

QIN, Yujie mail@qinyujie.net

Part 1: Using Neural Network with Gradient Decent, Line Search and Conjugate Gradient methods to classify.

1.Function Codes (Gradient Decent, Line Search and Conjugate Gradient)

```
%calculate Gradient Decent with single layer network once time
function [w] = GradientDescent(x, t, w, eta)

H = x * x';
b = - x * t';
g = H * w + b;

%update synaptic weights
w = w - eta .* g;

end
```

```
%calculate Line Search once time
function [w] = LineSearch(x, t, w)

H = x * x';
b = - x * t';
g = H * w + b;
alpha = -(g' * g) ./ (g' * H * g);

%update synaptic weights
w = w + alpha .* g;

end
```

```
%calculate Conjugate Gradient once time
function [w, d] = ConjugateGradient(x, t, w, d, iter)

H = x * x';
b = - x * t';
g = H * w + b;

if (iter == 1)
    w = w;
    d = -g;
else
    alpha = -(d' * g) ./ (d' * H * d);
    w = w + alpha .* d;
    g1 = H * w + b;
    beta = -(g1' * g1) ./ (g' * g);
    d = g1 + beta .* d;
end

end
```

2.Main function and Grafiks

```
%function Pro5_1
%Problem 5, Question 1.
function Pro5_1

clear all;
close all;
```

```
% training dataset
target = [-0.1, 0.5, 0.5];
input = [-1, 0.3, 2];
% extend data matrix
input = [input; 1, 1, 1];

% initialize parameters
w = rand(2, 1);
iteration = 0;
criteria = 0.00001;
%init for GD
eta = LearningRateOfGradientDecent(input, target, w); %get best learning rate
w_gd = w;
run_gd = true;
err0_gd = 1;
err1_gd = 2;
err2_gd = 3;
visx_gd = [];
visy_gd = [];
visx2_gd = [];
visy2_gd = [];
%init for LS
w_ls = w;
run_ls = true;
err0_ls = 1;
err1_ls = 2;
err2_ls = 3;
visx_ls = [];
visy_ls = [];
visx2_ls = [];
visy2_ls = [];
%init for CG
w_cg = w;
d_cg = [];
run_cg = true;
err0_cg = 1;
err1_cg = 2;
err2_cg = 3;
visx_cg = [];
visy_cg = [];
visx2_cg = [];
visy2_cg = [];

% learning loop
while (run_gd || run_ls || run_cg)
    iteration = iteration + 1;

    %calculate new weights with Gradient Descent
    if (run_gd)
        %record visualization
        visx2_gd = [visx2_gd w_gd(1)];
        visy2_gd = [visy2_gd w_gd(2)];
        %calculate the output
        output = w_gd' * input;
        %calculate the weights
        w_gd = GradientDescent(input, target, w_gd, eta);
        %calculate error
        err2_gd = err1_gd;
        err1_gd = err0_gd;
        err0_gd = sum((output - target).^2);
    end
end
```

```

        %criteria
        if (abs(err0_gd - err1_gd) < criteria && abs(err1_gd - err2_gd) <
criteria)
            run_gd = false;
        end
        %record visualization
        visx_gd = [visx_gd iteration];
        visy_gd = [visy_gd err0_gd];
    end

    %calculate new weights with Line Search
    if (run_ls)
        %record visualization
        visx2_ls = [visx2_ls w_ls(1)];
        visy2_ls = [visy2_ls w_ls(2)];
        %calculate the output
        output = w_ls' * input;
        w_ls = LineSearch(input, target, w_ls);
        %calculate error
        err2_ls = err1_ls;
        err1_ls = err0_ls;
        err0_ls = sum((output - target).^2);
        %criteria
        if (abs(err0_ls - err1_ls) < criteria && abs(err1_ls - err2_ls) <
criteria)
            run_ls = false;
        end
        %record visualization
        visx_ls = [visx_ls iteration];
        visy_ls = [visy_ls err0_ls];
    end

    %calculate new weights with Conjugate Gradient
    if (run_cg)
        %record visualization
        visx2_cg = [visx2_cg w_cg(1)];
        visy2_cg = [visy2_cg w_cg(2)];
        %calculate the output
        output = w_cg' * input;
        %calculate the weights
        [w_cg, d_cg] = ConjugateGradient(input, target, w_cg, d_cg, iteration);
        %calculate error
        err2_cg = err1_cg;
        err1_cg = err0_cg;
        err0_cg = sum((output - target).^2);
        %criteria
        if (abs(err0_cg - err1_cg) < criteria && abs(err1_cg - err2_cg) <
criteria) || (run_gd == false)
            run_cg = false;
        end
        %record visualization
        visx_cg = [visx_cg iteration];
        visy_cg = [visy_cg err0_cg];
    end
end

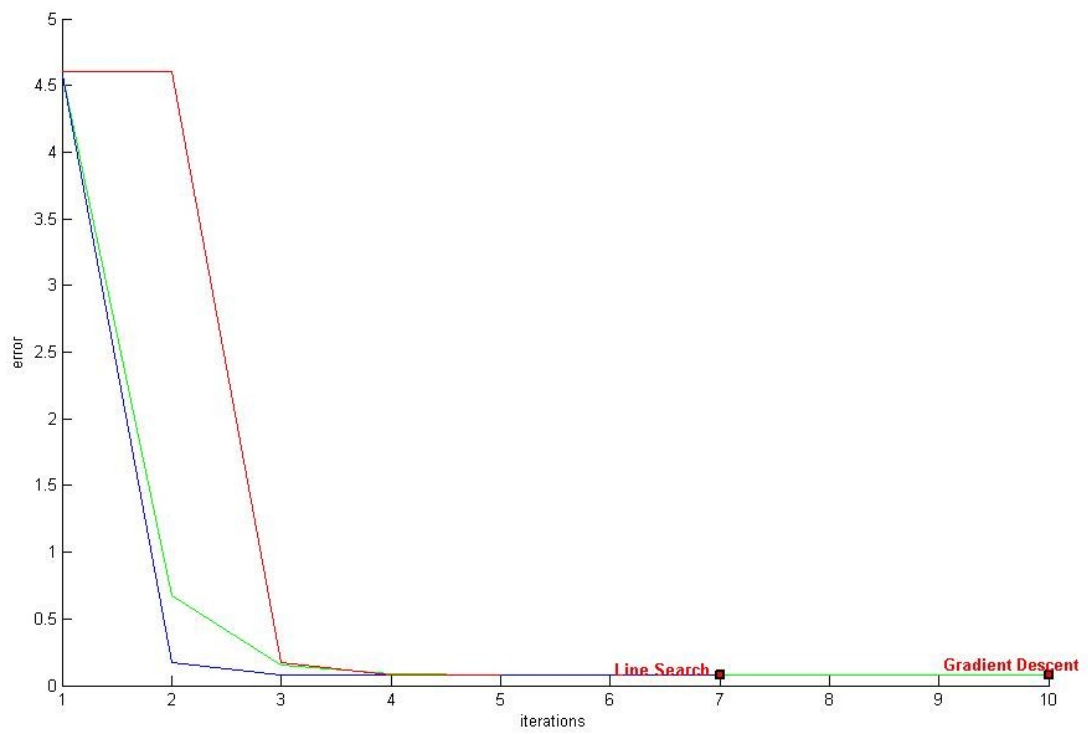
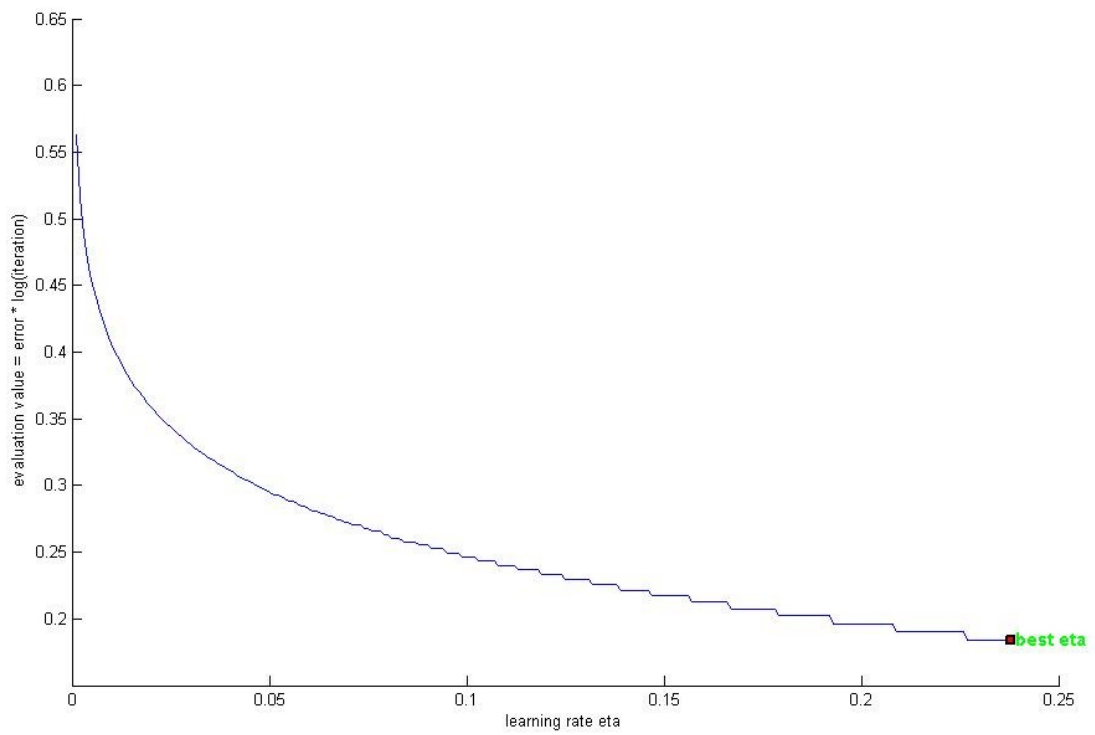
%visualization
figure('Name', 'Performance of 3 Methods: Error');
xlabel('iterations');
ylabel('error');

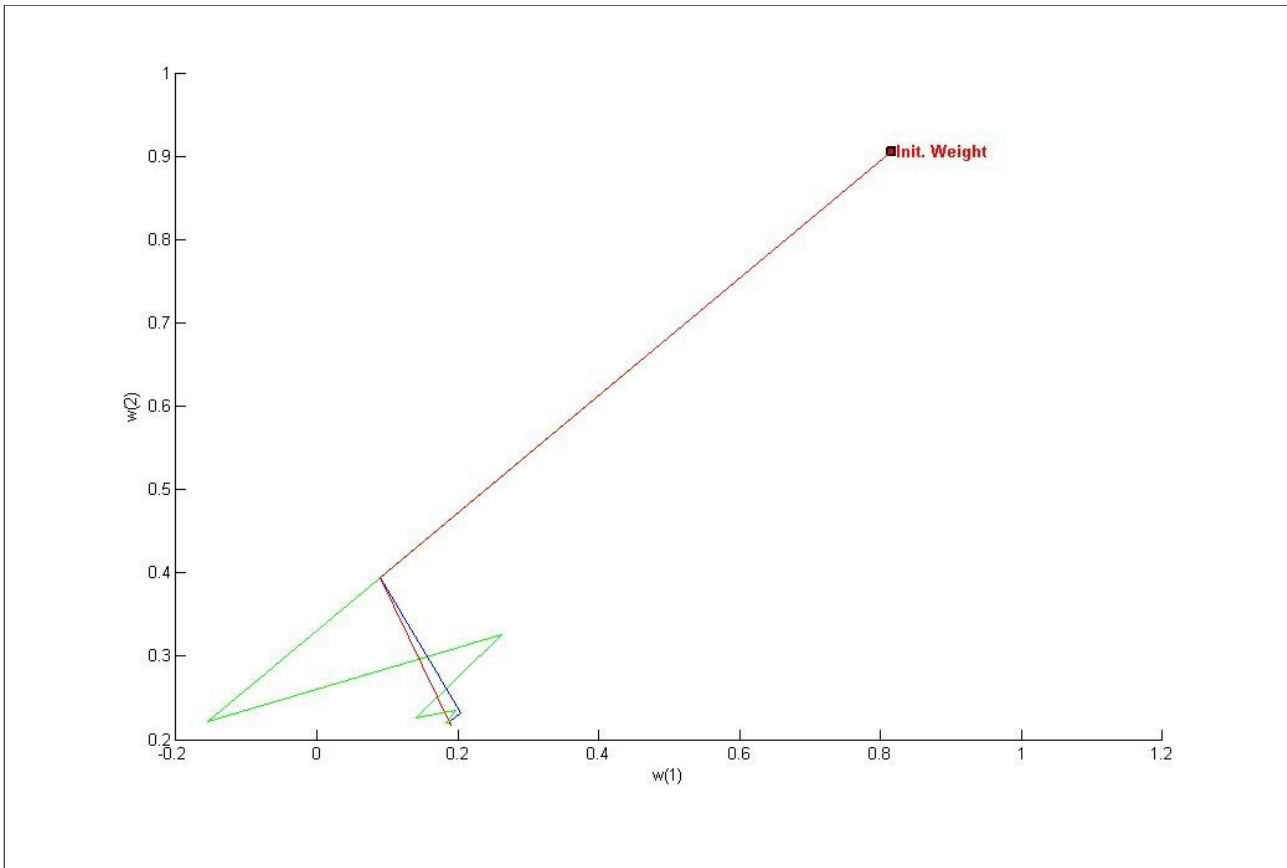
```

```
hold on;
line(visx_gd, visy_gd, 'Color', 'g');
plot(visx_gd(end), visy_gd(end), '--rs', 'LineWidth', 2, 'MarkerEdgeColor', 'k',
'MarkerFaceColor', 'r', 'MarkerSize', 5)
text(visx_gd(end)-1, visy_gd(end)*2, '\bf\fontsize{10}\color[rgb]{1 0 0}
Gradient Descent');
hold off;
hold on;
line(visx_ls, visy_ls, 'Color', 'b');
plot(visx_ls(end), visy_ls(end), '--rs', 'LineWidth', 2, 'MarkerEdgeColor', 'k',
'MarkerFaceColor', 'r', 'MarkerSize', 5)
text(visx_ls(end)-1, visy_ls(end)*1.5, '\bf\fontsize{10}\color[rgb]{1 0 0} Line
Search');
hold off;
hold on;
line(visx_cg, visy_cg, 'Color', 'r');
plot(visx_cg(end), visy_cg(end), '--rs', 'LineWidth', 2, 'MarkerEdgeColor', 'k',
'MarkerFaceColor', 'r', 'MarkerSize', 5)
text(visx_cg(end)-1, visy_cg(end)*1, '\bf\fontsize{10}\color[rgb]{1 0 0}
Conjugate Gradient');
hold off;

figure('Name', 'Performance of 3 Methods: Weights');
xlabel('w(1)');
ylabel('w(2)');
%axis([-0.25 1.25 -0.25 1.25]);
hold on;
plot(w(1), w(2), '--rs', 'LineWidth', 2, 'MarkerEdgeColor', 'k',
'MarkerFaceColor', 'r', 'MarkerSize', 5)
text(w(1), w(2), '\bf\fontsize{10}\color[rgb]{1 0 0} Init. Weight');
plot(w_cg(1), w_cg(2), '--rs', 'LineWidth', 2, 'MarkerEdgeColor', 'k',
'MarkerFaceColor', 'r', 'MarkerSize', 5)
text(w_cg(1), w_cg(2), '\bf\fontsize{10}\color[rgb]{1 0 0} Target Weight');
hold off;
hold on;
line(visx2_gd, visy2_gd, 'Color', 'g');
hold off;
hold on;
line(visx2_ls, visy2_ls, 'Color', 'b');
hold off;
hold on;
line(visx2_cg, visy2_cg, 'Color', 'r');
hold off;

e
```





Part 2: KNN

1. Function Codes

```

%function [X, T] = GenerateData4N(N)
%Generate 4 * N training patterns for XOR problem(4 centers, 2 classes)
function [X, T] = GenerateData4N(N)

sigma = sqrt(0.1);

X = [sigma*randn(N, 2) + repmat([0, 1], N, 1); ...
     sigma*randn(N, 2) + repmat([1, 0], N, 1);...
     sigma*randn(N, 2) + repmat([0, 0], N, 1);...
     sigma*randn(N, 2) + repmat([1, 1], N, 1)];

T = [ones(2*N, 1); -ones(2*N, 1)];

end

%function [dist] = CountDistance2D(point, pointSet)
% point set - point set [(x1, y1); (x2, y2);(x3, y3);...]

function [dist] = CountDistance2D(pointSet1, pointSet2)

N1 = size(pointSet1, 1);
N2 = size(pointSet2, 1);

XX = sum(pointSet1.*pointSet1, 2);
YY = sum(pointSet2.*pointSet2, 2);
dist = repmat(XX, 1, N2) + repmat(YY', N1, 1) - 2*pointSet1*pointSet2';

```

```

end

function Pro5_2

close all;
clear all;

%generate training data
[Input, Target] = GenerateData4N(20);

%creat the plane for classification
YMIN = min(Input(:,2));
YMAX = max(Input(:,2));
XMIN = min(Input(:,1));
XMAX = max(Input(:,1));
[XX, YY] = meshgrid(YMIN:0.1:YMAX, XMIN:0.1:XMAX);
Test = [reshape(XX, size(XX, 1) * size(XX, 2), 1), reshape(YY, size(YY, 1) *
size(YY, 2), 1)];

k = [1, 7, 11, 19];
%Loop for different k
for i=1:length(k)
    %KNN
    Classes = knn(k(i), Input, Target, Test);
    Classes = reshape(Classes, size(XX, 1), size(XX, 2));
    %Visualization
    figurename = ['KNN: k = ' num2str(k(i))];
    figure('Name', figurename);
    %refine axes
    axis([XMIN, XMAX, YMIN, YMAX]);
    %Contour
    contour(XX, YY, Classes);
    % plot training data set
    IP = find(Target == 1);
    IN = find(Target == -1);
    hold on;
    plot(Input(IP, 1), Input(IP, 2), 'r+', Input(IN, 1), Input(IN, 2), 'bo');
    hold off;
end

end

```

2.Main function and Grafiks

```

function Pro5_2

close all;
clear all;

%generate training data
[Input, Target] = GenerateData4N(20);

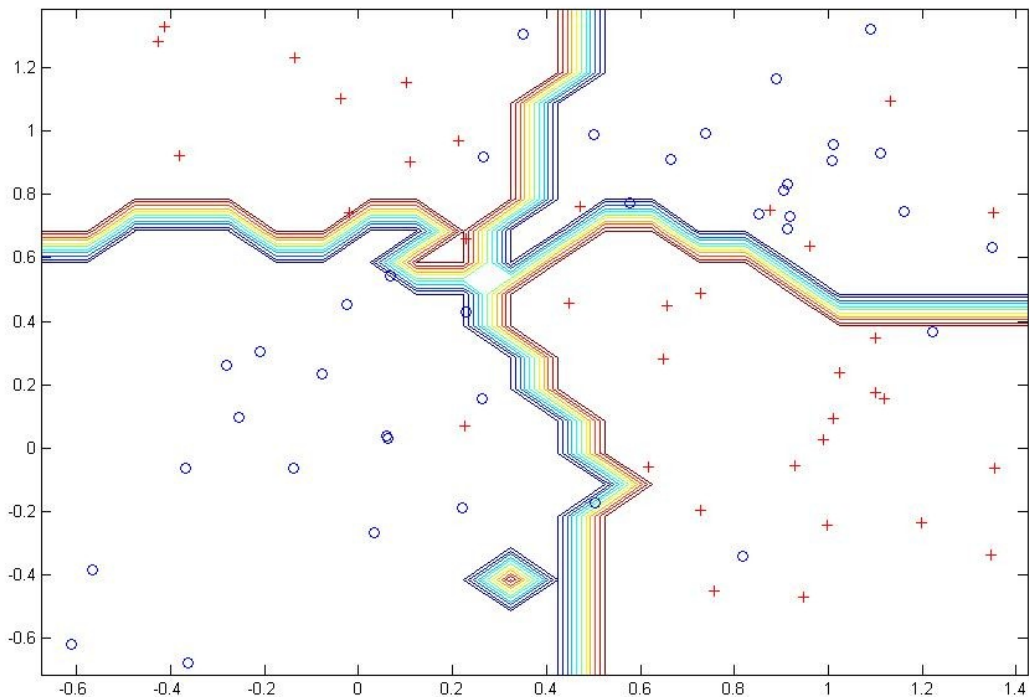
%creat the plane for classification
YMIN = min(Input(:,2));
YMAX = max(Input(:,2));
XMIN = min(Input(:,1));
XMAX = max(Input(:,1));
[XX, YY] = meshgrid(YMIN:0.1:YMAX, XMIN:0.1:XMAX);
Test = [reshape(XX, size(XX, 1) * size(XX, 2), 1), reshape(YY, size(YY, 1) *
size(YY, 2), 1)];

```

```

k = [1, 7, 11, 19];
%Loop for different k
for i=1:length(k)
    %KNN
    Classes = knn(k(i), Input, Target, Test);
    Classes = reshape(Classes, size(XX, 1), size(XX, 2));
    %Visualization
    figurename = ['KNN: k = ' num2str(k(i))];
    figure('Name', figurename);
    %refine axes
    axis([XMIN, XMAX, YMIN, YMAX]);
    %Contour
    contour(XX, YY, Classes);
    % plot training data set
    IP = find(Target == 1);
    IN = find(Target == -1);
    hold on;
    plot(Input(IP, 1), Input(IP, 2), 'r+', Input(IN, 1), Input(IN, 2), 'bo');
    hold off;
end
end

```



Part 3: Parzen Window

1. Function Codes

```

function [classes] = ParzenWindows(sigma, X, T, new)

dist = CountDistance2D(X, new);

```

```

% Ne presents to which group the first k neighbors belong to.
cof = exp(-dist./(2.*sigma.*sigma));
Ne = T' * cof;
% As there are only two groups, the problem seems to be easy. When the sum
% of the first k neighbors' group numbers is greater than 0, that means
% more of them belong to group 1. As a result, we assign the test data also
% group 1. Otherwise -1.
classes = sign(Ne);

end

```

2.Main function and Grafiks

```

function Pro5_3

close all;
clear all;

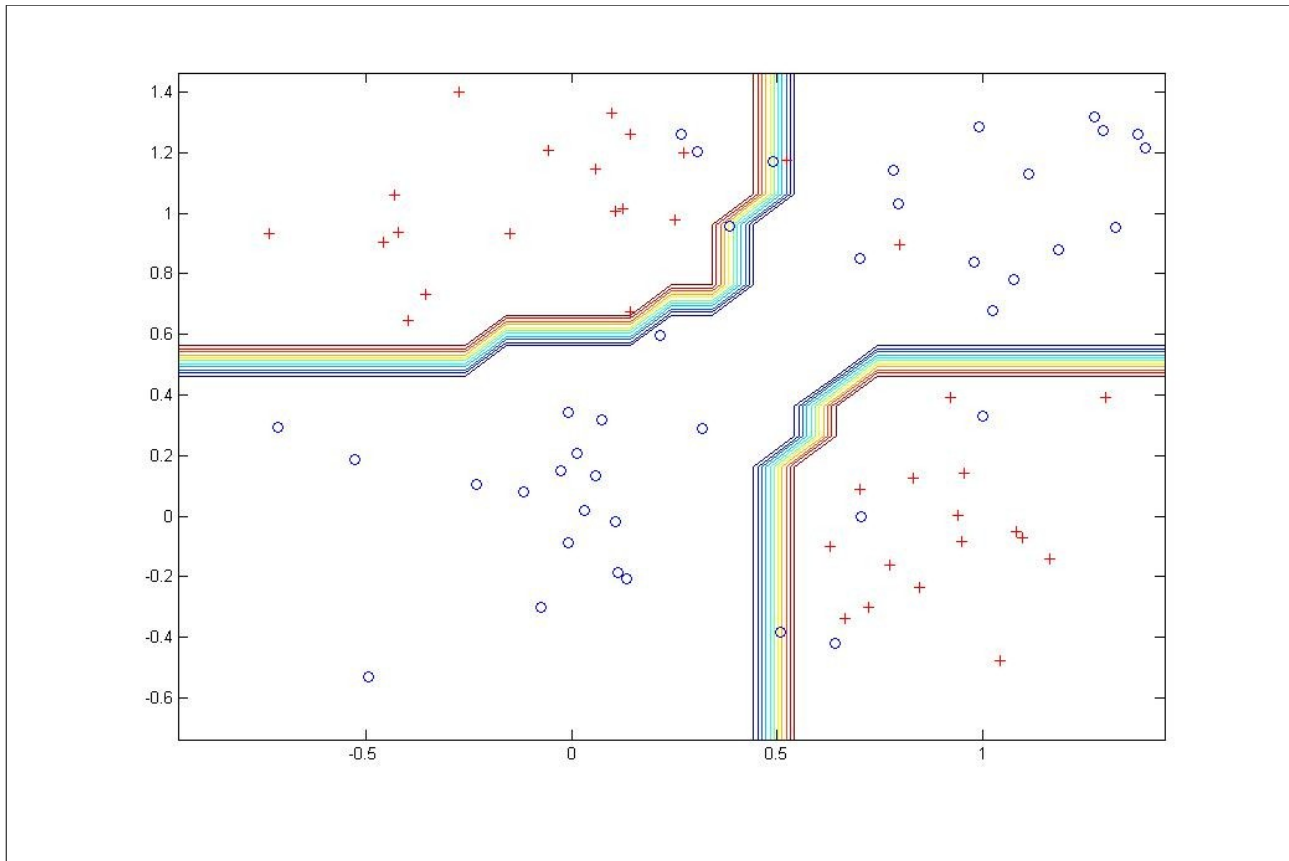
[Input, Target] = GenerateData4N(20);

%creat the plane for classification
YMIN = min(Input(:,2));
YMAX = max(Input(:,2));
XMIN = min(Input(:,1));
XMAX = max(Input(:,1));
[XX, YY] = meshgrid(YMIN:0.1:YMAX, XMIN:0.1:XMAX);
Test = [reshape(XX, size(XX, 1) * size(XX, 2), 1), reshape(YY, size(YY, 1) *
size(YY, 2), 1)];

sigma = sqrt([0.01, 0.05, 0.1, 0.5, 1, 5]);
%Loop for different sigma
for i=1:length(sigma)
    %KNN
    Classes = ParzenWindows(sigma(i), Input, Target, Test);
    Classes = reshape(Classes, size(XX, 1), size(XX, 2));
    %Visualization
    figurename = ['Parzen Windows: sigma^2 = ' num2str(sigma(i)^2)];
    figure('Name', figurename);
    %refine axes
    axis([XMIN, XMAX, YMIN, YMAX]);
    %Contour
    contour(XX, YY, Classes);
    % plot training data set
    IP = find(Target == 1);
    IN = find(Target == -1);
    hold on;
    plot(Input(IP, 1), Input(IP, 2), 'r+', Input(IN, 1), Input(IN, 2), 'bo');
    hold off;
end

end

```



Part 4: RBF

1. Function Codes

```
function [classes] = RBF(k, sigma, X, T, new)

%find centers
[A, cX] = kmeans(X, k);
cT = zeros(k, 1);
for i = 1:k
    IP = find(A == i);
    cT(i) = sign(sum(T(IP)));
end

%Design matrix
dist = CountDistance2D(X, cX);
phi = exp(-dist./(2.*sigma.*sigma));
%calculate synaptic weights
w = pinv(phi) * T;

%classify
dist = CountDistance2D(cX, new);
phi = exp(-dist./(2.*sigma.*sigma));
Y = phi' * w;
classes = sign(Y);

end
```

2. Main function and Grafiks

```
function Pro5_4

close all;
clear all;

%generate training data
[Input, Target] = GenerateData4N(20);

%creat the plane for classification
YMIN = min(Input(:,2));
YMAX = max(Input(:,2));
XMIN = min(Input(:,1));
XMAX = max(Input(:,1));
[XX, YY] = meshgrid(YMIN:0.1:YMAX, XMIN:0.1:XMAX);
Test = [reshape(XX, size(XX, 1) * size(XX, 2), 1), reshape(YY, size(YY, 1) *
size(YY, 2), 1)];

k = [4, 7, 15];
sigma = sqrt([0.01, 0.05, 0.1, 0.5]);
%Loop for different k
for i=1:length(k)
    %loop for different sigma
    for t=1:length(sigma)
        %RBF
        Classes = RBF(k(i), sigma(t), Input, Target, Test);
        Classes = reshape(Classes, size(XX, 1), size(XX, 2));
        %Visualization
        figurename = ['RBF: k = ' num2str(k(i)) ' sigma^2 = '
num2str(sigma(t)^2)];
        figure('Name', figurename);
        %refine axes
        axis([XMIN, XMAX, YMIN, YMAX]);
        %Contour
        contour(XX, YY, Classes);
        % plot training data set
        IP = find(Target == 1);
        IN = find(Target == -1);
        hold on;
        plot(Input(IP, 1), Input(IP, 2), 'r+', Input(IN, 1), Input(IN, 2),
'bo');
        hold off;
    end
end
end
```

