Problem 1.

(a) Find the equation of the hyperplane (in terms of w) WITHOUT solving a quadratic programming problem. Make a sketch of the problem.

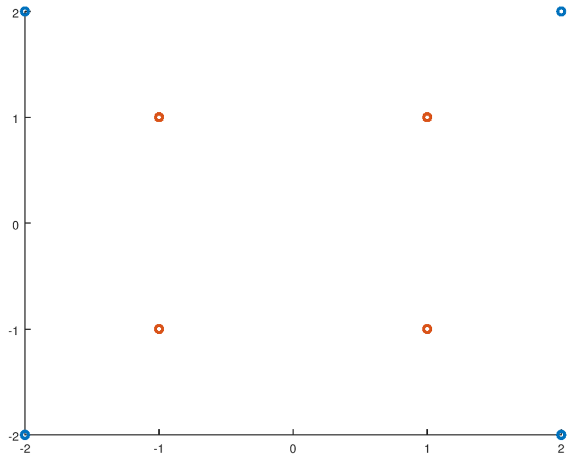
(b) Calculate the margin.

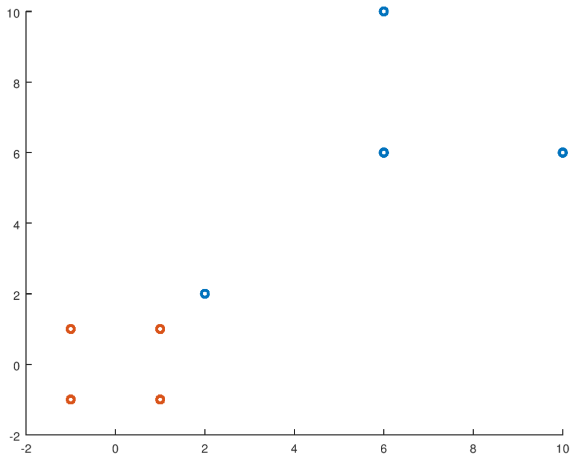
(c) Find the α's of the SVM for classification.

(d) Perform SVM with Matlab or R with RBF kernel and plot the result. Is this result different from the one you obtained by hand?

Solution

Part (a)


(a) Original Data


(b) Transformed Data

From this transformed data, we can clearly see our two support vectors (one for each class). For the orange class (-1) this is at point [1,1] and for the blue class (+1) this is at point [2,2]. SVM requires that all points be on their respective "positive" sides of the hyperplane, thus we can see that this hyperplane must pass between the two support vectors we have chosen. This means, that if we imagine a line segment connecting connecting the points {[1,1], [2,2]} our hyperplane MUST intersect that line segment. Further, as these points are support vectors, they must lie at equal distance to the hyperplane, from these observations we conclude that the hyperplane will intersect these two points at their midpoint, this will be the point $\frac{1}{2}[1+2, 1+2] = [\frac{3}{2}, \frac{3}{2}]$.
As our transformed data exists in two dimensions, our hyperplane will take the form of a line, as we know an intercept of this line, the only thing that remains to be found is the

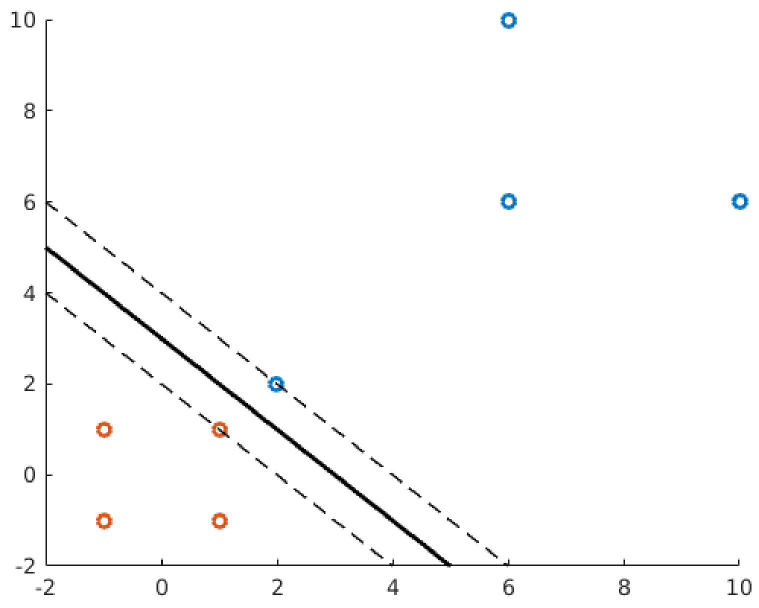
slope of the line, once we have that we can derive an equation for this line in the desired form.
Intuitively we would expect the hyperplane to be perpendicular to our imaginary line segment connecting our two support vectors. To see why this must be the case, simply consider the case where it is not. At one extreme, our hyperplane is in line with both support vectors, producing a margin equal to 0, as we rotate our line, this margin will increase in size. It will continue to increase in size until it reaches the perpendicular we are considering, if we continue to rotate beyond the perpendicular, each edge will begin to approach the opposite class, thus resulting in a decrease again in our margin. Thus maximum margin is achieved at the perpendicular. As the line segment through {[1,1],[2,2]} has slope 1, this implies our hyperplane has slope -1.
In standard form this means our line will take the form:

$$ax + b = -1x + 3$$

Some simple algebra will produce the vectorized form:

$$\begin{aligned} -x + 3 &= y \\ x + y - 3 &= 0 \\ [1; 1]^T [x; y] - 3 &= 0 \end{aligned}$$

From this we have w = [1, 1] and b = -3.
To check our work we need to consider our class labels (else we would simply multiply both sides by -1), we expect $w'[2; 2] + b = 1$ and $w'[1; 1] + b = -1$. Plugging these values into matlab produces the expected results so I'll assume I did my work correctly.



Part (b)

The margin will simply be the distance between my two support vectors.

$$\begin{aligned} dist([1,1],[2,2]) &= sqrt((2-1)^2 + (2-1)^2) \\ &= sqrt(2(1)^2) \\ &= sqrt(2) \end{aligned}$$

Thus we find we have a margin of $sqrt(2)$.

Part (c)

Note that the majority of our points are not support vectors, for all of these points we have that $\alpha = 0$. Note that our support vectors are observation 1 and 5, thus we still need to find α_1 and α_5. From my notes we have that $w^\intercal = \Sigma_{i=1}^{m} \alpha_i Y_i x_i^\intercal$, noting that each class only has one support vector we can simplify this as follows

$$w^\intercal = \alpha_1 * [2;2]^\intercal + -\alpha_5 * [1;1]^\intercal$$

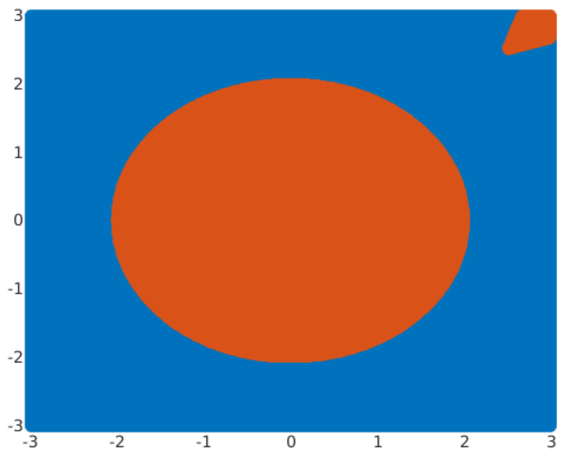
This is a 2x2 linear system, format it as such.

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_5 \end{bmatrix}$$

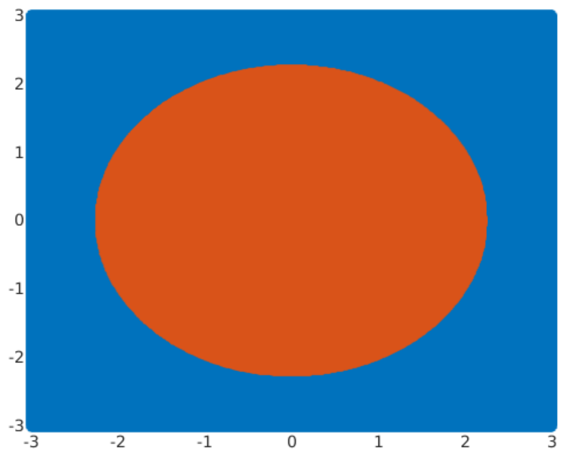
From here it is pretty obvious that $\alpha_5 = \alpha_1$ denote this δ and we have
$1 = 2\delta - 1\delta = (2-1)\delta = \delta$. Thus $\delta = 1$.

$$\begin{aligned} \alpha_1 &= 1 \\ \alpha_5 &= 1 \\ \alpha_i &= 0 \text{ otherwise} \end{aligned}$$

Part (d)



(c) By Hand



(d) Matlab

As shown in the figure the classifier computed by hand appears to misclassify points in the top right corner, I only had time to to check each classifier within the bounds [-3,3] on each axis, so I don't know if this trend would continue if a larger bound was used. By performing some simple arithmetic (see code below), given a -step of 0.005 over the given bounds, the hand written classifier disagreed with matlabs svm classifier 7.772% of the time.
Despite this difference, the results performed by hand are very close to the result produced by matlab.

```
function p1

clear all;
close all;
hold on;

%% Data

data = [
    % Class 1
    2, 2, 1;
    2, -2, 1;
    -2, -2, 1;
    -2, 2, 1;
    % Class -1
    1, 1, -1;
    1, -1, -1;
    -1, -1, -1;
    -1, 1, -1
];

%% Manual SVM

axis([-2 10 -2 10]);

classA = data(data(:,3)==1, 1:2);
classB = data(data(:,3)==-1, 1:2);

classA = phi(classA);
classB = phi(classB);

scatter(classA(:,1), classA(:,2), 'LineWidth', 2)
scatter(classB(:,1), classB(:,2), 'LineWidth', 2)

% Plot the decision boundary.
x = [-2, 10];
y = -1 * x + 3;
[1; 1]' * x' - 3
plot(x, y, 'k', 'LineWidth', 2);


% Plot the margins.
y1 = -1 * x + 3 + 1;
y2 = -1 * x + 3 - 1;
```

```
plot(x, y1, '--k', 'LineWidth', 1);
plot(x, y2, '--k', 'LineWidth', 1);

%% Matlab SVM

X = data(:,1:2);
Y = data(:,3);
mdl = fitcsvm(X, Y, 'KernelFunction', 'rbf', 'OptimizeHyperparameters', 'auto',...
    'HyperparameterOptimizationOptions', struct('Optimizer', 'bayesopt', 'Kfold', 10));

%% Comparison

% Generate a grid of data on each axis between [-3,3]
delta = 0.005;
[XX, YY] = meshgrid(-3:delta:3, -3:delta:3);
X = [reshape(XX, numel(XX), 1) reshape(YY, numel(YY), 1)];

% Make a prediction using Matlabs model.
LABELS0 = predict(mdl, X);
LABELS1 = p1predict(X);

% If they Agree:
% 1 - 1 = 0
%  or
% -1 - -1 = -1 + 1 = 0
% But if they Disagree:
% 1 - -1 = 2 (divide by 2 = 1)
%  or
% -1 - 1 = -2 (divide by 2 = -1, abs() for 1)
% Thus the sum is the number of labels my classifier and matlab disagree on.
DISAGREE = sum(abs(LABELS0 - LABELS1) ./ 2);
DISAGREE / length(X) % 0.0772

% classA = X(LABELS(:,1)==1, :);
% classB = X(LABELS(:,1)==-1, :);

% scatter(classA(:,1), classA(:,2), 'LineWidth', 2)
% scatter(classB(:,1), classB(:,2), 'LineWidth', 2)

end
```

```
function [out] = phi(data)

x1 = data(:,1);
```

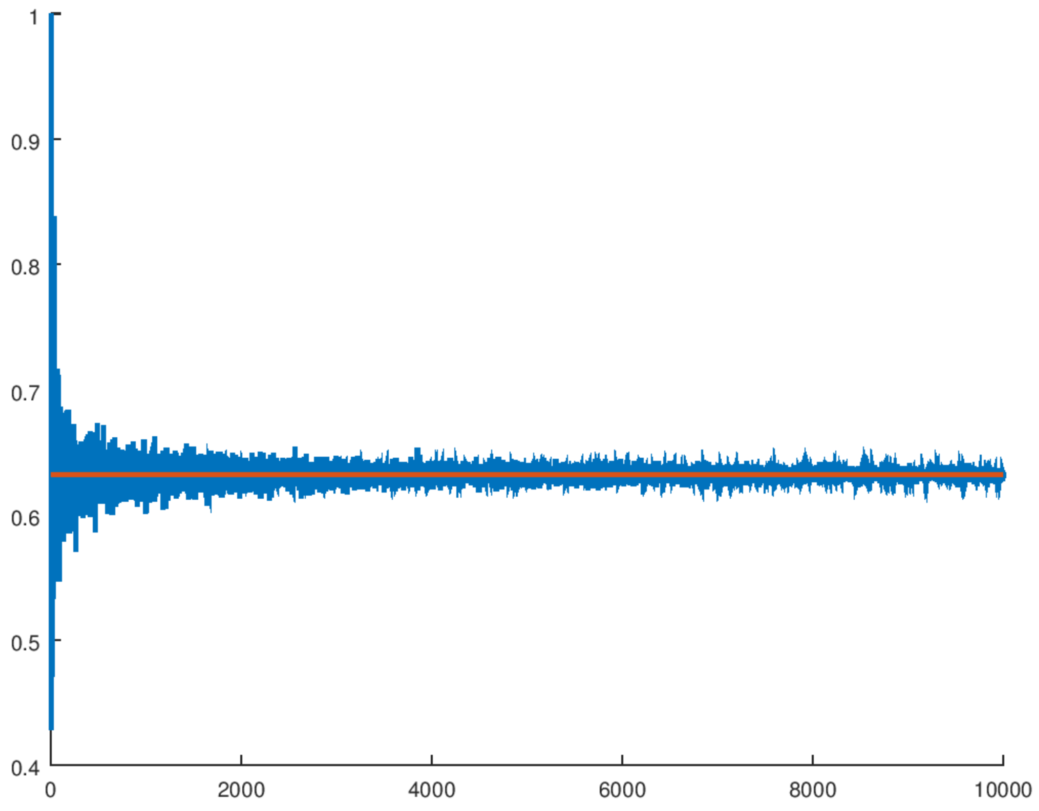
```
x2 = data(:,2);

if sqrt(x1.^2 + x2.^2) > 2
    out = [ 4 - x2 + abs(x1 - x2), 4 - x1 + abs(x1 - x2) ];
else
    out = [ x1, x2 ];
end

end
```

```
function [Y] = p1predict(X)

w = [1; 1];
b = -3;
C = length(X);
Y = zeros(C, 1);

for i = 1:C
    Y(i, 1) = sign(w' * phi(X(i,:))' + b);
end

end
```

Problem 2.

Verify by means of simulation that each bootstrap sample will contain 1 - 1/e ≈ 63.2 of the original sample.

Solution



The graph mostly speaks for itself, the orange line denotes 1 - 1/e, and the blue denotes each iteration of bootstrapping. As the number of iterations increases, the percentage the bootstrap contains of the original sample approaches the orange line (1 - 1/e). Even at 10000 iterations there appears to be an impact of the random values, I would imagine as we continued to increase the number of iterations this would taper off significantly. Again, code is provided below.

```matlab
function p2

clear all;
close all;
hold on;

M = 10000; % How many simulations are we going to run?
ans = zeros(M,1); % This should approach 1 - 1/e.
actual = (1 - 1/e); % The actual result we are expecting.

% Run M bootstrap samples.
for n = 1:M
    % Random normal data set of size 'n'.
    X = randn(n, 1);

    % Generate random indecies in the range [1,n].
    idx = ceil(rand(n,1)*n);

    % The number of samples contained in X is equal
    % to the number of UNIQUE indecies in idx.
    % To get the fraction simply divide by n.
    ans(n) = length(unique(idx)) / n;
end

plot(1:M, ans, 'LineWidth', 2);
plot([1 M], [actual actual], 'LineWidth', 2);

end
```

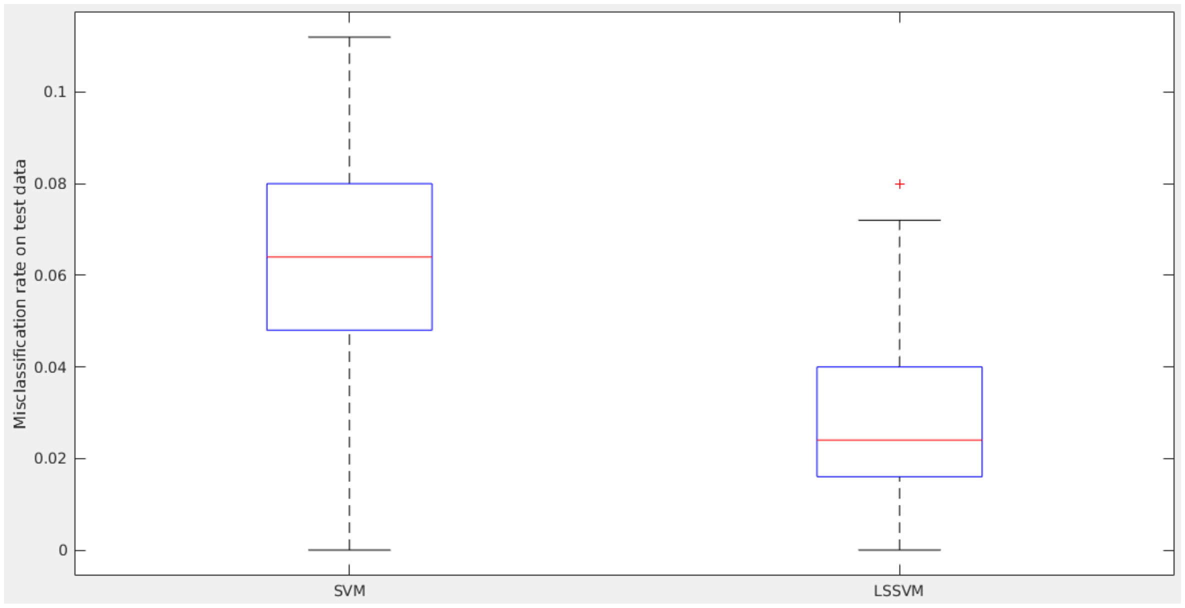
Problem 3.

Compare SVM and LS-SVM on the ozon level detection data set on Blackboard. This dataset contains 72 measurement variables and was measured between 1/1/1998 and 12/31/2004. All missing values have been removed. The goal is to detect whether there was too much ozon (class label 0) or a normal day (class label 1). The class label is the last variable. Set up the simulation and clearly describe what you are doing and why. Finally, state, according to your findings (boxplots, ROC curves, etc.), the best classifier for this problem.

Solution

I ran svm and lssvm using the available kernel functions, then for each run usdross validation to estimate the misclassification rate. In general, svm took much longer than lssvm, and I could not get matlab to use the polynomial kernel on the data at all (see comments for full details). The best result produced via svm was via the rbf kernenl, which had a misclassification rate of 6.71% and took 148.5 seconds. In contrast each kernel worked for lssvm and took roughly half as long as their svm counterparts. The linear kernel of lssvm performed worse than any of the the linear and rbf kernels of svm, but the rbf kernel for lssvm performed the best out of all options I tested with a misclassification rate of 5.96%. In terms of computation time, rbf-lssvm was slower than other lssvm kernels, but still plenty fast compared to the svm kernels (taking 75.9 seconds).

Seeing as the rbf kernel seemed to be performing best for both, I generated a box plot to get a closer look at their performance. For sake of computation time, these were found by using a random subset of 500 samples for each of my 100 iterations (0.75/0.25 training/test split, see code for details). With the exception of where they were lucky enough to have roughly 0% misclassification error, we can clearly see from the box plot that lssvm consistently performs better than the svm classifier.



```
function p3

clear all;
close all;
hold on

%% Read the data from the given spreadsheet.
data = xlsread('ozon.xlsx');

X = data(:,1:72); % All of our data, but no labels (1874 Entries).
Y = data(:,73);          % Labels corresponding to each entry in X.

%% SVM

tic;

% Kernel Functions: rbf, linear, or polynomial
mdl = fitcsvm(X, Y, 'KernelFunction', 'rbf', 'OptimizeHyperparameters', 'auto',...
    'HyperparameterOptimizationOptions', struct('Optimizer', 'bayesopt', 'Kfold', 10));
% linear and polynomial kernels weren't working with the extra parameters
% but it worked after removing them
% mdl = fitcsvm(X, Y, 'KernelFunction', 'linear');

% Get our misclassification rate estimate through cross validation.
kfoldLoss(crossval(mdl))
toc;

% rbf                  0.0671 # 148.50 seconds
% linear        0.0693 # 334.09 seconds
% polynomial         NaN    # 2.0702 seconds

%% LSSVM

tic;

% Kernel Functions: lin_kernel, poly_kernel, RBF_kernel
mdl = initlssvm(X, Y, 'c', [], [], 'poly_kernel', 'p');
mdl = tunelssvm(mdl, 'simplex', 'crossvalidatelssvm', {10, 'misclass'});
mdl = trainlssvm(mdl);
% Get our misclassification rate estimate through cross validation.
crossvalidate(mdl, 10, 'misclass')

toc;
```

```
% lin_kernel        0.0693 # 48.76 seconds
% poly_kernel        0.1012 # 43.01 seconds
% RBF_kernel        0.0596 # 75.90 seconds

%% Box Plots

B = 100;
mcrsvm   = zeros(B,1);
mcrlssvm = zeros(B,1);

tic;

% Basically just copy and paste the prof's example code.
for b = 1:B

    % For sake of performance, use a random sample size
    % of 500 (375 Train, 125 Test).
    % We already have an estimate of misclassification error,
    % so we really are only interested in the variance, so this should be fine.

    N = 500; % How big is the random subset we are considering?
    r = randperm(size(X,1)); % Need a random permutation over ALL indecies...
    r = r(1:N); % but only want N when we are all said and done.
    ntr = ceil(0.75 * N); % The first 75% are for training.

    % Break up our data into training...
    Xtr = X(r(1:ntr), :);
    Ytr = Y(r(1:ntr));

    % and test sets.
    Xtest = X(r(ntr+1:N), :);
    Ytest = Y(r(ntr+1:N));

    %%%%%%%
    % SVM %
    %%%%%%%
    mdl = fitcsvm(X, Y, 'KernelFunction', 'rbf', 'OptimizeHyperparameters', 'auto',...
        'HyperparameterOptimizationOptions', struct('Optimizer', 'bayesopt', 'Kfold', 10
    yh = predict(mdl, Xtest);
    mcrsvm(b, 1) = sum(Ytest ~= yh) / size(Ytest, 1);

    %%%%%%%%%
    % LSSVM %
    %%%%%%%%%
    mdl = initlssvm(X, Y, 'c', [], [], 'RBF_kernel', 'p');
    mdl = tunelssvm(mdl, 'simplex', 'crossvalidatelssvm', {10, 'misclass'});
    mdl = trainlssvm(mdl);

    % For whatever reason lssvm is only letting me do one prediction at a time,
    % so take this pain in the ass approach to get our predictions.
    T = size(Xtest, 1);
    yh = zeros(T, 1);
    for idx = 1:T
        yh(idx, 1) = predict(mdl, Xtest(idx, :), 1);
    end

    mcrlssvm(b, 1) = sum(Ytest ~= yh) / size(Ytest, 1);

    close all;
end

toc;

legend = { 'SVM', 'LSSVM' };
boxplot([mcrsvm, mcrlssvm], legend);
ylabel('Misclassification rate on test data');

end
```