Instance-Based Learning

Key idea: Just store all training examples \( \langle x_i, f(x_i) \rangle \)

Nearest neighbor:
- Given query instance \( x_q \), first locate nearest training example \( x_n \), then estimate \( f(x_q) \approx f(x_n) \)

\( k \)-Nearest neighbor:
- Given \( x_q \), take vote among its \( k \) nearest neighbors (if discrete-valued target function)
- Take mean of \( f \) values of \( k \) nearest neighbors (if real-valued)

\[
\hat{f}(x_q) \leftarrow \frac{1}{k} \sum_{i=1}^{k} f(x_i)
\]

Distance Measures

- Numeric features:
  - Euclidean, Manhattan, \( L^n \)-norm:
    \[
    L^n(x_1, x_2) = \sqrt{\sum_{i=1}^{\# \text{dim}} |x_{1,i} - x_{2,i}|^n}
    \]
  - Normalized by: range, std. deviation

- Symbolic features:
  - Hamming/overlap
  - Value difference measure (VDM):
    \[
    \delta_{\text{val}_i, \text{val}_j} = \sum_{h=1}^{\# \text{bases}} |P(c_h | \text{val}_i) - P(c_h | \text{val}_j)|^n
    \]

- In general: Arbitrary, encode knowledge

Advantages and Disadvantages

Advantages:
- Training is very fast
- Learn complex target functions easily
- Don't lose information

Disadvantages:
- Slow at query time
- Lots of storage
- Easily fooled by irrelevant attributes
Behavior in the Limit

$\epsilon^*(x)$: Error of optimal prediction
$\epsilon_{NN}(x)$: Error of nearest neighbor

**Theorem:** $\lim_{n \to \infty} \epsilon_{NN} \leq 2\epsilon^*$

**Proof sketch (2-class case):**

$$\epsilon_{NN} = p_+ p_{NN_{-}} + p_- p_{NN_{+}}$$

$$= p_+ (1 - p_{NN_{+}}) + (1 - p_+) p_{NN_{+}}$$

$$\lim_{n \to \infty} p_{NN_{-}} = p_+^+,$$

$$\lim_{n \to \infty} p_{NN_{+}} = p_+^-$$

$$\lim_{n \to \infty} \epsilon_{NN} = p_+^+ (1 - p_+^-) + (1 - p_+^+) p_+^- = 2\epsilon^*(1 - \epsilon^*) \leq 2\epsilon^*$$

$\lim_{n \to \infty}$ (Nearest neighbor) = Gibbs classifier

**Theorem:** $\lim_{n \to \infty, k \to \infty, h/n \to 0} k_{NN} = \epsilon^*$

Distance-Weighted $k$-NN

Might want to weight nearer neighbors more heavily . . .

$$\hat{f}(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

and $d(x_q, x_i)$ is distance between $x_q$ and $x_i$

Notice that now it makes sense to use all training examples instead of just $k$

Curse of Dimensionality

- Imagine instances described by 20 attributes, but only 2 are relevant to target function
- **Curse of dimensionality:**
  - Nearest neighbor is easily misled when hi-dim $X$
  - Easy problems in low-dim are hard in hi-dim
  - Low-dim intuitions don’t apply in hi-dim
- **Examples:**
  - Normal distribution
  - Uniform distribution on hypercube
  - Points on hypergrid
  - Approximation of sphere by cube
  - Volume of hypersphere

Feature Selection

- **Filter approach:**
  - Pre-select features individually
    - E.g., by info gain
- **Wrapper approach:**
  - Run learner with different combinations of features
    - Forward selection
    - Backward elimination
    - Etc.

---

**Forward Selection (FS)**

FS: Set of features used to describe examples

Let $SS = \emptyset$

Let $BestEval = 0$

Repeat

Let $BestF = None$

For each feature $F$ in $FS$ and not in $SS$

Let $SS' = SS \cup \{F\}$

If $Eval(SS') > BestEval$

Then Let $BestF = F$

Let $BestEval = Eval(SS')$

If $BestF \neq None$

Then Let $SS = SS \cup \{BestF\}$

Until $BestF = None$ or $SS = FS$

Return $SS$

**Backward Elimination (FS)**

FS: Set of features used to describe examples

Let $SS = FS$

Let $BestEval = Eval(SS)$

Repeat

Let $WorstF = None$

For each feature $F$ in $SS$

Let $SS' = SS - \{F\}$

If $Eval(SS') \geq BestEval$

Then Let $WorstF = F$

Let $BestEval = Eval(SS')$

If $WorstF \neq None$

Then Let $SS = SS - \{WorstF\}$

Until $WorstF = None$ or $SS = \emptyset$

Return $SS$
Feature Weighting

- Stretch jth axis by weight \( z_j \), where \( z_1, \ldots, z_n \) chosen to minimize prediction error
- Use gradient descent to find weights \( z_1, \ldots, z_n \)
- Setting \( z_j \) to zero eliminates this dimension altogether

Reducing Computational Cost

- Efficient retrieval: k-D trees
  (only work in low dimensions)
- Efficient similarity comparison:
  - Use cheap approx. to weed out most instances
  - Use expensive measure on remainder
- Form prototypes
- Edited k-NN:
  Remove instances that don’t affect frontier

Edited k-Nearest Neighbor

\[
\text{EDITED}_k\text{-NN}(S) \\
\text{S}: \text{Set of instances} \\
\text{For each instance } \mathbf{x} \text{ in } S \\
\quad \text{If } \mathbf{x} \text{ is correctly classified by } S - \{\mathbf{x}\} \\
\quad \quad \text{Remove } \mathbf{x} \text{ from } S \\
\text{Return } S \\
\]

\[
\text{EDITED}_k\text{-NN}(S) \\
\text{S}: \text{Set of instances} \\
T = \emptyset \\
\text{For each instance } \mathbf{x} \text{ in } S \\
\quad \text{If } \mathbf{x} \text{ is not correctly classified by } T \\
\quad \quad \text{Add } \mathbf{x} \text{ to } T \\
\text{Return } T \\
\]

Locally Weighted Regression

k-NN forms local approx. to \( f \) for each query point \( x_q \)

Why not form an explicit approximation \( \hat{f}(x) \) for region surrounding \( x_q \)?

- Fit linear function to k nearest neighbors
- Fit quadratic, …
- Produces “piecewise approximation” to \( f \)

Several choices of error to minimize:

- Squared error over k nearest neighbors
  \[
  E_1(x_q) = \sum_{x \in kNN(x_q)} (f(x) - \hat{f}(x))^2
  \]
- Distance-weighted squared error over all neighbors
  \[
  E_2(x_q) = \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))
  \]
- …
Radial Basis Function Networks

- Global approximation to target function, in terms of linear combination of local approximations
- Used, e.g., for image classification
- A different kind of neural network
- Closely related to distance-weighted regression, but “eager” instead of “lazy”

Training Radial Basis Function Networks

Q1: What \( x_u \) to use for each kernel function \( K_u(d(x_u, x)) \)
- Scatter uniformly throughout instance space
- Use training instances (reflects distribution)
- Cluster instances and use centroids

Q2: How to train weights (assume here Gaussian \( K_u \))
- First choose variance (and perhaps mean) for each \( K_u \)
  - E.g., use EM
- Then hold \( K_u \) fixed, and train linear output layer
  - Efficient methods to fit linear function
- Or use backpropagation

Case-Based Reasoning

Can apply instance-based learning even when \( X \neq \mathbb{R}^n \)
- Need different “distance” measure

Case-based reasoning is instance-based learning applied to instances with symbolic logic descriptions

Widely used for answering help-desk queries

((user-complaint error53-on-shutdown)
  (cpu-model PentiumIII)
  (operating-system Windows2000)
  (network-connection Ethernet)
  (memory 128MB)
  (installed-applications Office PhotoShop VirusScan)
  (disk 10GB)
  (likely-cause ???))

Case-Based Reasoning in CADET

CADET: Database of mechanical devices
- Each training example: (qualitative function, mechanical structure)
- New query: desired function
- Target value: mechanical structure for this function

Distance measure: match qualitative function descriptions

Water faucet

A problem specification:

A stored case:

A junction pipe

Function:

Structure:

Q_1, T_1

Function:

Structure:
Case-Based Reasoning in CADET

- Instances represented by rich structural descriptions
- Multiple cases retrieved (and combined) to form solution to new problem
- Tight coupling between case retrieval and problem solving

Collaborative Filtering
(AKA Recommender Systems)

**Problem:**
Predict whether someone will like a Web page, newsgroup posting, movie, book, CD, etc.

**Previous approach:**
Look at content

**Collaborative filtering:**
- Look at what similar users liked
- Similar users = Similar likes & dislikes

Lazy vs. Eager Learning

**Lazy:** Wait for query before generalizing
- k-nearest neighbor, case-based reasoning

**Eager:** Generalize before seeing query
- ID3, FOIL, Naive Bayes, neural networks, . . .

Does it matter?
- Eager learner must create global approximation
- Lazy learner can create many local approximations
- If they use same \( H \), lazy can represent more complex functions (e.g., consider \( H = \) linear functions)

Collaborative Filtering

- Represent each user by vector of ratings
- Two types:
  - Yes/No
  - Explicit ratings (e.g., 0 *****)
- Predict rating:
  \[
  \hat{R}_{ik} = \overline{R}_i + \alpha \sum_{j \in \mathbf{N}_i} W_{ij}(R_{jk} - \overline{R}_j)
  \]
- Similarity (Pearson coefficient):
  \[
  W_{ij} = \frac{\sum_k (R_{ik} - \overline{R}_i)(R_{jk} - \overline{R}_j)}{\sqrt{\sum_k (R_{ik} - \overline{R}_i)^2 \sum_k (R_{jk} - \overline{R}_j)^2}}
  \]

Fine Points

- Primitive version:
  \[
  \hat{R}_{ik} = \alpha \sum_{j \in \mathbf{N}_i} W_{ij}R_{jk}
  \]
  \[
  \alpha = (\sum |W_{ij}|)^{-1}
  \]
- \( \mathbf{N}_i \) can be whole database, or only \( k \) nearest neighbors
- \( R_{jk} \) = Rating of user \( j \) on item \( k \)
- \( \overline{R}_j \) = Average of all of user \( j \)’s ratings
- Summation in Pearson coefficient is over all items rated by both users
- In principle, any prediction method can be used for collaborative filtering

**Example**

<table>
<thead>
<tr>
<th></th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
<th>( R_4 )</th>
<th>( R_5 )</th>
<th>( R_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>2</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>-</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Chris</td>
<td>5</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Diana</td>
<td>3</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>4</td>
</tr>
</tbody>
</table>
Instance-Based Learning: Summary

- k-Nearest Neighbor
- Other forms of IBL
- Collaborative filtering