Übung zur Vorlesung

Maschinelles Lernen 2

Sommersemester 2009

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

Blatt 6

Abgabe bis Montag, 1. Juni 2008, 13 Uhr bei Mikio Braun (mikio@cs.tu-berlin.de).
Ausarbeitung schriftlich abgeben, Programm per Email.

Aufgaben

1. **(10 Punkte)** Zeige, dass die duale Darstellung der One-Class-SVM mit Soft Margin (vgl. Seite 26 der Folien) tatsächlich die Gestalt hat

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i k(x_i, x_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \\ \text{so dass} \quad & \sum_{i=1}^{n} \alpha_i = 1 \quad \text{and} \quad 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, n, \end{aligned}$$

   wobei $\{x_1, \ldots, x_n\}$ Trainingsdaten sind und k einer beliebigen Kernfunktion entspricht.

2. **(5 Punkte)** Gib die allgemeine duale quadratische Darstellung der One-Class SVM in Matrix-/Vektornotation in der Form

$$\begin{aligned} \max_{\alpha} \quad & c^T \alpha + \alpha^T B \alpha \\ \text{so dass} \quad & d^T \alpha = e, \\ & l_i \leq \alpha_i \leq u_i, \text{ für } 1 \leq i \leq n. \end{aligned}$$

3. **(7 Punkte)** Ergänze im Programmskelett die Funktion `oneclass`, die gegeben die Kernmatrix $K \in \mathbb{R}^{n \times n}$ die Kernmatrix und die Regularisierungskonstante $C \geq 0$, den Vektro $\alpha \in \mathbb{R}^n$ von Kernkoeffizienten aus der dualen Darstellung berechnet. Verwende zur Optimierung die Funktion `pr_loqo2`.

4. **(8 Punkte)**. Finde mit der One-Class SVM Hacker-Angriffe in einem vorverarbeiteten Datensatz von Netzwerkverkehr. Du findest den Datensatz auf der Veranstaltungsseite. Der Datensatz ist in eine Trainings- und Testpartition unterteilt. Wie in einem realistischen Einsatz enthalten beide Partitionen Angriffe.

   Verwende einen linearen Kern. Die Regularisierungskonstante C kann durch Raten, Probieren oder durch Kreuzvalidierung ermittelt werden. Gib das von Dir gewählte C explizit an, und dokumentiere, wie Du C gewählt hast. Übergib eine Datei mit den Positionen der Angriffe des Testdatensatzes.

   Hinweise: Der Datensatz wurde aus echten Verbindungen und Angriffen des HTTP-Protokolls generiert. Jede Verbindung x wurde über 3-Gramme in einen Vektor $\phi(x)$ abgebildet. Die Daten sind normalisiert, d.h. $||\phi(x)||_1 = 1$.

   Das Programmskelett enthält keine Kreuzvalidierung oder ähnliches. Dies müsste ggf. im oberen Teil des Programmskelettes eingefügt werden.

   Wie in der Vorlesung besprochen, wird die Abweichung eines Punktes z vom erlernten Model α ermittelt durch

$$a(z) = k(z, z) - 2 \sum_{i=1}^{n} \alpha_i k(x_i, z) + \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j).$$

   Die Funktion `compute_scores` berechnet diese Größe.

```
function sheet05
fprintf('loading data...\n')
load('stud-data.mat')

% compute kernel matrices
fprintf('computing kernel matrices...\n')
KR = full(Xtr'*Xtr);
KS = full(Xts'*Xts);
KSR = full(Xts'*Xtr);

% compute the alphas
fprintf('learning one-class-SVM...')
C = ?; % <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< adjust C accordingly!
[a, alpha] = oneclass(KR, C);

% compute predicted outlier scores
as = compute_scores(KS, KSR, KR, alpha);

Ap = (as > 1);

predicted_attacks = find(Ap)'
% ...

function [x,y] = pr_loqo2(c, H, A, b, l, u)
%[X,Y] = PR_LOQO2(c, H, A, b, l, u)
%
%loqo solves the quadratic programming problem
%
%minimize   c' * x + 1/2 x' * H * x
%subject to A'*x = b
%           l <= x <= u
%
% Dimensions: c : N-column vector
%             H : NxN matrix
%             A : N-row vector
%             b : real number
%             l : N-column vector
%             b : N-column vector
%
%             x : N-column vector
%             y : ??
%
%for a documentation see R. Vanderbei, LOQO: an Interior Point Code
%                        for Quadratic Programming

% function body hidden for saving space...

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

% 3. Train a one-class SVM given the kernel matrix K and the
% regularization constant C.
function [a, alpha] = oneclass(K, C)
```

```
% ...

% 4. Compute the outlier scores given
%
%    KR: kernel matrix on training data
%    KS: kernel matrix on test data
%    KSR: kernel matrix on test data / training data
%    alpha: learned kernel coefficients
function scores = compute_scores(KS, KSR, KR, alpha)
% ...
```
