Abgabe: FleischmannRohrmann

compute_scores.m

```
function scores = compute_scores(KS, KSR, KR, alpha)
% inputs: KR       kernel matrix on training data
%         KS       kernel matrix on test data
%         KSR      kernel matrix on test data/training data
%         alpha    learnt dual vector
% output: scores   vector of outlier scores
N=length(KS);
scores = zeros(N,1);
for i=1:N
    scores(i,1)=KS(i,i)-2*KSR(i,:)*alpha+alpha'*KR*a
end
```

dont use loops,
loops get slow and
inaccurate for huge
amounts of data.

this can be a
one-line-statement!
10 / 10

hacker_detection.m

```
load('stud-data.mat')

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

% compute the alphas
disp('learning one-class-SVM...')
C = 0.002; % adjust C
alpha = oneclass(KR, C);

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

Ap = (as > 1);

predicted_attacks = find(Ap)';

hold on;
plot(as);
plot(1:length(as),1,'r-');

predicted_attacks
```

oneclass.m

```
function alpha = oneclass(K, C)
% inputs:  K       N x N kernel matrix
%          C       regularization constant
% outputs: alpha   N-dimensional dual solution vector

dim=length(K);

%variables for quadratic program
%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
%             u : N-column vector
%
%             x : N-column vector
%             y : Objective value
%ERROR: chol works not propperly on very slow (close to zero) data
%c=diag(K)';
%H=-K;
%b=1;
%A=ones(dim,1);
%l=zeros(dim,1);
%u=ones(dim,1)*C;
%[x,y] = pr_loqo2(c, H, A, b, l, u);
%alpha = x;

H=-K;
f=diag(K);
l=zeros(dim,1);
u=ones(dim,1)*C;
Aeq = ones(dim,1)';
beq = ones(1);
alpha = quadprog(H,f,[],[],Aeq,beq,l,u);
```

pr_loqo2 and quadprog both solve minimization problems, yet we have formulated a maximization problem. -> solvers wont converge, ever (problems are concave, thus become unconstrained)

f= -diag(K)

H = 2K. we have tu turn it around, because of this

10 / 20 , for correct constraints.

pr_loqo2.m

```
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
```

```
%                u : N-column vector
%
%                x : N-column vector
%                y : Objective value
%
%for a documentation see R. Vanderbei, LOQO: an Interior Point Code
%                          for Quadratic Programming
margin = 0.05;
bound  = 100;
sigfig_max = 8;
counter_max = 50;
[m, n] = size(A);
H_x     = H;
H_diag = diag(H);
b_plus_1 = 1;
c_plus_1 = norm(c) + 1;
one_x = -ones(n,1);
one_y = -ones(m,1);

for i = 1:n
    H_x(i,i) = H_diag(i) + 1;
end;

eig(H_x)

H_y = eye(m);
c_x = c;
c_y = 0;
R = chol(H_x);
H_Ac = R \ ([A; c_x'] / R)';

H_A = H_Ac(:,1:m);
H_c = H_Ac(:,(m+1):(m+1));

A_H_A = A * H_A; A_H_c = A * H_c;
H_y_tmp = (A_H_A + H_y); y = H_y_tmp \ (c_y + A_H_c);
x = H_A * y - H_c; g = max(abs(x - l), bound);
z = max(abs(x), bound); t = max(abs(u - x), bound);
s = max(abs(x), bound); mu = (z' * g + s' * t)/(2 * n);
sigfig = 0; counter = 0; alfa = 1;
while ((sigfig < sigfig_max) * (counter < counter_max)),
  counter = counter + 1; H_dot_x = H * x;
  rho = - A * x + b; nu = l - x + g; tau = u - x - t;
  sigma = c - A' * y - z + s + H_dot_x;
  gamma_z = - z; gamma_s = - s;
  x_dot_H_dot_x = x' * H_dot_x;
  primal_infeasibility = norm([tau; nu]) / b_plus_1;
  dual_infeasibility = norm([sigma]) / c_plus_1;
  primal_obj = c' * x + 0.5 * x_dot_H_dot_x;
  dual_obj = - 0.5 * x_dot_H_dot_x + l' * z - u' * s + b'*y; %%%
  old_sigfig = sigfig;
  sigfig = max(-log10(abs(primal_obj - dual_obj)/(abs(primal_obj) + 1)), 0);
  hat_nu = nu + g .* gamma_z ./ z; hat_tau = tau - t .* gamma_s ./ s;
  d = z ./ g + s ./ t;
  for i = 1:n H_x(i,i) = H_diag(i) + d(i); end;
  H_y = 0;  c_x = sigma - z .* hat_nu ./ g - s .* hat_tau ./ t;
  c_y = rho; R = chol(H_x); H_Ac = R \ ([A; c_x'] / R)';
```

```
  H_A = H_Ac(:,1:m); H_c = H_Ac(:,(m+1):(m+1));
  A_H_A = A * H_A; A_H_c = A * H_c; H_y_tmp = (A_H_A + H_y);
  delta_y = H_y_tmp \ (c_y + A_H_c); delta_x = H_A * delta_y - H_c;
  delta_s = s .* (delta_x - hat_tau) ./ t;
  delta_z = z .* (hat_nu - delta_x) ./ g;
  delta_g = g .* (gamma_z - delta_z) ./ z;
  delta_t = t .* (gamma_s - delta_s) ./ s;
  gamma_z = mu ./ g - z - delta_z .* delta_g ./ g;
  gamma_s = mu ./ t - s - delta_s .* delta_t ./ t;
  hat_nu = nu + g .* gamma_z ./ z;
  hat_tau = tau - t .* gamma_s ./ s;
  c_x = sigma - z .* hat_nu ./ g - s .* hat_tau ./ t;
  c_y = rho; H_Ac = R \ ([A; c_x'] / R)';
  H_A = H_Ac(:,1:m); H_c = H_Ac(:,(m+1):(m+1));
  A_H_A = A * H_A; A_H_c = A * H_c;
  H_y_tmp = (A_H_A + H_y); delta_y = H_y_tmp \ (c_y + A_H_c);
  delta_x = H_A * delta_y - H_c; delta_s = s .* (delta_x - hat_tau) ./ t;
  delta_z = z .* (hat_nu - delta_x) ./ g;
  delta_g = g .* (gamma_z - delta_z) ./ z;
  delta_t = t .* (gamma_s - delta_s) ./ s;
  alfa = - 0.95 / min([delta_g ./ g; delta_t ./ t;
                       delta_z ./ z; delta_s ./ s; -1]);
  mu = (z' * g + s' * t)/(2 * n);
  mu = mu * ((alfa - 1) / (alfa + 10))^2;
  x = x + delta_x * alfa; g = g + delta_g * alfa;
  t = t + delta_t * alfa; y = y + delta_y * alfa;
  z = z + delta_z * alfa; s = s + delta_s * alfa;
end
```

test_ocsvm.m

```
function [ ] = test_ocsvm( predicted_attacks )

load('full-data.mat')
At = (Yts' > 0.5);
true_attacks = find(At)'
missing_attacks = setdiff(true_attacks, predicted_attacks)
false_positives = setdiff(predicted_attacks, true_attacks)

end
```

Machine learning 2

Exercise sheet 5

FLEISCHMANN Kay, Matrnr: 352247
ROHRMANN Till, Matrnr: 343756

May 20, 2013

40 + 20 + 10 = 70

1 One-class-SVM: Theory

(a) Derive the dual program for the one-class SVM.

Primal form

The primal form of the one-class SVM has the following form:

$$\min_{\boldsymbol{\mu},r,\boldsymbol{\xi}} r^2 + C\sum_{i=1}^{N}\xi_i$$

such that

$$\begin{aligned}\|\phi(x_i) - \boldsymbol{\mu}\|^2 &\leq r^2 + \xi_i \\ \xi_i &\geq 0\end{aligned}$$

for $i = 1, \ldots, n$. Using Lagrange multipliers gives us the unconstrained form:

$$\min_{\boldsymbol{\mu},r,\boldsymbol{\xi}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}\geq 0} \underbrace{\left\{ r^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\alpha_i\left(\|\phi(x_i) - \boldsymbol{\mu}\|^2 - r^2 - \xi_i\right) - \sum_{i=1}^{N}\beta_i\xi_i \right\}}_{L(\boldsymbol{\mu},r,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}$$

The dual optimization problem is now given by

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}\geq 0} g(\boldsymbol{\alpha},\boldsymbol{\beta})$$

with g being defined by

$$g(\boldsymbol{\alpha},\boldsymbol{\beta}) = \min_{\boldsymbol{\mu},r,\boldsymbol{\xi}} L(\boldsymbol{\mu}, r, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{1}$$

To compute the minimum of L w.r.t. $\boldsymbol{\mu}, r$ and $\boldsymbol{\xi}$ we take the partial derivative and set it afterwards to zero.

$$\begin{aligned}\nabla_{\boldsymbol{\mu}} L &= \nabla_{\boldsymbol{\mu}}\left(\sum_{i=1}^{N}\alpha_i\left(\phi(x_i) - \boldsymbol{\mu}\right)^T\left(\phi(x_i) - \boldsymbol{\mu}\right)\right) \\ &= \sum_{i=1}^{N}\alpha_i\left(2\boldsymbol{\mu} - 2\phi(x_i)\right) \qquad (2) \\ \frac{\partial L}{\partial r} &= 2r - 2\sum_{i=1}^{N}\alpha_i r \qquad (3) \\ \frac{\partial L}{\partial \xi_j} &= C - \alpha_j - \beta_j \qquad (4)\end{aligned}$$

Setting equations (2),(3) and (4) to 0 we obtain

$$\begin{aligned}\boldsymbol{\mu}\sum_{i=1}^{N}\alpha_i &= \sum_{i=1}^{N}\alpha_i\phi(x_i) \qquad (5) \\ (1 - \sum_{i=1}^{N}\alpha_i)r &= 0 \qquad (6) \\ C &= \alpha_i + \beta_i \qquad (7)\end{aligned}$$

Assuming that we have at least 2 distinct data points, we know that $r > 0$ holds. Thus equation (6) gives us

$$\sum_{i=1}^{N} \alpha_i = 1 \tag{8}$$

and thus equation (5) can be expressed by

$$\boldsymbol{\mu} = \sum_{i=1}^{N} \alpha_i \phi(x_i) \tag{9}$$

This equation says that one can express the optimal solution for $\boldsymbol{\mu}$ as a linear combination of the data points in feature space. Plugging equations (7) and (9) into equation (1) gives us

$$\begin{aligned} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= r^2 + \sum_{i=1}^{N} (\alpha_i + \beta_i)\xi_i + \sum_{i=1}^{N} \alpha_i \left(\|\phi(x_i) - \sum_{i=1}^{N} \alpha_i \phi(x_i)\|^2 - r^2 - \xi_i \right) - \sum_{i=1}^{N} \beta_i \xi_i \\ &= r^2 - r^2 \sum_{i=1}^{N} \alpha_i + \sum_{i=1}^{N} \alpha_i \left(\phi(x_i) - \sum_{j=1}^{N} \alpha_j \phi(x_j) \right)^T \left(\phi(x_i) - \sum_{j=1}^{N} \alpha_j \phi(x_j) \right) \end{aligned}$$

Using equation (8) gives us

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x_i) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \phi(x_i)^T \phi(x_j)$$

with the additional constraints

$$\begin{aligned} \sum_{i=1}^{N} \alpha_i &= 1 \\ C = \alpha_i + \beta_i &\Rightarrow 0 \leq \alpha_i \leq C \end{aligned}$$

Assuming we have a kernel function k expressing the inner product $\phi(x)^T \phi(y) = k(x, y)$ we finally end up at the final formulation:

$$\max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{N} \alpha_i k(x_i, x_i) - \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \right\} \tag{10}$$

subject to

well done!
30 / 30

$$\begin{aligned} \sum_{i=1}^{N} \alpha_i &= 1 \\ 0 \leq \alpha_i &\leq C \text{ with } i = 1, \dots, n \end{aligned}$$

(b) Show that the dual problem is a linearly constrained quadratic problem.

Setting $(\boldsymbol{b})_i = k(x_i, x_i)$ and $(A)_{i,j} = -k(x_i, x_j)$ we can reformulate equation (10) in its matrix/vector notation

$$(10) \quad = \quad \max_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T A \boldsymbol{\alpha} + \boldsymbol{b}^T \boldsymbol{\alpha}$$

Furthermore by setting $v = 1$, $\boldsymbol{u} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$, $l_i = 0$ and $m_i = C$ for $i = 1, \ldots, n$ we can rewrite the constraints:

$$\sum_{i=1}^{N} \alpha_i = 1 \quad \Leftrightarrow \quad \boldsymbol{u}^T \boldsymbol{\alpha} = v$$
$$0 \leq \alpha_i \leq C \quad \Leftrightarrow \quad l_i \leq \alpha_i \leq m_i$$

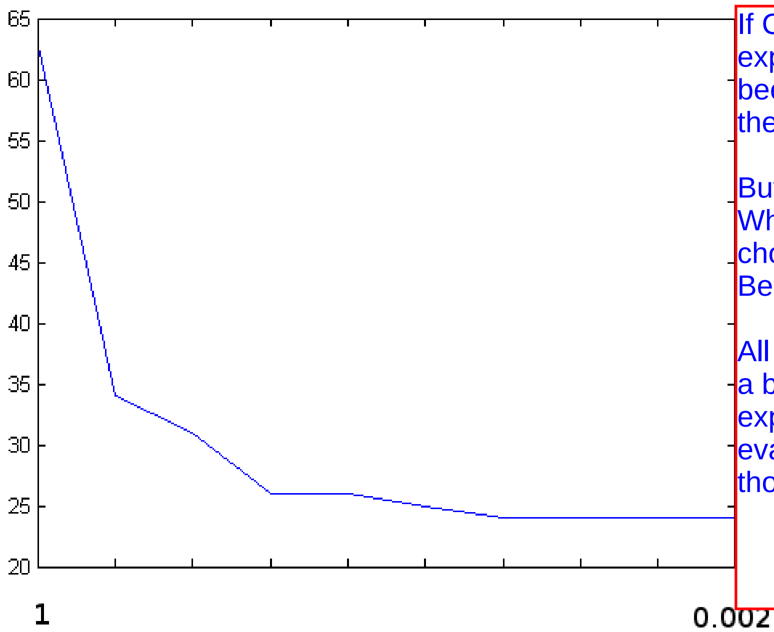
10 / 10

2 Implementation

see attached matlab implementation.
*pr_loqo*2 is running into small value (close to zero) issues. Tested with 64Bit/Win7/(Matlab 2012 b/2013a). quadprog() worked instead.

3 Result of the One-class SVM vs. hackers

With the help of the slack-variables ξ_i the One-class-SVM try to fit best on the normal data in order to find the anormal hacker activities. Using an appropriate value of C is important to distinguish hacker activities from normal ones. The following plot shows the number of hacker activities found if C ist changed, starting with $C = 1$ and halved in each step.



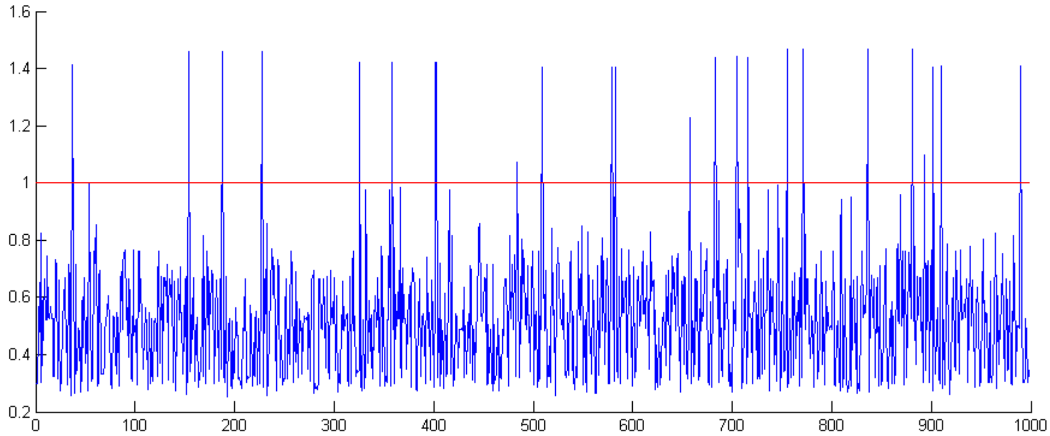
If C = 0.002 is interesting, maybe a bit exploration around that value would have been nice. In your plot,it is at the end of the evaluated spectrum.

But Why is this value of C appropriate? What are your observations, how do you choose it, based on what reasoning? Because it is the last value you tried?

All in all the evaluation of this problem is a bit lacking. 10 / 30 for performing experiments. 20 points missing for no evaluation of documentation of your thought process.

Maybe an appropriate value for may $C = 0.002$. This value is applied to test data to find hacker activities on event-logs.

Hacker atttacks found



Explicit position of hacker attacks

37 154 188 228 326 358 402 403 484 509 579 583 658 683 705 716 755 771 836 881 893 902 910 990