Feature and Descriptor – part 2

Byeongjoon Noh

Dept. of AI and Bigdata, SCH Univ.

powernoh@sch.ac.kr

Contents

2. Descriptor

Concept

아래 두 사진이 같은/다른 사진인지 판단하는 방법?

- ➔ 같은 위치에 있는 요소(특징)가 모두 같으면 같은 사진

특징: edge, local feature, region, etc.

비교 ➔ Matching

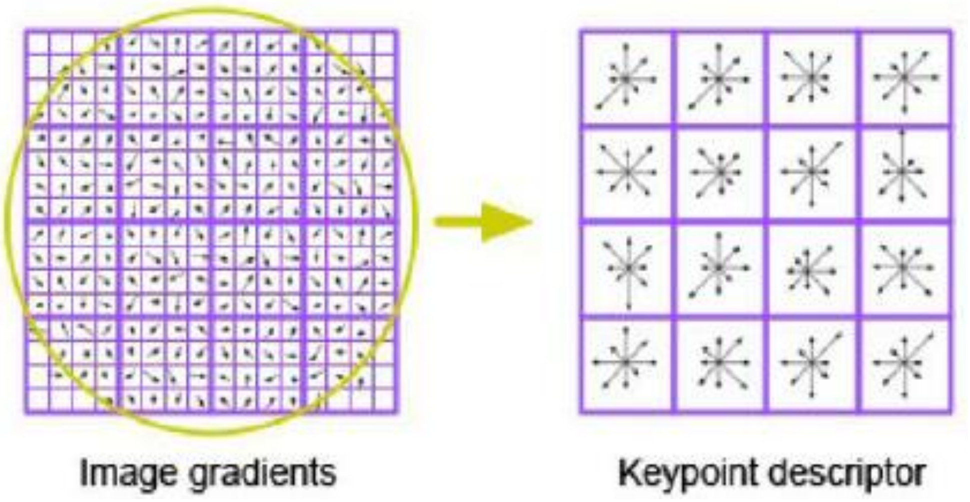
Concept

Feature

- A piece of data that is important for completing the computations necessary for a specific application
- Point, edge, local feature, region, etc.. ➔ matching하기에는 빈약한 정보
- 검출된 특징 내부 또는 주위를 상세히 조망하여 풍부한 정보를 추출할 필요가 있음
  - ➔ 특징의 성질을 기술 (Description)

Descriptor

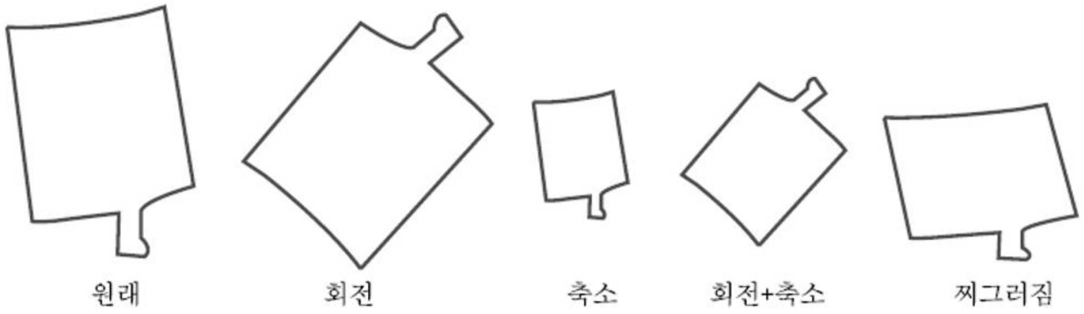
- 사물을 인식하기 위해 (목적에 맞게) 영상으로부터 유의미한 feature를 추출한 feature들의 집합 (Feature descriptor)
- ➔ Feature descriptor의 표현: <u>feature vector</u>



Descriptor

Requirement

- 높은 분별력
- 다양한 변환에 불변
  - 기하 불변성, 광도 불변성
  - 변환에도 불구하고 같은(유사한) 값을 갖는 feature vector 추출
  - Example) Descriptor: 면적 ➔ Scale이 변하면 무용지물



- Feature vector의 차원 문제
  - 차원 (dimension)이 낮을 수록 계산 속도가 빠름
  - ➔ Feature space를 줄여야 함

주어진 문제에 따라 불변/공변에 대한 특징을 잘 추출해야 함

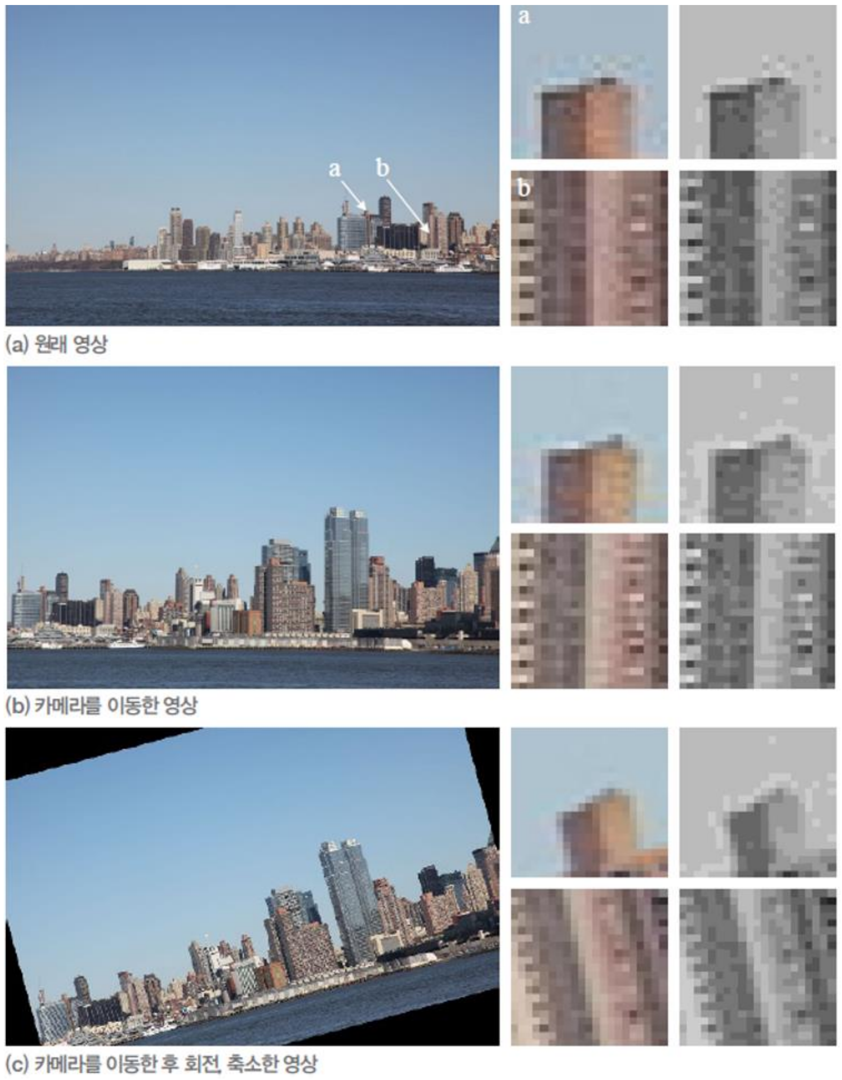
Interest point descriptor

Definition

- Interest point(관심점)을 위한 descriptor
- 관심점으로 선정된 화소(영역)에 대해
  올바른 matching을 위한 descriptor

Interest point descriptor extraction

- 특징점 주위를 살펴
  풍부한 정보를 가진 descriptor를 추출
- (y, x)를 중심으로 window를 씌우고,
  window 내부를 살펴봄
- 고려사항 ➔ Window에서 어떤 것을 볼 것인가?
  - Window의 모양, 크기, 내부 정보 등



21x21 window

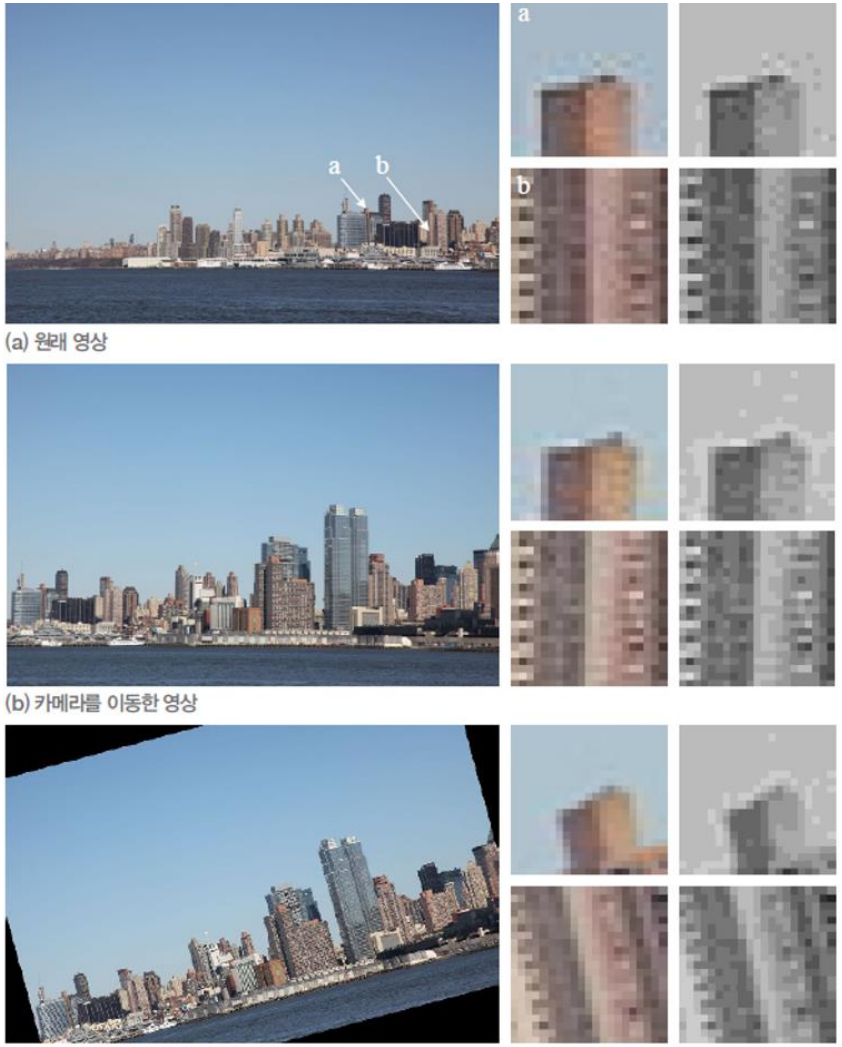
Interest point descriptor

Interest point descriptor extraction

- 카메라의 위치 이동만 발생한 경우 (b)
  - ➔ any descriptors not bad
  - Ex) Harris corner (no scale information)

- 회전, 축소 등의 변화가 발생한 경우 (c)
  - ➔ Very bad
  - Scale σ에 따라 window 설정이 필요함
  - Ex) SIFT, SURT (scale information)

- Scale/rotation 등에 불변한 descriptor 고안 필요



(a) 원래 영상

(b) 카메라를 이동한 영상

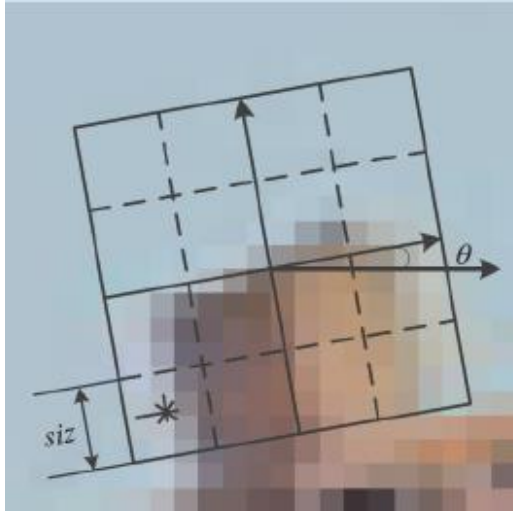
(c) 카메라를 이동한 후 회전, 축소한 영상

21x21 window

SIFT descriptor

SIFT keypoint (interest point)

- (In previous chapter)
- Octave o
- Octave 내 scale σ_o
- Octave 내 영상에서의 위치 (r, c)



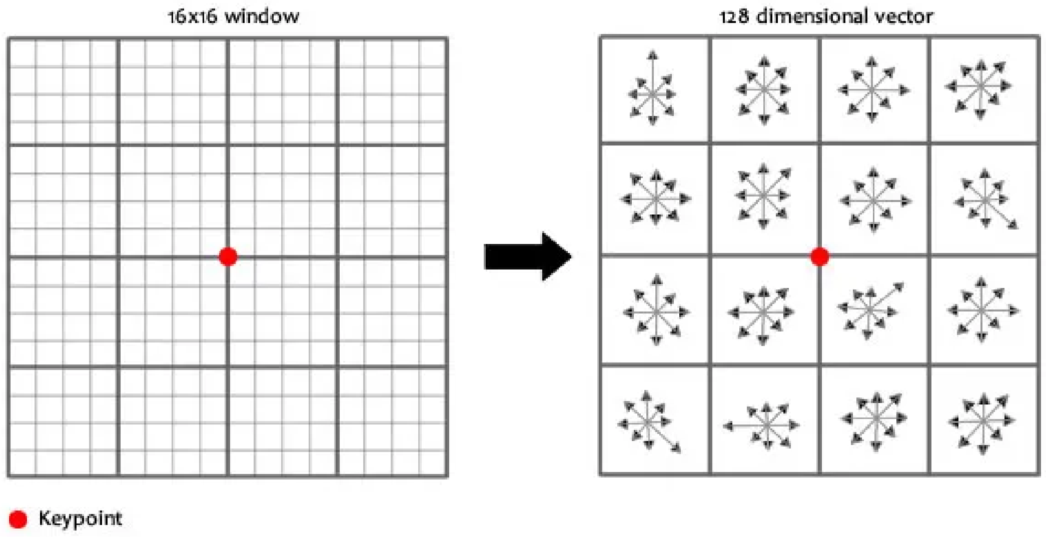
Invariant character of SIFT descriptor

- Scale invariant
  - Window를 o, σ_o, (r, c)에 적용
- Rotation invariant
  - 지배적인 방향 계산 ➔ 이 방향으로 window 적용
- 광도 invariant ➔ feature vector $\mathbf{x}$를 $||\mathbf{x}||$로 나누어 정규화

SIFT descriptor extraction algorithm

Algorithm

- Window를 4x4의 16개의 block으로 분할
  - 각 block은 gradient 방향 히스토그램 계산
  - Gradient 방향은 8개로 quantization
- 4x4x8=128차원 feature vector **x** 획득
- 화소 값은 interpolation (보간법)으로 계산



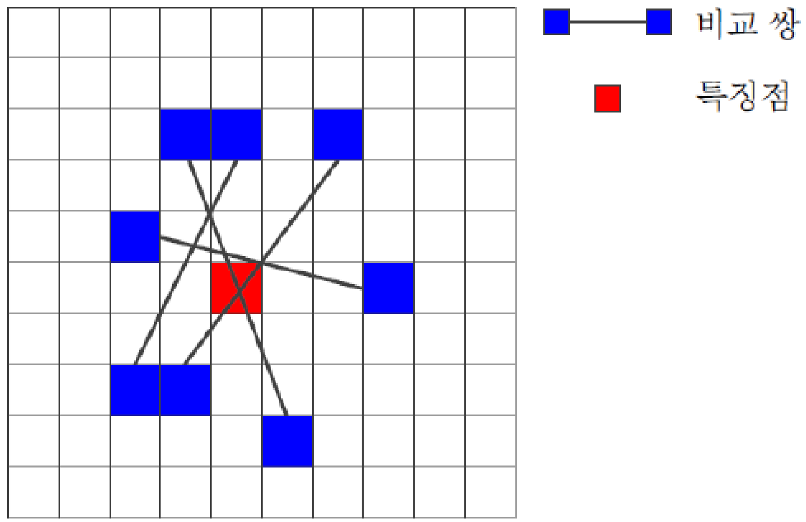

```
kp, des = sift.detectAndCompute(gray, None)

print('keypoint:',len(kp), 'descriptor:', des.shape)
print(des)
```

```
keypoint: 2290 descriptor: (2290, 128)
[[ 29.   23.   15. ...  15.    1.    2.]
 [ 45.   14.    1. ...   0.    0.  134.]
 [  0.    0.    0. ...   0.   15.    5.]
 ...
 [ 12.    0.    1. ...   1.    0.    0.]
 [ 16.    1.    1. ...   1.    2.    0.]
 [  0.    0.    0. ...   0.    0.   15.]]
```

Binary descriptor

이진 기술자

- 비디오 처리 등 빠른 matching을 위해 feature vector를 이진열로 표현
  - 비교 상의 대소 관계에 따라 0 or 1
  - 비교 쌍을 구성하는 방식에 따라 여러 변형



- SIFT 기술자 대비 작은 feature space
  - SIFT 특징점 1개 = 512 Bytes 라면,
    Binary descriptor = 512bit (8배 적음)
- Hamming distance를 이용하여 빠르게 수행함
  - * Hamming distance: 1101과 1011의 hamming distance = 2

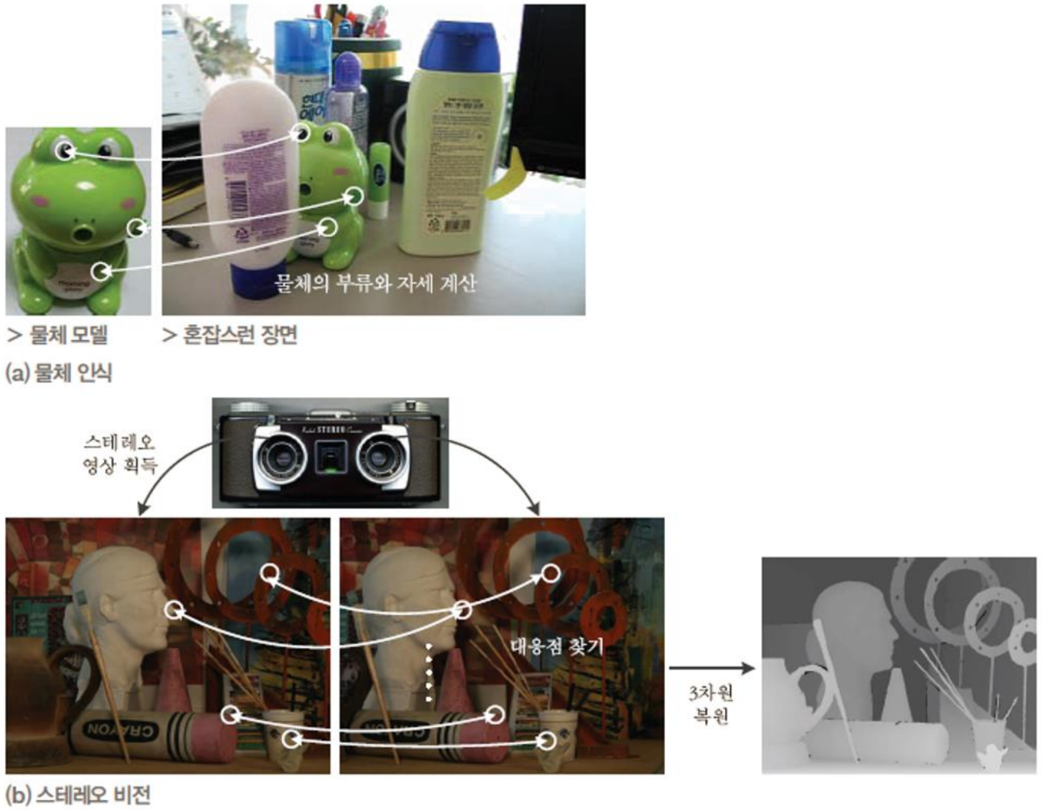
3. Matching

Preview

Matching

- 어떤 대상을 다른 것과 비교하여 같은 것인지 알아내는 과정
- 여러 가지 문제를 해결하기 위해 필요 (물체 인식, 자세 추정, 카메라 캘리브레이션 등)

- 매우 까다로운 문제
  - 두 영상 모두 특징점이 많아
    후보쌍이 매우 많음
  - 잡음이 섞인 descriptor



Matching 전략

Problem definition

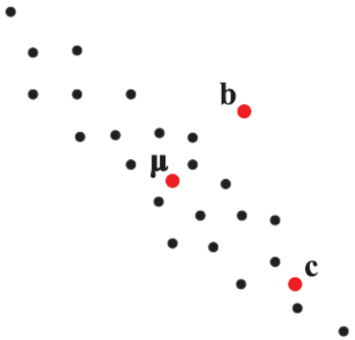
- 두 영상에서 추출한 feature vector set $A = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m\}$, $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m\}$에서 같은 물체의 같은 곳에서 추출된 $\mathbf{a}_i$ $\boldsymbol{b}_j$ 쌍을 모두 찾는 문제

- 매칭을 적용하는 다양한 상황
  - 물체 인식 ➔ 물체의 모델 영상이 A, 장면 영상이 B
  - 물체 추적 ➔ 이전 영상에서의 물체 A, 현재 영상에서의 물체 B

- Simple strategy
  - mn개 쌍 각각에 대해 거리를 계산하고, 그 거리가 임계값보다 작은 쌍을 모두 취함

Matching 전략

거리 기반의 전략



← μ는 b와 c 중 누구와 더 가까운가?

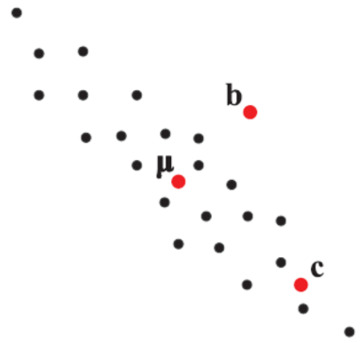
- Manhattan distance

$$d(\mathbf{a},\mathbf{b}) = \sum_{k=1,d} (a_k - b_k) = \|\mathbf{a}-\mathbf{b}\|_1$$

- Euclidian distance

$$d(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{k=1,d} (a_k - b_k)^2} = \|\mathbf{a}-\mathbf{b}\|_2$$

Matching 전략



← μ는 b와 c 중 누구와 더 가까운가?

거리 기반의 전략

- Mahalanobis distance
  - 평균과의 거리가 표준편차의 몇 배인지를 나타내는 값
  - 공분산 행렬 Σ를 이용하여 확률분포를 고려

$$d(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})\mathbf{\Sigma}^{-1}(\mathbf{a}-\mathbf{b})^{\mathrm{T}}}$$

  - * 공분산 행렬 (Covariance matrix, Σ)
    - 변수들 사이의 공분산을 행렬 형태로 나타낸 것

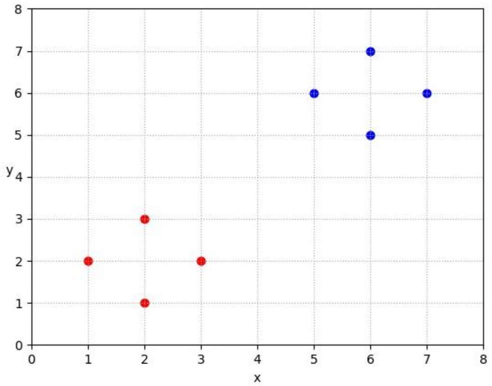
$$\Sigma = \begin{pmatrix} cov(\mathbf{x},\mathbf{x}) & cov(\mathbf{x},\mathbf{y}) \\ cov(\mathbf{x},\mathbf{y}) & cov(\mathbf{y},\mathbf{y}) \end{pmatrix}$$

    - 공분산: 2개의 변수가 함께 변하는 정도(joint variability)를 측정하는 척도
      - 두 변수 중 한 변수가 커지면 다른 변수의 값도 증가 ➔ 공분산 > 0
      - 공분산을 통해 두 변수의 선형적 관계를 설명할 수 있음
    - x, y의 공분산 $cov(\mathbf{x},\mathbf{y}) = \sigma_{\mathbf{xy}} = \sigma(\mathbf{x},\mathbf{y}) = E[(\mathbf{x}-E[\mathbf{x}])(\mathbf{y}-E[\mathbf{y}])]$

Matching 전략

Covariance matrix 예시

$$cov(\mathbf{x},\mathbf{y}) = E[(\mathbf{x}-E[\mathbf{x}])(\mathbf{y}-E[\mathbf{y}])]$$
$$= \mathrm{E}[\mathbf{x}\mathbf{y}^{\mathrm{T}}] - E[\mathbf{x}]E[\mathbf{y}]$$



$$\Sigma = \begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(x,y) & cov(y,y) \end{pmatrix}$$

- 빨간점 벡터 $\mathrm{X} = [(1,2),(2,1),(2,3),(3,2)]$
- $E[\mathbf{x}] = 2, E[\mathbf{y}] = 2$
- $cov(\mathbf{x},\mathbf{y}) = \frac{(1-2)(2-2)+(2-2)(1-2)+(2-2)(3-2)+(3-2)(3-2)}{4} = 0$
- $cov(\mathbf{x},\mathbf{x}) = \frac{(1-2)^2+(2-2)^2+(2-2)^2+(3-2)^2}{4} = \frac{1}{2} = Var(x)$
- $cov(\mathbf{y},\mathbf{y}) = \frac{(2-2)^2+(1-2)^2+(3-2)^2+(3-2)^2}{4} = \frac{1}{2}$

$$\Rightarrow \quad \Sigma = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Matching 전략

$d(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})\boldsymbol{\Sigma}^{-1}(\mathbf{a}-\mathbf{b})^{\mathrm{T}}}$

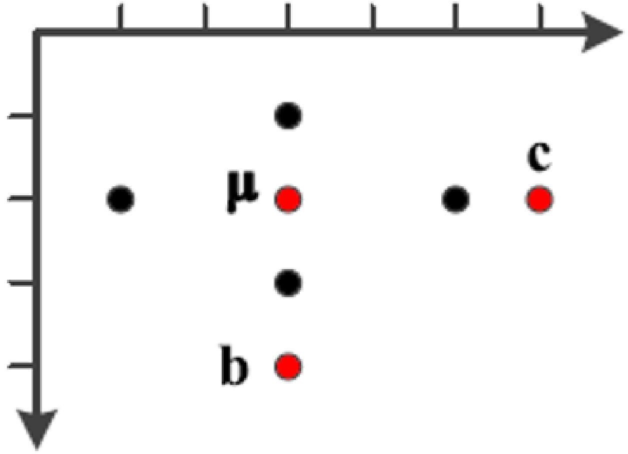
거리 기반의 전략

- Mahalanobis distance (cont.)
  - Gaussian distribution
    - $N(\mu,\sigma^2) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma}\exp\left(-\frac{(x-u)^2}{2\sigma^2}\right)$ ➔ $N(\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$
  - (maximum likelihood 유도 / discriminant function) ➔ $g(x) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$

Matching 전략

거리 기반의 전략

- Mahalanobis distance (cont.)
  - 예제
    - 네 점 {(2, 1), (1, 3), (2, 5), (3, 3)}이 확률 분포를 이루는 상황 (**μ**=(2, 3))
    - Euclidian ➔ **b**가 **c**보다 **μ**에 더 가까움
      - $d_E(\mathbf{\mu}, \mathbf{b}) = 2$, $d_E(\mathbf{\mu}, \mathbf{c}) = 3$
    - Mahalanobis ➔**c**가 **b**보다 **μ**에 더 가까움
      - $\Sigma = \begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix}, d_M(\mathbf{b}) = 2.8284,\ d_M(\mathbf{c}) = 2.1213$



Matching 전략

고정 임계값

- 거리를 기준으로 특정 임계값 이하인 경우 모든 쌍을 매칭

최근접 이웃

- a_i는 가장 가까운 b_i를 찾고, $d(a_i, b_i) < T$를 만족하면 매칭

최근접 이웃 거리 비율

- 최근접 b_i와 두 번째 최근접 b_k가 $\frac{d(a_i,b_i)}{d(a_i,b_k)} < T$를 만족하면 매칭
- ➔ 실험에 따르면 이 방법이 가장 좋은 성능을 보임
- ➔ 어떻게 최근접 이웃 거리 비율을 빠르게 찾을 것인가?

Matching 성능 측정

Performance evaluation

- 알고리즘 개선이나 최선의 알고리즘을 선택하는 기준
  - ➔ 제시한 알고리즘의 현장 투입 여부를 결정하는 기준
- Computer vision에서 매우 중요함

		Predicted	
		Negative (**N**) -	Positive (**P**) +
Actual	Negative -	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive +	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

Precision (정밀도) $= \frac{TP}{TP+FP}$

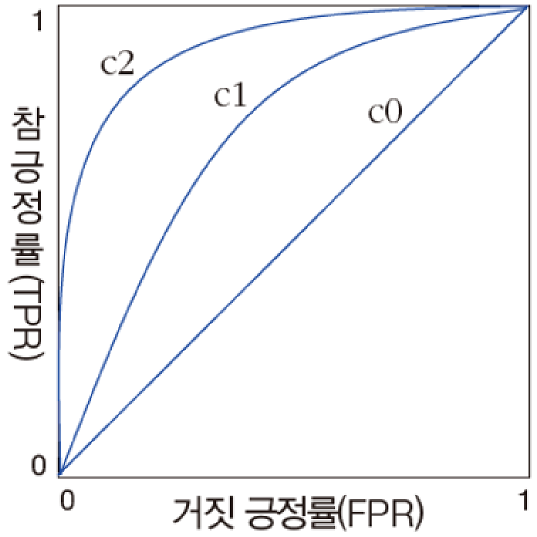
Recall (재현율, 민감도(Sensitivity)) $= \frac{TP}{TP+FN}$

Specificity (특이도) $= \frac{TN}{FP+TN}$

F1 score $= \frac{2*Precision*Recall}{Precision+Recall}$

Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$

Matching 성능 측정



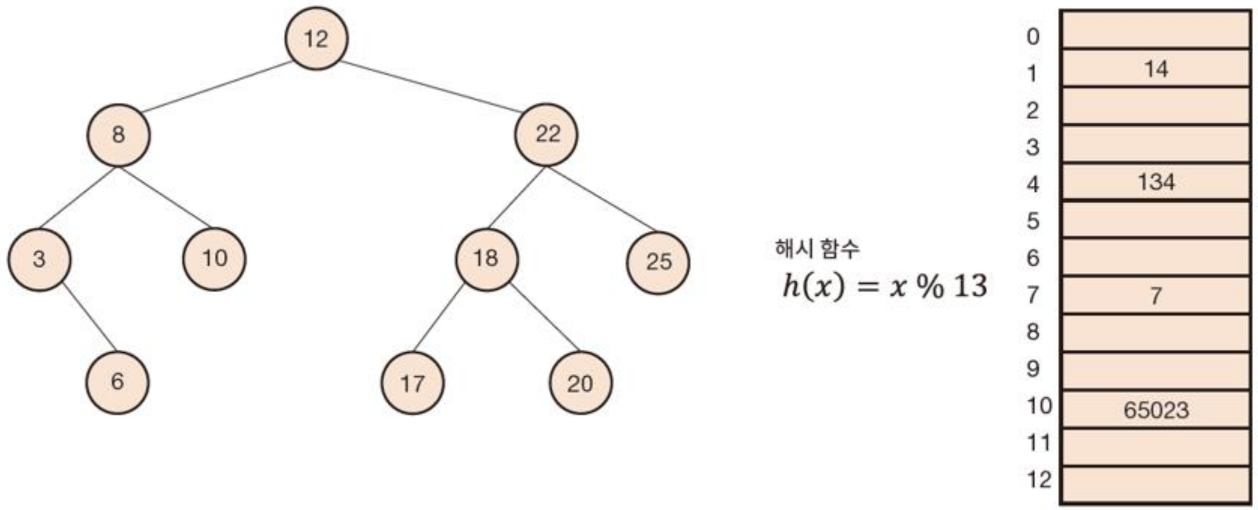
Performance evaluation

- ROC (Receiver Operating Char.) 곡선
  - X축(FPR): 1-specificity = $1 - \frac{TN}{FP+TN} = \frac{FP+TN-TN}{FP+TN} = \frac{FP}{FP+TN}$
    - FP = positive를 잘못 예측함 ➔ 실제로 negative
    - TN = negative를 negative로 잘 예측함 ➔ 실제로 negative
    - TPR = 실제로 negative인 것들 중에 positive로 잘못 예측한 비율
  - Y축(TPR): sensitivity $\frac{TP}{TP+FN}$
    - Sensitivity: True를 True로 잘 예측?
    - TP = positive를 positive로 잘 예측함 ➔ 실제로 positive
    - FN = negative를 잘못 예측함 ➔ 실제로 positive
    - FPR = 실제로 positive인 것들을 positive로 옳게 예측한 비율

Matching algorithm

성능 지표: “Speed”

- 실시간성이 요구되는 응용에서 양보할 수 없는 강한 조건
- Computer science에서 실시간성 ➔ “주어진 시간 내 처리“
- Data structure 기반의 매칭 기법 ➔ 빠르게 최근접 이웃 (비율) 찾기
    - Kd 트리
    - Hashing



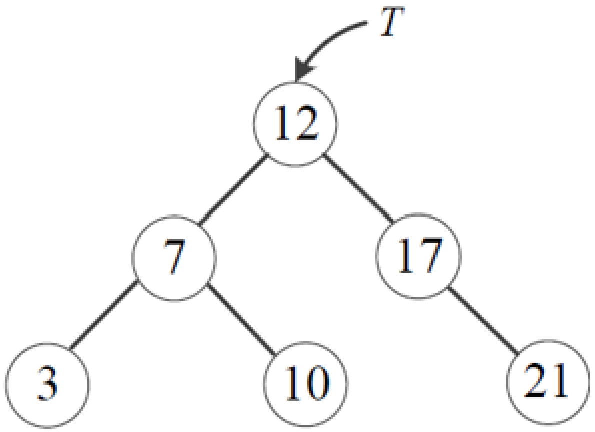
(a) 이진 탐색 트리

(b) 해싱

Kd트리

Background

- 특징점 매칭의 독특한 성질로 인해 이진 탐색 트리 그대로 적용 X
  - 특징점: 여러 값으로 구성된 특징 벡터의 형태
  - 같은 값이 아니라 최근접 이웃을 찾음
- ➔ kd트리는 이러한 특성에 적합한 자료 구조
  - 1) kd트리 만들기
  - 2) kd트리에서 탐색



* 이진 탐색 트리 (Binary Search Tree, BST)

- Root node를 기준으로 왼쪽에는 상위(부모) node보다 작은 값, 오른쪽에는 큰 값을 갖는 트리
  - 균형 잡힌 트리인 경우 탐색 시간 $O(logn)$

Kd트리

Notation

- kd트리를 구성하는 n개의 벡터 $X = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_n}\}$; $\mathbf{x_i}$는 d차원 벡터

Idea

- Root node는 X를 두 개의 부분집합 X_{left}, X_{right}로 분할
- 분할 기준
  - 1) d개의 차원(축) 중 축 선택
    - ➔ 분할 효과를 극대화하려면 각 차원의 최대 분산을 갖는 축 k를 선택
  - 2) 축 선택 후 n개의 샘플 중 어떤 것을 기준으로 X를 분할할 것인가?
    - ➔ X_{left}, X_{right}의 크기 차이를 최소화하여 균형 잡힌 트리를 구성해야 함
    - X를 차원 k로 정렬 후 그 결과의 중앙값을 분할 기준으로 설정
  - 과정 1, 2를 재귀적으로 반복하여 kd트리 생성

Kd트리

Example

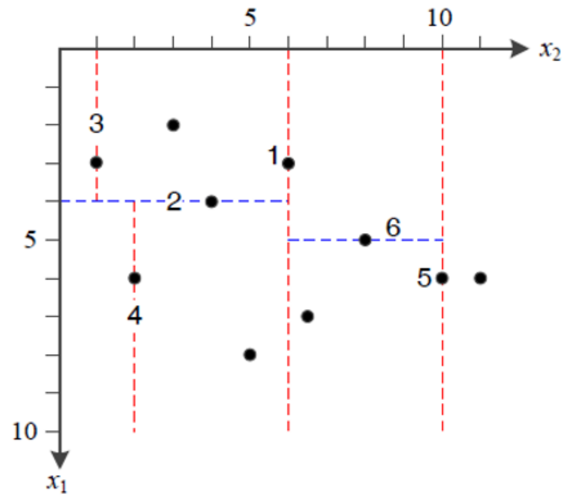
- $d = 2$인 특징 벡터 X

  $X = \{\mathbf{x_1} = (3,1), \mathbf{x_2} = (2,3), \mathbf{x_3} = (6,2), \mathbf{x_4} = (4,4), \mathbf{x_5} = (3,6),$
  $\mathbf{x_6} = (8,5), \mathbf{x_7} = (7,6.5), \mathbf{x_8} = (5,8), \mathbf{x_9} = (6,10), \mathbf{x_{10}} = (6,11)\}$

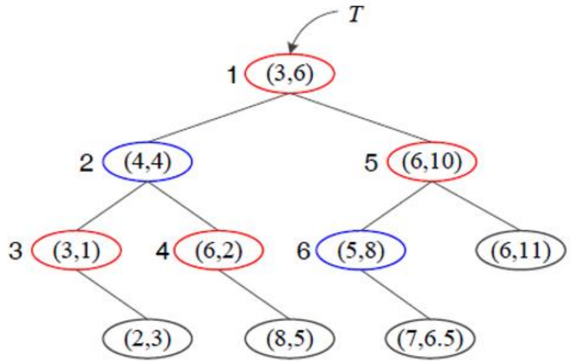
- 1) root node로 결정할 만한 분할 기준 탐색
  - 축 결정: 각 차원에 대한 분산 중 분산이 큰 쪽이 축으로 결정됨

    {3, 2, 6, 4, 3, 8, 7, 5, 6, 6} vs {1, 3, 2, 4, 6, 5, 6.5, 8, 10, 11}

    ➔ 두번째 차원의 축이 더 큼 $k = 2$를 기준으로 정렬
- 2) 정렬 후 중앙값 기준으로 분할 시작
  - 정렬 결과: $X_{sorted} = \{\mathbf{x_1}, \mathbf{x_3}, \mathbf{x_2}, \mathbf{x_4}, \mathbf{x_6}, \mathbf{x_5}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}\}$
  - 중앙값 ($\mathbf{x_5}$)기준으로 분할 ➔ $X_{left} = \{\mathbf{x_1}, \mathbf{x_3}, \mathbf{x_2}, \mathbf{x_4}, \mathbf{x_6}\}$, $X_{right} = \{\mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}\}$



(a) 특징 벡터의 집합

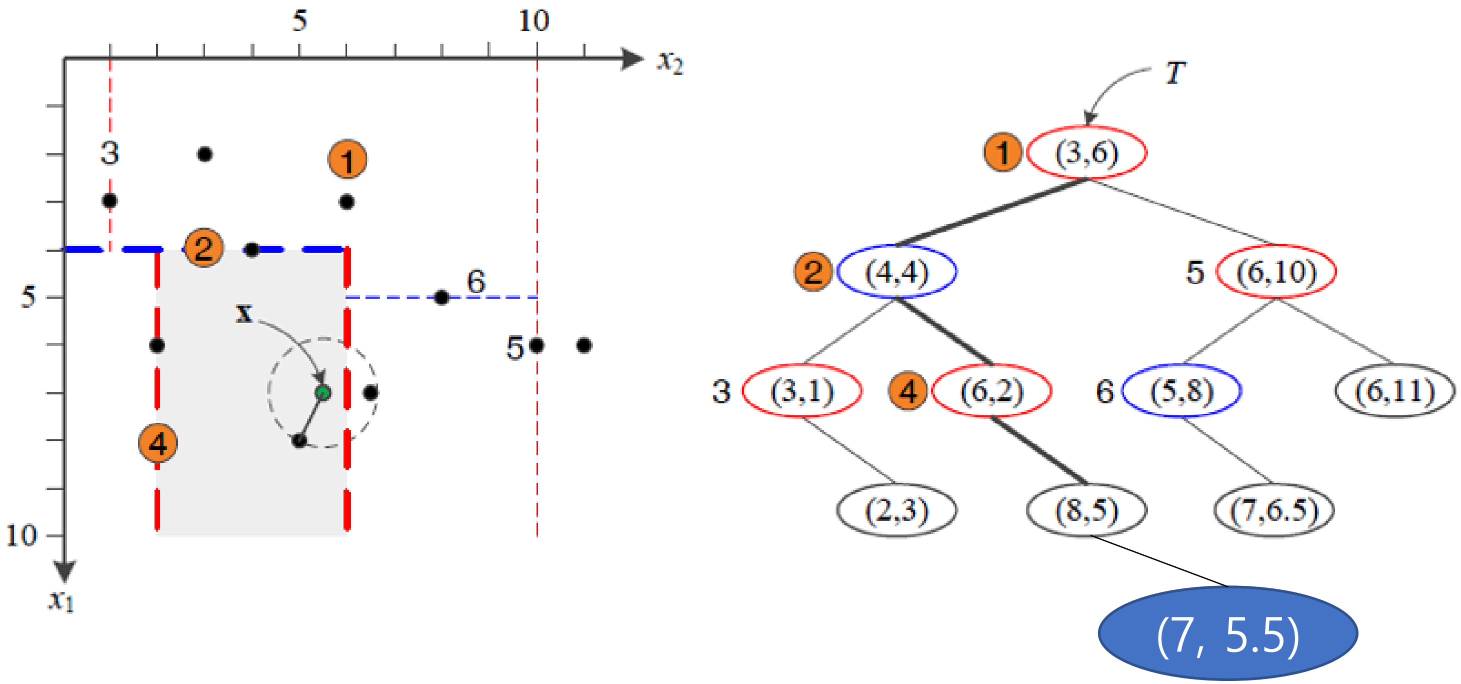


(b) 완성된 kd 트리

Kd트리

Example

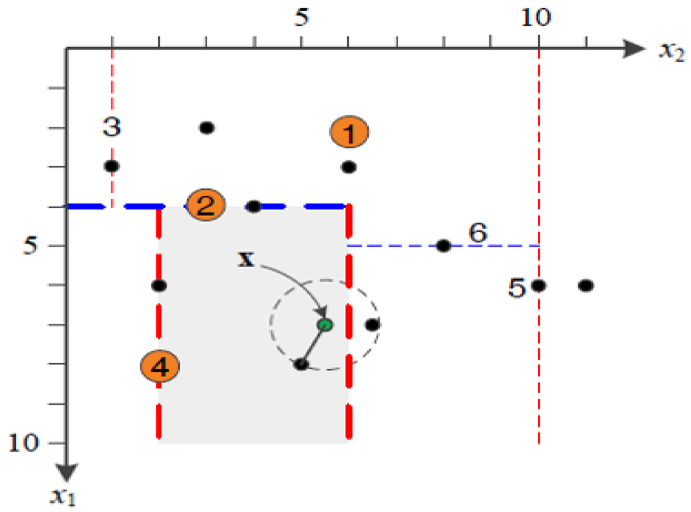
- 새로운 특징 벡터$\mathbf{x} = (7, 5.5)$입력 시
  - $k = 2$ ➔ x_2 축을 기준으로 5.5와 6 비교
    ➔ 왼쪽으로 ➔ 5.5와 4 비교 ➔ 오른쪽으로… 반복

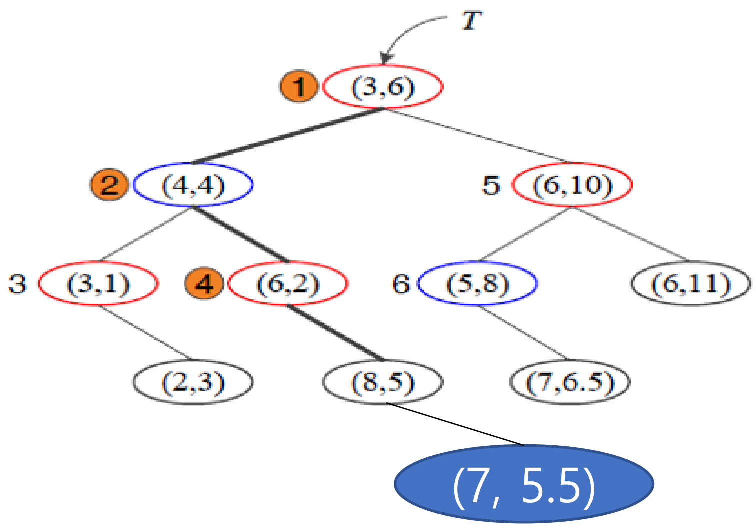


Kd트리

Example

- 여기가 최선인가?
  - ➔ 최근접일 가능성은 있지만 반드시 그렇지는 않음
  - 분할 평면의 건너 편에 더 가까운 node 존재 가능성이 있음
  - Stack을 이용한 back tracking 필요 (한정 분기 적용)
- $d \geq 10$ 인 경우 속도가 급격히 저하됨





Hashing

Idea

- General hashing
  - Hash function은 key 값을 테이블의 주소로 변환
  - 테이블에 고르게 배치될수록 좋은 hash function
  - $O(1)$ 시간 안에 탐색 달성
  - But, 충돌 해결책 필요

    ➔ 여러가지 충돌 해결 기법이 개발되어 있음
  - Matching에 hashing 활용 방법 (정반대의 목표를 가짐)
  - Key: 단일 값이 아니라 실수 벡터의 형태
  - 동일 요소가 아니라 최근접 이웃을 찾도록 함
  - Matching에서의 hashing: 유사한 벡터를 같은 통에 담도록 해야함

  ➔ 위치 의존 hashing

해시 함수 $h(x)=x \bmod 13$

0	
1	27
2	
3	
4	147
5	
6	19
7	
8	8
9	
10	23
11	1311
12	

Hashing

위치 의존 hashing

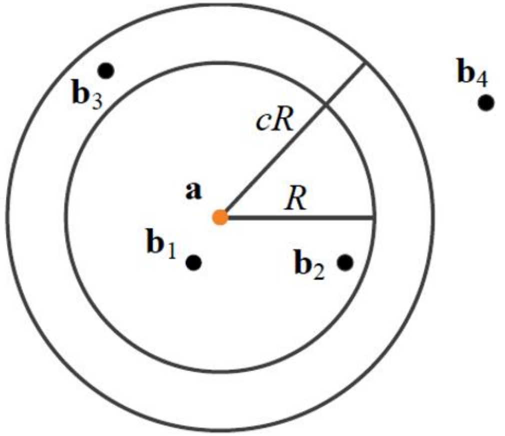
- 하나의 hash function이 아니라 hash function 집합 H에서 여러 개를 임의 선택하여 사용
- H가 속한 hash function이 아래를 만족하면 H는 위치 의존적

임의의 두 벡터 $\mathbf{a}$, $\mathbf{b}$에 대해,

$$\|\mathbf{a}-\mathbf{b}\| \le R \ \rightarrow p\big(h(\mathbf{a}) = h(\mathbf{b})\big) \ge p_1$$

$$\|\mathbf{a}-\mathbf{b}\| \le cR \ \rightarrow p\big(h(\mathbf{a}) = h(\mathbf{b})\big) \le p_2$$

이때 $c > 1, p_1 > p_2$



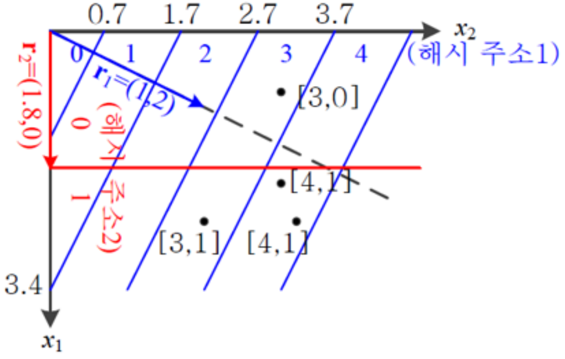
- * 위치의존의 의미
  - 가까운 두 벡터는 같은 통에 담길 (Hash function 값이 같을) 확률이 크고,
    먼 벡터는 같은 통에 담길 확률이 적음

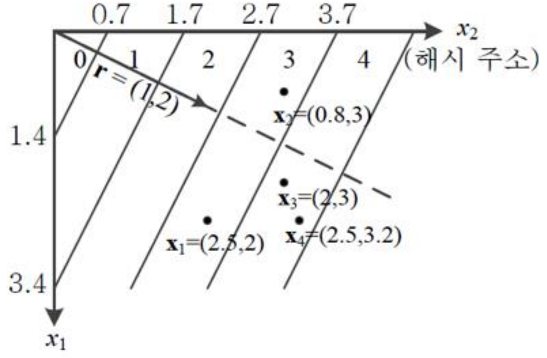
Hashing

H생성 방법

- 다양한 방법이 개발되어져 있으나, 일반적으로 난수를 활용
- 난수로 $\mathbf{r}$과 b를 설정하여 원하는 수만큼 함수 생성 가능

$$h(\mathbf{x}) = \left\lfloor \frac{\mathbf{r} \cdot \mathbf{x} + b}{w} \right\rfloor$$





(a) 해시 함수 한 개 사용

동작 방법

- d차원 공간을 $\mathbf{r}$에 수직인 초평면으로 분할
- w는 구간의 간격 ➔ 작으면 촘촘하게 분할됨

예시

- $\mathbf{x} = (x_1, x_2)$는 2차원, $w = 2$, 난수 생성 $\mathbf{r} = (\mathbf{1}, \mathbf{2}), b = 0.6$ 인 경우 ➔ $h(\mathbf{x}) = \left\lfloor \frac{x_1 + 2x_2 + 0.6}{2} \right\rfloor$
- $\mathbf{x_1} = (2.5, 2)$➔ 3, $\mathbf{x_2} = (0.8, 3)$➔ 3, $\mathbf{x_3} = (2, 3)$➔ 4, $\mathbf{x_4} = (2.5, 3.2)$➔ 4

FLANN 알고리즘

FLANN 알고리즘

- Fast Library for Approximate Nearest Neighbors
- 빠른 feature matching을 보장함
- 3_3_SIFT_FLANN.py

```
flann_matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)
knn_match = flann_matcher.knnMatch(des1, des2, 2)
```

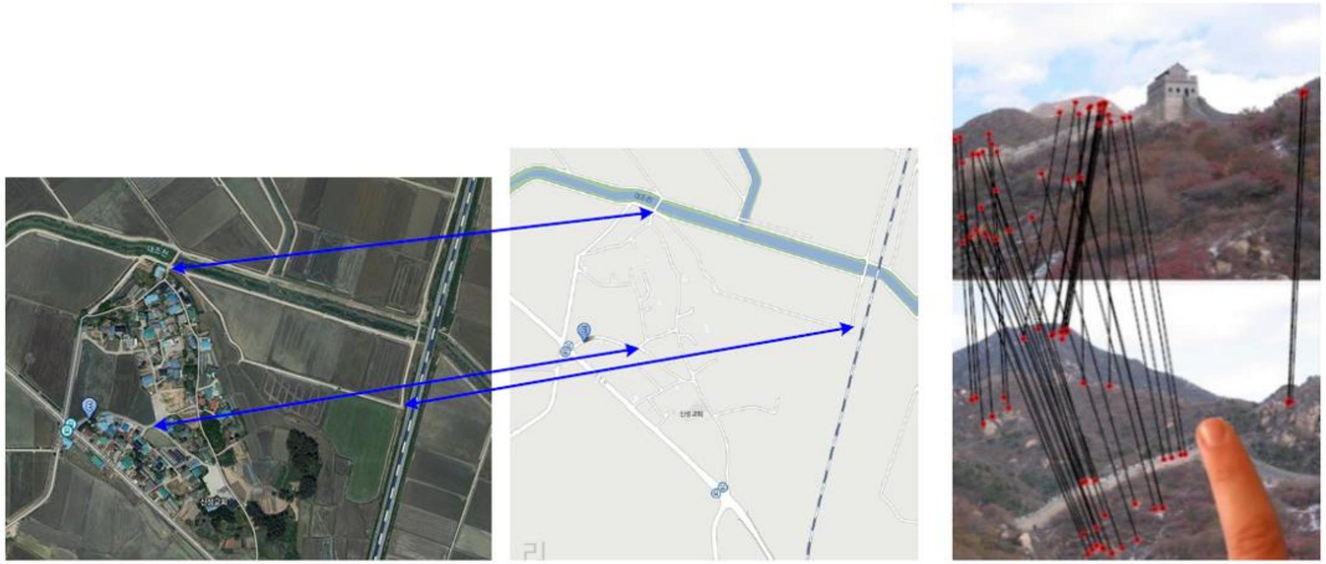
```
T = 0.7
good_match = []
for nearest1, nearest2 in knn_match:
    if(nearest1.distance / nearest2.distance) < T:
        good_match.append(nearest1)

img_match = cv2.drawMatches(img1, kp1, img2, kp2,
                            good_match, img_match,
                            flags = cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

기하 정렬과 변환 추정

* 아웃라이어: 데이터의 분포에서 현저하게 벗어나 있는 관측값

특징 벡터가 개별적으로 matching 수행

- 아웃라이어 matching (False Positive)
- 사람이 개입하여 아웃라이어가 없을 수도 있음 (항공사진 비교, 의료 영상 정합 등)
- ➔ 기하 정렬을 이용하여 인라이어 집합을 찾아내고 변환 행렬 추정해야 함



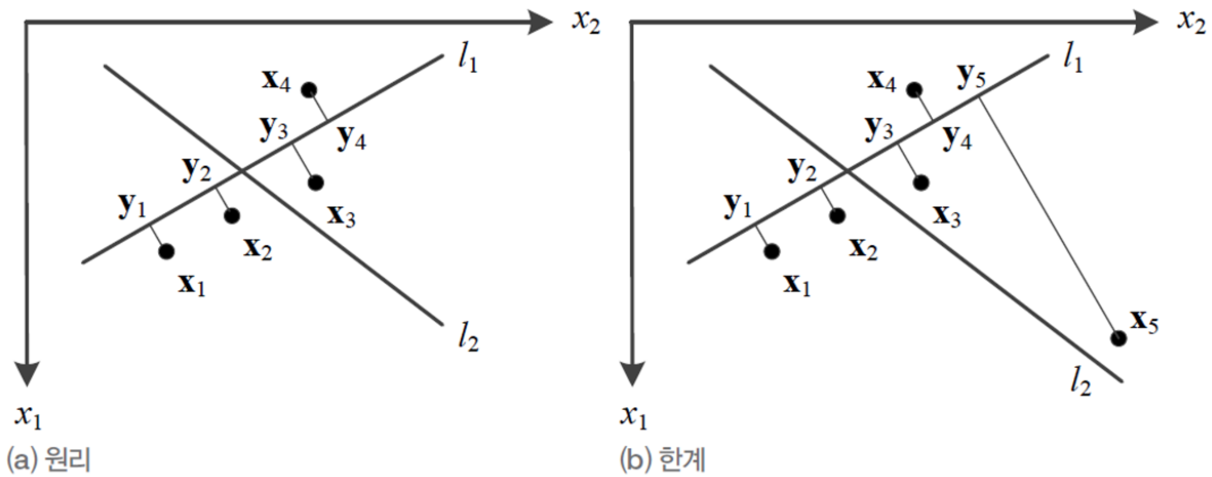
(a) 대응 쌍이 모두 옳음

(b) 거짓 긍정이 포함된 경우

최소제곱법과 강인한 추정 기법

최소 제곱법

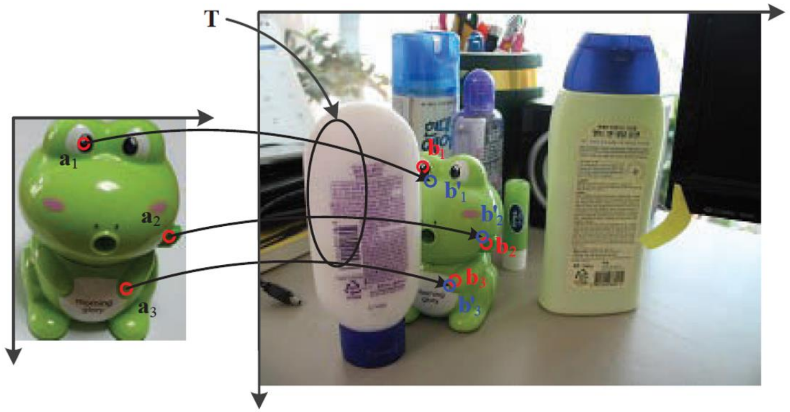
- 오래 전부터 수학/통계 분야에서 사용된 기법
- Example
  - X={x1, x2, x3, x4}를 가장 잘 대표하는 직선을 찾아라 (regression)



- 직선 l까지의 거리의 합을 오차로 공식화 $E(l) = \sum \|\mathbf{x}_i - \mathbf{y}_i\|^2$

최소화 !

최소제곱법과 강인한 추정 기법



(a) 모델 영상 (b) 장면 영상

최소 제곱법의 matching 문제로 확장

- 입력: matching 쌍 집합 $X = \{(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)\}$
- 모델 ➔ 변환 행렬

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix}$$

- 오차함수 $E(T)$

$$\mathbf{b}'_i = \mathbf{a}_i \mathbf{T} \Rightarrow (b'_{i1} \;\; b'_{i2} \;\; 1) = (a_{i1} \; a_{i2} \;\; 1) \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix}$$

$$E(\mathbf{T}) = \sum_{i=1}^{n} \left((b_{i1} - (t_{11}a_{i1} + t_{21}a_{i2} + t_{31}))^2 + (b_{i2} - (t_{12}a_{i1} + t_{22}a_{i2} + t_{32}))^2 \right)$$

최소제곱법과 강인한 추정 기법

최소 제곱법의 matching 문제로 확장

- $E(T)$를 최소화하는 T ➔ 최적화 문제

T의 요소들

$$\begin{pmatrix} \sum_{i=1}^{n} a_{i1}^2 & \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i1} & 0 & 0 & 0 \\ \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i2}^2 & \sum_{i=1}^{n} a_{i2} & 0 & 0 & 0 \\ \sum_{i=1}^{n} a_{i1} & \sum_{i=1}^{n} a_{i2} & \sum_{i=1}^{n} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sum_{i=1}^{n} a_{i1}^2 & \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i1} \\ 0 & 0 & 0 & \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i1}^2 & \sum_{i=1}^{n} a_{i2} \\ 0 & 0 & 0 & \sum_{i=1}^{n} a_{i1} & \sum_{i=1}^{n} a_{i2} & \sum_{i=1}^{n} 1 \end{pmatrix} \begin{pmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_{i1}b_{i1} \\ \sum_{i=1}^{n} a_{i2}b_{i1} \\ \sum_{i=1}^{n} b_{i1} \\ \sum_{i=1}^{n} a_{i1}b_{i2} \\ \sum_{i=1}^{n} a_{i2}b_{i2} \\ \sum_{i=1}^{n} b_{i2} \end{pmatrix}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} r_i^2$$

- 아웃라이어의 영향력을 약화시키는 함수 $\rho(.)$를 사용하는 M-추정

$$\text{M - 추정} : \hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(r_i)$$

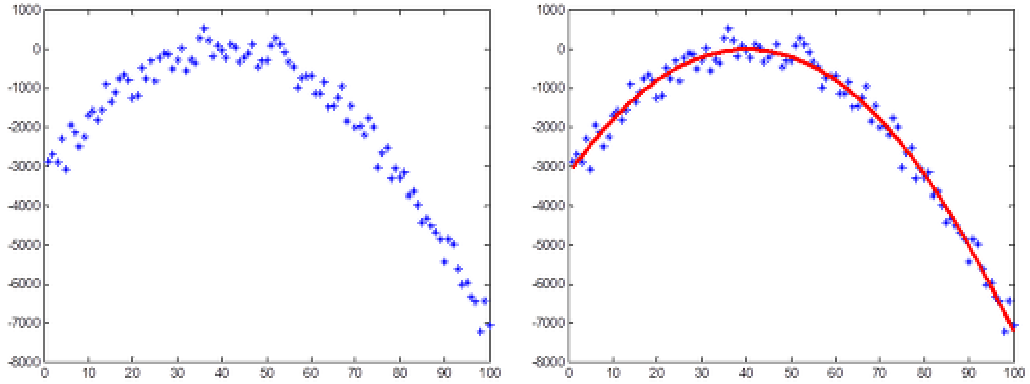
$$\rho(r) = \begin{cases} \frac{1}{2} r^2, & |r| \le c \\ \frac{1}{2} c(2|r| - c), & |r| > c \end{cases}$$

- 아웃라이어는 중앙값 계산하는 단계까지만 참여하는 최소제곱 중앙값

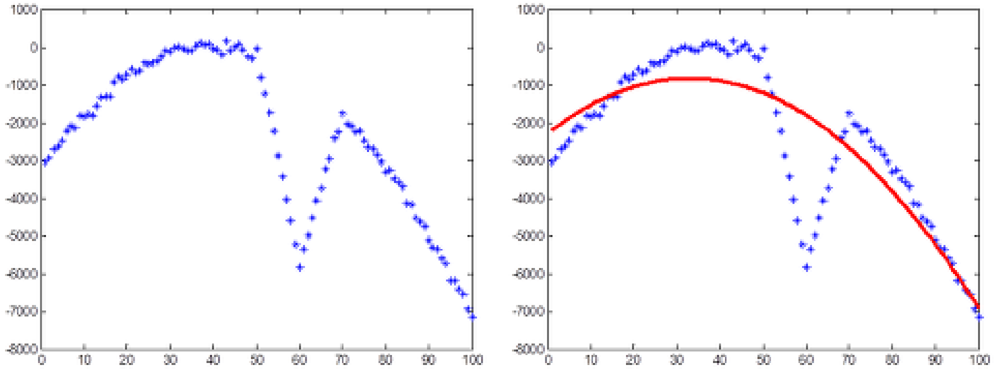
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \underset{i}{med}\, r_i^2$$

RANSAC

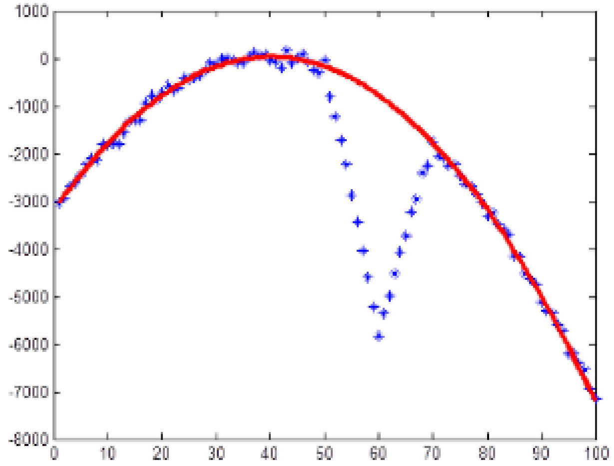
최소 제곱법의 한계



이상적인 fitting model



아웃라이어가 있는 경우

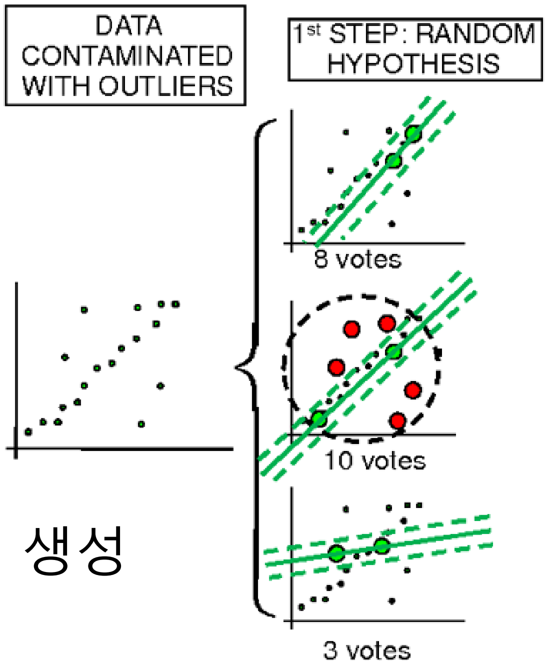


What we want

RANSAC

RANdom Sample Consensus

- 무작위로 샘플 데이터를 뽑은 후 최대로 컨센서스가 형성된 모델을 선택
- 최소 제곱법과 다르게 가장 많은 데이터 수의 지지를 받는 (컨센서스가 최대)인 모델을 선택
- 결국 무엇을 기준으로 모델의 파라미터를 찾는 지의 차이!



RANSAC 전략

- 1. N개의 샘플 데이터 선택
- 2. 샘플 데이터를 인라이너로 가정하고 모델 파라미터 계산(예측)
- 3. 예측된 모델과 일치하는 데이터 집계

  ➔ 집계된 데이터 수가 이전 최댓값보다 큰 경우 새로운 모델 파라미터 생성
- 4. 1, 2, 3 과정 반복

RANSAC

RANSAC 파라미터

- p: 인라이너로만 이루어진 샘플을 획득할 확률
- α: 주어진 데이터셋에서 인라이너의 비율
- m: 회당 추출하는 데이터 수
- N: 반복 횟수

End of slide