Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Signal processing and identification in control of mechatronic devices

Signal processing and identification in monitoring of mechatronic devices

Topic: Classification and regression

Aim: Classification of 2D data using a simple classifier

Issues covered: Data clusters, dividing data for training and testing, outliers, overfitting, decision tree classifier, classifier training

Classification dataset generation

We'll start today's work with generation of a simple classification problem. Let's assume that our data are divided into two classes, there are 5 clusters of data in A class and 4 clusters of data in B class. Using this code we will generate cluster centers and display them on screen:

```
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]);

save Clusters Clusters
```

Exemplary result should look similar to that shown in Fig 1.

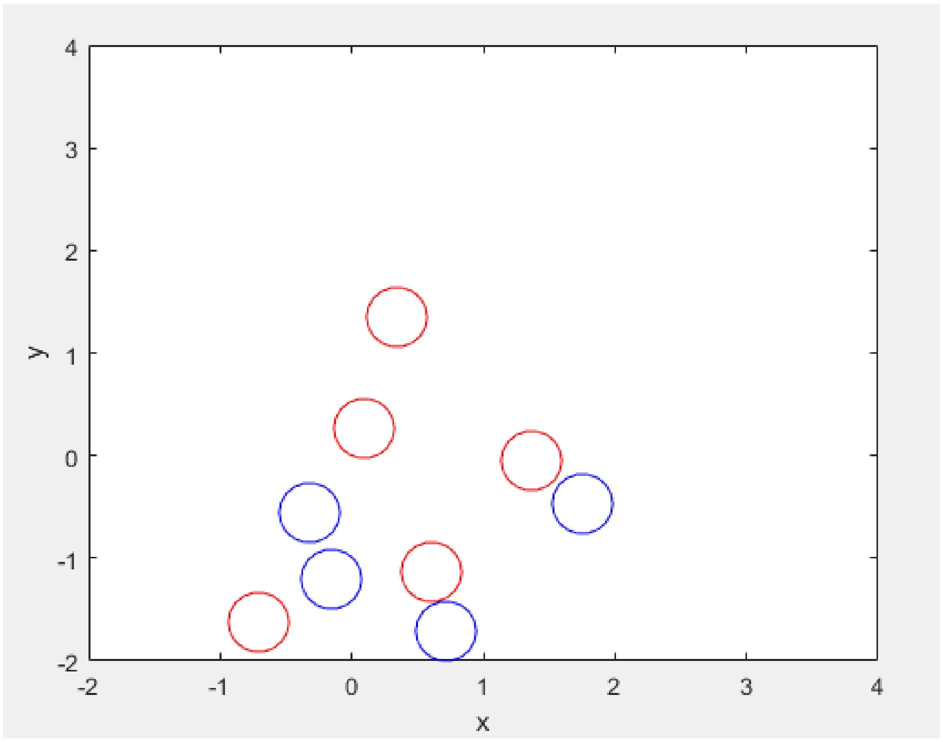

Fig 1 – Cluster centers for two-class classification problem. The data are not linearly separable.

Task 3.1: Please run the script to generate your data clusters. Circles marked in the plot should not overlap, the problem should NOT be linearly separable. After obtaining a result that fulfills these requirements ask LA to check whether data look OK. Then save *Clusters* structure on disk – this will define your classification task for the rest of this instruction.

After obtaining cluster centers we can fill them with data. This code uses *Clusters* structure so make sure that you are using the very same clusters set that was approved by LA:

```
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])


save DATA DATA
```

Note that *DataDivision* value determines split of data into classes (we can have more examples in one class than in the other) while *v* value determines how heavy are tails of the data distribution – the lower the parameter, the heavier are distribution's tails (we are using T-Student's distribution here). These parameter values allowed for obtaining dataset visible in Fig 2. It is worth noting that using T-Student distribution instead of a gaussian allowed for generation of a bunch of outliers and conveniently connected clusters to each other while still maintaining separate cluster centers, so we've obtained a nice and interesting classification problem.

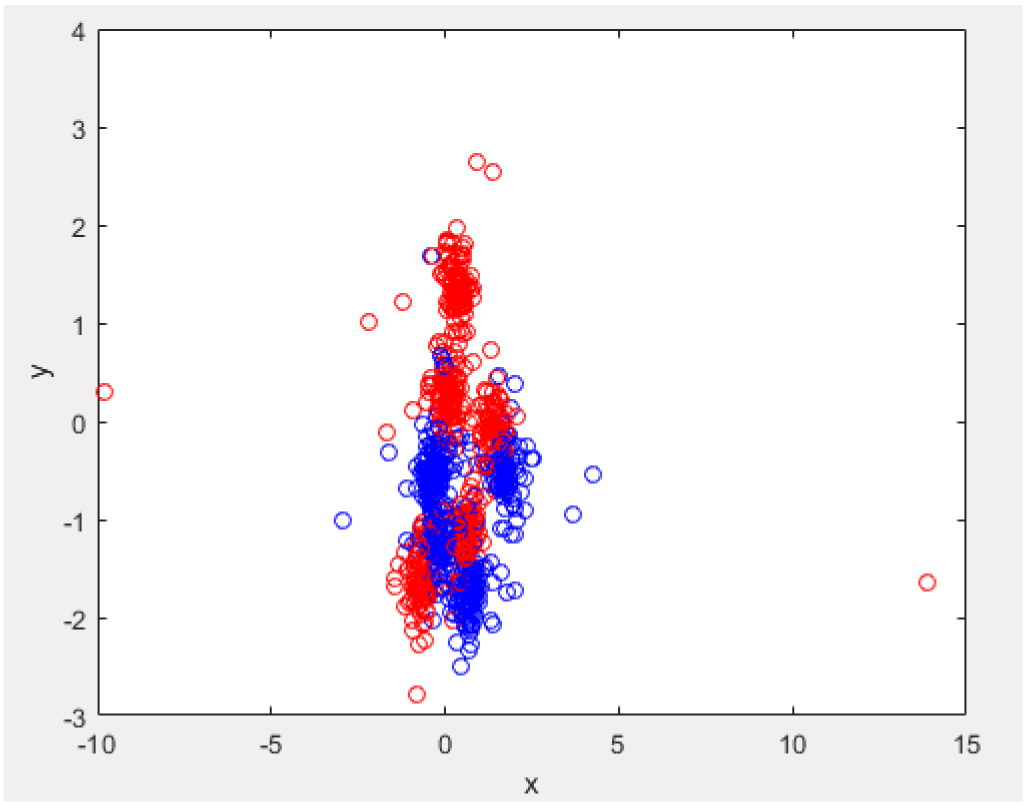

Fig 2 – Generated dataset

Training and testing data

Our data should now be prepared for the purpose of training and testing classifiers. We'll randomly divide it into three subsets: training, validation and testing, with 50%, 25% and 25% of data samples, respectively. In our case it would equal 500, 250 and 250 samples. To this end we can use the following code:

```
Indices = randperm(length(DATA));
DATA_permutated = DATA(:,Indices)

TR_number = ceil(length(DATA)*0.5);
VA_number = ceil(length(DATA)*0.25);
TE_number = ceil(length(DATA)*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
```

Task 3.2: Using your *Clusters* structure generate dataset containing 1000 samples divided equally among two classess, using *v=2.* Save generated dataset on disk so it could be used to train and test classifiers. Save both the original datatset (DATA) and its divided subsets. Save also a script that was used to generate data – so it could later be used to generate other datasets as well.

Design of a simple linear classifier

We've made sure that our dataset is not linearily separable. Nontheless, we will try and divide these data using a straight line. To design a classifier we'll just ask a simple question regarding each data point: “Is this point above or below a predefined line?”

Lets formulate equation for it:

$$W1 * x1 + W2 * x2 + b > 0 \quad (1)$$

And now lets find such *W1* and *W2,* to maximize efficiency of classification. Our classifier should look like this:

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

Such a classifier can be saved as a function and then used to classify our data as in here:

```
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
```

The above script rendered results that are shown in Fig 3. There were 34 errors in A class and 61 errors in B class – not great, but so far we've picked W1 and W2 parameters quite randomly.

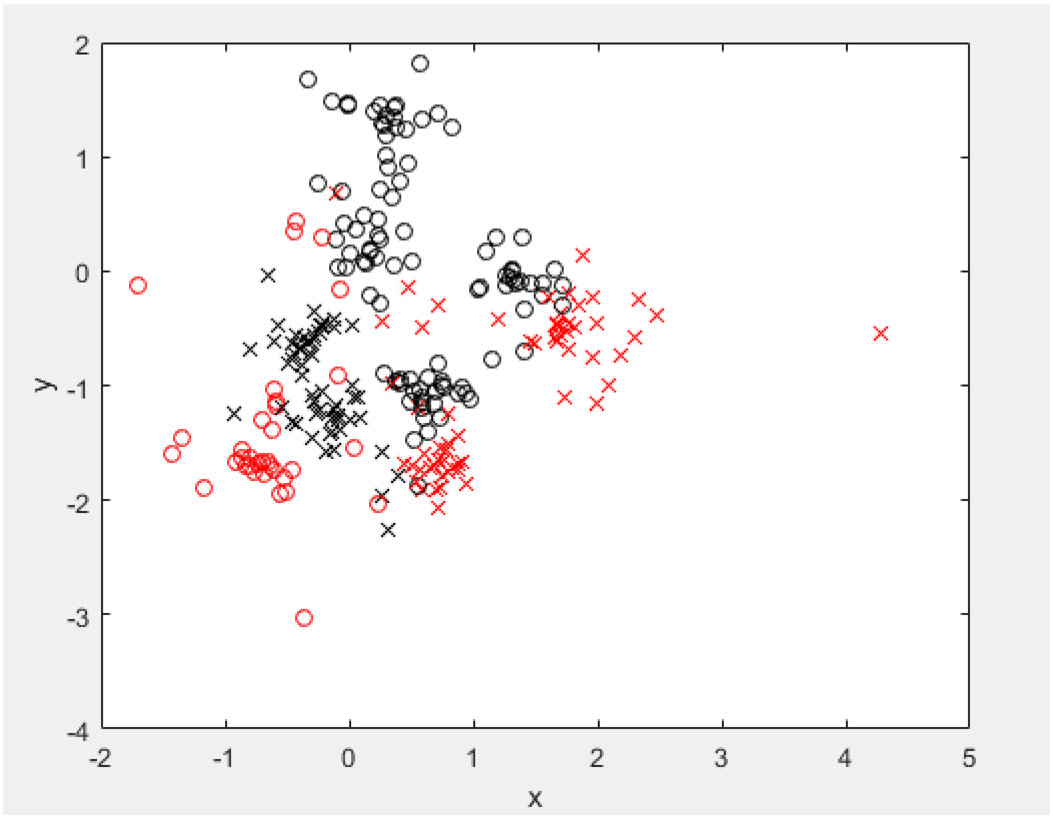

Fig 3 – Results of classification. Errors are marked in red

Could we score better? Lets note that we have a two-parameter, one-criterion optimization problem. Our objective function that we would want to minimize is a sum of errors in both classes.

We already have tools that can deal with this kind of problems, namely: various optimization algorithms developed in scope of 1st and 2nd laboratory. It is worth noting that despite having a *continuous* problem (we can assign floating point values to ***W1*** and ***W2***) we cannot use gradient descent algorithm. The reason for this is as follows: If we move our separation line and cross with it location of any data point the objective function value would not change gradually – instead it will be incremented or decremented. Very small change of any parameter would likely not cause any change of the objective function value.

In order to use any of the solutions developed before (e.g. 1+1 or genetic algorithm) we'll need to save our script for using a classifier as a function – taking parameter values as input (marked in green in the code), returning a sum of errors (marked in cyan) and commenting out lines for plotting (marked in grey).

Task 3.3: Using scripts developed in class 1 and 2 optimize parameters of your linear classifier. Use a grid search algorithm. To optimize parameters use training data (**TR_DATA**). After successful optimization test your classifier using validation data (**VA_DATA**). Save the script so it could be checked by the LA later.

Task 3.4: Using scripts developed in class 1 and 2 optimize parameters of your linear classifier. Use a genetic algorithm. To optimize parameters use training data (**TR_DATA).** After successful optimization test your classifier using validation data **(VA_DATA).** Save the script so it could be checked by the LA later.

Classifier with more degrees of freedom

Our classifier won't allow for correct classification or even just relatively small error value – because it can't classify the data that are not linearily separable. Lets allow it to have more *degrees of freedom* – so it could provide more complex classification rule.

We'll modify *InitialClassifier* by adding a *ClassLikelihood* variable that will be incremented each time when condition (1) is met and decremented each time the condition (1) is not met. Number of degrees of freedom will be increased by increasing length of vectors of *Parameters.* Our new *Parameters* structure might look like this:

```
Parameters =
       struct with fields:

Parameters.W1 = [-1.7,4.8,0];
Parameters.W2 = [-1,-1,1];
Parameters.B = [-2.7,4,0];
```

Inside the classifier we'll thus need checking of all the linear conditions, using e.g. such *for* loop:

```
for k = 1:length(Parameters.W1)
      ...
end
```

In each passing of this loop we'll check whether the point is classified into Class A or Class B by each consecutive line incrementing or decrementing *ClassLikelihood.* If after this *for* loop it is positive, the classifier will return 1 indicating that it belongs to class A. Otherwise we'll return 0. Exemplary result of such a classification might look as in Fig. 4. It can be seen that the overall error was greatly reduced (to 63) and the classifier is able to develop a non-linear classification rule.

After genetic optimization the classifier with 10 degrees of freedom is able to reduce error number to 28. The result of this classification is shown in Fig 5.

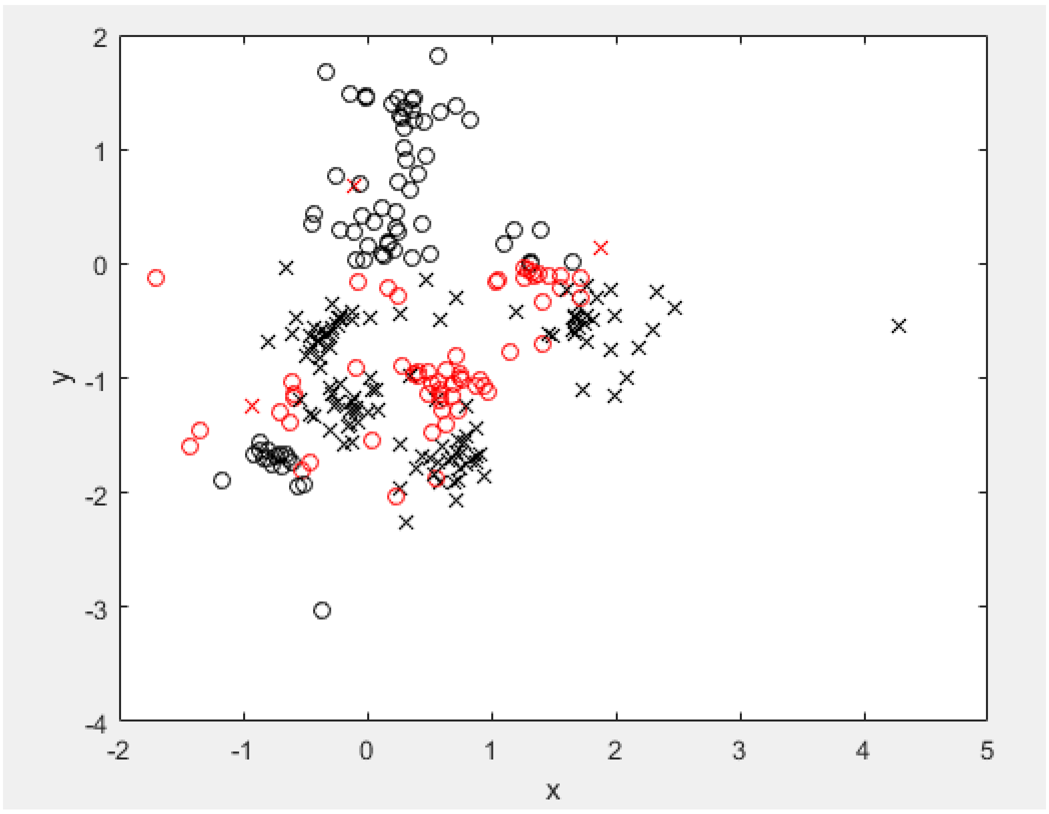

Fig 4 – Result of classifcation with classifier using three linear conditions.

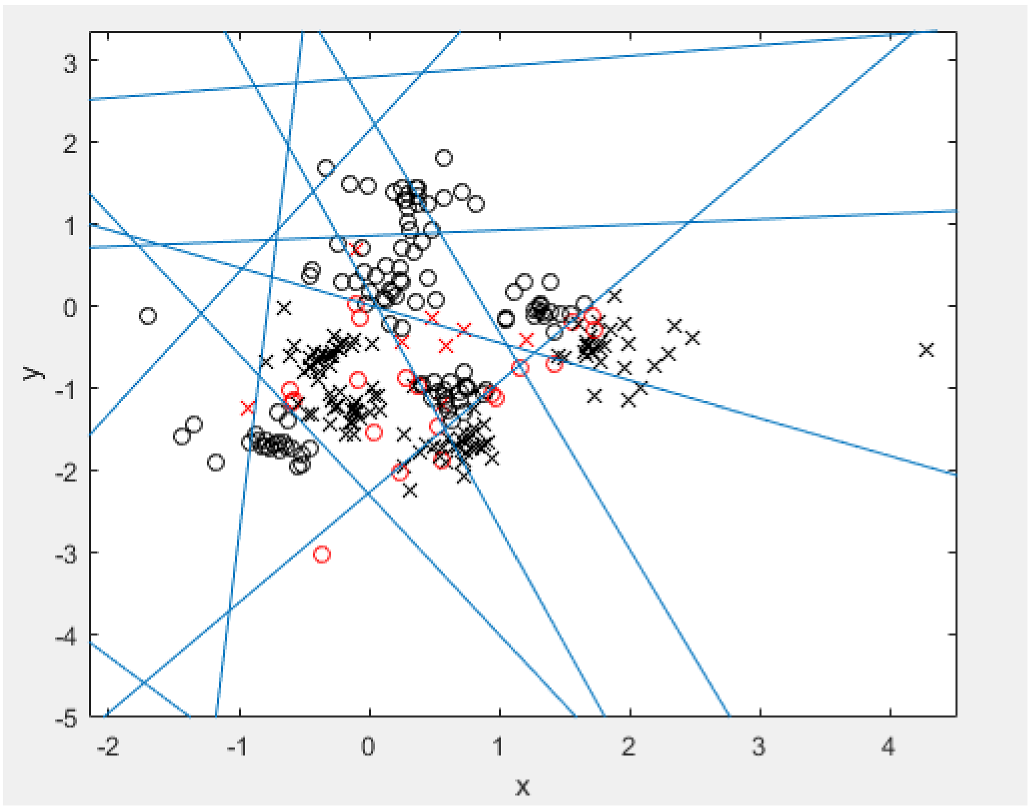

Fig 5 – 10-lines Classifier optimized with genetic algorithm resulted in 28 errors only. Linear conditions used are plotted as blue lines – each polygon contain a field in which data are classifier to particular class.

Task 3.5: Using scripts developed during instructions 1 and 2 optimize parameters of a 3, 5 and 7-line classifiers. In order to train a classifier use training data (that is: **TR_DATA**), Use a genetic algorithm. After optimization test the obtained solution using validation dataset (**VA_DATA**). Store the code for further evaluation by the LA.

Overfitting possibility

Was the efficiency of classification obtained in training and testing similar so far? Probably yes, because number of adjustable parameters of a classifier was so low that overfitting was practically not possible. However, if we had less training data and more advanced classifier, it could have been an issue. We have to protect the classifier from it using a validation dataset during training.

First, however, let us consider a problem in which overfitting might occur. This time **let us generate only 100 data samples (including 50 training ones, 25 for validation and 25 for test)**. Exemplary dataset might look as in Fig. 6

Now, we will optimize the classifier using genetic algorithm similarily as before (using **TR_DATA**) but for a stopping criterion, instead of checking only number of iterations of our optimization algorithm we'll also check performance on validation data (**VA_DATA** set) and stop the training as soon as it starts to deteriorate. In other words, we'll assess fitness of individuals on the basis of **TR_DATA** only (and based on that fitness select individuals for reproduction) but check efficiency of selected elite individual on **VA_DATA** and use results of this latter check to either continue or stop the GA. If stopping criterion based on **VA_DATA** efficiency deterioration triggers, GA should return the historical best solution (not a current, deteriorated one).

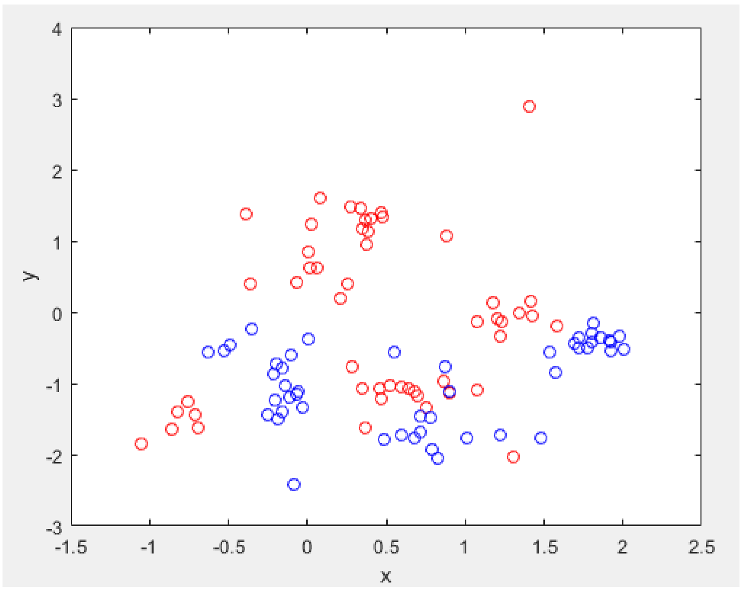

Fig 6 – Reduced dataset.

Task 3.6: Optimize parameters of a 10-line classifier (30 degrees of freedom) on a basis of a **small** training dataset (containing 50 samples). As a stopping criterion use efficiency on validation data (**VA_DATA**). Then, compare efficiency of this new training method with the previous one (10-line classifier with number of interations as the only stopping criterion) using testing data (**TE_DATA**). Save the script so it could be assessed later by the LA.

Additional tasks:

Task 3.7: Up until now we rarely used a final testing dataset (TE_DATA). Optimize metaparameters of a GA used to train our 10-line classifier. Use large (1000 sample) dataset. After each run or series of runs of GA test the obtained solution using VA_DATA. Finally, after deciding which parameters' values appear to be optimal test the final optimized classifier on a TE_DATA. Provide statistical results in each case. Is TE_DATA efficiency similar to VA_DATA efficiency obtained for the final classifier?

Task 3.8: Let the classification problem have non-uniform distribution of data among classess. Let *DataDivision* be equal to 0.1. Does it affect overall final results? Des it affect results in particular classess? Is the obtained percentage efficiency different than before? What can we do in such a situation? Solve this problem using two approaches: (1) by reduction of data in more numerous data class and (2) by using as objective function weighted sum of errors in both classes – where weight for less numerous class should be proportionally higher than that for more numerous class. Which of this two approaches render statistically better overall results? Which allowed for increase of efficiency in less numerous dataset?

Task 3.9: Build a classifier that is using circles instead of lines (Variable *ClassLikelihood* should be incremented or decremented if a point is inside or outside of a particular circle). Each circle should be defined by coordinates of its center and its radius. Negative radius mean that the class the circle respond to is reversed (class B inside, class A outside of a circle). Train the classifier using GA. Is this problem more or less dificult than before? Why?

Task 3.10: How many degrees of freedom should our classifier have for the best efficiency? Check it using two approaches: (1) – train classifiers with different number of degrees of freedom starting from 8 lines, increasing this parameter e.g. by 3 until the efficiency clearly starts to decrease. What is a source of this deterioration? Is it the increase of diffifulty of training? Is it the overfitting? (2) Include number of lines used by the classifier in GA genome (the algorithm is allowed to mutate genome of an individual in such a way, that it adds or deletes one line. Are the results in both approaches consistent? Are we able to obtain classifier structure that was found optimal in (1) approach also using (2)?