09s1: COMP9417 Machine Learning and Data Mining

Instance Based Learning

April 22, 2009

Acknowledgement: Material derived from slides for the book
Machine Learning, Tom M. Mitchell, McGraw-Hill, 1997
`http://www-2.cs.cmu.edu/~tom/mlbook.html`

and slides by Andrew W. Moore available at
`http://www.cs.cmu.edu/~awm/tutorials`

and the book Data Mining, Ian H. Witten and Eibe Frank,
Morgan Kauffman, 2000. `http://www.cs.waikato.ac.nz/ml/weka`

and the book Pattern Classification, Richard O. Duda, Peter E. Hart
and David G. Stork. Copyright (c) 2001 by John Wiley & Sons, Inc.

Aims

This lecture will enable you to describe and reproduce machine learning approaches within the framework of instance based learning. Following it you should be able to:

- define instance based learning
- reproduce the basic k-nearest neighbour method
- describe locally-weighted regression
- describe radial basis function networks
- define case-based reasoning
- describe lazy versus eager learning

Reading: Mitchell, Chapter 8 & exercises (8.1), 8.3
Relevant WEKA programs: IB1, IBk, LWL

Introduction

- Simplest form of learning: rote learning
  - Training instances are searched for instance that most closely resembles new instance
  - The *instances* themselves represent the knowledge
  - Also called *instance-based* learning
- The *similarity function* defines what is "learned"
- Instance-based learning is *lazy* learning
- Methods: $nearest\text{-}neighbour$, $k\text{-}nearest\text{-}neighbour$, IBk, ...

Distance function

- Simplest case: one numeric (continuous) attribute
  - Distance is the difference between the two attribute values involved (or a function thereof)
- Several numeric attributes: normally, Euclidean distance is used and attributes are normalized
- Nominal (discrete) attributes: distance is set to 1 if values are different, 0 if they are equal
- Generalised distance functions: can use discrete and continuous attributes
- Are all attributes equally important?
  - Weighting the attributes might be necessary

Nearest Neighbour

Stores all training examples $\langle x_i, f(x_i)\rangle$.

Nearest neighbour:

- Given query instance x_q, first locate nearest training example x_n, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

k-Nearest neighbour:

- Given x_q, take vote among its k nearest neighbours (if discrete-valued target function)
- take mean of f values of k nearest neighbours (if real-valued)

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

k-Nearest Neighbour Algorithm

Training algorithm:

- For each training example $\langle x_i, f(x_i)\rangle$, add the example to the list *training_examples*.

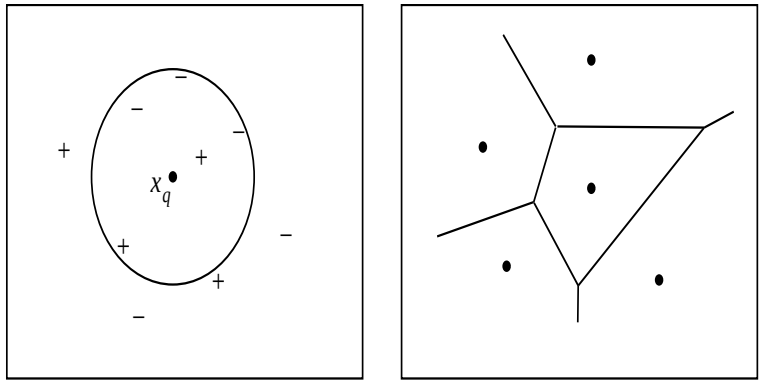
Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ be the k instances from *training_examples* that are *nearest* to x_q by the distance function
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and 0 otherwise.

"Hypothesis Space" for Nearest Neighbour



2 classes, $+$ and $-$ and query point x_q. On left, note effect of varying k. On right, $1-$NN induces a Voronoi tessellation of the instance space. Formed by the perpendicular bisectors of lines between points.

Distance function again

The distance function defines what is learned ...

- Most commonly used is *Euclidean distance*
  - instance x described by a feature vector (list of attribute-value pairs)

$$\langle a_1(x), a_2(x), \ldots a_m(x) \rangle$$

    where $a_r(x)$ denotes the value of the rth attribute of x
  - distance between two instances x_i and x_j is defined to be

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{m} (a_r(x_i) - a_r(x_j))^2}$$

- Many other distances, e.g. *Manhattan* or *city*-block (sum of absolute values of differences)

The idea of distance functions will appear again in *kernel methods*.

Normalization and other issues

- Different attributes measured on different scales
- Need to be *normalized*

$$a_r = \frac{v_r - \min v_r}{\max v_r - -\min v_r}$$

  where v_r is the actual value of attribute r

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

When To Consider Nearest Neighbour

- Instances map to points in $\Re^n$
- Less than 20 attributes per instance
- Lots of training data

Advantages:

- Statisticians have used k-NN since early 1950s
- Can be very accurate
  - As $n \to \infty$ and $k/n \to 0$ error approaches minimum
- Training is very fast
- Can learn complex target functions
- Don't lose information by generalization - keep all instances

Disadvantages:

- Slow at query time: basic algorithm scans entire training data to derive a prediction
- Assumes all attributes are equally important, so easily fooled by irrelevant attributes
  - Remedy: attribute selection or weights
- Problem of noisy instances:
  - Remedy: remove from data set (not easy)

What is the inductive bias of k-NN ?

- an assumption that the classification of query instance x_q will be most similar to the classification of other instances that are nearby according to the distance function

k-NN uses terminology from statistical pattern recognition (see below)

- *Regression* approximating a real-valued target function
- *Residual* the error $\hat{f}(x) - f(x)$ in approximating the target function
- *Kernel function* function of distance used to determine weight of each training example, i.e. kernel function is the function K s.t. $w_i = K(d(x_i, x_q))$

Distance-Weighted kNN

- Might want to weight nearer neighbours more heavily ...
- Use distance function to construct a weight w_i
- Replace the final line of the classification algorithm by:

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

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

For real-valued target functions replace the final line of the algorithm by:

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

Now we can consider using *all* the training examples instead of just k

$\rightarrow$ using all examples (i.e. when $k = n$) with the rule above is called Shepard's method

Curse of Dimensionality

Bellman (1960) coined this term in the context of dynamic programming

Imagine instances described by 20 attributes, but only 2 are relevant to target function

Curse of dimensionality: nearest neighbour is easily mislead when high-dimensional X – problem of irrelevant attributes

One approach:

- Stretch jth axis by weight z_j, where $z_1, \ldots, z_n$ chosen to minimize prediction error
- Use cross-validation to automatically choose weights $z_1, \ldots, z_n$
- Note setting z_j to zero eliminates this dimension altogether

See Moore and Lee (1994) *"Efficient Algorithms for Minimizing Cross Validation Error"*

Instance-based methods (IBk)

- attribute weighting: class-specific weights may be used (can result in unclassified instances and multiple classifications)
- get Euclidean distance with weights

$$\sqrt{\sum w_r(a_r(x_q) - a_r(x))^2}$$

- Updating of weights based on nearest neighbour
  - Class correct/incorrect: weight increased/decreased
  - $|a_r(x_q) - a_r(x)|$ small/large: amount large/small

Locally Weighted Regression

Use kNN to form a local approximation to f for each query point x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

where $a_r(x)$ denotes the value of the rth attribute of instance x

Key ideas:

- Fit linear function to k nearest neighbours
- Or quadratic or higher-order polynomial ...
- Produces "piecewise approximation" to f

In statistics known as LO(W)ESS (Cleveland, 1979)

Recall a global method of learning a function ...

Minimizing squared error summed over the set D of training examples:

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

using gradient descent training rule:

$$\Delta w_r = \eta \sum_{x \ D} (f(x) - \hat{f}(x)) a_r(x)$$

Going from a *global* to a *local* approximation there are several choices of error to minimize:

1. Squared error over k nearest neighbours

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} (f(x) - \hat{f}(x))^2$$

2. Distance-weighted squared error over all neighbours

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \ K(d(x_q, x))$$

3. Combine 1 and 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

Gives *local* training rule:

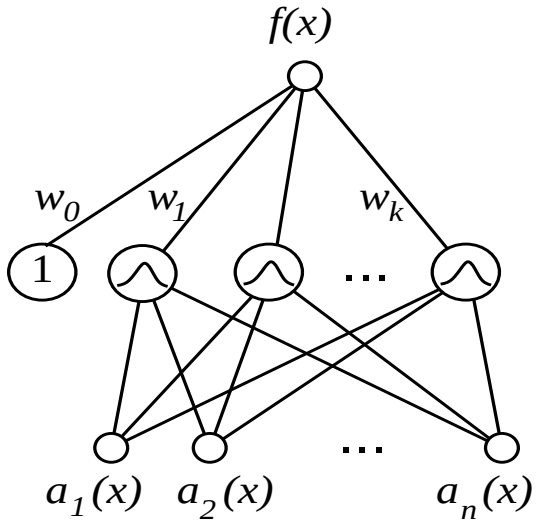
$$\Delta w_r = \eta \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} K(d(x_q, x))(f(x) - \hat{f}(x)) a_r(x)$$

Note: use more efficient training methods to find weights.

Atkeson, Moore, and Schaal (1997) "Locally Weighted Learning."

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"



In the diagram $a_r(x)$ are the attributes describing instance x.

The learned hypothesis has the form:

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

where each x_u is an instance from X and the *kernel function* $K_u(d(x_u, x))$ decreases as distance $d(x_u, x)$ increases.

One common choice for $K_u(d(x_u, x))$ is

$$K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

i.e. a Gaussian function.

Training Radial Basis Function Networks

Q1: What x_u (subsets) to use for each kernel function $K_u(d(x_u, x))$

- Scatter uniformly throughout instance space
- Or use training instances (reflects instance distribution)
- Or prototypes (found by clustering)

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

Instance-based learning (IBk)

Recap – Practical problems of 1-NN scheme:

- Slow (but: fast tree-based approaches exist)
  - Remedy: removing irrelevant data
- Noise (but: k-NN copes quite well with noise)
  - Remedy: removing noisy instances
- All attributes deemed equally important
  - Remedy: attribute weighting (or simply selection)
- Doesn't perform explicit generalization
  - Remedy: rule-based NN approach

Edited NN

- Edited NN classifiers discard some of the training instances before making predictions
- Saves memory and speeds-up classification
- IB2: incremental NN learner that only incorporates misclassified instances into the classifier
  - Problem: noisy data gets incorporated
- Other approach: Voronoi-diagram-based
  - Problem: computationally expensive
  - Approximations exist

Dealing with noise

- Excellent way: cross-validation-based k-NN classifier (but slow)
- Different approach: discarding instances that donst perform well by keeping success records (IB3)
  - Computes confidence interval for instances success rate and for default accuracy of its class
  - If lower limit of first interval is above upper limit of second one, instance is accepted (IB3: 5%-level)
  - If upper limit of first interval is below lower limit of second one, instance is rejected (IB3: 12.5%-level)

Case-Based Reasoning

Can apply instance-based learning even when $X \neq \Re^n$

$\rightarrow$ need different "distance" metric

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

```
((user-complaint error53-on-shutdown)
 (cpu-model PowerPC)
 (operating-system Windows)
 (network-connection PCIA)
 (memory 48meg)
 (installed-applications Excel Netscape VirusScan)
 (disk 1gig)
 (likely-cause ???))
```


Case-Based Reasoning in CADET

CADET: 75 stored examples of mechanical devices

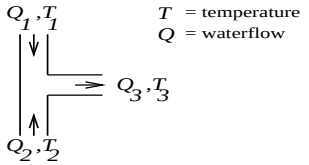
- each training example: $\langle$ qualitative function, mechanical structure$\rangle$
- new query: desired function,
- target value: mechanical structure for this function

Distance metric: match qualitative function descriptions

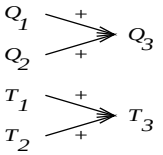
Case-Based Reasoning in CADET

A stored case: T−junction pipe

Structure:



Function:

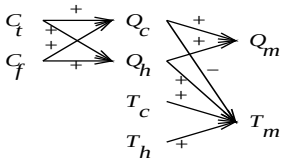


A problem specification: Water faucet

Structure:

?

Function:



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

Bottom line:

- Simple matching of cases useful for tasks such as answering help-desk queries
- Area of ongoing research

Summary: Lazy vs Eager Learning

Lazy: wait for query before generalizing

- k-Nearest Neighbour, Case based reasoning

Eager: generalize before seeing query

- Radial basis function networks, ID3, Backpropagation, NaiveBayes, ...

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 fns (e.g., consider $H =$ linear functions)