Application of ML ⇒ self driving car, computer visions, etc

What is ML ⇒ field of study that gives computer

Arthur Samuel (1959) the ability to learn without being explicitly programmed

ML has 4 types 1) Supervised learning (used most in real world)

2) Unsupervised learning

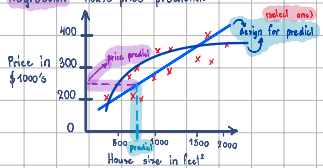
3) Recommender systems

4) Reinforcement learning

1) Supervised learning ⇒ learn with example with right answer

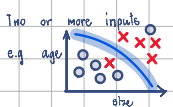
	correct	Example input	output	Application
	X input → Y output	ad, user info	click	online advertising
		English	Another language	machine translation

Regression : House price prediction (predict number (learn with input and output))



Classification ⇒ predict categories (can be num, non num e.g cat dog, 50, 1.5, cancer (benign, malignant))

output ⇒ class, category (small number of possible outputs)



2. Unsupervised learning ⇒ (unlabeled data) made algorithm figure by yourself

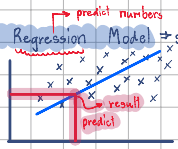
X input ≠ not has output Y

clustering ⇒ group of data that has same or similar topics

Anomaly detection ⇒ Find unusual data points

Dimensionality reduction ⇒ Compress data using fewer numbers

Linear Regression Model ⇒ straight line (predict numbers)



Terminology ⇒ data used to train the model

x = "input" variable or features

y = "output" variable or "target" variable

(x, y) = single trainning example

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

m = number of trainning examples

flowchart training set ⇒ 1) features 2) targets

↓

learning algorithm

↓

x (feature) → function (or hypothesis, model) → $\hat{y}$ "y-hat" (prediction, estimated y)

How to represent f? $f_{w,b}(x) = wx + b$ (value prediction; w, b or weights, parameters, coefficients)

model generate best linear function but not just only linear

you can using curve line for more accuracy

Univariate (one variable) linear regression

Cost function ⇒ tell how well the model is doing (or call it Squared error cost function)

Find w, b ⇒ $\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$

$\hat{y} - y$ ⇒ (call) error (how far value that predicting to target)

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - (y^{(i)}))^2$$

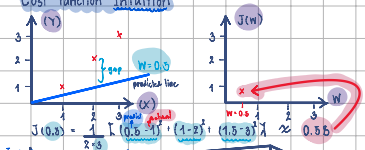
(sum of training, missing 1/2 because always positive value)

from $\hat{y}^{(i)} = f_{w,b}(x^{(i)})$

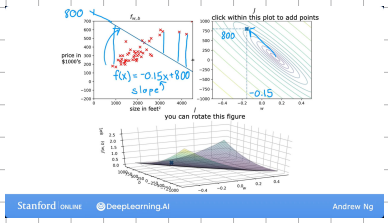
1/m ⇒ calculate average value

1/2 ⇒ make the derivative cleaner (clean)

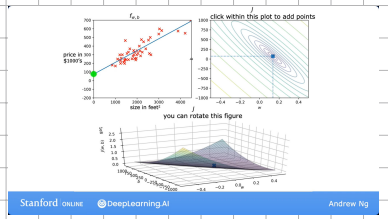
Cost function Intuition (Idea or concept behind the formula)



contour plots ⇒ a convenient way to visualize the 3-D cost function (J) but if plotted in 2D



not fit

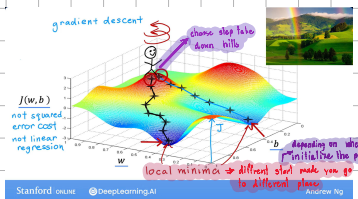


fit

Gradient Descent ⇒ use in every machine learning

using by keep changing w, b to reduce J(w,b)

until settle at or near a minimum.



Implementing Gradient Algorithm

$$w = w - \alpha \frac{d}{dw} J(w,b)$$

learning rate ⇒ control step you take down hills

Derivative ⇒ what direction you want to take you step

$$b = b - \alpha \frac{d}{db} J(w,b)$$

Simultaneously update w and b ⇒ update 2 parameter real time

$$temp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$temp\_b = b - \alpha \frac{\partial}{\partial w} J(w,b)$$

∂ term

$w = temp\_w \quad b = temp\_b$

we want to update w before using it

calculate temp_b

Gradient Descent Intuition

$J(w)$

$w = w - \alpha \frac{d}{dw} J(w)$

$w = w - \alpha \cdot$ (positive nums)

make to decrease function

make you close to local minima

$w = w - \alpha \cdot$ (negative nums)

make to increase direction

learning Rate

too small ⇒ small step using (long time)

long time to achieve local minima

$w = w - \alpha \cdot 0 = w$

too large ⇒ skip or pass local minima, never reach minimum (over shoot)

Gradient Descent for Linear Regression

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \rightarrow \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \rightarrow \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

}

(Optional)

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)})^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)}) 2x^{(i)} = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)})^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)}) 2 \quad \text{no } x^{(i)} \quad = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

convex function ⇒ bowl shape (It has global minima)

Running Gradient Descent = "Batch" gradient descent

Batch ⇒ each step of gradient descent uses all the training example (or The Batch)

Make linear regression more powerful

Multiples Features (variables)

original: $y = wx + b$ (output, input)

but with multiples using by this

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ = features of i^{th} training example (another name is "row vector" or "column vector")

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

e.g.

	x_1	x_2	x_3	x_n } n feature
	1000	10	2	5
$\vec{x}^{(2)}$ (row 2)	1500	20	1	6
$x_3^{(2)} = 1$ (2nd row, 3 column)	2000	30	3	1

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

(model; parameters; features; number)

$\vec{w} = [w_1, w_2, \dots, w_n]$ (vector) — linear algebra count from 1

b is a number

$\vec{x} = [x_1, x_2, \dots, x_n]$

write again $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$ (dot product)

Vectorization → make code shorter and enable gpu hardware

using for manage vector allow code to run more efficiently

$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$ → `f = np.dot(w,x) + b`

Why it run so fast?

Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

t_0 `f + w[0] * x[0]`

t_1 `f + w[1] * x[1]`

...

t_{15} `f + w[15] * x[15]`

Vectorization

```
np.dot(w,x)
```

t_0 w[0] w[1] ... w[15]

in parallel * * ... *

t_1 x[0] x[1] ... x[15]

w[0]*x[0] + w[1]*x[1] + ... + w[15]*x[15]

efficient → scale to large datasets

Gradient descent $\vec{w} = (w_1 \ w_2 \ \cdots \ w_{16})$ parameters

derivatives $\vec{d} = (d_1 \ d_2 \ \cdots \ d_{16})$

```
w = np.array([0.5, 1.3, ... 3.4])
d = np.array([0.3, 0.2, ... 0.4])
```

learning rate α

compute $w_j = w_j - 0.1d_j$ for $j = 1 \dots 16$

Without vectorization

$w_1 = w_1 - 0.1d_1$

$w_2 = w_2 - 0.1d_2$

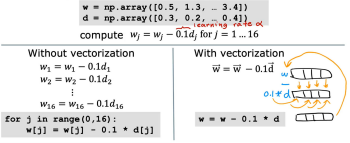
⋮

$w_{16} = w_{16} - 0.1d_{16}$

```
for j in range(0,16):
    w[j] = w[j] - 0.1 * d[j]
```

With vectorization

$\vec{w} = \vec{w} - 0.1\vec{d}$




```
w = w - 0.1 * d
```

Gradient Descent for Multiple Regression (features ≥ 2)

$$w_1 = w_1 - \alpha \frac{1}{M} \sum_{i=1}^{M} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_1^{(i)}$$

⋮

$$w_n = w_n - \alpha \frac{1}{M} \sum_{i=1}^{M} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{M} \sum_{i=1}^{M} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

An alternative to linear regression

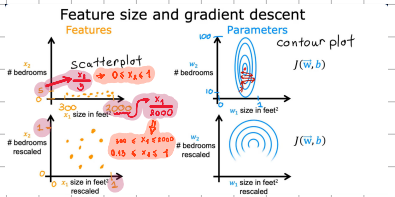
- Normal Equation 1) Only for linear regression
  2) Solve for w,b without iterations

Disadvantages 1) Doesn't generalize with other learning algorithms

2) Slow when number of feature is large ($n > 10{,}000$)

Gradient descent is the recommend method to find w,b

Feature Scaling → make gradient descent run much faster



divide maximum value is not only way to feature scaling

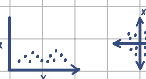
another way call **Mean normalization**

$x_i' = \frac{x_i - \mu_i}{\max - \min}$ (feature i'th; average value of all data in i^{th} feature)

And another method is **Z-score normalization** (it tells you how far data points to mean)

standard deviation σ (calculate in each features)

$x_i = \frac{x_i - \mu_i}{\sigma_i}$ $\quad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(x_i - \mu_i)^2}$ (variance)



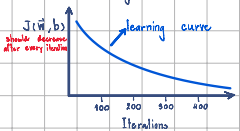
aim for about $-1 \le x_i \le 1$ for each features

$-3 \le x_i \le 3$

$-0.3 \le x_i \le 0.3$ (acceptable ranges)

Checking Gradient Descent for convergence

⇒ make sure gradient descent is working correctly



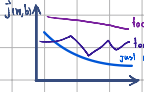
Automatic convergence test

ε "epsilon" ⇒ 10^{-3}

if J(w,b) decrease by ≤ ε in one iteration,

stop model training

Choosing the learning Rate ⇒ select from J(w,b) will decrease every iterations



try like this ⇒ 0.001, 0.01, 0.1, 1, ...

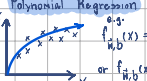
Feature Engineering

e.g. $f_{w,b}(\vec{x}) = w_1 x_1 + w_2 x_2 + b$ (x1: price, x2: size)

area = $(x_1)(x_2)$ (new feature) ← transforming or combining original features

Polynomial Regression ⇒ curve function

(multiplies linear regression + feature engineering)



e.g. $f_{w,b}(x) = w_1 x + w_2 x^2 + b$

or $f_{w,b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b$

Choice of features

$f_{w,b}(x) = w_1 x + w_2 \sqrt{x} + b$

Never forget to study the last 3 optional labs in Module 2

Classification with logistic regression

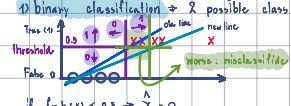
negative class (0) | positive class (1)

false | true

no | yes

e.g. Is this email spam? ? possible class or category

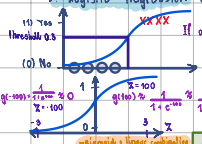
1) binary classification ⇒ 2 possible class or category



if $f_{w,b}(x) < 0.5 \rightarrow \hat{Y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{Y} = 1$

2 logistic Regression (use most in real world)



If output between 0,1 using sigmoid function or logistic function

$g(z) = \frac{1}{1+e^{-z}}$ $\quad 0 < g(z) < 1$

$g(-100) = \frac{1}{1+e^{100}} \approx 0$ $\quad g(100) \approx \frac{1}{1+e^{-100}} \approx 1$

linear combination	polynomial function
$z = \vec{w} \cdot \vec{x} + b$	$g(z) = \frac{1}{1+e^{-z}}$

$f_{w,b}(\vec{x}) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$

Decision Boundary

$f_{w,b}(\vec{x}) = g(z) = g(w_1 x_1 + w_2 x_2 + b)$ (w1 = 1, w2 = 1, b = -3)

$z = \vec{w} \cdot \vec{x} + b = 0$

$= x_1 + x_2 - 3 = 0$

$x_1 + x_2 = 3$

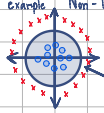
$x_2 = 3 - x_1$

$(x_1, x_2) = (0, 3)$

$(x_1, x_2) = (3, 0)$

Yes: $\hat{Y} = 1$	No: $\hat{Y} = 0$
$f_{w,b}(\vec{x}) \geq 0.5$	$z < 0$
$g(z) \geq 0.5$	$\vec{w} \cdot \vec{x} + b < 0$
$z \geq 0$	$\hat{Y} = 0$
$\vec{w} \cdot \vec{x} + b \geq 0$	
$\hat{Y} = 1$	

example Non-linear decision boundaries



$f_{w,b}(\vec{x}) = g(z) = g(w_1 x_1^1 + w_2 x_2^1 + b)$ (w1 = 1, w2 = 1, b = -1)

$z = x_1^2 + x_2^2 - 1 = 0$

$x_1^2 + x_2^2 = 1$

Cost Function for logistic Regression ⇒ How to choose $\vec{w}$ and b

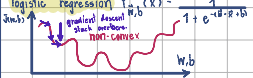
(0 or 1) true label

model predict probability for input x

Squared error cost $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}(f_{w,b}(\vec{x}^{(i)}) - y^{(i)})^2 = h(f_{w,b}(\vec{x}^{(i)}), y^{(i)})$

linear regression $f_{w,b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

logistic regression $f_{w,b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$



loss is lowest when $f_{\vec{w},b}(\vec{x}^{(i)})$ predicts close to true label $y^{(i)}$

$h(f_{w,b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{w,b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{w,b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$ (2 classes)

check how close the model's predicted value is to the true label $y^{(i)}$

example model predicts 0.01 (this predicted value closer to label 0)

$-\log(0.01) \approx 14$ (High loss) ← check predicted value closer to 1

$-\log(1-0.01) \approx 0.01$ (low loss) ← check predicted value closer to 0

Simplified Cost Function

original $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$

if $y^{(i)} = 1$: $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -\log(f_{\vec{w},b}(\vec{x}^{(i)}))$

if $y^{(i)} = 0$: $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$

from cost function $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})\right]$

$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))\right]$

Gradient Descent Implementation

Linear regression same like logistic regression

but here this is different

1) linear regression $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

2) logistic regression $f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$

Same concepts: 1) Monitor gradient descent (learning curve)

2) Vectorized implementation

3) Feature scaling

The problem of overfitting

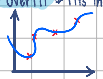
- underfit → does not fit the training set well (high bias)

$w_1 x + b$

- generalization → fits training set pretty well

$w_1 x + w_2 x^2 + b$ (high variance)

- overfit → fits the training set extremely well



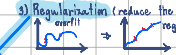
$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

Addressing Overfitting

How to fix overfit → 1) collect more training examples

2) select useful features (wrong or useful features could be lost)

3) Regularization (reduce the size of parameters w_j)



Cost Function with Regularization

make w_3, w_4 really small (≈ 0)

$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 \approx 0$

choose λ too small → still overfit

choose λ too large → prediction can be straight line of y-axis → b

if we have n features

$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2 + \frac{\lambda}{2m} b^2$

($\lambda > 0$)

lambda or regularization parameter

number of features

λ balances both goals: 1 fit data, 2 decrease w_j small

regularization term

Regularized Linear Regression + Gradient Descent

from Gradient descent $w_j = w_j - \alpha \frac{d}{dw_j} J(\vec{w},b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j$

don't have to regularize b

$w_j = 1 w_j - \alpha\left(\frac{\lambda}{m}\right) w_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}$

shrink w_j — usual update

$w_j (1 - \alpha\frac{\lambda}{m})$

$w_j (1 - 0.01 \frac{1}{50})$

$w_j (0.9999)$

Regularized logistic regression

$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-z}}$

$J_{\vec{w},b} = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$

n → numbers of features