Übung zur Vorlesung

Maschinelles Lernen 1

Wintersemester 2008/2009

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

Blatt 11

Abgabe bis Mittwoch, 21. Januar 2009, 14:00 Uhr
Ausarbeitung im Sekretariat FR6052, oder bei Mikio Braun, FR6058 (notfalls unter der Tür durchschieben),
praktischen Teil unter https://ml01.service.tu-berlin.de/~mikio/pass.pl?conf=blatt11.conf.

Aufgaben (Modellselektion)

Diesmal sollen verschiedene Modellselektionsalgorithmen implementiert werden. Das Programmskelett generiert einen Trainings- und Testdatensatz und implementiert Least-Squares-Regression mit einer Basisfunktion bestehend aus "Bumps" der Form

$$\phi_j(x) = \exp\left(-\frac{(x-c_j)^2}{2w^2}\right),$$

wobei w die Breite des Bumps ist, und c_j der Ort (im Programm `Xbase`).

Ergänze den fehlenden Code im Programmskelett:

1. `cp_statistic`: Implementiere die C_p-Statistik

   $$C_p = \text{Trainingsfehler} + \frac{2}{n}\text{trace}(S)\hat{\sigma}_\varepsilon^2,$$

   wobei S die sogenannte "hat matrix" ist, d.h. die Matrix, die aus den Eingabelabeln Y die Vorhersagen $\hat{Y}$ auf allen Trainingsbeispielen berechnet, und $\hat{\sigma}_\varepsilon^2$ ist die Schätzung der Rauschvarianz, gegeben durch

   $$\hat{\sigma}_\varepsilon^2 = \frac{\text{Trainingsfehler}}{n - \text{trace}(S)}.$$

   Hierbei verwendet man üblicherweise ein Modell mit vielen Freiheitsgraden, damit die Schätzung verzerrungsfrei ist (d.h. eine kleine Kernbreite). Zurückgegeben werden sollen sowohl die Freiheitsgrade (Spur von S), als auch die C_p-Statistik selbst.

2. `cv`: Implementiere k-fache Kreuzvalidierung. Gib alle k Testfehler als Spaltenvektor zurück.

Für die Implementierung beider Funktionen kann bzw. soll auf die bereits vorhandenen Funktionen, insbesondere `learn` und `predict`, zurückgegriffen werden. Für die C_p-Statistik finden sich die entsprechenden Codestücke zum Aufstellen der "hat matrix" ebenfalls dort.

```
function sheet11_solution

%
% Generate some data
%
[X, Y] = generate_data(50); % training data
[XE, YE] = generate_data(1000); % test data

figure(1);
plot(X, Y, 'k+', 'LineWidth', 3, 'MarkerSize', 10);

Xbase = linspace(-15, 15, 100)';

% Plot the fit for a number of weights
widths = [0.1, 1, 10];
STYLES = {'r-', 'b-', 'g-'};
LEGENDS = cell(1, length(widths)+1);
```

```
LEGENDS{1} = 'data';
hold on;
for wi = 1:length(widths)
  w = widths(wi);
  C = learn(w, Xbase, X, Y);
  YEh = predict(C, XE);
  plot(XE, YEh, STYLES{wi});
  LEGENDS{wi+1} = sprintf('width = %.1f', w);
end
legend(LEGENDS)
hold off;

% in the following, we will use these
% candidates for the width
widths = logspace(-2, 2, 50);

% collect all the different errors
TRAIN_ERR = zeros(1, length(widths));
TEST_ERR = zeros(1, length(widths));
DOF = zeros(1, length(widths));
CV5_ERR = zeros(5, length(widths));
CV10_ERR = zeros(10, length(widths));
CP_ERR = zeros(1, length(widths));
for wi = 1:length(widths)
  w = widths(wi);
  C = learn(w, Xbase, X, Y);
  Yh = predict(C, X);
  TRAIN_ERR(wi) = l2err(Y, Yh);
  YEh = predict(C, XE);
  TEST_ERR(wi) = l2err(YE, YEh);

  [DOF(wi), CP_ERR(wi)] = cp_statistic(w, Xbase, X, Y);
  CV5_ERR(:,wi) = cv(5, w, Xbase, X, Y);
  CV10_ERR(:,wi) = cv(10, w, Xbase, X, Y);
end

figure(2)
semilogx(widths, TRAIN_ERR, 'r.-', ...
         widths, TEST_ERR, 'b.-', ...
         widths, CP_ERR, 'g.-');
legend('train', 'test');
grid

figure(3)
semilogx(widths, DOF)
title('degrees-of-freedom');

% generate data from the sinc function
function [X, Y] = generate_data(N)
X = sort(30*rand(N,1) - 15);
Y = sin(X)./X + 0.1*randn(N,1);

% learn a function comprised of Gaussian bumps
% at certain base points
function C = learn(width, Xbase, X, Y)
PSI = design_matrix(width, Xbase, X);
C.W = (PSI*PSI' + 0.1 * eye(length(Xbase)))\(PSI*Y);
C.width = width;
C.Xbase = Xbase;

% compute pairwise distances between the rows of X and Y.
```

```
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';

% Set up the design matrix for a basis function of
% Gaussian bumps
function PSI = design_matrix(width, Xbase, X)
D = pwdist(Xbase, X);
PSI = exp(-D/width/width);

% Predict values for the learned parameters
function Yh = predict(C, X)
Yh = design_matrix(C.width, C.Xbase, X)'*C.W;

% compute the l2-error
function l = l2err(Y1, Y2)
l = mean((Y1 - Y2).^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Your solution below

% 1. Compute the CP statistics. This includes:
%
%   - computing the training error
%   - computing the trace of the hat matrix
%   - estimating the noise level
%
% Return the degrees-of-freedom and the C_p statistic.
function [DOF, CP] = cp_statistic(w, Xbase, X, Y)
DOF = 1; % placeholders to make the script run
CP = 0;
% ...

% 2. Estimate the cross-validation error. Return all the individual test
% errors as a column vector.
function ERR = cv(K, w, Xbase, X, Y)
ERR = zeros(K, 1); % placeholder to make the script run
% ...
```