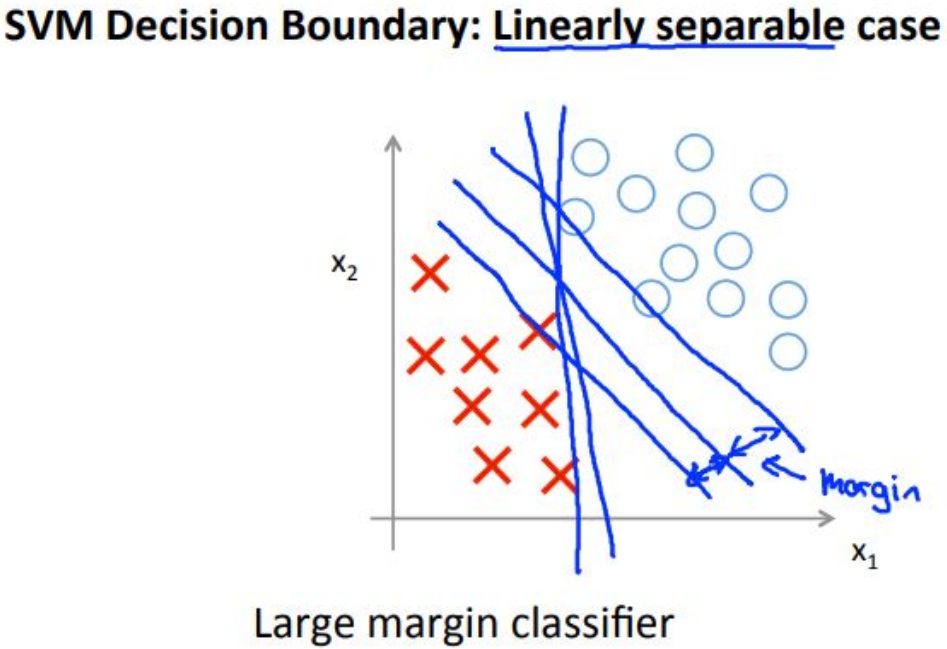
Support vector machine

Large Margin Classfication

왜 하나 - SVM을 위한 더 robust한 feature들을 찾기

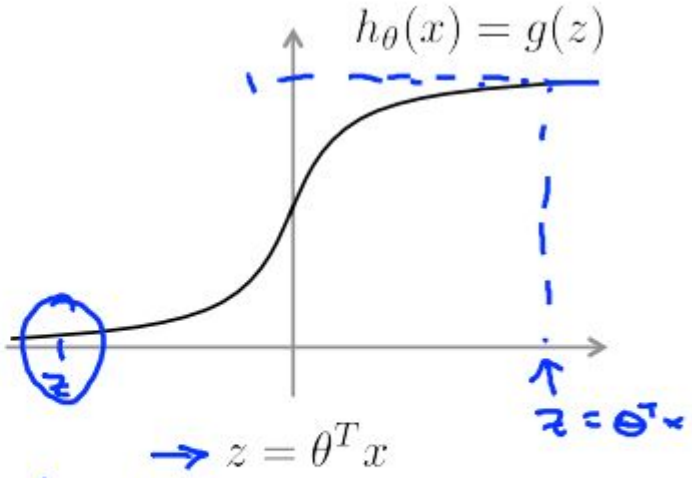
시대의 화두: Robustness, 일희일비 하지 않는다.



어떻게 하나(1)

잊지 말아야 할 기본개념, logistic regression, activation 등등

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



If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

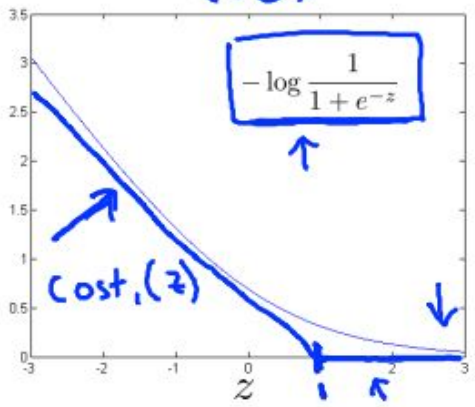
어떻게 하나(2)

최적화 해야할 함수를 아래와 같이 바꾼다.

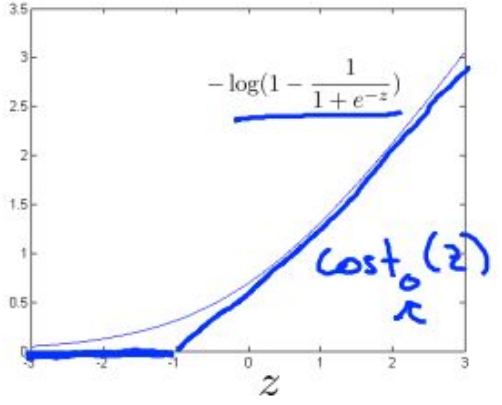
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

If $y=1$ (want $\theta^T x \gg 0$):



If $y=0$ (want $\theta^T x \ll 0$):



어떻게 하나(3)

최적화 해야할 함수를 아래와 같이 바꾼다.

Logistic regression:

Learning 알고리즘

감

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

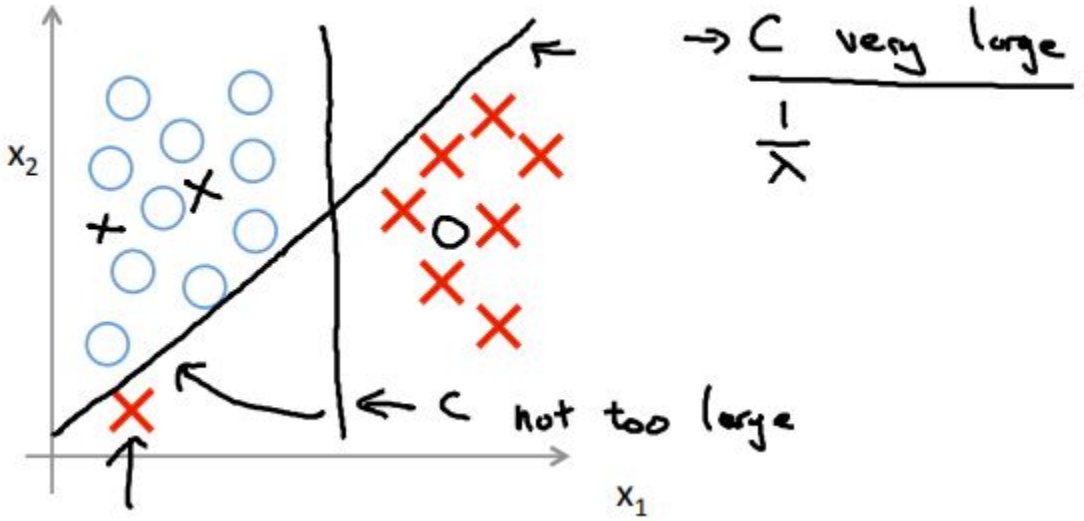
SVM hypothesis

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

어떻게 하나(4)

C 값에 따라서 robustness 가 달라진다.(뭐가 더 좋은 것인가?)

정신건강상 optional 수식설명은 생략



Support vector machine

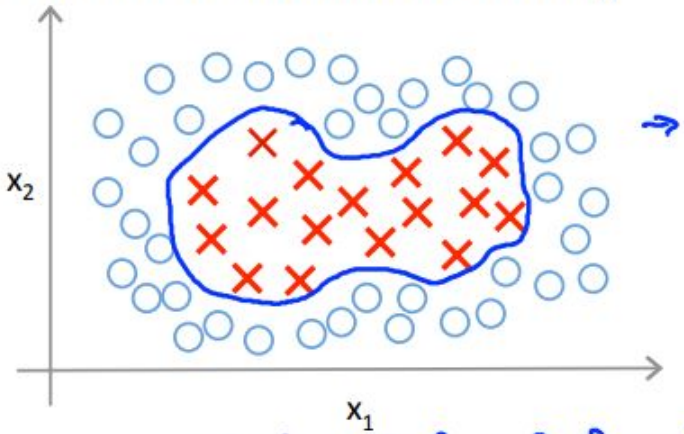
Kernels (Non-linear classifiers)

커널패닉?

왜 하나

High-order polynomial term들이 많으면 비싸고 피곤하다. 어떻게 고르나

Non-linear Decision Boundary



Predict $y = 1$ if

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

$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{otherwise} \end{cases}$

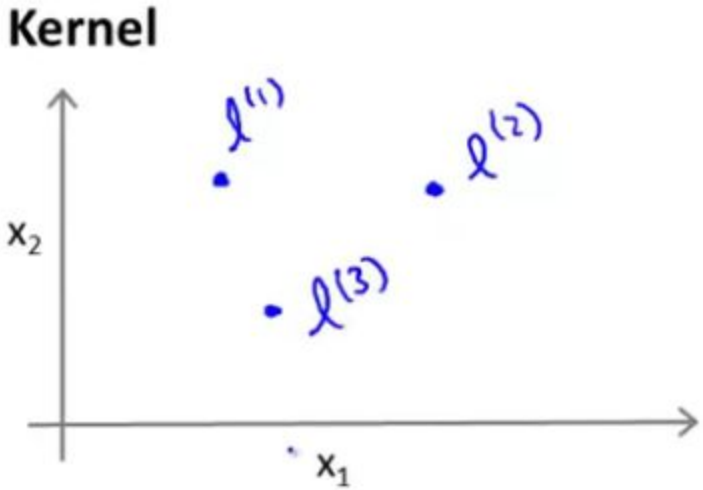
$\rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_1^2, \ldots$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

용어 정의

Kernel ~ Similarity function (여기서는 여러가지 종류 중에 Gaussian kernel 설명)



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

Hypothesis 어떻게 만드나

시그마의 역할은 뒤에서 설명

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

If x if far from $l^{(1)}$:

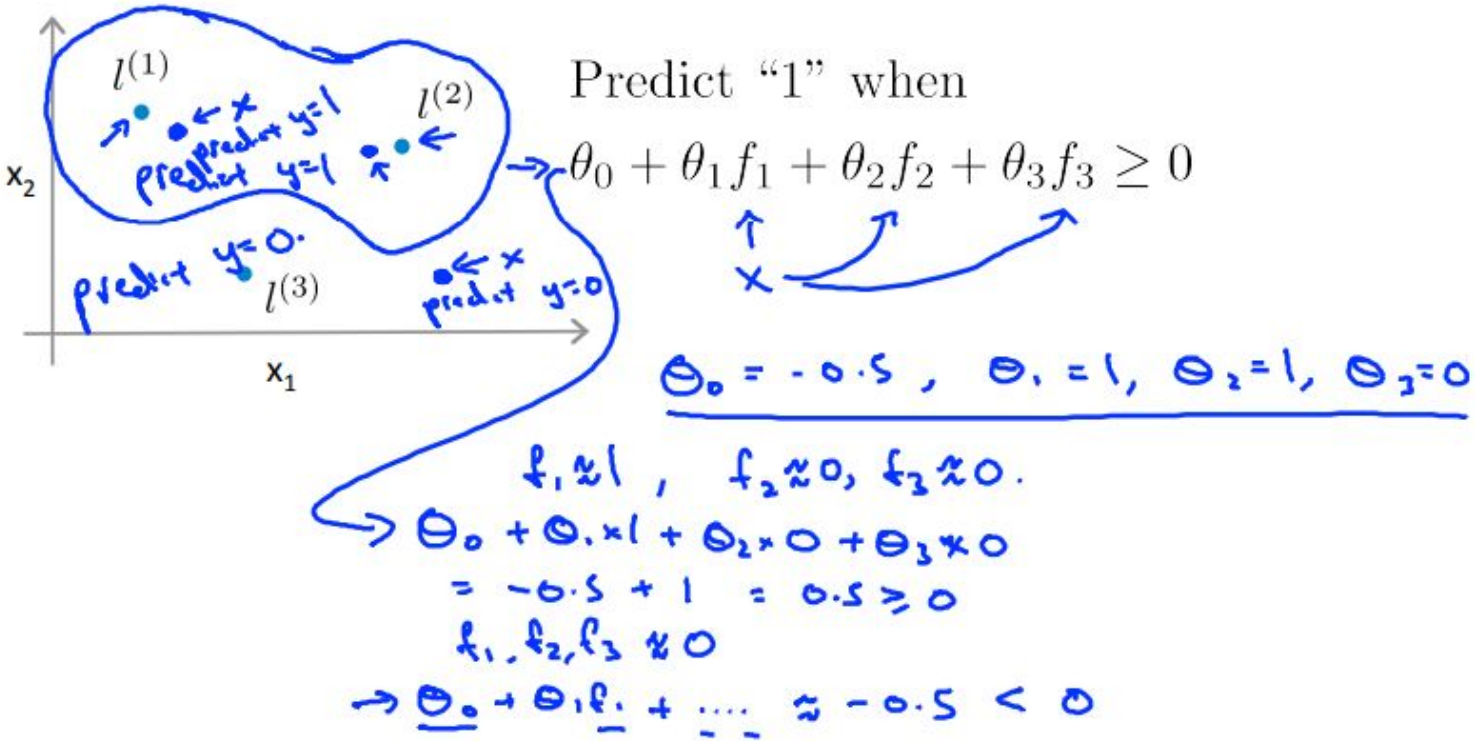
$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

$l^{(1)} \rightarrow f_1$

$l^{(2)} \rightarrow f_2$

$l^{(3)} \rightarrow f_3$.

그래서 어떻게 한다는 것인가? -아하!

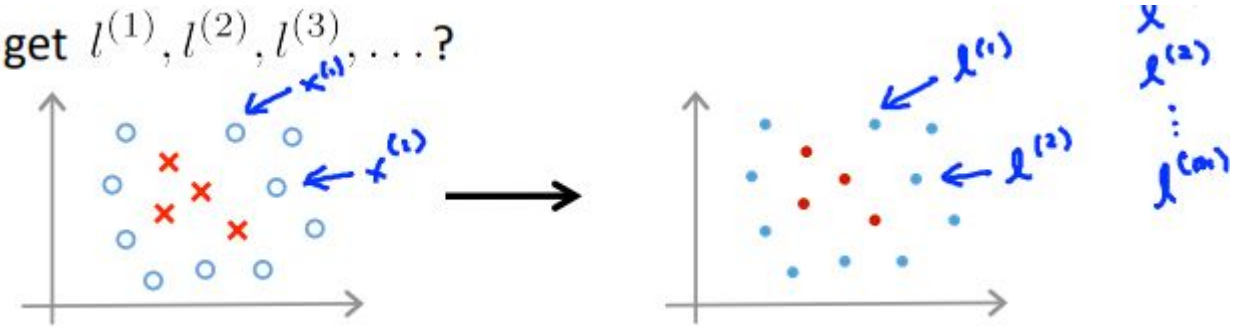


알겠다. 그런데 landmark들은 어디에 찍어야 하나

랜드마크 찾기

당연히 학습데이터(training set, test set)의 xi와 최대한 가까운 곳에 찍어야 한다.

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



→ Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

→ choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

이렇게 하면 결국 $f^{(i)}$를 최적화 하는 문제로 변신

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

→ Predict "y=1" if $\theta^T f \geq 0$

힘조절

Training:

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

SVM parameters:

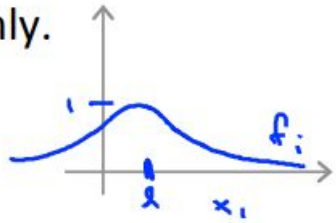
C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance. (small λ)

→ Small C: Higher bias, low variance. (large λ)

σ^2 Large σ^2: Features f_i vary more smoothly.

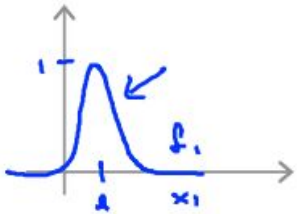
→ Higher bias, lower variance.

$\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



여기서 bias와 variance가 헷갈릴 줄이야.

Question

Suppose you train an SVM and find it overfits your training data. Which of these would be a reasonable next step? Check all that apply.

- [ ] Increase C
- [ ] Decrease C
- [ ] Increase σ^2
- [ ] Decrease σ^2

실제로 어떻게 쓰나

Use SVM software package (e.g. liblinear, libsvm, …) to solve for parameters θ.

Need to specify:

→ Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

Predict "y = 1" if $\theta^T x \geq 0$

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

→ n large, m small

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

Feature는 많은데 학습데이터는 적을 때 -> 무리하지 말고 linear로 가라

→ Gaussian kernel:

$$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right), \text{ where } l^{(i)} = x^{(i)}.$$

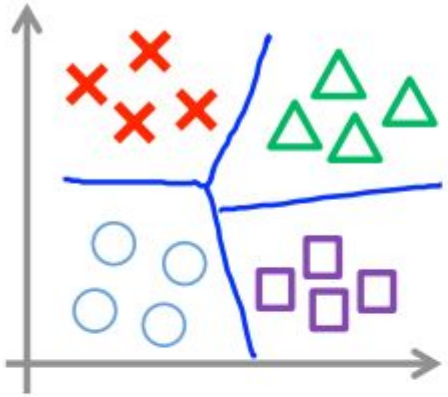
Need to choose σ^2.

$x \in \mathbb{R}^n$, n small

and/or n large

학습데이터가 많아서 복잡하게 fit 해야할 때는 Gaussian Kernel로

Multi-class classification은? → 툴이 해결



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

→ Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

Pick class i with largest $(\theta^{(i)})^T x$

y=1 y=2 ... θ=K

외우자;

Logistic regression vs. SVMs

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

- If n is large (relative to m): (e.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)
  - Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate: ($n = 1 - 1000$, $m = 10 - 10{,}000$)
  - Use SVM with Gaussian kernel
- If n is small, m is large: ($n = 1 - 1000$, $m = 50{,}000+$)

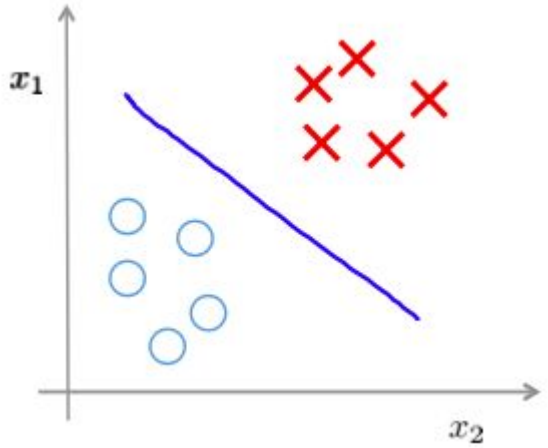
  - Create/add more features, then use logistic regression or SVM without a kernel
- Neural network likely to work well for most of these settings, but may be slower to train.

Unsupervised Learning

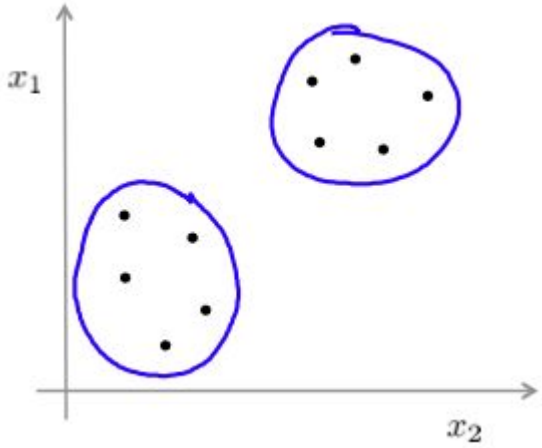
Clustering

Intro

Supervised



Unsupervised



Applications of clustering

- Market segmentation
- Social network analysis
- Organize computing clusters
- Astronomical data analysis

K means Algorithm

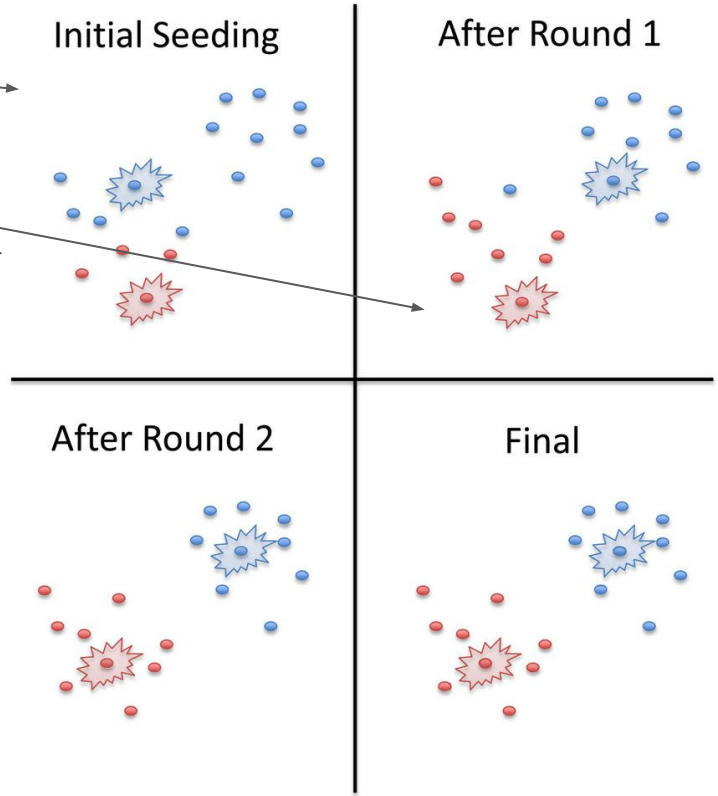
1. cluster assignment step
2. move centroid step

평균이 움직이지 않을 때까지 위 2단계를 반복

Input

1. 파라미터 k
2. X들(y가 없음)

K 고르기 어려울 때

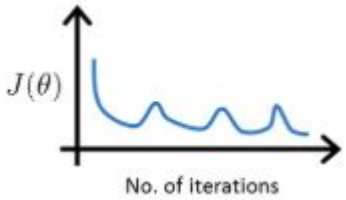


Optimization Objective

목표 최적화하는 이유

1. 알고리즘을 디버그하고 kmeans 가 잘 되고 있는지 확인할수있음

2. kmeans 가 find better costs, avoid local optima 하는데 도움



Optimization Objective

Optimization objective:

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

$$\min_{c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$$

Distortion

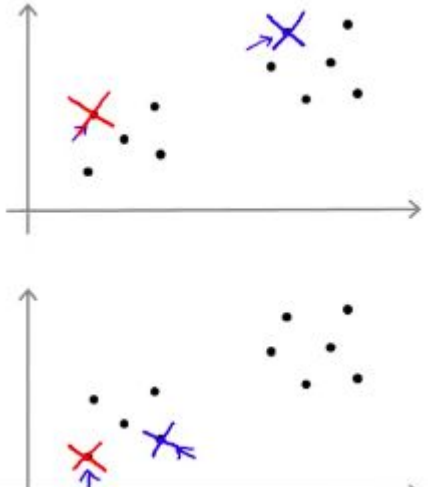
k-means algorithm as trying to optimize this cost function J

random initialization

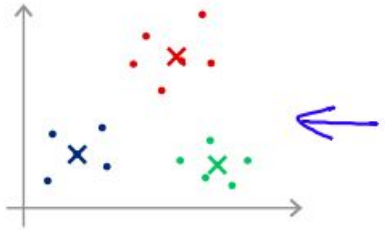
여러가지 방법이 있음

방법1

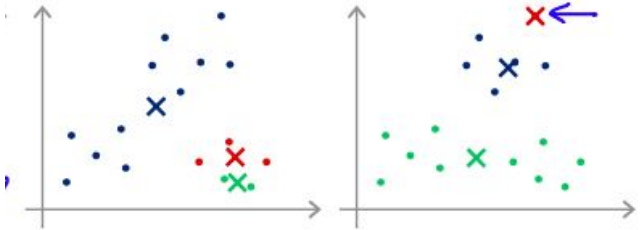
- 훈련 예시개수m보다 작게 K를 설정
- 랜덤하게 K개의 의 점을 골라 첫 centroid이 됨



random initialization



우상단: global optima(good)



아래 두개: bad local optima

문제 해결방법 : 시도를 여러번 하기

50~1000번 정도의 전체 kmeans 을 돌리고 그중 cost J 가 낮은걸 고르기

+K가 10보다 클경우에는 여러번 하지 않아도 잘될 것

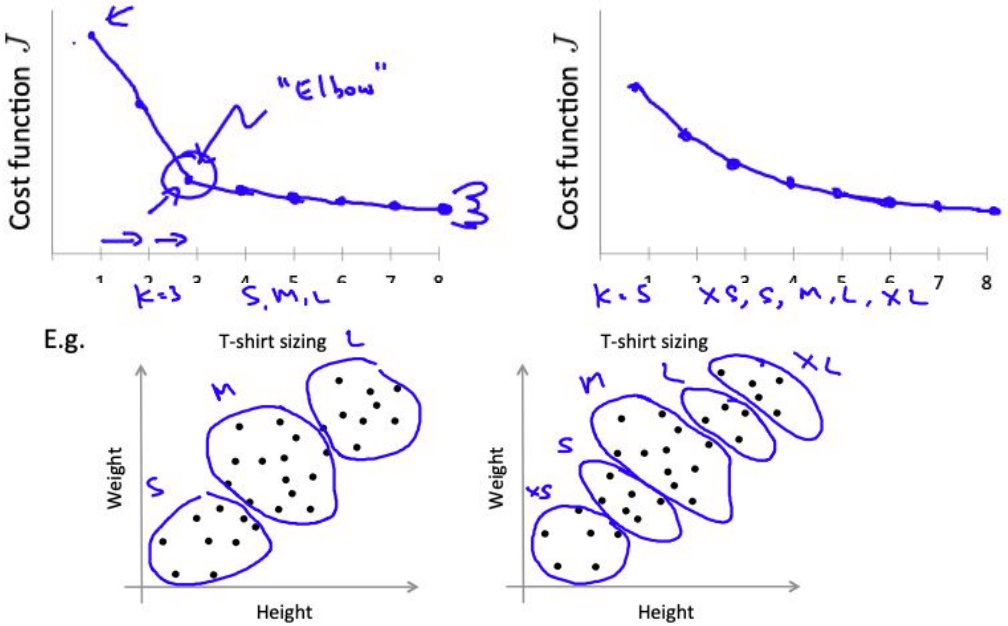
특히 2~4개의 K인 경우에는 여러번 돌리는게 좋을 것

choosing number of clusters

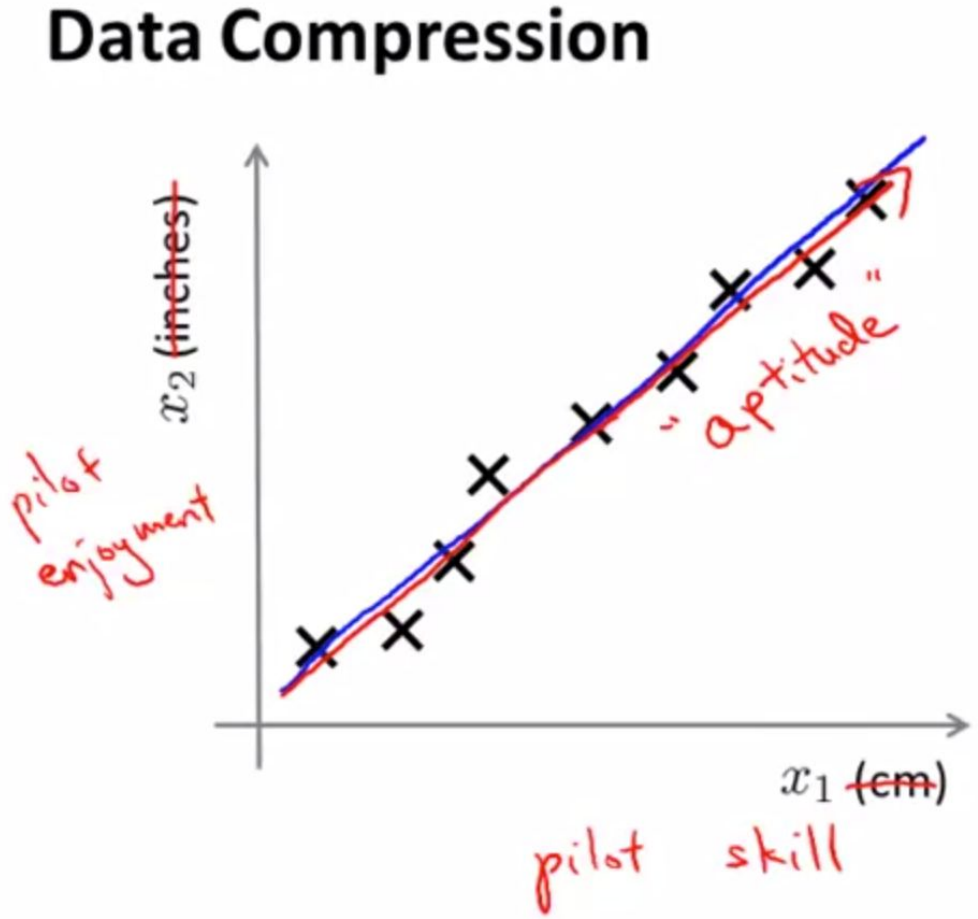
그래프로 보고/손으로 고르는게 제일 좋음. 별 묘수 없음

맞는 하나의 정답은 없음.

- elbow methods
- later purpose



Data Compression 2D -> 1D



Reduce data from 2D to 1D

2D -> 1D Data Reduction 예시

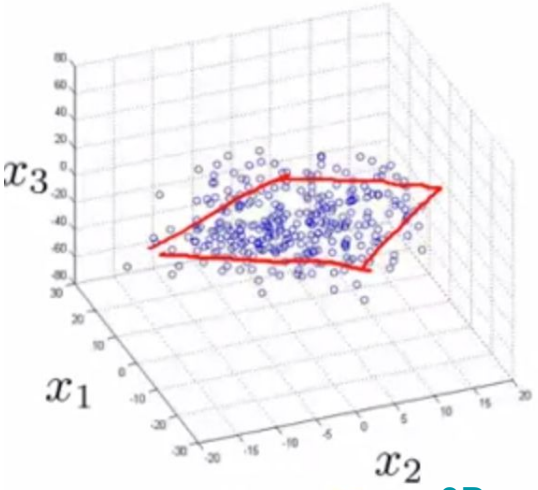
- Inch 와 cm
- Pilot skill 과 Pilot enjoyment
- 두 feature로 그래프를 그리면 Aptitude(경향성)를 보임

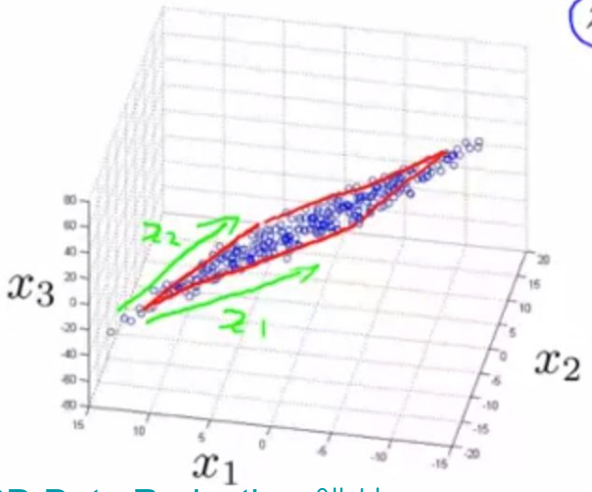
Data Compression 3D -> 2D

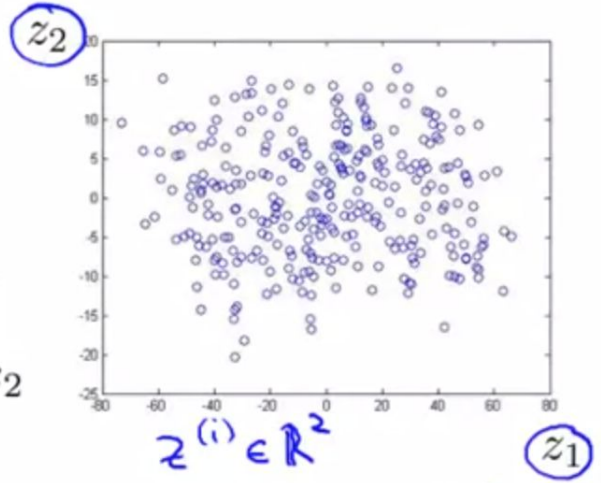
Data Compression

10000 → 100D

Reduce data from 3D to 2D







$x^{(i)} \in \mathbb{R}^3$

3D -> 2D Data Reduction 예시

- 3D data 를 2D로 프로젝션(평면화)
- 평면화한 데이터를 2D Plot에 배치
- 2 Dimensional vectors

$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ $z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$

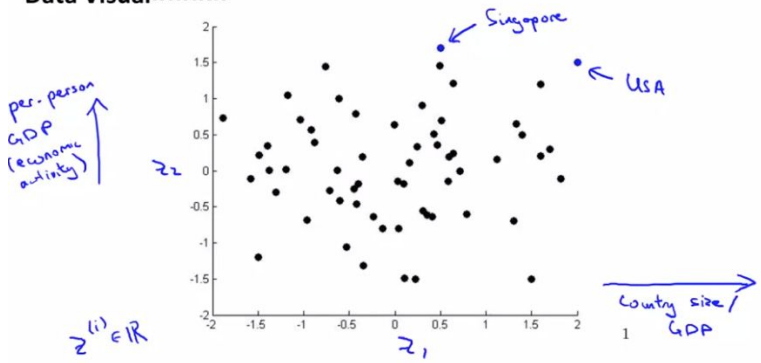
Visualization

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

Data Visualization

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

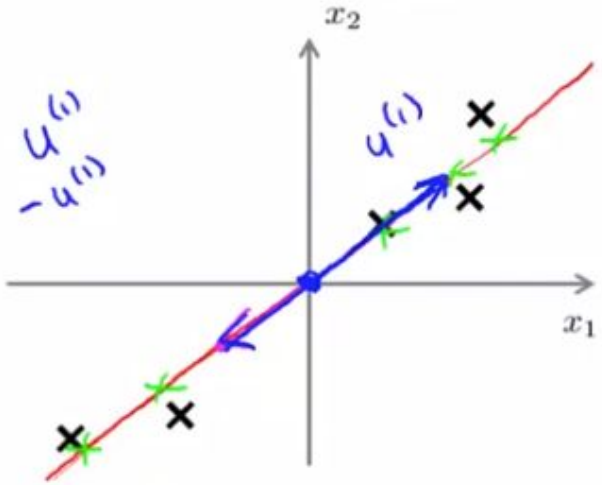
Data Visualization

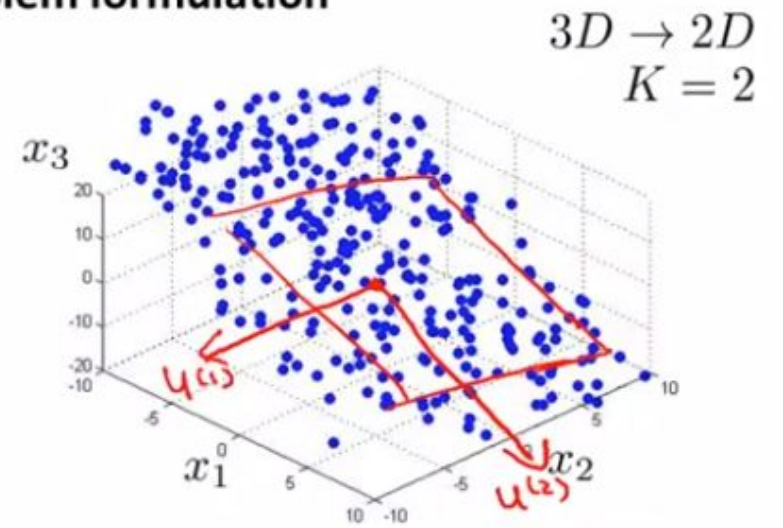


- 50개의 dimension 을 가진 데이터를 2개의 dimension 으로 압축
- 압축된 2개의 dimension 은 가장 대표적인 수치들을 대변함

Data Compression 2D -> 1D

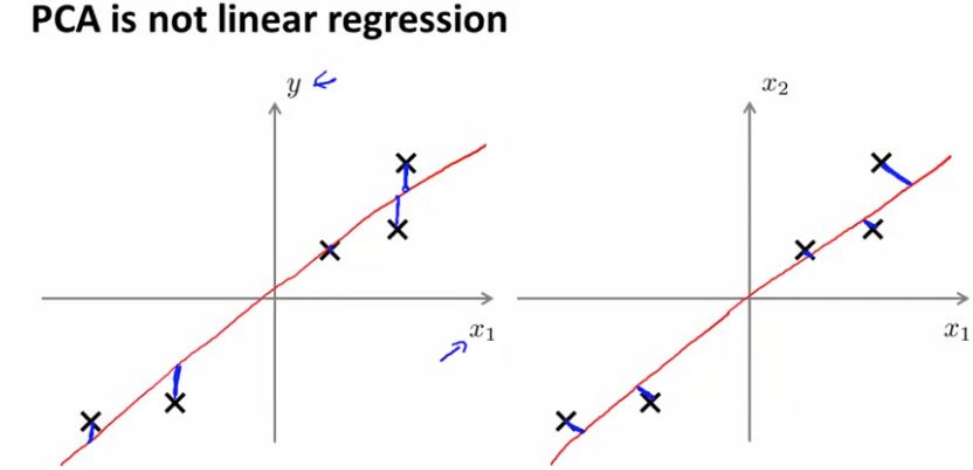
Principal Component Analysis (PCA) problem formulation





- Projection error: Vector 값과 sample의 거리 (파란 선 길이)
- PCA 하기 전 필수: mean normalization to have 0 mean
- 2D to 1D
  - (Squared) Projection error를 최소화하는 1D vector를 찾을 것
- N-D To K-D
  - (Squared) Projection error를 최소화하는 k-D vector를 찾을 것

PCA VS Linear Regression



- Linear regression: 포인트간 거리 계산 (x1의 예측값 - 실제값)
- PCA: 최단거리(직각) 계산
- PCA는 LR과 달리 예측하려는 Y 값이 없음

Mean normalization for PCA

Data preprocessing

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 $x_j - \mu_j$.

If different features on different scales (e.g., x_1 = size of house, x_2 = number of bedrooms), scale features to have comparable range of values.

- PCA 전 mean normalization의 필요성
  - 서로 다른 scale을 비교하기 위해
  - x - average(x): zero mean
  - / Standard deviation
  - / max(x) - min(x)

Mean normalization [edit]

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

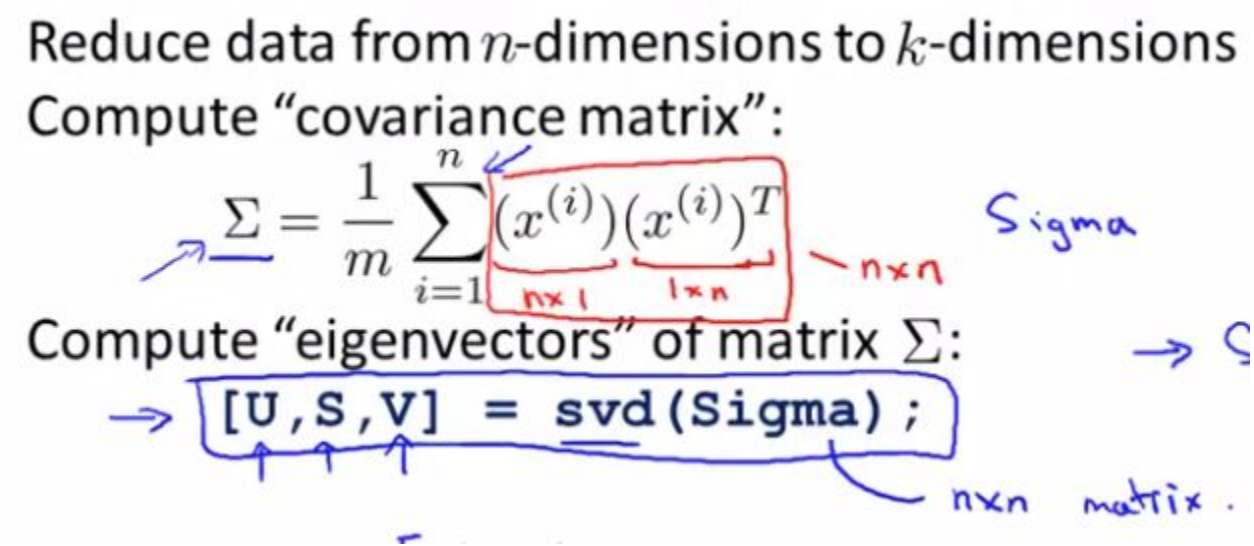
Standardization (Z-score Normalization) [edit]

See also: Standard score

In machine learning, we can handle various types of data, e.g. subtracting the mean in the numerator) and unit-variance. This of calculation is to determine the distribution mean and standa

$$x' = \frac{x - \bar{x}}{\sigma}$$

PCA Algorithm 1/2



- PCA에서 Principal Components(u1, u2…) 구하기
  - Covariance Matrix(시그마 Σ)
  - Σ 의 Eigenvector (또는 Singular value decomposition)
    - Covariance matrix 는 언제나 Symmetric positive semi-definite matrix 의 특성을 가지고 있기 때문
  - x(i): n x 1 / x(i)T = 1 x n
  - Σ: n x n matrix

Linear Algebra Recap: Eigenvalues and EigenVectors

1. 고유값, 고유벡터란?

고유값(eigenvalue), 고유벡터(eigenvector)에 대한 수학적 정의는 비교적 간단하다.

행렬 A를 선형변환으로 봤을 때, 선형변환 A에 의한 변환 결과가 자기 자신의 상수배가 되는 0이 아닌 벡터를 고유벡터(eigenvector)라 하고 이 상수배 값을 고유값(eigenvalue)라 한다.

즉, n x n 정방행렬(고유값, 고유벡터는 정방행렬에 대해서만 정의된다) A에 대해 $Av = \lambda v$를 만족하는 0이 아닌 열벡터 v를 고유벡터, 상수 λ를 고유값이라 정의한다.

$$A\mathbf{v}=\lambda\mathbf{v} \quad \text{---(1)}$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}\begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}=\lambda\begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} \quad \text{--- (2)}$$

좀더 정확한 용어로는 λ는 '행렬 A의 고유값', v는 '행렬 A의 λ에 대한 고유벡터'이다.

즉, 고유값과 고유벡터는 행렬에 따라 정의되는 값으로서 어떤 행렬은 이러한 고유값-고유벡터가 아에 존재하지 않을수도 있고 어떤 행렬은 하나만 존재하거나 또는 최대 n개까지 존재할 수 있다.

Source: https://darkpgmr.tistory.com/105

Example : 2D

$$A = \begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix}$$

Step 1

$$\det(A - \lambda I) = 0$$

$$\downarrow$$

$$\begin{vmatrix} 1-\lambda & 2 \\ 3 & -4-\lambda \end{vmatrix} = (1-\lambda)(-4-\lambda) - 2 \cdot 3$$

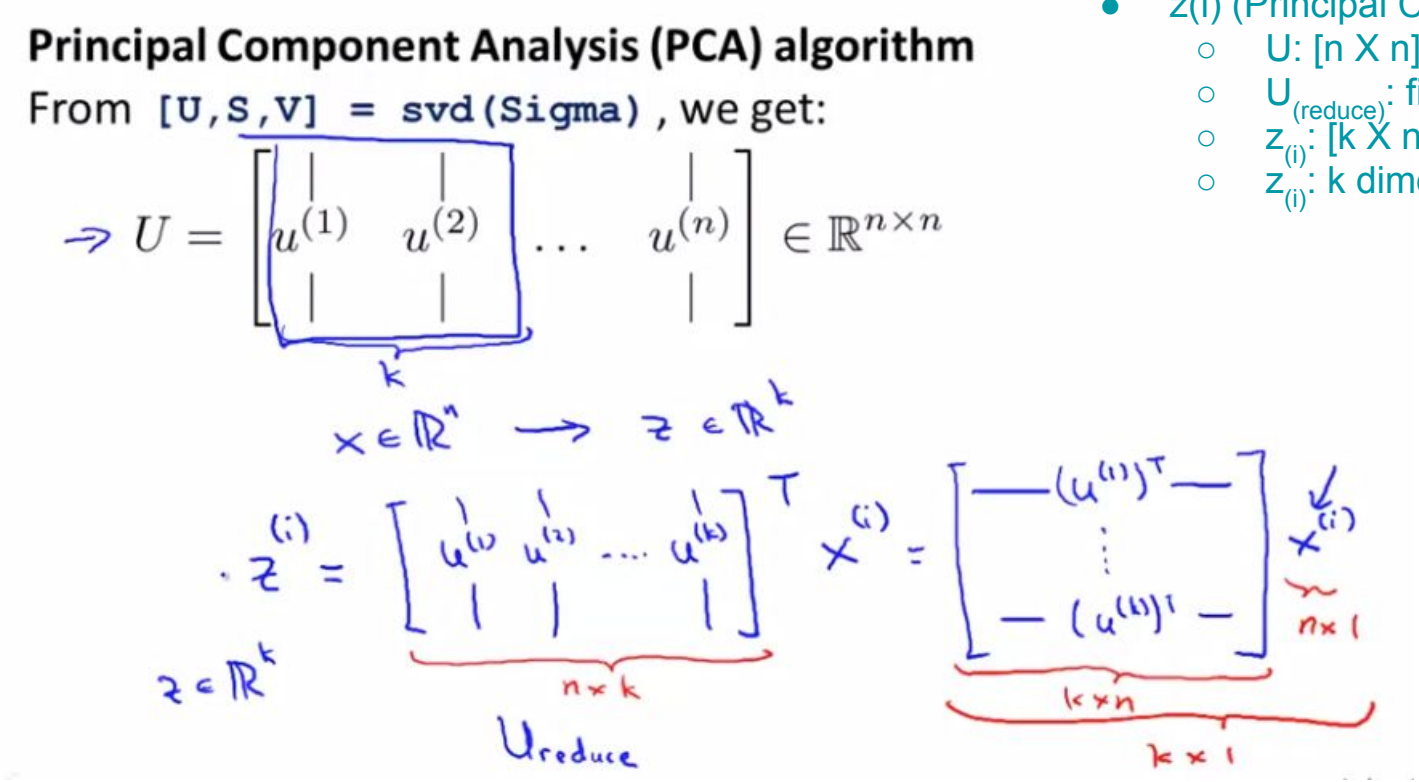
$$= -4 - \lambda + 4\lambda + \lambda^2 - 6$$

$$= \lambda^2 + 3\lambda - 10$$

$$= (\lambda - 2)(\lambda + 5) = 0$$

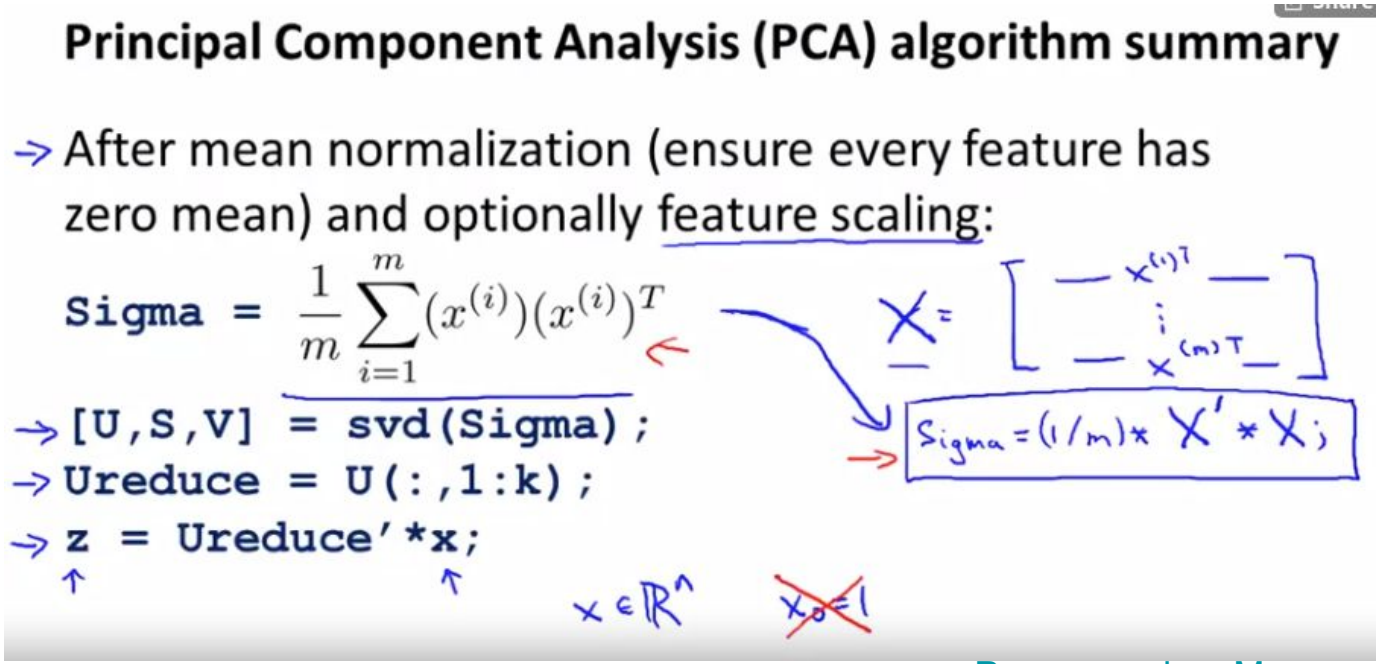
$$\therefore \lambda_1 = 2, \ \lambda_2 = -5$$

PCA Algorithm 2/2



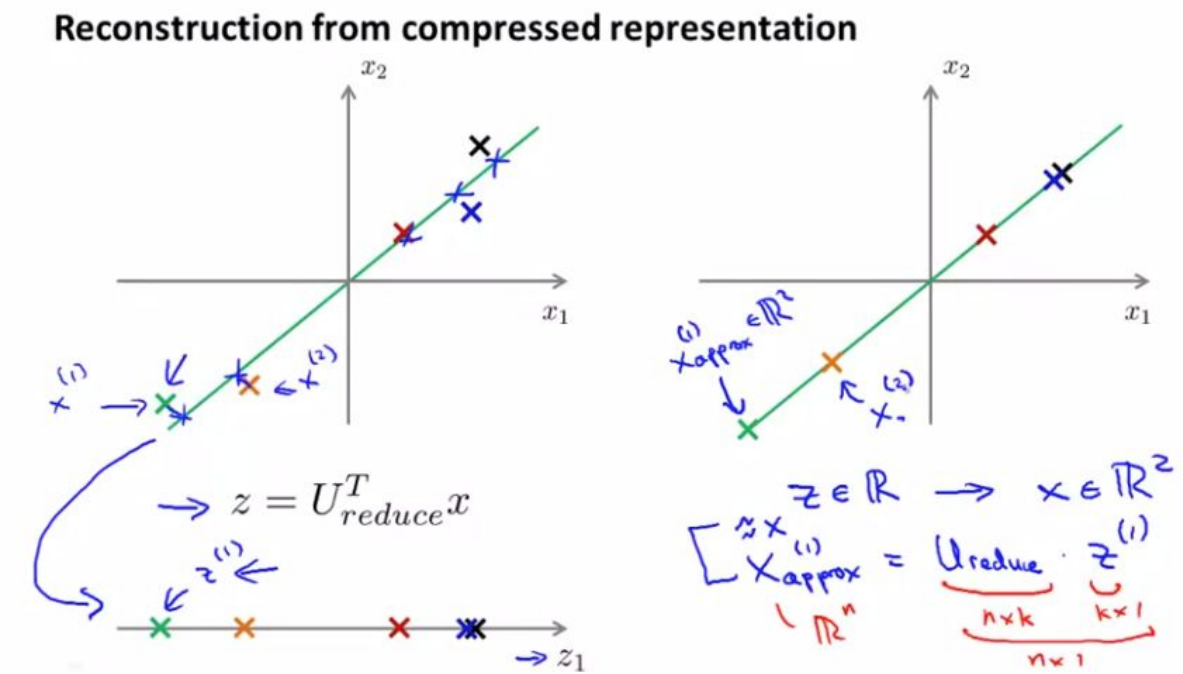
- z(i) (Principal Components)
  - U: [n X n]
  - $U_{(reduce)}$: first k columns from U
  - $z_{(i)}$: [k X n] x [n x 1] => [k x 1]
  - $z_{(i)}$: k dimensional vector

PCA Algorithm Summary



- Preprocessing: Mean normalization or feature scaling
- Covariance matrix
- SVD (or eigenvector) to get U
- U_{reduce} = U(:, 1:k)
- $z = U_{reduce}^{t} \times x$ (k dimensional vector)
-

Reconstruction from compressed Representation



- 압축된 데이터를 복원
- z가 주어졌을 때 x_{approx} 를 구하는 방법
  - $z = U_{reduce}^{t} \times x$ (k dimensional vector)
  - $X_{approx}\ (n \times 1) = U_{reduce}\ (n \times k) \times z^{(1)}\ (k \times 1)$

Choosing the number of principal components

Choosing k (number of principal components)

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

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

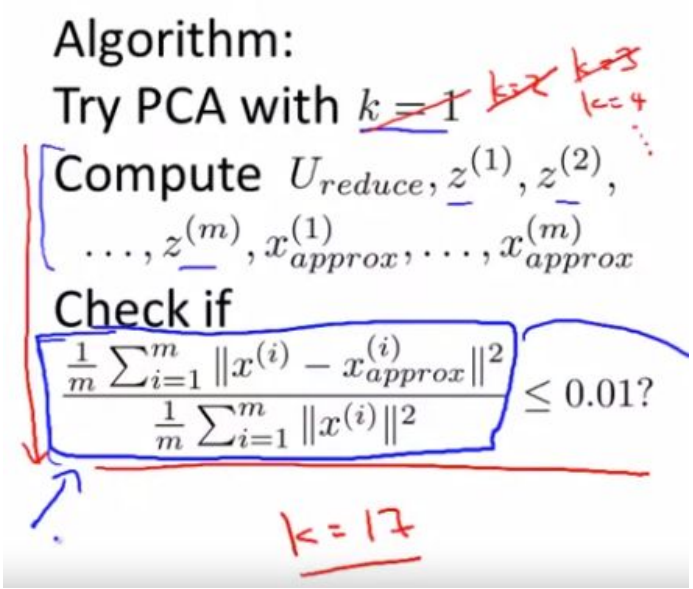
Typically, choose k to be smallest value so that

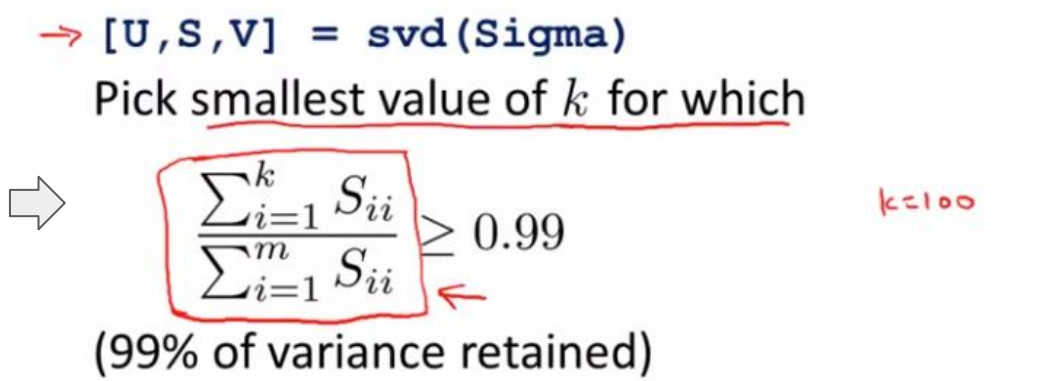
$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01 \quad (1\%)$$

"99% of variance is retained"

- k 값의 선택(Principal components의 수)
  - 평균 squared projection Error / 데이터 전체의 분산
  - <= 0.01 보다 작거나 같으면
    - 99% 의 분산이 함유되었다고 해석
  - 0.05, 0.10, 0.15 도 가능

Choosing the number of principal components





- 기존 공식이 너무 복잡함
- SVD [U, S, V] 에서 S를 이용
- S: Square matrix, diagonal matrix
- 1 - (sum of S_{ii} from i to k / Sum of S_{ii} from i to n)
  - 이전 슬라이드의 variance 구하는 공식과 같음

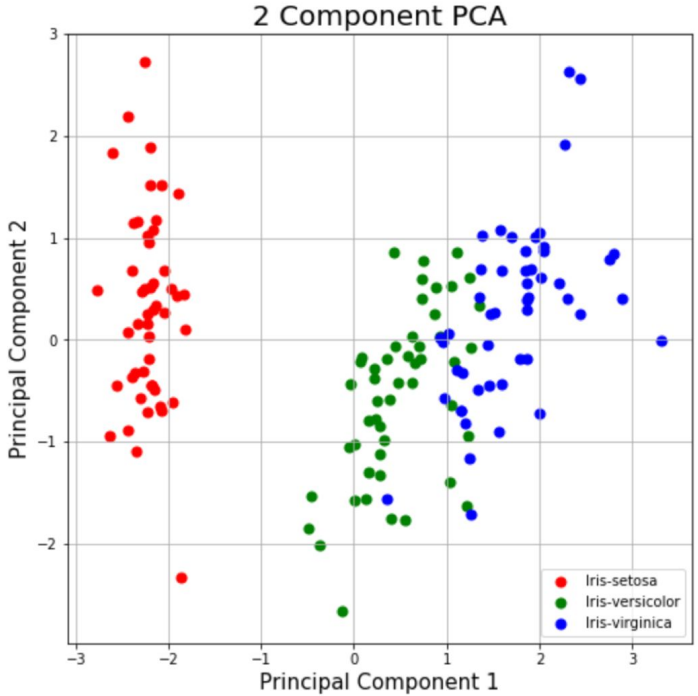
Application of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm ⇐

    Choose k by % of variance retain

- Visualization

  $k=2$ or $k=3$



Bad use of PCA

Bad use of PCA: To prevent overfitting

→ Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$.

Thus, fewer features, less likely to overfit.

Bad!

- Overfitting 방지를 위해 PCA를 사용하면 안되는 이유
  - PCA 는 x 만 가지고 압축함 (y 는 포함하지 않음)
  - Variance retained가 99% 라도 중요한 정보가 생략될 수 있음

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- ~~Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$~~
- Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x^{(i)}_{test}$ to $z^{(i)}_{test}$. Run $h_\theta(z)$ on $\{(z^{(1)}_{test}, y^{(1)}_{test}), \ldots, (z^{(m)}_{test}, y^{(m)}_{test})\}$

- PCA 적용 전 RAW data 부터 시도해볼 것
- Raw data 가 너무 크거나 메모리가 모자라는 등 확실한 이유가 있을 경우에만 PCA 를 쓸 것