

Machine Learning Short Notes

Follow Sandhya Mishra



WEEK #1

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

WEEK #4

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

WEEK #7

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

WEEK #10

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

WEEK #2

- 1) Environment Setup Instr.
 - 2) Multivariate Linear Regr.
 - 3) Computing Parantis Analtc.
 - 4) Submitting Progr. Assignts.
 - 5) Review Quiz
 - 6) Octave/MATLAB Tutorial
 - 7) Review Quiz
- Programming Assignment

WEEK #3

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

WEEK #5

- 1) Cost fcn & Backpropagtn
- 2) Backpropagtn in practice
- 3) Applicatn of NN
- 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 #11

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

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 #9

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

① What is Machine Learning

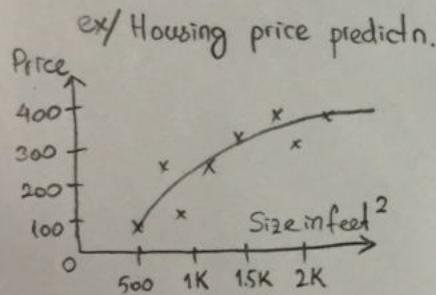
- Science of getting computers to learn without being explicitly programmed.
- NN mimics the human brain.

② INTRODUCTION

2.1. Welcome

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

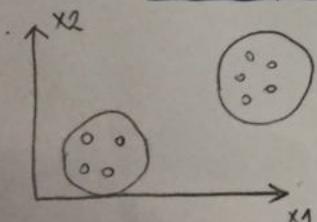
2.2. Supervised Learning.



ex/ Breast cancer
(malignant, benign)

- ⇒ Supervised Learning: "right answers" given
- ⇒ Regression: Predict cont's valued outputs
- ⇒ Classification: Discrete 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.

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

Cocktail Party Problem Algorithm:

$[W, S, V] = \text{svd}((\text{repmat}(\text{sum}(X, 1), \text{size}(X, 1), 1) . * X) . * X');$

③ QUIZ #1 (100% ✓)

④ OTHER MATERIALS:

The Course Wiki

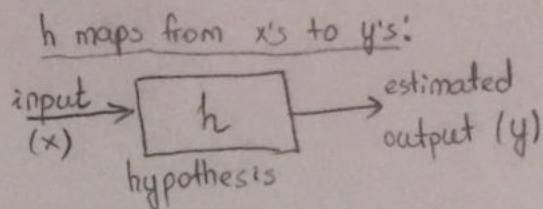
5. MODEL & COST FUNCTION

5.1. Model Representation:

m = # of training examples

x 's = "input" variable/features

y 's = "output" / "target" variable.



$$h_{\theta}(x) = \theta_0 + \theta_1 x \quad (\text{univariate linear regression})$$

5.2. Cost Function:

θ_0, θ_1 : parameters.

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

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

(squared error func.n)

Cost func.

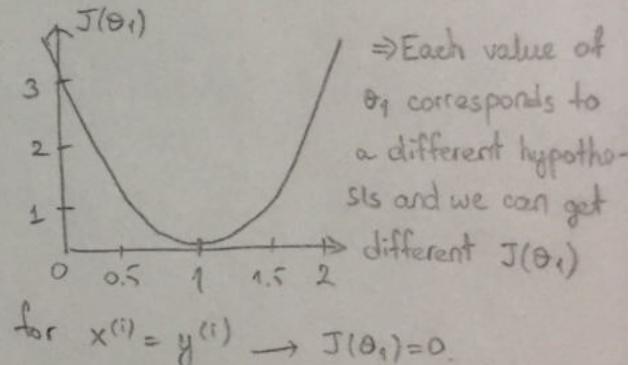
5.3. Cost Func'n - Intuition:

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

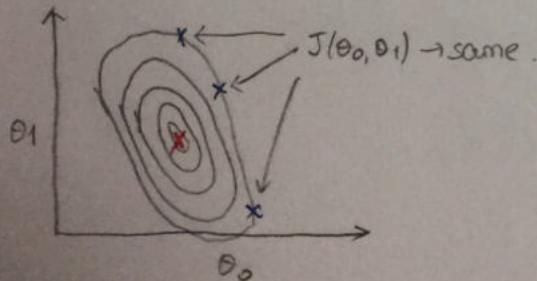
Params: θ_0, θ_1

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

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



5.4. Cost Func'n - Intuitn 2:



6. PARAMETER LEARNING

6.1. Gradient Descent:

- for minimizing some arbitrary fn J. ✓ Have some fn J(θ_0, θ_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 pt on the hill, look around and find that the best dirn is to take a little step downhill is roughly that dirn.

Gradient Descent Algorithm:

repeat until convergence {

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

}

learning rate (step size)

⇒ simultaneous update:

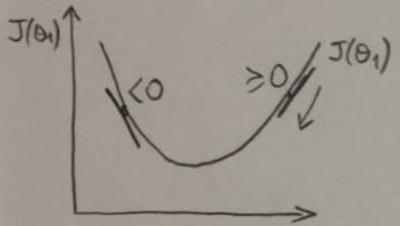
$$\text{tempo} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{tempo}$$

$$\theta_1 := \text{temp1}.$$

6.2. Gradient descent 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

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

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:

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

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

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

"Batch" Gradient descend: Each step of gradient descend uses all 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.

(2) MULTIVARIATE LINEAR REGRESSION

2.1. Multiple features:

ex/ size x_1 } $x_j^{(i)}$ = value of feature j in i^{th} training example.
 2) # of bedrooms x_2 } Price y $X^{(i)}$ = input (features) of i^{th} training example.
 3) # of floors x_3
 4) Age of home x_4

$$\text{h}_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

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" dimensional 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 {

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

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

for $n=2$:

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

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

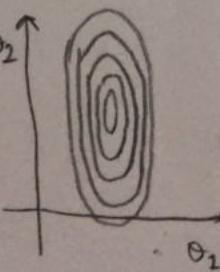
$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

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/ $x_1 = \text{size (0-2000 feet}^2)$
 $x_2 = \# \text{ of bedrooms (1-5)}$

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

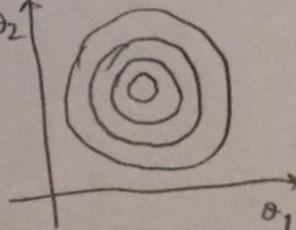


$$x_1 = \frac{\text{size (feet}^2)}{2000}$$

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

$$0 \leq x_1, x_2 \leq 1$$

Can converge much faster.



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

$$0 \leq x_1 \leq 3 \checkmark$$

$$-2 \leq x_2 \leq 0.5 \checkmark$$

$$-100 \leq x_3 \leq 100 \times$$

$$-0.0001 \leq x_4 \leq 0.0001 \times$$

(5)

Mean Normalization:

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

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

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

$$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$$

avg. value of x_1 in training set
range (or standard deviation)
(max-min)

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

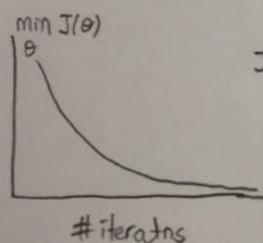
2.4. Gradient Descent in Practice 2 - Learning Rate:

Gradient descent

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

"Debugging": how to make sure grad. desc. is working correctly

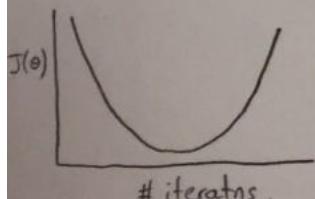
- How to choose learning rate α .



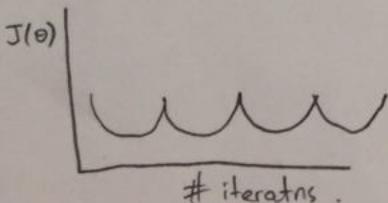
$J(\theta)$ should decrease after every iteration

Example automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than ϵ in one iter'n.



if grad. desc. does not work \Rightarrow use smaller α ,



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

\Rightarrow 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, ...

(6)

2.5. Features and Polynomial Regression:

$$h_{\theta}(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2} \Rightarrow \text{You can create new features by yourself.}$$

$$h_{\theta}(x) = \theta_0 + \theta_1 \times \underbrace{x_1}_{\text{land area } (x_1 \cdot x_2)}$$

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

3. COMPUTING PARAMETERS ANALYTICALLY.

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

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

$$X = \begin{bmatrix} -(x^{(1)})^T \\ -(x^{(2)})^T \\ \vdots \\ -(x^{(m)})^T \end{bmatrix} \quad (m \times (n+1))$$

$$y = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad (m \times 1)$$

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

$$\text{pinv}(X^T X)^{-1} X^T y$$

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

m training example, n features:

Gradient descent:

- Need to choose α
- Needs many iteratns.
- Works well even when n is large (10k)

Normal Eqn:

- No need to choose α
- Don't need to iterate.
- 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 regularization

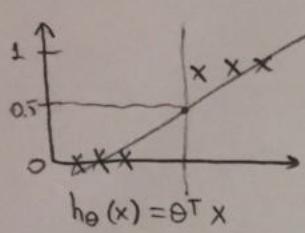
- use more training examples.

④ Programming Assignment (✓)

① CLASSIFICATION & REPRESENTATION.

1.1. Classification:

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



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 classification problem often is not a great idea

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

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

1.2. Hypothesis Representation:

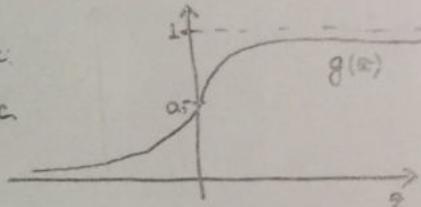
Logistic Regression Model

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

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

$$g(z) = \frac{1}{1 + e^{-z}}$$

sigmoid func.
logistic func.



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

Interpretation of hypothesis output:

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

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

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

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

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

1.3. Decision Boundary:

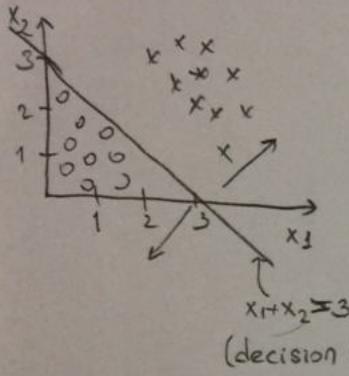
(9)

$h_{\theta}(x) = g(\theta^T x)$ → Try to understand better when this hypothesis will make predictions that $y=1$ vs. when it might make predictions that $y=0$.
 $g(z) = \frac{1}{1+e^{-z}}$
 → Understand better what hypothesis fn looks like particularly when we have more than one feature.

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

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

Decision Boundary: (2 features)



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

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

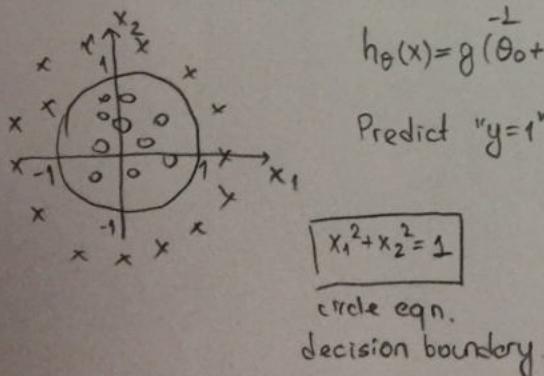
$$x_1 + x_2 \geq 3 \rightarrow y=1$$

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

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

Once we have particular values for the params $\theta_0, \theta_1, \theta_2 \Rightarrow$ 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:



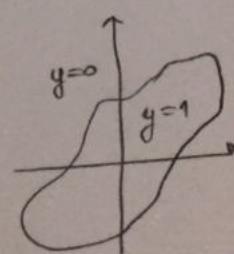
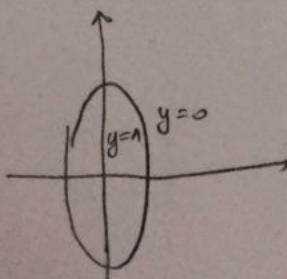
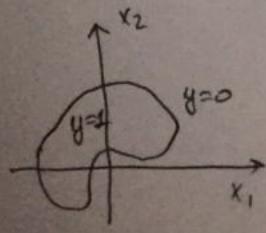
$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

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

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

⇒ 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

(10)

2.1. Cost Function

Training set - m examples

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

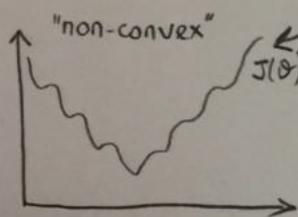
How to choose params θ ?

Cost fn:

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

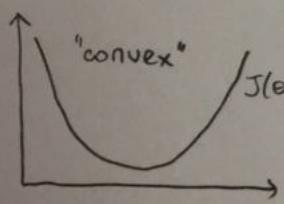
$$\text{cost}(h_{\theta}(x), y) = \frac{1}{2} (h_{\theta}(x) - y)^2$$

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



"non-convex" $J(\theta)$ → if you run grad. desc. on this fn, it is not guaranteed to converge to the global minimum.

"LOGISTIC REGRESSION" — $J(\theta)$ — nonconvex fn
if we define a square cost fn,

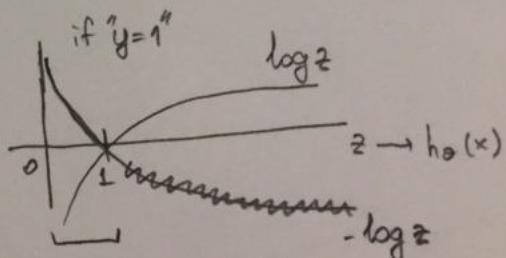


"LINEAR REGRESSION"

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

Logistic Regression Cost fn:

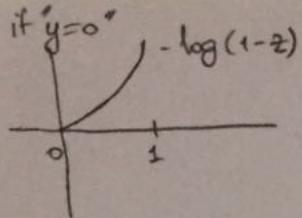
$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y=1 \\ -\log(1-h_{\theta}(x)) & \text{if } y=0 \end{cases}$$



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

as $h_{\theta}(x) \rightarrow 0$, Cost $\rightarrow \infty$

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:

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

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

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

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

To fit params θ : $\min_{\theta} J(\theta)$

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

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

Repeat {

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

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

looks like identical to linear regression!

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

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

\Rightarrow 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 {

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

}

Optimization algorithms:

→ Gradient Descent

→ Conjugate Gradient

→ BFGS

→ L-BFGS

Pros:

Cons:

- No need to manually pick α
- has clever inner-loop: line-search algorithm automatically tries out different α & picks α .
- Often faster than grad.desc. (converge much faster).

- more complex

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

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

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

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

function [jVal, gradient] = costFunction(theta)

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

$$\text{gradient} = \text{zeros}(2, 1) \rightarrow$$

$$\text{gradient}(1) = 2(\theta_1 - 5) \rightarrow \frac{\partial}{\partial \theta_1} J(\theta)$$

$$\text{gradient}(2) = 2(\theta_2 - 5) \rightarrow \frac{\partial}{\partial \theta_2} J(\theta)$$

options = optimset('GradObj', 'on', 'MaxIter', '100');

initialTheta = zeros(2, 1);

[optTheta, functionVal, exitFlag, ...] =

= fminunc(@costFunction, initialTheta, options);

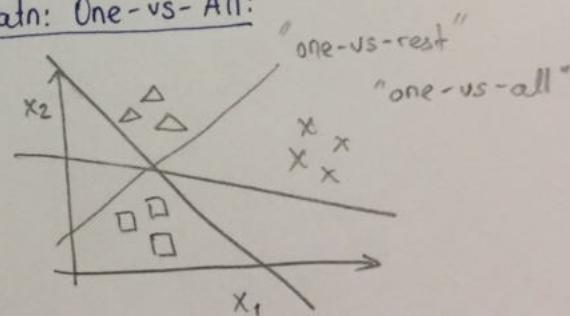
[fminunc = func.minimization
unconstrained]

③ MULTICLASS CLASSIFICATION.

3.1. Multiclass Classification: One-vs-All:

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

Weather: Sunny, Cloudy, Rain, Snow



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

↓
3 separate classification problem.

⇒ 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% ✓)

1. MOTIVATIONS.

1.1. Non-linear hypothesis:

Non-linear classification:

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 \cdot x_2^2 + \dots)$$

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 cubic 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 ≈ 3 million, features. if you try to classify imgs only by looking the pxl values, probably you will overfit! cannot classify imgs.

1.2. Neurons and the Brain:

NNs: Algorithms that try to mimic the brain.

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

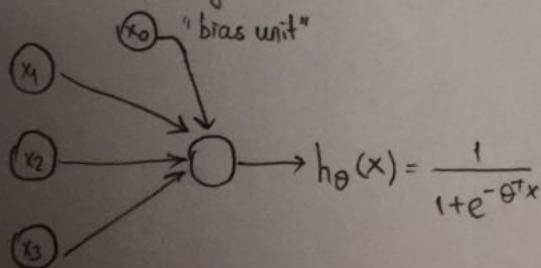
2. NEURAL NETWORKS

2.1. Model Representation - 1:

Dendrite — "input wires"

Axon — "output wire"

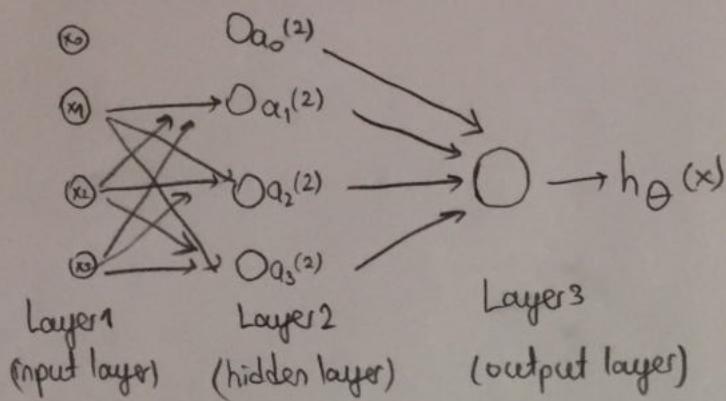
Neural model: Logistic unit



Sigmoid (logistic) activation fn

Weights = θ '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 fn 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 $\Theta^{(j)}$ dim will be $s_{j+1} \times (s_j + 1)$

2.2. Model Representatn - 2:

Forward Propagatn: Vectorized implementatn.

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

$$a^{(2)} = g(z^{(2)}) \quad \downarrow \mathbb{R}^3 \quad \downarrow \mathbb{T}\mathbb{R}^3$$

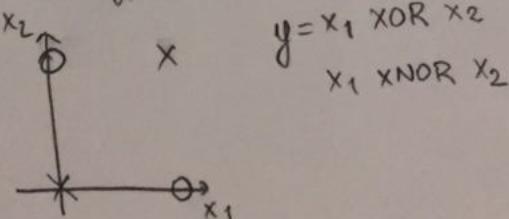
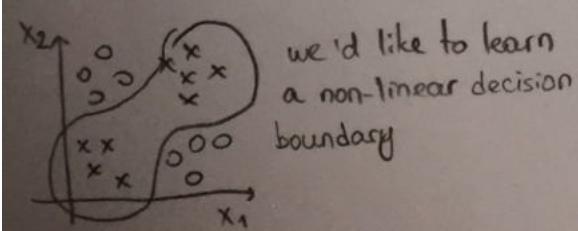


Instead of being contrained to feed the features (x_1, x_2, x_3) to logistic regression, it gets to learn its own features (a_1, a_2, a_3) to feed into logistic regression.

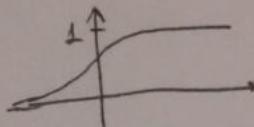
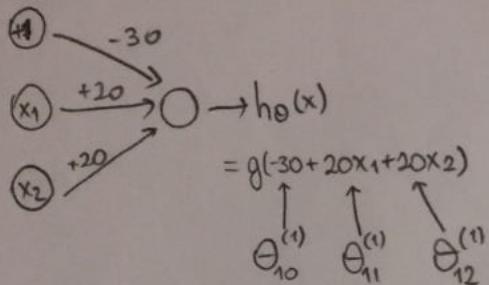
⇒ it can learn some pretty interesting and complex features & can end up with better hypothesis

③ APPLICATIONS.

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



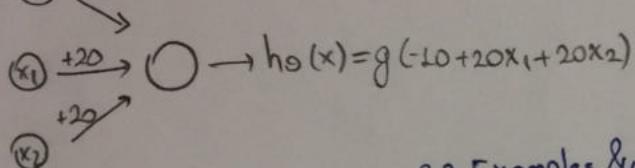
AND $y = x_1 \text{ AND } x_2$



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	$g(10) \approx 1$

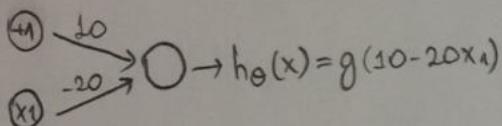
$h_\theta(x) \approx x_1 \text{ AND } x_2$

OR:

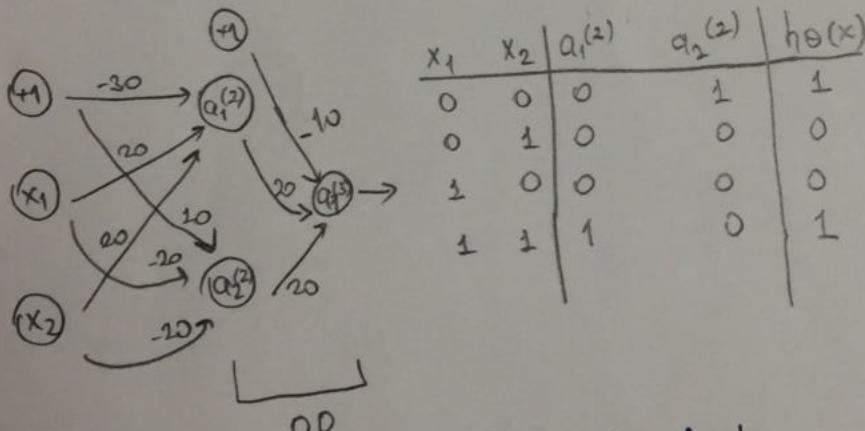


3.2. Examples & Intuitions - 2:

NOT:



XNOR:

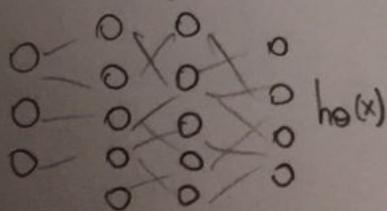


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

3.3. Multiclass Classification:

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 $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$.

① COST FUNCTION AND BACKPROPAGATION.

1.1. Cost Function:

NN (Classification)

$L = \text{Total \# of layers in network.}$

$s_l = \# \text{ of units in layer } l \text{ (excluding bias unit)}$

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

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

Cost Func:

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

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. Backpropagation Algorithm:

$$\min_{\theta} J(\theta)$$

θ

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

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

Gradient Computation:

$(x, y) \leftarrow \text{training example.}$

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

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

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

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

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

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

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

Backpropagation Algorithm:

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

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

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

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} \cdot \underbrace{g'(z^{(2)})}_{a^{(3)} \cdot (1-a^{(3)})}$$

$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} \cdot \underbrace{g'(z^{(2)})}_{a^{(2)} \cdot (1-a^{(2)})}$$

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

For the huge # of training set.

19

$$\Delta_{ij}^{(l)} = 0 \quad (\text{for } \forall i, j, l) \quad (\text{used to compute } \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta))$$

for $i=1$ to m

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

Perform forward propagation to compute $a^{(l)}$ for $l=2, 3, \dots, L$
using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

$$\text{compute } \delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)} \quad \cancel{\delta^{(1)}}$$

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

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

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

end.

1.3. Backpropagation Intuition:

$$z_1^{(2)} = \theta_{10}^{(2)} + \theta_{11}^{(2)} \cdot a_1^{(2)} + \theta_{12}^{(2)} \cdot a_2^{(2)}$$

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

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

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. Implementation Note: Unrolling Parameters:

NN: ($L=4$)

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices ($\Theta_1, \Theta_2, \Theta_3$)

$D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D_1, D_2, D_3)

"Unroll" into vectors.

ex/ $S_1=10, S_2=10, S_3=1$
 $\Theta^{(1)} \in \mathbb{R}^{10 \times 11}, \Theta^{(2)} \in \mathbb{R}^{10 \times 11}, \Theta^{(3)} \in \mathbb{R}^{1 \times 1}$
 $D^{(1)} \in \mathbb{R}^{10 \times 11}, D^{(2)} \in \mathbb{R}^{10 \times 11}, D^{(3)} \in \mathbb{R}^{1 \times 1}$

$\text{thetaVec} = [\Theta_1(:); \Theta_2(:); \Theta_3(:)]$;

$DVec = [D_1(:); D_2(:); D_3(:)]$;

$\Theta_1 = \text{reshape}(\text{thetaVec}(1:110), 10, 11)$;

Learning Algorithms:

- have initial params $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$

- 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.

2.1 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 implementation.

$$\text{gradApprox} = (J(\theta + \varepsilon) - J(\theta - \varepsilon)) / (2\varepsilon)$$

for $i=1$ to n

$$\theta_p = \theta;$$

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

$$\theta_m = \theta;$$

$$\theta_m(i) = \theta_m(i) - \varepsilon;$$

$$\text{gradApprox}(i) = (J(\theta_p) - J(\theta_m)) / (2\varepsilon);$$

end

Check that

$$\text{gradApprox} \approx D\text{Vec}$$

from numerical

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 Initialization:

→ Zero initialization: $\theta_{ij}^{(l)} = 0$ for $\forall i, j, l$. for every training examples $a_1^{(2)} = a_2^{(2)}$.

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

$$\delta_1^{(2)} = \delta_2^{(2)}$$

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

⇒ After each update, parameters corresponding to inputs going into each of two hidden units are identical. $\alpha_1^{(2)} = \alpha_2^{(2)}$

⇒ $\alpha_1^{(2)} = \alpha_2^{(2)} = \alpha_3^{(2)} = \dots = \alpha_n^{(2)}$ means that all 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 $[-\varepsilon, \varepsilon]$

$$\text{Theta1} = \text{rand}(10, 11) * (2\varepsilon) - \varepsilon; \rightarrow -\varepsilon < \text{Theta1} < \varepsilon$$

$$\text{Theta2} = \text{rand}(1, 11) * (2\varepsilon) - \varepsilon; \rightarrow -\varepsilon < \text{Theta2} < \varepsilon$$

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 ($\frac{\partial}{\partial \theta_j} J(\theta)$)

for $i=1$ to m
 FP & BP using $(x^{(i)}, y^{(i)})$
 (Get activations $a^{(l)}$ & deltas $\delta^{(l)}$ for
 $\{l=2, 3, \dots, L\}$)
 $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
 end
 compute $\frac{\partial}{\partial \theta_j} J(\theta)$

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 fn. of θ .

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

! What will it 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?!

③ APPLICATION OF NN.

22

→ Autonomous 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

! What will it we apply

BRIEF based logic into NNs,
rather than CNNs.

(23)

WEEK #6
Advice for Applying M.L.

① EVALUATING A LEARNING ALGORITHM.

1.1. Deciding What to Try Next.

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

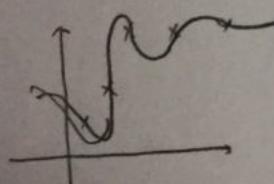
⇒ Debugging a learning algorithm.

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

- Get more training examples
- Try smaller sets of features (to prevent overfitting)
- Try getting additional features (may be current features aren't informative enough & you 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

↓ test set

(70%)

(30%)

→ Learn paramtr θ from training data

→ Compute test error:

1.3. Model Selection and Train / Validation / Test Sets:

⇒ Model Selection:

d = degree of polynomial.

$d=1 \rightarrow$ Training error ($\theta^{(1)}$) → Test error ($\theta^{(1)}$)

$d=5 \rightarrow$ Training error ($\theta^{(5)}$) → Test error ($\theta^{(5)}$)

$J_{\text{test}}(\theta^{(5)})$

$J_{\text{test}}(\theta^{(5)})$ 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%) → $J_{\text{train}}(\theta)$
 cross-validation set (20%) → $J_{\text{cv}}(\theta)$
 test set (20%) → $J_{\text{test}}(\theta)$

(24)

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

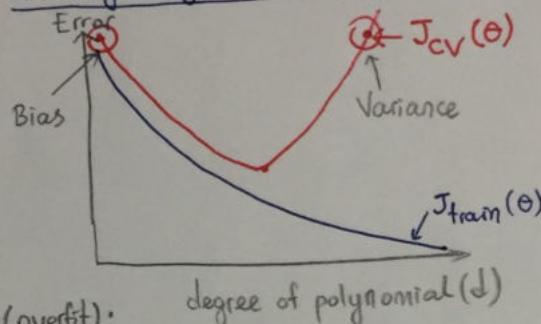
Pick the optimum hypothesis (ex/ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$)

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

② BIAS Vs. VARIANCE

2.1. Diagnosing bias Vs. variance:

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



Bias (underfit):

$J_{\text{train}}(\theta) \rightarrow$ high
 $J_{\text{cv}}(\theta) \rightarrow$ high

Variance (overfit):

$J_{\text{train}}(\theta) \rightarrow$ low
 $J_{\text{cv}}(\theta) \gg J_{\text{train}}(\theta)$

degree of polynomial (d)

2.2. Regularization 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

$\lambda = 10^4$, $\theta_1 \approx 0, \theta_2 \approx 0$

$h_\theta(x) \approx 0$

Intermediate λ :

Just Right

Small λ :

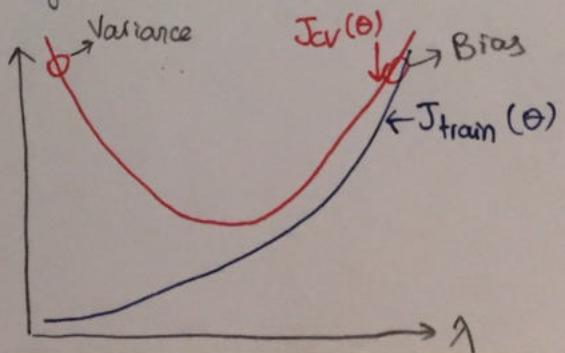
High Variance, Overfit

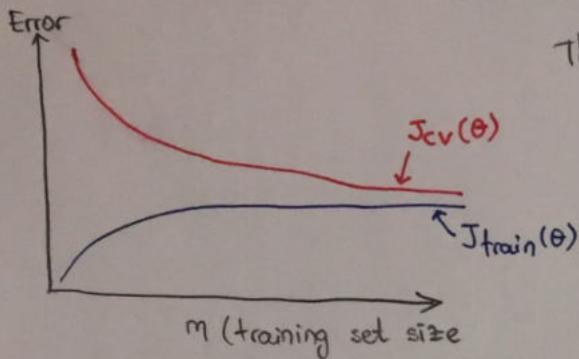
$\lambda = 0$.

⇒ Choosing the regularization error:

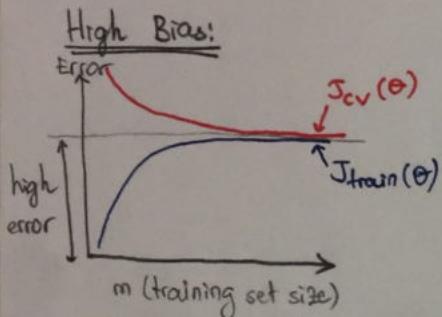
- 1) $\lambda = 0 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{\text{cv}}(\theta^{(1)})$
- 2) $\lambda = 0.01 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(2)} \rightarrow J_{\text{cv}}(\theta^{(2)})$
- 3) $\lambda = 0.02 \rightarrow \min_{\theta} J(\theta) \rightarrow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

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



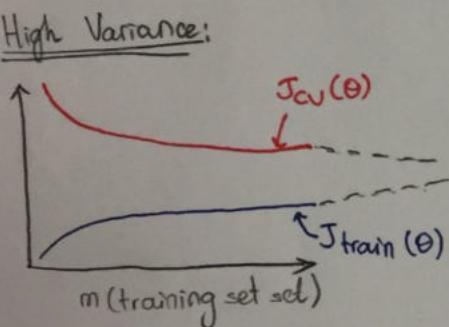
2.3. Learning Curves:

The more data you have \Rightarrow the better the hypothesis you fit.



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.



→ 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 \rightarrow fixes high variance
- Try smaller sets of features \rightarrow fixes high variance
- Try getting additnl features \rightarrow fixes high bias
- Try adding polynomial features \rightarrow fixes high bias
- Try decreasing $\lambda \rightarrow$ fixes high bias
- Try increasing $\lambda \rightarrow$ fixes high variance

Small NN

- computationally cheaper
- prone to underfitting

Large NN

- more expensive
- prone to overfitting
- Use regulztn \uparrow
- Select # of hidden layers.

(3.) REVIEW: QUIZ + P.A.

④ BUILDING A SPAM CLASSIFIER.

(26)

4.1. Prioritizing What to Work On:

1) $x = \text{features of email}$, $y = 1 (\text{spam}) / 0 (\text{non-spam})$

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 (medicine)

4.2. Error Analysis:

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

ex/ 100 errors on categorization 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 $y = \text{predictCancer}(x)$
 End $y=0$; → 0.5% error
 gives

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

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

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

(5.) HANDLING SKEWED DATA.

(27)

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.

Predict 1 if $h_\theta(x) \geq 0.7$ } → Higher precision.
 Predict 0 if $h_\theta(x) < 0.7$ } → Lower recall.

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

1 if $h_\theta(x) \geq 0.3$ } → Higher recall
 0 if $h_\theta(x) < 0.3$ } → Lower precision.

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 strn:

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

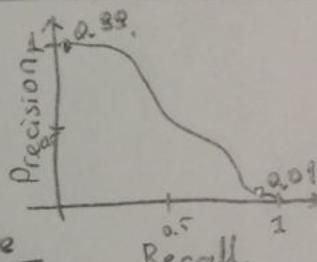
Algorithms:

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

- ⇒ Use a learn. alg. with many params (hidden units)
- low bias algorithms. $J_{\text{train}}(\theta)$ will be small.
- ⇒ Use a very large training set
- low variance

$$J_{\text{train}}(\theta) \approx J_{\text{test}}(\theta)$$

will be small



WEEK #7

Support Vector Machines.

(28)

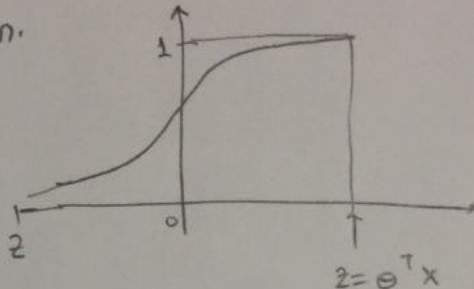
① LARGE MARGIN CLASSIFICATN.

1.1. Optimizatn Objective

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

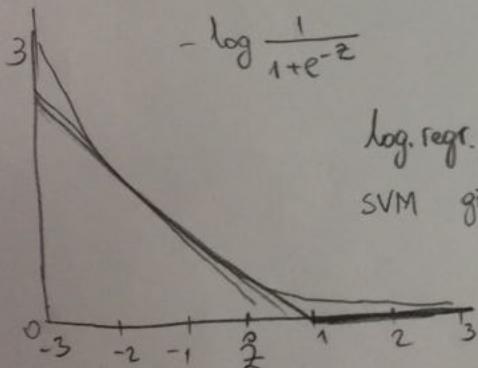
Alternative view of logistic regression.

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

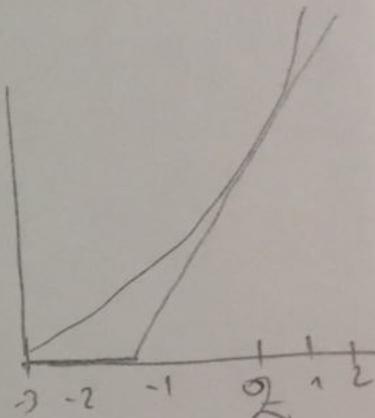


Cost of example: $-(y \log h_{\theta}(x) + (1-y) \log (1-h_{\theta}(x)))$

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



log. reg. gives cost of zero when $z \geq 3$
SVM gives cost of zero when $z \geq 1$



Logistic Regression:

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

SVM:

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

$$\text{Log. Regr: } A + \lambda B$$

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

↓
small

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

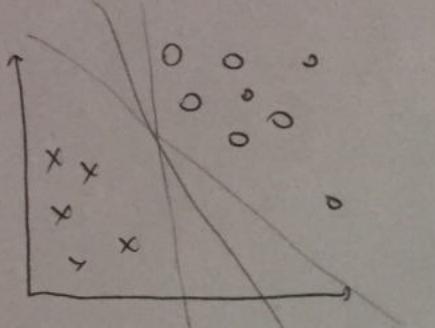
1.2. Large Margin Classifiers:

29

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

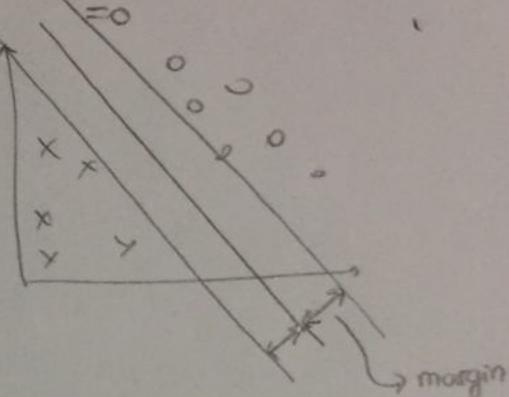
if $y=0$, we want $\theta^T x < 1$ (not just < 0)

when $C = 1000$ (very large)



several decision boundaries exist

$$\min_{\theta} C \sum_{i=1}^m [\dots] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



margin

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.

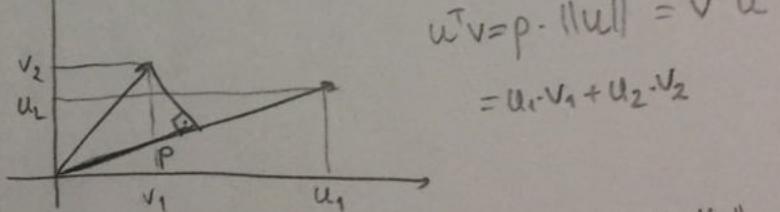
1.3. Mathematical Behind Large Margin Classification:

Vector Inner Product.

$u^T v$

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

$p = \text{length of projection of } v \text{ onto } u$.



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

\rightarrow we want minimize $\|\theta\|$, thus we are going to maximize $p^{(i)}$

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

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

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

$u^T v$

stands for margin.

② KERNELS

(30)

21. Kernel 1:

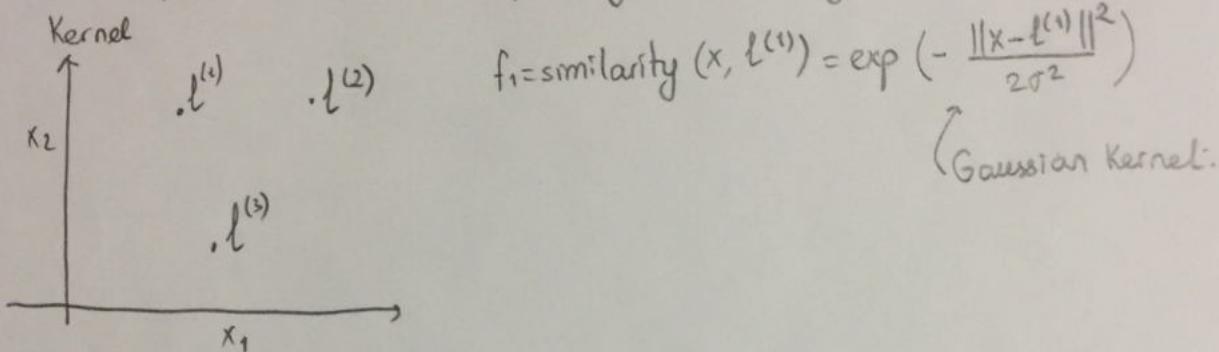
Non-linear decision boundary.

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

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

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

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



Kernels & Similarity:

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

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

22. Kernel 2:

Choosing the landmarks:

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

SVM with Kernels:

$$\sum \theta_i^2 = \theta^\top \theta = \|\theta\|^2$$

$C = \left(\frac{1}{\lambda}\right)$ 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

(31)

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

→ Choice of paramtr C

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

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

→ Choice of σ^2 n large, m small, do not try nonlinear decision boundary.

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 logst. regr. or svm without kernel.

WEEK #8

Unsupervised Learning

(32)

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 $c^{(i)} = \min_k \|x^{(i)} - \mu_k\|^2$
 $c^{(i)} :=$ index (from 1 to K) of cluster centroid
 closest to $x^{(i)}$

for $k=1$ to K
 $\mu_k :=$ average(mean) of pts assigned to cluster

}

1.3. Optimization 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 fn.

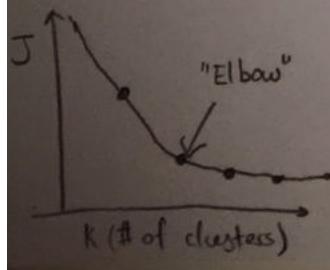
1.4. Random Initialization:

Depending on the random initialization K-means can end up at different solutns.

Initialize K-means lots of time, \Rightarrow Pick clustering that gave lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

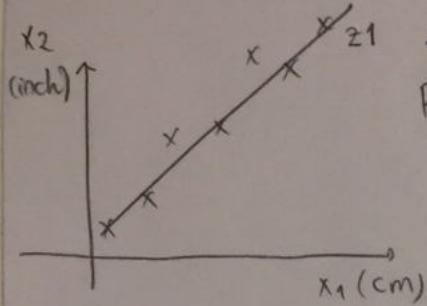
↑
valid for $K=2-10$; if $K \geq 100$ the 1st initialization gives a similar (optimal) solutn.

1.5. Choosing # of Clusters:



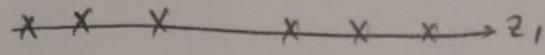
(2.) MOTIVATION.

(33)



2.1. Data Compression:

Reduce data from 2D to 1D.



2.2. Visualizatn:

(3.) PCA - Principal Component Analysis.

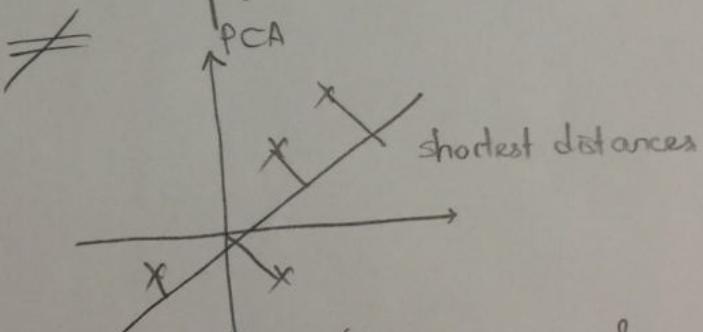
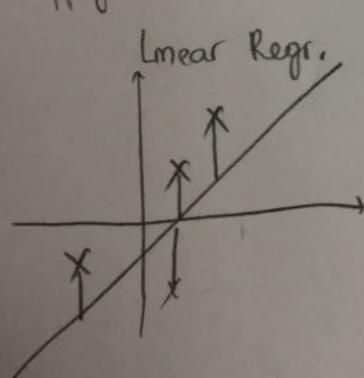
3.1. PCA Problem formulatn:

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

→ mean normalization / feature scaling.

→ Apply PCA

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



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

Data preprocessing (feature scaling / mean normalization).

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \rightarrow \quad x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

Reduce data from n-dimensions to k-dimensions:

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

2) Compute "Eigenvectors" of mtx Σ . $[U, S, V] = svd(\Sigma)$;
 $U_{\text{reduce}} = U(:, 1:k)$;
 $z = U_{\text{reduce}}^{-1} * x$;

(4) APPLYING PCA.

(34)

4.1. Reconstruction from Compressed Representation:

$$\underline{X}_{\text{approx}} = \underbrace{U_{\text{reduce}}}_{n \times k} \cdot \underbrace{\Sigma}_{k \times 1}$$

4.2. Choosing the # of Principal Components:

$$\text{Average squared projtn error: } \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \underline{x}_{\text{approx}}^{(i)}\|^2 \rightarrow A$$

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

$\frac{A}{B} \leq 0.01$ (1%)	"99% of variance is retained".
0.05 (5%)	"85% "

Try PCA with $k=1$

Compute $U_{\text{reduce}}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, \underline{x}_{\text{approx}}^{(1)}, \dots, \underline{x}_{\text{approx}}^{(m)}$

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

else

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

$$S = \begin{bmatrix} s_{11} & & & \\ s_{21} & s_{22} & & \\ \vdots & \ddots & \ddots & \\ 0 & 0 & \ddots & s_{nn} \end{bmatrix}$$

For given k :

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

iterate until satisfies.

4.3. Advice for Applying PCA:

PCA is used to speed up Supervised Learning.

$$\left. \begin{array}{l} x^{(1)}, x^{(2)}, \dots, x^{(m)} \in \mathbb{R}^{10000} \\ \downarrow \text{PCA} \\ z^{(1)}, z^{(2)}, \dots, z^{(m)} \in \mathbb{R}^{1000} \end{array} \right\} \begin{array}{l} \text{Mapping } x^{(i)} \rightarrow z^{(i)} \text{ should be defined} \\ \text{by running PCA only on the training data.} \\ \text{This mapping can be applied to } x_{\text{cv}}^{(i)} \text{ &} \\ x_{\text{test}}^{(i)}. \end{array}$$

Application of PCA:

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

- Visualizatn:

WEEK #9

Anomaly Detection.

(35)

① DENSITY ESTIMATION.

1.1. Problem Motivation:

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

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 Distribution:

Gaussian (Normal) Distribution

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

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

1.3. Algorithm:

Density Estimation:

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

1. Choose features x_i that you think might be indicative of anomalous examples.

2. Fit parameters $\mu_1, \mu_2, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

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

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

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

Anomaly if $p(x) < \varepsilon$

② BUILDING AN ANOMALY DETECTION SYSTEM.

2.1. Developing and Evaluating an Anomaly Detection System:

1000 good (normal) engines Training set: 6000 good engines
20 flawed engines (anomalous) 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
 $y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$

Possible evaluation metrics:
-Precision / Recall
-F1-Score

Use CV set to choose parameter ε .

2.2. Anomaly Detectn vs. Supervised Learning:

(36)

Anomaly detectn

vs.

Supervised Learning:

very small # of pos've examples ($y=1$)
large # of neg've examples ($y=0$)
ex/Fraud detectn

Large # of pos.ve & neg've examples.

Manufacturing (aircraft engines)

ex/Email spam classifictn.

Monitoring machines in a data center

Weather predictn

2.3. Choosing what features to Use:

Non-gaussian features:

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

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 $p(x_1), p(x_2), \dots$ etc separately! Model $p(x)$ all in one go.

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

3.2. Anomaly Detectn using Multivariate Gaus. Distrn:

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

ORIGINAL MODEL:

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

MULTIVARIATE GAUSSIAN:

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

* manually create features to capture anomalies where x_1, x_2 take unusual combinatns of values.
* OK for small m values.

* Automatically captures correlations btw features.

* more expensive

* Must have $m > n$.