Machine Learning HW2
Lilong Jiang (jiang.573)

1. Problem 1

The linear SVM code is shown below:

```
Y = [zeros(1000,1); ones(1000,1)];
load train79.mat;
SVMModel = fitcsvm(d79,
Y,'KernelFunction','linear','Standardize',true,'ClassNames',{'0','1'});
load test79.mat;
[labels,score] = predict(SVMModel,d79);
error = 0;
for i = 1 : 1000
    if(labels{i} ~= '0')
        error = error + 1;
    end
end

for i = 1000 : 2000
    if(labels{i} ~= '1')
        error = error + 1;
    end
end
errorRate = error / 2000
```

The error rate is 0.0685.
For least squares linear classifier, the code is shown below:

```
Y = [ones(1000, 1); -ones(1000, 1)];
load train79.mat;
W = lsqlin(d79, Y);
load test79.mat;
labels = d79 * W;

error = 0;
for i = 1 : 1000
    if labels(i) < 0
        error = error + 1;
    end
end

for i = 1000 : 2000
    if labels(i) > 0
        error = error + 1;
    end
end
errorRate = error / 2000
```

The error rate is 0.0660.
The result shows that least squares linear is slight better than SVM.

2. Problem 2

For training dataset, I randomly sample 10% of data for testing and 90% dataset for training. I varies kernel width from [10, 100, 500, 1000, 1500] and lambda in [0.1, 0.5, 1, 2].

```
load('train79.mat');
```

```
trainData = d79;
load('test79.mat');
testData = d79;

N = size(trainData, 1);
percent = 0.1;

sigmas = [10, 100, 500, 1000, 1500];
lamdas = [0.1, 0.5, 1, 2];
for lidx = 1: size(lamdas, 2)
    lamda = lamdas(lidx);
    disp('lamba:');
    lamda
    for idx = 1: size(sigmas, 2)
        % train
        sigma = sigmas(idx);
        disp('sigma:');
        sigma

        [TrainInd, TestInd] = crossvalind(N, percent);
        trainN = size(TrainInd, 1);
        K = zeros(trainN, trainN);
        I = eye(trainN);
        Y = zeros(trainN, 1);

        for i = 1: trainN
            iIdx = TrainInd(i);
            if iIdx <= 1000
                Y(i) = 1;
            else
                Y(i) = -1;
            end
            for j = 1: trainN
                jIdx = TrainInd(j);
                K(i, j) = GaussKernel(trainData(iIdx,:), trainData(jIdx,:),
sigma);
            end
        end
        W = (K + lamda * I) \ Y;

        % train error
        testN = size(TestInd, 1);
        trainError = 0;
        for i = 1: testN
            iIdx = TestInd(i);
            p = zeros(trainN, 1);
            for j = 1: trainN
                jIdx = TrainInd(j);
                p(j) = GaussKernel(trainData(iIdx,:), trainData(jIdx,: ),
sigma);
            end
            val = W' * p;
            if val < 0
                classifyLab = -1;
            else
                classifyLab = 1;
            end
```

```
            if iIdx <= 1000
                correctLab = 1;
            else
                correctLab = -1;
            end
            if correctLab ~= classifyLab
                trainError = trainError + 1;
            end
        end
        disp('train error:');
        trainErrorRate = trainError / testN

        % test error
        testError = 0;
        for i = 1: N
            p = zeros(trainN, 1);
            for j = 1: trainN
                jIdx = TrainInd(j);
                p(j) = GaussKernel(testData(i,:), trainData(jIdx,:), sigma);
            end
            val = W' * p;
            if val < 0
                classifyLab = -1;
            else
                classifyLab = 1;
            end
            if i <= 1000
                correctLab = 1;
            else
                correctLab = -1;
            end
            if correctLab ~= classifyLab
                testError = testError + 1;
            end
        end
        disp('test error:');
        testErrorRate = testError * 1.0 / N
    end
end
```

Result:

From the following table, it seems that both for training and testing error, it increases with the kernel width under different lamda. In most cases, the testing error is lower than training error.

When kernel width is less than 500, it seems that testing error is always better than linear SVM.

Also lamda = 0.1 seems best in terms of testing error.

Lamda	Kernel Width	Training Error Rate	Testing Error Rate
0.1	10	0.0100	0.0235
0.1	100	0.0450	0.0215
0.1	500	0.0800	0.0450
0.1	1000	0.0550	0.0555
0.1	1500	0.1050	0.0620

0.5	10	0.0200	0.0245
0.5	100	0.0550	0.0295
0.5	500	0.0950	0.0575
0.5	1000	0.1100	0.0645
0.5	1500	0.0750	0.0705
1	10	0.0100	0.0245
1	100	0.0400	0.0345
1	500	0.0750	0.0615
1	1000	0.0850	0.0700
1	1500	0.0800	0.0690
2	10	0.0250	0.0220
2	100	0.0550	0.0420
2	500	0.0400	0.0655
2	1000	0.0800	0.0740
2	1500	0.1400	0.1075

3. Problem 3

Result: There seems no obivious pattern.

Number of Features	**100**	**200**	**300**	**500**	**1000**	**2000**
Error Rate	**0.5040**	**0.5100**	**0.4905**	**0.4930**	**0.5200**	**0.4810**

```
Y = [-ones(1000, 1); ones(1000, 1)];
load('train79.mat');
trainData = d79;
load('test79.mat');
testData = d79;
[N, col] = size(trainData);

% sample
ks = [100, 500, 1000, 2000, 5000];
for i = 1: size(ks, 2)
    k = ks(i)
    mu = zeros(col, 1);
    sigma = eye(col);

    trainzxs = ones(N, 2 * k);
    testzxs = ones(N, 2 * k);

    lambda = 0.1;

    W = mvnrnd(mu, sigma, k);

    for i = 1: N
        trainx = trainData(i, :);
        testx = testData(i, :);
        for j = 1: k
            trainzxs(i, j) = cos(W(j, :) * trainx');
            testzxs(i, j) = cos(W(j, :) * testx');
        end
        for j = k + 1: 2 * k
            trainzxs(i, j) = sin(W(j - k, :) * trainx');
            testzxs(i, j) = sin(W(j - k, :) * testx');
```

```
        end
        trainzxs(i) = sqrt(trainzxs(i) ./ k);
        testzxs(i) = sqrt(testzxs(i) ./ k);
    end

    W = (trainzxs' * trainzxs + eye(2*k) .* lambda) \ trainzxs' * Y;

    error = 0;

    for i = 1: 1000
        classifyVal = testzxs(i, :) * W;
        if classifyVal > 0
            error = error + 1;
        end
    end

    for i = 1001: 2000
        classifyVal = testzxs(i, :) * W;
        if classifyVal < 0
            error = error + 1;
        end
    end

    errorRate = error * 1.0 / 2000
end
```

4. Problem 4

Evaluation Function: An **evaluation function** over the Hilbert space of functions H is a linear functional that evaluates each function in the space at the point t.

Hilbert space: A Hilbert space H is a reproducing kernel Hilbert space (RKHS) if the evaluation functions are bounded.

We can define $g(x) = \begin{cases} C\ when\ x = 0.5 \\ f(x)\ others \end{cases}$, where C is a arbitrary real number. For this case, we cannot bound the evaluation function.

5. Problem 5

Smoothing splines is used to estimate the regression function. It is pretty similar to kernel least square regression. In order to avoid overfitting, it also control coefficients with a regularization term lamda (same idea used in kernel regression). The idea that places knots at all points is similar to kernel method that each point is associated with a function in RKHS. The solution of coefficients is similar to least squares regression.