Lecture 7

Instance-Based Learning

Preview

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

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 $\hat{f}(x_q) \leftarrow f(x_n)$

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)

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

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

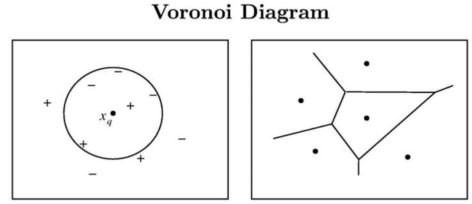
Distance Measures

- **Numeric features:**
  - Euclidean, Manhattan, L^n-norm:

    $L^n(\mathbf{x}_1, \mathbf{x}_2) = \sqrt[n]{\sum_{i=1}^{\#\text{dim}} |\mathbf{x}_{1,i} - \mathbf{x}_{2,i}|^n}$
  - Normalized by: range, std. deviation
- **Symbolic features:**
  - Hamming/overlap
  - Value difference measure (VDM):

    $\delta(val_i, val_j) = \sum_{h=1}^{\#\text{classes}} |P(c_h|val_i) - P(c_h|val_j)|^n$
- **In general:** arbitrary, encode knowledge

Voronoi Diagram



S: Training set

Voronoi cell of $\mathbf{x} \in S$:
All points closer to $\mathbf{x}$ 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

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

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

Proof sketch (2-class case):

$$\begin{aligned}\epsilon_{NN} &= p_+ p_{NN\in-} + p_- p_{NN\in+} \\ &= p_+(1 - p_{NN\in+}) + (1 - p_+) p_{NN\in+}\end{aligned}$$

$\lim_{n\to\infty} p_{NN\in+} = p_+, \quad \lim_{n\to\infty} p_{NN\in-} = p_-$

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

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

Theorem: $\lim_{n\to\infty,\, k\to\infty,\, k/n\to 0} \epsilon_{kNN} = \epsilon^*$

Distance-Weighted k-NN

Might want to weight nearer neighbors more heavily ...

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

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 = ∅
Let BestEval = 0
Repeat
    Let BestF = None
    For each feature F in FS and not in SS
        Let SS' = SS ∪ {F}
        If Eval(SS') > BestEval
            Then Let BestF = F
                 Let BestEval = Eval(SS')
    If BestF ≠ None
        Then Let SS = SS ∪ {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') ≥ BestEval
            Then Let WorstF = F
                 Let BestEval = Eval(SS')
    If WorstF ≠ None
        Then Let SS = SS − {WorstF}
Until WorstF = None or SS = ∅
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 $\mathbf{x}$ in S
    If $\mathbf{x}$ is correctly classified by $S - \{\mathbf{x}\}$
        Remove $\mathbf{x}$ from S
Return S

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

Overfitting Avoidance

- Set k by cross-validation
- Form prototypes
- Remove noisy instances
  - E.g., remove $\mathbf{x}$ if all of $\mathbf{x}$'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

$$E_1(x_q) \equiv \sum_{x \in \; kNN(x_q)} (f(x) - \hat{f}(x))^2$$

- Distance-weighted squared error over all neighbors

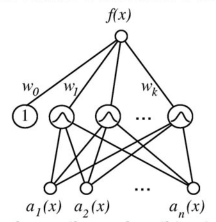
$$E_2(x_q) \equiv \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

- ...

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"

Radial Basis Function Networks



where $a_i(x)$ are the attributes describing instance x, and

$$f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

Common choice for K_u: $\quad K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$

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 $X \neq \Re^n$
$\rightarrow$ 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

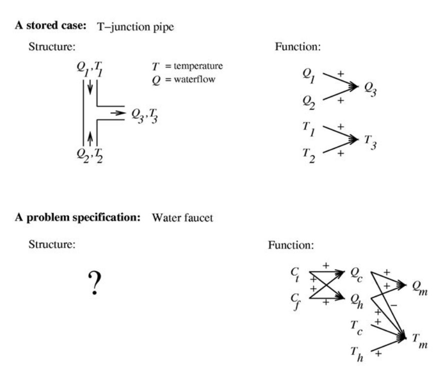
```
((user-complaint error53-on-shutdown)
 (cpu-model PentiumIII)
 (operating-system Windows2000)
 (network-connection Ethernet)
 (memory 128MB)
 (installed-applications Office PhotoShop VirusScan)
 (disk 10GB)
 (likely-cause ???))
```

Case-Based Reasoning in CADET

CADET: Database of mechanical devices

- Each training example:
  $\langle$qualitative function, mechanical structure$\rangle$
- 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, 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

(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 – $*****$)
- Predict rating:

$$\hat{R}_{ik} = \overline{R}_i + \alpha \sum_{X_j \in \mathbf{N}_i} W_{ij}(R_{jk} - \overline{R}_j)$$

- Similarity (Pearson coefficient):

$$W_{ij} = \frac{\sum_k (R_{ik} - \overline{R}_i)(R_{jk} - \overline{R}_j)}{\sqrt{\sum_k (R_{ik} - \overline{R}_i)^2 (R_{jk} - \overline{R}_j)^2}}$$

Fine Points

- Primitive version:

$$\hat{R}_{ik} = \alpha \sum_{X_j \in \mathbf{N}_i} W_{ij} R_{jk}$$

- $\alpha = (\sum |W_{ij}|)^{-1}$
- $\mathbf{N}_i$ 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

	R_1	R_2	R_3	R_4	R_5	R_6
Alice	2	-	4	4	-	5
Bob	1	5	4	-	3	4
Chris	5	2	-	2	1	-
Diana	3	-	2	2	-	4

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
    and compare with actual rating

Instance-Based Learning: Summary

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