Übung zur Vorlesung

# Maschinelles Lernen 1

Wintersemester 2010/2011

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 10

Abgabe bis Mittwoch, den 19. Januar 2011 um 12:00

Die Lösungen bitte im **Postfach** von Dr. Konrad Rieck abgeben.

## Support-Vektor-Maschinen

In dieser Übung soll eine Support-Vektor-Maschine (SVM) implementiert werden. Wie in der Vorlesung vorgestellt kann eine SVM durch das folgende quadratische Optimierungsproblems trainiert werden.

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

Ergänze die fehlenden Funktionen im Programmskelett:

1. `trainSVM`: Trainiere eine SVM. Um das quadratische Problem (1) zu lösen, steht unter Matlab die Funktion `quadprog` zur Verfügung und unter Octave die Funktion `qp`. Als Ergebnis soll eine Struktur mit folgenden Feldern zurück gegeben werden

   **alpha:** die gelernten Gewichte

   **b:** der gelernte Offset

   **y:** der Vektor von Trainingslabeln.

   Wie wird $b$ gelernt? **(15 Punkte)**

2. `predictSVM`: Sage neue Label voraus. Übergeben wird eine Kernmatrix, deren Einträge paarweise aus den Test- und Trainingsdaten berechnet werden. **(8 Punkt)**

3. `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! **(7 Punkte)**

---

```
function sheet10

% generate some data
sheet10_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).
% Input:
%   K          - kernel matrix (n x n) of training data
%   Y          - labels (1 x n)
%   C          - regularization constant
% Output:
%   svm.alpha  - learned alphas
%   svm.b      - learned b
%   svm.y      - training Ys
function svm = trainSVM(K, Y, C)
% ...

% 3b. Predict the labels given the kernel matrix built from the
% test/training data points, and the svm structure returned by trainSVM.
% Input:
%   K:         - kernel matrix (m x n) of test/training data
%   svm:       - svm structure
% Output:
%   Yk         - predicted labels (1 x m)
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!
% Input:
%   w:         - kernel width
%   X:         - Input vectors (d x n)
%   Y:         - Input vectors (d x m)
% Output:
%   K:         - Kernel matrix (n x m)
function K = rbfkern(w, X, Y)
% ...
```

---

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