Lecture 7

Instance-Based Learning

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)

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

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

Distance Measures

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

- Symbolic features:
  - Hamming/overlap
  - Value difference measure (VDM):
    \[
    \delta(\text{val}_i, \text{val}_j) = \sum_{\text{dim}} |P(\text{val}_i|\text{class}) - P(\text{val}_j|\text{class})|^n
    \]

- In general: arbitrary, encode knowledge

Voronoi Diagram

S: Training set

Voronoi cell of \(x \in S\):
All points closer to \(x\) than to any other instance in \(S\)

Region of class \(C\):
Union of Voronoi cells of instances of \(C\) in \(S\)
Behavior in the Limit

\( \varepsilon^*(x) \): Error of optimal prediction
\( \varepsilon_{NN}(x) \): Error of nearest neighbor

**Theorem:** \( \lim_{n \to \infty} \varepsilon_{NN} \leq 2\varepsilon^* \)

**Proof sketch (2-class case):**
\[
\varepsilon_{NN} = p_0p_{NN-} + p_p p_{NN+}
\]
\[
= p_0 (1 - p_{NN+}) + (1 - p_p) p_{NN+}
\]
\[
\lim_{n \to \infty} p_{NN+} = p_+, \quad \lim_{n \to \infty} p{NN-} = p_-
\]
\[
\lim_{n \to \infty} \varepsilon_{NN} = p_+ (1 - p_+) + (1 - p_p) p_+ = 2\varepsilon^* (1 - \varepsilon^*) \leq 2\varepsilon^*
\]
\[
\lim_{n \to \infty} (\text{Nearest neighbor}) = \text{Gibbs classifier}
\]

**Theorem:** \( \lim_{n \to \infty, k \to \infty, k/n \to 0} \varepsilon_{ANN} = \varepsilon^* \)

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
\[
w_i = \frac{1}{d(x_q, x_i)^2}
\]

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 \( BestF = \emptyset \)
  - 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 \( BestF = \emptyset \)
      - Then Let \( SS = SS \cup \{ BestF \} \)
Until \( BestF = \emptyset \) 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 \( WorstF = None \)
      - 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

Edited k-NN(S)
S: Set of instances
For each instance $x$ in $S$
  If $x$ is correctly classified by $S - \{x\}$
    Remove $x$ from $S$
  Return $S$

Edited k-NN(S)
$T$: Set of instances
$T = \emptyset$
For each instance $x$ in $S$
  If $x$ is not correctly classified by $T$
    Add $x$ to $T$
  Return $T$

Overfitting Avoidance

- Set $k$ by cross-validation
- Form prototypes
- Remove noisy instances
- E.g., remove $x$ if all of $x$’s $k$ nearest neighbors are of another class

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"

\[
\text{Training Radial Basis Function Networks}
\]

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

**Q2:** How to train weights (assume here Gaussian \( K_a \))
- First choose variance (and perhaps mean) for each \( K_a \)
  - E.g., use EM
- Then hold \( K_a \) 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 error33-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
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

Lazy vs. Eager Learning

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

Eager: Generalize before seeing query
- ID3, FOIL, Naïve 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

(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

Collaborative Filtering

- Represent each user by vector of ratings
- Two types:
  - Yes/No
  - Explicit ratings (e.g., 0 - 1 - 2 - 3 - 4 - 5)

- Predict rating:
  \[
  \hat{R}_{ik} = \bar{R}_i + \alpha \sum_{x \in N_i} W_{ij}(R_{jk} - \bar{R}_j)
  \]

- Similarity (Pearson coefficient):
  \[
  W_{ij} = \frac{\sum_{x \in N_i}(R_{ik} - \bar{R}_i)(R_{jk} - \bar{R}_j)}{\sqrt{\sum_{x \in N_i}(R_{ik} - \bar{R}_i)^2(R_{jk} - \bar{R}_j)^2}}
  \]

Fine Points

- Primitive version:
  \[
  \hat{R}_{ik} = \alpha \sum_{x \in N_i} W_{ij}R_{jk}
  \]

- \(\alpha = (\sum |W_{ij}|)^{-1}
- N_i can be whole database, or only k nearest neighbors
- \(R_{jk}\) = Rating of user j on item k
- \(\bar{R}_j\) = 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>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</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>
Second Project

- Implement collaborative filtering system
- Apply to EachMovie database (reduced version)
  - ~ 500 movies
  - ~ 5000 users
  - ~ 750,000 votes
- Improve and compare
  - Training and test databases
  - For each user in test database:
    - Predict rating for random movie
    - Compare with actual rating

Instance-Based Learning: Summary

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