Learning From Data
Lecture 17
Memory and Efficiency in Nearest Neighbor

Memory
Efficiency

M. Magdon-Ismail
CSCI 4100/6100
RECAP: Similarity and Nearest Neighbor

**Similarity**

\[ d(x, x') = \| x - x' \| \]

1. Simple.
2. No training.
3. Near optimal \( E_{out} \):
   \[ k \to \infty, \frac{k}{N} \to 0 \implies E_{out} \to E^*_{out}. \]
4. Good ways to choose \( k \):
   \[ k = 3; \quad k = \left\lceil \sqrt{N} \right\rceil; \text{ validation/cross validation}. \]
5. Easy to justify classification to customer.
6. Can easily do multi-class.
7. Can easily adapt to regression or logistic regression:
   \[ g(x) = \frac{1}{k} \sum_{i=1}^{k} y[i](x) \]
   \[ g(x) = \frac{1}{k} \sum_{i=1}^{k} \left[ y[i](x) = +1 \right] \]
8. **Computationally demanding.**
Computational Demands of Nearest Neighbor

**Memory.**

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

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

**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

$\approx 3$ms (compute $g(x)$)

$\approx 1$hr (compute CV error)

$> 1$month (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.
Throw Away Irrelevant Data

\[ k = 1 \]
Decision Boundary Consistent

\[ g(x) \text{ unchanged} \]
Training Set Consistent

\[ g(x_n) \text{ unchanged} \]
Decision Boundary Vs. Training Set Consistent

\[ g(\mathbf{x}) \text{ unchanged} \]
versus
\[ g(\mathbf{x}_n) \text{ unchanged} \]
Consistent Does Not Mean $g(x_n) = y_n$

$k = 3$
Training Set Consistent \( (k = 3) \)

\[ g(x_n) \text{ unchanged} \]
CNN: Condensed Nearest Neighbor \((k = 3)\)

Consider the solid blue point:
1. blue w.r.t. selected points
2. red w.r.t. \(\mathcal{D}\)
CNN: Condensed Nearest Neighbor

Consider the solid blue point:

i. blue w.r.t. selected points

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

Add a red point:

i. not already selected

ii. closest to the inconsistent point
CNN: Condensed Nearest Neighbor

Consider the solid blue point:
   i. blue w.r.t. selected points
   ii. red w.r.t. $\mathcal{D}$

Add a red point:
   i. not already selected
   ii. closest to the inconsistent point

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 $\mathcal{D}$.

Minimum consistent set (MCS)? $\leftarrow$ NP-hard
Nearest Neighbor on Digits Data

1-NN rule

21-NN rule
Condensing the Digits Data

1-NN rule

21-NN rule
Finding the Nearest Neighbor

1. $S_1, S_2$ are ‘clusters’ with centers $\mu_1, \mu_2$ and radii $r_1, r_2$.

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

3. The distance from $x$ to any point in $S_2$ is at least

   $$\|x - \mu_2\| - r_2$$

4. [Bound] So we are done if

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

A branch and bound algorithm
Can be applied recursively
When Does the Bound Hold?

Bound condition: \( \| x - \hat{x}[1] \| \leq \| x - \mu_2 \| - r_2. \)

\[
\| x - \hat{x}[1] \| \leq \| x - \mu_1 \| + r_1
\]

So, it suffices that

\[
r_1 + r_2 \leq \| x - \mu_2 \| - \| x - \mu_1 \|.
\]

\( \| x - \mu_1 \| \approx 0 \) means \( \| x - \mu_2 \| \approx \| \mu_2 - \mu_2 \|. \)

It suffices that

\[
r_1 + r_2 \leq \| \mu_2 - \mu_1 \|.
\]

*within cluster spread* should be less than *between cluster spread*
Finding Clusters – Lloyd’s Algorithm

1. Pick well separated centers for each cluster.
Finding Clusters – Lloyd’s Algorithm

1. Pick well separated centers for each cluster.

\[ \mu_j = \frac{1}{|S_j|} \sum_{x_n \in S_j} x_n; \]
\[ r_j = \max_{x_n \in S_j} ||x_n - \mu_j||. \]
1. Pick well separated centers for each cluster.

\[ \mu_j = \frac{1}{|S_j|} \sum_{x_n \in S_j} x_n; \]
\[ r_j = \max_{x_n \in S_j} ||x_n - \mu_j||. \]
Finding Clusters – Lloyd’s Algorithm

1. Pick well separated centers for each cluster.
Finding Clusters – Lloyd’s Algorithm

1. Pick well separated centers for each cluster.

2. Compute Voronoi regions as the clusters.
Finding Clusters – Lloyd’s Algorithm

1. Pick well separated centers for each cluster.

2. Compute Voronoi regions as the clusters.

3. Update the Centers.

\[ \mu_j = \frac{\sum_{x_n \in S_j} x_n}{|S_j|}; \quad r_j = \max_{x_n \in S_j} ||x_n - \mu_j||. \]
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:

\[ \mu_j = \frac{1}{|S_j|} \sum_{x_n \in S_j} x_n; \quad r_j = \max_{x_n \in S_j} \|x_n - \mu_j\|. \]
Radial Basis Functions (RBF)

$k$-Nearest Neighbor: Only considers $k$-nearest neighbors.
   each neighbor has equal weight

What about using all data to compute $g(x)$?

**RBF**: Use all data.
   data further away from $x$ have less weight.