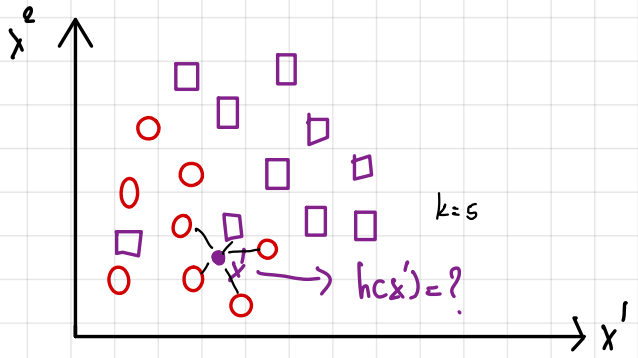
k-Nearest Neighbor (k-NN)

- feature vector เป็น point ใน space

e.g. $x = \mathbb{R}^2 \rightarrow \vec{x}$ is a 2D-point



$D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$

$x_i \in \mathbb{R}^2$, $y_i \in \{\square, O\}$

ค่า k

การ: รู้ h(x') ได้ต้องหา เพื่อนบ้านที่ใกล้ที่สุด (k)

- ควรเป็น odd
- ปรับ k ให้ loss ต่ำสุด

- Training: store all data points
- Testing: For a test point x', the prediction h(x') is determined by the majority of its nearest neighbors. (k)
- k-NN's assumption: close points should have similar labels

closness => similarity

Formal definition of k-NN

- given a test point $\vec{x}$
- Denote $S_{\vec{x}}$ the set of k-NN of $\vec{x}$

$S_{\vec{x}} \subseteq D$ such that $|S_{\vec{x}}| = k$ and

$\forall (\vec{x}', y') \in D \setminus S_{\vec{x}}$ (complement S_x) distance$(\vec{x}, \vec{x}') \geq$ max distance$(\vec{x}, \vec{x}'')$ $(\vec{x}'', y'') \in S_x$

หยิบ $S_{\vec{x}}$ มาใน D โดยที่ขนาดของ $S_{\vec{x}} = k$

และพิจารณา $\vec{x}', y'$ ใน D ที่ไม่อยู่ใน $S_{\vec{x}}$

- The classifier h() is define as:

$$h(\vec{x}) = \text{mode}(\{y'' : (x'', y'') \in S_x\})$$

Distance function

- เป็นการหาระยะทางระหว่าง x' กับ ข้อมูลใกล้ๆ เพื่อหา knn ที่ใกล้ที่สุด
- the most common choice is the "minkowski distance"

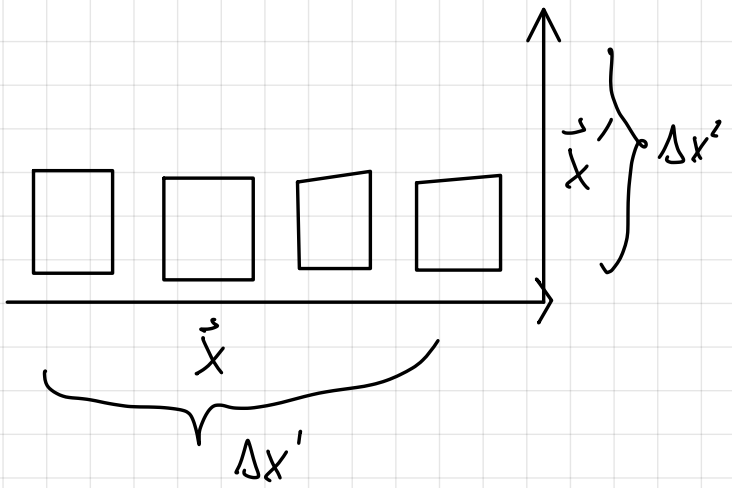
$$\text{distance}(\vec{x}, \vec{x}') = \left(\sum_{i=1}^{d} |\vec{x}^i - \vec{x}'|^p \right)^{1/p}$$

Example

① $p=1$ (Manhattan distance)

$$\vec{z} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} \quad \vec{z}' = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

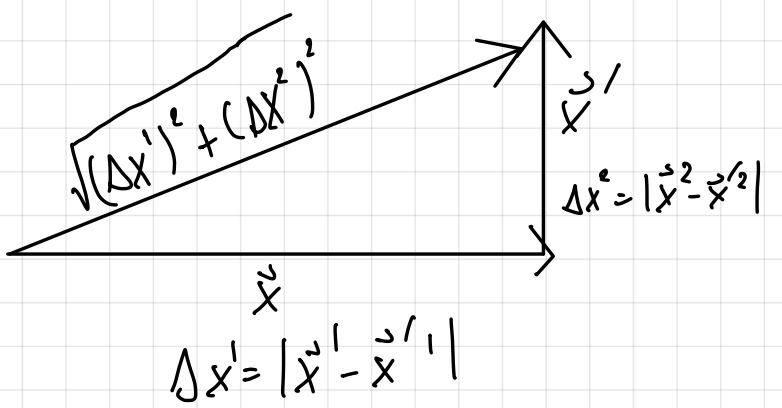
$$\text{dist}(\vec{z}, \vec{z}') = \left| \begin{bmatrix} 2 \\ 5 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix} \right| = \left[|2-5| + |5-3| \right] = 5$$



② $p=2$ (Euclidean distance)

$$\text{dist}(\vec{z}, \vec{z}') = \left[|2-5|^2 + |5-3|^2 \right]^{\frac{1}{2}}$$

$$= \sqrt{9+4} = \sqrt{13}$$



③ กรณี $-p: +\infty$

$-p: -\infty$

$p: \infty$

$$: \left(|2-5|^{\infty} + |5-3|^{\infty} \right)^{\frac{1}{\infty}}$$

$$: (3^{\infty} + \underbrace{2^{\infty}}_{\ll 3^{\infty}})^{\frac{1}{\infty}} = (3^{\infty})^{\frac{1}{\infty}} \approx 3$$

$p: -\infty$

$$\left(\underbrace{3^{-\infty}}_{\ll 2^{-\infty}} + 2^{-\infty} \right)^{-\frac{1}{\infty}} \approx 2$$

Implement of k-NN:

- Assume we already store all the training data

Step 1: The algorithm simply "sweps" through all the traning data points, and calculates the distance from the test point to each particular point at the same time. (complexity of step 1 is O(dn))

Step 2: the algorithm sorts the training data points using its distance to the test point as keys, and pick the points whose distance is k-smallest (O(n log n))

Overall: O(dn) + O(n log n) = O(dn)

Pros and Cons of k-NN

- Given a good and small-dimensional data set (d = small), k-NN can give a good classifier (with high accuracy of prediction)

- However, k-NN has 2 main downsides when used with high-dimensional data set:

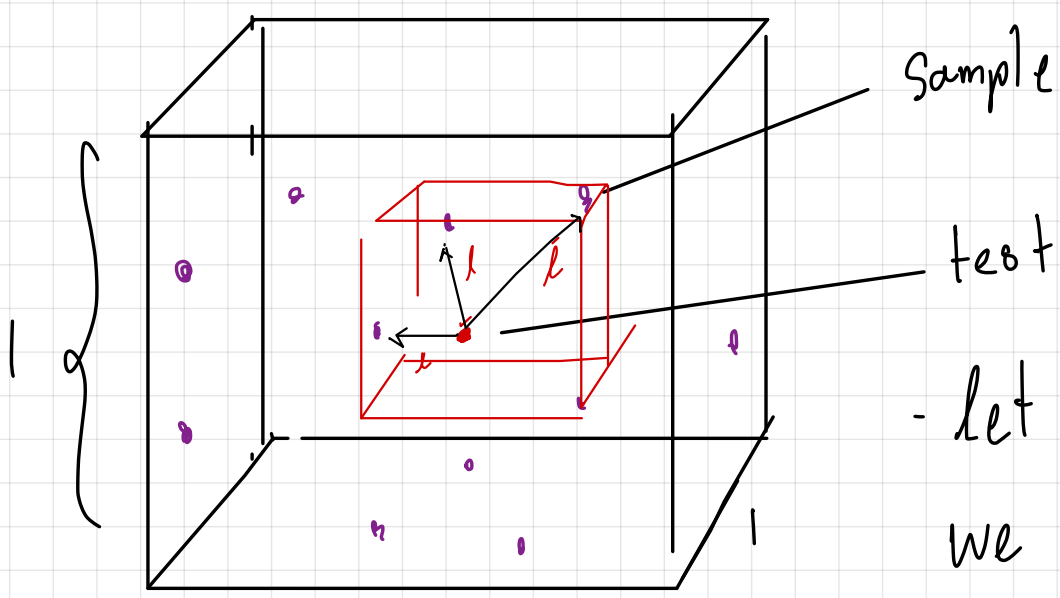
① The classifier can be very slow when either n or d is large (or both).

② If our data is high-dimensional (d>>0 ; e.g. image data), k-NN will give a bad classifier (know as "curse of Dimensionality") ถ้าจะแก้ต้องมี data sample เยอะๆ

Curse of Dimensionality

- The phenomenon that data points in high dimensional space are down from a probability distributional, and they tend to be far apart from each other (there is no close points in high dimesional space; this is where the k-NN's assumption can go very wrong)

Illustration: Imagine the unit cube $[0,1]^d$, and all training data points are sampled uniformly at random with in the cube.



- we can think of this cube as a box of Vol $|x_1 .. x_1| = 1^d$ containing n points

- let think of k=10, Then in expectation we can hope to find $S_{\vec{x}}$ of a test point $\vec{x}$ with in a smaller cube whose length is some value l. This means we should imagine that this smaller cube is of volume l^d and it should contain k points out of n points

- ในกรณี d สูง แทบไม่เหลือ 100 k ที่ใกล้

Q: How big is l?

A: $l^d \simeq \frac{k}{n} \Rightarrow l \approx \left(\frac{k}{n}\right)^{\frac{1}{d}}$

l เปลี่ยนตาม n, k

Simulation under k=10

n	d	l
1000	2	0.1
1000	10	0.63
1000	100	0.955
1000	1000	0.9954

→ <u>observation</u>: As d becomes larger, all k-NN tend to be far from $\vec{x}$