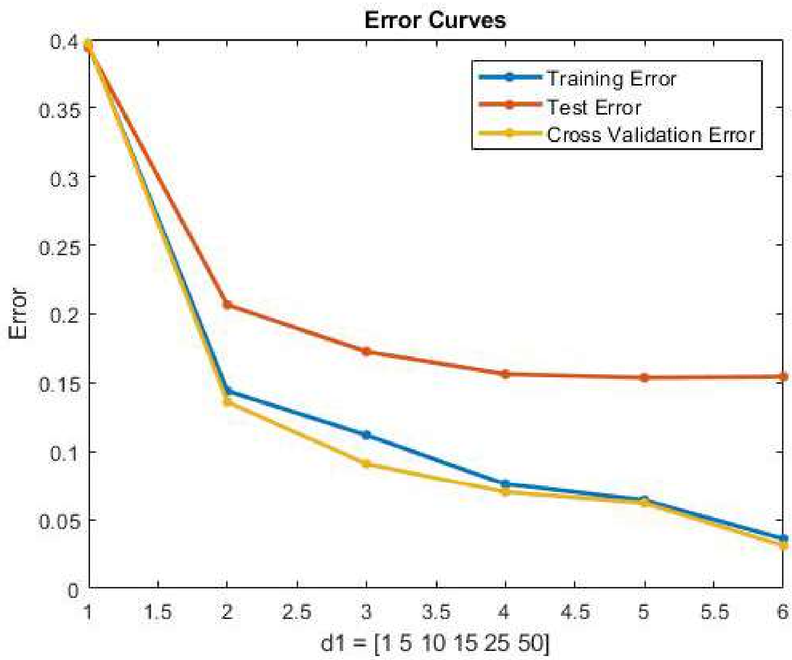
3. Solution to problem 3 goes here

   (a) Solution goes here
   The plot:



The selected d1 is 50. The corresponding training error is 0.0360. The corresponding test error is 0.1544. The corresponding cross validation error is 0.0310.
Matlab Codes:
g.m is the function used to calculate the activation function.

```
function g_z = g(z, activation)
    if activation == "sigmoid"
        g_z = 1./(1+exp(-z));
    elseif activation == "relu"
        g_z = max(0, z);
    end
end
```

Matlab Codes:
dg.m is the function used to calculate the derivative of the activation function.

```
function dg_z = dg(z, activation)
    if activation == "sigmoid"
        dg_z = g(z, activation) .* (1 - g(z, activation));
    elseif activation == "relu"
        dg_z = z > 0;
    end
end
```

z1.m is the function used to calculate $z^{(1)}$.

```
function z = z1(x, w_1, b1)
    z = w_1'*x'+b1';
```

```
end
```

a1.m is the function used to calculate $a^{(1)}$.

```
function a = a1(x, w_1, b1, activation)
    a = g(z1(x, w_1, b1), activation);
end
```

fwb.m is the function used to calculate $f_{w,b}(x)$.

```
function fwbx = fwb(x, w_1, w_2, b1, b2, activation)
    fwbx = transpose(w_2) * g(z1(x, w_1, b1), activation) + b2;
end
```

update_wb.m is the function used to update the weights and biases.

```
function [updated_w_1, updated_b1, updated_w_2, updated_b2] =
    update_wb(x, y, w_1, w_2, b1, b2, eta, activation, task)
    [m, d] = size(x);
    [~, d1] = size(w_1);
    coefficient = 1;

    if task == "regression"
        coefficient = 2;
    end

    common_term = transpose(g(fwb(x, w_1, w_2, b1, b2, activation),
        activation)) - y;
    db2 = coefficient * mean(transpose(common_term), 2);

    dw_2 = coefficient * mean(transpose(repmat(common_term, 1, d1) .*
        transpose(a1(x, w_1, b1, activation))), 2);

    new_term = repmat(common_term, 1, d1) .* transpose(repmat(w_2, 1,
        m) .* dg(z1(x, w_1, b1), activation));
    db1 = coefficient * mean(new_term, 1);

    dw_1 = zeros(d1, d);
    for i = 1:m
        dw_1 = dw_1 + transpose(new_term(i, :))*x(i, :)/m;
    end
    dw_1 = coefficient * dw_1';

    updated_w_1 = w_1 - eta*dw_1;
    updated_b1 = b1 - eta*db1;
    updated_w_2 = w_2 - eta*dw_2;
    updated_b2 = b2 - eta*db2;
end
```

ps2q3.m is the functions that does the majority of the rest of the work, including data collection, plot, parameter selection, etc.

```
%Problem Set 2 Qurstion 3
%number of nodes
d1s = [1 5 10 15 25 50];
%step size
eta = 0.1;
%number of iterations
N = 5000;
%activation function
activation = "sigmoid";
%task
task = "classification";

train = importdata("Problems-1-2-3/Spam-Dataset/train.txt");
test = importdata("Problems-1-2-3/Spam-Dataset/test.txt");

%train size
[m, d] = size(train);
d = d - 1;
%test size
[m_t, d_t] = size(test);
d_t = d_t - 1;

%data and labels
train_data = train(1:m, 1:d);
train_label = train(1:m, d+1);
test_data = test(1:m_t, 1:d_t);
test_label = test(1:m_t, d_t+1);

train_imports = ["Problems-1-2-3/Spam-Dataset/CrossValidation/Fold1/cv-train.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold2/cv-train.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold3/cv-train.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold4/cv-train.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold5/cv-train.txt"];
test_imports = ["Problems-1-2-3/Spam-Dataset/CrossValidation/Fold1/cv-test.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold2/cv-test.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold3/cv-test.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold4/cv-test.txt" ...
    "Problems-1-2-3/Spam-Dataset/CrossValidation/Fold5/cv-test.txt"];

base = "Problems-1-2-3/Spam-Dataset/setting-files/";

cv_size = length(train_imports);
cv_errors = zeros(1, length(d1s));
```

```
training_errors = ones(1, length(d1s));
test_errors = ones(1, length(d1s));

for index = 1:length(d1s)
    d1 = d1s(index);
    load(base + "w1_" + d1 + ".mat")
    load(base + "b1_" + d1 + ".mat")
    load(base + "w2_" + d1 + ".mat")
    load(base + "b2_" + d1 + ".mat")
    %train and test
    X = train_data;
    Y = train_label;

    adjusted_Y = (Y + 1)/2;

    %train tmd
    for n = 1:N
        [w_1, b1, w_2, b2] = update_wb(X, adjusted_Y, w_1, w_2, b1, b2
            , eta, activation, task);
    end

    eta_hat = g(fwb(X, w_1, w_2, b1, b2, activation), activation);
    predictions = sign(eta_hat - 1/2);
    training_errors(index) = classification_error(predictions', Y);
    eta_hat_t = g(fwb(test_data,  w_1, w_2, b1, b2, activation),
        activation);
    predictions_t = sign(eta_hat_t - 1/2);
    test_errors(index) = classification_error(predictions_t',
        test_label);

    %cross validation
    for fold = 1:cv_size
        cv_train = importdata(train_imports(fold));
        cv_test = importdata(test_imports(fold));

        %train size
        [cv_m, cv_d] = size(cv_train);
        cv_d = cv_d - 1;
        %test size
        [cv_m_t, cv_d_t] = size(cv_test);
        cv_d_t = cv_d_t - 1;

        %data and labels
        cv_train_data = cv_train(1:cv_m, 1:cv_d);
        cv_train_label = cv_train(1:cv_m, cv_d+1);
        cv_test_data = cv_test(1:cv_m_t, 1:cv_d_t);
        cv_test_label = cv_test(1:cv_m_t, cv_d_t+1);

        %make it clear for error checking
        cv_X = cv_train_data;
        cv_Y = cv_train_label;
```

```

        load(base + "w1_" + d1 + ".mat")
        load(base + "b1_" + d1 + ".mat")
        load(base + "w2_" + d1 + ".mat")
        load(base + "b2_" + d1 + ".mat")

        adjusted_cv_Y = (cv_Y + 1)/2;

        %train tmd
        for n = 1:N
            [w_1, b1, w_2, b2] = update_wb(cv_X, adjusted_cv_Y, w_1,
               w_2, b1, b2, eta, activation, task);
        end

        eta_hat = g(fwb(cv_X,  w_1, w_2, b1, b2, activation),
            activation);
        predictions = sign(eta_hat - 1/2);
        cv_errors(index) = cv_errors(index) + classification_error(
            predictions', cv_Y)/cv_size;
    end
end

figure

xlabels = [1:6];

plot(xlabels, training_errors, '.-', xlabels, test_errors, '.-', ...
    xlabels, cv_errors, '.-', 'MarkerSize', 15, 'LineWidth', 2)

title('Error Curves')
xlabel('d1 = [1 5 10 15 25 50]')
ylabel('Error')
lngd = legend('Training Error','Test Error', 'Cross Validation Error')
    ;
set(lngd, 'Location', 'NorthEast')
set(lngd, 'fontsize', 10)

selected_index = find(cv_errors == min(cv_errors));
if length(selected_index) > 1
    selected_index = selected_index(1);
end
disp("d1")
d1s(selected_index)
disp("Training Error")
training_errors(selected_index)
disp("Test Error")
test_errors(selected_index)
disp("CV Error")
cv_errors(selected_index)
```

(b) Solution goes here

Algorithm	**Training Error**	**Test Error**
Linear logistic regression (from PS1)	0.0240	0.1066
Linear SVM	0.0320	0.1122
Kernel SVM, polynomial kernel $(\mathbf{x}^\top \mathbf{x}' + 1)^q$	0.0080	0.1204
Kernel SVM, RBF kernel $e^{-\gamma\lVert\mathbf{x}-\mathbf{x}'\rVert_2^2}$	0.0160	0.1117
Neural network (sigmoid units)	0.0360	0.1544
1-NN	0	0.2117
k-NN	0	0.2117

1-NN is best at training data because it literally finds itself, but this feature is useless. k-NN might easily be biased with the existence of 1-NN. Polynomial SVM is good at training data. Linear logistic regression and SVMs all perform well at linear problems.