Statistical Machine Learning : SVM [Assignment #01]

by José Ananías Hilario Reyes

Contents

Task #1

```
load('dns_data_kernel.mat');
C = [0.01, 0.1, 1, 10, 100];
s = struct('Parameters',0,'nr_class',0,'totalSV',0,'rho',0,'Label',0,'sv_indices',0,'ProbA',0,'ProbB',0,'nSV',0,'sv_coef',0,'SVs',0);

Svm_trn = repmat(s, 1, length(C));
m_trn = length(Trn.Y);
m_val = length(Val.Y);
bias_trn = zeros(size(C));
alpha_trn = zeros(length(C), m_trn);
nSV_trn = zeros(size(C));

scores_trn = zeros(length(C), m_trn);
predY_trn = zeros(length(C), m_trn);
trnErr_trn = zeros(size(C));

scores_val = zeros(length(C), m_val);
predY_val = zeros(length(C), m_val);
trnErr_val = zeros(size(C));

t_columns = zeros(length(C), 3);
for i = 1:length(C)
    m = length(Trn.Y);
    Svm_trn(i) = svmtrain(Trn.Y, [[1:m]' Trn.K], ['-s 0 -t 4 -c ' num2str(C(i))]);

    bias_trn(i)       = -Svm_trn(i).rho;
    a                 = zeros(m, 1);
    a(Svm_trn(i).SVs) = Svm_trn(i).sv_coef(:);
    alpha_trn(i,:)    = a';
    nSV_trn(i) = Svm_trn(i).totalSV;

    % Trn
    scores_trn(i,:) = (Trn.K*alpha_trn(i,:)' + bias_trn(i))';
    predY_trn(i,:) = 2*double(scores_trn(i,:) >= 0) - 1;
    trnErr_trn(i) = mean(predY_trn(i,:)' ~= Trn.Y(:));

    % Val
    scores_val(i,:) = (Val.K*alpha_trn(i,:)' + bias_trn(i))';
    predY_val(i,:) = 2*double(scores_val(i,:) >= 0) - 1;
    trnErr_val(i) = mean(predY_val(i,:)' ~= Val.Y(:));

    % Result
    t_columns(i,:) = [trnErr_trn(i) trnErr_val(i) nSV_trn(i)];
end

t = cell2table(cell(3,length(C)), 'VariableNames', {'C_001', 'C_01', 'C_1', 'C_10', 'C_100'});
t.Properties.RowNames = {'R_trn', 'R_val', 'nSV'};
t.C_001 = t_columns(1,:)';
t.C_01 = t_columns(2,:)';
t.C_1 = t_columns(3,:)';
t.C_10 = t_columns(4,:)';
t.C_100 = t_columns(5,:)';
t
writetable(t, 'errors_vs_c.xls');

f = figure;
plot(C, trnErr_trn);
hold on;
plot(C, trnErr_val);
title('Error vs C');
xlabel('Regularization Constant [C]');
ylabel('Error');
legend('Training Error', 'Validation Error');
set(gca,'xscale','log');
saveas(f, 'errors_vs_c.png');
```

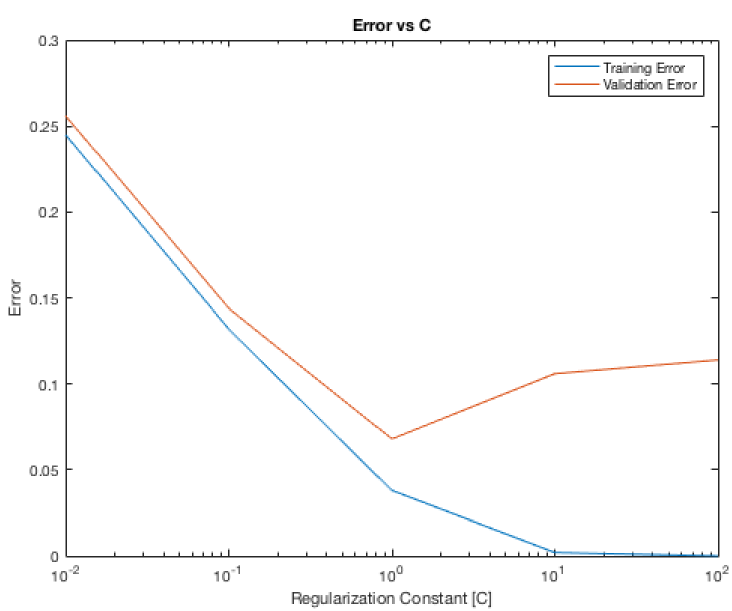
```
*
optimization finished, #iter = 500
nu = 1.000000
obj = -9.583942, rho = -0.143882
nSV = 1000, nBSV = 1000
Total nSV = 1000
*
optimization finished, #iter = 501
nu = 0.856651
obj = -67.442877, rho = -0.491564
nSV = 876, nBSV = 834
Total nSV = 876
*.*
optimization finished, #iter = 1360
nu = 0.424782
obj = -288.313362, rho = -0.124357
nSV = 543, nBSV = 305
Total nSV = 543
..*..*
optimization finished, #iter = 4689
nu = 0.120007
obj = -628.558518, rho = -0.021492
nSV = 386, nBSV = 14
Total nSV = 386
..*...*
optimization finished, #iter = 5293
nu = 0.012860
obj = -643.027774, rho = -0.019940
nSV = 376, nBSV = 0
Total nSV = 376

t =

  3×5 table

             C_001    C_01     C_1      C_10     C_100
             _____    _____    _____    _____    _____

    R_trn    0.245    0.132    0.038    0.002        0
    R_val    0.256    0.144    0.068    0.106    0.114
    nSV       1000      876      543      386      376
```



Task #2

Since we have a test set that is independent from training set, we can use Hoeffding

```
min = inf;
for i = 1:length(C)
    if min > t_columns(i,2)
        min = t_columns(i,2);
        bestC = C(i);
```

```
        alpha = alpha_trn(i,:);
        weights = Svm_trn(i).sv_coef;
        bias = bias_trn(i);
    end
end
bestC

scores_tst = (Tst.K*alpha' + bias)';
predY_tst = 2*double(scores_tst >= 0) - 1;
R_tst = mean(predY_tst' ~= Tst.Y(:))
m = length(Tst.Y);

conf = 0.99;
eps = sqrt((log(2) - log(1 - conf)) / (2 * m))
R_true = [R_tst - eps, R_tst + eps]
% In case we don't have independent test samples and in turn you need to account for infinite number of hypothesis
%  - Kernel defines Hilbert space, but this does it affect VC dimension?
%  - Turns out it does. Growth function approx is bad upper bound.
%  - Instead use margin to calculate VC dimension.
%  - margin = 2 / norm(weights);
%  - dvc = 1 + 1/margin^2; % Features generated by similarities between strings (kernel)
%  - m = length(Trn.Y);
%  - delta = 1 - conf;
%  - eps = sqrt(8*(dvc * log(2 * m * exp(1) / dvc) + log(4 / delta)) / m);
```

```
bestC =

     1


R_tst =

    0.0785


eps =

    0.0364


R_true =

    0.0421    0.1149
```

Task #3

```
urls = [{'google.com'}, {'facebook.com'}, {'atqgkfauhuaufm.com'}, {'vopydum.com'}];
kernel = zeros(length(urls));
normalized_kernel = zeros(length(urls));
lambda = 0.4;
q = 3;
for i = 1:length(urls)
    for j = 1:length(urls)
        kernel(i,j) = subseq_kernel(char(urls(i)), char(urls(j)), q, lambda);
    end
end
for i = 1:length(urls)
    for j = 1:length(urls)
        normalized_kernel(i,j) = kernel(i,j)/(sqrt(kernel(i,i))*sqrt(kernel(j,j)));
    end
end
k_table = array2table(kernel, 'VariableNames', {'google_com', 'facebook_com', 'atqgkfauhuaufm_com', 'vopydum_com'});
k_table.Properties.RowNames = {'google_com', 'facebook_com', 'atqgkfauhuaufm_com', 'vopydum_com'}
writetable(k_table, 'subsequence_string_kernel.xls');

nk_table = array2table(normalized_kernel, 'VariableNames', {'google_com', 'facebook_com', 'atqgkfauhuaufm_com', 'vopydum_com'});
nk_table.Properties.RowNames = {'google_com', 'facebook_com', 'atqgkfauhuaufm_com', 'vopydum_com'}
writetable(nk_table, 'subsequence_string_normalized_kernel.xls');
```

```
k_table =

  4×4 table

                          google_com    facebook_com    atqgkfauhuaufm_com    vopydum_com
                          __________    ____________    __________________    ___________
```

```
    google_com            6.8233        2.4428         1.5209              1.681
    facebook_com          2.4428        21.115         2.5464             1.8473
    atqgkfauhuaufm_com    1.5209        2.5464         168.25             2.0194
    vopydum_com            1.681        1.8473         2.0194             8.6267


nk_table =

  4×4 table

                         google_com    facebook_com    atqgkfauhuaufm_com    vopydum_com
                         __________    ____________    __________________    ___________

    google_com                   1       0.20351        0.044888               0.21911
    facebook_com           0.20351             1        0.042722               0.13687
    atqgkfauhuaufm_com    0.044888      0.042722               1              0.053005
    vopydum_com            0.21911       0.13687        0.053005                     1
```

Published with MATLAB® R2017b