advice for Applying PCA:

PCA is used to speed up Supervised Learning.

$x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$

$\downarrow$ PCA

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

Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training data. This mapping can be applied to $x_{cv}^{(i)}$ & $x_{test}^{(i)}$.

Applicatn of PCA:

- Compression: 1) Reduce memory/disk needed to store data
  2) Speed up learning algorithm.
- Visualizatn:

WEEK #9
Anomaly Detection.

① DENSITY ESTIMATION.

1.1. Problem Motivatn:

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

Is x_{test} anomalous?

Model $P(x)$ from data

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

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

1.2. Gaussian Distributn:

Gaussian (Normal) Distributn

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

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

1.3. Algorithm:

Density Estimatn:

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

1. Choose features x_i that you think might be indicative of anomalous examples.
2. Fit paramtrs $\mu_1, \mu_2, \ldots, \mu_n, \sigma_1^2, \ldots \sigma_n^2$

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

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

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

Anomaly if $p(x) < \varepsilon$

② BUILDING AN ANOMALY DETECTN SYSTEM.

2.1. Developing and Evaluatng an Anomaly Detectn Stm:

1000 good (normal) engines
20 flawed engines (anomalous)

Training set: 6000 good engines
CV: 2000 good engines $(y=0)$, 10 anomalous $(y=1)$
Test: 2000 " " $(y=0)$, 10 anomalous $(y=1)$

Fit model $p(x)$ on training set
On a CV, Test set x, predict

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

Possible evaluatn metrics:
- Precision / Recall
- F_1-Score

Use CV set to choose parameter ε

2.2. Anomaly Detectn vs. Supervised Learning:

Anomaly detectn	vs.	Supervised Learning:
very small # of pos've examples ($y=1$) large # of neg've examples ($y=0$)		Large # of pos.ve & neg've examples.
ex/ Fraud detectn Manufacturing (aircraft engines) Monitoring machines in a data center		ex/ Email spam classificatn. Weather predictn

2.3. Choosing what features to Use:

Non-gaussian features:

$$\left.\begin{array}{l} x_1 \leftarrow \log(x_1) \\ x_2 \leftarrow \log(x_2+C) \\ x_3 \leftarrow \sqrt{x_3} \\ x_4 \leftarrow x_4^{1/3} \end{array}\right\} \text{Gaussian features.}$$

Error Analysis for anomaly detectn:

Want $p(x) \uparrow$ for normal

$p(x) \downarrow$ for anomalous

Most common problem: $p(x)$ is comparable (both large/small) for normal & anomalous exs.

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

x_1, x_2, x_3, x_4 exist, create $x_5 = \frac{\text{CPU load}}{\text{network traffic}}$, $x_6 = \frac{x_3^2}{x_4}$

3. MULTIVARIATE GAUSSIAN DISTRIBUTION.

3.1. Multivariate Gaussian Distributn:

In some cases, actual anomalous examples can be seen as normal.

→ Use modified anomaly detectn algorithm: Multivariate Gaussian distributn.

Don't model $p(x_1), p(x_2), \ldots$ etc separately! Model $p(x)$ all in one go.

Paramtrs $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance mtx).

3.2. Anomaly Detectn using Multivariate Gaus. Distrn:

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

ORIGINAL MODEL:		MULTIVARIATE GAUSSIAN:
$p(x) = p(x_1;\mu_1,\sigma_1^2) \times \ldots p(x_n;\mu_n,\sigma_n^2)$	$\longleftrightarrow$	$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\lvert\Sigma\rvert^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$
* manually create features to capture anomalies where x_1, x_2 take unusual combinatns of values. * cheaper * OK for small m values.		* Automatically captures correlatns btw features. * more expensive * Must have $m > n$.