Appendix:

../app.m

```
startup

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

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

[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
    [~, ~, best_features] = Sequential_Feature_Selection(train_X', train_y', ...
        ['[''Forward'',', num2str(n) , ',''LS'',[]]']);
    X_new = [y X(:, best_features)];
    save(['dataset_', num2str(n), '_features', '.mat'], 'X_new');
end

% 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';    % 'sigm' (sigmoid) or 'tanh_opt' (optimal tanh).
    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(i))]);
    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, alpha(i))]);
    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');
```

../startup.m

```
% Clean environment
clc; clear all; close all;

% Load Classification Toolbox
addpath(genpath('/opt/Classification_toolbox'));

% Load Neural Networks Toolbox
addpath(genpath('/opt/DeepLearnToolbox'));

% Load dataset
dataset = load('leaf.csv');
```

../perform_knn_on_data.m

```
features = 14;
neighbors = 12;

accuracy_PCA = zeros(features-1, neighbors);
accuracy_features = zeros(features, neighbors);

dataset = load('leaf.csv');
y = dataset(:, 1);

%% Train and compute accuracies
for m=1:features
    %user feedback
    disp(['Performing KNN with ', num2str(m), ' features']);

    if(m<14)
        if(m>1)
            load(['dataset_', num2str(m), '_features.mat'], 'X_new');
            [X_new_train, X_new_test] = data_partition(X_new(:, 2:end), y);
        end
        load(['dataset_pca_', num2str(m), '.mat'], 'X_pca');
        [X_pca_train,X_pca_test] = data_partition(X_pca(:, 2:end), y);
    else
        [X_new_train, X_new_test] = data_partition(dataset(:, 3:end), y);
    end
    for(k=1:neighbors)
        if(m>1)
            learned_test_features_class = Nearest_Neighbor(X_new_train(:, 2:end)', ...
                X_new_train(:, 1)', X_new_test(:, 2:end)', k);
            true_test_class_features = X_new_test(:, 1);
            accuracy_features(m,k) = sum(learned_test_features_class' == true_test_class_features) ...
                / length(true_test_class_features);
        end
```

```
            if(m<14)
                learned_test_PCA_class = Nearest_Neighbor(X_pca_train(:, 2:end)', X_pca_train(:, 1)', ...
                    X_pca_test(:, 2:end)', k);
                true_test_class_PCA = X_pca_test(:, 1);
                accuracy_PCA(m,k) = sum(learned_test_PCA_class' == true_test_class_PCA) ...
                    / length(true_test_class_PCA);
            end
        end
end

%% Graphs
disp('generating graphs')
figure
surf(1-accuracy_features)
set(gca,'YDir','Reverse')
ylabel('features')
ylim([1 14])
xlabel('neighbors')
xlim([1 12])
title('Error using KNN with Forward Feature Selection')
zlabel('error')
zlim([0 1])
set(gcf, 'InvertHardCopy', 'off');
figure
surf(1-accuracy_PCA)
set(gca,'YDir','Reverse')
ylabel('features')
ylim([1 14])
xlabel('neighbors')
xlim([1 12])
title('Error using KNN with Principal Component Analysis')
zlabel('error')
zlim([0 1])
set(gcf, 'InvertHardCopy', 'off');

%% eliminate temp variables
clear X_pca_train X_pca X_new X_new_train X_new_test
clear learned_test_PCA_class learned_test_features_class true_test_class_PCA true_test_class_features

disp('Accuracy results may be seen in accuracy_PCA and accuracy_features')
```

../data_partition.m

```
function [ train, test ] = data_partition( X, target )
% data_partition - Split dataset into train and test set

    C = unique(target)';

    train = [];
    test = [];
    for c = C
        Data_given_c = [target(target == c) X(target == c, :)];
        train = cat(1, train, Data_given_c(1:end-2, :));
        test = cat(1, test, Data_given_c(end-1:end, :));
    end
end
```

../feature_selection_2_plot.m

```
function feature_selection_2_plot()
% FEATURE_SELECCTION_2_PLOT - generate 2d plot for
%        dataset from feature selection method

display(' ');
display('Generating plot (feature selection). Press any key to continue...');
pause();

load 'dataset_2_features.mat';

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

[~, test] = data_partition(X, y);
```

```
y = test(:, 1);
x1 = test(:, 2);
x2 = test(:, 3);

plot2(x1, x2, y, '2D test dataset (Feature selection)');
```

../pca_2_plot.m

```
function pca_2_plot()
% PCA_2_PLOT - generate 2d plot for
%          dataset from PCA method

display(' ');
display('Generating plot (PCA). Press any key to continue...');
pause();

load 'dataset_pca_2.mat';

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

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

y = test(:, 1);
x1 = test(:, 2);
x2 = test(:, 3);

plot2(x1, x2, y, '2D test dataset (PCA)');
```

../plot2.m

```
function plot2( x1, x2, y, plot_title )
% PLOT2 - draw 2D plot from leaf data

figure;
K = unique(y);
markers = '.ox+*sdv^<>pd';
L = {}; % legend

for k = K'
    X1 = x1(y == k);
    X2 = x2(y == k);
    index = fix(1 + (length(markers)-1) * rand);
    marker = markers(index);

    scatter(X1, X2, marker); hold on
    L{end+1} = ['C', num2str(k)];
end
hold off
title(plot_title);
xlabel('x1');
ylabel('x2');
legend(L);
```

../save_bayesian_params.m

```
function save_bayesian_params( train_x, train_y, output )
% SAVE_BAYESIAN_PARAMS - Save bayesian parameters to ".mat" file

[~, d] = size(train_x);
K = unique(train_y);
Sigma = zeros(length(K), d, d); % Covariance for each class
for i = 1:length(K)
    X_given_y = train_x(train_y == K(i),:);
    [~, Sigma(i,:,:)] = mle(X_given_y);
end

[mu, Sigma] = Bayesian_parameter_est(train_x', train_y', Sigma);

save(['mu_', output, '.mat'], 'mu');
save(['Sigma_', output, '.mat'], 'Sigma');
```