

Intro. to machine learning (CSI 5325)

Lecture 20: Instance-based learning

Greg Hamerly

Some content from Tom Mitchell.

- 1 Locally weighted regression
- 2 Radial basis functions
- 3 Learning linear functions
- 4 Lazy and eager learning

Locally Weighted Regression

Note k -NN forms local approximation 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, constant, cubic, etc...
- Produces “piecewise approximation” to f

Locally Weighted Regression (2)

Several options of error functions:

- Squared error over k nearest neighbors (in $N(x_q, k)$)

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

- Distance-weighted squared error over all neighbors

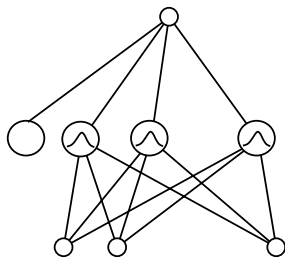
$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

- Combination of the two (kernel-weighted over k neighbors)

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

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

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

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

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
 - Or use training instances (reflects instance distribution)
 - Or use the means of clusters (found by k -means, Gaussian EM, etc.)
- 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

Methods for learning linear functions

Consider a linear function of x , where $x_0 = 1$:

$$f(x) = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d$$

Mitchell has discussed several methods for learning the weights of a linear function based on gradient descent.

However, if we assume a squared error function:

$$E(\hat{f}) = \frac{1}{2} \sum_{i=1}^n (f(x_i) - \hat{f}(x_i))^2$$

then much more efficient training methods exist.

Deriving linear weights (WARNING: linear algebra)

Rewrite the error function in matrix form:

$$\begin{aligned} E(\hat{f}) &= \frac{1}{2} \sum_{i=1}^n (f(x_i) - \hat{f}(x_i))^2 \\ &= \frac{1}{2} \sum_{i=1}^n (f(x_i) - (w_0x_0 + w_1x_1 + \dots + w_dx_d))^2 \\ &= \frac{1}{2} (Xw - y)^T (Xw - y) \end{aligned}$$

Where here

- $X = [x_1, x_2, x_3, \dots, x_n]^T$ is an $n \times (d + 1)$ matrix of n input vectors
- $y = [f(x_1), f(x_2), \dots, f(x_n)]^T$ is a vector of n outputs
- $w = [w_0, w_1, \dots, w_d]^T$ is a vector of $d + 1$ weights

Deriving linear weights

Take the derivative of E with respect to the weights:

$$\begin{aligned}E(\hat{f}) &= \frac{1}{2}(Xw - y)^T(Xw - y) \\ \frac{\partial E}{\partial w} &= (X^T)(Xw - y) \\ &= X^T Xw - X^T y\end{aligned}$$

Setting this equal to zero to find the minimum value:

$$\begin{aligned}X^T Xw - X^T y &= 0 \\ X^T Xw &= X^T y \\ (X^T X)^{-1}X^T Xw &= (X^T X)^{-1}X^T y \\ \hat{w} &= (X^T X)^{-1}X^T y\end{aligned}$$

Easily computed in any numerical package (e.g. Matlab), with the majority of the cost being a $(d + 1) \times (d + 1)$ matrix inversion.

General applicability of linear models

Linear models are extremely popular because:

- they can be solved efficiently (see previous slides)
- represent more complex functions through 'basis expansions'
 - construct a linear combination of nonlinear features

Many machine learning algorithms are related to some sort of linear model:

- linear output perceptron (without thresholding)
- radial basis network
- locally weighted regression
- non-linear (e.g. polynomial) regression
- support vector machines
- etc.

Lazy and Eager Learning

Lazy: wait for query before generalizing

- k -NEAREST NEIGHBOR, 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 functions (e.g., consider $H =$ linear functions)