Dawid Jantosz

Signal processing and identification in control of mechatronic devices

Topic: Classification and regression

Introduction:

Classification is a predictive model that approximates a mapping function from input variables to identify discrete output variables, which can be labels or categories. The mapping function of classification algorithms is responsible for predicting the label or category of the given input variables. A classification algorithm can have both discrete and real-valued variables, but it requires that the examples be classified into one of two or more classes.

Ex.1 Classification dataset generation

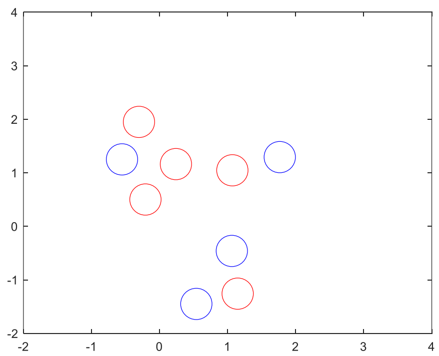
Classification of data is performed by the use of clastering. Process that requires assigning each sample with a cluster, using an unsupervised algorithm.

Code used for generating the clasters of data:

```
close all
rng('shuffle'); % To get different results each time
Clusters.ClustersA = 5; % How many clusters of data exist in class 1?
Clusters.ClustersB = 4; % How many clusters of class 2 exist?
% Definition of clusters centers
%Clusters.ACoordinates = randn(2,Clusters.ClustersA);
%Clusters.BCoordinates = randn(2,Clusters.ClustersB);
for k = 1:Clusters.ClustersA
 plot(Clusters.ACoordinates(1,k),Clusters.ACoordinates(2,k),'or','MarkerSize',25); hold on
end
for k = 1:Clusters.ClustersB
 plot(Clusters.BCoordinates(1,k),Clusters.BCoordinates(2,k),'ob','MarkerSize',25); hold on
end
xlim([-2 4]);
ylim([-2,4]);
```

Clusters that we will be using in the next exercises have to not overlap, should not be linearly separable.

Example of valid result:



Ex. 2 Generating and dividing dataset

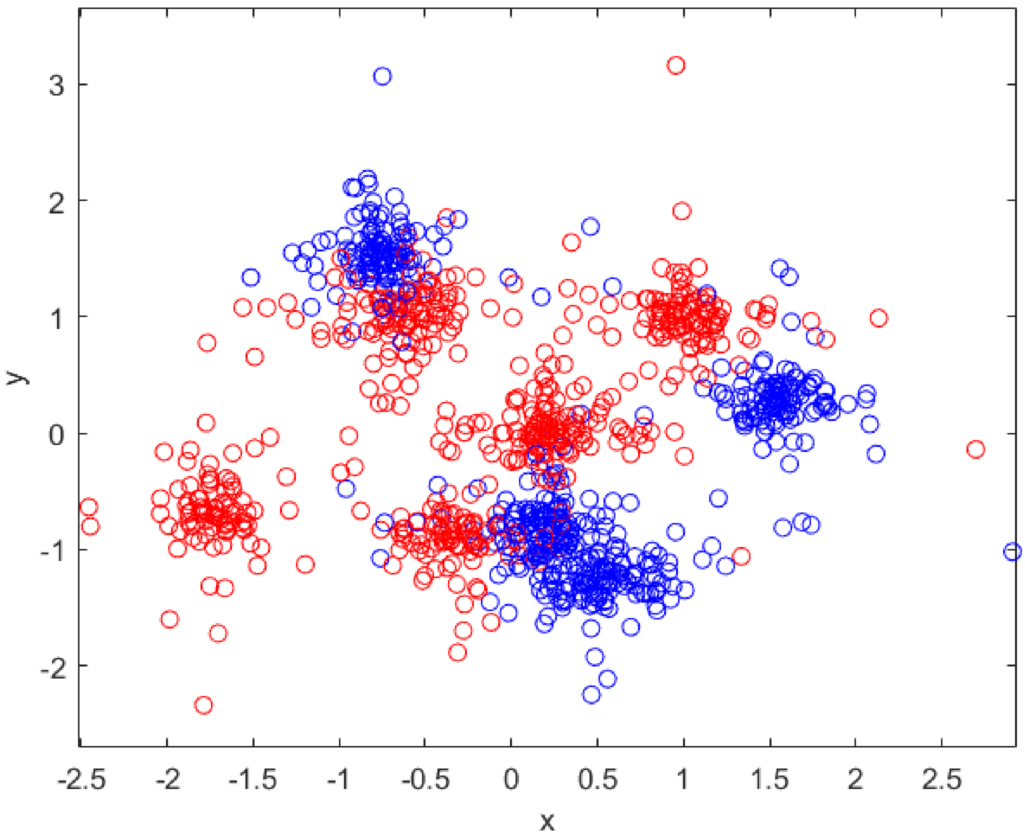
In machine learning to obtain proper results that secure us from overfitting (classifier or rergressor obtain high accuracy on training dataset, but much lower on a testing dataset), we devide set to subsets corresponding to testing, validation and training.

Train dataset: The actual dataset that we use to train the model.

Validation dataset: In this dataset we configure the classifier to obtain mataparameter values, which will improve our results.

Testing dataset: In this dataset we test the classifier estimating its future accuracy. It is only used once a model is completely trained(using the train and validation sets)

```
clear all
close all
clc
load Clusters
Samples = 1000; % How many data samples there are?
DataDivision = 0.5; % How many data samples fall into which class?
v = 2; % v parameter of T Student's distribution
% Definition of data
for k = 1:Samples
 if(rand()>DataDivision)
 DATA(1,k) = 1;
 Ind = randi(Clusters.ClustersA);
 DATA(2,k) = Clusters.ACoordinates(1,Ind)+random('T',v)*0.15;
 DATA(3,k) = Clusters.ACoordinates(2,Ind)+random('T',v)*0.15;
 else
 DATA(1,k) = 0;
 Ind = randi(Clusters.ClustersB);
 DATA(2,k) = Clusters.BCoordinates(1,Ind)+random('T',v)*0.15;
 DATA(3,k) = Clusters.BCoordinates(2,Ind)+random('T',v)*0.15;
 end
end
for k = 1:Samples
 if(DATA(1,k) == 1)
 plot(DATA(2,k),DATA(3,k),'or'); hold on
 else
 plot(DATA(2,k),DATA(3,k),'ob'); hold on
 end
end
xlabel('x');
ylabel('y');
ylim([-3 4])
```



The code above fills our clusters with data.

Worth mentioning is v value that determines how heavy are tails of the data distribution.

```
Indices = randperm(length(DATA1));
DATA_permutated = DATA1(:,Indices)
TR_number = ceil(length(DATA1)*0.5);
VA_number = ceil(length(DATA1)*0.25);
TE_number = ceil(length(DATA1)*0.25);
TR_DATA = DATA_permutated(:,1:TR_number);
VA_DATA = DATA_permutated(:,TR_number+1:TR_number+VA_number);
TE_DATA = DATA_permutated(:,TR_number+VA_number+1:end);
save TR_DATA TR_DATA
save VA_DATA VA_DATA
save TE_DATA TE_DATA
```

The code above creates sets for training, validation and testing. In our case it would equal 500, 250 and 250 samples respectively.

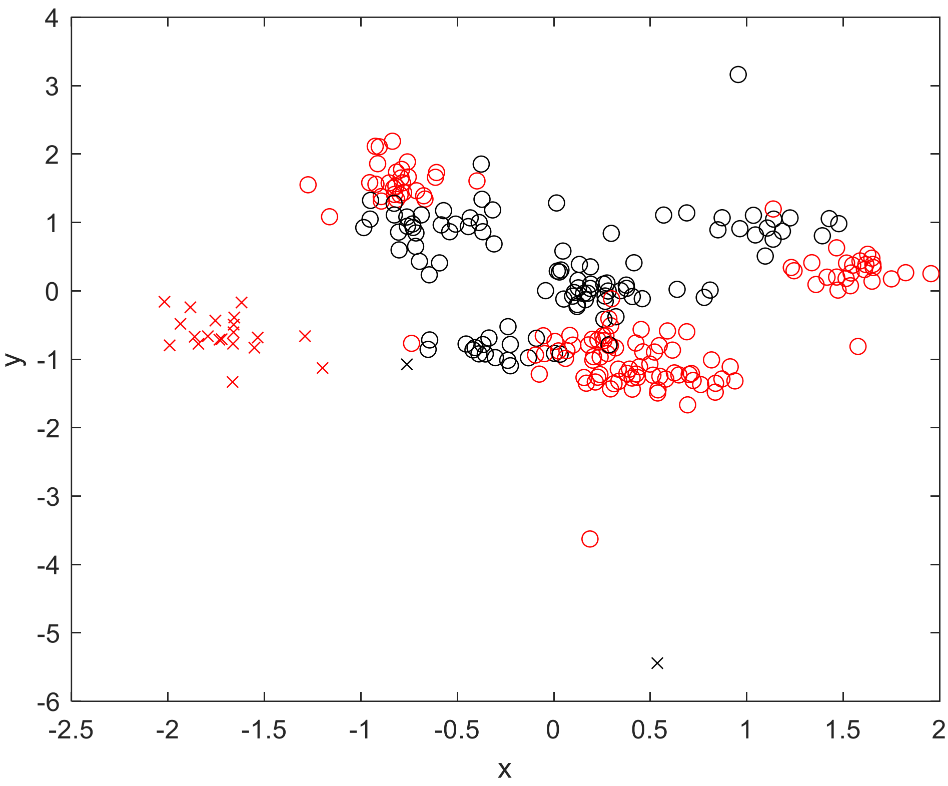
Ex.3-6

A classifier in machine learning is an algorithm that automatically orders or categorizes data into one or more of a set of “classes.” In this exercise we will desing simple linear classifier, even though our data is not linearly separable.

```
function [ClassLabel] = InitialClassifier(x,y,Parameters)
 if(Parameters.W1*x + Parameters.W2*y + Parameters.B > 0)
 ClassLabel = 1;
 else
 ClassLabel = 0;
end
end
```

Classification of our data:

```
load VA_DATA
Parameters.W1 = 1;
Parameters.W2 = 0.3;
Parameters.B = 1;
ErrorsA = 0;
ErrorsB = 0;
for k = 1:length(VA_DATA)
 if(InitialClassifier(VA_DATA(2,k),VA_DATA(3,k),Parameters) == 1)
 % Data point classified as A
 if(VA_DATA(1,k) == 1)
 % Data point classified correctly!
 plot(VA_DATA(2,k),VA_DATA(3,k),'ok'); hold on
 else
 plot(VA_DATA(2,k),VA_DATA(3,k),'or') ; hold on
 ErrorsA = ErrorsA + 1;
 end
 else
 % Data point classified as B
 if(VA_DATA(1,k) == 0)
 % Data point classified correctly!
 plot(VA_DATA(2,k),VA_DATA(3,k),'xk'); hold on
 else
 plot(VA_DATA(2,k),VA_DATA(3,k),'xr') ; hold on
 ErrorsB = ErrorsB + 1;
 end
 end
end
xlabel('x');
ylabel('y');
ErrorsA
ErrorsB
ErrorsA+ErrorsB
```



Results:

ErrorsA = 126

ErrorsB = 20

ans = 146

Obtained results are far form optimal. We have to optimize our algorithm, Using scripts developed in class 1 and 2 to optimize parameters of my linear classifier. I will use a grid search algorithm.

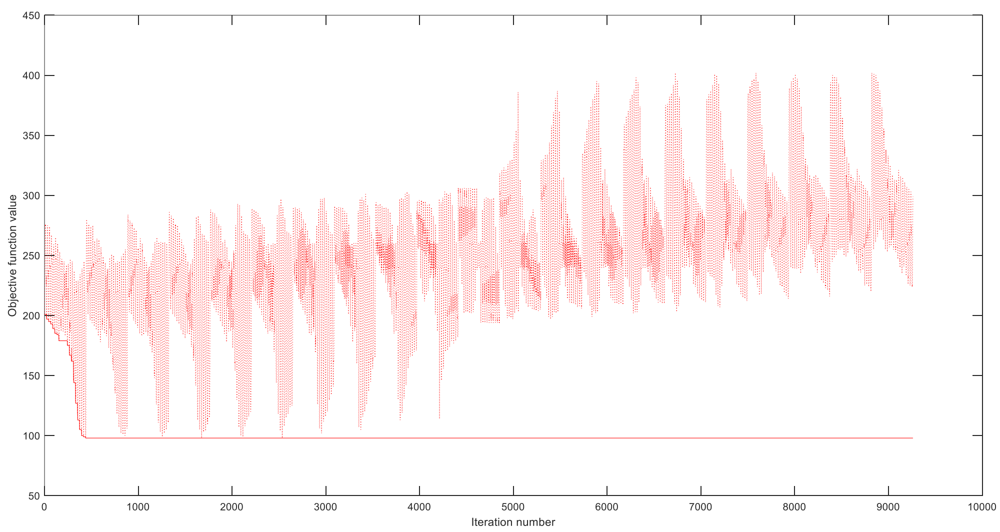
Grid code:

```
% A simple random optimization algorithm. It tries new locations until it
% runs out of time. Delay serves as a way of slowing FunctionPlot.
% It requires a function for optimization (any function from folder
% "FunctionsForOptimization"
clear all
addpath FunctionsForOptimization
%% Optimization task:
FunctionForOptimization = str2func('zadanie3_TR');
%% Adjustable parameters:
MaxRangeX = [-10 10];  % Range of parameters for optimization
MaxRangeY = [-10 10];
MaxRangeZ = [-10 10];
MaxSteps = 100;         % How many iterations do we perform?
FunctionPlot = 0;       % change to 0 If you want to get rid of the underlying function plot
PointPlot = 0;          % Change to 0 if you want to get rid of the visualization
ConvergenceColor = 'r'; % Change color of the convergence curve here
%close all              % Comment this out if you want to have many convergence curves plotted
```

```
ViewVect = [0,90];             % Initial viewpoint
Delay = 0.001;                 % Inter-loop delay  - to slow down the visualization
FunctionPlotQuality = 0.05;    % Quality of function interpolation. Lower for a quicker run
%% Map initialization
InitialRangeX = MaxRangeX;      % This is the range from which we can draw points.
InitialRangeY = MaxRangeY;
InitialRangeZ = MaxRangeZ;

%% Map visualization (this code is not used for problem solving)
if(FunctionPlot == 1)
    figure(1);
    clf
        vectX = [MaxRangeX(1):FunctionPlotQuality:MaxRangeX(2)];
        vectY = [MaxRangeY(1):FunctionPlotQuality:MaxRangeY(2)];
        vectZ = [MaxRangeZ(1):FunctionPlotQuality:MaxRangeZ(2)];
        [X,Y,Z] = meshgrid(vectX,vectY,vectZ);    indx = 1;  indy = 1;
        for x = vectX
            indy = 1;
            for y = vectY
                Val(indx,indy) = FunctionForOptimization([x,y]);
                indy = indy + 1;
            end
            indx = indx + 1;
        end
        mesh(X,Y,Val);    surf(X,Y,Val,'LineStyle','none');
        view(ViewVect);   colormap(bone);    hold on
else end

%% Storing of a best solution
    CurrentMin = 50000;
    ResultX = 1;
    ResultY = 1;
    ResultZ = 1;
%% The main optimization loop
    EndingCondition = 0;
    iter = 0;
    tic;
    step_x = 1;
    for NewX = MaxRangeX(1):step_x:MaxRangeX(2)
        for NewY = MaxRangeY(1):step_x:MaxRangeY(2)
            for NewZ = MaxRangeZ(1):step_x:MaxRangeZ(2)
        iter = iter + 1;
% If you'd like to provide function as a 2D image or use here any other objective function,
% following line needs to be modified. The 0 passed to the function denotes the fact,
% that the function is constant in time.
        CurrentValue =  FunctionForOptimization(NewX,NewY,NewZ);
        if(CurrentValue < CurrentMin)
            CurrentMin = CurrentValue;  % Storing of a historically best result
            ResultX = NewX;
            ResultY = NewY;
            ResultZ = NewZ
            % FunctionPlot (green, if we have a new minimum):
            if(PointPlot == 1)
                figure(1);   plot3(NewY, NewX, NewZ, CurrentValue,'.g'); hold on
```

```
            end
        else
            % FunctionPlot (red, if we don't have a new minimum):
            if(PointPlot == 1)
                figure(1);   plot3(NewY, NewX, NewZ, CurrentValue,'.r'); hold on
            end
        end

        % Command-window stuff for monitoring of algorithm's progress:
        SimTime = toc;
        clc
        fprintf('\nCurrent best:  %f',CurrentMin);
        fprintf('\nCurrent:       %f',CurrentValue);
        fprintf('\n\n\n');
        fprintf('\nIteration:     %d',iter);
        fprintf('\nTime:          %d',SimTime);
        BestHistory(iter) = CurrentMin;         % Here we store our historically best result
        CurrentHistory(iter) = CurrentValue;    % Here we store our currently investigated result
        % If we'd like to slow down the simulation - this line is where it
        % is done:
        pause(Delay);
    end
        end
    end
    figure(2);
    plot(BestHistory,'Color',ConvergenceColor); hold on
    plot(CurrentHistory,'Color',ConvergenceColor,'LineStyle',':'); hold on
    xlabel('Iteration number');
    ylabel('Objective function value');
```

Grid search optimization:



Code for genetic algorithm without stop condition:

```
% A simple random optimization algorithm. It tries new locations until it

% runs out of time. Delay serves as a way of slowing FunctionPlot.

% It requires a function for optimization (any function from folder

% "FunctionsForOptimization"

clear all

addpath FunctionsForOptimization

%% Optimization task:

FunctionForOptimization = str2func('zadanie6_TR');

%% Adjustable parameters:

InitialStep = 3; % Exploration/exploitation balance parameters:

P1 = 2;

P2 = 2;

lines=10;

P_size = 40; % Population size

n = 10; % Parameter n for n best succession

Step = 0.1; % Mutation range

MaxRangeX = [-10 10]; % Range of parameters for optimization

MaxRangeY = [-10 10];

MaxRangeZ = [-10 10];

MaxSteps = 20; % How many iterations do we perform?

FunctionPlot = 0; % change to 0 If you want to get rid of the underlying function plot

PointPlot = 0; % Change to 0 if you want to get rid of the visualization

ConvergenceColor = 'r'; % Change color of the convergence curve here

%close all % Comment this out if you want to have many convergence curves plotted

ViewVect = [0,90]; % Initial viewpoint

Delay =0.001; % Inter-loop delay - to slow down the visualization

FunctionPlotQuality = 0.05; % Quality of function interpolation. Lower for a quicker run

%% Map initialization

InitialRangeX = MaxRangeX; % This is the range from which we can draw points.

InitialRangeY = MaxRangeY;

InitialRangeZ = MaxRangeZ;

%% Map visualization (this code is not used for problem solving)

TimePercent = 0;

if(FunctionPlot == 1)

 figure(1);

 clf

 vectX = [MaxRangeX(1):FunctionPlotQuality:MaxRangeX(2)];

 vectY = [MaxRangeY(1):FunctionPlotQuality:MaxRangeY(2)];

 [X,Y] = meshgrid(vectX,vectY); indx = 1; indy = 1;

 for x = vectX

 indy = 1;

 for y = vectY

 Val(indx,indy) = FunctionForOptimization([x,y]);

 indy = indy + 1;

 end

 indx = indx + 1;

 end

 mesh(X,Y,Val); surf(X,Y,Val,'LineStyle','none');

 view(ViewVect); colormap(bone); hold on

else end

%% Storing of a best solution

 CurrentMin = 50000;
```

```
ResultX = 1;
ResultY = 1;
%% The main optimization loop
EndingCondition = 0;
iter = 0;
tic;
   for k = 1:P_size
        Population(k).OF = Inf;
        for line=1:lines
        Population(k).Parameters.W1(line) = InitialRangeX(1) + ...
        rand()*(InitialRangeX(2) - InitialRangeX(1));
        Population(k).Parameters.W2(line) = InitialRangeY(1) + ...
        rand()*(InitialRangeY(2) - InitialRangeY(1));
        Population(k).Parameters.B(line) = InitialRangeY(1) + ...
        rand()*(InitialRangeY(2) - InitialRangeY(1));
   end
   end
while(EndingCondition == 0);
   iter = iter +1;
   Step(iter) = InitialStep * (1/(1+exp((iter-(MaxSteps/P1))/P2)));
   for k = 1:P_size
           Population(k).OF = FunctionForOptimization(Population(k).Parameters.W1,Population(k).Parameters.W2,Population(k).Parameters.B);
   end
   [~,Indices] = sortrows([Population(:).OF]');
   Population(Indices(1))
   if(FunctionPlot == 1)
       figure(1);
       clf
       surf(X,Y,Z,Val,'LineStyle','none');
       view(ViewVect)
       colormap(bone)
       hold on
   else
   end
   if(PointPlot == 1)
   for k = 1:1:P_size
       plot3([Population(k).Parameters(2)],[Population(k).Parameters(1)],[Population(k).OF],'.r'); hold on
   end
   end
   BestHistory(iter) = Population(Indices(1)).OF;
   CurrentHistory(iter) = Population(Indices(floor(P_size/2))).OF;
   BestIndividualGenome(iter) = Population(Indices(1));
   NewPopulation(1) = Population(Indices(1));
   for k = 2:1:P_size
       ind1 = randi(n);
       ind2 = randi(n);
       NewPopulation(k) = Population(Indices(ind1));
      NewPopulation(k).Parameters.W1(2) = Population(Indices(ind2)).Parameters.W1(2);
       NewPopulation(k).Parameters.W2(2) = Population(Indices(ind2)).Parameters.W2(2);
        NewPopulation(k).Parameters.B(2) = Population(Indices(ind2)).Parameters.B(2);
       NewPopulation(k).Parameters.W1 = NewPopulation(k).Parameters.W1 +Step(iter)*randn(size(NewPopulation(k).Parameters.W1));
       NewPopulation(k).Parameters.W2 = NewPopulation(k).Parameters.W2 +Step(iter)*randn(size(NewPopulation(k).Parameters.W2));
       NewPopulation(k).Parameters.B = NewPopulation(k).Parameters.B +Step(iter)*randn(size(NewPopulation(k).Parameters.B));
       NewPopulation(k).OF = Inf;
```

```
    end
    Population = NewPopulation;
     SimTime = toc;
     clc
     fprintf('\nCurrent best: %f',BestHistory(end));
     fprintf('\nIteration: %d',iter);
    fprintf('\nTime: %d',SimTime);
     if(iter > MaxSteps)
     EndingCondition = 1;
     else
     end
     pause(Delay);
end
figure(2);
plot(BestHistory,'Color',ConvergenceColor); hold on
plot(CurrentHistory,'Color',ConvergenceColor,'LineStyle',':'); hold on
xlabel('Iteration number');
ylabel('Objective function value');
figure(4);
plot(Step)
xlabel('iteration');
ylabel('mutation step value');
```

If we want to obtain stop condition we have to include this fragment of code:

```
    % with stop
    testing(iter)=zadanie6_TE(Population(Indices(1)).Parameters.W1,Population(Indices(1)).Parameters.W2,Population(Indices(1)).Parameters.B);
    [val,ind]=min(testing);
    if(iter-ind>2)
         EndingCondition = 1;
         Population(Indices(1))=BestIndividualGenome(ind);
    end
    %
```

It works as following.

If the error value of the best individual is not improving in the testing set for longer than two iterations, we stop the training.

In the next exercises we repeated the procedure of training and validation and we obtained such results for corresponding algorithms.

	TR	VA
Grid	98	43
Genetic	97	44
3-line Genetic	81	41
5-line Genetic	83	41
7-line Genetic	42	19
10 line genetic without stop condition	5	11
10 line genetic with stop condition	8	6

As we can see if we increase the number of lines in genetic algorithm, we obtain better solution. Although when we decrease the size of dataset and increase the number of genetic lines without stoping condition we obtain worse solution than the one with stop condition. That is because, when we include stop condition we prevent overfiting, which is cricuial in obtaining satisfying solution.