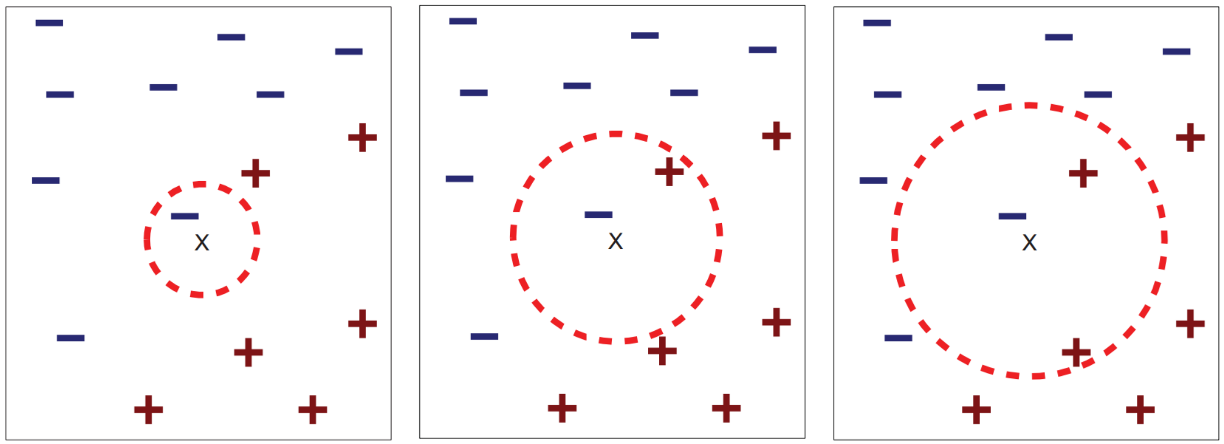
# k-Nearest Neighbors

고태훈 (taehoonko@dm.snu.ac.kr)

# k-nearest neighbors (k-NN)

❖ k-nearest neighbors (k-최근접 이웃) algorithm

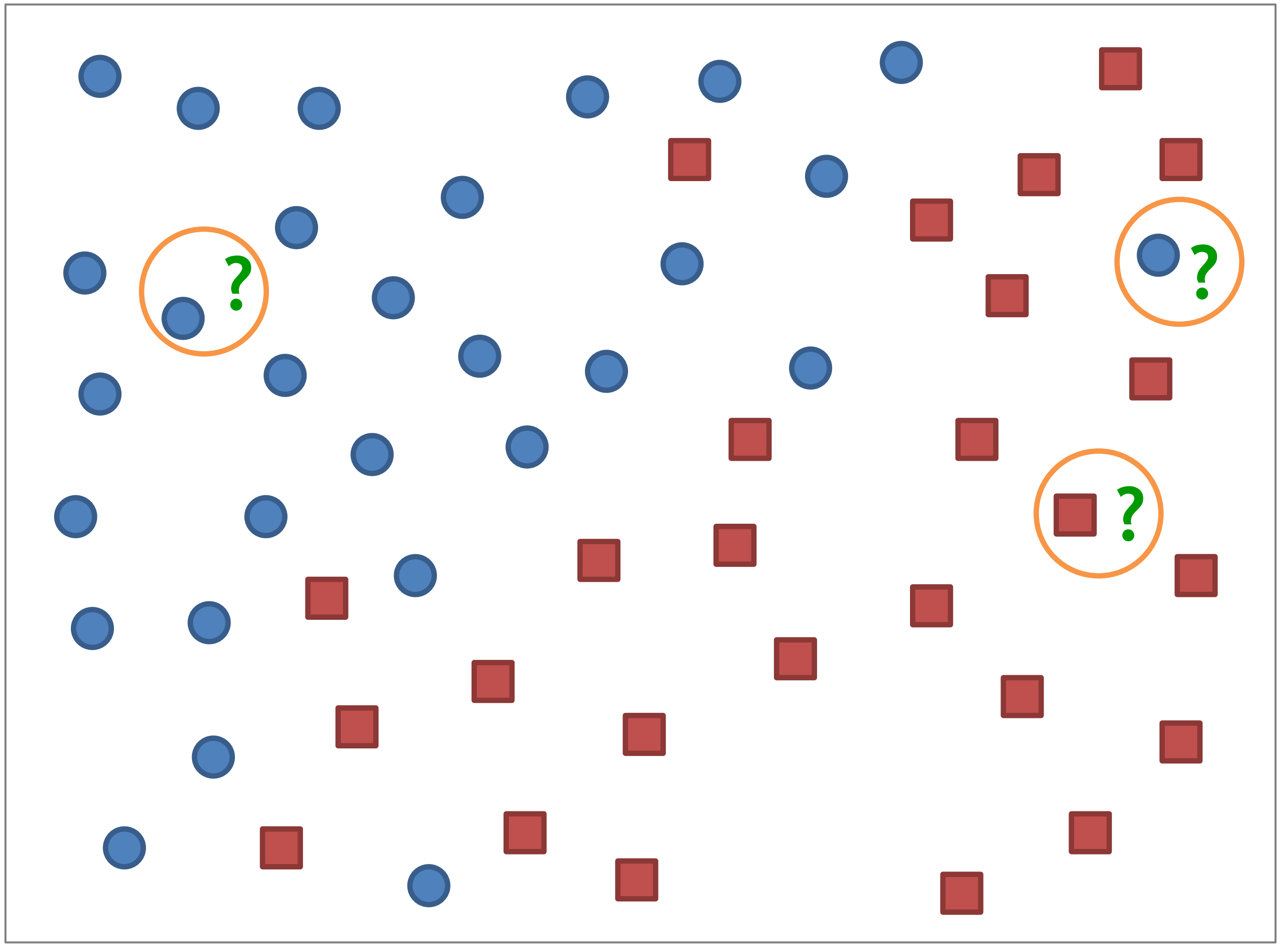
- 목표: 특정 포인트에 가장 가까운 k개의 이웃 포인트들을 참조하여 그 포인트의 출력변수 Y를 예측하는 알고리즘
- One of the simplest machine learning algorithms
- No explicit training or model
- Can be used both for classification and regression



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

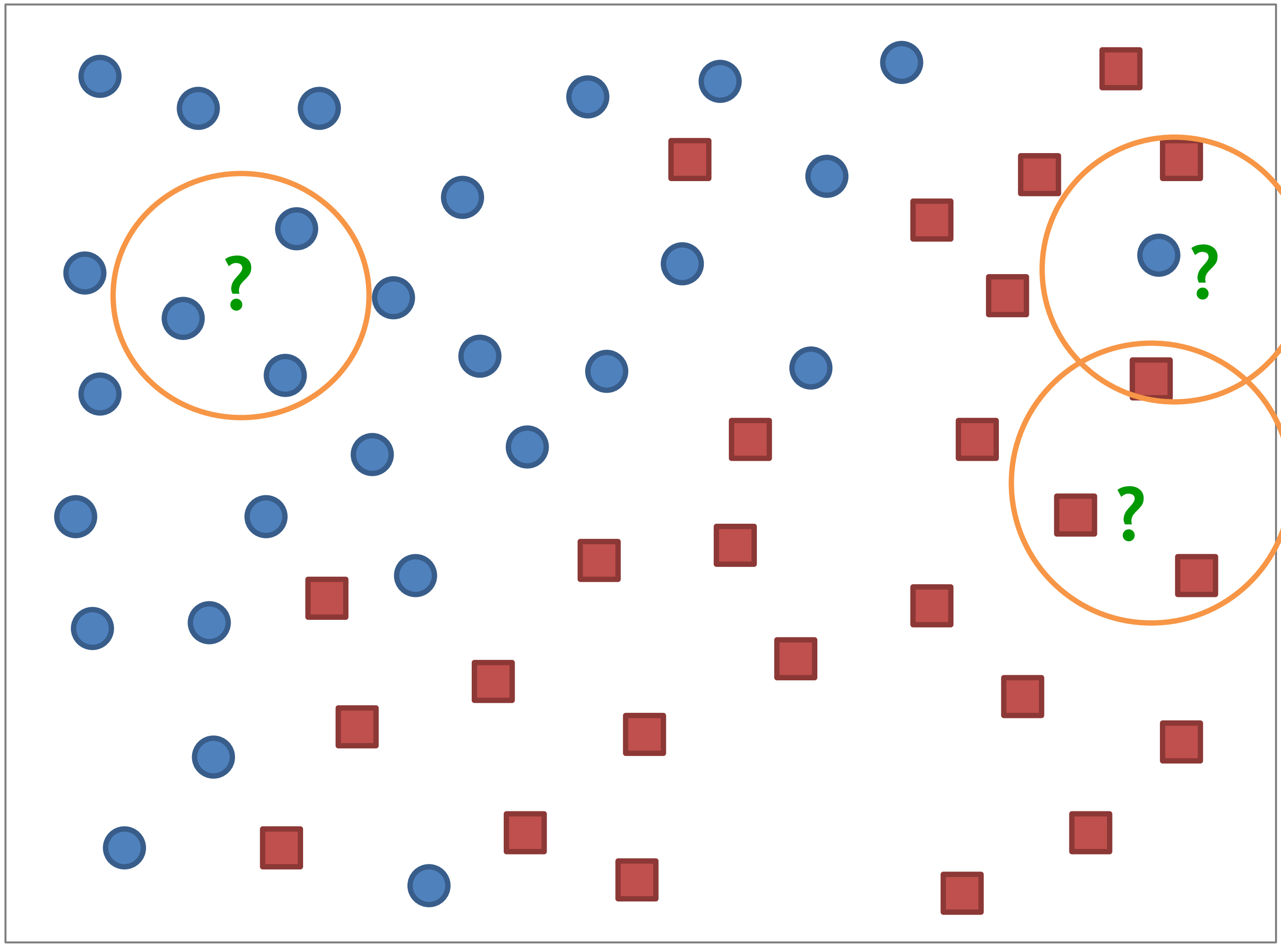
## k-NN classification: Example

- 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: Inference

- Given training data $(\mathbf{X}, \mathbf{Y})$
  - $\mathbf{X}$: Input variables
  - $\mathbf{Y}$: Output variable
- Suppose there is a new point $Q$

In computer science, this new point is called a "*query* point"

  - For $i$ in range(1,number of training points)
    - Compute $\boxed{\text{distance } d(X_i, Q)}$
  - Compute set $I$ containing indices for the $\boxed{k}$ smallest distances $d(X_i, Q)$
  - $\boxed{\textbf{Return } \hat{y} \text{ corresponding to the new point } Q \text{ using } \{y_i \text{ for } i \in I\}}$

# k-NN: Distance and similarity measures

## How to measure the distance $d(X, Q)$

training point: $X = (x_1, x_2, \cdots, x_p)^T$

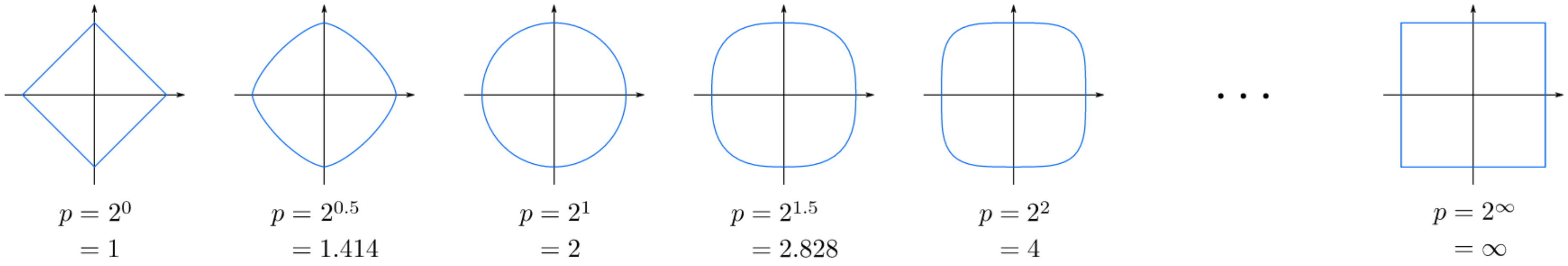
query point: $Q = (q_1, q_2, \cdots, q_p)^T$

❖ **Minkovski distance with order *p***

$$d(X,Q) = \left( \sum_{i=1}^{p} |x_i - q_i|^p \right)^{\frac{1}{p}}$$

- When p = 2, it is the **Euclidean distance**.

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



<Minkovski distance>

# k-NN: Distance and similarity measures

**How to measure the distance $d(X,Q)$**

**= How to measure the similarity $sim(X,Q)$**

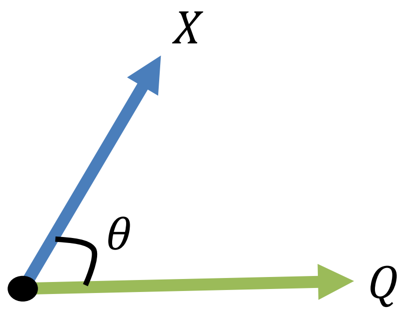
training point: $X = (x_1, x_2, \ldots, x_p)^T$

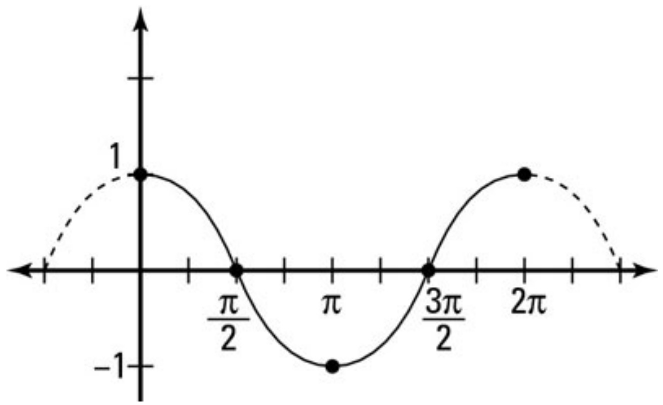
query point: $Q = (q_1, q_2, \ldots, q_p)^T$

## Cosine similarity

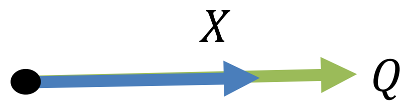
- Bounded between -1 and 1
- Bounded between 0 and 1 if *X* and *Q* are nonnegative

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

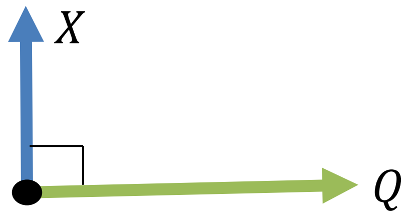




$$\cos 0 = 1$$



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



# k-NN: Distance and similarity measures

How to measure the distance $d(X,Q)$

= How to measure the similarity $sim(X,Q)$

training point: $X = (x_1, x_2, \cdots, x_p)^T$

query point: $Q = (q_1, q_2, \cdots, q_p)^T$

## Pearson's correlation coefficient

- Bounded between -1 and 1
- Equal to cosine similarity with zero-centered *X* and *Q*

$$sim\left(X,Q\right) = r(X,Q) = \frac{\sum_{i=1}^{p}(x_i - \overline{x})(q_i - \overline{q})}{\sqrt{\sum_{i=1}^{p}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{p}(q_i - \overline{q})^2}}$$

# 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 |

## How to represent distance in terms of similarity

- **distance = 1 - similarity**
  - cosine similarity는 -1과 1사이의 값을 갖는다.
  - 1 - cosine similarity는 0과 2사이의 값을 갖고 거리 척도로 쓰일 수 있다.

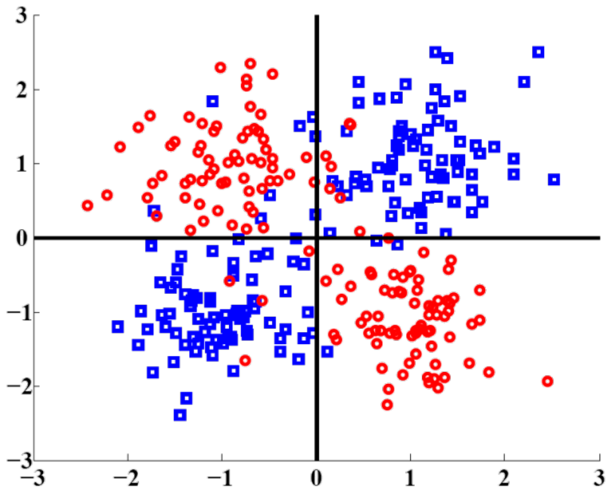
# k-NN: Distance and similarity measures

- **Euclidean distance**
  - 가장 널리 쓰이는 거리 지표
  - 입력변수값의 scaling이 반드시 필요
    - ex) Income varies 10,000-1,000,000 while height varies 1.5-1.8 meters
    - Normalization or Standardization!
  - 입력변수의 수가 많아지면(즉, 포인트의 차원이 커지면) 거리가 증가하는 측면이 있음
- **Cosine similarity and Pearson's correlation coefficient**
  - 입력변수 간의 scale 차이가 영향을 미치지 않음
  - 데이터가 Sparse matrix (희소행렬) 인 경우, Euclidean distance보다 더 나은 선택이 될 수 있음
    - ex) Document-term matrix for text mining

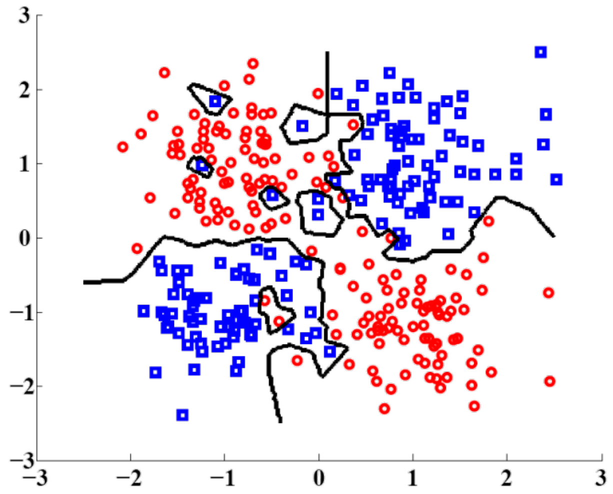
## k-NN: How to select *k*

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

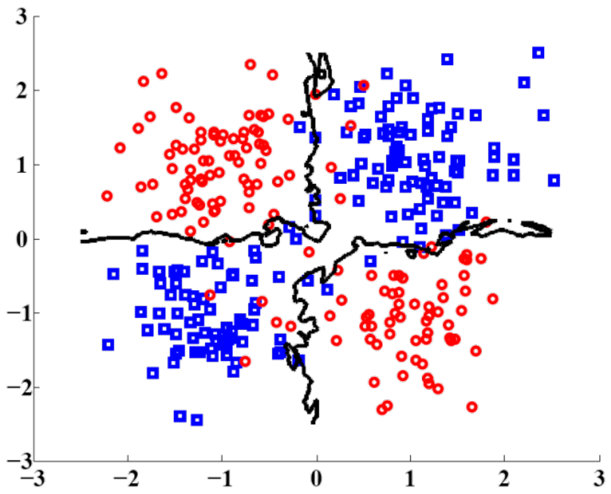
## k-NN: How to select *k*



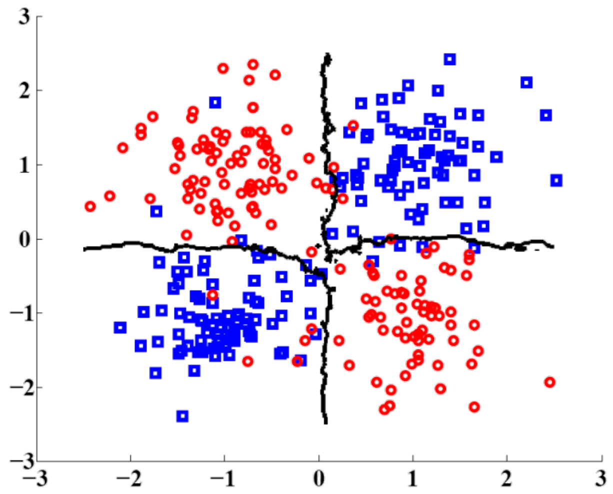
(a) The ideal boundary.



(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 classify a new point

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

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

- weight for $i^{\text{th}}$ 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

❖ **Example 1: k=5**

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 <u>**F** by the majority voting</u>, while classified as <u>**M** by the weighted voting</u>

## k-NN: How to classify a new point

- **Example 2: Considering the cut-off value with k = 5**
  - Assume that $N(C_M) = 100$, $N(C_F) = 400$

For a new point **Q**

| Neighbor | Class |
|---|---|
| N1 | M |
| N2 | F |
| N3 | M |
| N4 | F |
| N5 | F |

**Majority voting**

**P(X=M)=0.4**

- If the cut-off is set to 0.5 (assuming equal class distribution), then Q is classified as **F**.
- If the cut-off is set to 0.2 (proportion of M among the people), then Q is classified as **M**.

## k-NN: How to predict of output value of a new point

- Simple average vs. Weighted average
- Example 1: k=5

For a new point **Q**

| Neighbor | BFS | 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

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

- Weighted average

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

## k-NN: Pros and Cons

- **Pros**
    - Simple and powerful. No need for tuning complex parameters to build a model.
    - No training involved (“lazy”). New training examples can be added easily.

## k-NN: Pros and Cons

- **Cons**
  - Expensive and slow: O(md), m= # examples, d= # dimensions
    - To determine the nearest neighbor of a new point x, must compute the distance to all m training examples. Runtime performance is slow, but can be improved.
      - Pre-sort training examples into fast data structures
      - Compute only an approximate distance
      - Remove redundant data (condensing)