Machine Learning from Data

Lecture 17: Spring 2021

Today’s Lecture

- Memory and Efficiency of Nearest Neighbor

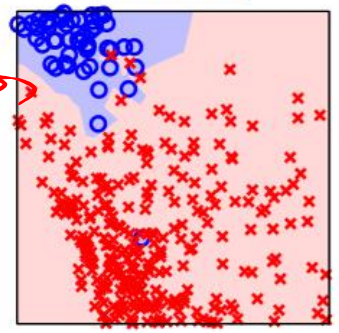
RECAP:

Similarity and Nearest Neighbor

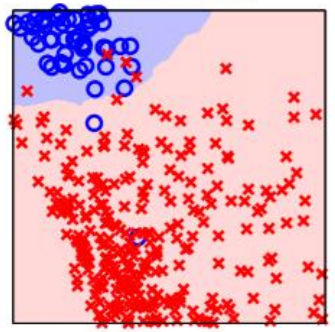
Similarity

$d(\mathbf{x}, \mathbf{x}') = \| \mathbf{x} - \mathbf{x}' \|$

1-NN rule



21-NN rule



1. Simple.
2. No training.
3. Near optimal E_{out}:

   $k \to \infty,\ k/N \to 0 \implies E_{\text{out}} \to E^*_{\text{out}}.$
4. Good ways to choose k:

   $k = 3$; $k = \lceil \sqrt{N} \rceil$; validation/cross validation.
5. Easy to justify classification to customer.
6. Can easily do multi-class.
7. Can easily adapt to regression or logistic regression

   $g(\mathbf{x}) = \frac{1}{k} \sum_{i=1}^{k} y_{[i]}(\mathbf{x})$ $\qquad$ $g(\mathbf{x}) = \frac{1}{k} \sum_{i=1}^{k} [\![y_{[i]}(\mathbf{x}) = +1]\!]$
8. **Computationally demanding.**

Computational Demands of Nearest Neighbor

1

Memory.

Need to store all the data, $O(Nd)$ memory.

$N = 10^6$, $d = 100$, double precision $\approx$ 1GB

2

Computational.

Finding the nearest neighbor of a test point.

Need to compute distance to every data point, $O(Nd)$.

$N = 10^6$, $d = 100$, 3GHz processor

$\approx$ 3ms (compute $g(\mathbf{x})$)

$\approx$ 1hr (compute CV error)

$>$ 1month (choose best k from among 1000 using CV)

Two Basic Approaches

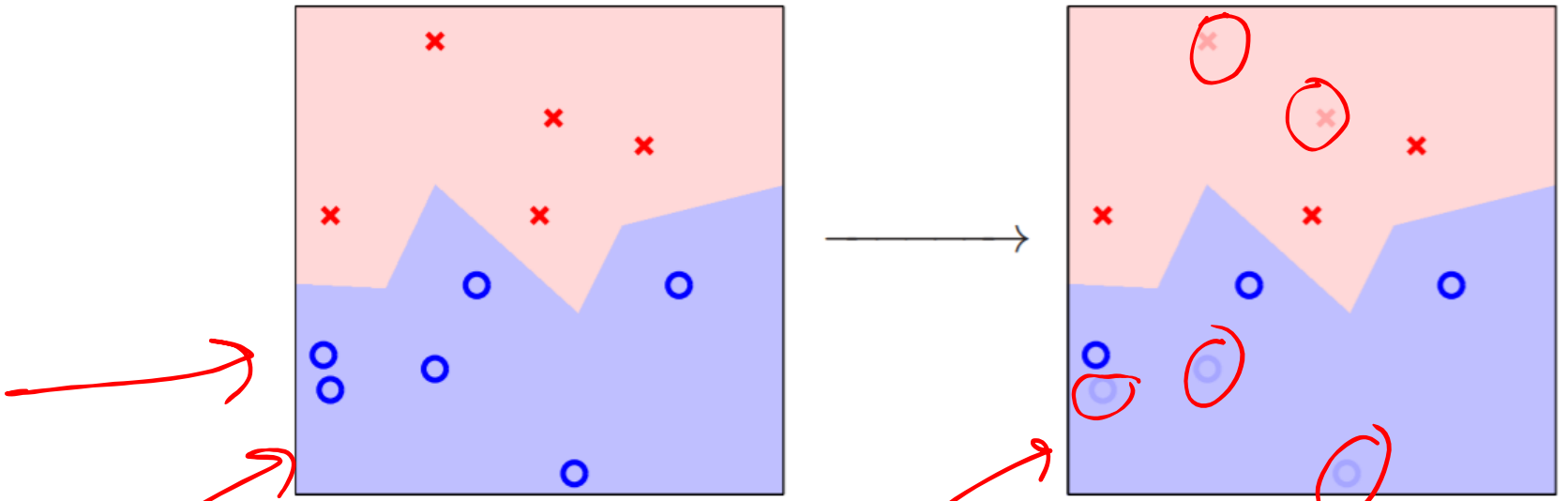
Reduce the amount of data.

The 5-year old does not remember every horse he has seen, only a few *representative* horses.

Store the data in a specialized data structure.

Ongoing research field to develop geometric data structures to make finding nearest neighbors fast.

Decision Boundary Consistent



Condensed

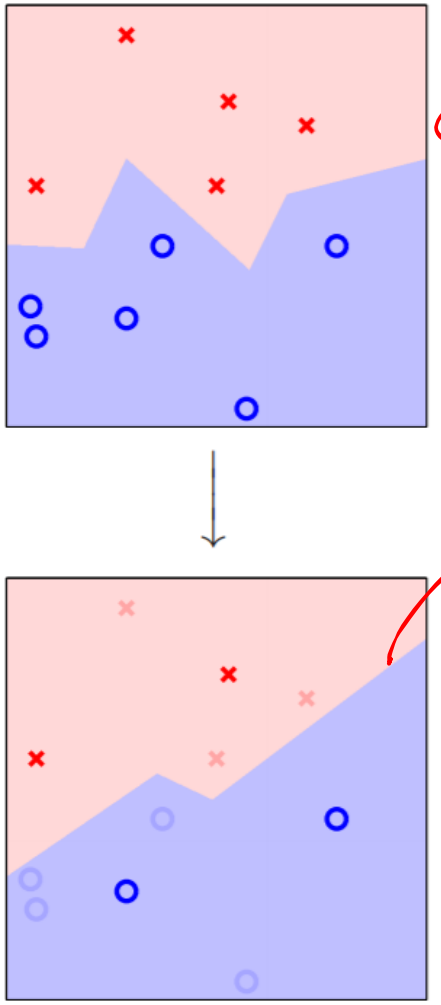
$R = 1$

E_{in} change? No

E_{out} change? No

$g(\mathbf{x})$ unchanged

Training Set Consistent



g

hypothesis

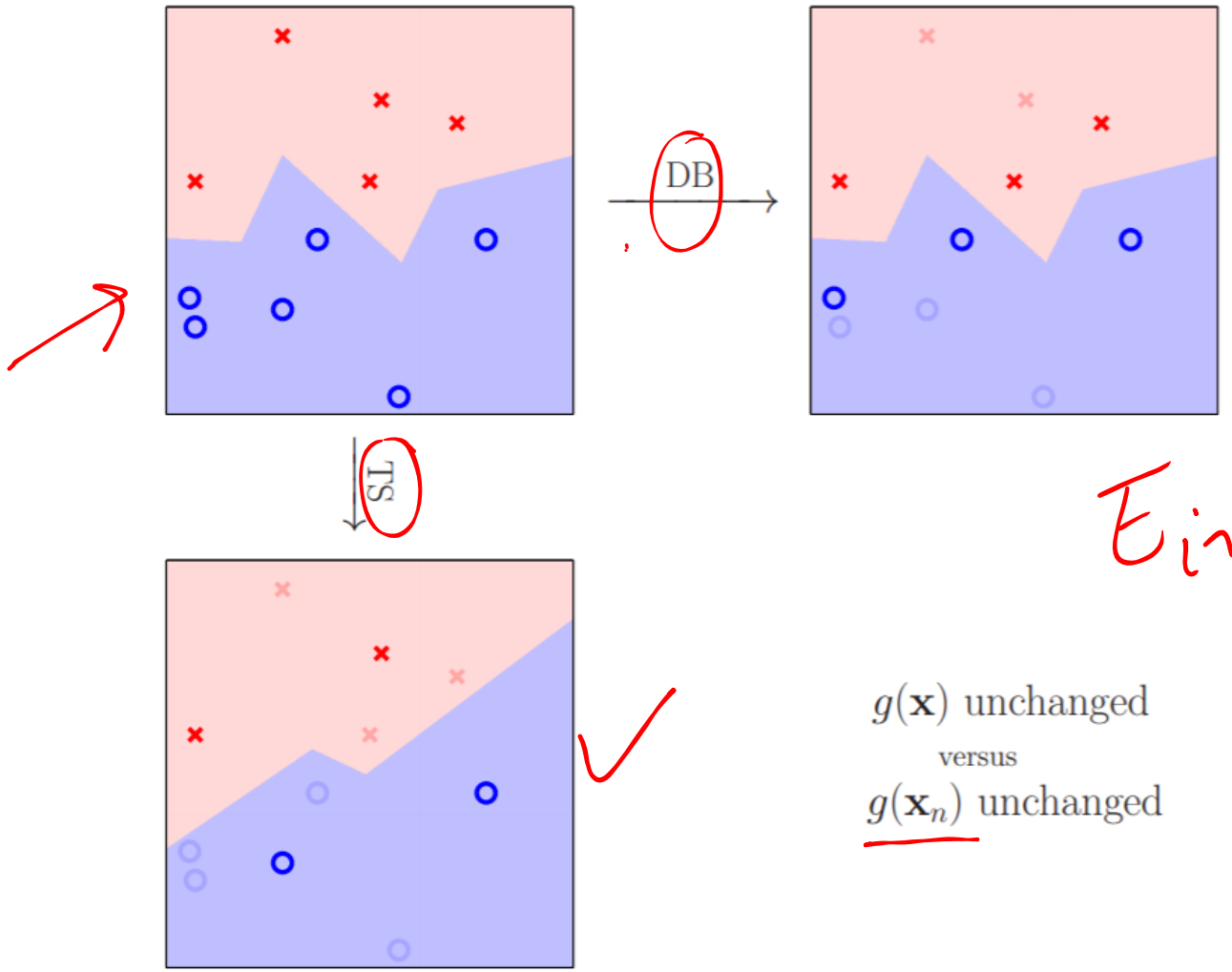
g

$g(\mathbf{x}_n)$ unchanged

E_{in} → No.

E_{out} → hypothesis has changed so E_{out} has changed!

Decision Boundary Vs. Training Set Consistent



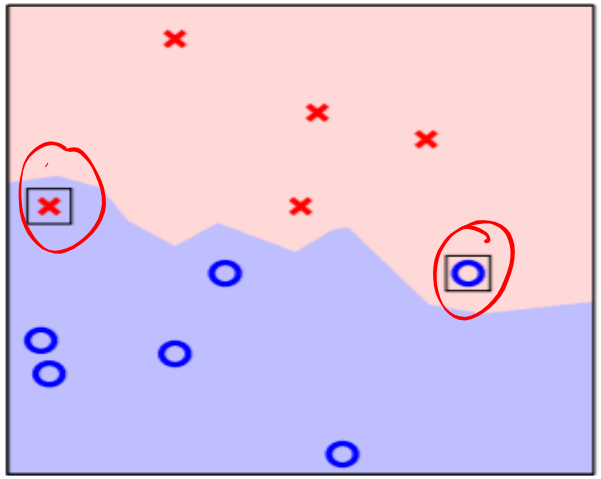
$g(\mathbf{x})$ unchanged

versus

$g(\mathbf{x}_n)$ unchanged

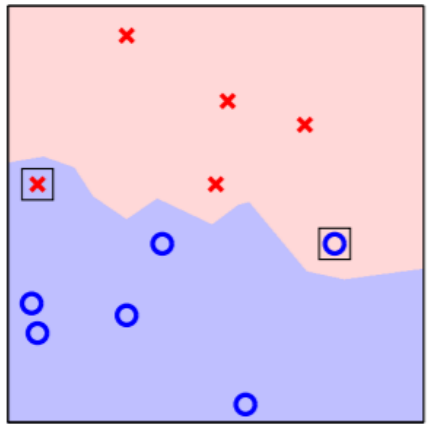
$E_{in} = 0$ if $k = 3$

Consistent Does Not Mean $g(\mathbf{x}_n) = y_n$

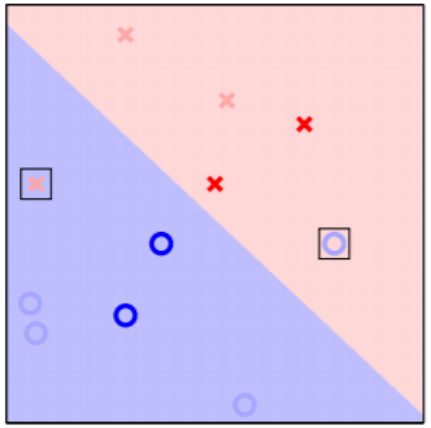


$$k = 3$$

Training Set Consistent $(k = 3)$



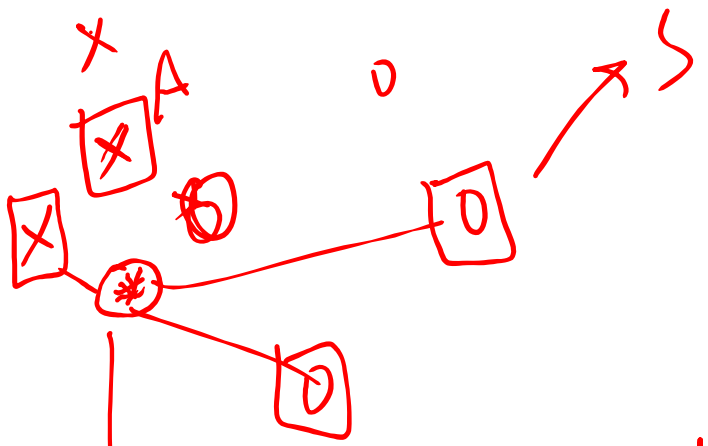
$E_{in} \neq 0$



$g(\mathbf{x}_n)$ unchanged

Condensed Nearest Neighbor (CNN)

Take data → condense it → NN, $k = 3$



$D = \{x_1, x_2 \cdots x_N\}$
$\{y_1, y_2 \cdots y_N\}$

S (working set)

$g_D(x)$

$\hat{y}_1 \; \hat{y}_2 \; \hat{y}_3 \cdots \hat{y}_N$

$\hat{y}_i = g_D(x_i)$

$0 \rightarrow +1$

$g_S(*) = +1$

$g_D(*) = -1$

x_* reveals a training set inconsistency

$S \rightarrow g_S(x)$

$g_D(x_i) = g_S(x_i)$

$S = \phi$, $k = 3$

① If S is TS consistent ⟶ we are done

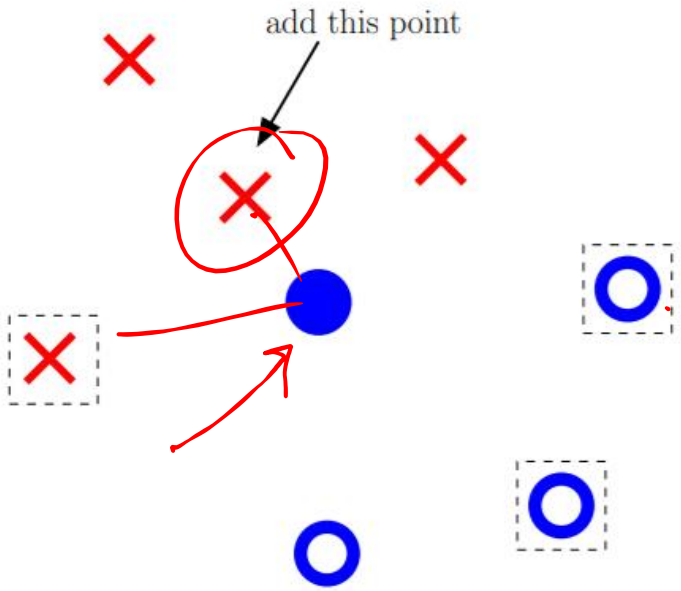
② $x_*(y_*)$ ⟶ $g_D(x_*) \neq g_S(x_*)$

↳ Add a point to your S

③ Add this point

④ Repeat these steps ⟶ TS consistent.

CNN: Condensed Nearest Neighbor



Consider the solid blue point:

i. blue w.r.t. selected points

ii. red w.r.t. $\mathcal{D}$

1. Randomly select k data points into $\mathcal{S}$.
2. Classify all data according to $\mathcal{S}$.
3. Let $\mathbf{x}_*$ be an inconsistent point and y_* its class w.r.t. $\mathcal{D}$.
4. Add the closest point to $\mathbf{x}_*$ not in $\mathcal{S}$ that has class y_*.
5. Iterate until $\mathcal{S}$ classifies all points consistently with $\mathcal{D}$.

Q1. How do we know such a point exists?

Q2. " " " that this point is in D?

↳ Yes

Q3. Algorithm stops or not?

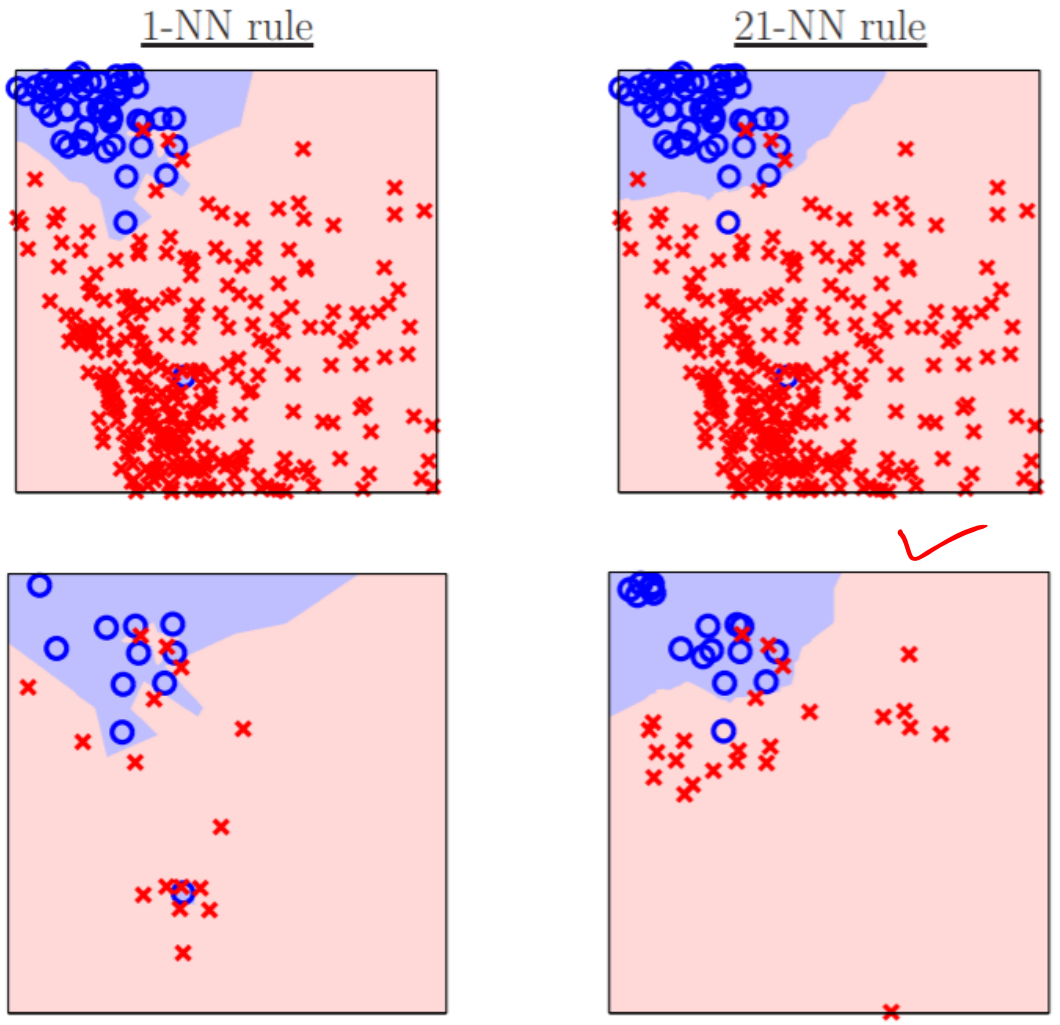
N steps.

Theorem (1) Algorithm is well defined.

2) At most N steps.

3) On termination S must be TS consistent.

Condensing the Digits Data



Original →

CNN →

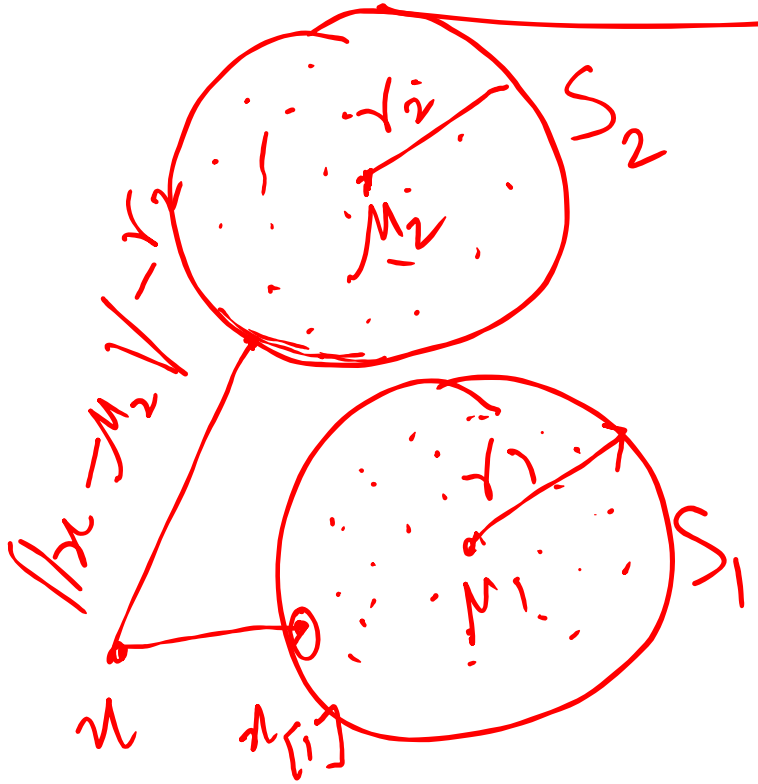
TS
↓
smallest subset?

NP-hard

k, N

Algorithmic efficiency → finding NN (Quickly)

Branch & Bound



1) Chose S_1 → Branch Step

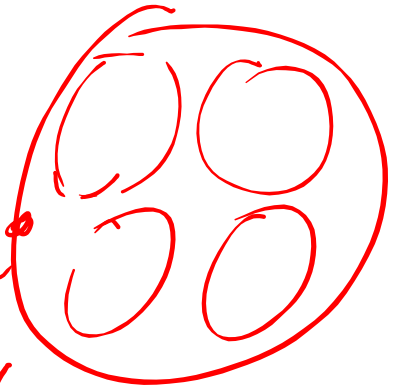
2) BOUND CRITERIA

If $\|x-\mu_2\| - r_2 \geq \|x-\hat{x}_{[1]}\|$

No need to Search S_2

Observations

i) Produces the NN (guaranteed).
ii) Recursively
iii) Worst case $O(n)$



Relax → W.h.p → logarithmic

E_{out}

Finding the Nearest Neighbor

1. S_1, S_2 are 'clusters' with centers $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$ and radii r_1, r_2.

2. **[Branch]** Search S_1 first $\rightarrow \hat{\mathbf{x}}_{[1]}$.

3. The distance from $\mathbf{x}$ to any point in S_2 is at least

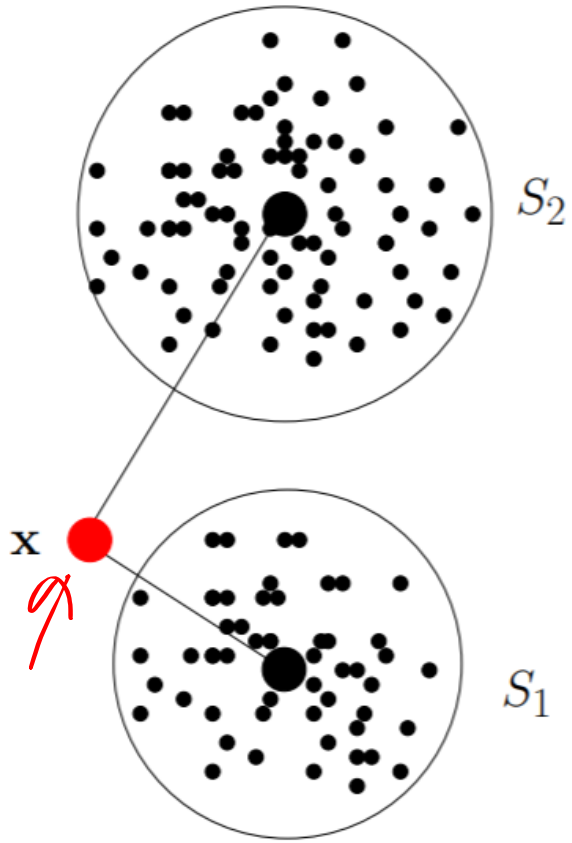
$$\| \mathbf{x} - \boldsymbol{\mu}_2 \| - r_2$$

4. **[Bound]** So we are done if

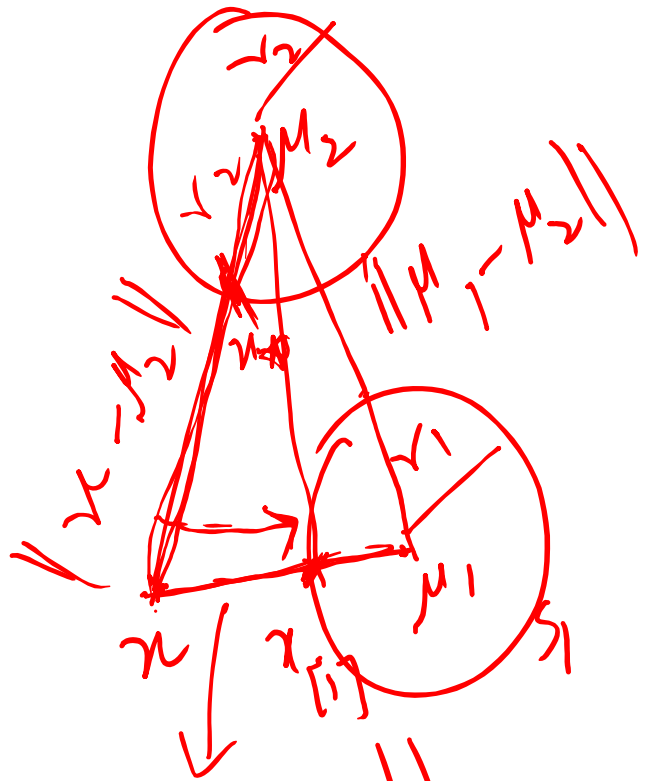
$$\| \mathbf{x} - \hat{\mathbf{x}}_{[1]} \| \leq \| \mathbf{x} - \boldsymbol{\mu}_2 \| - r_2$$

A *branch and bound* algorithm

Can be applied *recursively*



When do we gain? $r_1 + r_2 \rightarrow$ Inter, Intra



$$\|x - \hat{x}_{[1]}\| \leq \underbrace{\|x - \mu_1\| + r_1}_{\text{any point in cluster 1}}$$

$$\|x - \mu_1\| + r_1 \leq \|x - \mu_2\| - r_2$$

$\rightarrow$ Gain! $\rightarrow$ Sufficient condition.

$$\|x - \mu_1\| + r_1 \leq \|x - \mu_2\| - r_2$$

$$r_1 + r_2 \leq \|x - \mu_2\| - \underbrace{\|x - \mu_1\|}_{\text{small}}$$

$$\sim \|\mu_1 - \mu_2\|$$

$r_1 + r_2 < \|\mu_1 - \mu_2\|$

When Does the Bound Hold?

Bound condition: $\| \mathbf{x} - \hat{\mathbf{x}}_{[1]} \| \leq \| \mathbf{x} - \boldsymbol{\mu}_2 \| - r_2$.

$$\| \mathbf{x} - \hat{\mathbf{x}}_{[1]} \| \leq \| \mathbf{x} - \boldsymbol{\mu}_1 \| + r_1$$

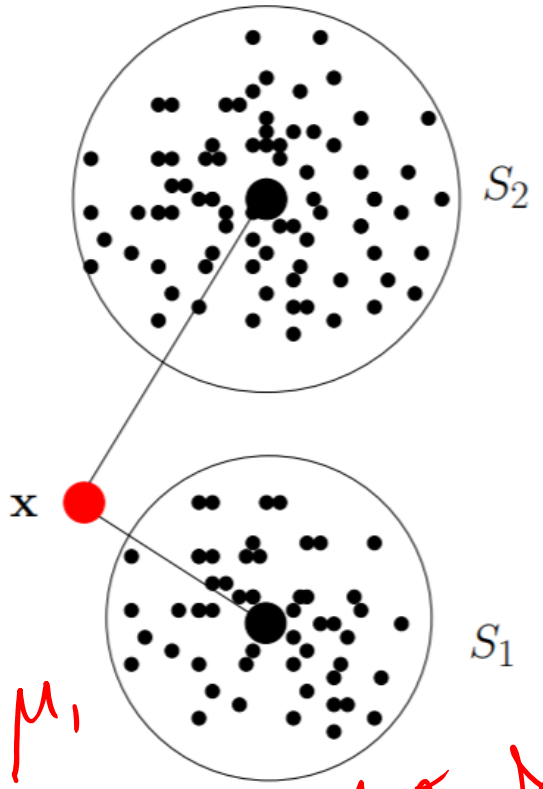
So, it suffices that

$$r_1 + r_2 \leq \| \mathbf{x} - \boldsymbol{\mu}_2 \| - \| \mathbf{x} - \boldsymbol{\mu}_1 \|.$$

$\| \mathbf{x} - \boldsymbol{\mu}_1 \| \approx 0$ means $\| \mathbf{x} - \boldsymbol{\mu}_2 \| \approx \| \boldsymbol{\mu}_2 - \boldsymbol{\mu}_2 \|$.

It suffices that

$$r_1 + r_2 \leq \| \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1 \|.$$

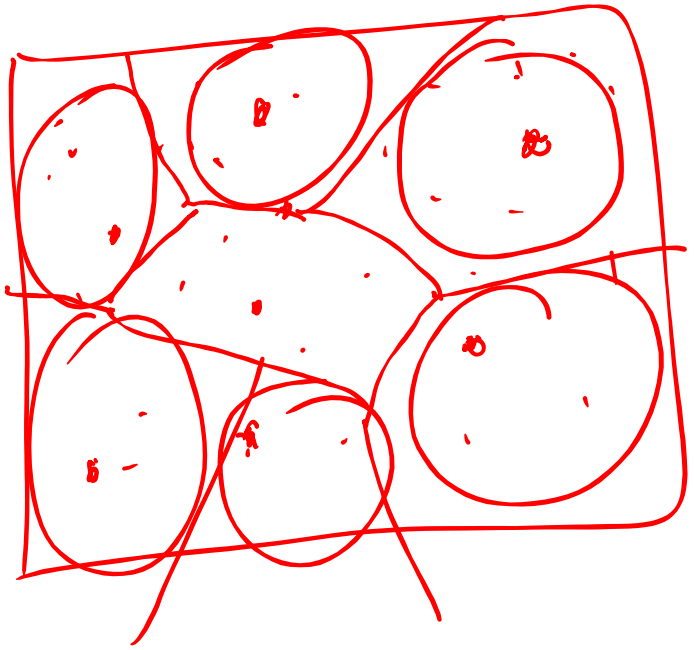


$r_1 + r_2 << \mu_2, \mu_1$

tight & well separated.

within cluster spread should be less than *between cluster spread*

Lloyd's Algo



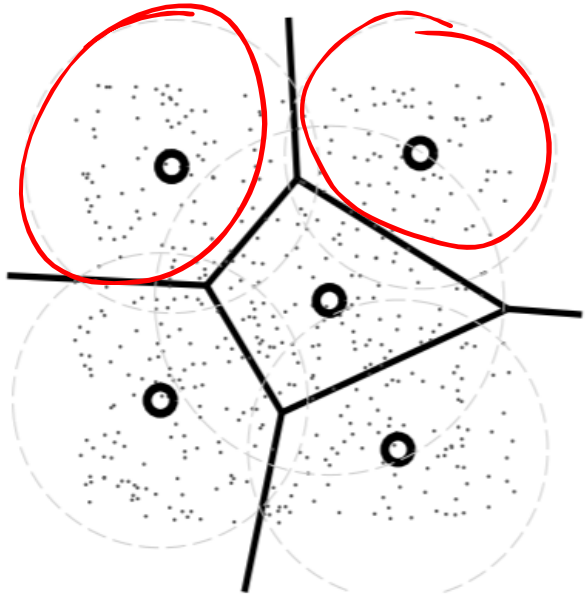
1) Pick centers randomly.
2) Pick next as far as possible, Voronoi.
3) Update the center to the actual center

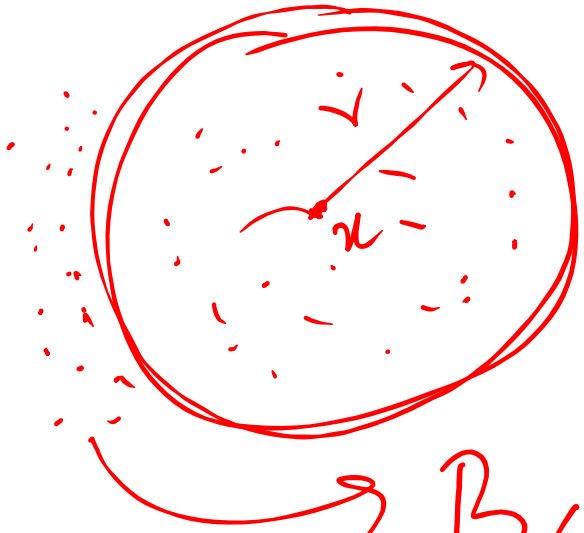
$$r_1 + r_2 << \|\mu_1 - \mu_2\|$$

Finding Clusters – Lloyd's Algorithm

1. Pick well separated centers for each cluster.
2. Compute Voronoi regions as the clusters.
3. Update the Centers.
4. Update the Voronoi regions.
5. Compute centers and radii:

$$\boldsymbol{\mu}_j = \frac{1}{|S_j|} \sum_{\mathbf{x}_n \in S_j} \mathbf{x}_n; \qquad r_j = \max_{\mathbf{x}_n \in S_j} \| \mathbf{x}_n - \boldsymbol{\mu}_j \|.$$





Bad → Weights

Radial Basis functions.

Thanks!