Table of Contents

```
clc;
clear all;
close all;

N = 200;
```

Class definitions

Class 0 definition

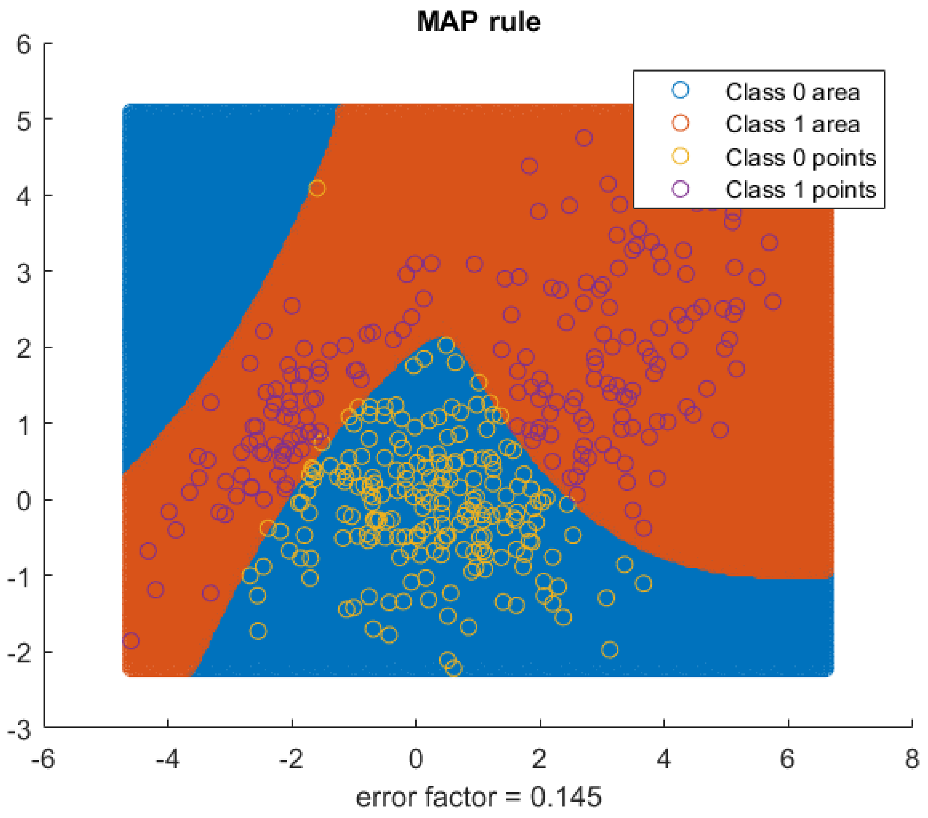
```
theta1 = 0;
m = [0 0]';
lambda_1 = 2;
lambda_2 = 1;
u_1 = [cos(theta1) sin(theta1)]';
u_2 = [-sin(theta1) cos(theta1)]';
C1 = [u_1 u_2]*diag([lambda_1,lambda_2])*inv([u_1 u_2]);
C1_pts = mvnrnd(m,C1,N);

% Class 1 definition
theta2_a = -3*pi/4;
m_a = [-2 1]';
pi_a = 1/3;
lambda_a1 = 2;
lambda_a2 = 1/4;
u1_a = [cos(theta2_a) sin(theta2_a)]';
u2_a = [-sin(theta2_a) cos(theta2_a)]';
C_a = [u1_a u2_a]*diag([lambda_a1,lambda_a2])*inv([u1_a u2_a]);

theta2_b = pi/4;
pi_b = 2/3;
m_b = [3 2]';
lambda_b1 = 3;
lambda_b2 = 1;
u1_b = [cos(theta2_b) sin(theta2_b)]';
u2_b = [-sin(theta2_b) cos(theta2_b)]';
C_b = [u1_b u2_b]*diag([lambda_b1,lambda_b2])*inv([u1_b u2_b]);
C2(:,:,1) = C_a;
C2(:,:,2) = C_b;
gm = gmdistribution([m_a';m_b'],C2,[pi_a pi_b]);
```

```
C2_pts = random(gm,N);
sample_data = [C1_pts;C2_pts]';
sample_min = [min(sample_data(1,:)), min(sample_data(2,:))];
sample_max = [max(sample_data(1,:)), max(sample_data(2,:))];
```

MAP decision rule

```
t_map = [];
class0_map = [];
class1_map = [];
% Apply MAP decision rule on sample data
for i = 1:size(sample_data,2)
   Px0 = det(2*pi*C1)^(-1/2)*exp(-1/2*(sample_data(:,i)-
m)'*inv(C1)*(sample_data(:,i)-m));
   Px1 = pdf(gm,sample_data(:,i)');
   if Px0 > Px1
       class0_map = [class0_map sample_data(:,i)];
       t_map(i) = 0;
   else
       class1_map = [class1_map sample_data(:,i)];
       t_map(i) = 1;
   end
end

% Decision boundary
% This is done by creating a coordinate grid, calculating the
 probability
% for each point to belong to both classes. The point belongs to the
 class
% that returns the highest probability.
X_db =
 [linspace(sample_min(1),sample_max(1),N);linspace(sample_min(2),sample_max(2),N)]
class0_db = [];
class1_db = [];
for i = 1:N
    for j = 1:N
        x = [X_db(1,i);X_db(2,j)];
        Px0 = det(2*pi*C1)^(-1/2)*exp(-1/2*(x-m)'*inv(C1)*(x-m));
        Px1 = pdf(gm,x');
        if Px0 > Px1
            class0_db = [class0_db x];
        else
            class1_db = [class1_db x];
        end
    end
end
% Incorrect classification
error_MAP = 0;
for i = 1:size(t_map,2)
    if (t_map(i) == 1 && i <= 200) || (t_map(i) == 0 && i > 200)
        error_MAP = error_MAP + 1;
    end
end
```

```
error_MAP_factor = error_MAP/size(t_map,2);

figure(1);
hold on;
scatter(class0_db(1,:), class0_db(2,:));
scatter(class1_db(1,:), class1_db(2,:));
scatter(class0_map(1,:),class0_map(2,:));
scatter(class1_map(1,:),class1_map(2,:));
hold off;
legend('Class 0 area','Class 1 area','Class 0 points','Class 1
 points');
title('MAP rule');
str = ['error factor = ' num2str(error_MAP_factor)];
xlabel(str);
```

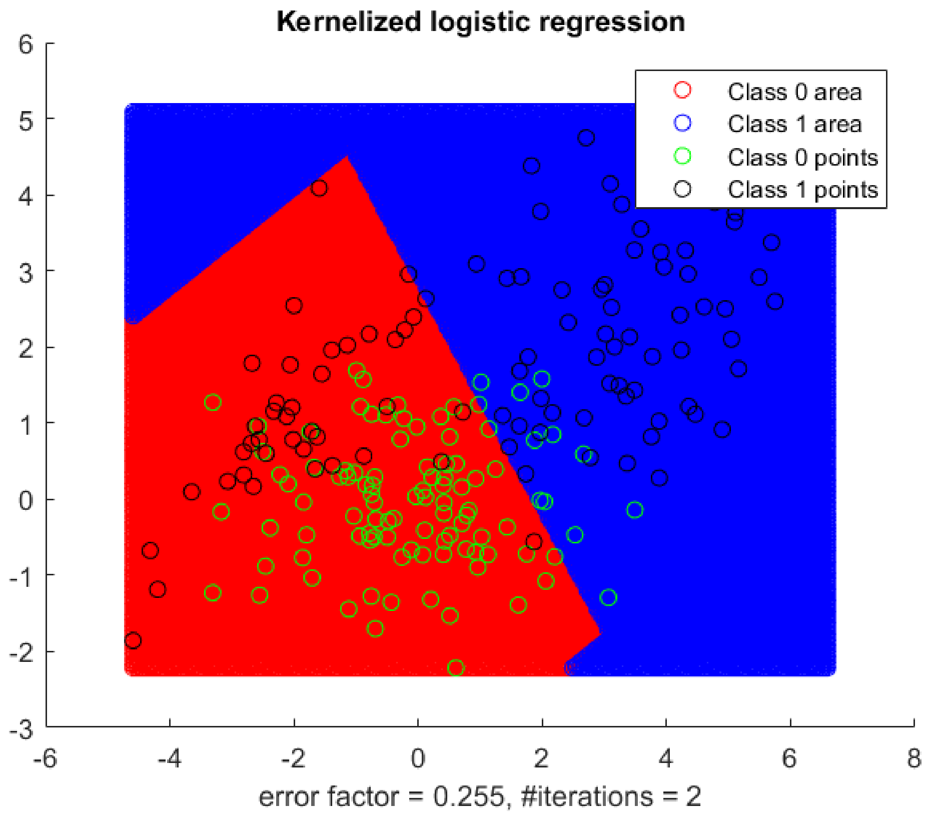


Kernelized logistic regression

```
X = [C1_pts(1:N/2,:)' C2_pts(1:N/2,:)'];

l = 0.1;
lambda = 10;

K = zeros(N,N);
a = zeros(size(K,1),1);
t = [zeros(N/2,1); ones(N/2,1)];
```

```
for i = 1:N
    for j = 1:N
        norm = sqrt((X(1,i)-X(1,j))^2 + (X(2,i)-X(2,j))^2);
        K(i,j) = exp(-(norm)^2/(2*l^2));
    end
end

% Newton iteration
convergence = 0;
newton_iter = 0;
while convergence == 0
    z = a'*K;
    y = 1./(1+exp(-z))';
    R = diag(y.*(1-y));
    H = K*R*K+lambda*K;
    gradEmap = K*(y-t+lambda*a);
    a_new = a - H\gradEmap;
    if sum(abs(a_new-a),1) < 0.01
        convergence = 1;
    end
    if newton_iter > 50
        convergence = 1;
        disp('Newton iteration did not converge in Kernelized logistic
 regression');
    end
    a = a_new;
    newton_iter = newton_iter + 1;
end

% Decission boundary
K_db = zeros(N,1);
class0_db_k = [];
class1_db_k = [];
for i = 1:N
    for j = 1:N
        x = [X_db(1,i);X_db(2,j)];
        for u = 1:size(X_db,2)
            norm = sqrt((x(1)-X_db(1,u))^2 + (x(2)-X_db(2,u))^2);
            K_db(u) = exp(-(norm)^2/(2*l^2));
        end
        z_opt = a'*K_db;
        if z_opt < 0
            class0_db_k = [class0_db_k x];
        else
            class1_db_k = [class1_db_k x];
        end
    end
end

% Incorrect classification
error_k = 0;
K_db = zeros(N,1);
for i = 1:size(X,2)
    x = [X(1,i);X(2,i)];
```

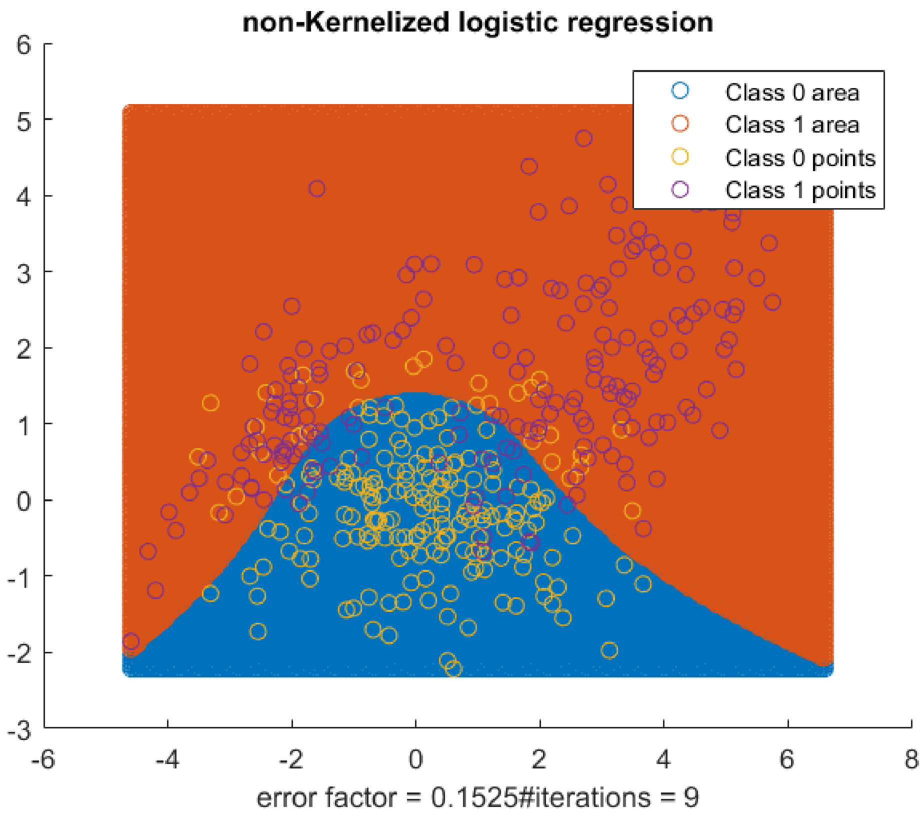
```
        for u = 1:size(X,2)
            norm = sqrt((x(1)-X_db(1,u))^2 + (x(2)-X_db(2,u))^2);
            K_db(u) = exp(-(norm)^2/(2*l^2));
        end
        z_opt = a'*K_db;

        if (z_opt < 0 && i > N/2) || (z_opt > 0 && i < N/2)
            error_k = error_k + 1;
        end
    end
    error_k_factor = error_k/size(X,2);

    figure(2);
    hold on;
    scatter(class0_db_k(1,:),class0_db_k(2,:),'r');
    scatter(class1_db_k(1,:),class1_db_k(2,:),'b');
    scatter(C1_pts(1:N/2,1),C1_pts(1:N/2,2),'g');
    scatter(C2_pts(1:N/2,1),C2_pts(1:N/2,2),'k');
    hold off;
    legend('Class 0 area','Class 1 area','Class 0 points','Class 1
     points');
    title('Kernelized logistic regression');
    str = ['error factor = ' num2str(error_k_factor) ', #iterations = '
     num2str(newton_iter)];
    xlabel(str);
```



non-Kernelized logistic regression

```
x1 = [C1_pts(:,1); C2_pts(:,1)];
x2 = [C1_pts(:,2); C2_pts(:,2)];

phi = [ones(1,2*N);x1'; x2'; x1'.^2; x2'.^2; x1'.*x2';
 x1'.^3;x1'.^2.*x2';x1'.*x2'.^2;x2'.^3];
w = zeros(size(phi,1),1);
t = [zeros(N,1); ones(N,1)];

% Newton update
convergence = 0;
newton_iter = 0;
while convergence == 0
    z = w'*phi;
    y = 1./(1+exp(-z))';
    R = diag(y.*(1-y));
    w_new = w - (phi*R*phi')\phi*(y-t);
    if sum(abs(w_new-w),1) < 0.01
        convergence = 1;
    end
    if newton_iter > 50
        convergence = 1;
        disp('Newton iteration did not converge in non-Kernelized
 logistic regression');
    end
    w = w_new;
    newton_iter = newton_iter + 1;
end

% Decission boundary
class0_db = [];
class1_db = [];
for i = 1:N
    for j = 1:N
        phi_db =
 [1;X_db(1,i);X_db(2,j);X_db(1,i)^2;X_db(2,j)^2;X_db(1,i)*X_db(2,j);X_db(1,i)^3;X_
        if w'*phi_db < 0
            class0_db = [class0_db;X_db(1,i) X_db(2,j)];
        else
            class1_db = [class1_db;X_db(1,i) X_db(2,j)];
        end
    end
end

% Incorrect classification
error_nk = 0;
for i = 1:2*N
    phi_ic = [1; x1(i); x2(i); x1(i)^2; x2(i)^2; x1(i)*x2(i); x1(i)^3;
 x1(i)^2*x2(i);x1(i)*x2(i)^2;x2(i)^3];
    if (w'*phi_ic > 0 && i < N) || (w'*phi_ic < 0 && i > N)
        error_nk = error_nk + 1;
    end
```

```
end
error_nk_factor = error_nk/(2*N);


figure(3);
hold on;
scatter(class0_db(:,1),class0_db(:,2));
scatter(class1_db(:,1),class1_db(:,2));
scatter(C1_pts(:,1),C1_pts(:,2));
scatter(C2_pts(:,1),C2_pts(:,2));
hold off;
legend('Class 0 area','Class 1 area','Class 0 points','Class 1
 points');
legend('Class 0 area','Class 1 area','Class 0 points','Class 1
 points');
title('non-Kernelized logistic regression');
str = ['error factor = ' num2str(error_nk_factor), '#iterations = '
 num2str(newton_iter) ];
xlabel(str);
```



Comments

There is some overfitting with the different methods. When we check overfitting with the MAP-rule, we get a error factor about 15%. This tells me that the two distributions are very close together, making a perfect logistic regression algorithm very difficult. After tuning the regularization parameters, I get the error factor down to 15,8% with the non-kernelized logistic regression. This is a good result, since it is almost as low as the MAP rule. Using the kernelized logistic regression resulted in some more overfitting with an error factor about 25,5% after trimming the regularization factor. It is expected that the Kernelized logistic regression would deliver poorer result compared to non-kernelized, but I think the error factor could have been reduced if the distributions were chosen with greater difference in mean value.

Published with MATLAB® R2017a