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 14

Abgabe bis Mittwoch, 11. Februar 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=blatt14.conf.

Mercerkerne und Support-Vektor-Maschinen

1. Untersuchen Sie, ob die Funktion

$$k(x, y) = \tanh(\langle x, y \rangle + 1)$$

ein Mercerkern ist oder nicht. **(10 Punkte)**

2. In dieser Aufgabe soll eine Support-Vektor-Maschine unter Benutzung eines generischen quadratischern Optimierers implementiert werden. Für matlab ist dies die Funktion `quadprog`, für octave `qp`. Ergänze die fehlenden Funktionen im Programmskelett `sheet14.m`:

   (a) `trainSVM`: Trainiere eine SVM. Zurückgegeben wird eine Struct mit den Feldern

   alpha: die gelernte Gewichte

   b: der gelernte Offset

   y: der Vektor von Trainingslabeln.

   Verwendet werden soll der generische quadratische Optimierer auf dem dualen Problem

$$\begin{aligned} \max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(X_i, X_j) \\ \text{so dass } 0 \leq \alpha_i \leq C, \qquad 1 \leq i \leq n \\ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}$$

   Wie wird b gelernt? **(10 Punkte)**

   (b) `predictSVM`: Sage neue Label voraus. Übergeben wird eine entsprechende Kernmatrix, die im ersten Argument die Ausgabepunkte und im linken Argument die Eingabepunkte enthält. **(5 Punkt)**

   (c) `rbfkern`: Berechne den rbf-Kernel

$$k(x, y) = \exp\left(\frac{\|x - y\|^2}{w}\right).$$

   Falls Y fehlt, soll $X = Y$ angenommen werden. Verwende keine for-Schleifen, um die paarweisen Abstände zu berechnen! **(5 Punkte)**

```
function sheet14

% generate some data
sheet14_data

% plot data
plot(X(Y == 1, 1), X(Y == 1, 2), 'r+', ...
     X(Y == -1, 1), X(Y == -1, 2), 'bo');

% learn an SVM
```

```
width = 0.01;
C = 1e5;

K = rbfkern(width, X);
svm = trainSVM(K, Y, C);
Yh = predictSVM(K, svm);

% predict the solution
N = 50;
[MX, MY] = meshgrid(linspace(0, 1, N), linspace(0, 1, N));
XP = [reshape(MX, N*N, 1), reshape(MY, N*N, 1)];
KP = rbfkern(width, XP, X);
YP = predictSVM(KP, svm);
F = reshape(YP, N, N);
hold on
contour(MX, MY, F, [-1, 0, 1]);
% comment out next line for octave
surf(MX, MY, F, 'FaceAlpha', 0.2); shading interp; caxis([-2 2])
hold off
grid
colormap([linspace(1, 0, 100)', linspace(1, 0, 100)', ones(100, 1), ;
          0 0 0;
          ones(100, 1), linspace(0, 1, 100)', linspace(0, 1, 100)'])

title(sprintf('Error rate: %.2f%%', mean(sign(Yh) ~= Y) * 100));

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

% 3a. train a support vector machine using the built-in generic quadratic
% optimizer (quadprog for matlab, qp for octave). Return a structure with
% the following entries:
%
%   svm.alpha - learned alphas
%   svm.b     - learned b
%   svm.y     - training Ys
function svm = trainSVM(K, Y, C)
N = length(Y);
% ...

% 3b. Predict the labels given the kernel matrix built from the
% test/training data points, and the svm structure returned by trainSVM.
function Yh = predictSVM(K, svm)
% ...

% 3c. Compute the rbf kernel. If Y is missing, assume X = Y. Do not use
% for loops to compute the pairwise distances!
function K = rbfkern(w, X, Y)
% ...
```
