Übung zur Vorlesung

Maschinelles Lernen 2

Sommersemester 2010

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

Blatt 7

Abgabe bis Montag, 31. Mai 2010, 13:00 Uhr bei Dr. Konrad Rieck.

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

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

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

2. **(5 Punkte)** Gib die duale Darstellung der One-class SVM in Matrix-/Vektornotation, so dass sie die folgende allgemeine quadratische Form hat.

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

3. **(5 Punkte)** Ergänze im Programmskelett die Funktion `oneclass`, die gegeben eine Kernmatrix $K \in \mathbb{R}^{n \times n}$ und eine Regularisierungskonstante $C \ge 0$, den Vektor $\alpha \in \mathbb{R}^n$ von Koeffizienten aus der dualen Darstellung berechnet. Verwende zur Optimierung die Funktion `pr_loqo2`.

4. **(10 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. Lerne zunächst ein Modell auf dem Trainingsdatensatz und wende dieses dann auf die Testdaten an.

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

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

   Implementiere diese Funktion `compute_scores`.

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

   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. $||\phi(x)||_1 = 1$.

```
function sheet07
disp('loading data...')
load('stud-data.mat')

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

```
KSR = full(Xts'*Xtr);

% compute the alphas
disp('learning one-class-SVM...')
C = ?; % adjust C
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... See full file.

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

% 3. Train a one-class SVM given the kernel matrix K and the
% regularization constant C.
function 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)
% ...
```

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