기계학습 (2022년도 2학기)

Nearest Neighbors

전북대학교 컴퓨터공학부

Inductive Learning (귀납적 학습)

- Given a training set of examples of the form $(x, f(x))$
  - x is the input, $f(x)$ is the output

- Return a function h that approximates f
  - h is called the hypothesis (가설)

Hypothesis Space (가설공간)

- Hypothesis space H
  - Set of all hypotheses h that the learner may consider
  - Learning is a search through hypothesis space

- Objective: find h that minimizes
  - Misclassification
  - Or more generally some error function

  with respect to the training examples

- But what about unseen examples?

Generalization (일반화)

- A good hypothesis will generalize well
  - i.e., predict unseen examples correctly
    (즉, 처음 보는 입력들도 정확하게 예측해야 함)

- Usually...
  - Any hypothesis h found to approximate the target function f well over a sufficiently large set of training examples will also approximate the target function well over any unobserved examples

→ 우리가 다루는 예측 문제의 모든 출력을 표현하는 함수를 target function f(실제로는 알 수 없음)라고 할 때

→ 충분히 큰 학습 데이터에 대해 어떤 가설 h가 잘 동작(예측)한다면, 일반적으로 가설 h는 (알 수 없는) 분포 함수 f를 잘 근사한다고 생각

Inductive Learning

- Goal: find an h that agrees with f on a training set
  - h is **consistent** if it agrees with f on all examples

- Finding a consistent hypothesis is not always possible
  - Insufficient hypothesis space:
    - E.g., it is not possible to learn exactly $f(x) = ax + b + xsin(x)$ when H = space of polynomials of finite degree
  - Noisy data
    - E.g., in weather prediction, identical conditions may lead to rainy and sunny days

Inductive Learning

- A learning problem is **realizable** if the hypothesis space contains the target function otherwise it is **unrealizable**
  - Difficult to determine whether a learning problem is realizable since the target function is not known

- It is possible to use a very large hypothesis space
  - For example: H = class of all Turing machines
    (계산 가능한 모든 문제들을 풀 수 있는 가상 기계)

- But there is a **tradeoff** between **expressiveness** of a hypothesis class and the **complexity** of finding a good hypothesis

Introduction

- We're focused on supervised learning.
- This means we're given a training set consisting of inputs and corresponding labels
- Machine learning - learning a program. Labels are the expected output of the correct program when given the inputs.

Task	Inputs	Labels
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
⋮	⋮	⋮

- Goal: correctly predict labels for data not in the training set (“in the wild") i.e. our ML algorithm must generalize

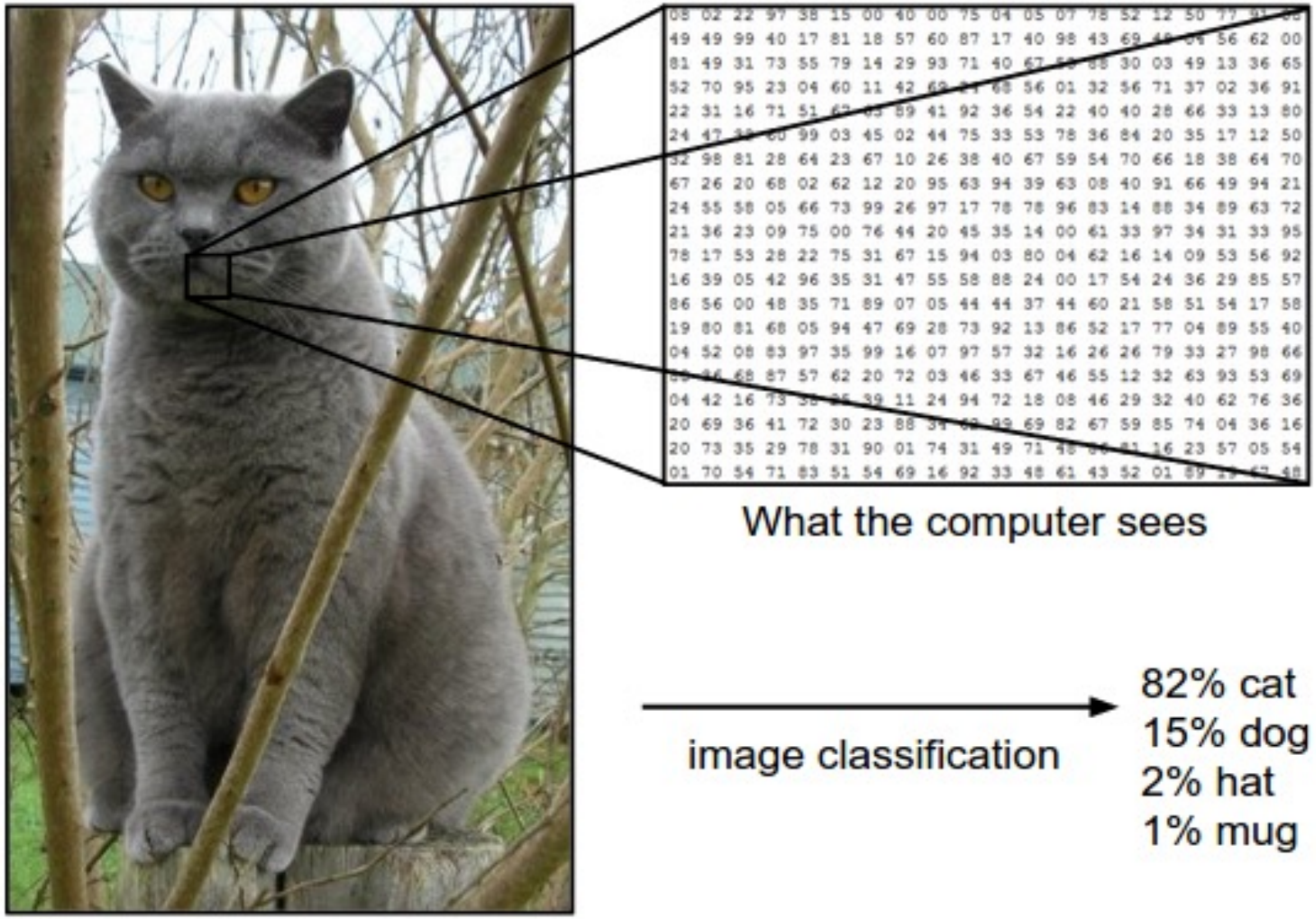
Input Vectors

- Machine learning algorithms need to handle lots of types of data: images, text, audio waveforms, credit card transactions, etc.

- Common strategy: represent the input as an input vector in $\mathbb{R}^d$
  - Representation = mapping to another space that's easy to manipulate
  - Vectors are a great representation since we can do linear algebra!

Input Vectors

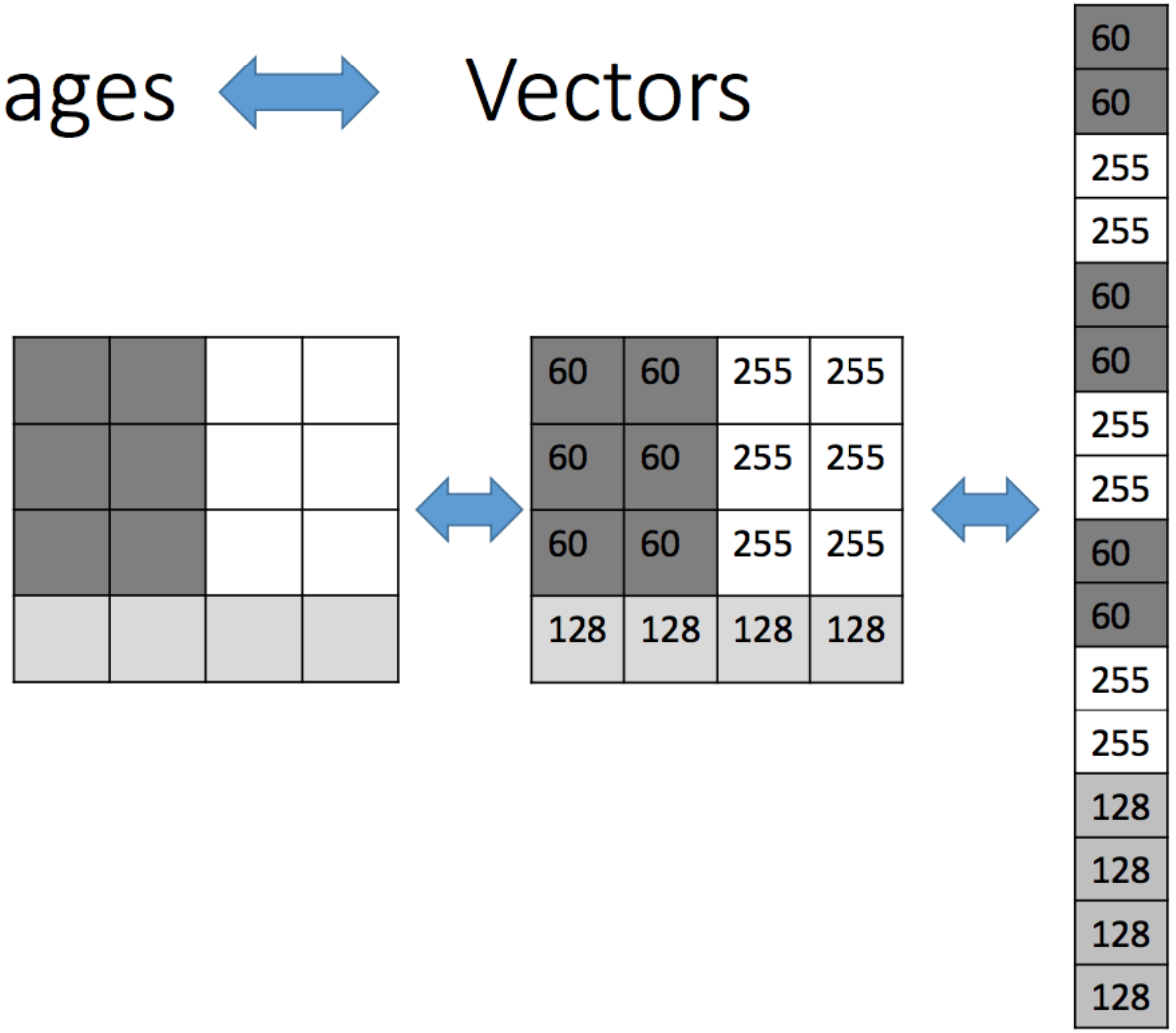
- What an image looks like to the computer:



[Image credit: Andrej Karpathy]

Input Vectors

- Can use raw pixels:



Can do much better if you compute a vector of meaningful features.

Input Vectors

- Mathematically, our training set consists of a collection of pairs of an input vector $\mathbf{x} \in \mathbb{R}^d$ and its corresponding target, or label, t
  - Regression: t is a real number (e.g. stock price)
  - Classification: t is an element of a discrete set $\{1, \ldots, C\}$
  - These days, t is often a highly structured object (e.g. image)

- Denote the training set $\{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$
  - Note: these superscripts have nothing to do with exponentiation!

Nearest Neighbors

- Suppose we're given a novel input vector ***x*** we'd like to classify.
- The idea: find the nearest input vector to ***x*** in the training set and copy its label.
- Can formalize “nearest" in terms of Euclidean distance

$$||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2}$$

Algorithm:

1. Find example $(\mathbf{x}^*, t^*)$ (from the stored training set) closest to $\mathbf{x}$. That is:

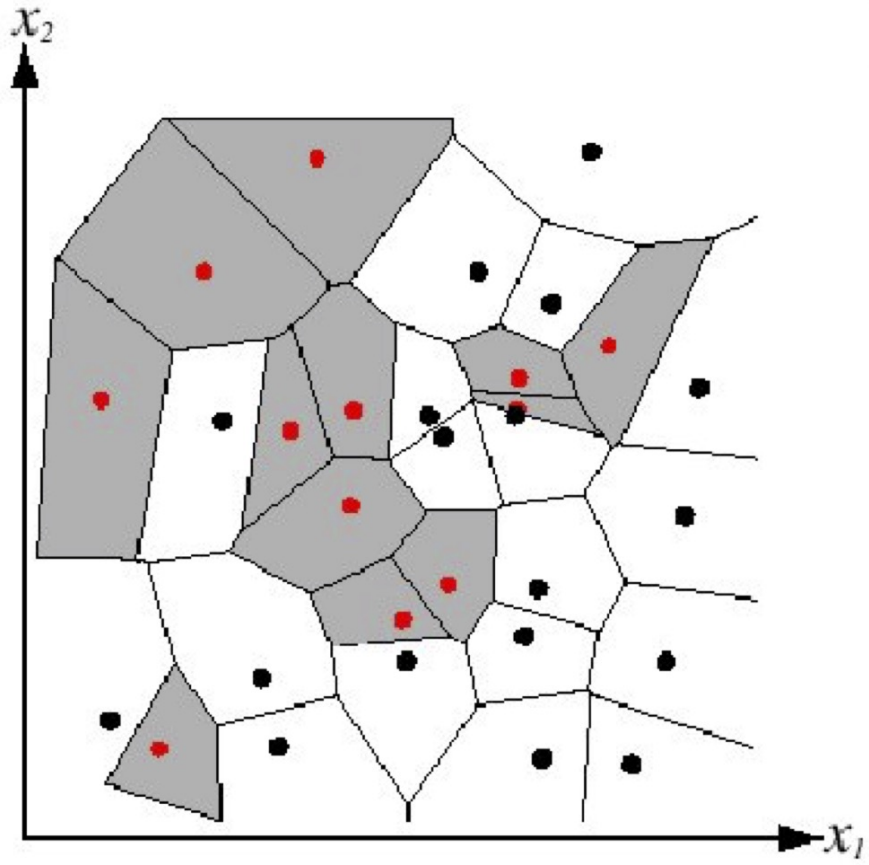
$$\mathbf{x}^* = \underset{\mathbf{x}^{(i)} \in \text{train. set}}{\text{argmin}} \ \text{dist}(\mathbf{x}^{(i)}, \mathbf{x})$$

2. Output $y = t^*$

- Note: we don't need to compute the square root. Why?

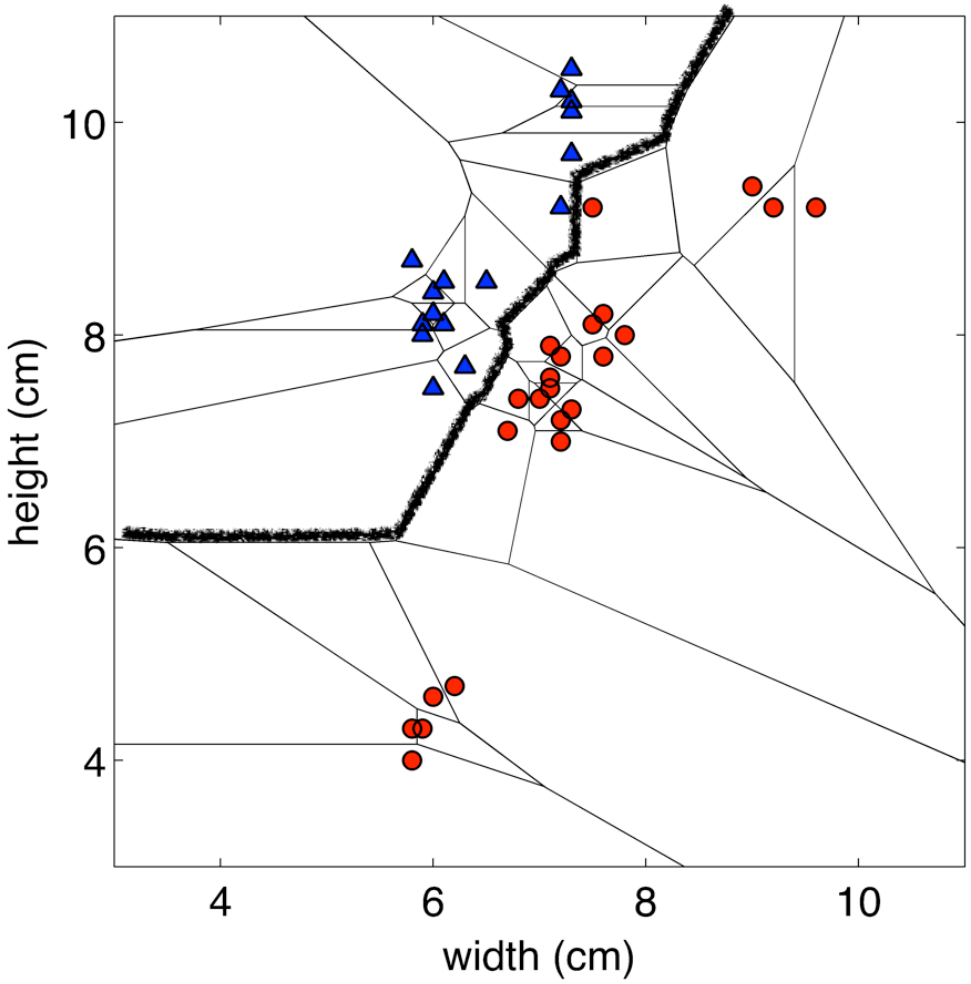
Nearest Neighbors: Decision Boundaries

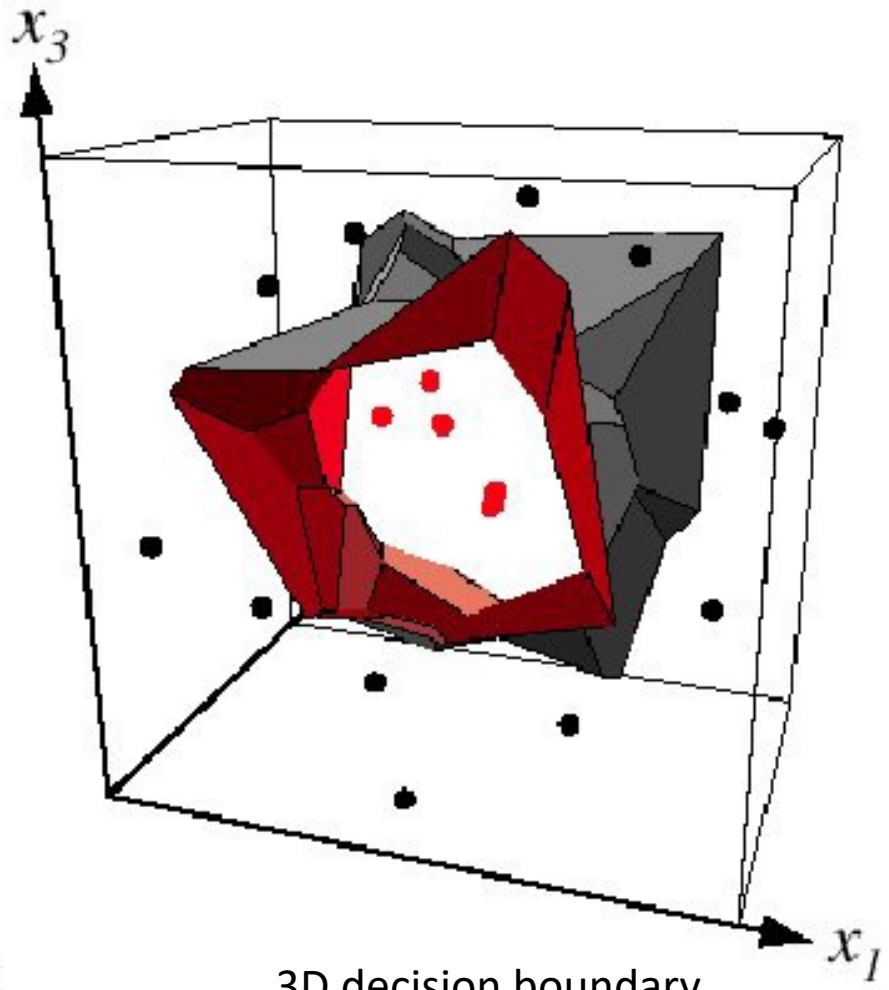
- We can visualize the behavior in the classification setting using a Voronoi diagram.
  - Partition implied by nearest neighbor assuming Euclidean distance



Nearest Neighbors: Decision Boundaries

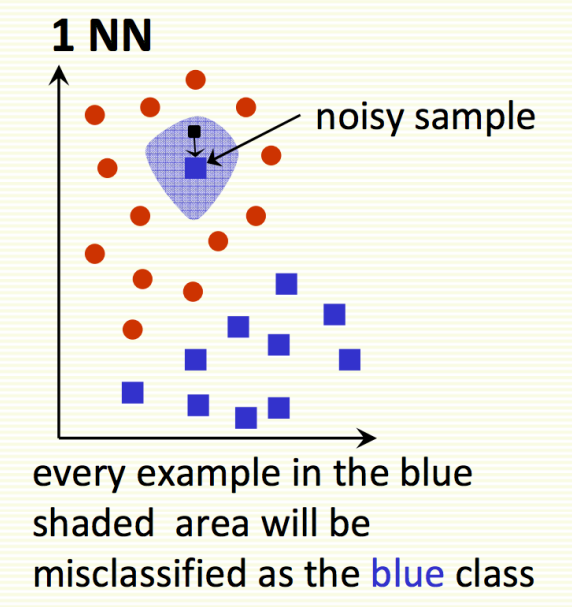
- **Decision boundary**: the boundary between regions of input space assigned to different categories.

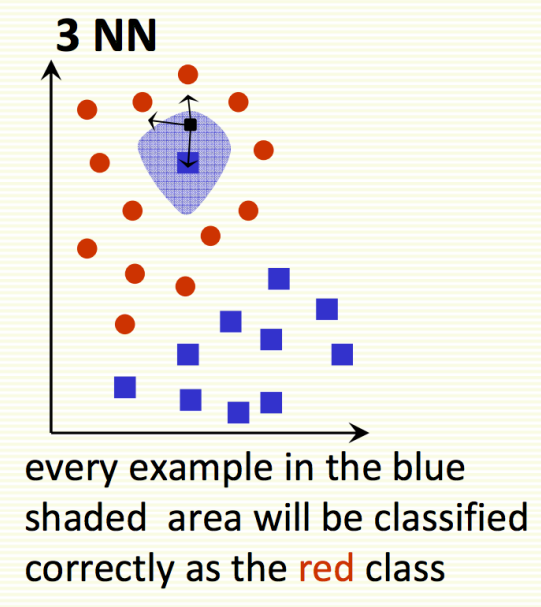




3D decision boundary

k-Nearest Neighbors





- Nearest Neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?
- Smooth by having k nearest Neighbors vote

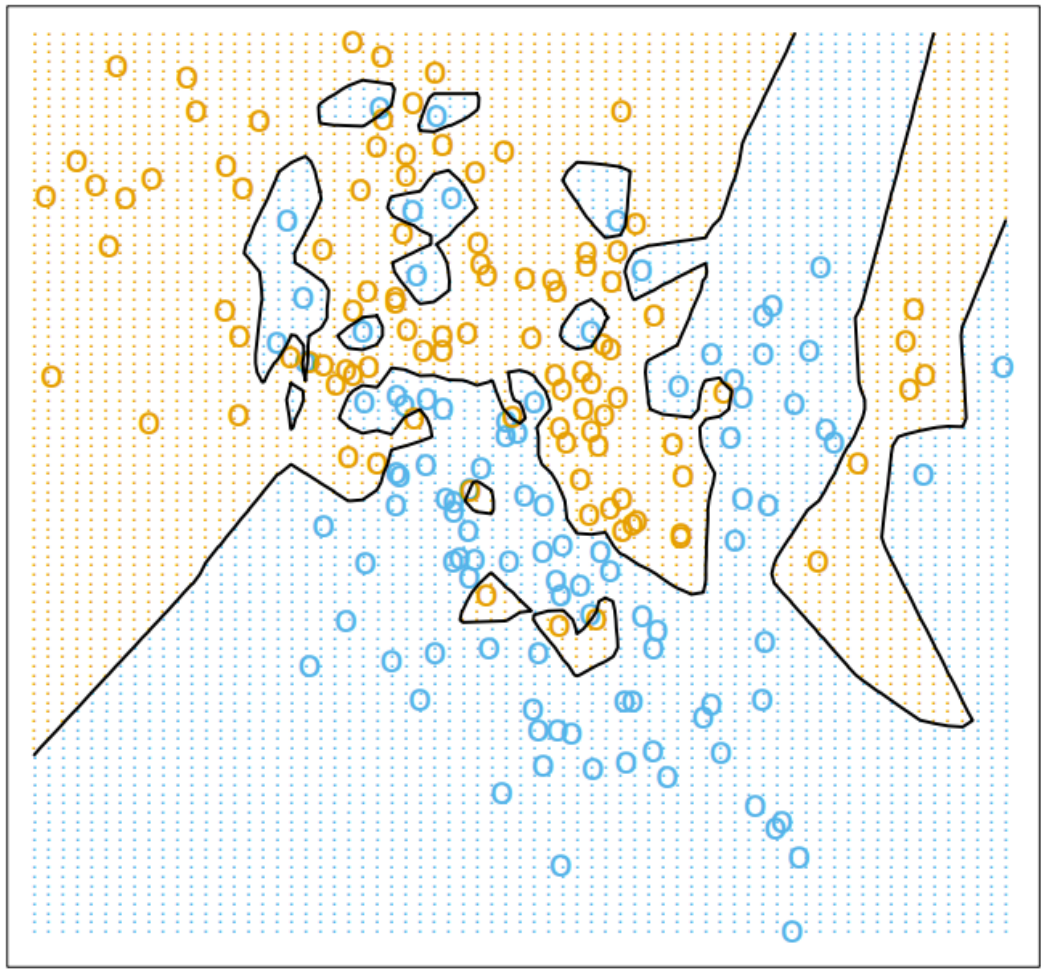
Algorithm (kNN):

1. Find k examples $\{(\mathbf{x}^{(r)}, t^{(r)})\}_{r=1}^{k}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

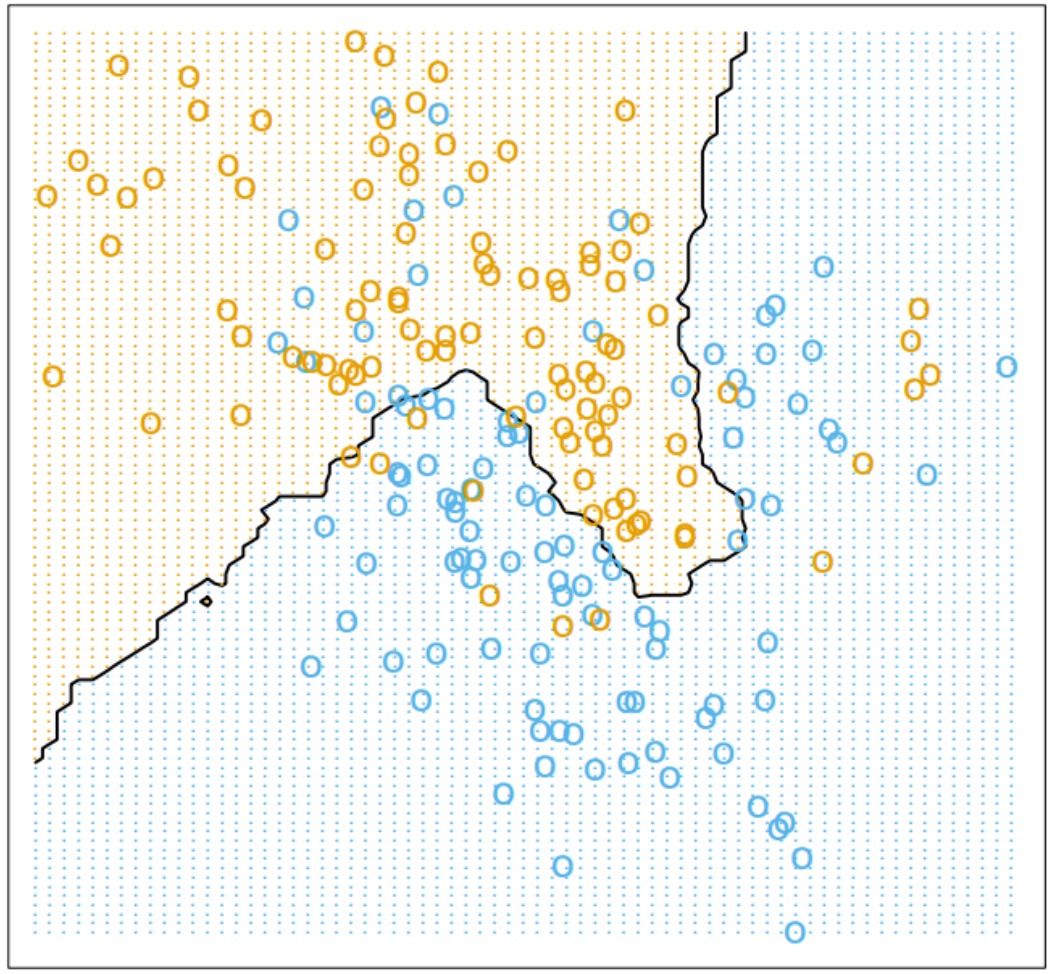
$$y = arg\max_{t} \sum_{r=1}^{k} \mathbb{I}[t = t^{(r)}]$$

K-Nearest Neighbors

k=1



k=15



[Image credit: "The Elements of Statistical Learning"]

k-Nearest Neighbors

- Tradeoffs in choosing *k*? Remember: the goal is to correctly classify unseen examples
- Small *k*
  - Good at capturing fine-grained patterns
  - May overfit, i.e. be sensitive to random idiosyncrasies in the training data

  (Overfit의 주요 원인: Classifier가 너무 복잡(too expressive), Noisy data, Lack of data)
- Large *k*
  - Makes stable predictions by averaging over lots of examples
  - May underfit, i.e. fail to capture important regularities

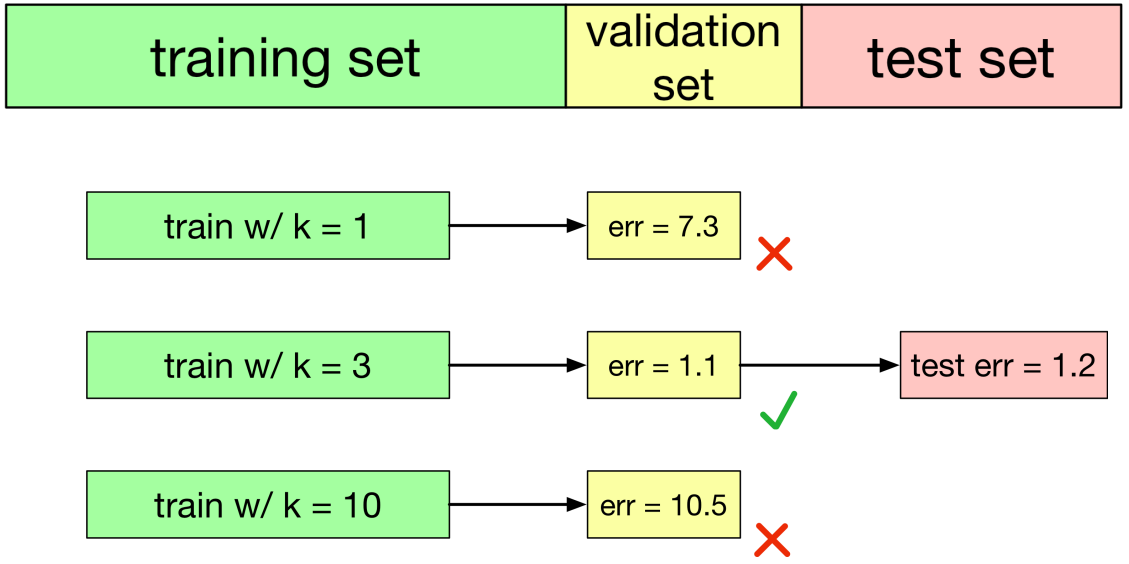
  (Underfit 원인: Classifier가 너무 단순 (not expressive enough))
- Rule of thumb: $k < \text{sqrt}(n)$, where n is the number of training examples

Choosing Hyperparameters using a Validation Set

- *k* is an example of a hyperparameter, something we can't fit as part of the learning algorithm itself, but which controls the behavior of the algorithm
  - 데이터를 통한 학습이 아닌 설계자가 미리 정해야 하는 값
- We want to choose hyperparameters based on how well the algorithm generalizes
- Thus, we separate some of our available data into a validation set, distinct from the training set
- Model's performance on the validation set indicates how well it generalizes
  - choose hyperparameters which leads to best performance (lowest error) on validation set
  - Note: error here means number of incorrectly classified examples

Test Set

- Now hyperparameters might have overfit to the validation set! Validation performance is not good assessment of generalization of final algorithm
- Solution: separate an additional test set from the available data and evaluate on it once hyperparameters are chosen
  - Available data partitioned into 3 sets: training, validation, and test



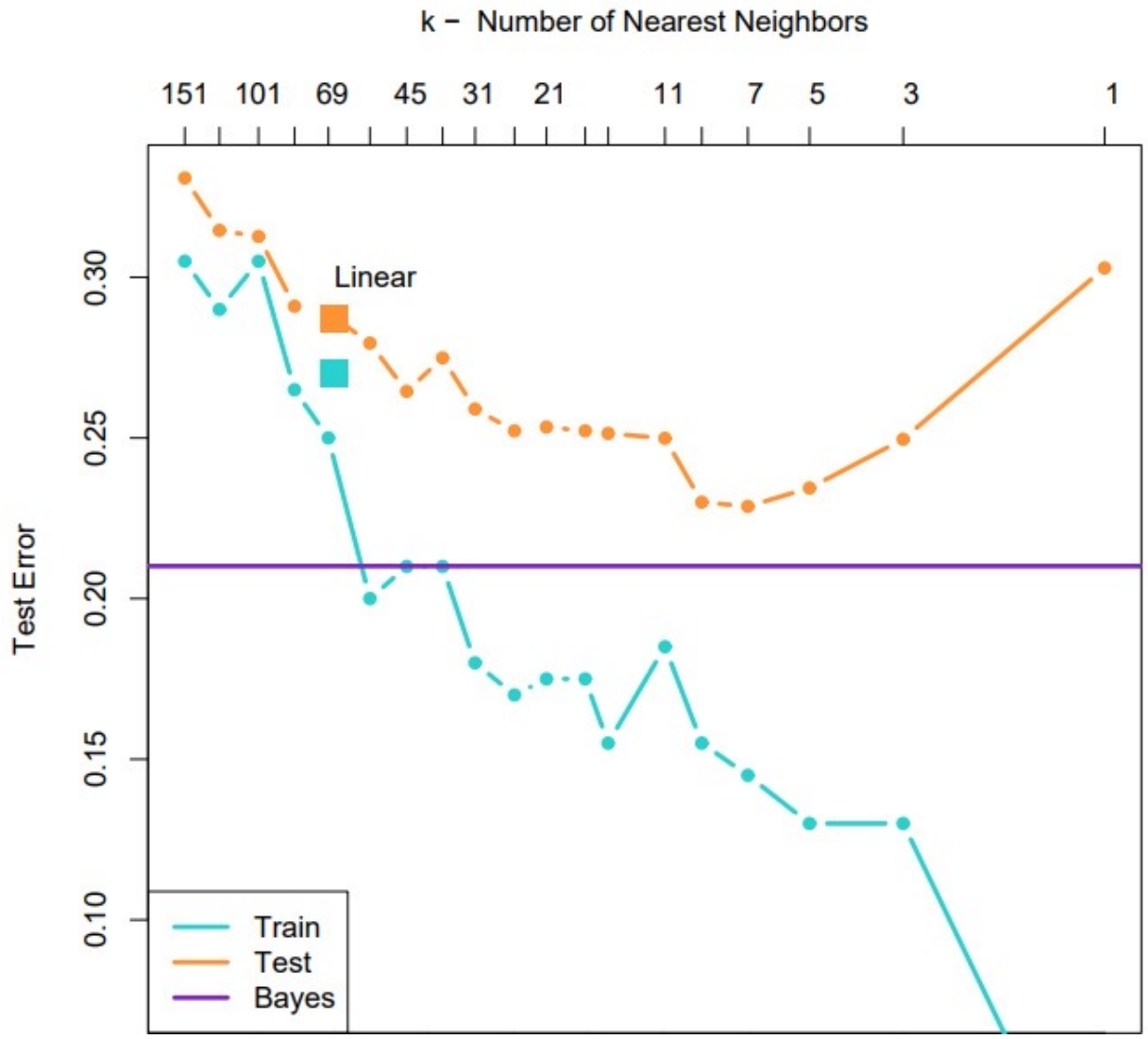
- The test set is used only at the very end, to measure the generalization performance of the final configuration.

The effect of K

- Best *k* depends on
  - Problem
  - Amount of training data

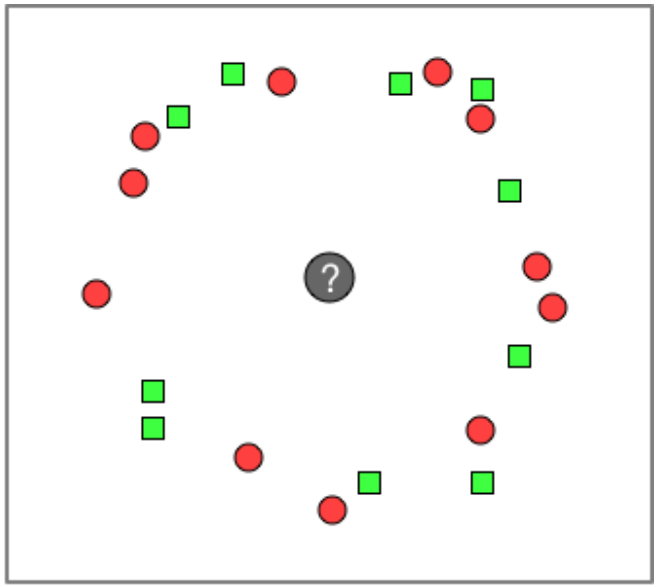
(*k*=1일 때 test error는?)

(underfitting이 발생하는 구간은?)



[Image credit: ”The Elements of Statistical Learning”]

The Curse of Dimensionality

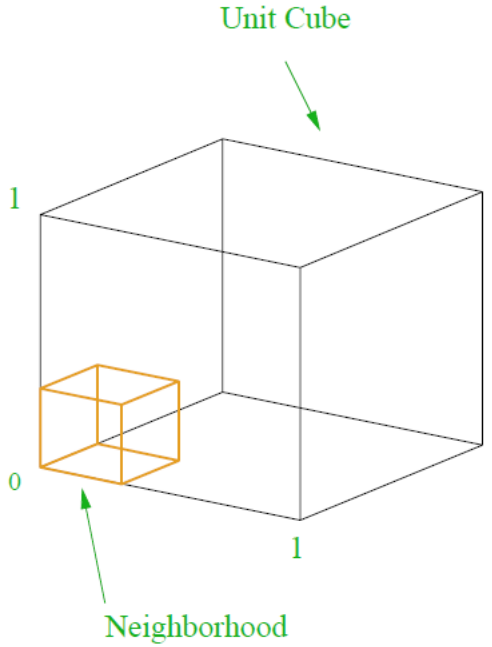


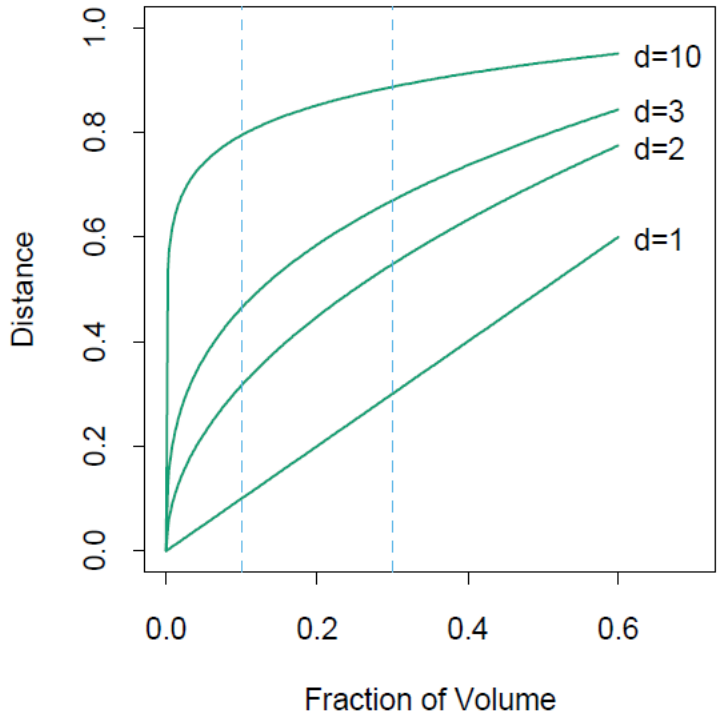
- Low-dimensional visualizations are misleading!
  - Given a new point, we want to classify it based on a point only a small distance away
  - But in high dimensions, “most" points are far apart.
- At least how many points are needed to guarantee the nearest neighbor is closer than ϵ?
  - The volume of a single ball of radius ϵ is $O(\epsilon^d)$
  - The total volume of $[0,1]^d$ is 1.
  - Therefore $O\left(\left(\frac{1}{\epsilon}\right)^d\right)$ balls are needed to cover the volume.
- Assuming data follows uniform distribution, training set size must grow exponentially with the number of dimensions for points to be close by!

The Curse of Dimensionality

- Edge length of hypercube required to occupy given fraction *r* of volume of unit hypercube $[0,1]^d$ is $r^{\frac{1}{d}}$ (edge 길이를 l 이라 하면 $l^d \approx r \times 1 \Rightarrow l \approx r^{\frac{1}{d}}$)
  - If *d* = 10 and *r* = 0.1, the edge length required is $0.1^{\frac{1}{10}} \approx 0.8$
  - To use 10% of the data to make our decision, must cover 80% of the range of each dimension

➔ 큰 차원 데이터를 다루는 문제일수록 많은 학습 데이터가 필요!





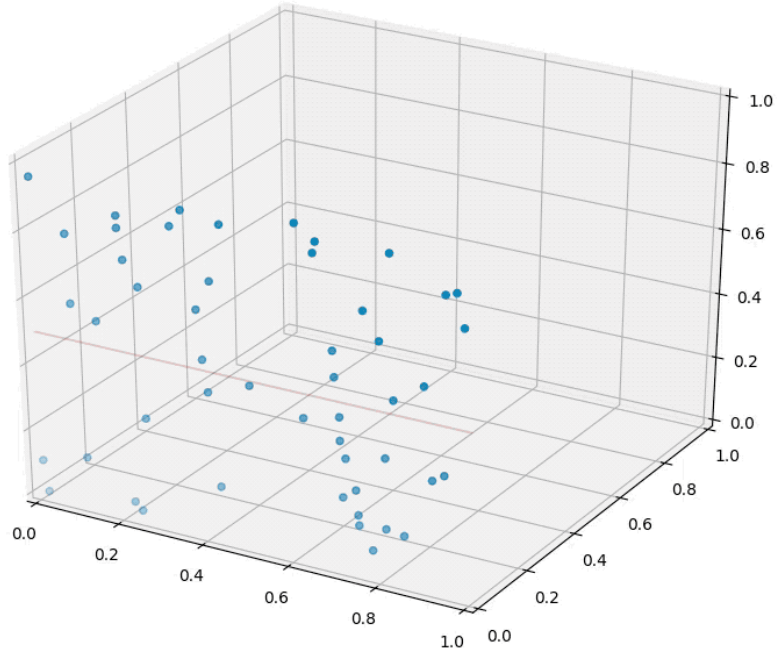
[Image credit: ”The Elements of Statistical Learning”]

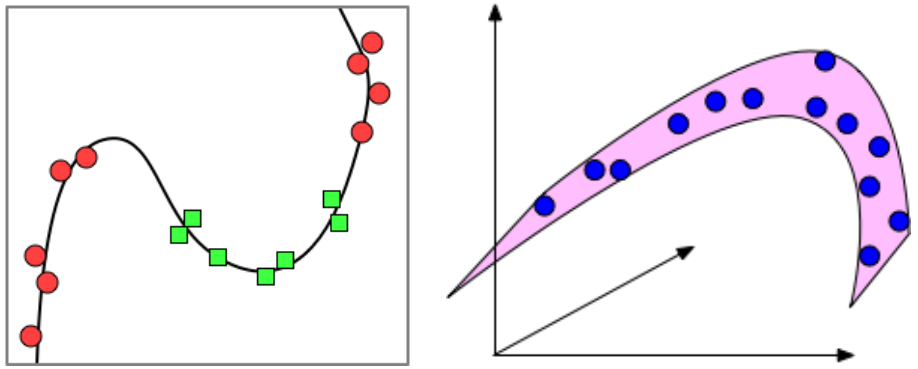
The Curse of Dimensionality

- In high dimensions, “most" points are approximately the same distance.
- Saving grace: some datasets (e.g. images) may have low intrinsic dimension, i.e. lie on or near a low-dimensional manifold

  → manifold: 고차원의 데이터를 저차원으로 비선형 축소(임베딩)할 때 데이터를 잘 설명하는 집합의 모형

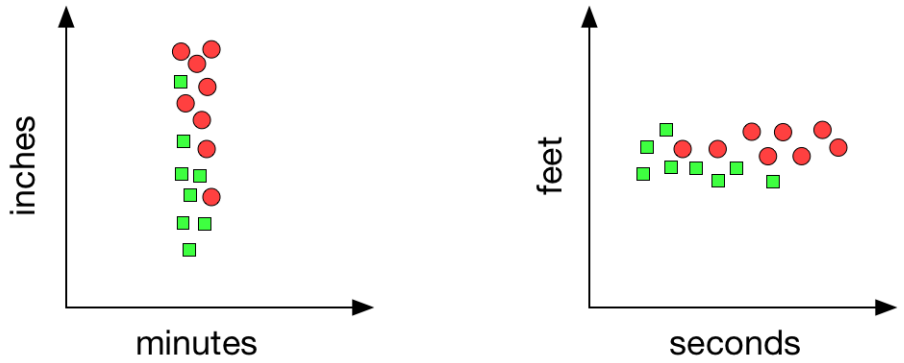
  - Human faces: 18M pixels → less than 50 attributes (e.g. male/female, blond/dark hair, ...) along which faces vary.
- So nearest Neighbours sometimes still works in high dimensions.





Normalization

- Nearest Neighbors can be sensitive to the ranges of different features.
- Often, the units are arbitrary:



- Simple fix: normalize each dimension to be zero mean and unit variance. I.e., compute the mean μ_j and standard deviation σ_j, and take

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

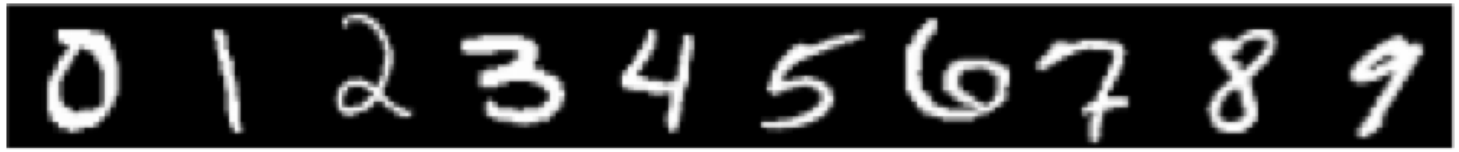
- Caution: depending on the problem, the scale might be important!
  - 특정 feature가 다른 feature보다 더 중요한 경우에는?

Computational Cost

- Number of computations at training time: 0
- Number of computations at test time, per query (naive algorithm)
  - Calculuate D-dimensional Euclidean distances with N data points: $O(ND)$
  - Sort the distances: $O(N \log N)$
- This must be done for each query, which is very expensive by the standards of a learning algorithm!
- Need to store the entire dataset in memory!
- Tons of work has gone into algorithms and data structures for efficient nearest Neighbours with high dimensions and/or large datasets.

Example: Digit Classification

- Decent performance when lots of data



- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: $d = 784$
  - 60,000 training samples
  - 10,000 test samples
- Nearest neighbour is competitive

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

Conclusions

- Simple algorithm that does all its work at test time, i.e. no learning!
- Can control the complexity by varying *k*
- Suffers from the Curse of Dimensionality

- Next time: decision trees, another approach to regression and classification