Linear Regression with One Variable

Model Representation :

regression : predict real-valued output → predict house prices

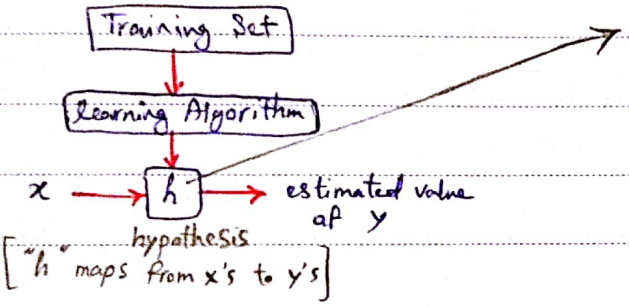
classification : classify into discrete-valued output → tumor classification (malignant or benign)

رگرسیون یک الگوریتم با نظارت (Supervised) است؛ زیرا برای تمام داده‌ها خروجی درست (لیبل) را از قبل داریم.

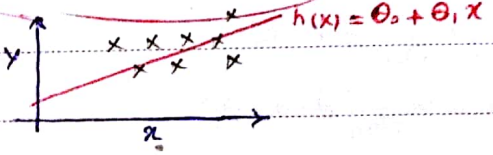
training set:
- m = number of examples
- x's = input variable / features
- y's = output variable / target

(x, y) = one training example

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



$h(x) = \theta_0 + \theta_1 x$



univariate linear regression (linear reg. with 1 variable)

چرا تابع رگرسیون را خطی فرض کردیم؟ برای راحتی کار (در شروع)

Cost Function :

Find best θ_0 and θ_1 that minimize $\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = J(\theta_0, \theta_1)$ ← Cost Func. (Squared error function)

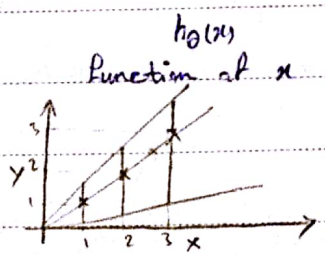
$h_\theta(x^{(i)}) = \hat{y}_i$: مقدار پیش‌بینی شده — $y^{(i)}$: مقدار واقعی

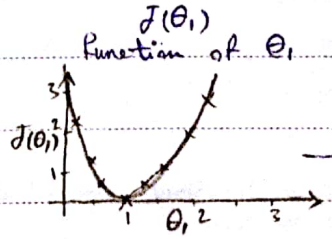
میانگین توان تابع هزینه در مسائل رگرسیون

چگونه θ ها را انتخاب کنیم؟ باید دقت شود

در تابع $h_\theta(x) = \theta_0 + \theta_1 x$

$\theta_0 = 0$ فرض :





minimize $J(\theta_1)$

اگر θ_0 را داشته باشیم، آن‌گاه $J(\theta_0, \theta_1)$ به دو متغیر بستگی دارد. پس یک نمودار سه بعدی خواهیم داشت : θ_0 , θ_1 , $J(\theta_0, \theta_1)$

این نمودار سه بعدی شبیه یک کاسه (bowl-shaped) خواهد بود که تقعر آن جایی است که تابع هزینه مینیمم می‌شود (Contour)

باید الگوریتمی داشته باشیم که بتواند به صورت اتوماتیک θ_0 , θ_1 مقدارهایی را بیابد که با آن‌ها تابع $J(\theta_0, \theta_1)$ مینیمم شود.

Gradient Descent :

الگوریتم کلی یا مینیمم محلی

الگوریتم gradient descent برای مینیمم کردن یک تابع به کار می‌رود و برای مینیمم کردن تابع هدف رگرسیون خطی هم می‌توانیم به کار برده شود.

Have Some function $J(\theta_0, \theta_1)$ → Want $\min_{\theta_0,\theta_1} J(\theta_0,\theta_1)$ → Start with Some θ_0, θ_1 → Keep changing θ_0, θ_1 to reduce $J(\theta_0,\theta_1)$ → end up at a minimum $J(\theta_0,\theta_1)$

↙ یا هر تابع دیگر با هر تعداد متغیر دیگر

Gradient descent algorithm :

repeat until convergence { $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ (for $j=0$ and $j=1$) }

assignment ↑ — derivate term ↑ — learning rate ↙

نرخ یادگیری (α) مشخص می‌کند که در هر تکرار چقدر جابه‌جا می‌شویم. اگر α بزرگ باشد قدم‌های ما به سمت نقطه مورد نظر (مثلاً مینیمم تابع) بزرگ‌تر خواهد بود و برعکس.

نکته : آپدیت شدن مقدار θ_0 و θ_1 در هر تکرار باید هم‌زمان باشد. یعنی :

$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

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

$\theta_0 := temp0$

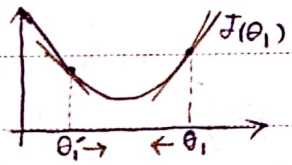
$\theta_1 := temp1$

(Simultaneous update)

اگر آپدیت شدن پارامترها هم‌زمان نباشد ؟ الگوریتم کار خواهد کرد ولی درست نیست.

Gradient Descent Intuition :

فرض $\theta_0 = 0$ و $\theta_1 \in \mathbb{R}$ و $J(\theta_1)$ به صورت مقابل باشد :



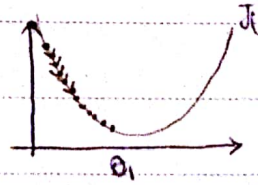
$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1}(J(\theta_1))}$ → شیب خط که در θ_1 مثبت است

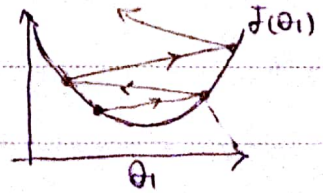
$\theta_1 := \theta_1 - \alpha$ (positive number) → پس θ_1 کوچک می‌شود

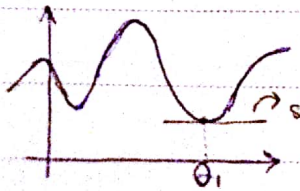
در نقطه θ_1' : $\frac{\partial}{\partial \theta_1} J(\theta_1)\Big|_{\theta_1=\theta_1'} < 0$ (شیب منفی) → $\theta_1' := \theta_1' - \alpha$ (negative number) → پس θ_1' بزرگ می‌شود

نرخ یادگیری (α)

- if α is too small, gradient descent can be slow
- if α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge!



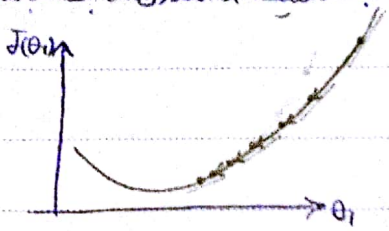




اگر θ_1 در نقطه اپتیمم محلی باشد چه اتفاقی می‌افتد ؟

$\theta_1 := \theta_1 - (\alpha . 0)$ → $\theta_1 := \theta_1$

نکته سوم : در حین انجام تکرارهای متوالی gradient descent نیازی به تغییر مقدار α نیست ! کافیست α مقداری ثابت باشد



هر چه به نقطه مینیمم موضعی نزدیک تر می‌شویم ، شیب نمودار کم تر می‌شود ، در نتیجه

ترم $\alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$ کوچک تر می‌شود !

Gradient Descent for Linear Regression :

باید ابتدا مشتق جزیی تابع J را نسبت به θ_0 , θ_1 بیابیم : $\theta_0 + \theta_1 x^{(i)}$

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

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

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

نکته : تابع هدف برای رگرسیون خطی همواره یک تابع Convex Function است و یک اپتیمم بیش‌تر ندارد . در نتیجه همگرا با gradient descent به اپتیمم سراسری می‌رسیم .

به الگوریتم مورد بحث batch gradient descent هم گفته می‌شود ؛ چرا که در هر تکرار از تمام دیتاست (training examples) استفاده می‌کنیم

روش‌های دیگری مانند Normal Equation Method هم برای محاسبه مقدار مینیمم تابع هدف وجود دارند . اما روش gradient descent در دیتاست‌های بزرگ‌تر بهتر از روش normal equation عمل می‌کند .

Linear Regresson with Multiple Variables

Multiple Features :

m = number of samples n = number of features

در رگرسیون چند متغیره ، هر رکورد تنها یک عدد نیست بلکه یک ستون از اعداد است

که هر عدد بیانگر مقدار هر ویژگی برای آن رکورد است :

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^n \quad \rightarrow \quad x_0 = 1 \quad \rightarrow \quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

تابع hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ ← بردار ویژگی‌های یک خانه

$$h_\theta(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3 x_3 - 2 x_4$$

base price ; θ_0 — price per square feet ; θ_1 — each Floor ; θ_2 — each bedroom ; θ_3 — age of the house ; θ_4

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \theta^T \cdot x = \underbrace{[\theta_0\ \theta_1 \cdots \theta_n]}_{\theta^T} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$ ← رگرسیون خطی چند متغیره :

PAPCO

Gradient Descent For Multiple Variables :

parameters: $\theta_0, \theta_1, \dots, \theta_n$ → θ : (n+1)-dimensioned vector

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

Gradient descent: ($n \geq 1$)

Repeat {

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

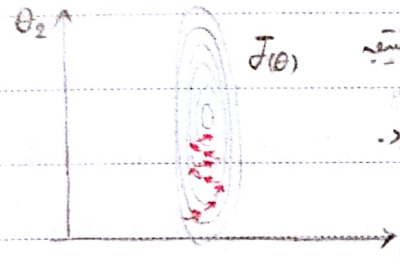
}

(simultaneously update θ_j for $j = 0, 1, \dots, n$)

Feature Scaling :

Idea : Make sure features are on a similar scale

اگر x_1 : $0 \leq x_1 \leq 2000$ و $1 \leq x_2 \leq 5$

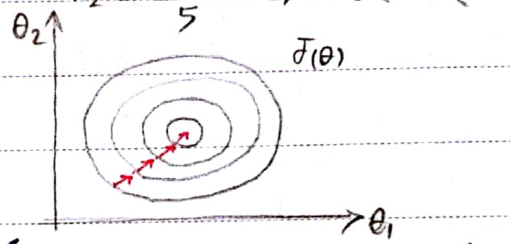


در این حالت کانتور بسیار کشیده خواهد شد، در نتیجه gradient descent الگوریتم بسیار کند و زمانبر خواهد بود.

Feature scaling →

$x_1 = \frac{x_1}{2000} \Rightarrow 0 \leq x_1 \leq 1$

$x_2 = \frac{x_2}{5} \Rightarrow 0 \leq x_2 \leq 1$



اگر دامنه تغییرات هریک از متغیرها (x) یکسان نباشد اما نزدیک باشد، مشکلی نیست. اما اگر مانند بالا دامنه تغییرات بسیار متفاوت باشد باید سعی کنیم همگی متغیرها در بازه $-1 \leq x_i \leq 1$ قرار بگیرند.

یکی از روش های کوچک کردن بازه تغییرات mean normalization است :

اگر $0 \leq x_1 \leq 2000$ → $x_1 = \frac{x_1 - 1000}{2000} \Rightarrow -0.5 \leq x_i \leq 0.5$

(μ_1) میانگین بازه

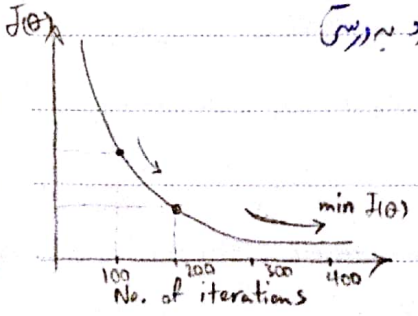
(max − min)

$x_i \leftarrow \frac{x_i - \mu_i}{s_i}$, $i \in [1, n]$

average value $\left(\frac{\max(x_i) - \min(x_i)}{2}\right)$

Standard deviation ($\max(x_i) - \min(x_i)$)

Learning Rate :



نکته مهم در استفاده از الگوریتم کاهش گرادیانی (gradient descent) این است که مطمئن شویم درست به درستی کار می‌کند (debugging) ⇐ کافیست پس از هر تکرار، مقدار تابع $J(\theta)$ کاهش یابد.

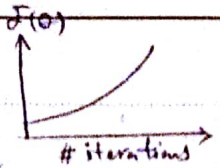
اما اگر دقت کنیم در نمودار روبرو مقدار $J(\theta)$ از تکرار ۳۰۰ به بعد تغییر خاصی نمی‌کند، پس لازم نیست تکرارها را تا ابد ادامه دهیم! این که تابع $J(\theta)$ پس از چند تکرار به یک مقدار نسبتاً ثابتی می‌رسد، بستگی به مسئله دارد.

می‌توانیم برای الگوریتم یک Convergence (همگرایی) مشخص کنیم :

Declare convergence if $J(\theta)$ decreases less than $\varepsilon = 10^{-3}$ in one iteration

اما مقدار ε می‌تواند متفاوت باشد، پس تشخیص از روی نمودار اولویت دارد.

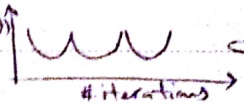
Date



باید نرخ یادگیری (α) را کوچک کنیم.

باید نرخ یادگیری را کوچک کنیم.

اگر نمودار تغییرات $J(\theta)$ با هر تکرار صعودی بود

اگر نمودار به صورت  بود باز هم

Summary:

نرخ یادگیری (α)

- بسیار کوچک: Slow convergence
- بسیار بزرگ: $J(\theta)$ may not decrease on every iterations; may not converge (Slow convergence also possible)

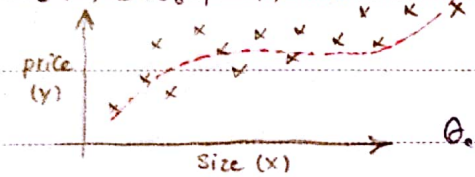
روش انتخاب درست نرخ یادگیری؟ صحیح و خطا.

Features and Polynomial Regression:

فرض کنید تابع پیش بینی قیمت خانه به صورت روبه رو باشد: $h_\theta(x) = \theta_0 + \theta_1 \times \overbrace{\text{frontage}}^{x_1} + \theta_2 \times \overbrace{\text{depth}}^{x_2}$

این تابع به دو متغیر طول و عرض خانه بستگی دارد، پس می‌توانیم این دو متغیر را تبدیل به یک متغیر مساحت کنیم: $\overbrace{X}^{\text{area}} = \text{frontage} * \text{depth}$

$h_\theta(x) = \theta_0 + \theta_1 \overbrace{X}^{\text{land area}}$



حالا فرض کنیم تابع قیمت خانه بر اساس مساحت خانه به صورت مقابل باشد:

بهترین تابع برای فیت شدن به دیتای مقابل یک تابع درجه ۳ است: $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

با توجه به این که تنها یک ویژگی برای دیتا داریم (مساحت) می‌توانیم سه متغیر بالا را این‌گونه بسازیم: $h_\theta(x) = \theta_0 + \theta_1 \underbrace{(x_1)}_{\text{(size)}} + \theta_2 \underbrace{(x_2)}_{(\text{size})^2} + \theta_3 \underbrace{(x_3)}_{(\text{size})^3}$

Feature scale برای ما بسیار مهم است.

← در دو مثال بالا دیدیم که می‌توانیم چند متغیر را ادغام کنیم یا متغیرهای جدید بر اساس قبلی‌ها بسازیم!

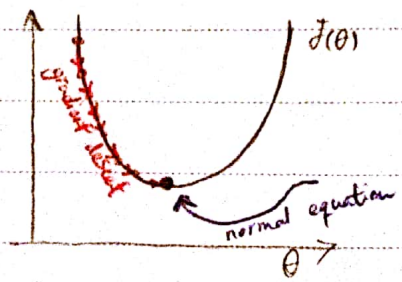
نکته: برای نمودار مصدود به تابع درجه ۳ ذکر شده نیستیم؛ ممکن است توابع دیگری مانند $h_\theta(x) = \theta_0 + \theta_1 (\text{size}) + \theta_2 \sqrt{(\text{size})}$ مناسب باشند.

روش انتخاب بهترین تابع، شکل داده‌های نمودار است: تابع انتخابی باید به بهترین شکل به دیتا fit شود.

به توابع غیرخطی که در بالا مشاهده کردیم، توابع رگرسیون چند جمله‌ای (polynomial) گفته می‌شود.

Normal Equation:

گاهی اوقات بهتر است به جای استفاده از gradient descent، تابع $J(\theta)$ را حل کنیم تا θ ی بهینه را بیابیم.



برای مثال در تابع روبه رو، اگر بخواهیم گام گرادیان گام به گام مقدار θ را تغییر می‌داد تا به θ ی بهینه برسد. درحالی که با حل یک معادله ساده می‌توانیم بهترین θ برای مینیمم شدن تابع $J(\theta)$ را بیابیم:

$J(\theta) = a\theta^2 + b\theta + c$

$\frac{\partial}{\partial \theta} J(\theta) = \dots = 0$ → Solve for θ!

Subject

Date

$m = \dots$
$n = 4$

x_0	Size	no. of bedrooms	no. of floors	age	price
1	⋮	⋮	⋮	⋮	⋮
1	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
1	⋮	⋮	⋮	⋮	⋮

$y = \begin{bmatrix} \vdots \end{bmatrix}_{m\times 1}$ ← price

$X = \begin{bmatrix} \cdots \\ \vdots \\ \cdots \end{bmatrix}_{m\times(n+1)}$ ← design matrix

فرض کنیم داده های ما به صورت روبه رو باشد.

آن گاه مقدار بهینه ماتریس θ به صورت زیر به دست می آید:

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

Octave/MATLAB: `pinv(X'*X)*X'*y` ($(X^TX)^{-1}$)

به این روش حل معادله و یافتن θ روش Normal Equation گفته می شود.

مقایسه دو روش ذکر شده برای یافتن θ:

Gradient Descent	Normal Equation
• need to chose α	• No need for α
• needs many iterations	• No need to iterate
• works well even when n is large.	• Need to Compute $(X^TX)^{-1}$ — $n\times n$ matrix $\rightarrow O(n^3)$
	• Slow if n is very large

نتیجه: برای n های کوچک روش normal equation بهتر است؛ اما اگر n بسیار بزرگ باشد (مثلاً $n > 10000$) روش کاهش گرادیان بهتر جواب خواهد داد.

Normal Equation Noninvertibility:

در معادله $\theta = (X^TX)^{-1}X^Ty$ اگر X^TX معکوس پذیر نباشد، چه اتفاقی خواهد افتاد؟

این اتفاق به ندرت رخ می دهد. اما توجه کنیم که Octave/MATLAB در عملگر معکوس کننده دارد: inv و pinv (pseudo-inverse)

دستور pinv که ما از آن استفاده می کنیم، معکوس X^TX را محاسبه می کند، حتا اگر معکوس پذیر نباشد!

★ در این جا وارد جزییات ریاضیات دستور pinv و تفاوت آن با inv نمی شویم. اما می خواهیم ببینیم که در چه صورتی ماتریس X^TX معکوس ناپذیر خواهد بود؟

۱) اگر Feature های اضافی داشته باشیم (که با یکدیگر هم بستگی خطی دارند) مثلاً: $\begin{cases} x_1 = \text{Size in } feet^2 \\ x_2 = \text{Size in } m^2 \end{cases}$

۲) اگر Feature های بسیار زیادی داشته باشیم (مثلاً $m \le n$) ← آن گاه باید برخی Feature ها را حذف کنیم یا از روش Regularization استفاده کنیم.

↳ پس اگر ماتریس X^TX معکوس ناپذیر بود باید به فکر کاهش تعداد Feature ها باشیم.

در آینده راجع به آن خواهیم گفت