Table of Contents


```
clear all, close all, clc;

data_filename = 'student_data.csv';
data_table = rows2vars(readtable(data_filename,'ReadRowNames',true));

hours = data_table.Hours;
pass_fail = data_table.Pass;
N = size(data_table, 1); % number of students

wiki_w0 = -4.0777;
wiki_w1 = 1.5046;
```

Part 1. Plotting the contour map of the cross-entropy cost fxn J

```
w0max = 10;
w1max = 10;

w0_range = -w0max:0.5:w0max;
w1_range = -w1max:0.5:w1max;
[W0, W1] = meshgrid(w0_range, w1_range);

J = zeros(size(W0));

for w0_ind = 1:length(w0_range)
    for w1_ind = 1:length(w1_range)
        w0 = w0_range(w0_ind);
        w1 = w1_range(w1_ind);
        fx = 1 ./ (1 + exp(-w0 - (w1 .* hours)));
        wiki_fx = 1 ./ (1 + exp(-wiki_w0 - (wiki_w1 .* hours)));
        J_temp = 0;
        wiki_J_temp = 0;
        for i = 1:N
            J_temp = J_temp + (pass_fail(i)*log(fx(i)) + (1 -
 pass_fail(i))*log(1-fx(i)));
            wiki_J_temp = wiki_J_temp + (pass_fail(i)*log(wiki_fx(i))
 + (1 - pass_fail(i))*log(1-wiki_fx(i)));
        end
        J(w1_ind, w0_ind) = (-1/N)*J_temp; % indexing into meshgrid-
sized array, m = len(y), n = len(x)
        wiki_J = (-1/N)*wiki_J_temp;
    end
end
```

```
figure();
surf(W0, W1, J,'EdgeColor','none', 'FaceAlpha', .4);
hold on;
scatter3(wiki_w0, wiki_w1, wiki_J, 30,'magenta','filled');
title('Surface of J Cross Entropy Fxn');
xlabel('w0')
ylabel('w1')
zlabel('J')
legend({'Surface of J','Optimal weights
 (Wiki)'},'location','southoutside')
hold off;

figure();
% surf(W0, W1, J,'EdgeColor','none', 'FaceAlpha', 0.3);
hold on;
contour(W0, W1, J);
title({'Contour Plot of J','Gradient Descent, LR = 2, 20 iter'}); %
 this title comes from the process in part 2
xlabel('w0')
ylabel('w1')
zlabel('J')
axis square
```

Part 2. Performing Gradient Descent on J (learning rate = 2, 20 iterations)

```
dL = 2; % learning rate
gd_iter_max = 20;
init_camp = [0, -4];
camp_coords = zeros([gd_iter_max+1, 2]);
camp_coords(1,:) = init_camp;

for gd_iter = 1:1:gd_iter_max
    w0_current = camp_coords(gd_iter, 1);
    w1_current = camp_coords(gd_iter, 2);
    delJ_w0 = 0;
    delJ_w1 = 0;
    for i = 1:N
        delJ_w0 = delJ_w0 + (-pass_fail(i)) * (1/(1 + exp(w0_current
 + w1_current * hours(i)))) + (1 - pass_fail(i))* (1/(1 + exp(-
w0_current - w1_current * hours(i))));
        delJ_w1 = delJ_w1 + hours(i) * (-pass_fail(i) * (1/(1 +
 exp(w0_current + w1_current * hours(i)))) + (1 - pass_fail(i))* (1/(1
 + exp(-w0_current - w1_current * hours(i)))));
    end
    delJ_w0 = (1/N) * delJ_w0;
    delJ_w1 = (1/N) * delJ_w1;
    camp_coords(gd_iter + 1, :) = [(w0_current - dL * delJ_w0),
 (w1_current - dL * delJ_w1)];
end
```

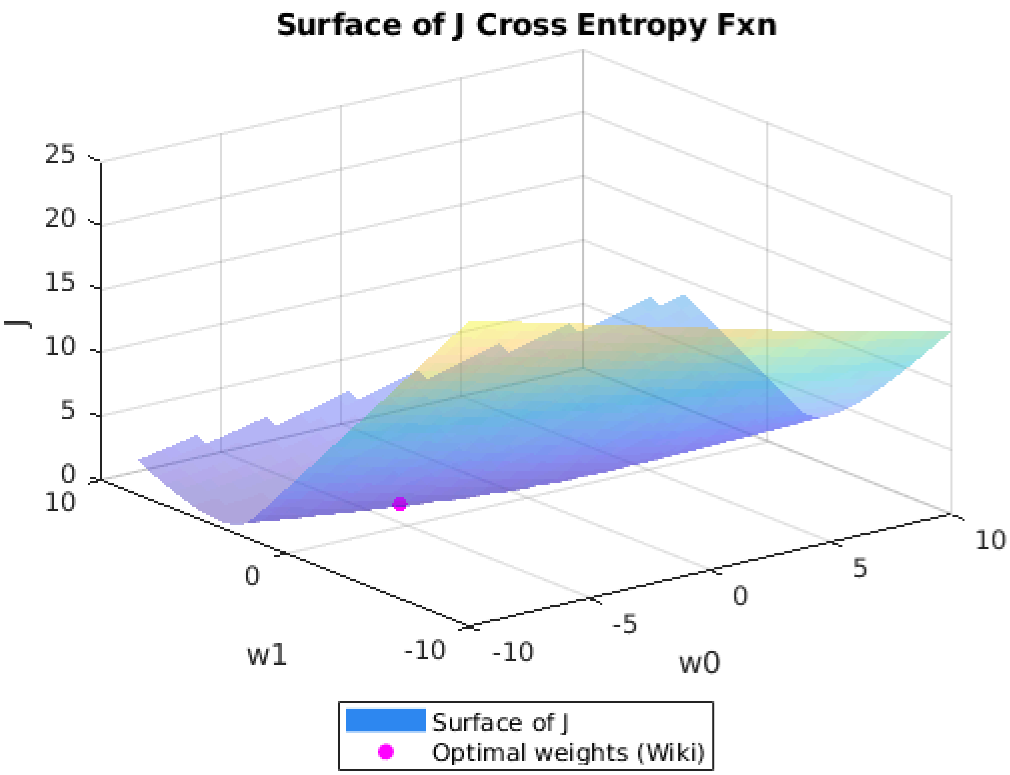
```
disp('Final "camp" coords [w0, w1] with learning rate 2: ')
disp(camp_coords(end,:));

gd1 = plot(camp_coords(:,1),camp_coords(:,2));
gd1.LineWidth = 1;
gd1.LineStyle = '--';
gd1.Color = [0.25 0.25 0.25];
gd1.MarkerSize = 4;
gd1.Marker = 'o';
gd1.MarkerEdgeColor = 'black';
gd1.MarkerFaceColor = 'black';
scatter(wiki_w0, wiki_w1, 30,'magenta','filled');

xtext = [wiki_w0 - 1.7, init_camp(1) + .25];
ytext = [wiki_w1 + 1, init_camp(2) + .15];
str = {'\bfw* \downarrow ', '\leftarrow \bfw^0'};
text(xtext,ytext,str)

legend({'\nablaJ','Gradient Descent Path','Optimum weights (Wiki)'})
hold off;

figure();
% surf(W0, W1, J,'EdgeColor','none', 'FaceAlpha', 0.3);
hold on;
contour(W0, W1, J);
title({'Contour Plot of J','Gradient Descent, LR = 1.745, 100 iter'});
xlabel('w0')
ylabel('w1')
zlabel('J')
axis square
```

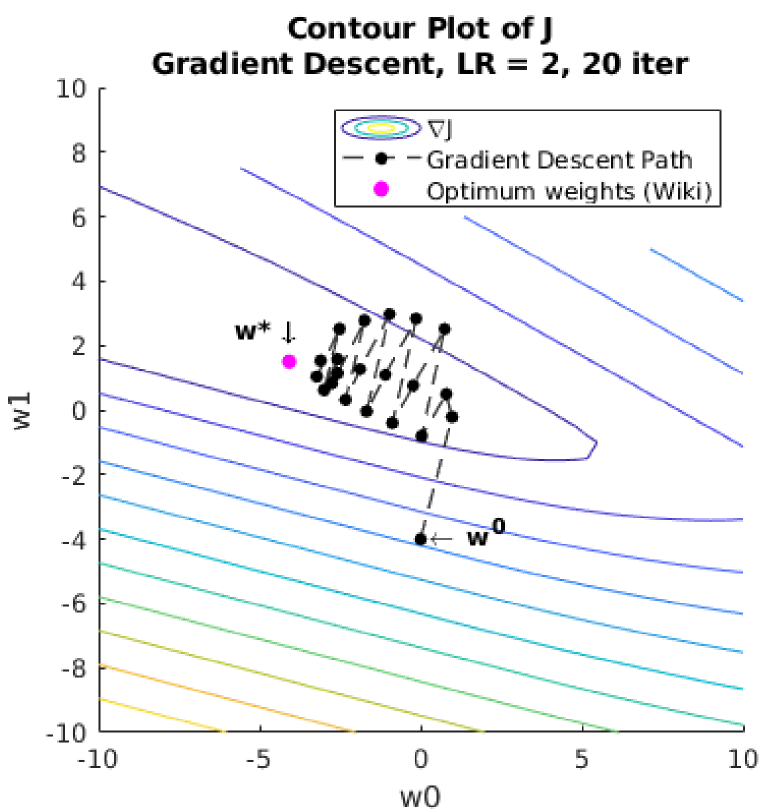
Part 3. Performing Gradient Descent on J (learning rate = best for 100 iterations)

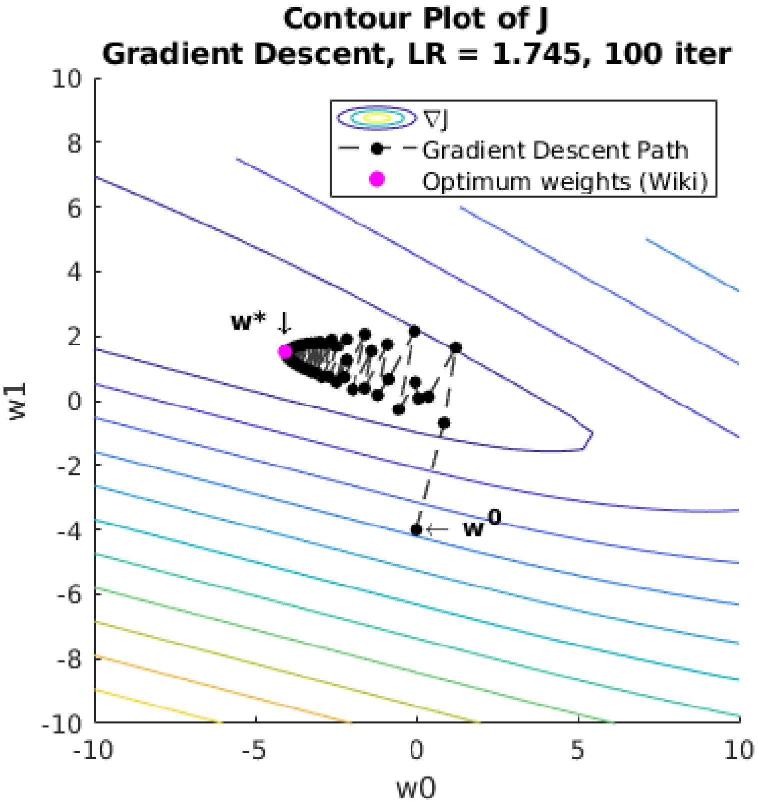
```
dL = 1.745; % learning rate
gd_iter_max = 100;
init_camp = [0, -4];
camp_coords = zeros([gd_iter_max+1, 2]);
camp_coords(1,:) = init_camp;

for gd_iter = 1:1:gd_iter_max
    w0_current = camp_coords(gd_iter, 1);
    w1_current = camp_coords(gd_iter, 2);
    delJ_w0 = 0;
    delJ_w1 = 0;
    for i = 1:N
        delJ_w0 = delJ_w0 + (-pass_fail(i)) * (1/(1 + exp(w0_current
 + w1_current * hours(i)))) + (1 - pass_fail(i))* (1/(1 + exp(-
w0_current - w1_current * hours(i))));
        delJ_w1 = delJ_w1 + hours(i) * (-pass_fail(i) * (1/(1 +
 exp(w0_current + w1_current * hours(i)))) + (1 - pass_fail(i))* (1/(1
 + exp(-w0_current - w1_current * hours(i)))));
```

```
    end
    delJ_w0 = (1/N) * delJ_w0;
    delJ_w1 = (1/N) * delJ_w1;
    camp_coords(gd_iter + 1, :) = [(w0_current - dL * delJ_w0),
 (w1_current - dL * delJ_w1)];
end

disp('Final "camp" coords [w0, w1] with learning rate 1.745: ')
disp(camp_coords(end,:));
gd2 = plot(camp_coords(:,1),camp_coords(:,2));
gd2.LineWidth = 1;
gd2.LineStyle = '--';
gd2.Color = [0.25 0.25 0.25];
gd2.MarkerSize = 4;
gd2.Marker = 'o';
gd2.MarkerEdgeColor = 'black';
gd2.MarkerFaceColor = 'black';
scatter(wiki_w0, wiki_w1, 30,'magenta','filled');

xtext2 = [wiki_w0 - 1.7, init_camp(1) + .25];
ytext2 = [wiki_w1 + 1, init_camp(2) + .15];
str2 = {'\bfw* \downarrow ', '\leftarrow \bfw^0'};
text(xtext2,ytext2,str2)

legend({'\nablaJ','Gradient Descent Path','Optimum weights (Wiki)'})
hold off;
```

```
Final "camp" coords [w0, w1] with learning rate 2:
   -3.1025    1.5374

Final "camp" coords [w0, w1] with learning rate 1.745:
   -4.0498    1.5100
```


Surface of J Cross Entropy Fxn
J
w1
w0
Surface of J
Optimal weights (Wiki)


Contour Plot of J
Gradient Descent, LR = 2, 20 iter
∇J
Gradient Descent Path
Optimum weights (Wiki)
w*
w0
w1
w0



Published with MATLAB® R2021a

```
clear all, close all, clc;

% opts = detectImportOptions('biopsy_data_missing_values.csv',
 'NumHeaderLines', 1);
% preview('biopsy_data_missing_values.csv', opts)

A = readtable('biopsy_data_missing_values.csv', 'NumHeaderLines', 1);
```

Part A - Data Formatting and Cleaning

```
var2_mt = find(strcmp(A.Var2, ''));
for i = 1:length(var2_mt)
    ii = var2_mt(i);
    A.Var2{ii} = 'Irregular';
end

var1_nan = find(~isfinite(A.Var1));
for i = var1_nan
    A.Var1(i) = i;
end

disp(A(:,1:2))
```

Part B - Naive Bayes

```
new_data = table();
new_data.Var1 = [1; 2; 3; 4; 5];
new_data.Var2 =
 {'Irregular'; 'Irregular'; 'Circle'; 'Circle'; 'Triangle'};
new_data.Var3 = {'Large'; 'Small'; 'Large'; 'Large'; 'Large'};
new_data.Var4 = {'Convex'; 'Flat'; 'Concave'; 'Convex'; 'Concave'};
new_data.Var5 = {'Rough'; 'Rough'; 'Smooth'; 'Smooth'; 'Smooth'};
new_data.Var6 = {'Neutral'; 'Red'; 'Neutral'; 'Dark'; 'Neutral'};

mal_inds = find(strcmp(A.Var7, 'Malignant'));
ben_inds = find(strcmp(A.Var7, 'Benign'));

NewBiopProbData = struct();
for biop = 1:height(new_data)

    sample = new_data(biop, :);

    BiopProbs = struct();
    temp_mal_probs = [];
    temp_ben_probs = [];

    for var_num = 2:length(sample.Properties.VariableNames)

        var = string(sample.Properties.VariableNames(var_num));
        biop_var_val = string(table2array(sample(:,var_num)));
```

```
        mal_cond_inds = find(strcmp(A.Var7, 'Malignant') & strcmp(A.
(var), biop_var_val));
        p_var_giv_mal = length(mal_cond_inds)/length(mal_inds);
        temp_mal_probs = [temp_mal_probs, p_var_giv_mal];
        BiopProbs.(strcat('P_of_', biop_var_val, '_given_Mal')) =
 p_var_giv_mal;

        ben_cond_inds = find(strcmp(A.Var7, 'Benign') & strcmp(A.
(var), biop_var_val));
        p_var_giv_ben = length(ben_cond_inds)/length(ben_inds);
        temp_ben_probs = [temp_ben_probs, p_var_giv_ben];
        BiopProbs.(strcat('P_of_', biop_var_val, '_given_Ben')) =
 p_var_giv_ben;

    end
    p_mal = length(mal_inds)/height(A);
    temp_mal_probs = [temp_mal_probs, p_mal];

    p_ben = length(ben_inds)/height(A);
    temp_ben_probs = [temp_ben_probs, p_ben];

    BiopProbs.('P_big_pos_Mal') = prod(temp_mal_probs);
    BiopProbs.('P_big_neg_Ben') = prod(temp_ben_probs);
    BiopProbs.('log_P_big_pos_Mal') = log(prod(temp_mal_probs));
    BiopProbs.('log_P_big_neg_Ben') = log(prod(temp_ben_probs));

    if BiopProbs.('P_big_pos_Mal') > BiopProbs.('P_big_neg_Ben')
        BiopProbs.('Predicted_Class') = {'Malignant'};
    elseif BiopProbs.('P_big_pos_Mal') < BiopProbs.('P_big_neg_Ben')
        BiopProbs.('Predicted_Class') = {'Benign'};
    else
        BiopProbs.('Predicted_Class') = {'Inconclusive'};
    end

    NewBiopProbData.(['biop' num2str(biop)]) = BiopProbs;
    clear BiopProbs temp_mal_probs temp_ben_probs;

end

fields = fieldnames(NewBiopProbData);
for biop_num = 1:length(fields)
    label = fields(biop_num);
    biop_prob_set = NewBiopProbData.(['biop' num2str(biop_num)]);

    disp(label)
    disp(biop_prob_set)
end

    Var1        Var2
    ____    _____________

      1     {'Circle'   }
      2     {'Circle'   }
      3     {'Circle'   }
```

```
  4     {'Irregular'}
  5     {'Circle'   }
  6     {'Circle'   }
  7     {'Circle'   }
  8     {'Irregular'}
  9     {'Triangle' }
 10     {'Circle'   }
 11     {'Irregular'}
 12     {'Irregular'}

{'biop1'}

P_of_Irregular_given_Mal: 0.1667
P_of_Irregular_given_Ben: 0.5000
    P_of_Large_given_Mal: 0.8333
    P_of_Large_given_Ben: 0.8333
   P_of_Convex_given_Mal: 0.1667
   P_of_Convex_given_Ben: 0.3333
    P_of_Rough_given_Mal: 0.5000
    P_of_Rough_given_Ben: 0.1667
  P_of_Neutral_given_Mal: 0.1667
  P_of_Neutral_given_Ben: 0.3333
           P_big_pos_Mal: 9.6451e-04
           P_big_neg_Ben: 0.0039
       log_P_big_pos_Mal: -6.9439
       log_P_big_neg_Ben: -5.5576
         Predicted_Class: {'Benign'}

{'biop2'}

P_of_Irregular_given_Mal: 0.1667
P_of_Irregular_given_Ben: 0.5000
    P_of_Small_given_Mal: 0.1667
    P_of_Small_given_Ben: 0.1667
     P_of_Flat_given_Mal: 0.3333
     P_of_Flat_given_Ben: 0.1667
    P_of_Rough_given_Mal: 0.5000
    P_of_Rough_given_Ben: 0.1667
      P_of_Red_given_Mal: 0.1667
      P_of_Red_given_Ben: 0.3333
           P_big_pos_Mal: 3.8580e-04
           P_big_neg_Ben: 3.8580e-04
       log_P_big_pos_Mal: -7.8602
       log_P_big_neg_Ben: -7.8602
         Predicted_Class: {'Inconclusive'}

{'biop3'}

 P_of_Circle_given_Mal: 0.8333
 P_of_Circle_given_Ben: 0.3333
  P_of_Large_given_Mal: 0.8333
  P_of_Large_given_Ben: 0.8333
P_of_Concave_given_Mal: 0.5000
P_of_Concave_given_Ben: 0.5000
```

```
 P_of_Smooth_given_Mal: 0.5000
 P_of_Smooth_given_Ben: 0.8333
P_of_Neutral_given_Mal: 0.1667
P_of_Neutral_given_Ben: 0.3333
         P_big_pos_Mal: 0.0145
         P_big_neg_Ben: 0.0193
     log_P_big_pos_Mal: -4.2358
     log_P_big_neg_Ben: -3.9482
       Predicted_Class: {'Benign'}

{'biop4'}

P_of_Circle_given_Mal: 0.8333
P_of_Circle_given_Ben: 0.3333
 P_of_Large_given_Mal: 0.8333
 P_of_Large_given_Ben: 0.8333
P_of_Convex_given_Mal: 0.1667
P_of_Convex_given_Ben: 0.3333
P_of_Smooth_given_Mal: 0.5000
P_of_Smooth_given_Ben: 0.8333
  P_of_Dark_given_Mal: 0.6667
  P_of_Dark_given_Ben: 0.3333
        P_big_pos_Mal: 0.0193
        P_big_neg_Ben: 0.0129
    log_P_big_pos_Mal: -3.9482
    log_P_big_neg_Ben: -4.3536
      Predicted_Class: {'Malignant'}

{'biop5'}

P_of_Triangle_given_Mal: 0
P_of_Triangle_given_Ben: 0.1667
   P_of_Large_given_Mal: 0.8333
   P_of_Large_given_Ben: 0.8333
 P_of_Concave_given_Mal: 0.5000
 P_of_Concave_given_Ben: 0.5000
  P_of_Smooth_given_Mal: 0.5000
  P_of_Smooth_given_Ben: 0.8333
 P_of_Neutral_given_Mal: 0.1667
 P_of_Neutral_given_Ben: 0.3333
          P_big_pos_Mal: 0
          P_big_neg_Ben: 0.0096
      log_P_big_pos_Mal: -Inf
      log_P_big_neg_Ben: -4.6413
        Predicted_Class: {'Benign'}
```

Published with MATLAB® R2021a

Table of Contents


```
clear all; close all; clc;
```

Part A

```
opts = detectImportOptions('Iris_dataset.csv', 'NumHeaderLines', 1);
% preview('Iris_dataset.csv', opts)
A = readtable('Iris_dataset.csv', 'NumHeaderLines', 1);
```

Part B

```
for var_ct = 1:length(table2array(A(1,1:end-1)))
    B(:,var_ct) = eval(['A.Var' num2str(var_ct)]);
end

species = unique(eval(['A.Var' num2str(var_ct+1)]));
species_instances = eval(['A.Var' num2str(var_ct+1)]);
for instance = 1:length(species_instances)
    onehot_spec = find(string(species_instances{instance}) ==
 species);
    B(instance,var_ct+1) = onehot_spec;
end

seto_inds = find(B(:,end) == find(strcmp(string(species), 'Iris-
setosa')));
vers_inds = find(B(:,end) == find(strcmp(string(species), 'Iris-
versicolor')));
virg_inds = find(B(:,end) == find(strcmp(string(species), 'Iris-
virginica')));

plot_switch = 0;

% xmin = 0;
% xmax = 10;
```

```
% ymin = 0;
% ymax = 10;

% My plots are really not legible with the axis set to min/max = 0/10
xmin = 3;
xmax = 9;
ymin = 0;
ymax = 5;

figure();
scatter(B(seto_inds, 1), B(seto_inds, 4), 'red');
hold on;
scatter(B(vers_inds, 1), B(vers_inds, 4), 'yellow');
scatter(B(virg_inds, 1), B(virg_inds, 4), 'green');
title('Iris Data')
xlabel('Sepal Length')
ylabel('Petal Width')

axis([xmin xmax ymin ymax]);
legend({'Setosa','Versicolor','Virginica'})
hold off;
```

Gaussian Mixtures

```
sig1 = 0.2;
sig2 = sig1;

seto_total = length(seto_inds);
vers_total = length(vers_inds);
virg_total = length(virg_inds);
spec_total = length(B(:,end));

x1 = xmin: 0.1 :xmax;
x4 = ymin: 0.1 :ymax;

[X1, X4] = meshgrid(x1, x4);
```

Setosa

```
p_seto = seto_total/spec_total;

p_x1_giv_seto = p_x_giv_c(X1, B(seto_inds,1), sig1);
p_x4_giv_seto = p_x_giv_c(X4, B(seto_inds,4), sig2);

p_seto_giv_x = p_x1_giv_seto .* p_x4_giv_seto .* p_seto;
```

Versicolor

```
p_vers = vers_total/spec_total;

p_x1_giv_vers = p_x_giv_c(X1, B(vers_inds,1), sig1);
p_x4_giv_vers = p_x_giv_c(X4, B(vers_inds,4), sig2);
```

```
p_vers_giv_x = p_x1_giv_vers .* p_x4_giv_vers .* p_vers;
```

Virginica

```
p_virg = virg_total/spec_total;

p_x1_giv_virg = p_x_giv_c(X1, B(virg_inds,1), sig1);
p_x4_giv_virg = p_x_giv_c(X4, B(virg_inds,4), sig2);

p_virg_giv_x = p_x1_giv_virg .* p_x4_giv_virg .* p_virg;
```

Plotting

```
figure();
hold on;
contour_levels = [0:0.03:0.15];

contour(X1, X4, p_seto_giv_x, contour_levels, 'red');
contour(X1, X4, p_vers_giv_x, contour_levels, 'yellow');
contour(X1, X4, p_virg_giv_x, contour_levels, 'green');

scatter(B(seto_inds, 1), B(seto_inds, 4), 'red');
scatter(B(vers_inds, 1), B(vers_inds, 4), 'yellow');
scatter(B(virg_inds, 1), B(virg_inds, 4), 'green');

title('Contours of Gaussian Mixture')
xlabel('Sepal Length')
ylabel('Petal Width')
axis square
```

Part C

```
new_data_x1 = [5.5; 7; 6.5; 6.2];
new_data_x4 = [0.5; 1.8; 1.5; 1.7];

new_data_probs = zeros([length(new_data_x1), length(species)]);

clrs=['b','c','m','k'];
for new_sample_num = 1:length(new_data_x1)
    sample_x1 = new_data_x1(new_sample_num);
    sample_x4 = new_data_x4(new_sample_num);
```

Setosa

```
    p_x1_giv_seto_sample = p_x_giv_c(sample_x1, B(seto_inds,1), sig1);
    p_x4_giv_seto_sample = p_x_giv_c(sample_x4, B(seto_inds,4), sig2);
```

Versicolor

```
    p_x1_giv_vers_sample = p_x_giv_c(sample_x1, B(vers_inds,1), sig1);
```

```
    p_x4_giv_vers_sample = p_x_giv_c(sample_x4, B(vers_inds,4), sig2);
```

Virginica

```
    p_x1_giv_virg_sample = p_x_giv_c(sample_x1, B(virg_inds,1), sig1);
    p_x4_giv_virg_sample = p_x_giv_c(sample_x4, B(virg_inds,4), sig2);
```

Combined

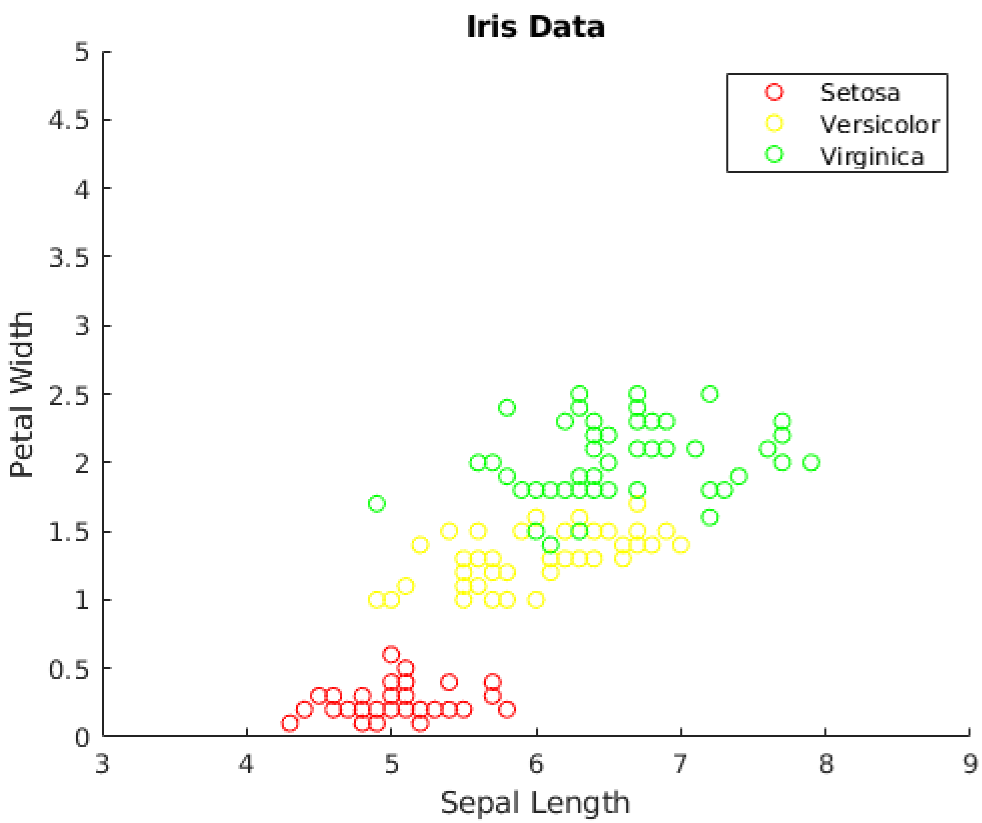
```
    p_seto_giv_x_sample = p_x1_giv_seto_sample .*
 p_x4_giv_seto_sample .* p_seto;
    p_vers_giv_x_sample = p_x1_giv_vers_sample .*
 p_x4_giv_vers_sample .* p_vers;
    p_virg_giv_x_sample = p_x1_giv_virg_sample .*
 p_x4_giv_virg_sample .* p_virg;

    new_data_probs(new_sample_num, :) = [p_seto_giv_x_sample,
 p_vers_giv_x_sample, p_virg_giv_x_sample];

    pt_color = clrs(new_sample_num);
    scatter(sample_x1, sample_x4, 50, pt_color, '^','filled')
end

legend({'Setosa Contour','Versicolor Contour',...
    'Virginica Contour','Setosa pts','Versicolor pts',...
    'Virginica pts','Sample 1','Sample 2','Sample 3','Sample 4'},...
    'location','best','NumColumns',3)
% legend('boxoff');
legend('Orientation','horizontal')
hold off;

[m, index] = max(new_data_probs, [], 2);

for sample_num = 1:length(new_data_probs(:,1))
    formatSpec = 'With a probability of %0.3f, Sample %d is most
 likely %s';
    prob_ans = m(sample_num);
    class_ans = string(species{index(sample_num)});
    str = sprintf(formatSpec, prob_ans, sample_num, class_ans);
    disp(str)
end
```

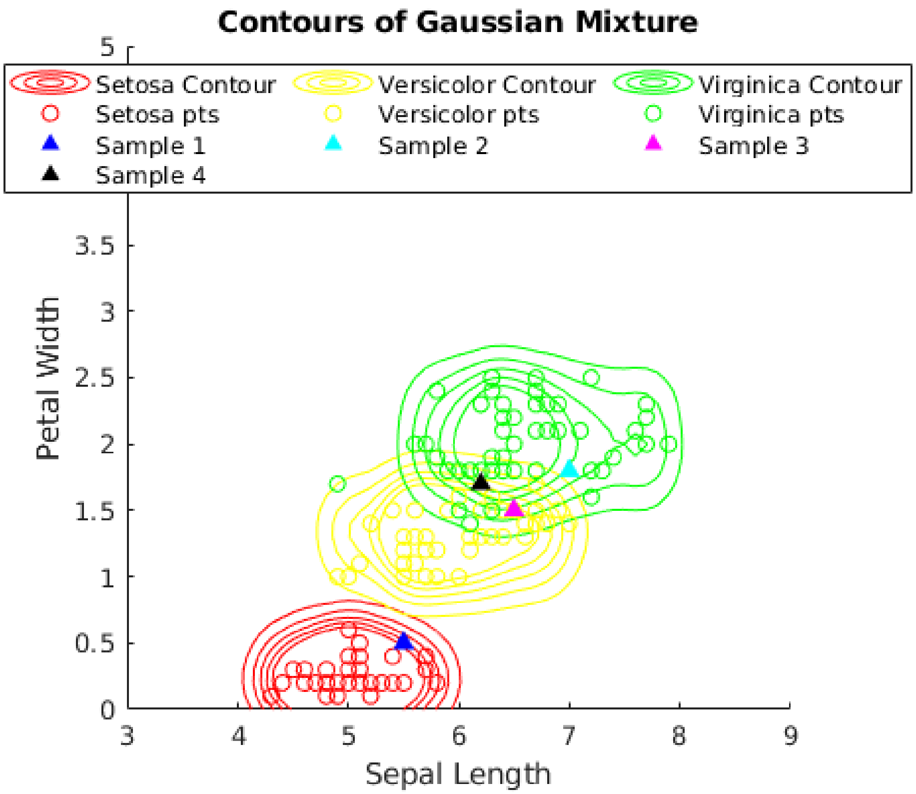
Functions

```
function p = p_x_giv_c(att_mesh, att_vals, sigma)

k = 1/(sigma*sqrt(2*pi));
m = length(att_vals);
p = 0;

for instance = 1:length(att_vals)
```

```
    p = p + exp((-1/(2*(sigma^2))).*(att_mesh -
 att_vals(instance)).^2);
end

p = p*k/m;

end
```

With a probability of 0.130, Sample 1 is most likely Iris-setosa
With a probability of 0.114, Sample 2 is most likely Iris-virginica
With a probability of 0.160, Sample 3 is most likely Iris-versicolor
With a probability of 0.159, Sample 4 is most likely Iris-virginica





Published with MATLAB® R2021a

HW2_P4_Jackson_Liam

March 26, 2021

0.1 HW2 Problem 4

0.2 Name: Liam Jackson

0.2.1 Imports

[1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import axes3d
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
↪roc_auc_score
```

0.2.2 1. Logistic regression

a.

[2]:
```
raw_data_df = pd.read_csv('data.csv')

radius_mean = raw_data_df['radius_mean'].to_numpy().reshape(-1,1)
diag_str = raw_data_df['diagnosis'].to_numpy()
diag = diag_str.copy()

for ind, diag_el in enumerate(diag_str):
    if str(diag_el) == 'M':
        diag[ind] = 1
    elif str(diag_el) == 'B':
        diag[ind] = 0
```

b.

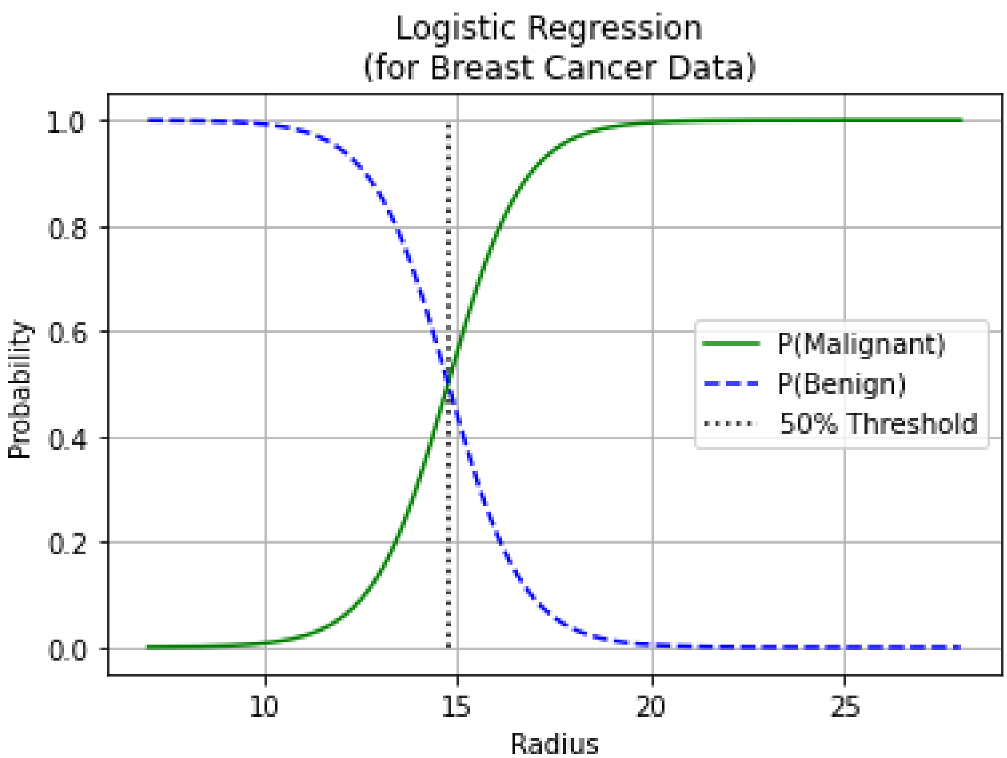
[3]:
```
log_reg = LogisticRegression()
log_reg.fit(radius_mean, diag_str)

rad_min = round(np.min(radius_mean))
rad_max = round(np.max(radius_mean))
rad_step = (rad_max - rad_min) * 100
rad_new = np.linspace(rad_min, rad_max, rad_step).reshape(-1,1)
```

```
diag_pred_prob = log_reg.predict_proba(rad_new)

fifty_ind = np.where((diag_pred_prob[:,0] >= .495) & (diag_pred_prob[:,0] <= .
 ↪505))
dec_bnd = np.mean(rad_new[fifty_ind,0])

print(f"The decision boundary is {dec_bnd} um")
```

```
The decision boundary is 14.75869461648404 um
```

c.

[4]:
```
plt.plot(rad_new, diag_pred_prob[:,1], "g-", label= "P(Malignant)")
plt.plot(rad_new, diag_pred_prob[:,0], "b--", label= "P(Benign)")
plt.vlines(dec_bnd, 0, 1, colors = 'k', linestyles = 'dotted', label = '50%␣
 ↪Threshold')
# plt.scatter(radius_mean, diag, label = 'Biopsies')
plt.title('Logistic Regression \n (for Breast Cancer Data)')
plt.xlabel('Radius')
plt.ylabel('Probability')
plt.legend()
plt.grid()
plt.show()
```



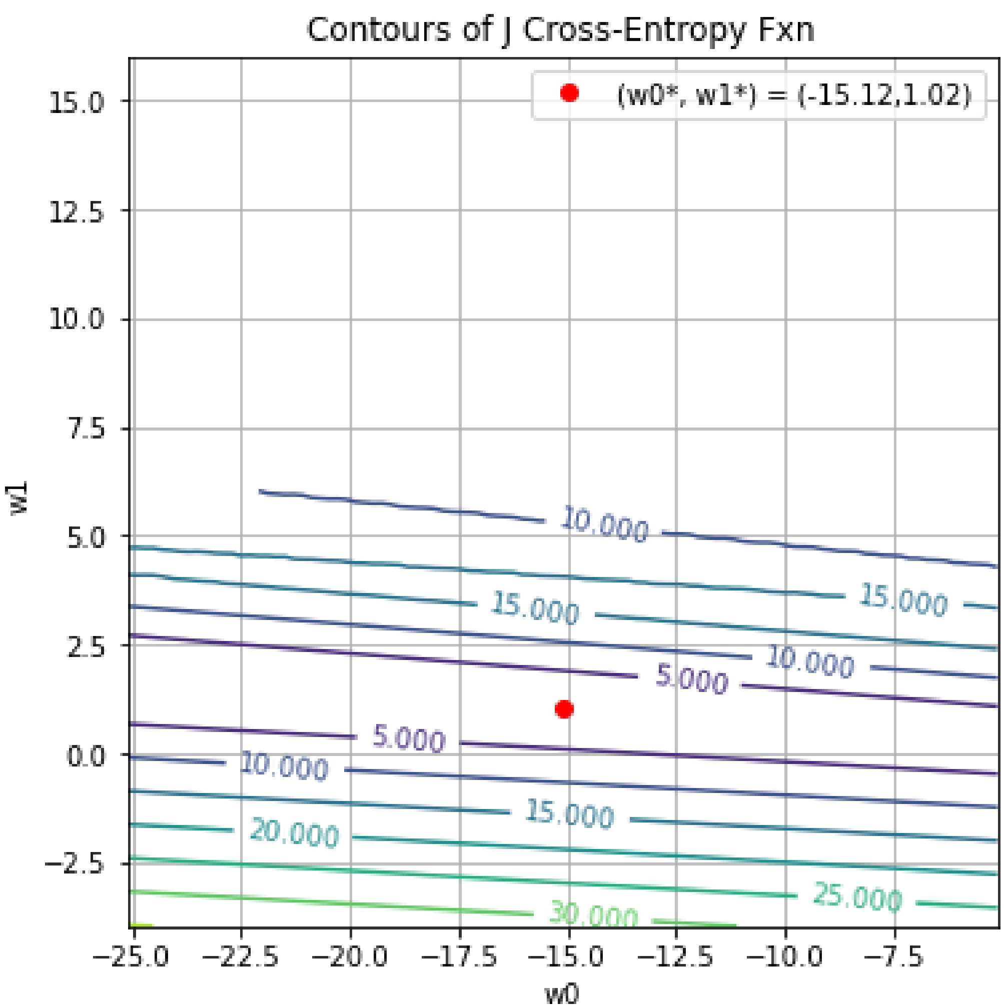
0.2.3 2. Cost function plot

a.

[5]:
```
w0_fitted = log_reg.intercept_
w1_fitted = log_reg.coef_[0]

w0_bnd = 10
w1_bnd = 5

w0_range = np.linspace(w0_fitted-w0_bnd, w0_fitted+w0_bnd, 100).squeeze()
w1_range = np.linspace(w1_fitted-w1_bnd, w1_fitted+w1_bnd, 100).squeeze()

W0, W1 = np.meshgrid(w0_range, w1_range)

N = radius_mean.shape[0]
J = np.zeros(W0.shape)

def sigmoid(x, w0_internal, w1_internal):
    S = np.zeros(len(x))
    for i in range(0,len(x)):
        S_temp = 1/(1+np.exp(-(w0_internal + w1_internal*x[i])))
        if S_temp == 0:
            S_temp = 0.000001
        elif S_temp == 1:
            S_temp = .999999
        S[i] = S_temp
    return S

for w0_ind, w0 in enumerate(w0_range):
    for w1_ind, w1 in enumerate(w1_range):

        fx = sigmoid(radius_mean, w0, w1)

        J_temp = 0
        for i in range(N):
            yi = diag[i]
            fxi = fx[i]
            J_temp += (yi*np.log(fxi) + (1 - yi)*np.log(1-fxi))

        J[w1_ind, w0_ind] = (-1/N)*J_temp
```

[6]:
```
plt.figure(figsize = (6,6))

CS = plt.contour(W0,W1,J) #, 20, colors = 'k')
plt.clabel(CS, inline = True, fontsize = 10)
```

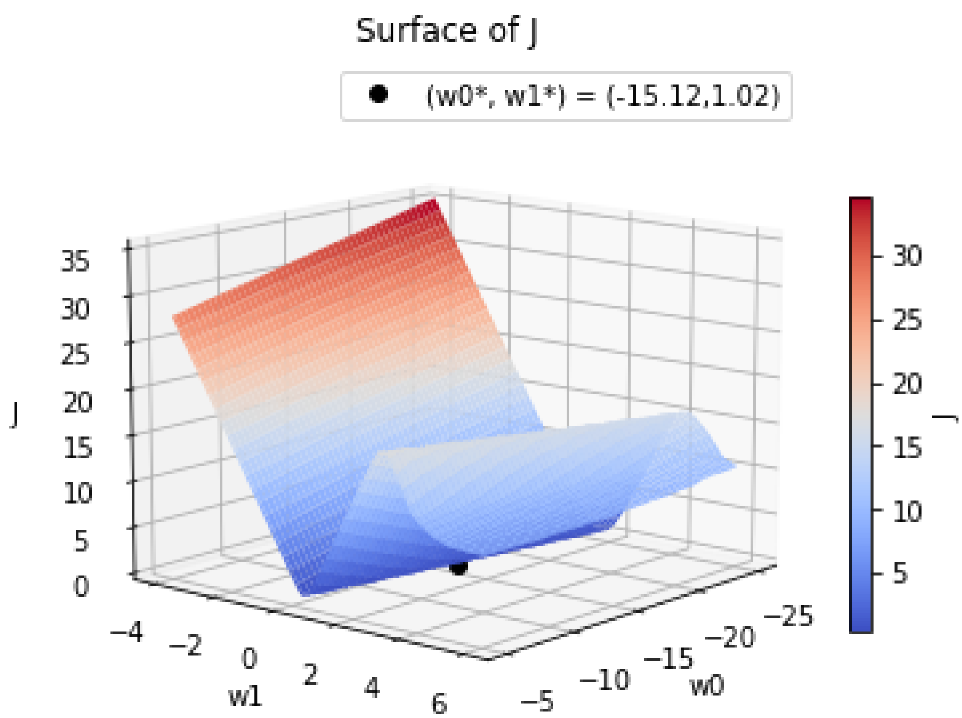
```
plt.plot(w0_fitted, w1_fitted, 'ro',
         label = f'(w0*, w1*) = ({"{:.2f}".format(w0_fitted[0])},{"{:.2f}".
 ↪format(w1_fitted[0])})')
plt.title('Contours of J Cross-Entropy Fxn')
plt.axis('square')
plt.xlabel('w0')
plt.ylabel('w1')
plt.legend()
plt.grid()
plt.show()
```



b.

```
[7]: fig = plt.figure(figsize = (6,6))
     ax = fig.add_subplot(111,projection='3d')
```

```
surf = ax.plot_surface(W0, W1, J, cmap = cm.coolwarm, linewidth = 20)
point = ax.plot(w0_fitted, w1_fitted, 'ko',
                label = f'(w0*, w1*) = ({"{:.2f}".format(w0_fitted[0])},{"{:.
↪2f}".format(w1_fitted[0])})')
fig.colorbar(surf, shrink = 0.5, label = 'J')

ax.set_title('Surface of J')
ax.set_xlabel('w0')
ax.set_ylabel('w1')
ax.set_zlabel('J')
plt.legend()

ax.view_init(12, 40)
plt.show()
```

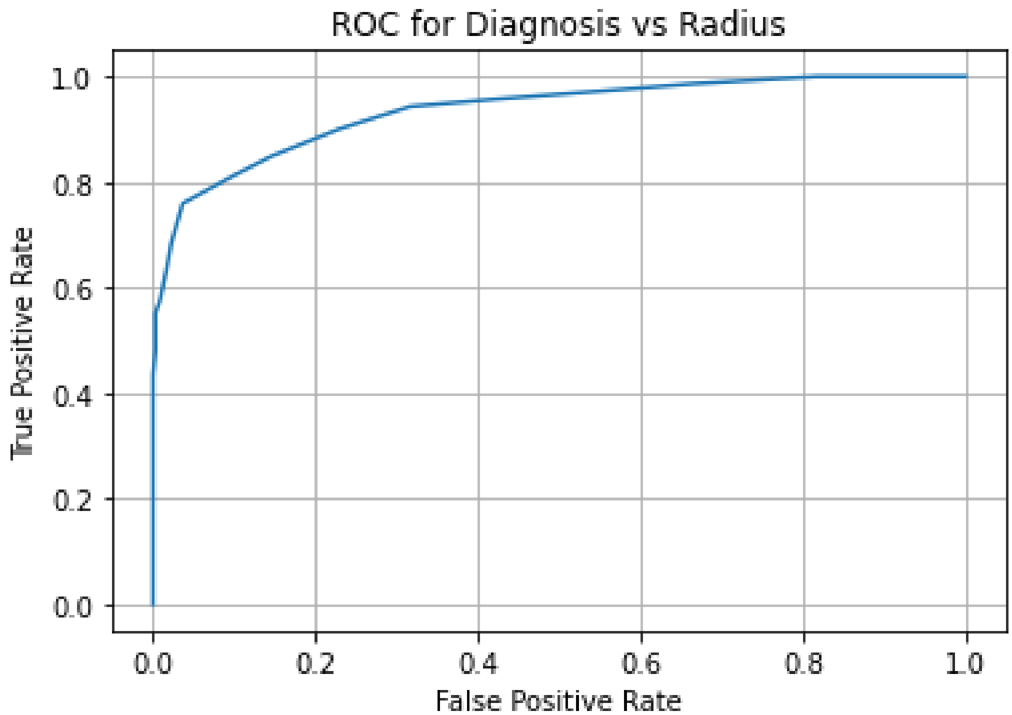


0.2.4 3. ROC

a.

[34]:
```
thresh_range = np.arange(5,30.5,.5)

roc = np.zeros([len(thresh_range), 3])
roc[:,0] = thresh_range
```

```
for thresh_ind, thresh in enumerate(thresh_range):
    tp_roc = 0
    fp_roc = 0
    tn_roc = 0
    fn_roc = 0
    for instance, radius in enumerate(radius_mean):
        if radius >= thresh and diag[instance] == 1:
            tp_roc += 1
        elif radius >= thresh and diag[instance] == 0:
            fp_roc += 1
        elif radius <= thresh and diag[instance] == 0:
            tn_roc += 1
        elif radius <= thresh and diag[instance] == 1:
            fn_roc += 1

    tp_fn = tp_roc + fn_roc
    tn_fp = tn_roc + fp_roc

    specificity = tn_roc / (tn_fp)
    fpr = 1 - specificity
    tpr = tp_roc / tp_fn
    roc[thresh_ind, 1:] = [fpr, tpr]

plt.plot(roc[:,1],roc[:,2])
plt.title('ROC for Diagnosis vs Radius')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.show()
```



b.

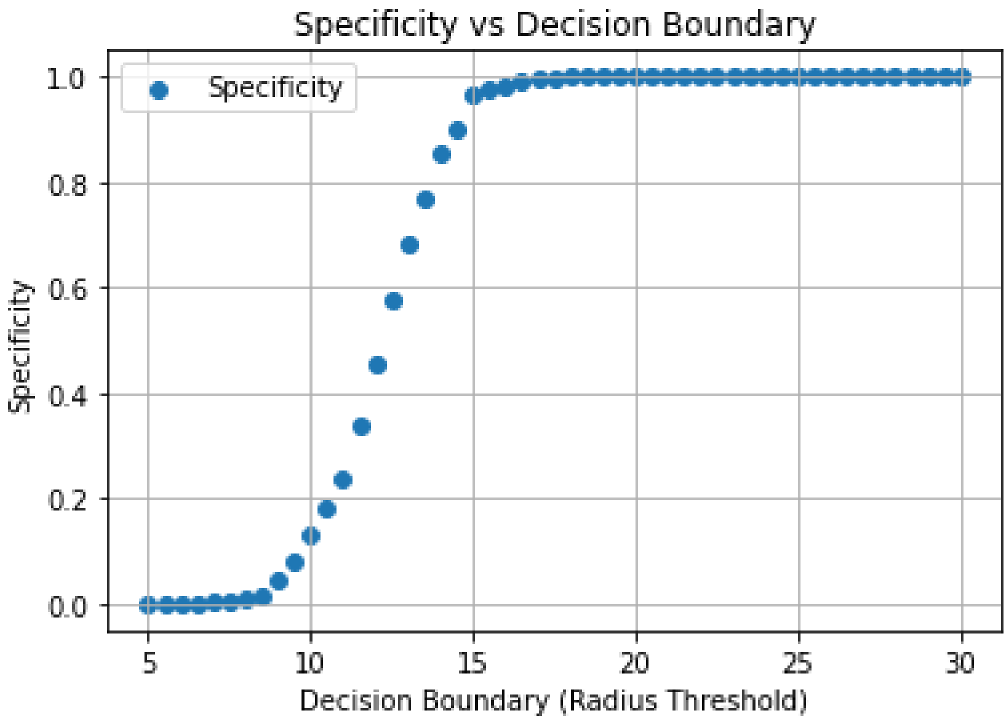
[24]:
```
# dint64 = diag.astype('int64')
# dpint64 = log_reg.predict_proba(radius_mean)[:,1].astype('int64')
# auc = roc_auc_score(dint64, dpint64)

zero = np.array([0])
fpr = roc[:,1]
tpr = roc[:,2]
tpr_diff = np.hstack((np.diff(tpr), zero))
fpr_diff = np.hstack((np.diff(fpr), zero))
auc = abs(np.dot(tpr, fpr_diff) + np.dot(tpr_diff, fpr_diff) / 2)

print(f'The AUC for the above ROC curve is: {auc}')
```

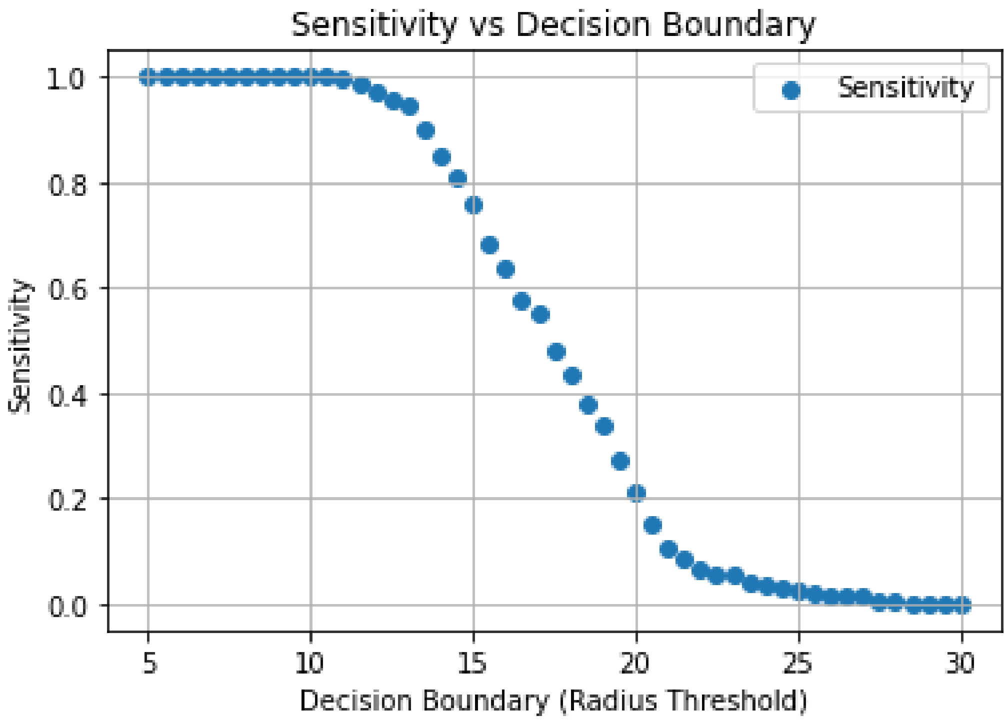
```
The AUC for the above ROC curve is: 0.9352174832197029
```

c.

[27]:
```
plt.scatter(roc[:,0],1-roc[:,1], label = 'Specificity')
plt.title('Specificity vs Decision Boundary')
plt.xlabel('Decision Boundary (Radius Threshold)')
plt.ylabel('Specificity')
plt.grid()
plt.legend()
```

```
plt.show()

plt.scatter(roc[:,0],roc[:,2], label = 'Sensitivity')
plt.title('Sensitivity vs Decision Boundary')
plt.xlabel('Decision Boundary (Radius Threshold)')
plt.ylabel('Sensitivity')
plt.grid()
plt.legend()
plt.show()
```





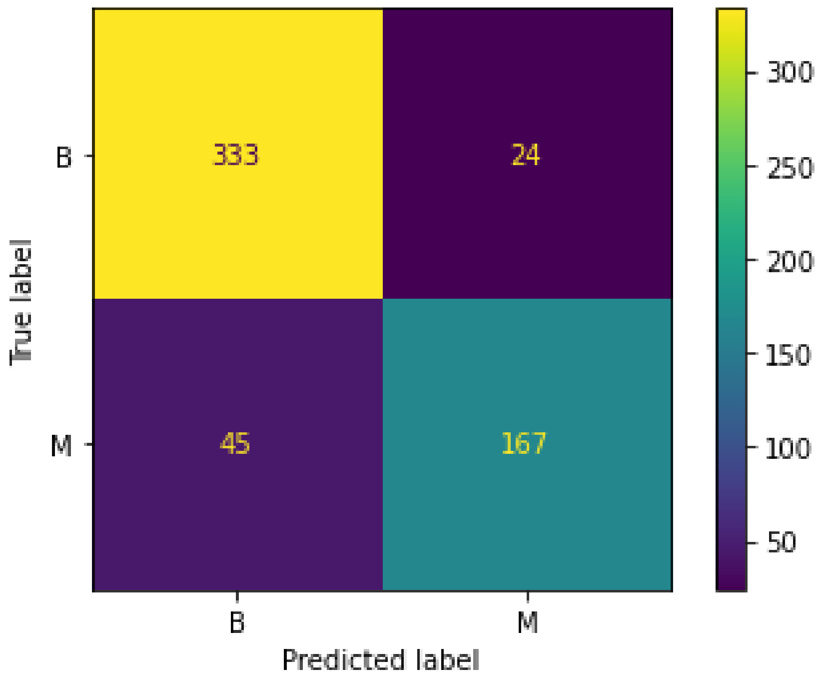
0.2.5 4. Confusion matrix

a.

[28]:
```
diag_pred = log_reg.predict(radius_mean)
diag_real = diag_str

conf_mtx = confusion_matrix(diag_real, diag_pred)
# "the count of true negatives is 0,0, false negatives is 1,0,
# true positives is 1,1 and false positives is 0,1"
tp = conf_mtx[1,1]
fp = conf_mtx[0,1]
tn = conf_mtx[0,0]
fn = conf_mtx[1,0]
print(f'tp = {tp} \nfp = {fp} \ntn = {tn} \nfn = {fn}')

disp = ConfusionMatrixDisplay(confusion_matrix=conf_mtx, display_labels=log_reg.
 ↪classes_)

disp.plot()
```

```
tp = 167
fp = 24
tn = 333
fn = 45
```

[28]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f205f3bd670>



b.

[29]:
```
print(f'The sensitivity for the optimized model is: {tp/(tp+fn)}')
print(f'The specificity for the optimized model is: {tn/(tn+fp)}')
```

```
The sensitivity for the optimized model is: 0.7877358490566038
The specificity for the optimized model is: 0.9327731092436975
```