Recall

- Classify a point as the label of the “most similar” training point

- Euclidean distance: \( d(\vec{x}, \vec{x}') = ||\vec{x} - \vec{x}'|| = \sum_{j=1}^{D} (x_j - x'_j)^2 \)

- Given \( \mathcal{D} = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_n, y_n)\} \) 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

\[ g(\vec{x}) = y_{[1]}(\vec{x}) \]
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 $\Rightarrow k$ affects how well the learned hypothesis will generalize
  - $k = 3$
  - $k = \lfloor \sqrt{n} \rfloor$
  - Cross-validation
\textbf{kNN Pros and Cons}

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

- The fundamental assumption of kNN 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 $g(\tilde{x}): O(nD + n \log(k))$

- 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 $g_D$ be the $k$NN hypothesis when trained on $\mathcal{D}$
- $S \subseteq \mathcal{D}$ is decision-boundary consistent if:
  \[ g_S(\tilde{x}) = g_D(\tilde{x}) \ \forall \ \tilde{x} \in \mathcal{X} \]
- 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 $g_D$ be the $k$NN hypothesis when trained on $D$
• $S \subseteq D$ is training-set consistent if:
  $$g_S(\bar{x}_i) = g_D(\bar{x}_i) \ \forall \ \bar{x}_i \in D$$
• Training-set consistent is a much weaker constraint than decision-boundary consistent
Condensed Nearest Neighbor (CNN)

• Input: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}, k$

• Compute $g_D(x_i) \forall x_i \in \mathcal{D}$

• Initialize $S$ to $k$ random points in $\mathcal{D}$ and compute $g_S(x_i) \forall x_i \in \mathcal{D}$

• While $\exists x_j \in \mathcal{D}$ s.t. $g_S(x_j) \neq g_D(x_j)$
  • Randomly pick a point $x_j \in \mathcal{D}$ s.t. $g_S(x_j) \neq g_D(x_j)$
  • Let $\hat{x}^*$ be the point closest to $x_j$ that is not already in $S$ and has label $y^* = g_D(x_j)$
  • Add $\hat{x}^*$ to $S$ and recompute $g_S(x_i) \forall x_i \in \mathcal{D}$

• Output: $S$, a training-set consistent subset of $\mathcal{D}$
CNN Example

$k = 3$

$g_S(\tilde{x}) = -1$, $g_D(\tilde{x}) = +1$
CNN Example

\[ k = 3 \]

\[ g_S(\vec{x}) = -1, \quad g_D(\vec{x}) = +1 \]
• 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 $\mathbf{x}^{j}_{[i]}(\mathbf{x})$ be the $i^{th}$ closest point to $\mathbf{x}$ in the cluster $S_j$

- $\|\mathbf{x} - \mu_j\| - r_j \leq \|\mathbf{x} - \mathbf{x}^{j}_{[1]}(\mathbf{x})\|$
- If $\exists S_i$ s.t. $\|\mathbf{x} - \mathbf{x}^{i}_{[1]}(\mathbf{x})\| \leq \|\mathbf{x} - \mu_j\| - r_j$ ...
- ... 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 $i^{th}$ closest point to $\vec{x}$ in the cluster $S_j$

- $||\vec{x} - \vec{\mu}_j|| - r_j \leq ||\vec{x} - \vec{x}_j^1(\vec{x})|| \leq \ldots \leq ||\vec{x} - \vec{x}_j^k(\vec{x})||$

- If $\exists S_i$ s.t. $||\vec{x} - \vec{x}_j^i(\vec{x})|| \leq ||\vec{x} - \vec{\mu}_j|| - r_j ...$

- ... then we don’t need to search $S_j$ for the $k$ nearest neighbors
- We want \( \|\hat{x} - \hat{x}_{[1]}(\bar{x})\| \leq \|\hat{x} - \mu_j\| - r_j \)

- \( \|\hat{x} - \hat{x}_{[1]}(\bar{x})\| \leq \|\hat{x} - \mu_i\| + \|\mu_i - \hat{x}_{[1]}(\bar{x})\| \)
  \[ \leq \|\hat{x} - \mu_i\| + r_i \]

- We want \( \|\hat{x} - \mu_i\| + r_i \leq \|\hat{x} - \mu_j\| - r_j \)

- Suppose \( \hat{x} \approx \mu_i \)
  - \( \|\hat{x} - \mu_i\| \approx 0 \) and \( \|\hat{x} - \mu_j\| \approx \|\mu_i - \mu_j\| \)

- We want \( r_i \leq \|\mu_i - \mu_j\| - r_j \Rightarrow r_i + r_j \leq \|\mu_i - \mu_j\| \)

- We want cluster centers to be far apart and cluster radii to be small
Clustering Inputs: Example

- Input: $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}, C$
- Initialize $M$ to a random point $\mathbf{x} \in \mathcal{D}$ and set $S_1 = \mathbf{x}$
- For $c = 2 \ldots C$
  - Find the point $\mathbf{x}^* \in \mathcal{D} \setminus M$ that is farthest from $M$, add $\mathbf{x}^*$ to $M$ and set $S_c = \mathbf{x}^*$:
    $$\mathbf{x}^* = \arg\max_{\mathbf{x}^* \in \mathcal{D} \setminus M} \left( \min_{\mathbf{x} \in M} ||\mathbf{x}^* - \mathbf{x}|| \right)$$
- For $i = 1 \ldots n$
  - Find the cluster center $\mathbf{x}_c \in M$ closest to $\mathbf{x}_i$ and assign $\mathbf{x}_i$ to $S_c$:
    $$\mathbf{x}_c = \arg\min_{\mathbf{x} \in M} (||\mathbf{x}_i - \mathbf{x}||)$$
- For $c = 1 \ldots C$
  - Update $S_c$’s cluster center and compute the radius:
    $$\mu_c = \frac{1}{|S_c|} \sum_{\mathbf{x} \in S_c} \mathbf{x} \text{ and } r_c = \max_{\mathbf{x} \in S_c} ||\mu_c - \mathbf{x}||$$
- Output: cluster centers $\mu = \{\mu_1, \mu_2, \ldots, \mu_C\}$, radii $r = \{r_1, r_2, \ldots, r_C\}$ and clusters $S = \{S_1, S_2, \ldots, S_C\}$
Making Predictions: Example

• Input: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}, C, k, \tilde{x}$

• Cluster the input into $C$ clusters ($O(CnD)$)

• Find $S_c$, the cluster whose center is closest to $\tilde{x}$

• Find $\tilde{x}^c_{[k]}(\tilde{x})$, the $k^{th}$ closest point to $\tilde{x}$ in $S_c$

• Find $\{\tilde{x}_{[1]}(\tilde{x}), \ldots, \tilde{x}_{[k]}(\tilde{x})\}$ the $k$ closest points to $\tilde{x}$, ignoring all clusters $S_{c'}$ s.t. $\|\tilde{x} - \tilde{x}^c_{[k]}(\tilde{x})\| \leq \|\tilde{x} - \mu_{c'}\| - r_{c'}$ (hopefully less than $O(nD + n \log(k))$)

• Output: $\{\tilde{x}_{[1]}(\tilde{x}), \ldots, \tilde{x}_{[k]}(\tilde{x})\}$
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, \( \{w_0, w_1, \ldots, w_D\} \)
  - 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: kNN, decision trees
  - Training data generally needs to be stored in order to make predictions
  - Can recover any target function given enough data