machine Learning — Prof. Anderson Rocha.

(1) Grew out of work in (AI)

(2) New capabilities for computers

Some examples

(1) Database mining, large datasets, web automation, web click data, medical records, social networks

(2) Applications can't program by hand
- autonomous car, helicopter
- handwritten recognition
- NLP
- vision

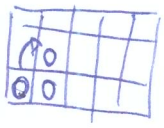
(3) Self-customizing
- Amazon
- Netflix product recommendations

(4) Understanding human learning.

What is (ML)

(1) field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel)

(2) Tom Mitchel: well-posed learning program ⇒ a computer program is said to learn from exp (E) wrt task (T) and some perf. measure (P) if its perf. on (T) as measured by (P), improves with exp. (E).



checkers.

Types of (ML)

- SL (vs) SSL (vs) unsup. learning
- Reinf learning, recommender systems

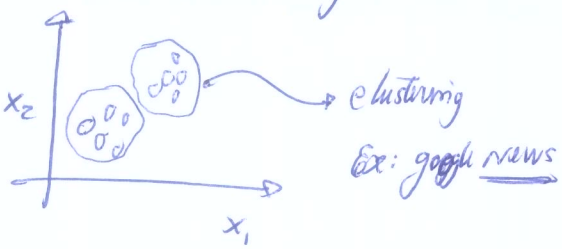
Practical advice: how to use and where to apply (ML).

Supervised Learning

- Regression ⇒ predict cont. output.
- classif ⇒ predict discrete value (class) output.

unsupervised Learning

find some structure on the data



Ex: google news.

Examples:
- Genes x Individuals
- datacenters.
- cohesive groups of people on social networks
- Group customers
- Astronomical data analysis (how galaxies are formed).

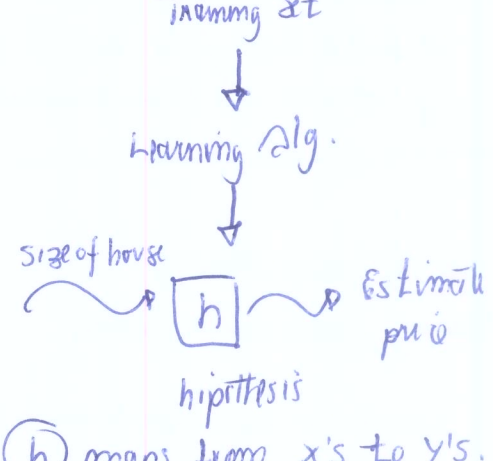
Cocktail party problem ⇒ overlapping voices.

Languages
- Octave
- matlab
- R

} first prototyping then ⇝ other languages

Notation

(m): nbr of training examples
x's input variable / features
y's out. variable / target variable
(x, y) one training example.
$(x^{(i)}, y^{(i)})$ i^{th} training example.

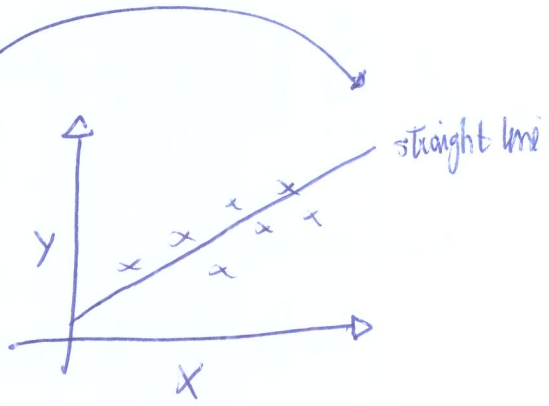


(h) maps from x's to y's.

How to represent h ?

Initial choice $h_\theta(x) = \theta_0 + \theta_1 x$

shorthand $h(x)$



why linear function ⇒ simple building block.

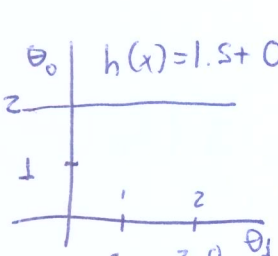
Here we have an example of a linear regression with one variable or univariate linear regression.

1 variable.

Defining a cost function

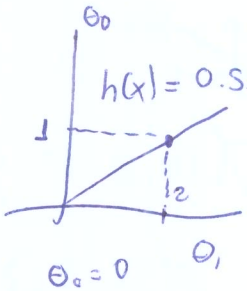
hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$ θs parameters

3 examples of hypothesis :



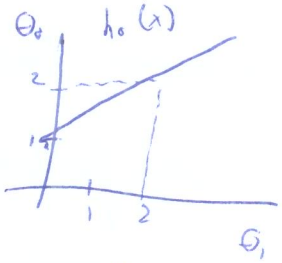
$\theta_0 = 2.0$

$\theta_1 = 0$



$\theta_0 = 0$

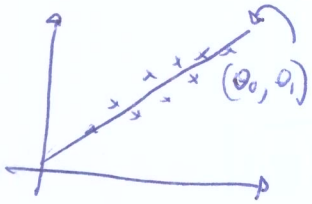
$\theta_1 = 0.5$



$\theta_0 = 1$

$\theta_1 = 0.5$

In linear Regression



$h_\theta(x)$ must be close to the training data.

More formally

minimize $\theta_0, \theta_1 \quad (h_\theta(x) - y)^2$.

$$\min_{\theta_0,\theta_1} \frac{1}{2M} \sum_{i=1}^{M} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$\frac{1}{2M}$ only for make it easier.

M = # Examples in the training.

perfect

$$\min_{\theta_0,\theta_1} \left(\frac{1}{2M} \sum_{i=1}^{M} \left(h_0(x^{(i)}) - y^{(i)} \right)^2 \right) = J(\theta_0, \theta_1).$$

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

with this, we define a cost function $J(\theta_0, \theta_1)$.

so we can simply say $\boxed{\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)}$ — cost function called squared Error cost function. most used for Regression.

(SSD).

Cost function intuition

① hypothesis $h_0(x) = \theta_0 + \theta_1 x$.

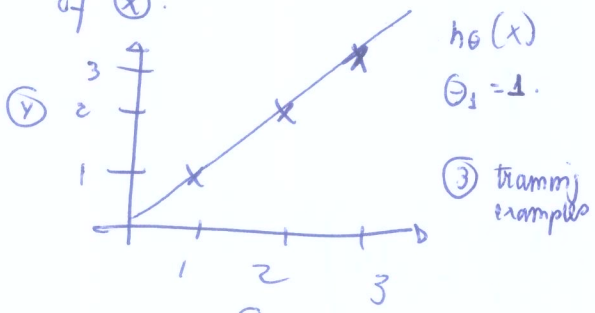
② Params θ_0, θ_1 → $h(x)$.

③ Cost function $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

④ Goal: $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

If we simplify $h_0(x) = \theta_1 x$. $\theta_0 = 0$.

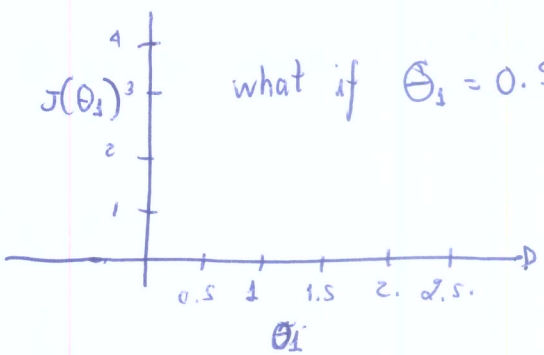
$\boxed{h_\theta(x)}$ for fixed θ_1, this is a function of x.



what is $J(\theta_1)$? $\frac{1}{2m} \left[\theta_1 (x^{(i)} - y^{(i)}) \right]$

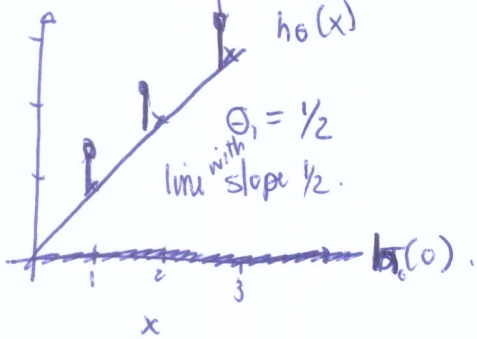
$= (0^2 + 0^2 + 0^2)/2m = \boxed{0}$

$\boxed{J(\theta_1)}$ is a function of the param θ_1.

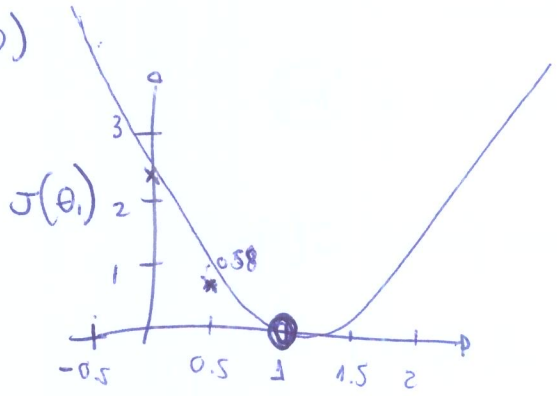


what if $\theta_1 = 0.5$?

$h_\theta(x)$



$J(\theta)$



$$J(0.5) = \frac{1}{2M}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$$

$$= \frac{1}{2\times 3}(3.5) \simeq 0.58$$

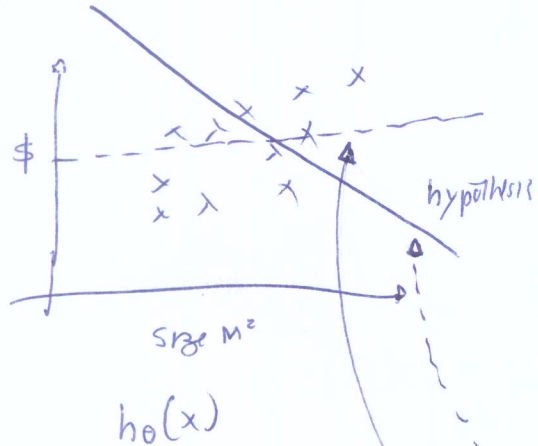
$J(0) = ?$

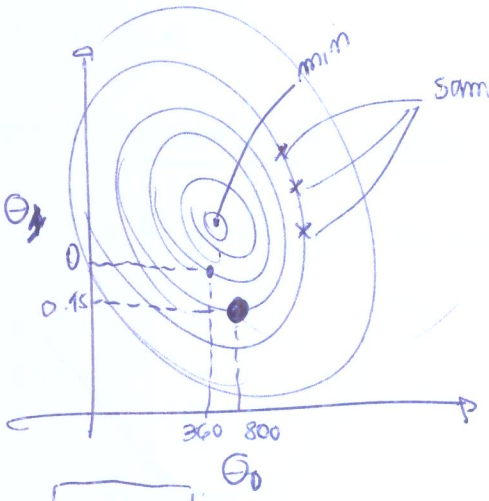
$$J(0) = \frac{1}{2m}\left(1^2+2^2+3^2\right)$$

$$\frac{1}{2m} \times 14 = 2.3$$

For each value of θ_1, we have a different hypothesis for $h_\theta(x)$ (different line).

Cost function (contour plots)





θ_0	θ_1
800	1.5

$\theta_0 = 0$
$\theta_1 = 360$

Gradient Descent

Used everywhere in (ML)

(1) Have some function: $J(\theta_0, \theta_1)$

(2) $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

(3) Outline
- (a) start with some θ_0, θ_1
- (b) keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum.

↝ what we need to define is α (the learning rate).

Implementation

Correct form (partial derivatives)

$$t_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$t_1 \leftarrow \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 \leftarrow t_0$$
$$\theta_1 \leftarrow t_1$$

} simultaneously update both θ_0 and θ_1

Incorrect form

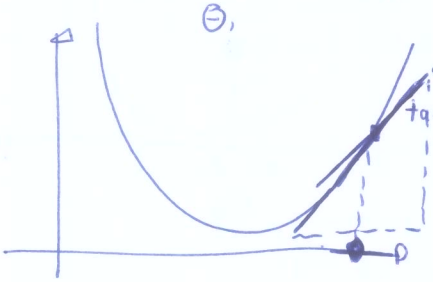
~~temp~~ $t_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\theta_0 \leftarrow t_0$

⋮

Intuition about the Gradient Descent

Suppose we have $\min_{\theta_1} J(\theta_1)$ $\theta_1 \in \mathbb{R}$.



$\theta_1 \in \mathbb{R}$.

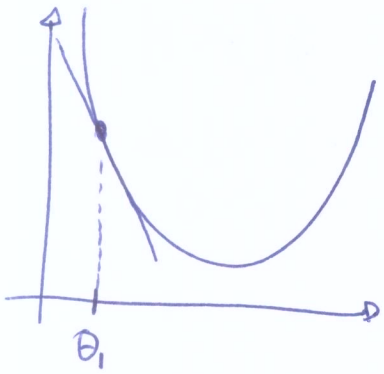
$\theta_1 = \theta_1 - \alpha \left[\frac{d}{d\theta_1} J(\theta_1) \right]$ ($\frac{\partial}{\partial \theta_0}$ partial derivative.)

derivative

slope of the line tangent to function.

≥ 0.

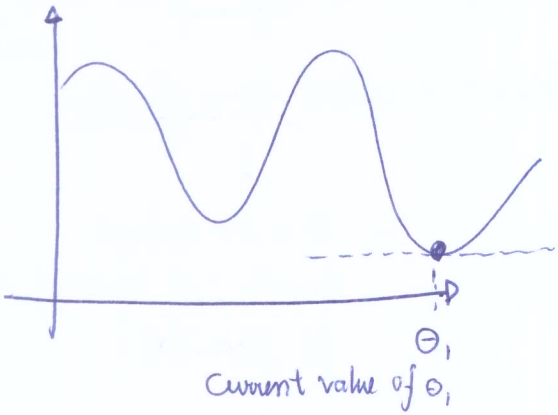
$\theta_1 \leftarrow \theta_1 - \alpha \cdot (\text{positive number}) \downarrow \theta$.



$$\frac{\partial J(\theta_1)}{d\theta_1} \leq 0.$$

$$\theta_1 \leftarrow \theta_1 - \alpha(\text{neg. number}) \uparrow \theta.$$

↝ What if (GD) is already at a minimum ?



$$\theta_1 \Leftarrow \theta_1 - \alpha \underbrace{\left[\frac{d}{d\theta_1} J(\theta_1)\right]}_{\text{zero}}$$

$$\theta_1 \leftarrow \theta_1 - \alpha \cdot 0$$

$$\theta_1 \leftarrow \theta_1.$$

θ_1 stays unchanged | that's why GD can converge to a local min even with α fixed.

GD automatically takes smaller steps as we approach a local min due to the derivative term. So, no need for decreasing α over time.

GD for linear regression

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

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

$$\frac{\partial}{x} (x^2 - 2)^2$$

$$2(x^2-2) \cdot 2x$$

$$4x(x^2-2).$$

$$\theta_0 \Rightarrow j = 0 : \frac{1}{m} \cdot \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right).$$

$$\theta_1 \Rightarrow j = 1 : \frac{1}{m} \cdot \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}.$$

~ The cost function for linear regression is always a bow shape function

~ The technical term for this is that it is called convex function

↓

- is a bow shape
- doesn't have local opt except for the global one.

The algorithm we just defined is what we call Batch Gradient Descent.

Batch : each step of GD uses all training examples.

~ there other version that use only a ~~few~~ subset of the training.

~ for the case of linear regression, there is a closed form solution (normal equations) but Gradient descent will scale better for larger datasets than normal equations.

~ The derivative is just the slope of the cost function J.

Generalization of Gradient Descent

Recalling, the GD algorithm is given by

repeat until convergence {

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

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

}

→ It can be susceptible to local optima.

→ But the cost function for the linear regression is always convex (bow-shape)

→ only the global optimum.

There are two extensions of GD that we must consider

(1) In $\min J(\theta_0, \theta_1)$, solve for θ_0, θ_1 exactly, without iterative GD algorithm

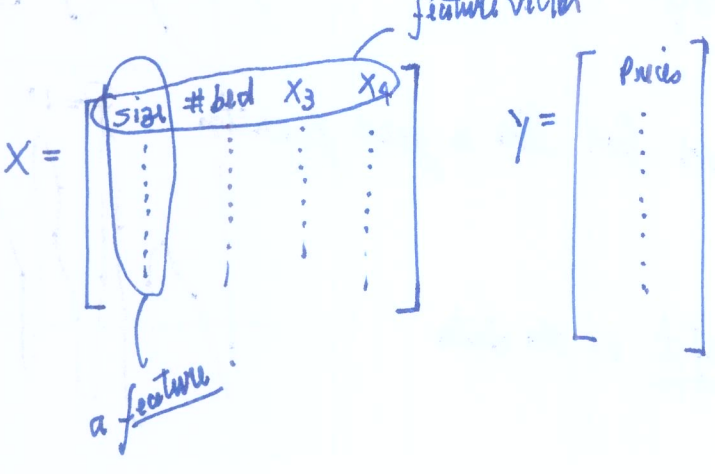
One advantage is that α is not necessary anymore but in some cases (~~large~~ high dim) the GD may be more appropriate.

(2) Longer ~~run~~ number of features. For instance

size, # bedrooms, # floors, # age of home ⇝ y (price).

x_1 x_2 x_3 x_4

Notation



Increasing the number of features

We have considered so far linear functions of several observed ~~variables~~ features

$x_1, x_2, \dots$

Suppose now that we have only one feature x_1 but we would like our predicter to be a non linear function of x $\hat{y}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \dots$

We can simply define new features $x_2 = x^2$, $x_3 = x^3$ just like we did with $x_0 = 1 = x^0$.

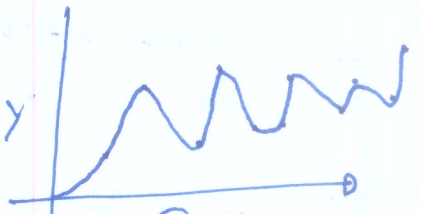
↝ The predicter then becomes $\hat{y}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$

It is : it still fits the linear regression model but in a new feature space with additional features that are deterministic functions of our observations

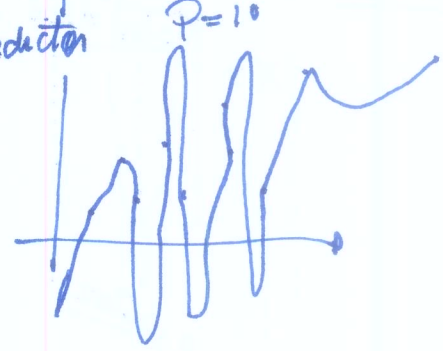
Applying the mse estimator $\hat{y}(x) = \sum_{i=1}^{10} \theta_i x^i$

Overfitting

In our polinomial fits for 11 data points, a new complex model $\hat{y}(x)$ a $p(x)$ with degree 10 fits the data perfectly and 11 coef.

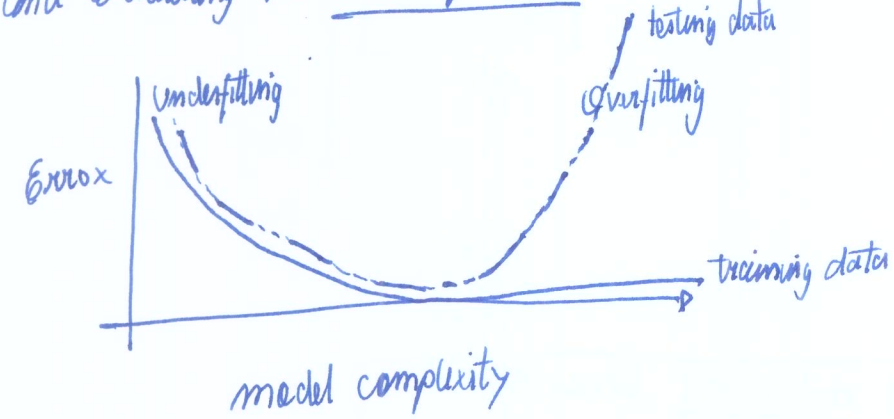


but it does not look like a good predicter



we fact an overfit to the data

We can see the **generalization** or test error simply by gathering more data and evaluating the **cost function**



Continuing Regression with multiple variables

size house (x_1), # bedrooms (x_2), floors (x_3), years (x_4), Price (y)

$x^{(i)}$ training example.

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

~ Previously we had a model/hypothesis: $\hat{y}(x) = \theta_0 + \theta_1 x$.

~ Let's update or upgrade our model

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_4 x_4.$$

Example $\hat{y}(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3 x_3 - 2 x_4$.

each year decreases de price.

for convenience let's define $x_0 = 1$.

zeroth feature always with the vale (1).

Then $x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

0-index

0-index vector

$h_\theta(x) = \hat{y}(x)$

$\hat{y}(x) = \theta_0 x_0 + \theta_1 x_1 \dots \theta_n x_n$ ($x_0 \to 1$)

$= \theta^T x$

$\theta^T = [\theta_0 \dots \theta_n]$ $\quad x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$

$\theta^T x$.

This is called multivariate linear regression

multiple variables/features

How do we do the (GD) with multiple variables/features?

Hypothesis $\hat{y}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 \dots \theta_n x_n$ ($x_0 \to 1$)

Params $\theta_0, \dots, \theta_n$. let's think it as a vector $\vec{\theta}$ (vector) $\in \mathbb{R}^{n+1}$

Cost function $J(\theta_0, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right)^2$.

$J(\vec{\theta})$.

Gradient Descent Repeat

$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\vec{\theta})$

update at the same time for all $j \in \{1, \dots, n\}$.

How to perform the partial derivative?

GD for $n \geq 1$. Repeat {

$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$ Only thing that changes

$x_0^{(i)} = 1$ by definition

}

Gradient Descent in practice 1: feature scaling

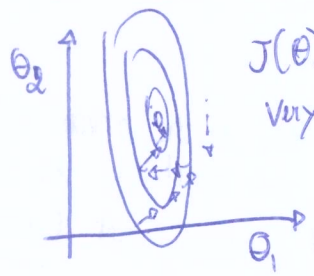
↝ what do we do if we have some features dominating the others?

↝ we need to scale the features in order GD can converge more quickly.

We need to make sure the features are in the same range of values

Example x_1 = size $(0-2000m^2)$

x_2 = nbr of bedrooms $(1-5)$



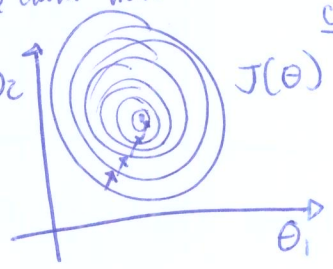
$J(\theta)$.

Very tall and skinny ovals.

if we run GD in here, it may take a while before converging

A nice workaround here would be to normalize the data then the contours would look like more circles

$x_1' = \frac{\text{size}\,(m^2)}{2000}$ $\quad x_2' = \frac{\#\text{ bedrooms}}{5}$.



$\left.\begin{array}{l} 0 \le x_1 \le 1 \\ 0 \le x_2 \le 1 \end{array}\right\}$

Feature scaling ↝ Get every feature into approximately a $\underbrace{-1 \le x_i \le 1}$ range (desired but not firmly fixed)

↝ x_0 is already in the range $(x_0 = 1)$

↝ $0 \le x_1 \le 3$ (is OK)

$-2 \le x_2 \le 4$ (is OK)

but

$-20 \le x_3 \le 40$ X

$-0.0001 \le x_4 \le 0.0001$ X.

↝ How to do it? training data.

In addition to scale normalization, sometimes we perform a step further and do mean normalization

↝ Replace x_i with $x_i - \mu_i$ for zero meaning the features
↝ Do not change $x_0 = 1$

Example $x_1 = \frac{size - 1000}{2000}$ $\quad x_2 = \frac{\#\,bedrooms - 2}{5}$ → if we do that, we will end up with values close enough to

$$-0.5 \leq x_1 \leq 0.5, \quad -0.5 \leq x_2 \leq 0.5.$$

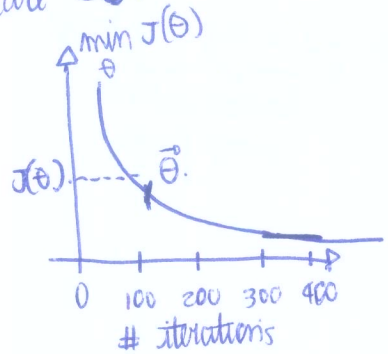
General rule $x_i \leftarrow \frac{x_i - \mu_i}{s_i}$ where μ_i is the mean of x_i in training set and s_i is the range of values in traing set ($\max_i - \min_i$).

sometimes it will interesting to use the std of feature

① $x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i}$ } z-norm / z-score — Data standardization

---

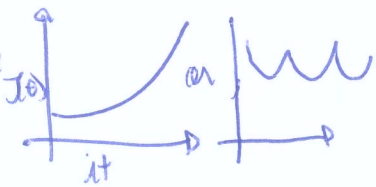
How do we know GD is working correctly and how to choose the learning rate α.



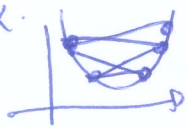
what you expect.

- J(θ) should decrease after iteration
- # of iterations vary according to the application
- An automatic convergence would be possible, for instance if J(θ) in one iteration is only slightly lower (10^{-3} for instance) than J(θ) in the previous iteration. But this is difficult.

if you have something like



then choose a smaller α.



Reason of increase: big α overshoots the minimum.

Two general observations:

① for small enough α, $J(\theta)$ should decrease on every iteration..

② for small enough α or too small α, GD can be slow to converge.

But... how to choose α ?

try... 0.001, 0.01, 0.1, 1, ... always plotting $J(\theta)$ x # iterations

or

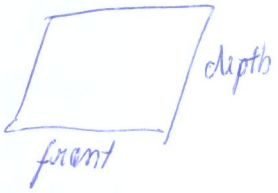
0.001, 3x0.001, 0.01, 3x0.01, ..., ...

or

make it linked to the nbr of iteration $\frac{\alpha}{\text{iteration}}$.

Features and Polynomial Regression

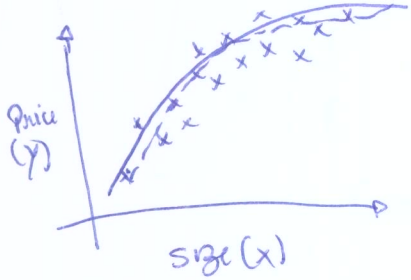
Housing prices $\hat{y}(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$



we can define a new feature x = front x depth (area)

then $\hat{y}(x) = \theta_0 + \underbrace{\theta_1 x}_{\text{area}}$.

Polymorial Regression



y_3 not good.

$\hat{y}_2(x) = \theta_0 + \theta_1 x + \theta_2 x^2$.

or

$\hat{y}_3(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

for using our multivariate linear regression model, we just define new features

$$\hat{y}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

$$\Downarrow$$

$$\hat{y}(x) = \theta_0 + \theta_1 x + \theta_2 x + \theta_3 x \quad \text{where}$$

$x_1 = \text{size}$, $x_2 = \text{size}^2$ and $x_3 = \text{size}^3$.

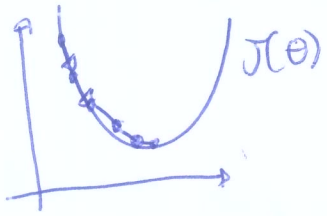
size	$1 - 10^3$
size^2	$1 - 10^6$
size^3	$1 - 10^9$

Note: if we do this, feature scaling is **paramount**

↝ How to choose the features ? There are some algorithms for that.

The Normal Equations

Gradient Descent



The Normal Equation is a method for solving for θ analytically (one step).

$$\theta \in \mathbb{R}^{n+1} \qquad \underset{\text{minimize}}{J(\theta_0 \dots \theta_m)} = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right)^2$$

as we want to minimize $J(\theta)$, we solve for

$$\frac{\partial J(\theta)}{\partial \theta_j} = \dots \text{ set to } 0 \quad \text{(derive and equal to zero)}$$

for every (1).

solving for $\theta_0, \theta_1, \dots, \theta_n$.

For example : for a given problem, put the feature vector in a matrix (X) and the outcomes (target) in a vector (Y).

$$X = \begin{bmatrix} & \text{area} & \text{bedrooms} & \text{floors} & \text{age} \\ 1 & 2104 & 5 & 1 & 45 \\ 1 & 1400 & 2 & 3 & 40 \\ 1 & a & b & c & d \\ 1 & e & f & g & h \end{bmatrix} \qquad y = \begin{bmatrix} \text{price (K\$)} \\ 460 \\ 232 \\ 500 \\ 200 \end{bmatrix}$$

$$\mathbb{R}^{m \times (n+1)} \qquad \mathbb{R}^{m}$$

$$\vec{\theta} = (X^T X)^{-1} X^T \underset{\Downarrow}{y} \quad \text{gives the min.}$$

General Case

(m) examples $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$, (n) features

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

↝ we will design what we call a design matrix

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

what does it mean $\vec{\theta} = (X^T X)^{-1} X^T y$.

① $(X^T X)^{-1}$ is inverse of matrix $X^T X$.

↝ when using Normal Equations, **feature scaling** is not necessary!

So, what should we use, (GD) or Normal Equation?

(GD)	Normal Equation
① Need to choose α.	① No α.
② Needs many iterations ↓	② No iterations ↑
① works well with a high number of features (n) ↑	① Needs to compute $(X^T X)^{-1}$ ↓
	② slow if (n) is very large.

** $X^T X \in \mathbb{R}^{n \times n}$, therefore normally it costs $O(n^3)$ for inverting on most inverse implementations

what is large?

- (n) in hundreds, go with Normal Equation
- (n) in thousands ($< 5K$), OK with Normal Eq.
- (n) $> 10K$ definitely go with (GD) or some alternatives.

finally the normal equation will not work for more complex problems such as classification, thats why (GD) is important and should be always thought as one option.

Normal Equations and non invertibility

↝ when computing $\vec{\Theta} = (X^T X)^{-1} X^T y$, what if $(X^T X)^{-1}$ is non-invertible/singular/degenerate?

↝ R and Octave and Matlab have workarounds (robust) inverse functions called pseudo-inverse that does the right thing.

↝ normally there are two main causes for degeneration:

(1) Redundant features (linearly dependent)
- ↳ solution: - dim. reduction
  - feature deletion/selection

Example x_1 = size in m^2. 1 km = 1000 meters
x_2 = size in km^2.
so $x_1 = (1000)^2 \cdot x_2$

(2) Too many features (# examples (m) ≪ # features (n))
- ↳ solution: - dim reduction
  - dim/feature selection/deletion
  - Regularization

Example: m = 10 examples, n = 100 dimensions; params $\in \mathbb{R}^{101}$

Organization of the class on August 14th, 2013

slide 9, 11, 12, 13, 14, 15, 16, 17, 18, (19).

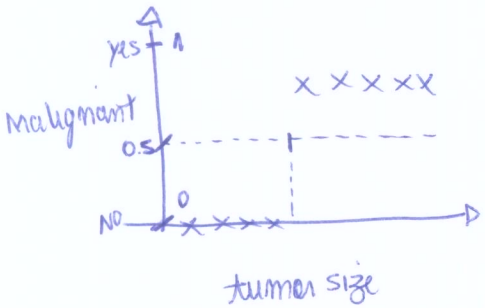
Logistic Regression and classification

1. Our outcome now is discrete-valued.
2. ~~LR~~ LogR is one of the most used learning algorithms.

Some classification problems: spam × non spam (e-mail)
fraudulent × non-fraudulent (transactions)
tumor malign × benign
going blind × not going blind (DR retinal) diab. retinopathy analysis

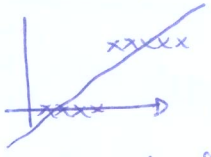
$y \in \{0, 1\}$ — 1: positive class; 0: neg. class (absence of something)

binary classif. problem.



what if we approach the problem as a linear regression one?

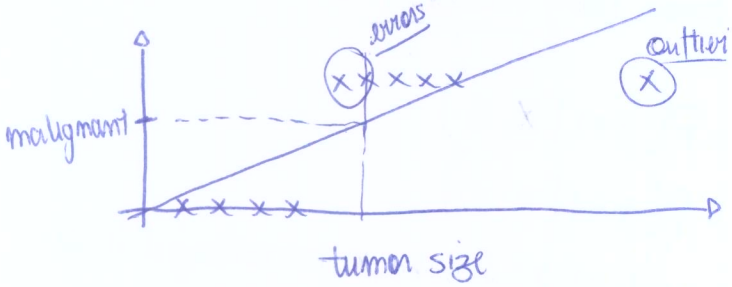
$\hat{y}(x) = \theta^T x$.



Then we could threshold classifiers output at $\hat{y}(x)$ @ 0.5:

if $\hat{y}(x) \geq 0.5$ predict 1
otherwise predict 0.

But let's change the problem:



often linear regression ~~is~~ isn't a good idea for classification.

↝ another problem with ~~LR~~ LogR linear regression is that even with training examples with $y \in \{0, 1\}$ it can output $\hat{y}(x) > 1$ or $\hat{y}(x) < 0$.

LogR solves this problem such that $0 \leq \hat{y}(x) \leq 1$ always.
↳ we will use it as a classif. algorithm. The name is historical.

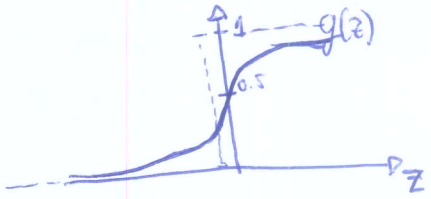
Hypothesis Representation

LogR model want $0 \leq \hat{y}(x) \leq 1$

when we had (LR) linear regression, $\hat{y}(x) = \vec{\theta}^T x$, for (LogR), we change it to $\hat{y}(x) = g(\vec{\theta}^T x)$,

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

sigmoid or logistic function synonyms.



By doing that, my new predictor will be

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

Now we need to (fit) parameters for (θ).

Interpretation of hypothesis output

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

Example $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

Suppose my outcome is $y(x) = 0.7$

Tell patient that 70% chance of tumor being malignant

Mathmatically, we have $\hat{y}(x) = P(y=1 \mid x; \theta)$

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

prob. of $y=1$ given x, parameterized by θ.

Properties $\begin{cases} P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1. \\ P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta). \end{cases}$

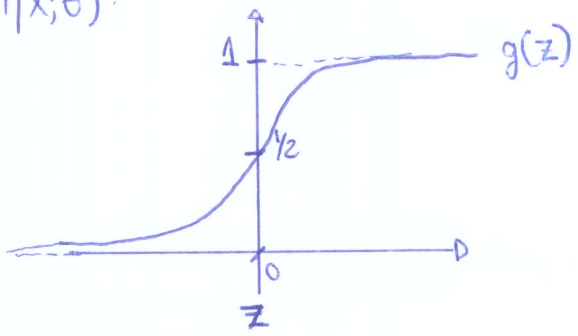
* (y) must take values in $\{0, 1\}$ binary value only.

Decision Boundary

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

(Log R) $\hat{y}(x) = g(\vec{\theta}^T x)$

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



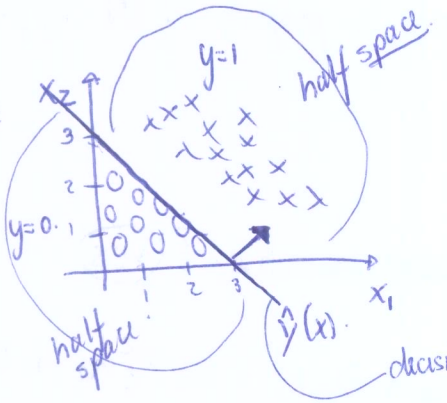
Suppose predict $y=1$ if $\hat{y}(x) \geq 0.5$

$y=0$, otherwise.

if we look at the sigmoid plot, we see $g(z) \geq 0.5$ when $z \geq 0$.

then given our predictor $\hat{y}(x) = g(\vec{\theta}^T x)$, it will be ≥ 0.5 when $\underbrace{\vec{\theta}^T x}_{z} \geq 0$.

Suppose we have training set



$\hat{y}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$.

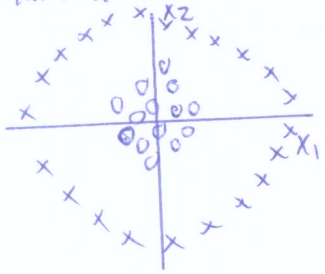
We still don't know how to choose θs but suppose $\theta_0 = -3$, $\theta_1 = 1$, $\theta_2 = 1$. $\vec{\theta} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$.

↝ **Predict** $y=1$ if $\underbrace{-3 + x_1 + x_2}_{\vec{\theta}^T x} \geq 0$ rewritting $\boxed{x_1 + x_2 \geq 3}$.

($\hat{y}(x) = 0.5$ exactly.)

↝ The decision boundary is a property of the hypothesis including the params $\theta_0 \dots \theta_2$. and not of the data set.

Let's take a look at a more complex example



↝ how can we fit (Log R) params to solve this problem?

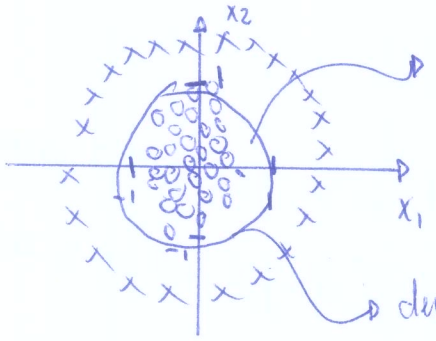
↝ we can transform the feature space to higher order just like we did with polynomials.

$\hat{y}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 \underbrace{x_1^2}_{\text{added features}} + \theta_4 x_2^2)$

Suppose we have found $\vec{\theta} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$.

My predicter will predict $\boxed{y=1}$ if $-1 + x_1^2 + x_2^2 \geq 0$

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



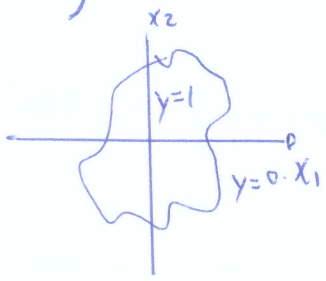
Equation of the circle $\boxed{x_1^2 + x_2^2 = 1}$.

decision boundary.

↳ Once again, the decision boundary is a property of the hypothesis/model and not of the dataset

↳ Can we have even more complex decision boundaries? (YES).

$$\hat{y}(x) = g\left(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 x_2 + \theta_5 x_1^2 \cdot x_2^2 + \theta_6 x_1^3 x_2 \cdot \right.$$
$$\left. \dots \right)$$



---

x

But, how do we automatically fit the params for a model/predictor? → for (LogR).

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

(m) examples $x \in \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$, $x_0 = 1$, $y \in \{0, 1\}$. ← classif. problem.

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

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

hypothesis

Question: how to choose parameters θ?

Recall that in the linear Regression hypothesis, we had $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(\hat{y}(x^{(i)}) - y^{(i)}\right)^2$

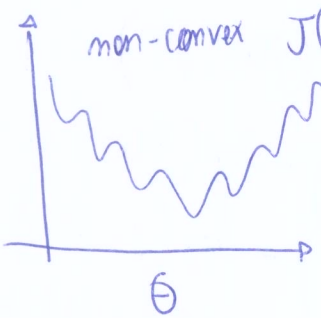
for (LogR), let's call $\frac{1}{2}\left(\hat{y}(x^{(i)}) - y^{(i)}\right)^2$ as $\text{cost}\left(y(x^{(i)}), y\right)$

so $\text{Cost}\left(\hat{y}(x^{(i)}), y^{(i)}\right) = \frac{1}{2} \cdot \left(\hat{y}(x^{(i)}) - y^{(i)}\right)^2$

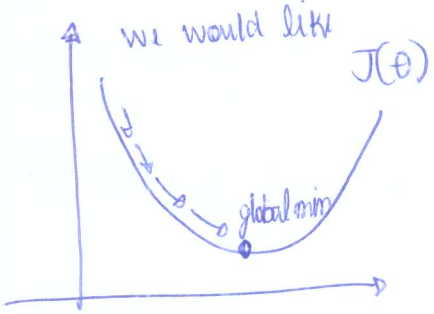
Simplifying, $\text{Cost}(\hat{y}(x), y) = \frac{1}{2} \cdot (\hat{y}(x) - y)^2$

for LogR has non linearity $\frac{1}{1 - e^{-\theta^T x}}$ complicated non linear function

this cost function works fine for (LR) but we are focused on (LogR) and if we minimize it for (LogR), it will be clear it isn't a convex function



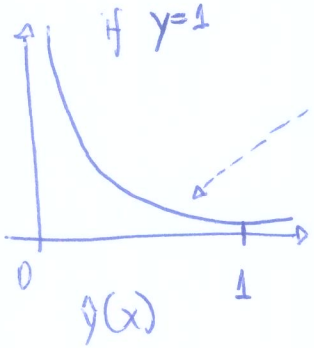
versus



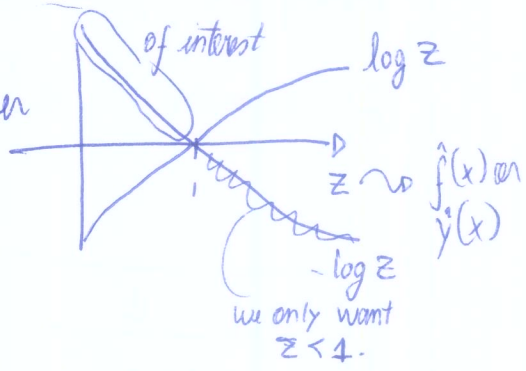
So we need to come up with a way of designing a different cost function that is convex.

Logistic Regression Cost function

$$\text{Cost}\left(\hat{f}(x), y\right) = \text{Cost}\left(\hat{y}(x), y\right) = \begin{cases} -\log\left(\hat{y}(x)\right) & \text{if } y=1 \\ -\log\left(1-\hat{y}(x)\right) & \text{otherwise } (y=0) \end{cases}$$



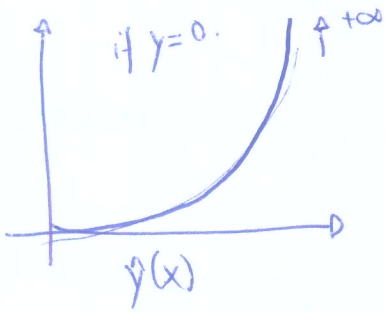
Remember



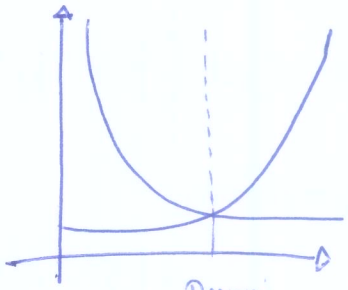
⇝ Cost = 0 if $y=1$, $\hat{y}(x)=1$ and prediction is

But as $\hat{y}(x) \rightsquigarrow 0$, Cost $\rightsquigarrow \infty$.

we penalize the learning alg. by a very large cost as it commits mistakes (go further away from the correcto value/output).



putting together



---

simplified Cost function and Gradient Descent

(LReg) cost function $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}\left(\hat{y}(x^{(i)}), y^{(i)}\right)$

where

$$\text{Cost}\left(\hat{y}(x), y\right) = \begin{cases} -\log\left(\hat{y}(x)\right) & \text{if } y=1 \\ -\log\left(1-\hat{y}(x)\right) & \text{otherwise} \end{cases}$$

$y \in \{0,1\}$ binary problem

Let's compress the two cases in one.

$$\text{Cost}\left(\hat{y}(x), y\right) = -y \log\left(\hat{y}(x)\right) - (1-y)\log\left(1-\hat{y}(x)\right).$$

so now, we can design our cost function as

$$\Downarrow$$

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

$$= \boxed{-\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log\left(\hat{y}(x^{(i)})\right) + \left(1-y^{(i)}\right)\cdot \log\left(1-\hat{y}(x^{(i)})\right)\right]}.$$

we choose it because it can derived from statists using principle of MLE max. likelihood estim. and it is convex.

~ To fit params $\vec{\theta}$: $\min_{\vec{\theta}} J(\vec{\theta})$.

~ To make a prediction for a new x, Output $\hat{y}(x) = \frac{1}{1+e^{-\vec{\theta}^T x}}$ which means $P(y=1 \mid x; \underbrace{\vec{\theta}}_{\text{learned}})$

Using GD for minimizing $J(\vec{\theta})$.

want $\min_{\vec{\theta}} J(\vec{\theta})$ Repeat{

$$\theta_j \leftarrow \theta_j - \alpha \cdot \underbrace{\frac{\partial}{\partial \theta_j} J(\theta)}_{**}$$

}

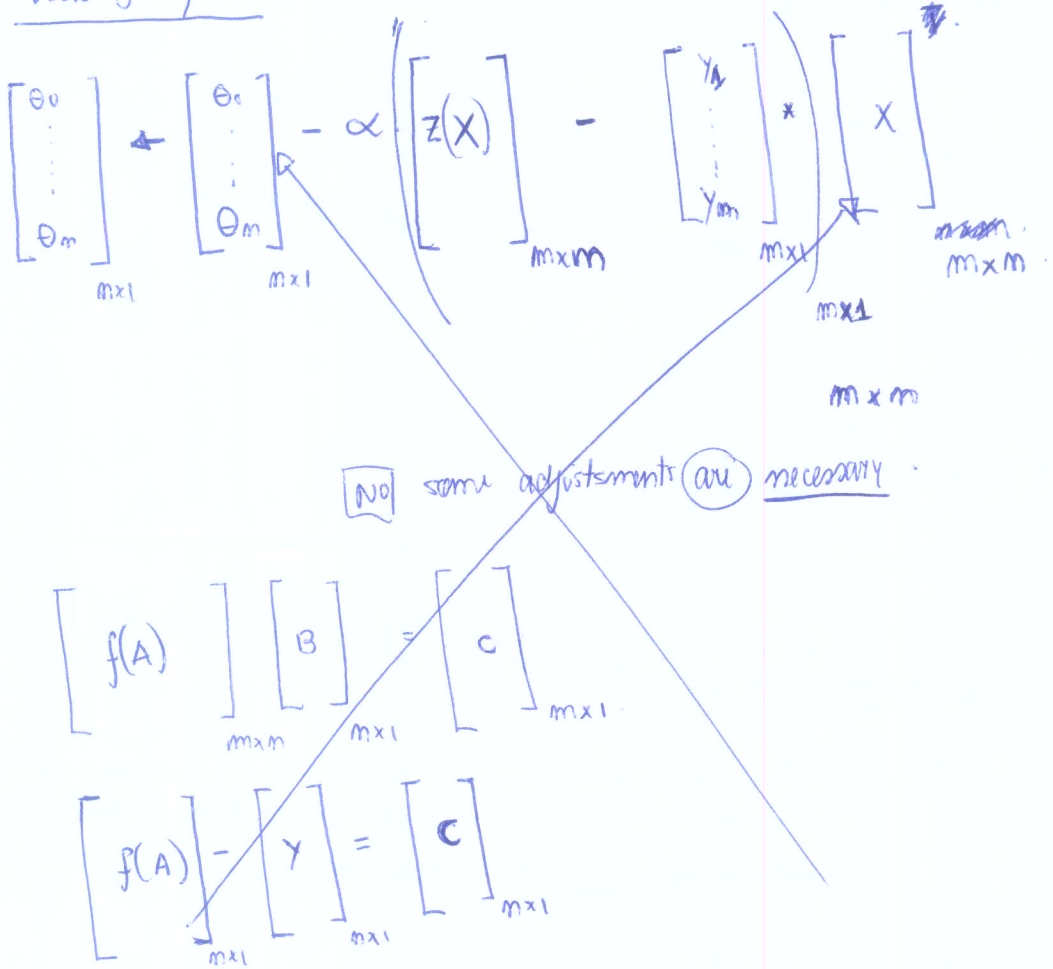
$$** \quad \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\cdot\sum_{i=1}^{m}\left(\hat{y}(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)}$$

what? it look exactly the same as before for LR. But the change is in $\hat{y}(x)$

before $\hat{y}(x) = \theta^T x$.

now $\hat{y}(x) = \frac{1}{1+e^{-\vec{\theta}^T x}}$ (non-linear)

In a vectorized form :



Logistic Regression - advanced optimization

So far, we have seen the opt. alg Gradient Descent and with it, we have a cost function $J(\vec{\theta})$ and want $\min_{\theta} (J(\theta))$.

Given $\vec{\theta}$, we compute ① $J(\theta)$

② $\frac{\partial}{\partial \theta_j} J(\theta)$ $\forall j = 0, \ldots, n_{features}$.

and (GD) does: Repeat:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

}.

But this is one alternative. If we show how to compute $J(\theta)$ and $\frac{\partial}{\partial \theta_j} J(\theta)$, we can use more sophisticated solutions such as
- Conjugate Gradient
- BFGS
- L-BFGS.

Normally, such alternatives ① Do not need α (learning rate) manually selected.
② often faster than GD.
however, they are more complex.

Example on how to use them:

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$ of course $\theta = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$ minimizes it and that's what we want to find.

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

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

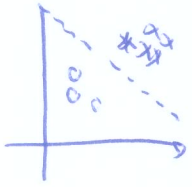
Normally, all we need to do is to write code for computing $J(\theta)$ and its derivatives.

Logistic Regression on multi-class problems - One vs All.

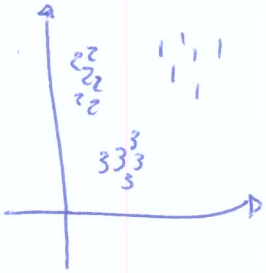
Example: digits, faces { profile left, profile right, frontal, age-group estimation, foldering/tagging photos emails,

foldering { family, business, gym ⋮ } $y \in \{1, \dots, k\}$ classes

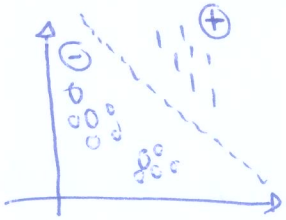
Binary Classif:



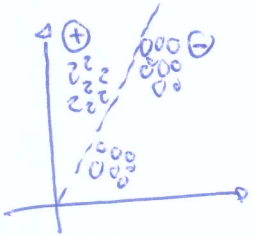
multi-class



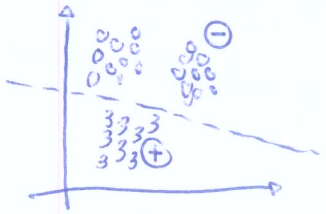
Let's do it on-vs-all (OVA): first we transform our problem on 3 problems



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



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



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

what we did was $f_\theta^{(i)}(x) = P(y=i \mid x; \theta)$ $i \in \{1, \dots, 3\}$

Basically, we train a (LogR) classifier $f_\theta^{(i)}(x)$ for each class (i) to predict $P(y=i)$. For a new input (x), for predicting, we select the class that maximizes $\boxed{\max_i f_\theta^{(i)}(x)}$

Regularization and the problem of overfitting

Regularization will allow us to diminish overfitting problems.

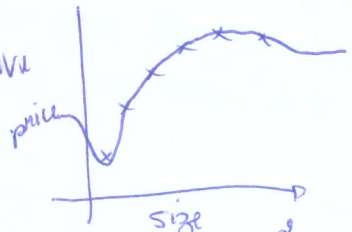
Lets return to the problem of housing

$\theta_0 + \theta_1 x$.

↝ underfit, high bias } algorithm has a very strong bias or pre-conception that the housing prices change linearly with the size even the data shows the contrary

both mean the model is not fitting the data very well.

alternatively, we could have



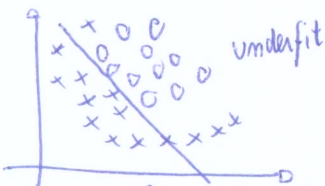
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfit, "high variance".

the model fitted the data perfectly and it is to just to it and the hypothesis is too variable since there is not enough data to prove it

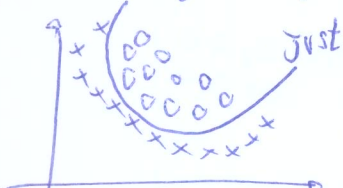
Overfitting: if we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} f_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$) but fail to generalize to new examples (predict prices on new data).

The same thing can happen with logistic regression as well.



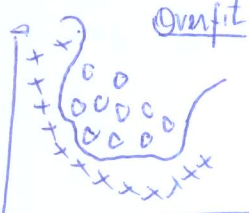
$f_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

g = sigmoid function



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

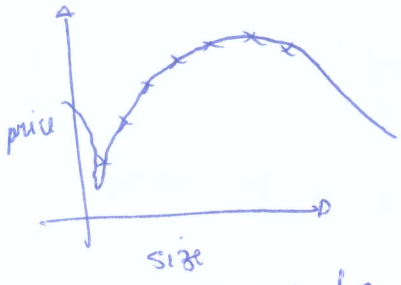
Overfit / high variance.



$f_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 \ldots)$.

polynomial

Addressing Overfitting



~ Plotting is one way but it wouldn't work for high-dimensional data

(Two) main options to deal with it

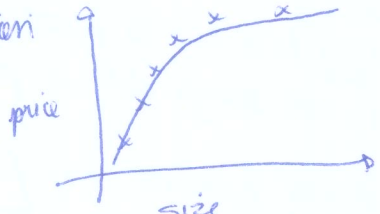
(1) Reduce number of features
- manually which features to keep
- model selection algorithm.
  automatic alg that select which features to keep/throw away.

* however, sometimes this means you are throwing away important info about the problem.

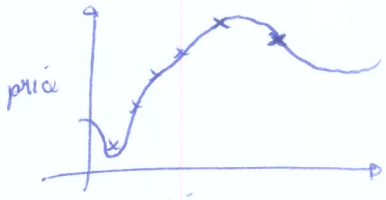
(2) Regularization
- Keep ALL the features but reduce magnitude/values/importance of parameters θ_j. works well when we have lots of features, each of which contributing a bit for predicting (y).

Regularization - Cost function

Intuition



$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize and make θ_3 and θ_4 really small

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} \left(f_\theta(x^{(i)}) - y^{(i)} \right)^2 + \boxed{1000\,\theta_3^2 + 1000\,\theta_4^2}$$

high penalization

this would give $\theta_3 \approx 0$
$\theta_4 \approx 0$

Regularization Cost function

Besides the intuition, here's the general idea:

↝ small values for parameters $\theta_0, \theta_1, \dots \theta_n$ { - Simpler hypothesis; - Less prone to overfitting.

Let's see the housing example

- features: $x_1, x_2, \dots, x_{100}$
- params: $\theta_0, \theta_1, \dots, \theta_{100} \in \mathbb{R}^{101}$

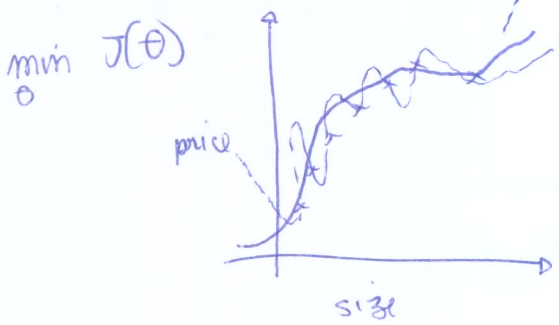
It is difficult to pick one or some to penalize beforehand. If it was easy, it would be the same as selecting the less important features!

So, here's what we do: keep the params small.

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

good fitting — Regularization

- not quadratic but smoother previously.
- do not penalize θ_0
- Regularization parameter

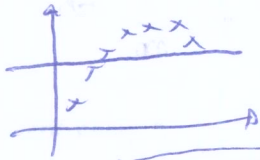


λ controls the tradeoff between good fitting to training data (vs) keep the parameters small.

and with both we want to keep the hypothesis simple and avoid overfitting.

Question: what if λ is too high (eg $\lambda = 10^{10}$) in a regularized linear regression?

↳ all θs except θ_0 will tend to zero and the model will be biased



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ → 0 zero.

therefore $f_\theta(x) = \theta_0$.

underfitting: too strong bias/pre-conception that a straight line in θ_0 will fit well in spite of data on the contrary.

↝ Be <u>carufull</u> when choosing λ.

Regularized Linear Regression

So far, for linear regression, we have seen <u>GD</u> and <u>Normal Equation</u> for fitting the parameters.

Here's the optimization function for regularized linear regression

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

$$\Downarrow$$

$$\min_\theta J(\theta).$$

Previously, the GD for minimizing this was

Repeat {

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

$j = 0, \cdots, m.$

}

if we separate θ_0:

Repeat {

$$\theta_0 \leftarrow \theta_0 - \frac{\alpha}{m}\cdot\sum_{i=1}^{m}\left(f(x^{(i)}) - y^{(i)}\right)x_0^{(i)}$$

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{m}\cdot\sum_{i=1}^{m}\left(f(x^{(i)}) - y^{(i)}\right)x_j^{(i)}.$$

}

$$\Downarrow$$

$\frac{\partial J(\theta)}{\partial \theta_0}$

Repeat {

$$\theta_0 \leftarrow \theta_0 - \alpha\cdot\frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)}) - y^{(i)}\right)\cdot x_0^{(i)}$$

$$\theta_j \leftarrow \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)} + \underbrace{\frac{\lambda}{m}\cdot\theta_j}_{\text{regularization}}\right]$$

}

$\frac{\partial J(\theta)}{\partial \theta_j}_{\text{regularized}}$

We can regroup things and then

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

$$\theta_j \leftarrow \theta_j \left(1 - \frac{\alpha\lambda}{m} \right) - \frac{\alpha}{m} \cdot \sum_{i=1}^{m} \left(f(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}$$

(small, small, target)

< 1 for instance $1 - \frac{\alpha\lambda}{m} < 1$

$1 - 0.01 = \boxed{0.99 \cdot \theta_j}$ shrinks θ_j a little bit.

what about normal equations ?

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}_{m \times (n+1)} \leftarrow \text{one training example} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

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

$$\vec{\theta} = (X^T X)^{-1} \cdot X^T y \quad \text{without Regularization}$$

for regularization;

$$\vec{\theta} = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \right)^{-1} \cdot X^T y$$

(zeros, zeros) $(n+1) \times (n+1)$

Example $m = 2$ $\rightarrow$ $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

what about non-invertibility now ? (Advanced) — non-invertible / singular.

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

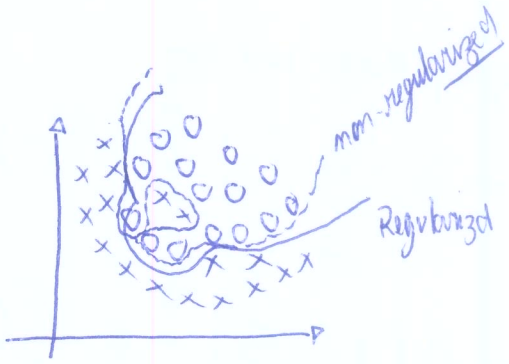
Suppose $m \leq n$ (# exam, # features)

but if $\lambda > 0$, $\vec{\theta} = \left(X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \end{bmatrix} \right)^{-1} \cdot X^T y$

we'll have no singularity anymore (why) some features will decrease magnitude and simplify model.

Regularized Logistic Regression

RegL can also be prone to overfitting:



$$\hat{f}_\theta(x) = g\left(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 \cdot x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots\right)$$

(g: sigmoid)

⇓

Cost function

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

introducing Regularization

We just add $A = \left[+\frac{\lambda}{2m}\sum_{j=1}^{m} \theta_j^2\right]$ (m: # dimensions) for $\theta_1, \dots \theta_n$

How to implement this?

Gradient Descent

Repeat {

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)}) - y^{(i)}\right) \cdot x_0^{(i)}$$

$$\theta_j \leftarrow \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)} - y^{(i)}\right) \cdot x_j^{(i)} \oplus \frac{\lambda}{m}\theta_j\right]$$

regularization

$j \in \{1, \dots, n\}$

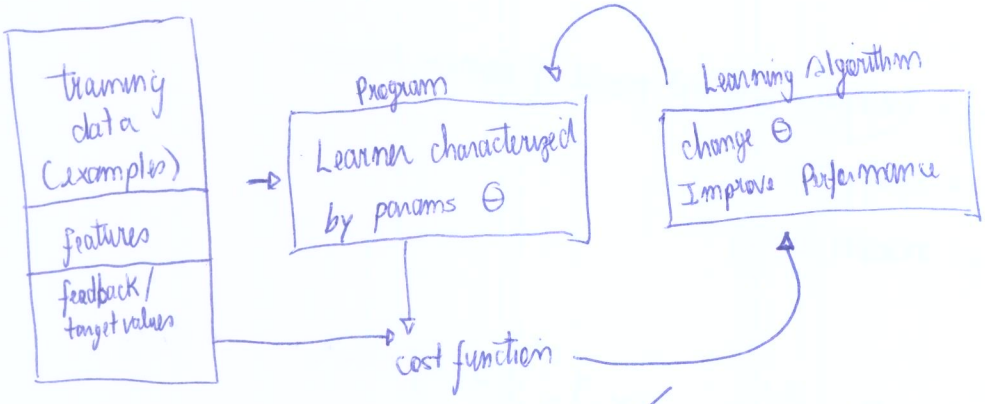
}

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

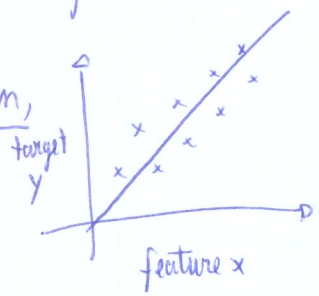
remember $f_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

Wrapping up what we had so far

notation: features x
targets y
predictions $\hat{y}$
params θ



In the case of linear regression,



"Predictor"
Evaluate line
$h = \theta_0 + \theta_1 x_1$

$\hat{y}(x) = \theta_0 + \theta_1 x_1$

$y \in \mathbb{R}$.

while with classification, we seek to predict a discrete value/target y.

Now that we already saw linear regression and logistic regression (classifier) let's see a similar linear algorithm for classification called perceptron.

Perceptron with 2 features

$\hat{f}(x) = \theta_1 x_1 + \theta_2 x_2 + \theta_0$.