Appendix:

../app.m

```
startup

y = dataset(:, 1);
X = dataset(:, 3:end);

[train, ~] = data_partition(X, y);
train_X = train(:, 2:end);
train_y = train(:, 1);

% Feature selection ------------------------------
N_feature = [2 3 4 5 6 7 8 9 10 11 12 13];
for n = N_feature
    X_new = [train_y Sequential_Feature_Selection(train_X', train_y', ['''Forward'',', num2str(n) , ',
    save(['dataset_', num2str(n), '_features', '.mat'], 'X_new');
end

X_full = [y X];
% Export full dataset
save('dataset_full', 'X_full');

% Export dataset with PCA dimension reduction ----------------------------
% chosen dimensions: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
M = [1 2 3 4 5 6 7 8 9 10 11 12 13];
for m=M
    [~,~,~,~,W] = PCA(train_X', [], m);
    X_pca = [y (W * X')'];

    save(['dataset_pca_', num2str(m), '.mat'], 'X_pca');
end

% Plot 2D selected feature test dataset ----------------------------
feature_selection_2_plot();

% Plot 2D projected test dataset ----------------------------
pca_2_plot();

% Experiment with Neural Nets with PCA and FS projected data ----------------------------
display('_');
display('Run_neural_networks_experiment._Press_any_key_to_continue...');
pause();

display('PCA_----------');
% alpha <- empirically optimized learning rates
alpha = [4 2 3 3 2 2 1 2 1 1 1 1 1];
Err_pca = zeros(1, 13);
for i = 1:13
    load(['dataset_pca_', num2str(i) ,'.mat']);

    X = X_pca(:, 2:end);
    y = X_pca(:, 1);

    [train, test] = data_partition(X, y);

    train_x = train(:, 2:end);
    train_y = train(:, 1);
    [r, d] = size(train_x);
    C = unique(train_y)';
    train_y = (train_y * (1 ./ C) == ones(r, length(C)));
    H = round(length(train_x) / (length(C) + d) * (length(train_x) / length(X)));

    test_x = test(:, 2:end);
    test_y = test(:, 1);
    [r, ~] = size(test_x);
    test_y = (test_y * (1 ./ C) == ones(r, length(C)));

    % normalize
    [train_x, mu, sigma] = zscore(train_x);
    test_x = normalize(test_x, mu, sigma);
```

```
    rand('state', 0); % fix the initial weight

    nn = nnsetup([d H length(C)]);          % nn structure [input, hidden, ..., hidden, output]
    nn.activation_function = 'tanh_opt';    % Activation functions of hidden layers: 'sigm' (sigmoid)
    nn.learningRate = alpha(i);             % Learning rate
    nn.scaling_learningRate = 0.999;        % Scaling factor for the learning rate (each epoch)
    %     nn.momentum = 0.5;

    opts.numepochs = 1000;
    opts.batchsize = 20; % [10, 14, 20]
    [nn, L] = nntrain(nn, train_x, train_y, opts);

    [er, bad] = nntest(nn, test_x, test_y);
    display(['er = ', num2str(er), ' (', num2str(i) ,'d PCA)' sprintf('\t\t[H=%d, alpha=%d]', H, alpha(
    Err_pca(i) = er;
end

display('Feature selection ----------------');
% alpha <- empirically optimized learning rates
alpha = [0 4 6 1 2 1 2 2 1 1 1 6 1];
Err_fs = zeros(1, 13);
for i = 2:13
    load(['dataset_', num2str(i) ,'_features.mat']);

    X = X_new(:, 2:end);
    y = X_new(:, 1);

    [train, test] = data_partition(X, y);

    train_x = train(:, 2:end);
    train_y = train(:, 1);
    [r, d] = size(train_x);
    C = unique(train_y)';
    train_y = (train_y * (1 ./ C) == ones(r, length(C)));
    H = round(length(train_x) / (length(C) + d) * (length(train_x) / length(X)));

    test_x = test(:, 2:end);
    test_y = test(:, 1);
    [r, ~] = size(test_x);
    test_y = (test_y * (1 ./ C) == ones(r, length(C)));

    % normalize
    [train_x, mu, sigma] = zscore(train_x);
    test_x = normalize(test_x, mu, sigma);

    rand('state', 0); % fix the initial weight

    nn = nnsetup([d H length(C)]); % nn structure [input, hidden, ..., hidden, output]
    nn.activation_function = 'tanh_opt';
    nn.learningRate = alpha(i); % Should decrease over time.
    nn.scaling_learningRate = 0.999;

    opts.numepochs = 1000;
    opts.batchsize = 20;
    [nn, L] = nntrain(nn, train_x, train_y, opts);

    [er, bad] = nntest(nn, test_x, test_y);
    display(['er = ', num2str(er), ' (', num2str(i) ,' features)' sprintf('\t\t[H=%d, alpha=%d]', H, al
    Err_fs(i) = er;
end

figure;
bar([Err_pca' Err_fs']);
title('Feed-forward neural nets error rate');
xlabel('Dimensions');
ylabel('Error');
legend('PCA', 'Feature Selection');
ylim();

% Experiment with Bayesian parameter estimation ----------------------------
%  with 5 features dataset
load 'dataset_5_features.mat'
```

```
X = X_new(:, 2:end);
y = X_new(:, 1);

[train, ~] = data_partition(X, y);
train_x = train(:, 2:end);
train_y = train(:, 1);

save_bayesian_params(train_x, train_y, '5_features');

% with 5D PCA
load 'dataset_pca_5.mat'

X = X_pca(:, 2:end);
y = X_pca(:, 1);

[train, ~] = data_partition(X, y);
train_x = train(:, 2:end);
train_y = train(:, 1);

save_bayesian_params(train_x, train_y, '5_pca');
```