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

Parameters : θ_0, θ_1

Cost function

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

m = # of samples

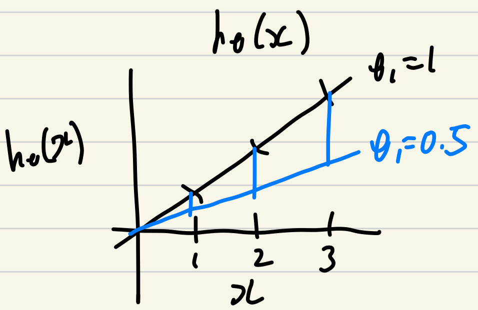
hypothesis $h_\theta(x)$: 추정치

y : 실측치

Goal: minimize$_{\theta_0, \theta_1}$ $\underbrace{J(\theta_0, \theta_1)}_{\text{cost function}}$

cost function = Squared error function

$\theta_0 = 0$ 일때 simplified cost function → goal minimize $J(\theta_1)$



$\theta_1 = 1$ 일때 $J(\theta_1)$는;

1) $J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)^2$

$= \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0$

$J(1) = 0$

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

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

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

$= \frac{1}{6}\cdot 14 = 2.33..$

* 실측치와 가장 잘 맞는 $\theta_1 = 1$ 에서 $J(\theta_1)$이 최소가 되는 것을 확인함.

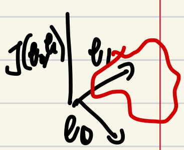
(기울기하강)

Gradient Descent : $J(\theta_0, \theta_1, \cdots, \theta_n)$을 minimize 하기 위한 방법

+) 꼭 선형회귀가 아니더라도 이 방식 사용가능

· start with some θ_0, θ_1

· keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$



· 언덕의 꼭대기 ($J(\theta_0,\theta_1)$)이 높은 지점부터 아래로 내려가기위한 제일 최단거리로 계속 내려간다.

· Gradient Descent algrithm

cf) $a := b$ means b를 a에 씌운다.
$a = b$ T or F

repeat until convergence

미분계수

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

simultaneously update θ_0 & θ_1
⊕ explain

learning rate (훈련비율)
: 내려가기위한 보폭을 어느정도로 설정할건인가.
$\alpha \uparrow$ - 공격적인 기울기 하강

⊕ explain

θ_0, θ_1을 구하고 나서 새로운 θ_0, θ_1을 갖고 다시 계산하는 것.
θ_0 구했다고 θ_1 구할때 바로 그 새로운 θ_0을 갖고 계산하지 X.

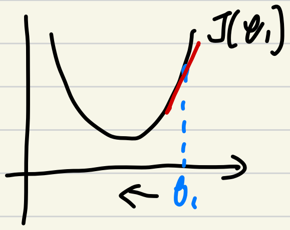
temp0 := 공식
temp1 := 공식 → 이걸 끝내고
θ_0 := temp0
θ_1 := temp1 ← 그 다음 이걸.

→ simultaneous update
θ_0, θ_1를 동시에.

미분계수

$\alpha \boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$ 개념.

$\min_{\theta_1} J(\theta_1)$ $\theta_1 \in \mathbb{R}$

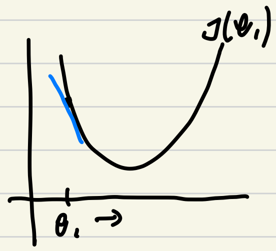


$\theta_1 \in \mathbb{R}$

θ_1에서의 기울기

$$\theta_1 := \theta_1 - \alpha \underbrace{\boxed{\frac{d}{d\theta_1} J(\theta_1)}}_{\geq 0}$$

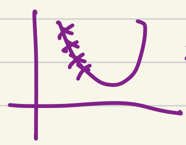
$\theta_1 := \theta_1 - \alpha$(positive number)
θ_1이 왼쪽으로 이동 : 최소$J(\theta)$에 가까워짐



$\frac{d}{d\theta} J(\theta_1) < 0$

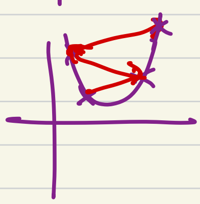
$\theta_1 := \theta_1 - \alpha$(negative number)
θ_1이 오른쪽으로 이동하게됨 : 최소$J(\theta)$에 가까워짐

: α이 작으면



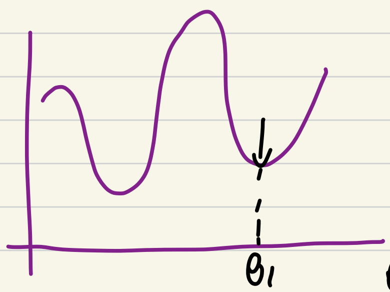
작게 움직이고

α이 크면



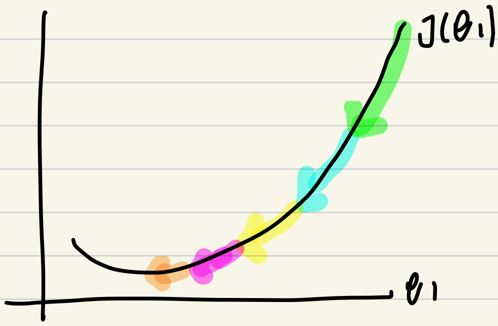
크게 움직이느라 지나칠수도 있다.

cf)



θ_1의 기울기가 0이면 아예 local minimum이라 움직이지 X.

· α가 고정되어 있더라도 계산을 반복할수록 미분계수의 절대값이 작아지기 때문에 최저값에 가까워질수록 이동거리가 작아진다. ∴ α를 변경하지 않아도 괜찮다.



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

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

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

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

Gradient Descent에 이용.

"Batch" gradient Descent : 집단 기울기 하강

---

vector : n×1 matrix

vector $y = \left[\quad\right]\}n$ 표시! $\mathbb{R}^4$

matrix 표시 $\mathbb{R}^{m\times n}$

y_i = ith element

1-indexed vs 0-indexed

index start가 1 이냐 0 이냐

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

행렬 표시 대문자 사용
vector, scalar 표시 소문자 사용] → 항상 그런건 아니지만 일반적

Multiple features (variables)

n = number of features

몇번째 표본? → $x^{(i)}$ = input (features) of i^{th} training sample

그 표본의 몇번째 변수값 → $x_j^{(i)}$ = value of feature j in i^{th} training sample

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

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

쉬운 이해를 위해 x_0 =1로 놓자. $(x_0^{(i)} = 1)$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \Big| \qquad \underbrace{\begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}}_{\theta^T} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$(n+1) \times 1$ matrix

Hyphothesis: $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots \theta_n x_n$

$= \theta^T x$ (으로 간편히 쓸수 있다)

Parameters: $\theta_0, \theta_1, \cdots \theta_n$ 이므로 θ 벡터 그 자체라고 놓을수 있다.

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

Gradient Descent

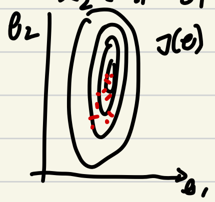
Repeat $\left\{ \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \right.$ simultaneously update θ_j for $j = 0, \cdots, n) \}$

Feature Scailing

: make sure features are on a similar scale

E.g x_1 = size (0-2000 $feet^2$)

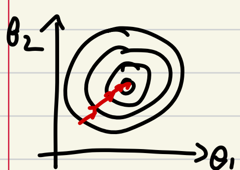
x_2 = # of bedroom (1-5)



→ it takes more time to find $\min J(\theta)$

큰 범위의 값일수록 θ가 천천히 이동하므로

So, $x_1 = \frac{size}{2000}$, $x_2 = \frac{\text{\# of bedroom}}{5}$



feature들이 비슷한 범위의 값을 갖도록 만들자. -3 to 3, $-\frac{1}{3}$ to $\frac{1}{3}$ 같이

- mean normalization 도 있음.

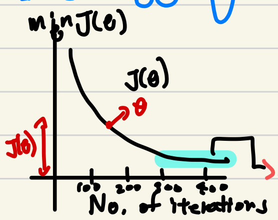
$x_1 = \frac{size - 1000}{2000}$

$x_2 = \frac{\#bedroom - 2}{5}$

이렇게. or $\frac{x_1 - \mu_1}{S_1}$ (MAX- Min)도 되고 standard deviation 도 된다.

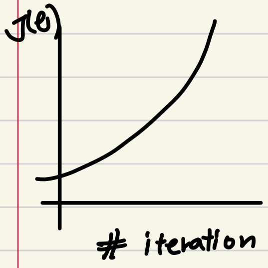
여기 있는 것은 다 근사값이므로 엄청 정확할 필요는X

Debugging: make sure gradient descent working correctly

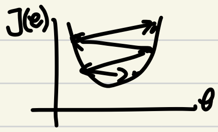


: 반복할 때마다의 J(θ) 그래프
correct : J(θ)가 점점 줄어들어야 한다, 수렴해야 함.
이 구간은 거의 수평. J(θ)가 수렴하고 있다.

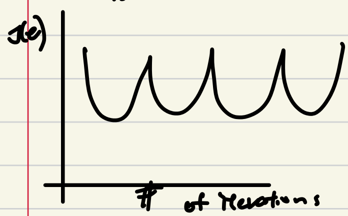
수렴하는지 판단하는 법
: J(θ)의 감소량이 10^{-3} 보다 작을때
↳ 이 임계값은 좀 불명확해서 이 방법은 잘 안씀. 그래프 그리는 것이 나음



: 이런 그래프는 α가 커서



하느라 그런것이므로 알파를 작게 하면 된다.



: 이 경우도 위와 같음.

Summary

: if α is too small, slow convergence
: 〃 large : J(θ) may not decrease

To choose α try : 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1 ..

가능한 가장 작은값, 큰값을 선택한후 작은 값에서 한 단계 큰값
큰값에서 한단계 작은 값을 선택해가며 α를 결정한다.

집의 길이, 너비가 있을 때 이 둘을 곱해서 넓이라는 하나의 feature을 가지는 모델을 만들수 있다.

Normal Equation 은 feature scaling 굳이 안해도 됨

$x_0, x_1, \cdots x_n$ 값들을 $X = [\ ]$ 에 한 vector 씩 다 넣는다
$m \times n+1$

$y = [\ ]$ } $m \times 1$ vector

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

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

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ — (x^{(m)})^T — \end{bmatrix}$$

design matrix

$m \times (n+1)$

If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ $\quad X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(1)} \\ 1 & \vdots \\ 1 & x_m^{(1)} \end{bmatrix}$ $\quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

$m \times 2$

octave : pinv(X' * X) * X' * y

Gradient Descent VS Normal Equation

– m training examples, n features

Gradient Descent	Normal Equation
· Need to choose α · Needs many iterations	· No need to choose α · Don't need to iterate
· Works well even when n is large · logistic algorythm 에도 적용가능 For large n $n > 10^6$	· Need to compute $(X^T X)^{-1}$ · Slow if n is very large For small n n이 1000 정도면 ok

· $X^T X$ 가 non-invertible 일때,

1) linearly dependent

2) too many features ($m \le n$), 작은 양의 예제로 파악하긴 어렵다.

→ 이럴 땐 불필요한 feature가 있는지 살펴보기

1) Unsupervised learning : y가 없다. unlabeled example.

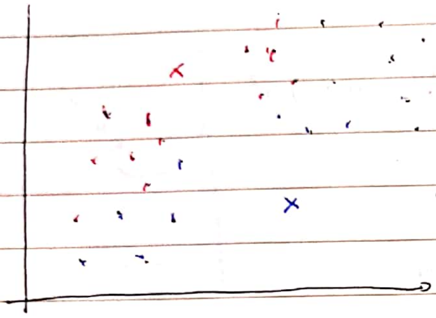
쓰이는 곳 ① 고객 세분화 : Market segmentation → 각 그룹에게 물품 팔기

② Social network analysis : 이메일을 주고 받는 것을 보고 그룹의 특성

③ Organize computing clusters : 데이터 센터 체계화

④ Astronomical data anaylsis

2) K-means algorithm. ① cluster assignment ② move centroids



1) randomly initialize two points → cluster centroids

cluster assignment 2개인 이유는 group 2개로 나누고 싶어서

2) 각 점들은 제일 가까운 centroid의 색이 됨

move centroids 3) 같은 색의 점들의 위치 평균을 그 그룹의 새로운 점으로 고정

4) ①부터 다시 반복. 이게 바뀌지 않을때 끝남.

Input:

- K (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

3) K-means algorithm.

- Initialize K cluster ~~algorithm~~ centroids $\mu_1, \dots \mu_K \in \mathbb{R}^n$

Repeat {

cluster assignment: for i=1 to m

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

x 와의 거리가 제일 작은 μ_k의 k가 $c^{(i)}$에 대입.

Move centroid: for k=1 to K

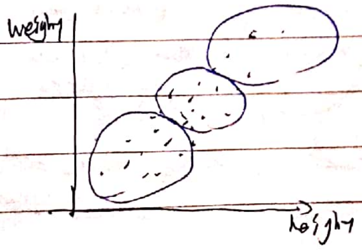
μ_k := average (mean) of points assigned to cluster K

ex) $x^{(1)}, x^{(3)}, x^{(4)}, x^{(6)}$이 μ_2였다면 ⇒ $c^{(1)}=2, c^{(3)}=2, c^{(4)}=2, c^{(6)}=2$

$$\boxed{\mu_2 = \frac{x^{(1)} + x^{(3)} + x^{(4)} + x^{(6)}}{4} \in \mathbb{R}^n}$$

이게 만약 0이 될 경우 그냥 K-1 cluster 나눠놓는게 낫다,

4) K-means for non-separated clusters



← 보통 market clustering

5) Optimization Objective

knowing what is the optimization objective of k-means

→ debug the learning algorithm

→ k-means is running correctly.

make sure

→ help k-means find better costs for this

→ avoid the local optima

$c^{(i)}$ = cluster index

μ_k : cluster centroid k

$\mu_{c^{(i)}}$ = 새로운 centroid

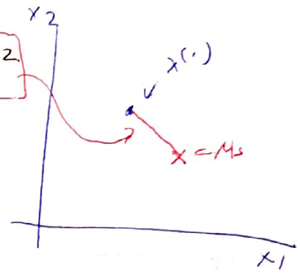
Optimization objective :

이걸 최소화하는 $c^{(i)}, \mu_{(i)}$들을 찾는다...?

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

$$\min_{\substack{c^{(1)}, \cdots, c^{(m)} \\ \mu_1, \cdots, \mu_K}} \underbrace{J(c^{(1)}, \cdots, c^{(m)}, \mu_1, \cdots, \mu_K)}$$

Distortion or Cost function of the k-means algorithm



6) Random Initialization. (실제 있는 점, training sample 중에서 첫번째 centroid를 설정하는 것이 낫다. 안에 꼭 그렇진 않긴 하지만)

Initialization을 어떻게 하느냐에 따라 아래처럼 될 수 있음.

⇒ try multiple initialization. $k = 2 \sim 10$ 일때 최적, 그 이후는 multiple initialization의 효과는 미미함

50~1000

For i = 1 to 100 {

Randomly initialize K-means

Run k-means. Get $c^{(1)}, \cdots, c^{(m)}, \mu_1, \cdots, \mu_K$

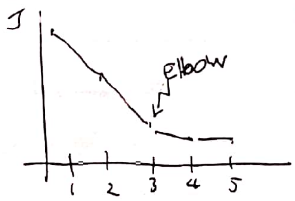
Compute cost function (distortion)

→ $J(c^{(1)}, \cdots, c^{(m)}, \mu_1, \cdots, \mu_K)$ }

Pick clustering that gave lowest cost $J(c^{(1)}, \cdots, c^{(m)}, \mu_1, \cdots, \mu_K)$

7) Choosing the number of clusters (사실 대충 보고 손대중으로 하는게 common) 시각화 해보면서..

① Elbow Method



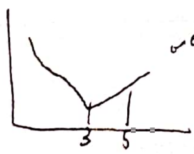
현실은 보통은

↙ 이런식으로 나옴.

→ 비즈니스의 목적에 따라 상황에 따라 그룹수를 정하는것이 일반적

티셔츠를 s size로 보낼지 2 size로 보낼지에 따라서.

간혹

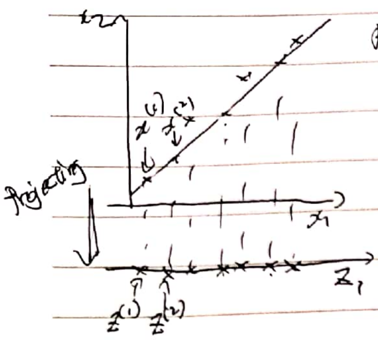


↙ 이런식이면

k-means got stuck in a bad local minimum,

You should try re-running k-means with multiple random initialization, Dimensionality Reduction

1) → Data compression ; compress the data
   └ speed up learning algorithm



Reduce data from 2D to 1D → reduce redundancy

$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$

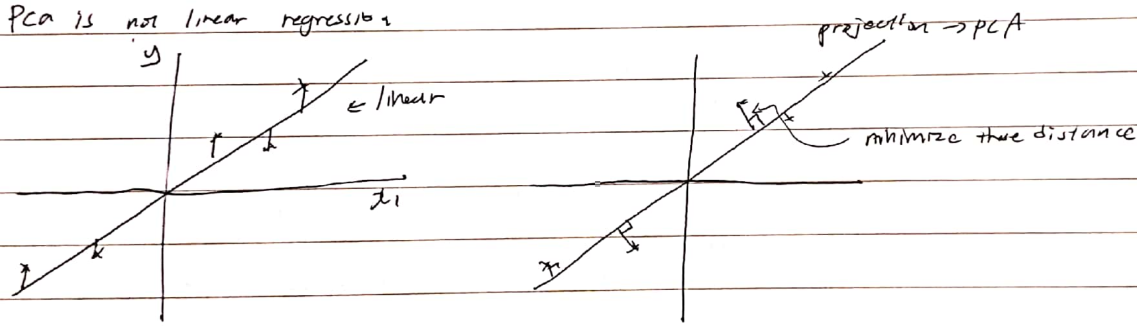
$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$

z_1 → 하나의 숫자만 알아도 위치를 알아낼수 있다 (data 양도 줄고, 속도는 빨라지고)

2) Data visualization

PCA ; Try to find lower dimensional surface onto which to project the data.

Pca is not linear regression



→ there's y to predict using x_i ⓥⓢ → there is no special value to predict.
all of features treated equally

DATA PROCESSING

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

PReprocessing (feature scaling / mean normalization) : [$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$, Replace each $x_j^{(i)}$ with $x_j - \mu_j$

[scale features to have comparable range of values

$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

Reduce data from 2D to 1D

$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$

PCA algorithm.

- Compute "Covariance matrix"

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n} \boxed{\underbrace{(x^{(i)})}_{n\times 1}\underbrace{(x^{(i)})^T}_{1\times n}} \; n\times n$$

- Compute 'eigenvectors' of matrix Σ:

[U,S V] = svd(Sigma);
↳ singular value decomposition

$$U = \left[\begin{array}{ccccc} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(m)} \\ | & | & | & & | \end{array}\right] \quad U \in \mathbb{R}^{n\times n}$$

$\underbrace{u^{(1)}\ u^{(2)}\ u^{(3)}}_{k}$; $u^{(1)},\dots,u^{(k)}$

$x \in \mathbb{R}^n \longrightarrow z \in \mathbb{R}^k$

$$z^{(i)} = \underbrace{\left[\begin{array}{cccc} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{array}\right]}_{n\times k \text{ matrix}}^T x = \underbrace{\underbrace{\left[\begin{array}{c} -(u^{(1)})^T- \\ -(u^{(k)})^T- \end{array}\right]}_{k\times n} \underbrace{x}_{n\times 1}}_{k\times 1}$$

~~$z_j = (u^{(j)})^T x$~~

$z_j = (u^{(j)})^T x$

$z = U_{reduce}^T \cdot x$

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

$z \in \mathbb{R}^k$

$$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T \rightsquigarrow \text{Sigma} = \left(\frac{1}{m}\right) * X' * X;$$

Choosing k.

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

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

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \le 0.01 \ (1\%)$$

0.1

Typically, choose k to be smallest value so that

: 99% of variance is retained.

90%, 변경 안 제출의 가능.

PCA가 어느정도로 잘 되었다는 것을 말해주는 것.

Algorithm.

k=1 부터

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

Check if $\boxed{\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2}} \le 0.01$ 이 아니면 ~~하고~~ k +1 씩 해서 늘려가기.

For given k,

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \le 0.01 \Rightarrow \boxed{\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \ge 0.99}$$

SVD 결과에 기반해서 계산

이게 더 간단. SVD 1번만 하고 보면 되기 때문에.

Applying PCA.

Supervised learning speedup.

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})$.

$x^{(i)} \in \mathbb{R}^{10000}$

Extract inputs: unlabeled dataset: $x^{(1)}, \ldots x^{(m)} \in \mathbb{R}^{10000}$

$\downarrow$ PCA

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

New training set

$(z^{(1)}, y^{(1)}) \cdots (z^{(m)}, y^{(m)})$

Application of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm
- Visualization
  - $k=2$ or $k=3$

Bad use of PCA: To prevent overfitting → 효과는 있지만 좋은 방법이 X

Regularization이 낫다. ⇐ 이건 y를 안 날리기 때문에 가치 있는 정보가 날라가지 않아서 PCA보다 더 낫다.

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

PCA를 무조건 사용하기보단, 일단 원래 data 가지고 해보고 속도가 느리거나 원하는 결과가 잘 안나오면 시도하라.