Contents


```
% ECE414 - Bayesian Machine Learning
% Authors   : Junbum Kim, Andy Jeong
% Project 3 : Classification (Generative, IRLS)
% Date      : October 30, 2019
% Reference : Pattern Recognition and Machine Learning
%                              by Chris. M. Bishop (2006)
clear all; close all; clc; warning('off','all');
```

Equations

1 Generative Model

$$\text{Shared Covariance } \Sigma = \pi\Sigma_1 * (1-\pi) * \Sigma_2$$

where $\pi =$ class prior probability

Maximize for data points from each class:

$$p(x_n, C_i) = p(C_i)p(x_n|C_i) = \pi N(x_n|\mu_1.\Sigma)$$

$$\text{Posterior: } p(C_i|\phi) = y(\phi) = \sigma(w^T\phi)$$

2 Iterative Reweighted Least Squares (IRLS)

$$\text{Design Matrix (N x M) } \Phi = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \dots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_{M-1}(x_N) \end{bmatrix}$$

Update eqn for the Newton-Raphson method,
minimizing the cross-entropy error function E(w):

$$w^{(new)} = w^{(old)} - H^{-1}\nabla E(w)$$

$$\text{Gradient (of the error fcn) } \nabla E(w) = \sum_{n=1}^{N}(y_n - t_n)\phi_n = \Phi^T(y-t)$$

$$\text{Hessian Matrix (of the error fcn) } H = \nabla\nabla E(w) = \sum_{n=1}^{N}(y_n(1-y_n)\phi_n\phi_n^T) = \Phi^T R\Phi$$

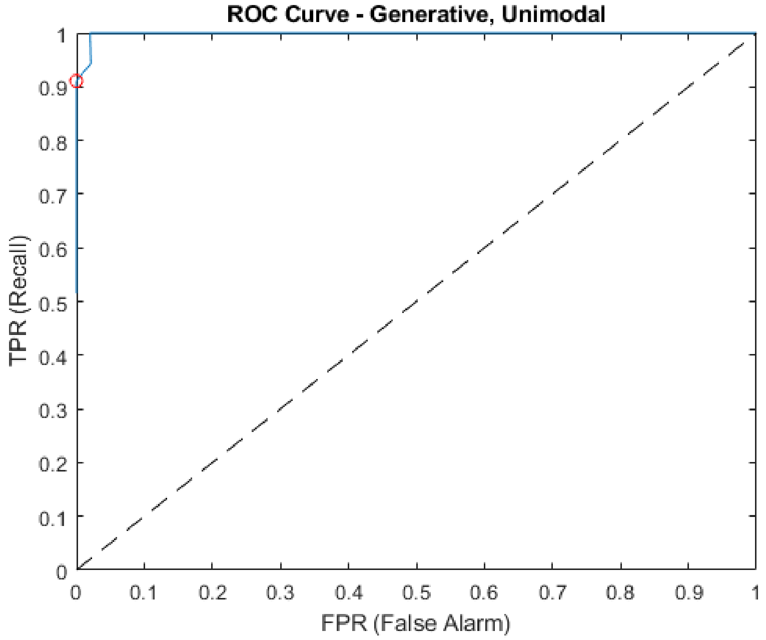
where R_{nn} (N x N matrix) $= y_n(1-y_n)$

1a) Gaussian Generative Model - Unimodal Data

Data type: Unimodal

```
% load data
load('mlData.mat');
x = unimodal.x;
y = unimodal.y;

% set seed for reproducibility
rng(42); % always the answer

% split train and test set by randomly sampling the train/test indices
% * ratio: 75% train - 25% test
train_idx = randsample(400,300);
test_idx = setdiff(1:400, train_idx);

% set train and test subsets
train_x = x(train_idx, :);
train_y = y(train_idx);
test_x = x(test_idx, :);
test_y = y(test_idx);

% MLE estimate
% set class labels for the training set
class1 = train_y == 0;
class2 = train_y == 1;

% PI = prior probabiity of class 1 (labelled as '0')
% mu1, mu2 = mean vector of class 1, class 2, respectively
% S1, S2 = covariance matrix of class 1, class 2, respectively
% S = probability distribution given prior and covariance of each class
PI = mean(class1);
mu1 = mean(train_x(class1,:))';
mu2 = mean(train_x(class2,:))';
S1 = cov(train_x(class1,:));
S2 = cov(train_x(class2,:));
S = PI * S1 + (1-PI) * S2;

% p(xn, Ci): probability of data points from each class 1 and 2 (with mu1, mu2 as mean
% vectors, respectively), with Gaussian class conditional densities
% p(x|Ci) and shared covariance matrix S
% -- take logarithmic, and pass through activation function (sigmoid)
pC1 = PI * mvnpdf(test_x,mu1',S);
pC2 = (1-PI) * mvnpdf(test_x,mu2',S);
a = log(pC1./pC2);
pred1 = round(sigmoid(a));
pred2 = round(1-sigmoid(a));
unimodal1 = mean(pred2==test_y)

% plot ROC
% red dot represents the test accuracy, blue is the ROC curve
roc_data = roc_curve(test_y, a);
idx = find(roc_data.accuracy==unimodal1, 1);
figure; plot(1-roc_data.specificity, roc_data.recall);
hold on; plot(1-roc_data.specificity(idx), roc_data.recall(idx),'ro');
ref = refline(1,0); ref.LineStyle = '--'; ref.Color = 'k';
xlim([0 1]); ylim([0 1]); ylabel('TPR (Recall)'); xlabel('FPR (False Alarm)');
title('ROC Curve - Generative, Unimodal');

% Tabular summary for Generative Model on unimodal data
datatype = {'Unimodal'};
method = {'Generative'};
accuracy = roc_data.accuracy(idx);
f1 = roc_data.F1(idx);
precision = roc_data.precision(idx);
recall = roc_data.precision(idx);
specificity = roc_data.specificity(idx);
sensitivity = roc_data.sensitivity(idx);
g_mean = roc_data.g_mean(idx);
T1 = table(datatype, method, accuracy, recall, precision, f1, specificity, sensitivity, g_mean);
snapnow;
```

```
unimodal1 =

    0.9600
```

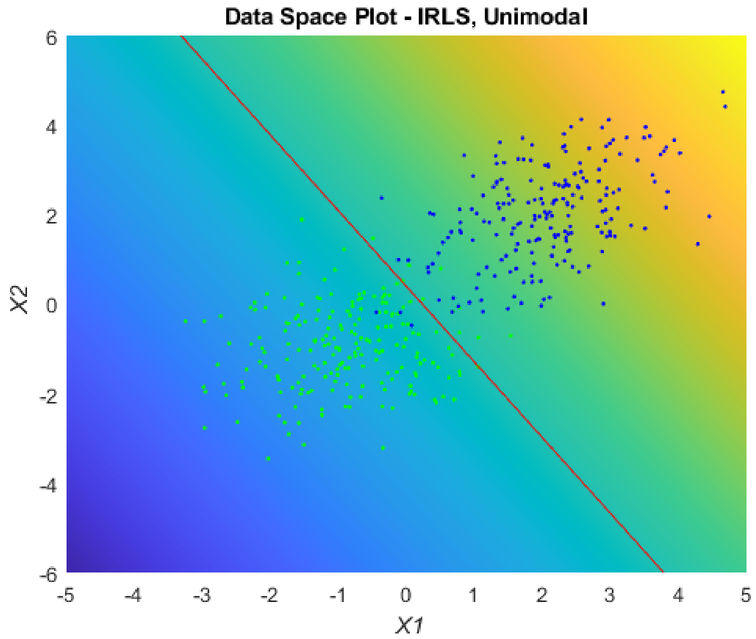


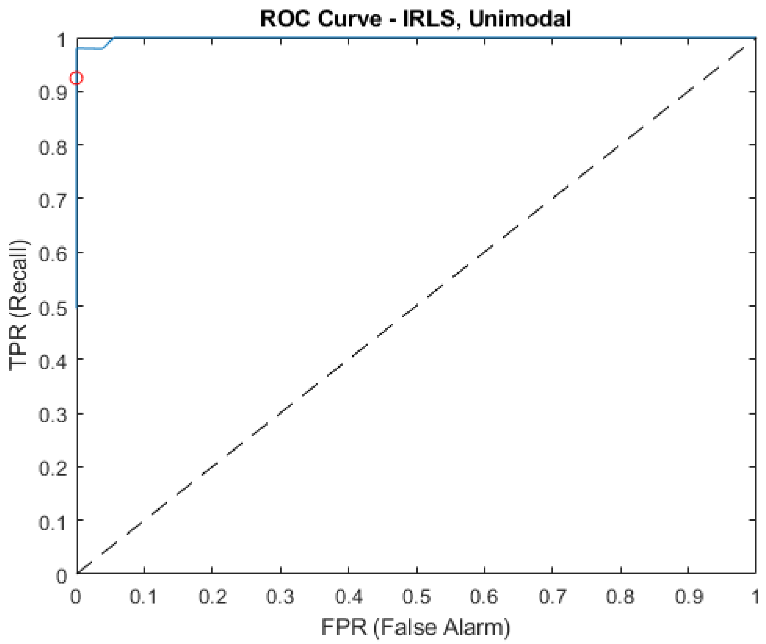
1b) IRLS method - Unimodal Data

Data type: Unimodal

```
% set seed for reproducibility
rng(42);

% add bias to adjust for shift in data points
% split train and test set
x = [ones(400,1), unimodal.x];
y = unimodal.y;

% set parameters for training
% batch size and learning rate set for better convergence of IRLS
epoch = 30;
sampling_rate = 100;
batch_size = 8;
learning_rate = 0.01;

% set train and test subsets
train_x = x(train_idx,:);
train_y = y(train_idx);
test_x = x(test_idx,:);
test_y = y(test_idx);

% pass through IRLS algorithm with the defined parameters
w = irls(train_x, train_y, epoch, learning_rate, batch_size);

% identify 2D grid, with the updated w parameters from IRLS
[w0, w1] = meshgrid(linspace(-5,5,sampling_rate)',linspace(-6,6,sampling_rate));
grid1 = ones(sampling_rate);
grid2 = w0;
grid3 = w1;
grid = grid1 * w(1) + grid2 * w(2) + grid3 * w(3);

% variables for the decision boundary (linear)
x_decision = linspace(-5,5,1000);
y_decision = -(w(1) + w(2) * x_decision)/w(3);

% plot the classified data points and decision boundary
figure;
pcolor(w0,w1,grid); shading interp; hold on;
plot(x(y==0, 2), x(y==0, 3), "g.", ...
     x(y==1, 2), x(y==1, 3), "b.", ...
     x_decision, y_decision,'r-'); hold off;
title('Data Space Plot - IRLS, Unimodal');
ylim([-6,6]); xlabel('{\it X1}'); ylabel('{\it X2}');

% evaluate (prediction)
prediction = test_x * w;
unimodal2 = mean(prediction > 0 == test_y);

% plot ROC
% red dot represents the test accuracy, blue is the ROC curve
roc_data = roc_curve(~test_y, prediction);
```

```
idx = find(roc_data.accuracy==unimodal2, 1);
figure; plot(1-roc_data.specificity, roc_data.recall); hold on;
plot(1-roc_data.specificity(idx), roc_data.recall(idx),'ro');
xlim([0 1]); ylim([0 1]);
ref = refline(1,0); ref.LineStyle = '--'; ref.Color = 'k';
title('ROC Curve - IRLS, Unimodal'); ylabel('TPR (Recall)'); xlabel('FPR (False Alarm)');
hold off;

% Tabular summary for IRLS on unimodal data
datatype = {'Unimodal'};
method = {'IRLS'};
accuracy = roc_data.accuracy(idx);
f1 = roc_data.F1(idx);
precision = roc_data.precision(idx);
recall = roc_data.precision(idx);
specificity = roc_data.specificity(idx);
sensitivity = roc_data.sensitivity(idx);
g_mean = roc_data.g_mean(idx);
T2 = table(datatype, method, accuracy, recall, precision, f1, specificity, sensitivity, g_mean);
```





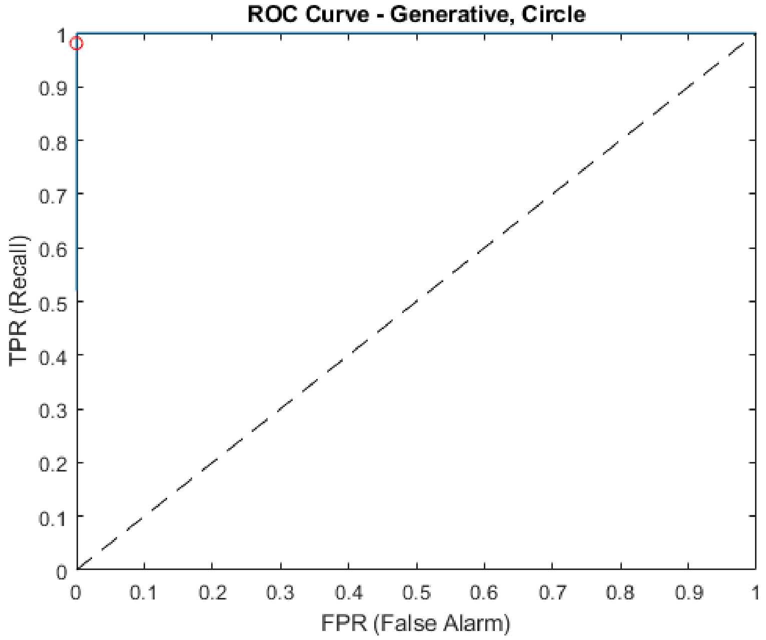
2a) Gaussian Generative Model - Circle Data

Data type: Circle

```
% load data
% Generative Models split train and test set, and add r2 as a feature
```

```
load('mlData.mat');
x = [circles.x, circles.x(:,1).^2 + circles.x(:,2).^2];
y = circles.y;

% set seed for reproducibility
rng(42);

% split train and test set by randomly sampling the train/test indices
% * ratio: 75% train - 25% test
train_idx = randsample(400,300);
test_idx = setdiff(1:400, train_idx);

% set train and test subsets
train_x = x(train_idx,:);
train_y = y(train_idx);
test_x = x(test_idx,:);
test_y = y(test_idx);

% MLE estimate
% set class labels for the training set
class1 = train_y == 0;
class2 = train_y == 1;

% PI = prior probabiity of class 1 (labelled as '0')
% mu1, mu2 = mean vector of class 1, class 2, respectively
% S1, S2 = covariance matrix of class 1, class 2, respectively
% S = probability distribution given prior and covariance of each class
PI = mean(class1);
mu1 = mean(train_x(class1,:))';
mu2 = mean(train_x(class2,:))';
S1 = cov(train_x(class1,:));
S2 = cov(train_x(class2,:));
S = PI * S1 + (1-PI) * S2;

% p(xn, Ci): probability of data points from each class 1 and 2 (with mu1, mu2 as mean
% vectors, respectively), with Gaussian class conditional densities
% p(x|Ci) and shared covariance matrix S
% -- take logarithmic, and pass through activation function (sigmoid)
pC1 = PI * mvnpdf(test_x,mu1',S);
pC2 = (1-PI) * mvnpdf(test_x,mu2',S);
a = log(pC1./pC2);
pred1 = round(sigmoid(a));
pred2 = round(1-sigmoid(a));
circle1 = mean(pred2==test_y)

% plot ROC
% red dot represents the test accuracy, blue is the ROC curve
roc_data = roc_curve(test_y, a);
idx = find(roc_data.accuracy==circle1, 1);
figure; plot(1-roc_data.specificity, roc_data.recall);
hold on; plot(1-roc_data.specificity(idx), roc_data.recall(idx),'ro');
xlim([0 1]); ylim([0 1]);
ref = refline(1,0); ref.LineStyle = '--'; ref.Color = 'k';
title('ROC Curve - Generative, Circle');
ylabel('TPR (Recall)'); xlabel('FPR (False Alarm)');

% Tabular summary for Generative Model on circle data
datatype = {'Circle'};
method = {'Generative'};
accuracy = roc_data.accuracy(idx);
f1 = roc_data.F1(idx);
precision = roc_data.precision(idx);
recall = roc_data.precision(idx);
specificity = roc_data.specificity(idx);
sensitivity = roc_data.sensitivity(idx);
g_mean = roc_data.g_mean(idx);
T3 = table(datatype, method, accuracy, recall, precision, f1, specificity, sensitivity, g_mean);
snapnow;
```

```
circle1 =

     1
```



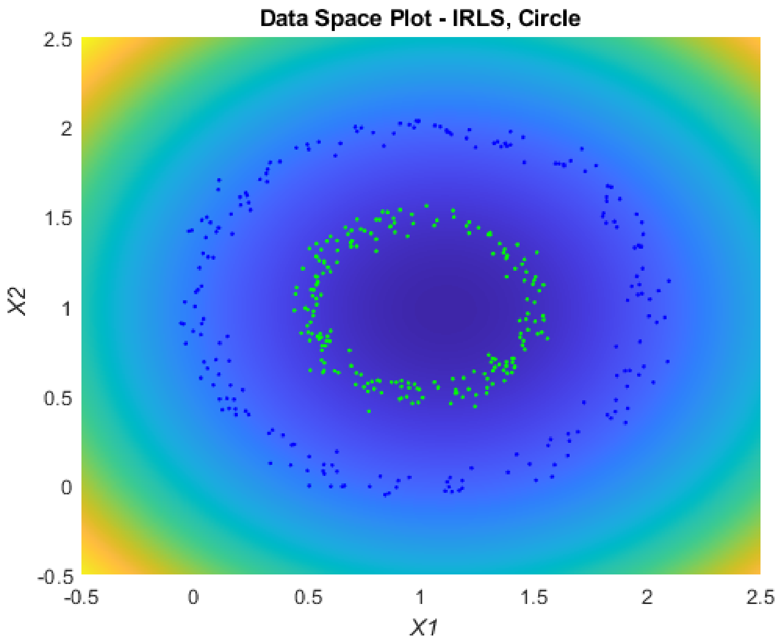
2b) IRLS method - Circle Data

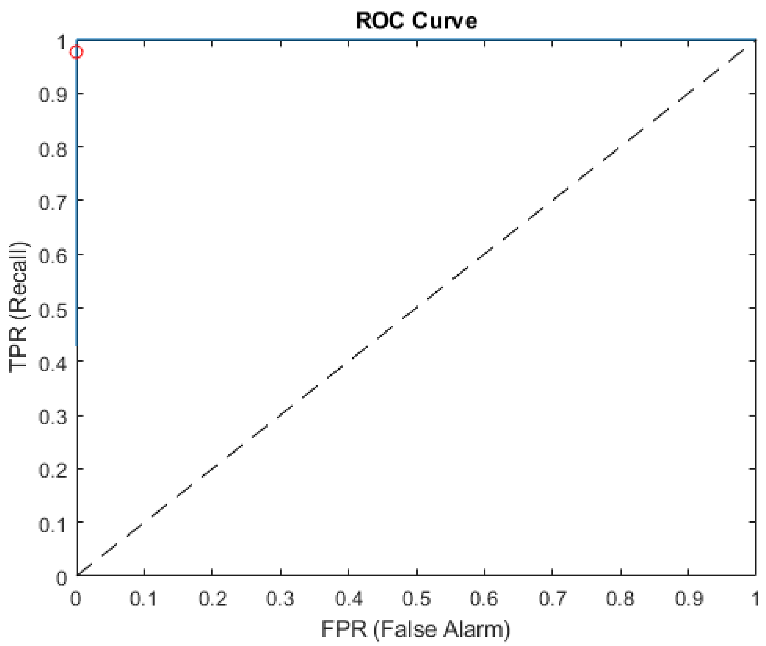
Data type: Circle

```
% load data
% adjustments: (1) add another feature (sum of square terms)
%                  with shift to the right (1,1) to fit to the given data
%              (2) add bias (1's) to adjust for data shift
x = [ones(400,1), circles.x, (circles.x(:,1)-1).^2 + (circles.x(:,2)-1).^2];
y = circles.y;

% parameters for training
% batch size and learning rate set for better convergence of IRLS
epoch = 200;
sampling_rate = 100;
batch_size = 8;
learning_rate = 0.01;

% randomly sample training and test sets
% with ratio 75% train - 25% test (~ 80-20 rule)
train_idx = randsample(400,300);
test_idx = setdiff(1:400, train_idx);

% set train and test subsets
train_x = x(train_idx,:);
train_y = y(train_idx);
test_x = x(test_idx,:);
test_y = y(test_idx,:);

% pass through IRLS algorithm with the defined parameters
w = irls(train_x, train_y, epoch, learning_rate, batch_size);

% define a 2D grid, with the updated w parameters from IRLS
[w0, w1] = meshgrid(linspace(-0.5,2.5,sampling_rate)',linspace(-0.5,2.5,sampling_rate));
grid1 = ones(sampling_rate);
grid2 = w0;
grid3 = w1;
grid4 = (w0-1).^2 + (w1-1).^2;
grid = grid1 * w(1) + grid2 * w(2) + grid3 * w(3) + grid4 * w(4);

% plot the classified data points and decision boundary
figure;
pcolor(w0,w1,grid); hold on; shading interp;
plot(circles.x(y==0, 1), circles.x(y==0, 2), "g.", ...
     circles.x(y==1, 1), circles.x(y==1, 2), "b."); hold off;
title('Data Space Plot - IRLS, Circle');
xlabel('{\it X1}'); ylabel('{\it X2}');  hold off;

% evaluate (prediction accuracy)
prediction = test_x * w;
circle2 = mean(prediction > 0 == test_y)

% plot ROC
% red dot represents the test accuracy, blue is the ROC curve
roc_data = roc_curve(~test_y, prediction);
```

```
idx = find(roc_data.accuracy==circle2, 1);
figure; plot(1-roc_data.specificity, roc_data.recall);
hold on; plot(1-roc_data.specificity(idx), roc_data.recall(idx),'ro');
xlim([0 1]); ylim([0 1]);
ref = refline(1,0); ref.LineStyle = '--'; ref.Color = 'k';
title('ROC Curve'); ylabel('TPR (Recall)'); xlabel('FPR (False Alarm)');

% Tabular summary for IRLS on circle data
datatype = {'Circle'};
method = {'IRLS'};
accuracy = roc_data.accuracy(idx);
f1 = roc_data.F1(idx);
precision = roc_data.precision(idx);
recall = roc_data.precision(idx);
specificity = roc_data.specificity(idx);
sensitivity = roc_data.sensitivity(idx);
g_mean = roc_data.g_mean(idx);
T4 = table(datatype, method, accuracy, recall, precision, f1, specificity, sensitivity, g_mean);
snapnow;
```

```
circle2 =

     1
```





Application to 'iris' dataset

```
load("fisheriris.mat");

% set seed for reproducibility
rng(42);
```

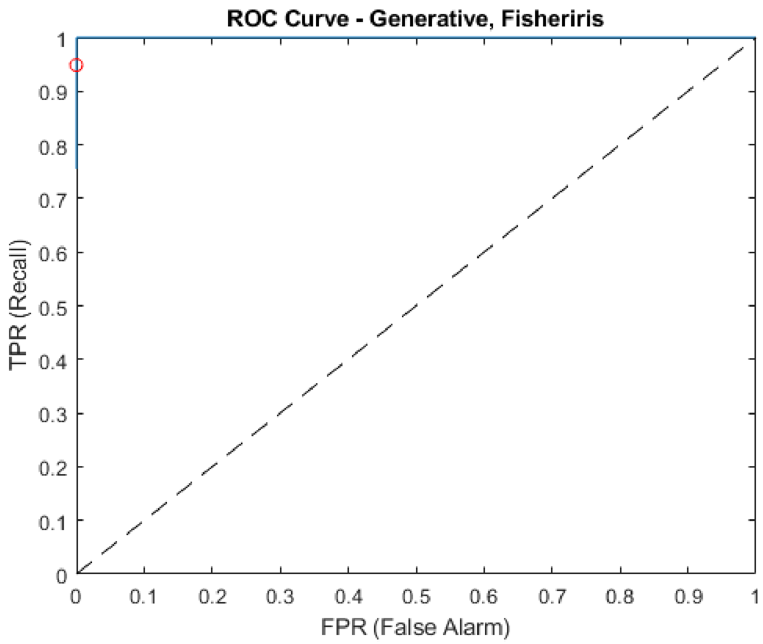
3a) Gaussian Generative Model - iris

4D feature space is reduced to 2D so that it closely follows previous unimodal and circle data, using principal component analysis

```
[~, score, ~] = pca(meas);
x = score(:,1:2);
y = species == "setosa"; % only consider 'setosa' class

% randomly sample training and test sets
% with ratio 2/3 train - 1/3 test
train_idx = randsample(150,100);
test_idx = setdiff(1:150, train_idx);

% set train and test subsets
train_x = x(train_idx,:);
train_y = y(train_idx);
test_x = x(test_idx,:);
test_y = y(test_idx);

% MLE estimate
% set class labels for the training set
class1 = train_y == 0;
class2 = train_y == 1;

% PI = prior probabiity of class 1 (labelled as '0')
% mu1, mu2 = mean vector of class 1, class 2, respectively
% S1, S2 = covariance matrix of class 1, class 2, respectively
% S = probability distribution given prior and covariance of each class
PI = mean(class1);
mu1 = mean(train_x(class1,:))';
mu2 = mean(train_x(class2,:))';
S1 = cov(train_x(class1,:));
S2 = cov(train_x(class2,:));
S = PI * S1 + (1-PI) * S2;

% p(xn, Ci): probability of data points from each class 1 and 2 (with mu1, mu2 as mean
% vectors, respectively), with Gaussian class conditional densities
% p(x|Ci) and shared covariance matrix S
% -- take logarithmic, and pass through activation function (sigmoid)
pC1 = PI * mvnpdf(test_x,mu1',S);
pC2 = (1-PI) * mvnpdf(test_x,mu2',S);
a = log(pC1./pC2);
pred1 = round(sigmoid(a));
pred2 = round(1-sigmoid(a));
iris1 = mean(pred2==test_y)

% plot ROC
% red dot represents the test accuracy, blue is the ROC curve
roc_data = roc_curve(test_y, a);
idx = find(roc_data.accuracy==iris1, 1);
figure; plot(1-roc_data.specificity, roc_data.recall);
hold on; plot(1-roc_data.specificity(idx), roc_data.recall(idx),'ro');
xlim([0 1]); ylim([0 1]);
ref = refline(1,0); ref.LineStyle = '--'; ref.Color = 'k';
title('ROC Curve - Generative, Fisheriris');
ylabel('TPR (Recall)'); xlabel('FPR (False Alarm)');

% Tabular summary for Generative Model on iris data
datatype = {'Fisheriris'};
method = {'Generative'};
accuracy = roc_data.accuracy(idx);
f1 = roc_data.F1(idx);
precision = roc_data.precision(idx);
recall = roc_data.precision(idx);
specificity = roc_data.specificity(idx);
sensitivity = roc_data.sensitivity(idx);
g_mean = roc_data.g_mean(idx);
T5 = table(datatype, method, accuracy, recall, precision, f1, specificity, sensitivity, g_mean);
snapnow;
```

```
iris1 =
```

1



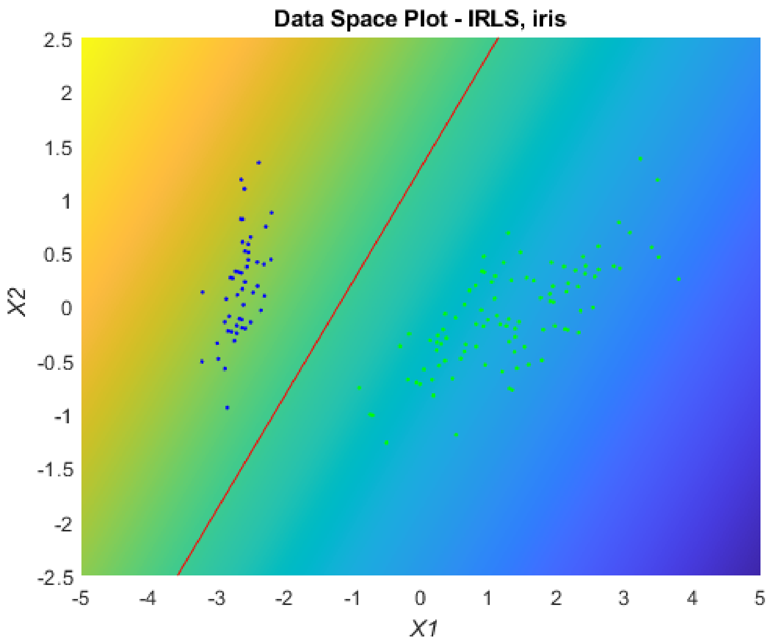
3b) IRLS method - iris

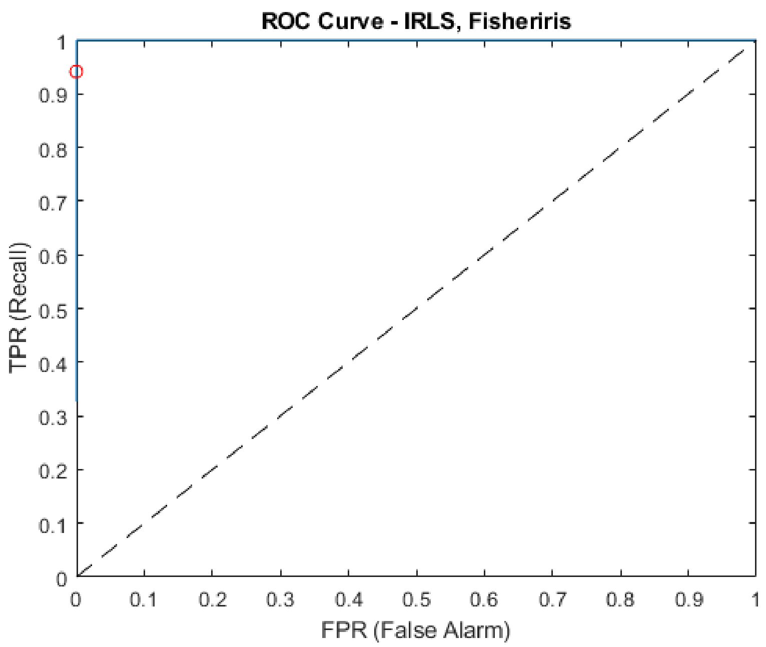
4D feature space is reduced to 2D so that it closely follows previous unimodal and circle data, using principal component analysis

```
[~, score, ~] = pca(meas);
x = [ones(150,1), score(:,1:2)];
y = species=="setosa"; % only consider 'setosa' class

train_idx = randsample(150,100);
test_idx = setdiff(1:150, train_idx);

% set train and test subsets
train_x = x(train_idx,:);
train_y = y(train_idx);
test_x = x(test_idx,:);
test_y = y(test_idx);

% parameter for plotting
% batch size and learning rate set for better convergence of IRLS
epoch = 100;
sampling_rate = 100;
batch_size = 8;
learning_rate = 0.05;

% pass through IRLS algorithm with the defined parameters
w = irls(train_x, train_y, epoch, learning_rate, batch_size);

% variables for the decision boundary (linear)
x_decision = linspace(-5,5,1000);
y_decision = -(w(1) + w(2) * x_decision)/w(3);

% identify 2D grid, with the updated w parameters from IRLS
[w0, w1] = meshgrid(linspace(-5,5,sampling_rate)',linspace(-2.5,2.5,sampling_rate));
grid1 = ones(sampling_rate);
grid2 = w0;
grid3 = w1;
grid = grid1 * w(1) + grid2 * w(2) + grid3 * w(3);

% plot the classified data points and decision boundary
figure; pcolor(w0,w1,grid); hold on; shading interp;
plot(x(y==0,2), x(y==0,3), "g.",x(y==1,2), x(y==1,3), "b.", x_decision, y_decision,'r');
hold off; title('Data Space Plot - IRLS, iris');
ylim([-2.5,2.5]); xlabel('{\it X1}'); ylabel('{\it X2}'); hold off;

% evaluate (prediction)
prediction = test_x * w;
iris2 = mean(prediction > 0 == test_y)

% plot ROC
% red dot represents the test accuracy, blue is the ROC curve
roc_data = roc_curve(~test_y, prediction);
```

```
idx = find(roc_data.accuracy==iris2, 1);
figure; plot(1-roc_data.specificity, roc_data.recall);
hold on; plot(1-roc_data.specificity(idx), roc_data.recall(idx),'ro');
xlim([0 1]); ylim([0 1]);
ref = refline(1,0); ref.LineStyle = '--'; ref.Color = 'k';
title('ROC Curve - IRLS, Fisheriris'); ylabel('TPR (Recall)'); xlabel('FPR (False Alarm)');


% Tabular summary for IRLS on iris data
datatype = {'Fisheriris'};
method = {'IRLS'};
accuracy = roc_data.accuracy(idx);
f1 = roc_data.F1(idx);
precision = roc_data.precision(idx);
recall = roc_data.precision(idx);
specificity = roc_data.specificity(idx);
sensitivity = roc_data.sensitivity(idx);
g_mean = roc_data.g_mean(idx);
T6 = table(datatype, method, accuracy, recall, precision, f1, specificity, sensitivity, g_mean);
snapnow;
```

```
iris2 =

     1
```





Results (Evaluation Statistics)

```
Results = [T1;T2;T3;T4;T5;T6]
hold off; warning('on','all');
snapnow;
```

```
Results =

  6×9 table

      datatype           method        accuracy    recall     precision      f1       specificity    sensitivity    g_mean
    ______________    ______________    ________    _______    _________    _______    ___________    ___________    _______

    {'Unimodal'  }    {'Generative'}      0.96      0.98039     0.98039     0.94427         1            0.9434      0.95431
    {'Unimodal'  }    {'IRLS'      }      0.97            1           1     0.96078         1           0.94231      0.96152
    {'Circle'    }    {'Generative'}         1            1           1     0.99029         1                 1      0.99034
    {'Circle'    }    {'IRLS'      }         1            1           1     0.98824         1                 1       0.9883
    {'Fisheriris'}    {'Generative'}         1            1           1     0.97368         1                 1      0.97402
    {'Fisheriris'}    {'IRLS'      }         1            1           1      0.9697         1                 1      0.97014
```

Utility Functions

```
% see Equation 2 for IRLS
function [w] = irls(iota, y, epoch, lr, batch_size)

    sz = size(iota);
    w = rand([sz(2),1]) * 10 - 5; % -5 to 5

    for i = 2:epoch
        batch = randsample(sz(1),batch_size);
        iota_i = iota(batch,:);
        t_i = y(batch);

        y_i = sigmoid(iota_i * w);
        R_i = diag(y_i .* (1-y_i));

        gradient = iota_i' * (y_i - t_i);
        hessian = iota_i' * R_i * iota_i;

        w = w - lr * pinv(hessian) * gradient;
    end
end

% activation function: sigmoid
function [out] = sigmoid(x)
    out = 1./(1+exp(-x));
end

% evaluation metric calculations
function ROC_data = roc_curve(target, prediction)

    thresholds = linspace(min(prediction), max(prediction), 100);
    % start point at (0,0)
    start.recall = 0; start.specificity = 1;
    recall = [start.recall, zeros(size(thresholds))];
    specificity = [start.specificity, zeros(size(thresholds))];
    sensitivity = zeros(size(thresholds));
    precision = zeros(size(thresholds));
    F1 = zeros(size(thresholds));
    g_mean = zeros(size(thresholds));
    accuracy = zeros(size(thresholds));

    for idx = 1:length(thresholds)
        thresh = prediction >= thresholds(idx);
        pos_idx = find(thresh==1);
        neg_idx = find(thresh==0);

        TP = length(intersect(find(target == 0), pos_idx));
        FP = length(intersect(find(target == 0), neg_idx));
        TN = length(intersect(find(target == 1), neg_idx));
        FN = length(intersect(find(target == 1), pos_idx));
        recall(idx+1)       = TP / (TP+FN);
        specificity(idx+1)  = TN / (FP+TN);
        sensitivity(idx) = TP / (TP+FN);
        precision(idx)    = TP / (TP+FP);
        accuracy(idx)     = (TP+TN)/(TP+TN+FP+FN);
        F1(idx) = 2 / (recall(idx)^-1 + precision(idx)^-1);
        g_mean(idx) = sqrt(specificity(idx) * recall(idx));
```

```
    end
    % mark end point at (1,1)
    recall(end+1) = 1; specificity(end+1) = 0;
    ROC_data.recall = recall;
    ROC_data.specificity = specificity;
    ROC_data.sensitivity = sensitivity;
    ROC_data.precision = precision;
    ROC_data.accuracy = accuracy;
    ROC_data.F1 = F1;
    ROC_data.g_mean = g_mean;
end
```

Published with MATLAB® R2019b