# CSE 417T: Introduction to Machine Learning

# Lecture 19: $k$-Nearest Neighbors

Henry Chai

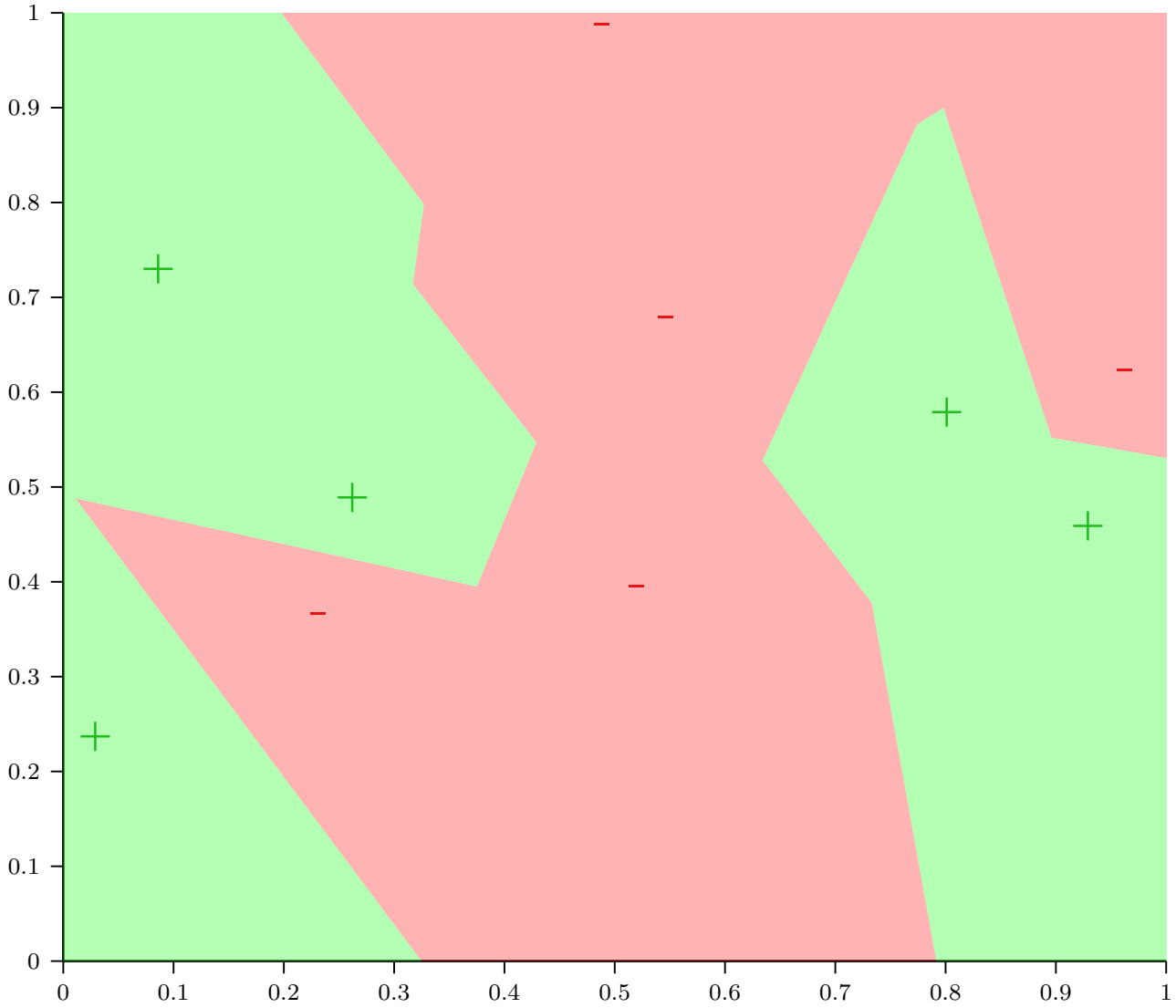
11/06/18

# Recall

- Classify a point as the label of the "most similar" training point
- Euclidean distance: $d(\vec{x}, \vec{x}') = \|\vec{x} - \vec{x}'\| = \sum_{j=1}^{D} \left(x_j - x_j'\right)^2$
- Given $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), \ldots, (\overrightarrow{x_n}, y_n)\}$ and a point $\vec{x}$, let $\vec{x}_{[i]}(\vec{x})$ be the $i^{th}$ closest point to $\vec{x}$ in $\mathcal{D}$

# The Nearest Neighbor Hypothesis

$$g(\vec{x}) = y_{[1]}(\vec{x})$$



# Generalization of Nearest Neighbor

- Claim: $E_{out}$ for the nearest neighbor hypothesis is not much worse than the best possible $E_{out}$!
- Formally: under certain conditions, with high probability, $E_{out}(g) \leq 2E_{out}(g^*)$ as $n \to \infty$
- Interpretation: half of the data's predictive power is in the nearest neighbor!

# $k$-Nearest Neighbors ($k$NN)

- Classify a point as the most common label among the labels of the $k$ nearest training points
- If we have a binary classification problem and $k$ is odd:

$$g(\vec{x}) = \text{sign}\left(\sum_{i=1}^{k} y_{[i]}(\vec{x})\right)$$

- $k$ controls the complexity of the hypothesis set $\Longrightarrow$ $k$ affects how well the learned hypothesis will generalize
  - $k = 3$
  - $k = \lfloor \sqrt{n} \rfloor$
  - Cross-validation

# $k$NN Pros and Cons

- Pros:
  - Intuitive / explainable
  - No training / retraining
  - Self-regularizes
  - Provably near-optimal in terms of $E_{out}$
- Cons:
  - Computationally expensive
    - Always needs to store all data: $O(nD)$
    - Computing $g(\vec{x})$ requires computing $d(\vec{x}, \vec{x}') \; \forall \; \vec{x}' \in \mathcal{D}$ and finding the $k$ closest points: $O(nD + n\log(k))$
  - Suffers from the "curse of dimensionality"

# Curse of Dimensionality

- The fundamental assumption of $k$NN is that "similar" points or points close to one another should have the same label
- The closer two points are, the more confident we can be that they will have the same label
- As the number of dimensions the input has grows, the less likely it is that two random points will be close
- As the number of dimensions the input has grows, it takes more points to "cover" the input space

# Curing the Curse of Dimensionality

- More data
- Fewer dimensions
- Blessing of non-uniformity: data from the real world is rarely uniformly distributed across the input space

# Computational Cost of $k$NN

- No training required!
- Memory: $O(nD)$
- Computing $g(\vec{x})$: $O(nD + n\log(k))$

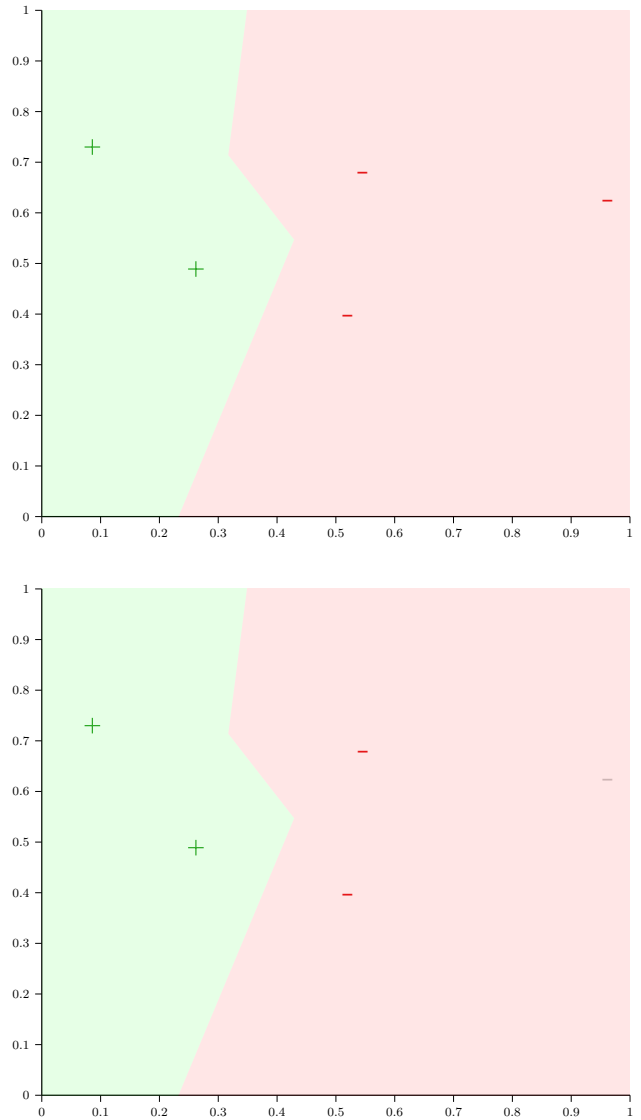
- Idea: preprocess inputs in order to speed up predictions
  - Reduce the number of inputs held in memory by eliminating redundancies
  - Organize inputs in data structures that make searching for nearest neighbors more efficient

# Data Condensing

- Reduce the number of inputs while maintaining the same decision boundary
- Let $g_{\mathcal{D}}$ be the $k$NN hypothesis when trained on $\mathcal{D}$
- $S \subseteq \mathcal{D}$ is decision-boundary consistent if:

$$g_S(\vec{x}) = g_{\mathcal{D}}(\vec{x}) \; \forall \; \vec{x} \in \mathcal{X}$$

- Decision-boundary consistent subsets are computationally expensive to find

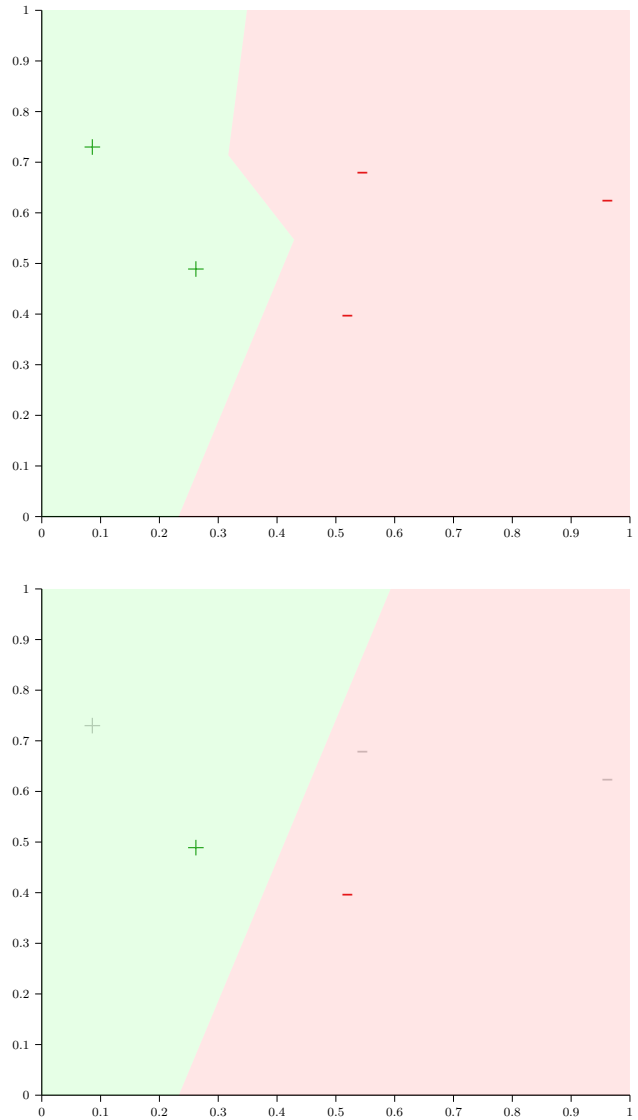


# Data Condensing

- Reduce the number of inputs while maintaining the same predictions on all inputs
- Let $g_{\mathcal{D}}$ be the $k$NN hypothesis when trained on $\mathcal{D}$
- $S \subseteq \mathcal{D}$ is training-set consistent if:

$$g_S(\overrightarrow{x_i}) = g_{\mathcal{D}}(\overrightarrow{x_i}) \; \forall \; \overrightarrow{x_i} \in \mathcal{D}$$

- Training-set consistent is a much weaker constraint than decision-boundary consistent

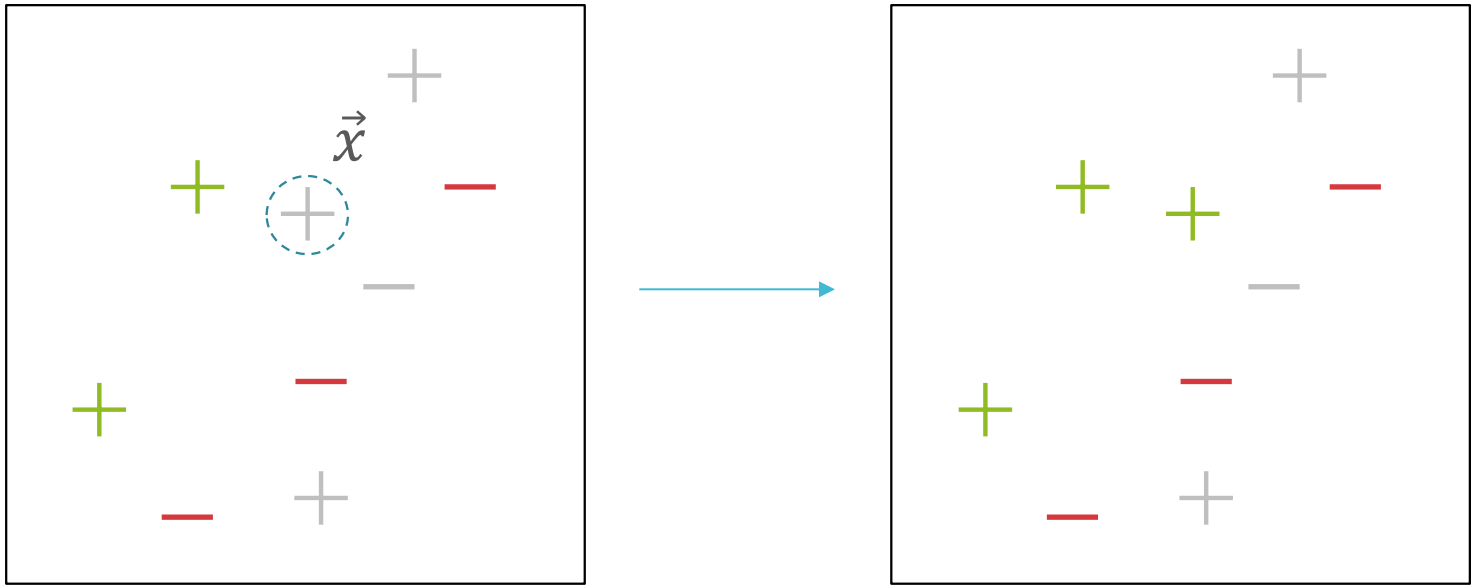


# Condensed Nearest Neighbor (CNN)

- Input: $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), \ldots, (\overrightarrow{x_n}, y_n)\}$, $k$
- Compute $g_{\mathcal{D}}(\overrightarrow{x_i})\ \forall\ \overrightarrow{x_i} \in \mathcal{D}$
- Initialize $S$ to $k$ random points in $\mathcal{D}$ and compute $g_S(\overrightarrow{x_i})\ \forall\ \overrightarrow{x_i} \in \mathcal{D}$
- While $\exists\ \overrightarrow{x_j} \in \mathcal{D}$ s.t. $g_S(\overrightarrow{x_j}) \neq g_{\mathcal{D}}(\overrightarrow{x_j})$
  - Randomly pick a point $\overrightarrow{x_j} \in \mathcal{D}$ s.t. $g_S(\overrightarrow{x_j}) \neq g_{\mathcal{D}}(\overrightarrow{x_j})$
  - Let $\vec{x}^*$ be the point closest to $\overrightarrow{x_j}$ that is not already in $S$ and has label $y^* = g_{\mathcal{D}}(\overrightarrow{x_j})$
  - Add $\vec{x}^*$ to $S$ and recompute $g_S(\overrightarrow{x_i})\ \forall\ \overrightarrow{x_i} \in \mathcal{D}$
- Output: $S$, a training-set consistent subset of $\mathcal{D}$

# CNN Example

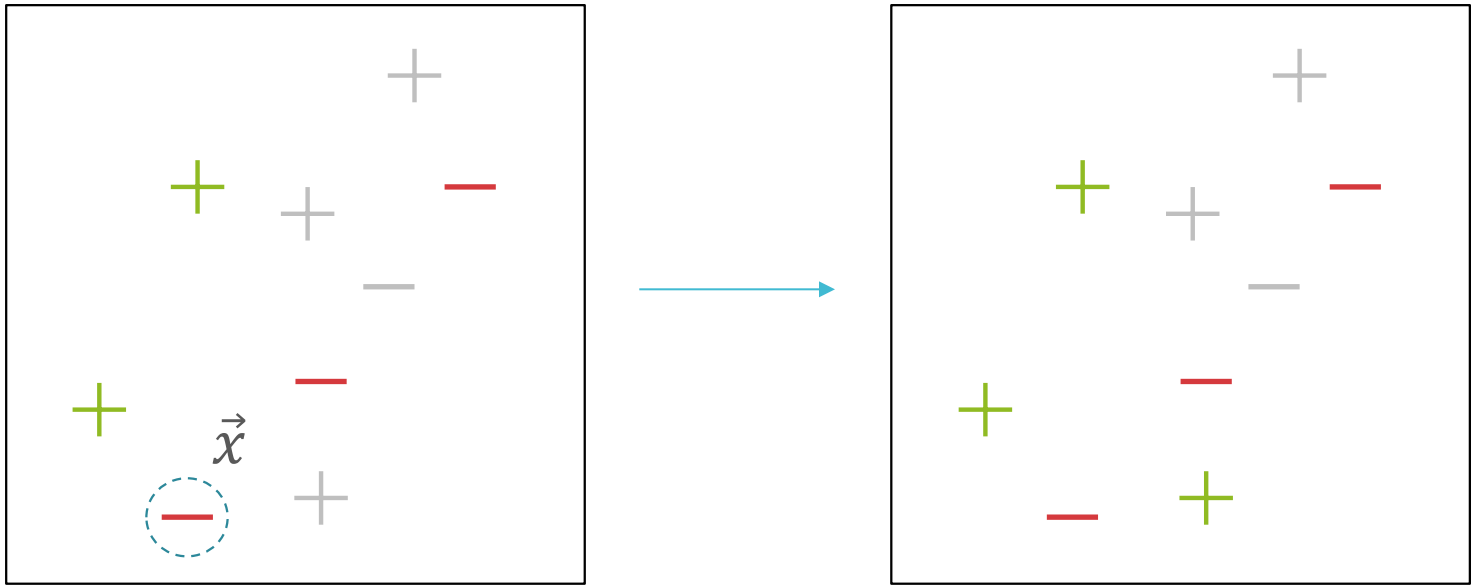
$k = 3$




$g_S(\vec{x}) = -1, g_{\mathcal{D}}(\vec{x}) = +1$

# CNN Example

$k = 3$

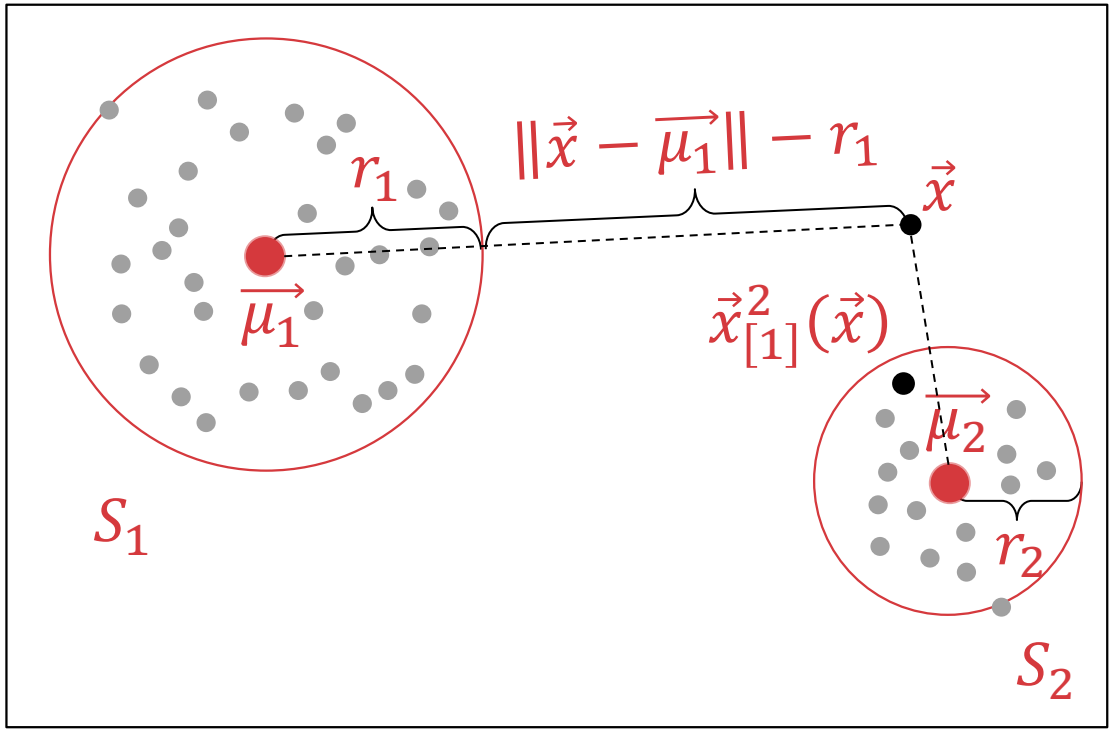



$g_S(\vec{x}) = -1, g_{\mathcal{D}}(\vec{x}) = +1$

# Organizing the Inputs

- Intuition: split the inputs into clusters, groups of points that are close to one another but far from other groups.
- If an input point is really close to one group of points and really far from the other groups …
- … then we can skip searching through the other groups and just look for nearest neighbors in the close group!
- Questions:
  - What does it mean for a point to be close to a group?
  - How can we split the input into clusters?

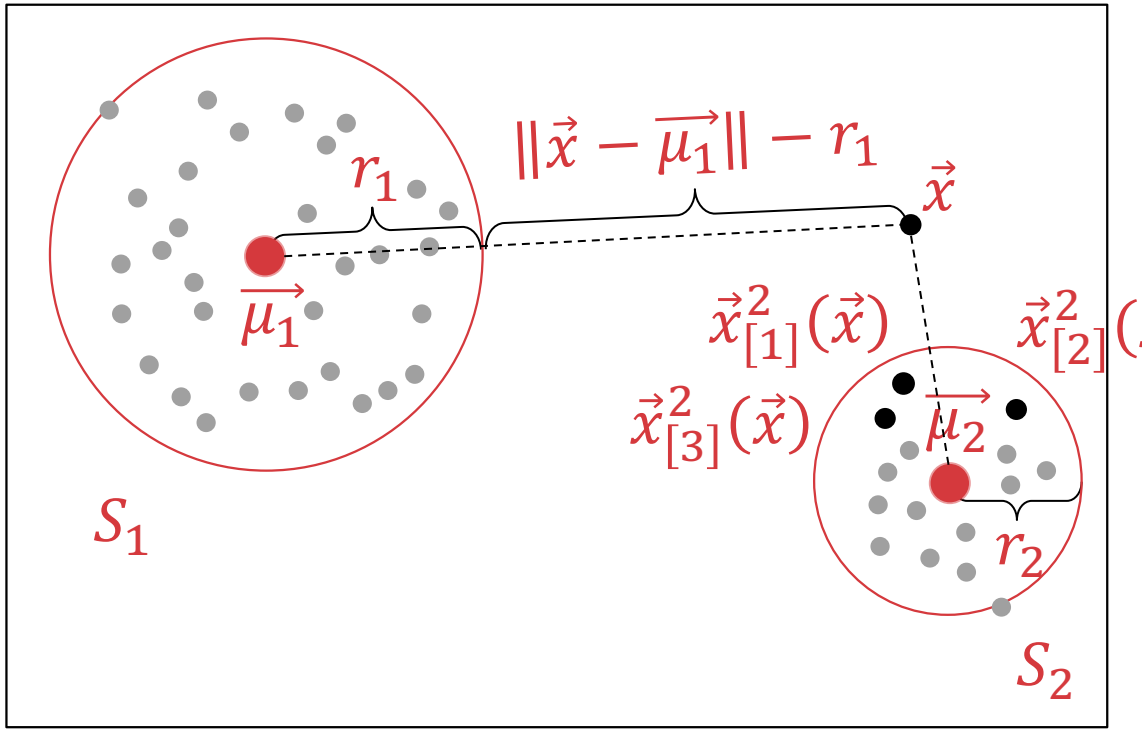
# Organizing the Inputs




Let $\vec{x}^j_{[i]}(\vec{x})$ be the $i^{\text{th}}$ closest point to $\vec{x}$ in the cluster $S_j$

- $\|\vec{x} - \overrightarrow{\mu_j}\| - r_j \leq \left\|\vec{x} - \vec{x}^j_{[1]}(\vec{x})\right\|$
- If $\exists\ S_i$ s.t. $\left\|\vec{x} - \vec{x}^i_{[1]}(\vec{x})\right\| \leq \|\vec{x} - \overrightarrow{\mu_j}\| - r_j$ …
- … then we don't need to search $S_j$ for the nearest neighbor

# Organizing the Inputs



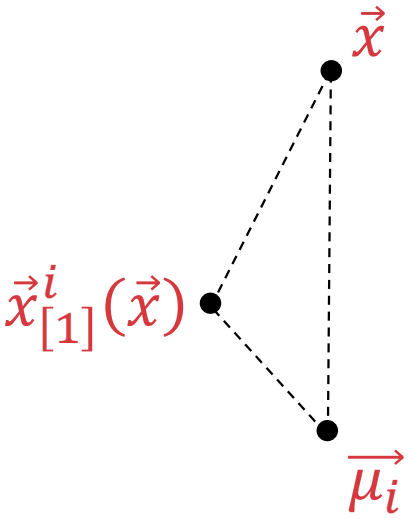

Let $\vec{x}^j_{[i]}(\vec{x})$ be the $i^{\text{th}}$ closest point to $\vec{x}$ in the cluster $S_j$

- $\|\vec{x} - \overrightarrow{\mu_j}\| - r_j \leq \left\|\vec{x} - \vec{x}^j_{[1]}(\vec{x})\right\| \leq \cdots \leq \left\|\vec{x} - \vec{x}^j_{[k]}(\vec{x})\right\|$
- If $\exists\ S_i$ s.t. $\left\|\vec{x} - \vec{x}^i_{[k]}(\vec{x})\right\| \leq \|\vec{x} - \overrightarrow{\mu_j}\| - r_j$ …
- … then we don't need to search $S_j$ for the $k$ nearest neighbors

# Clustering Inputs

- We want $\left\|\vec{x} - \vec{x}^i_{[1]}(\vec{x})\right\| \leq \left\|\vec{x} - \overrightarrow{\mu_j}\right\| - r_j$
- $\left\|\vec{x} - \vec{x}^i_{[1]}(\vec{x})\right\| \leq \|\vec{x} - \overrightarrow{\mu_i}\| + \left\|\overrightarrow{\mu_i} - \vec{x}^i_{[1]}(\vec{x})\right\|$

  $\leq \|\vec{x} - \overrightarrow{\mu_i}\| + r_i$
- We want $\|\vec{x} - \overrightarrow{\mu_i}\| + r_i \leq \left\|\vec{x} - \overrightarrow{\mu_j}\right\| - r_j$




- Suppose $\vec{x} \approx \overrightarrow{\mu_i}$
  - $\|\vec{x} - \overrightarrow{\mu_i}\| \approx 0$ and $\left\|\vec{x} - \overrightarrow{\mu_j}\right\| \approx \left\|\overrightarrow{\mu_i} - \overrightarrow{\mu_j}\right\|$
- We want $r_i \leq \left\|\overrightarrow{\mu_i} - \overrightarrow{\mu_j}\right\| - r_j \Longrightarrow r_i + r_j \leq \left\|\overrightarrow{\mu_i} - \overrightarrow{\mu_j}\right\|$
- We want cluster centers to be far apart and cluster radii to be small

# Clustering Inputs: Example

- Input: $\mathcal{D} = \{\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}\}$, $C$
- Initialize $\mathrm{M}$ to a random point $\vec{x} \in \mathcal{D}$ and set $S_1 = \vec{x}$
- For $c = 2 \ldots C$
  - Find the point $\vec{x}^* \in \mathcal{D} \backslash \mathrm{M}$ that is farthest from $\mathrm{M}$, add $\vec{x}^*$ to $\mathrm{M}$ and set $S_c = \vec{x}^*$:

$$\vec{x}^* = \underset{\vec{x}^* \in \mathcal{D} \backslash \mathrm{M}}{\operatorname{argmax}} \left( \min_{\vec{x} \in \mathrm{M}} \|\vec{x}^* - \vec{x}\| \right)$$

- For $i = 1 \ldots n$
  - Find the cluster center $\overrightarrow{x_c} \in \mathrm{M}$ closest to $\overrightarrow{x_i}$ and assign $\overrightarrow{x_i}$ to $S_c$:

$$\overrightarrow{x_c} = \underset{\vec{x} \in \mathrm{M}}{\operatorname{argmin}} (\|\overrightarrow{x_i} - \vec{x}\|)$$

- For $c = 1 \ldots C$
  - Update $S_c$'s cluster center and compute the radius:

$$\overrightarrow{\mu_c} = \frac{1}{|S_c|} \sum_{\vec{x} \in S_c} \vec{x} \text{ and } r_c = \max_{\vec{x} \in S_c} \|\overrightarrow{\mu_c} - \vec{x}\|$$

- Output: cluster centers $\mu = \{\overrightarrow{\mu_1}, \overrightarrow{\mu_2}, \ldots, \overrightarrow{\mu_C}\}$, radii $\vec{r} = \{r_1, r_2, \ldots, r_C\}$ and clusters $S = \{S_1, S_2, \ldots, S_C\}$

# Making Predictions: Example

- Input: $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), \ldots, (\overrightarrow{x_n}, y_n)\}$, $C$, $k$, $\vec{x}$
- Cluster the input into $C$ clusters $\left(O(CnD)\right)$
- Find $S_c$, the cluster whose center is closest to $\vec{x}$
- Find $\vec{x}^c_{[k]}(\vec{x})$, the $k^{\text{th}}$ closest point to $\vec{x}$ in $S_c$
- Find $\left\{\vec{x}_{[1]}(\vec{x}), \ldots, \vec{x}_{[k]}(\vec{x})\right\}$ the $k$ closest points to $\vec{x}$, ignoring all clusters $S_{c'}$ s.t. $\left\|\vec{x} - \vec{x}^c_{[k]}(\vec{x})\right\| \leq \|\vec{x} - \overrightarrow{\mu_{c'}}\| - r_{c'}$ (hopefully less than $O(nD + n\log(k))$)
- Output: $\left\{\vec{x}_{[1]}(\vec{x}), \ldots, \vec{x}_{[k]}(\vec{x})\right\}$

# Parametric vs. Non-parametric

- Parametric learning models
  - Hypotheses have a parametrized form
    - Example: linear regression $g(\vec{x}) = \vec{w}^T\vec{x}$ has $D + 1$ parameters, $\{w_0, w_1, \dots, w_D\}$
  - Parameters learned from training data; can discard the training data afterwards
  - Cannot exactly model every target function
- Non-parametric learning models
  - Hypotheses cannot be expressed using a finite number of parameters
    - Example: $k$NN, decision trees
  - Training data generally needs to be stored in order to make predictions
  - Can recover any target function given enough data