k-Nearest Neighbor

Taehoon Ko (thoon.koh@gmail.com)

목표

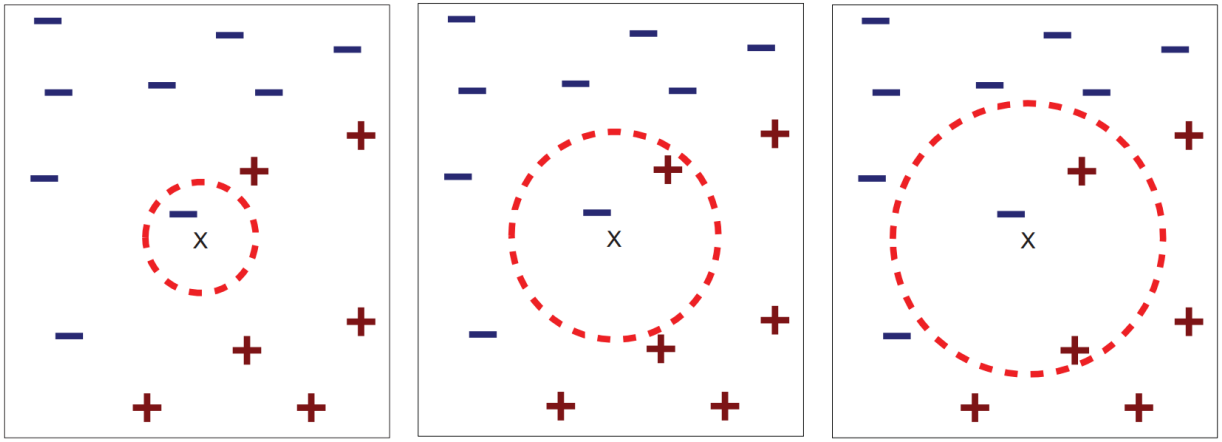
- **다음을 이해한다.**
  - 최근접 이웃 탐색 (Nearest neighbor search)
  - 데이터 포인트 간의 거리, 유사도 계산
  - k-nearest neighbor algorithm과 이를 활용한 분류모델과 회귀모델

Nearest neighbor search

- **최인접이웃 탐색**
  - 다음 영역에서의 핵심적 알고리즘
    - 검색 엔진 (information retrieval)
    - 협업 필터링 (collaborative filtering)
    - …
  - 자료 구조
    - k-dimensional tree (k-d tree)
    - Ball tree
    - Locality sensitive hashing (LSH)
    - …

k-nearest neighbors (k-NN)

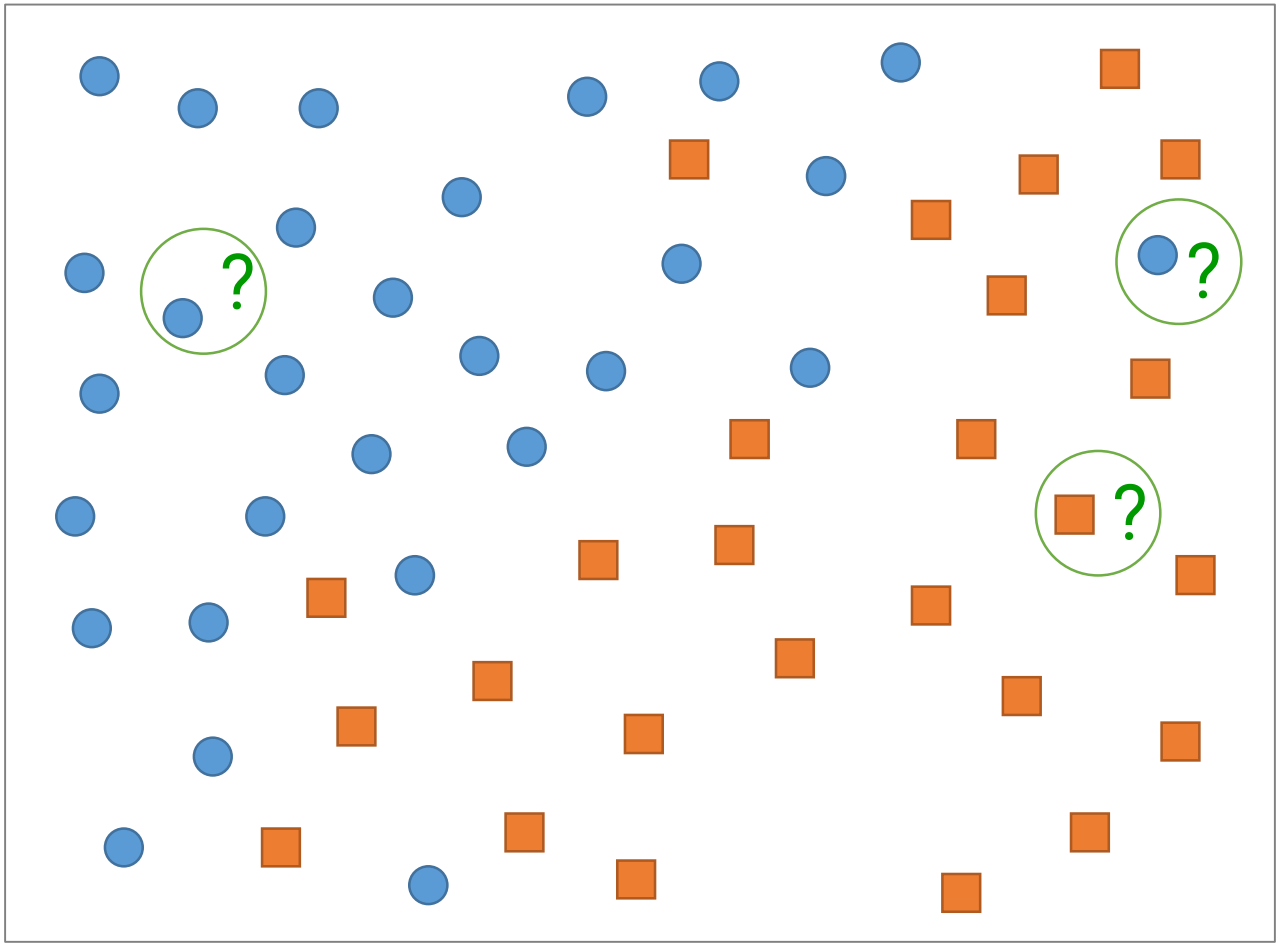
- k-nearest neighbors (k-**최근접 이웃**) algorithm
  - 목표: 특정 포인트에 가장 가까운 k개의 이웃 포인트들을 참조하여 그 포인트의 출력변수 Y를 예측하는 알고리즘
  - 가장 간단한 기계학습 모델 중 하나
  - 분류모델 (classifier), 회귀모델 (regressor) 모두 가능



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

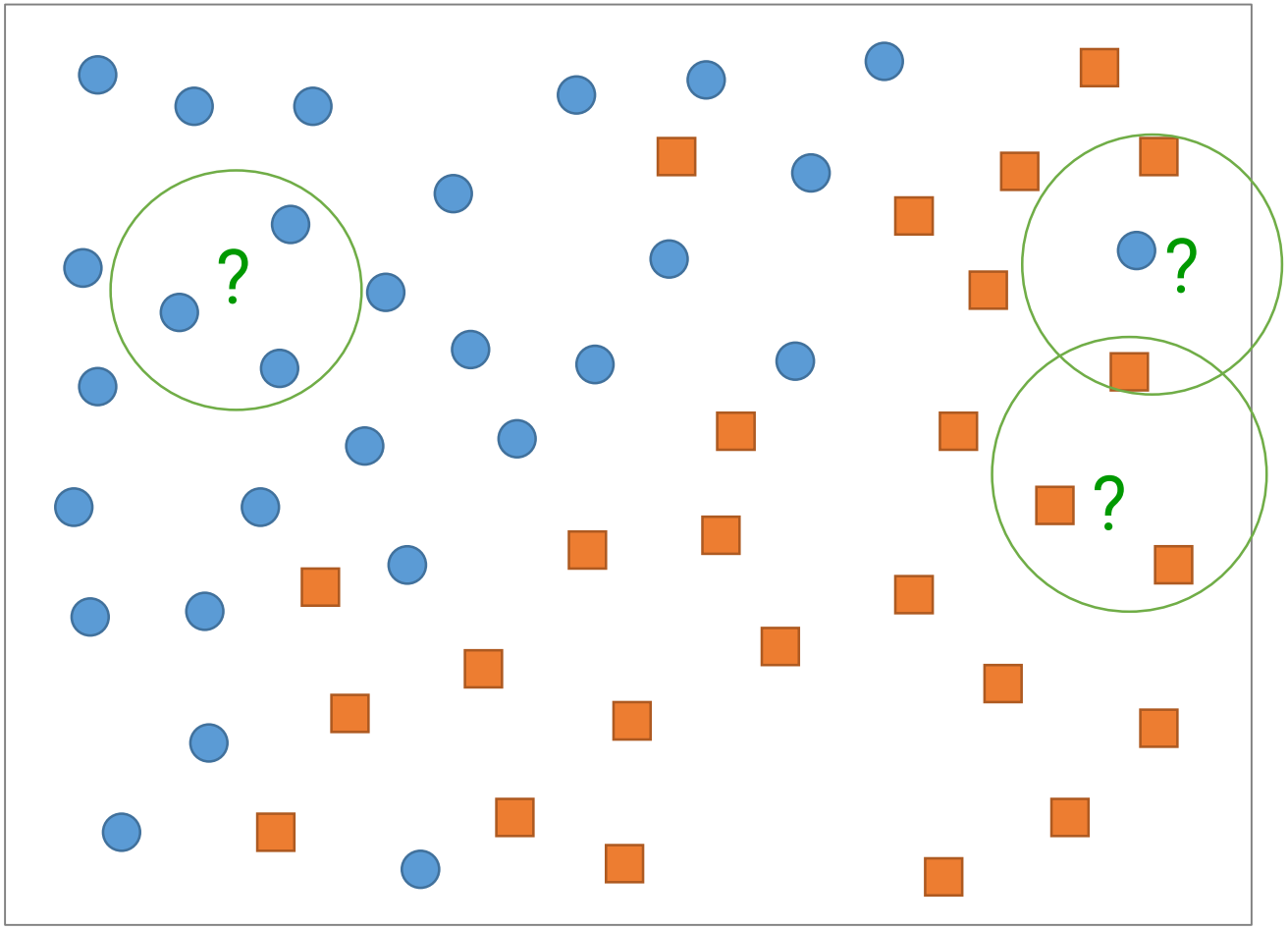
k-NN classification: Example

- Classify a new instance to the nearest neighbor's class



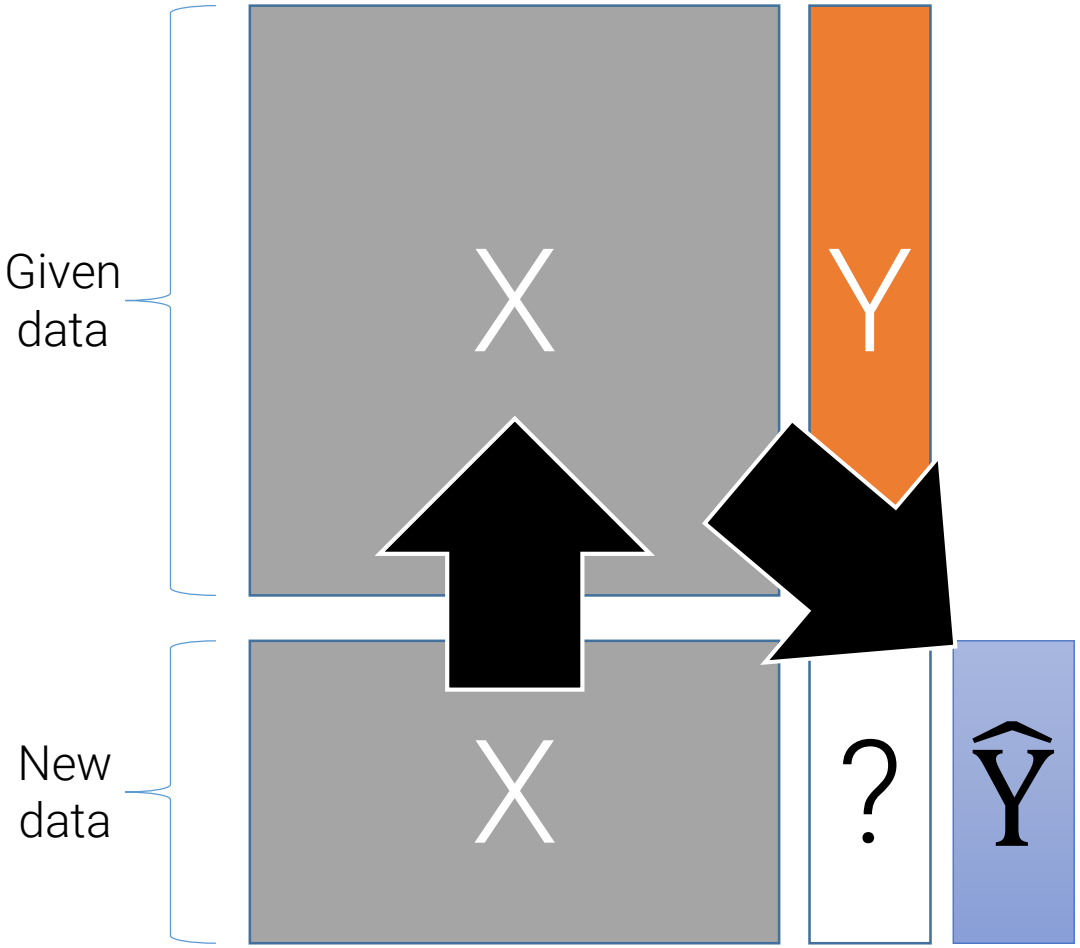
k-NN classification: Example

- Classify a new instance to the 3 nearest neighbors' class



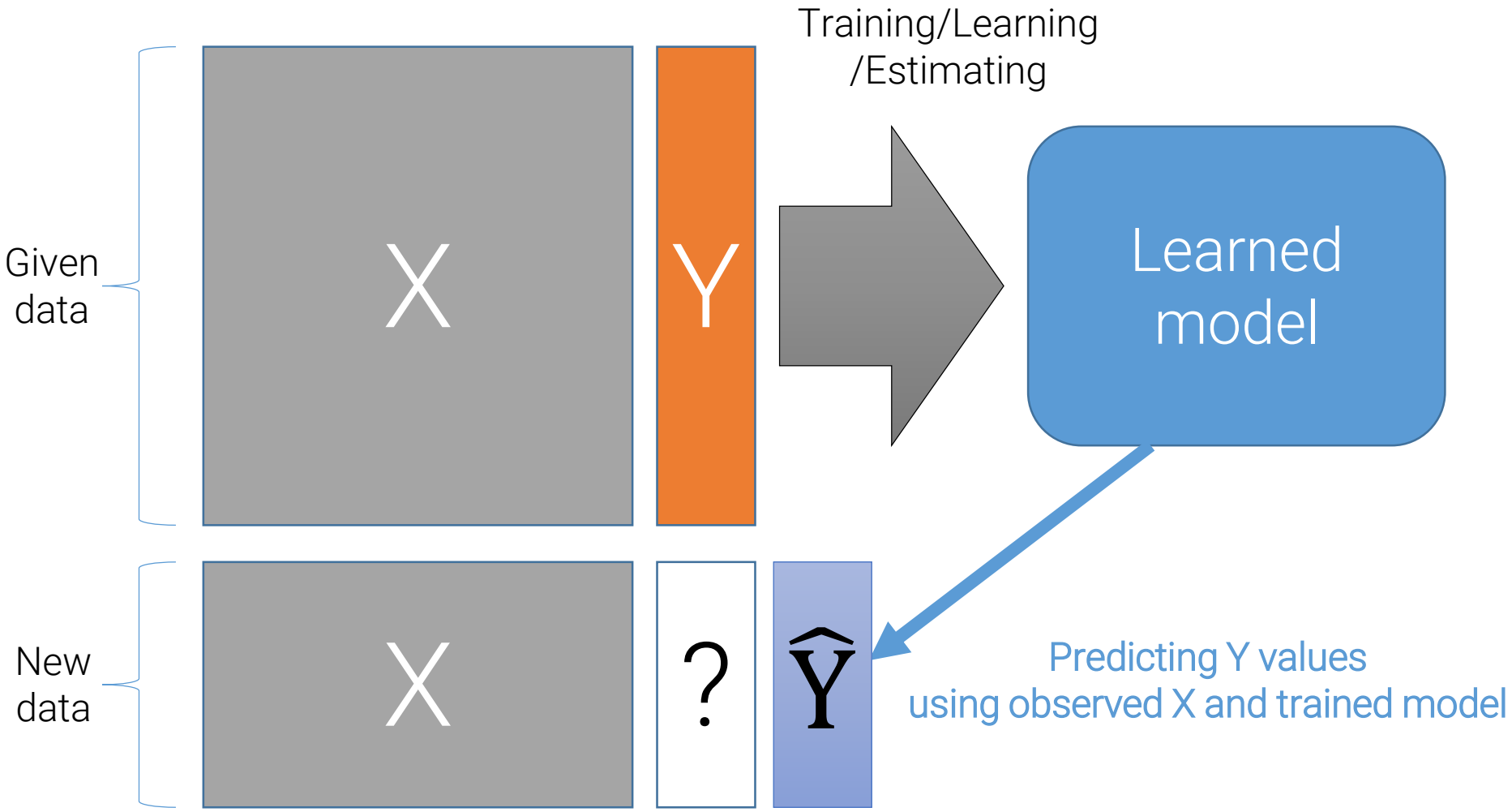
k-NN: 학습

- k-NN은 학습 과정이 없다! 오직 추론 과정만 있다.



Predicting Y values using observed X

(Revisited) Predictive modeling



k-NN: 추론 (Inference)

- **주어진 학습 데이터 $(\mathbf{X}, \mathbf{Y})$**
  - $\mathbf{X}$: Input variables
  - $\mathbf{Y}$: Output variable
- **새로운 포인트 Q가 등장했다고 가정하자.**
  - 1) Q와 모든 포인트 사이의 거리를 계산한다.
  - 2) 앞서 계산한 거리 중 가장 작은 k개의 거리들을 찾는다.
    이를 통해, Q에 가까운 k개의 포인트를 찾을 수 있다.
  - 3) k개의 이웃 포인트들의 y값을 확인한 후. 이를 기반으로 새로운 포인트 Q의 $\hat{y}$을 계산한다.

Suppose there is a new point Q

1) For i in range(1,number of training points)

   Compute distance $d(X_i, Q)$

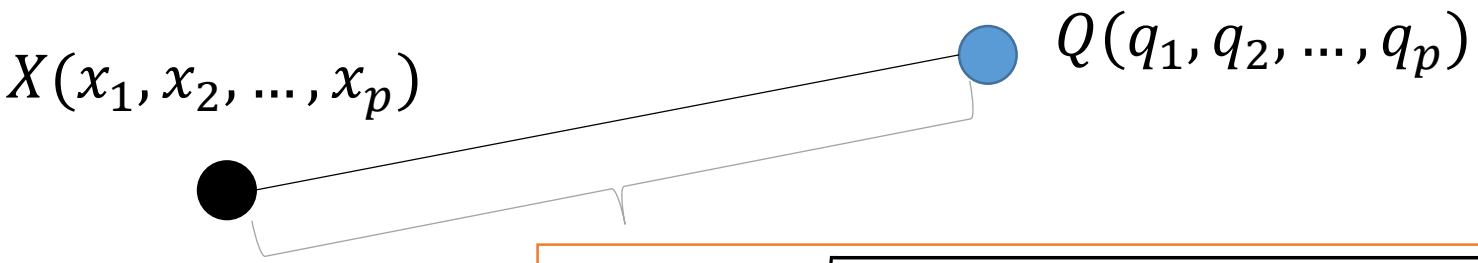
2) Compute set I containing indices for the k smallest distances $d(X_i, Q)$

3) Return $\hat{y}$ corresponding to the new point Q using $\{y_i \text{ for } i \in I\}$

세 가지에 대해 알아봅시다.

포인트 간의 거리 계산

- **유클리디안 거리** (Euclidean distance)
  - 두 점 사이의 거리 공식
  - 가장 널리 쓰이는 거리 척도



$$d(X,Q) = \sqrt{(x_1 - q_1)^2 + (x_2 - q_2)^2 + \cdots + (x_p - q_p)^2}$$
$$= \sqrt{\sum_{i=1}^{p}(x_i - q_i)^2}$$

  - 사용 시 주의사항
    - 변수 간의 스케일 차이가 보정되었는가?
    - 차원의 수가 지나치게 많지 않은가? (차원의 저주)

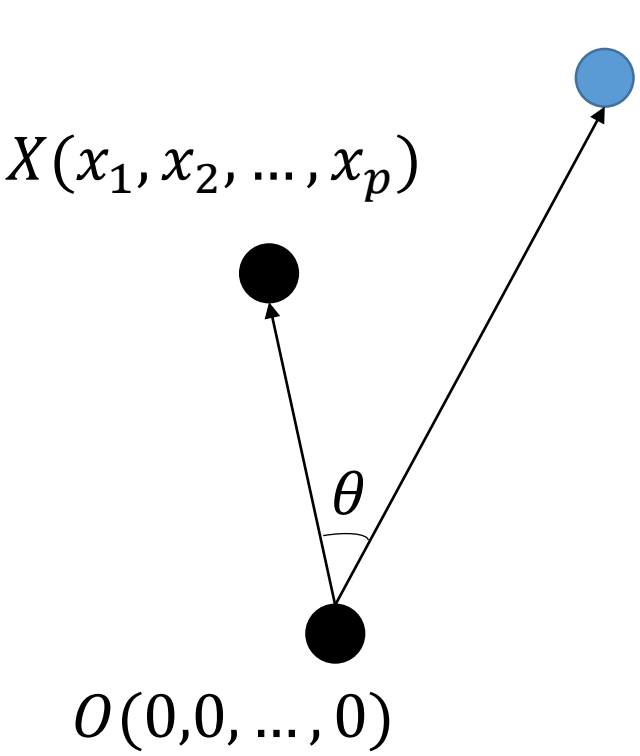
포인트 간의 거리 계산

=

포인트 간의 유사도 계산

포인트 간의 유사도 계산

- Cosine 유사도 (Cosine similarity)
  - 벡터 (즉, 포인트) 의 방향성이 얼마나 일치하는가?



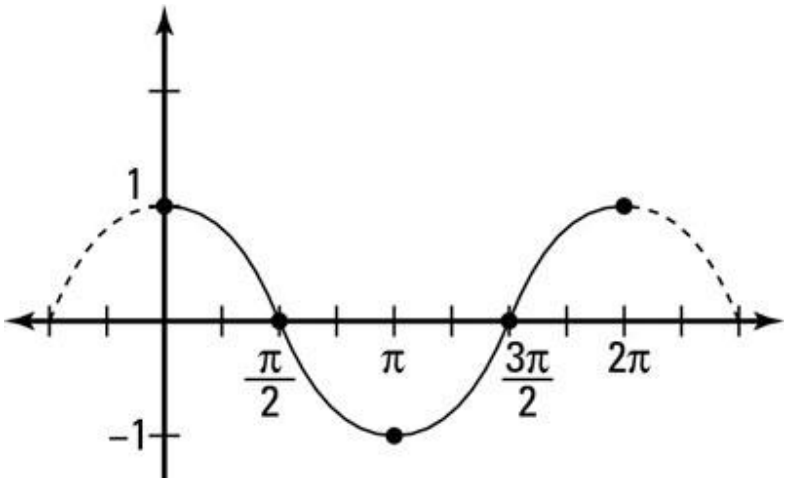
$$X \cdot Q = |X||Q|\cos\theta$$

$$x_1q_1 + x_2q_2 + \cdots + x_pq_p = \sqrt{x_1^2 + \cdots + x_p^2} \times \sqrt{q_1^2 + \cdots + q_p^2} \times \cos\theta$$

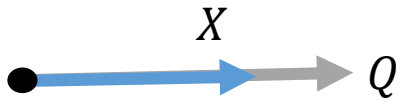
$$sim(X, Q) = \cos\theta = \frac{\sum_{i=1}^{p} x_i q_i}{\sqrt{\sum_{i=1}^{p} x_i^2}\sqrt{\sum_{i=1}^{p} q_i^2}}$$

포인트 간의 유사도 계산

- Cosine 유사도 (Cosine similarity)
  - 두 벡터 (포인트) 의 방향이 같으면 1
  - 두 벡터 (포인트) 가 직교하면 0
  - ...



$\cos 0 = 1$



$\cos 90^{\circ} = 0$

  - 유사도가 아닌 거리 척도로 변환할 때에는 $\mathrm{dist}(X, Q) = 1 - \cos(X, Q)$

(참고) 언제 cosine similarity가 좋은가?

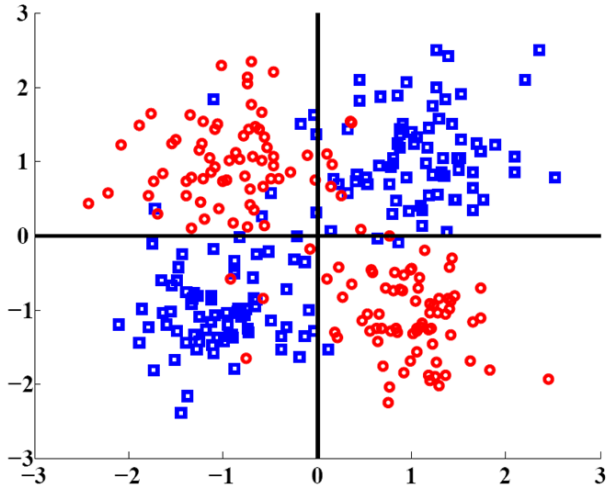
- Document - term matrix
  - 유클리디언 거리보다 코사인 유사도가 더 좋을 수 있다.
  - 예제: 문서1,2,3에 대해 각 단어가 포함된 수를 count
    - 유클리디언 거리의 경우
      - d(문서1, 문서2) = $\sqrt{3}$
      - d(문서3, 문서2) = $\sqrt{2}$
    - 코사인 유사도의 경우
      - sim(문서1, 문서2) = 1
      - sim(문서3, 문서2) = $\sqrt{3}$
    - 유클리디언 거리를 쓰는 경우, 문서2에 가장 가까운 것은 문서3
    - 코사인 유사도를 쓰는 경우, 문서2에 가장 가까운 것은 문서1

	단어1	단어2	단어3
문서1	2	2	2
문서2	1	1	1
문서3	0	0	1

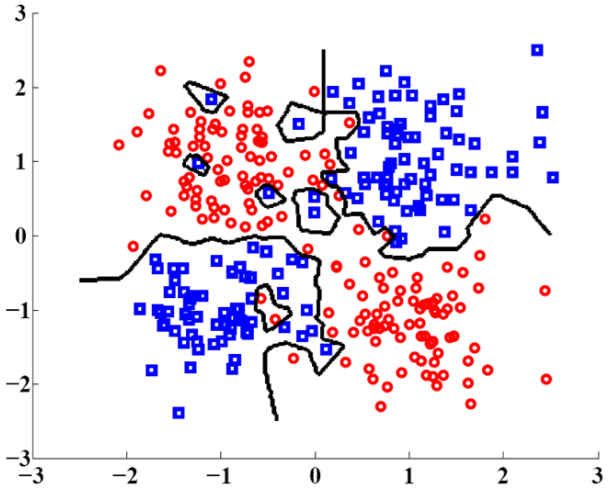
k-NN: How to select *k*

- **여러 개의 *k*값을 시도하여 가장 성능이 좋은 *k*를 선정**
  - k가 너무 작으면, 과적합(over-fitting)할 수 있으며 지역적인 노이즈에 민감
  - k가 너무 크면, 지역적인 데이터 구조를 잘 반영하지 못할 수 있음
  - 검증데이터(Validation set)을 이용하여 여러 개의 k값에 대한 예측 성능을 확인
    - 예측성능: Predictive performance (for classification or regression)

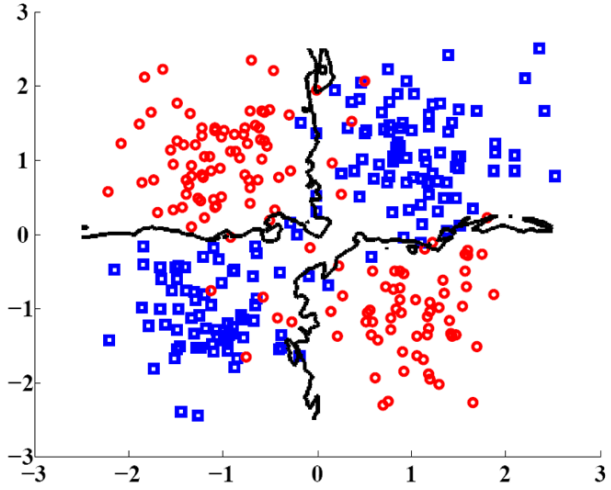
k-NN: How to select *k*



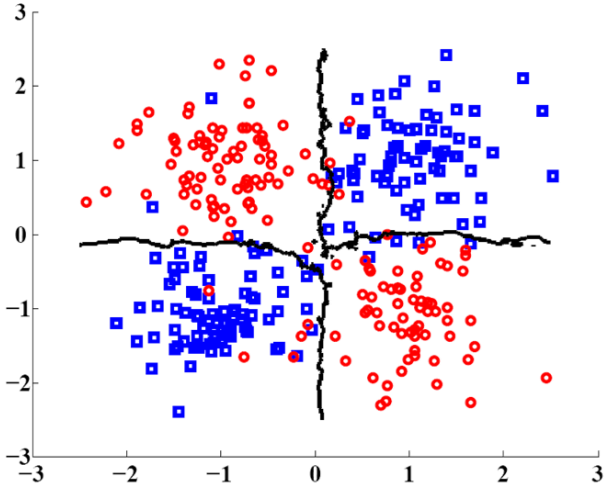
(a) The ideal boundary.



(b) k-NN classification with $k = 1$.

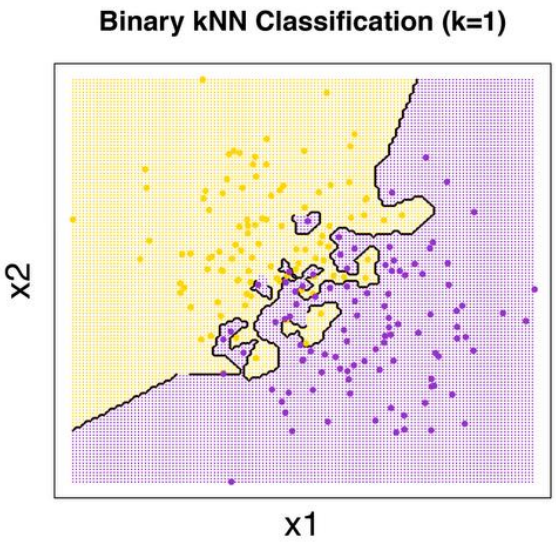


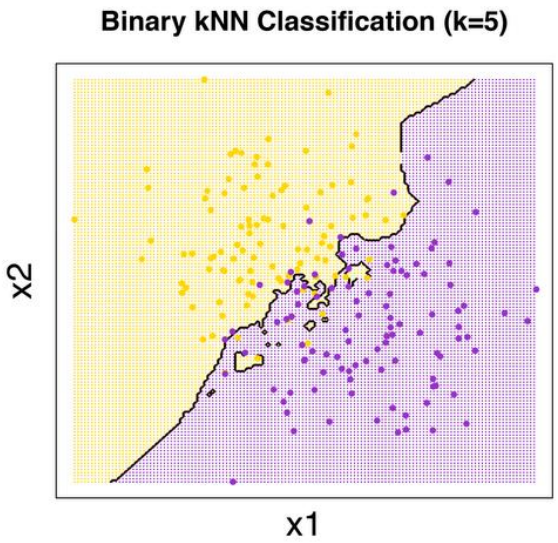
(c) k-NN classification with $k = 10$.

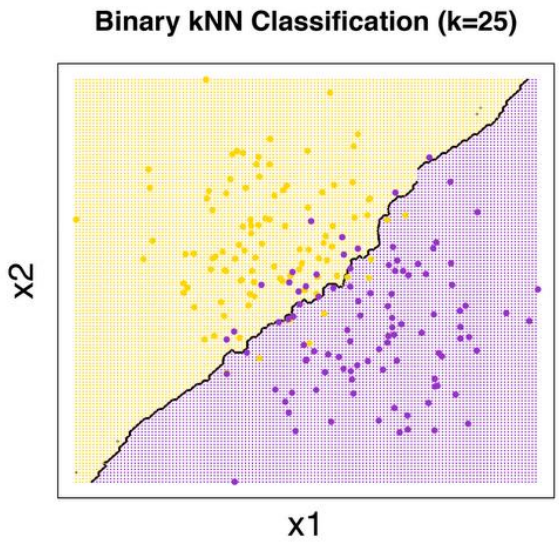


(d) k-NN classification with $k = 50$.

k-NN: How to select *k*







http://bdewilde.github.io/blog/blogger/2012/10/26/classification-of-hand-written-digits-3/

k-NN: How to classify a new point

- Majority voting vs. Weighted voting
  - 다수결 투표 (Majority voting)
    - Classify a new point as the majority class
  - 가중치 투표 (Weighted voting)
    - Assign 'weight' to the contribution of the neighbors.
    - Common weighting scheme

- distance between a new point and i^{th} neighbor: d_i

- weight for i^{th} neighbor : $w_i = \frac{1/d_i}{\sum_{j=1}^{k}(\frac{1}{d_j})}$

- Sum of weights: $\sum_{i=1}^{k} w_i = 1$

k-NN: How to classify a new point (Classification)

- Example: k=5로 하여, 새로운 포인트 Q에 대해 다음 5개의 이웃을 찾음

For a new point **Q**

Neighbor	Class	Distance	1/distance	Weight
N1	M	1	1.00	0.44
N2	F	2	0.50	0.22
N3	M	3	0.33	0.15
N4	F	4	0.25	0.11
N5	F	5	0.20	0.08

- Majority voting: $P(\hat{Y} = \mathrm{M}) = \frac{2}{5} = 0.4,\ P(\hat{Y} = \mathrm{F}) = 1 - 0.4 = 0.6$
- Weighted voting: $P(\hat{Y} = \mathrm{M}) = 0.44 + 0.15 = 0.59,$

$$P(\hat{Y} = \mathrm{F}) = 1 - 0.59 = 0.41$$

- Q is classified as F by the majority voting, while classified as M by the weighted voting

k-NN: How to predict of output value of a new point (Regression)

- Simple average vs. Weighted average
- Example: k=5로 하여, 새로운 포인트 Q에 대해 다음 5개의 이웃을 찾음

For a new point **Q**

Neighbor	Y	Distance	1/distance	Weight
N1	15.4	1	1.00	0.44
N2	17.2	2	0.50	0.22
N3	12.3	3	0.33	0.15
N4	11.5	4	0.25	0.11
N5	10.9	5	0.20	0.08

- Simple average

  : y of Q = (15.4+17.2+12.3+11.5+10.9)/5 = 13.46

- Weighted average

  : y of Q = 0.44*15.4+0.22*17.2+0.15*12.3+0.11*11.5+0.08*10.9 = 14.54

정리

- **k-NN 알고리즘의 특징**
  - 학습 과정이 없다.
    - 새롭게 축적된 데이터들을 쉽게 반영할 수 있다.
  - 비모수적 방법 (Non-parametric method)
    - 통계적 가정이 없다.
  - 데이터 포인트 수가 많고, 차원이 클 수록 (시간적, 공간적) 비용 증가
    - 새롭게 등장한 쿼리 포인트와 모든 포인트 간의 거리를 계산하는 것은 비효율적
    - 대안
      - 근사적 이웃 찾기 (Approximated nearest neighbor search)
      - 미리 데이터 인덱싱 구조를 생성 (Ball tree, LSH 등)

“For every complex problem

there is an answer that is clear, simple, and wrong.”

H. L. Mencken

How to stop worrying and learn to love Nearest Neighbors, NIPS 2017, Workshop for nearest neighbor (https://nn2017.mit.edu/schedule/)

- **복잡한 문제의 해결법이 반드시 "딥러닝"일 필요는 없음**
  - Nearest neighbor 방법을 이용한 예시 (NIPS 2017 Workshop)

Automatic Colorization

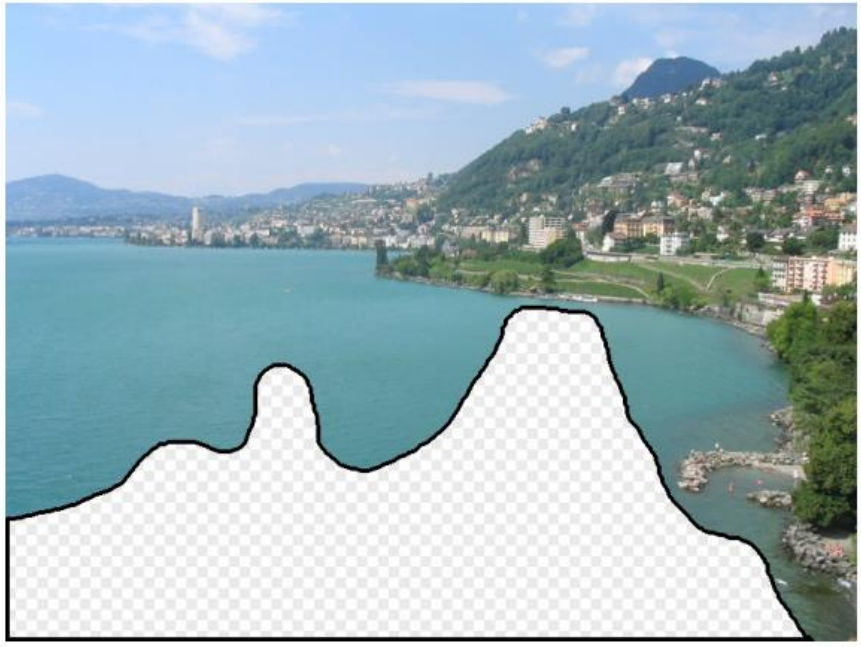
Grayscale input High resolution

Colorization of input using average

A. Torralba, R. Fergus, W.T.Freeman. 2008

How to stop worrying and learn to love Nearest Neighbors, NIPS 2017, Workshop for nearest neighbor (https://nn2017.mit.edu/wp-content/uploads/sites/5/2017/12/Efros-NIPS-NN-17.pdf)

- **복잡한 문제의 해결법이 반드시 "딥러닝"일 필요는 없음**
  - Nearest neighbor 방법을 이용한 예시 (NIPS 2017 Workshop)

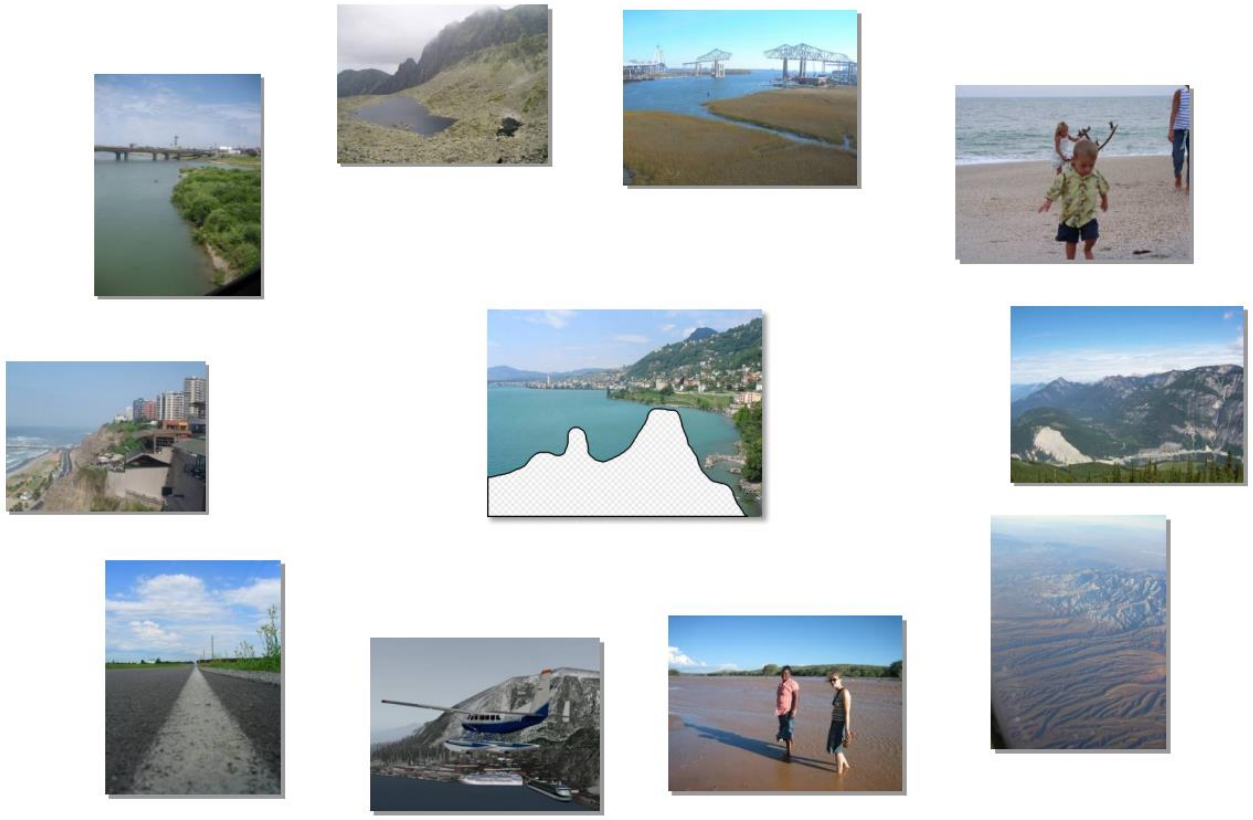


How to stop worrying and learn to love Nearest Neighbors, NIPS 2017, Workshop for nearest neighbor (https://nn2017.mit.edu/wp-content/uploads/sites/5/2017/12/Efros-NIPS-NN-17.pdf)

- **복잡한 문제의 해결법이 반드시 "딥러닝"일 필요는 없음**
  - Nearest neighbor 방법을 이용한 예시 (NIPS 2017 Workshop)

How to stop worrying and learn to love Nearest Neighbors, NIPS 2017, Workshop for nearest neighbor (https://nn2017.mit.edu/wp-content/uploads/sites/5/2017/12/Efros-NIPS-NN-17.pdf)

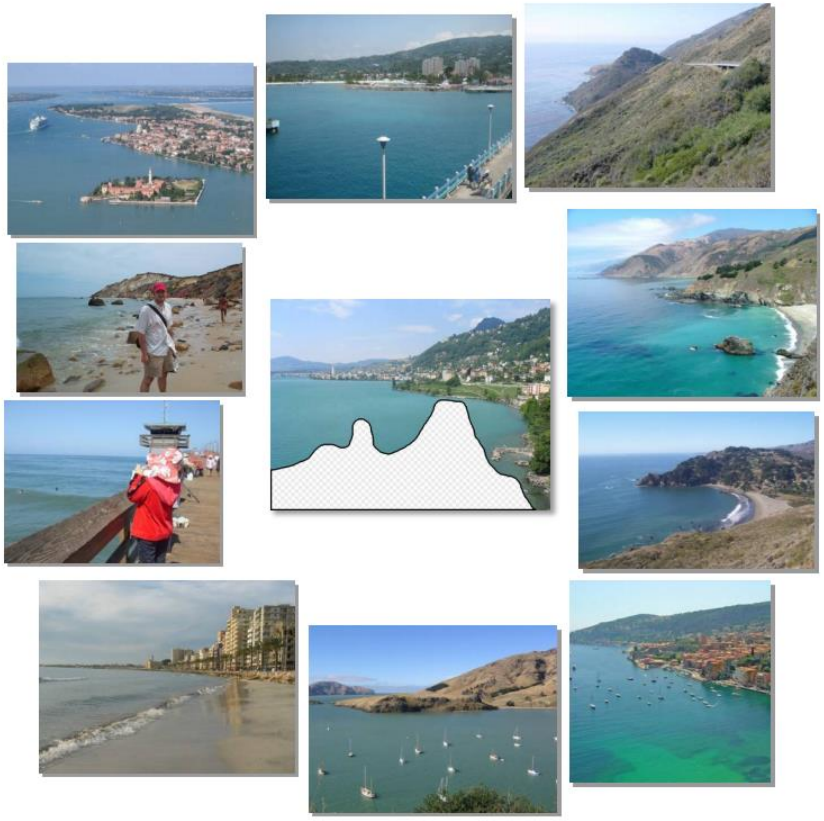
- **복잡한 문제의 해결법이 반드시 "딥러닝"일 필요는 없음**
  - Nearest neighbor 방법을 이용한 예시 (NIPS 2017 Workshop)



Nearest neighbors from a
collection of 20 thousand images

How to stop worrying and learn to love Nearest Neighbors, NIPS 2017, Workshop for nearest neighbor (https://nn2017.mit.edu/wp-content/uploads/sites/5/2017/12/Efros-NIPS-NN-17.pdf)

- **복잡한 문제의 해결법이 반드시 "딥러닝"일 필요는 없음**
  - Nearest neighbor 방법을 이용한 예시 (NIPS 2017 Workshop)



Nearest neighbors from a
collection of 2 million images

How to stop worrying and learn to love Nearest Neighbors, NIPS 2017, Workshop for nearest neighbor (https://nn2017.mit.edu/wp-content/uploads/sites/5/2017/12/Efros-NIPS-NN-17.pdf)