Übung zur Vorlesung

Maschinelles Lernen 1

Wintersemester 2009/2010

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und
theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Blatt 8

Abgabe bis Abgabe bis **Mittoch, den 9. Dezember 2009, 14 Uhr** bei Mikio Braun (FR 6058, notfalls unter der Türe durchschieben) und per Email an mikio@cs.tu-berlin.de.

Bias-Varianz-Dilemma

In der Vorlesung wurde die Bias-Varianz-Zerlegung vorgestellt. Wir betrachten ein Lernproblem mit Verteilung $(X, Y) \sim p(x, y)$. Es sei

$$f(x) = E(Y|X = x),$$

d.h. der Erwartungswert von Y am Punkt x. Dies ist die optimale Vorhersage bezüglich des quadratischen Abstandsmaßes $l(y, y') = (y - y')^2$.

Wir halten einen Punkt x fest und betrachten die Vorhersage eines Lernalgorithmus g unter einem zufällig gewählten Trainingsdatensatzes D.

$$E_D(g_D(x) - f(x))^2 = \underbrace{\left[E_D(g_D(x) - f(x))\right]^2}_{\text{Bias}^2} + \underbrace{E_D\left[g_D(x) - E_D(g_D(x))\right]^2}_{\text{Varianz}}$$

Auf diesem Aufgabenblatt soll der Bias und die Varianz experimentell gemessen werden.

Hierbei werden die Erwartungswerte geschätzt, in dem wiederholt einen Datensatz generiert, und die Vorhersagen an einer Anzahl von Stützstellen notiert und daraus Bias und Varianz berechnet, d.h.

$$\hat{b}(x) = \frac{1}{m}\sum_{i=1}^{m}(g_{D_i}(x) - f(x)).$$

$$\hat{v}(x) = \frac{1}{m-1}\sum_{i=1}^{m}\left(g_{D_i}(x) - \frac{1}{m}\sum_{j=1}^{m} g_{D_j}(x)\right)^2.$$

Da es sich um einen künstlich generierten Datensatz handelt, ist $f(x)$ bekannt.

Implementiere die folgenden Funktionen (jeweils 10 Punkte):

- `variance_per_point` Berechnet die Varianz $\hat{v}(x)$ für eine Reihe von Punkten $x_1, \ldots, x_n$ gelernt auf Datensätzen $D_1, \ldots, D_m$.
- `squared_bias_per_point` Berechnet den quadrierten Bias $\hat{b}(x)^2$ für eine Reihe von Punkten $x_1, \ldots, x_n$ gelernt auf Datensätzen $D_1, \ldots, D_m$.
- `squared_error_per_point` Berechnet den quadratischen Fehler

  $$\hat{e}(x) = \frac{1}{m}\sum_{i=1}^{m}(g_{D_i} - f(x))^2$$

  für eine Reihe von Punkten $x_1, \ldots, x_n$ gelernt auf Datensätzen $D_1, \ldots, D_m$.

<u>Übergeben wird jeweils eine Matrix Y, die die Vorhersagen für alle Punkte und Datensätze enthält.</u>

```
function sheet08

% The parameters:
M = 1000; % number of evaluation points
X = linspace(-4, 4, M)'; % the evaluation points
ITERS = 100; % number of resamples
WIDTHS = logspace(-3, 1, 9); % the kernel widths
N = 100; % number of data points
```

```
% set up some empty vectors to collect the information
BIAS2 = zeros(length(WIDTHS), 1);
VARIANCE = zeros(length(WIDTHS), 1);
ERR = zeros(length(WIDTHS), 1);

% plot individual means and variances for 9
% different widths
figure(1)
for WI = 1:length(WIDTHS)
  W = WIDTHS(WI)

  % collect predictions for ITERS many resamples
  % predictions for one resample are stored in the columns of Y
  Y = zeros(M, ITERS);
  for I = 1:ITERS
    [DX, DY] = generate_data(N);
    Y(:, I) = kernel_regression(W, DX, DY, X);
  end

  % compute mean and variance
  ME = mean(Y, 2);
  VA = variance_per_point(Y);

  % plot the result
  subplot(3, 3, WI)
  plot(DX, DY, '.', ... % the last data set
       X, f(X), 'b-', ... % the true function
       X, ME, 'r-', ... % the mean of the predictions
       X, ME + 2*sqrt(VA), 'r--', ... % +- 2*sqrt(variance)
       X, ME - 2*sqrt(VA), 'r--', ...
       X, Y(:, end), 'g-'); % the last learned function

  % compute mean squared bias, variance, and error
  BIAS2(WI) = mean(squared_bias_per_point(Y, f(X)));
  VARIANCE(WI) = mean(VA);
  ERR(WI) = mean(squared_error_per_point(Y, f(X)));
  title(sprintf('width = %f\n bias^2 = %f, var = %f', ...
                W, BIAS2(WI), VARIANCE(WI)));
  drawnow
end

% plot squared bias, variance, and error
figure(2)
semilogx(WIDTHS, BIAS2, WIDTHS, VARIANCE, WIDTHS, ERR)
legend('bias^2', 'variance', 'error')

% Generate data
function [X, Y] = generate_data(N)
X = sort(rand(N, 1) * 8 - 4);
Y = f(X) + 0.5 * randn(N, 1);

% the true function (sinc function + linear offset)
function Y = f(X)
Y = sin(4*X)./(4*X) + X/2;

% Compute kernel regression estimates
function YE = kernel_regression(W, X, Y, XE)
K = rbfkern(W, X, XE);
YE = Y'*K ./ sum(K);
```

```
% Compute a Gaussian kernel
function K = rbfkern(W, X1, X2)
D = pwdist(X1, X2);
K = exp(-D / W);

% Compute all pairwise distances
function D = pwdist(X, Y)
D = size(X, 2);
N = size(X, 1);
M = size(Y, 1);

XX = sum(X.*X, 2);
YY = sum(Y.*Y, 2);
D = repmat(XX, 1, M) + repmat(YY', N, 1) - 2*X*Y';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your solutions below

% 1. Compute the variance per point
%
% Arguments:
%  Y: matrix where the columns are the predictions for a single
%     version of the data set
function V = variance_per_point(Y)
% ...

% 2. Compute the squared bias per point
%
% Arguments:
%  Y: see 1
%  F: a column vector of the true function
function B2 = squared_bias_per_point(Y, F)
% ...

% 3. Compute the squared error per point.
%
% Arguments:
%  Y: see 1
%  F: see 2
function E = squared_error_per_point(Y, F)
% ...
```

Für Fragen zum Übungsblatte bitte in der Google Group http://groups.google.com/group/mikiobraun-lehre registrieren und die Frage an die Mailingliste stellen.