Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Przetwarzanie sygnałów i identyfikacja w sterowaniu urządzeń mechatronicznych,

Przetwarzanie sygnałów i identyfikacja w monitorowaniu urządzeń mechatronicznych

Temat: Klasyfikacja i regresja

Cel ćwiczenia: Wykonanie klasyfikacji danych 2D

Zagadnienia: Generowanie danych, klastry, uczenie klasyfikatora, klasyfikator oparty na drzewie decyzyjnym, outliery, przeuczenie

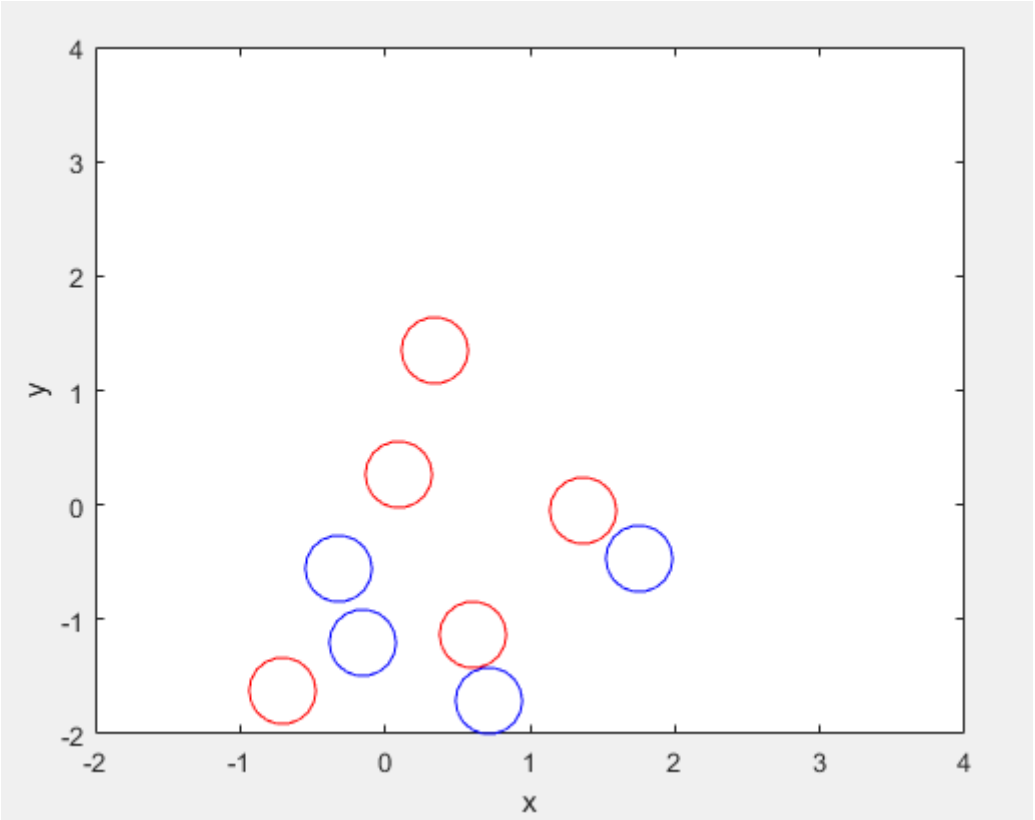
Prowadzący: dr inż. Ziemowit Dworakowski

Przygotowanie zbioru danych

Zadanie rozpoczniemy od wygenerowania prostego problemu do klasyfikacji. Powiedzmy, że nasze dane podzielone są na dwie klasy, w klasie A znajduje się 5 klastrów, w klasie B mamy 4 klastry. Za pomocą poniższego kodu wygenerujemy centra tych klastrów oraz wyświetlimy je na wykresie:

```
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
```

Przykładowy wynik może wyglądać na przykład tak, jak na rys. 1:



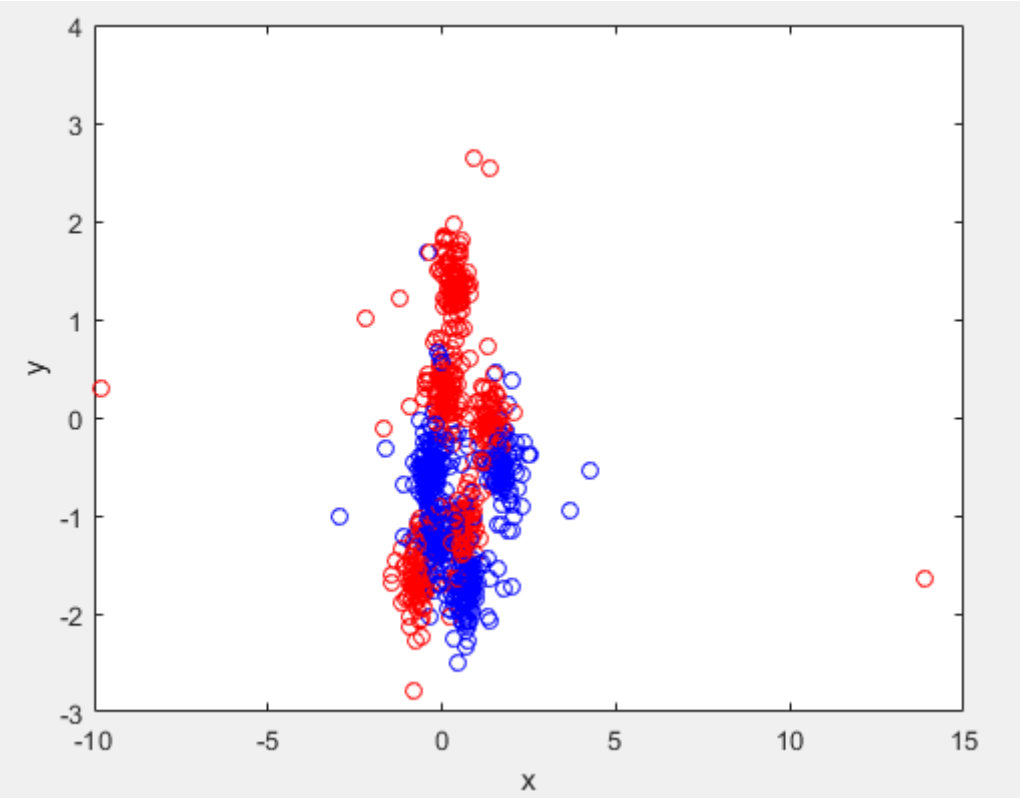
Rys 1 - Przykładowe wygenerowane, nieseparowalne liniowo klastry

Zadanie 3.1: Proszę uruchomić skrypt w celu wygenerowania rozkładu klastrów. Zaznaczone na rysunku okręgi powinny być rozłączne a problem nie powinien być liniowo separowalny. Po uzyskaniu wyniku który spełnia powyższe warunki skontaktuj się z prowadzącym w celu zatwierdzenia zadania które rozwiązywane będzie przez pozostałą część laboratorium a następnie zapisz na dysku wygenerowaną strukturę *Clusters*

Po wygenerowaniu centrów klastrów możemy "wypełnić je" danymi. Poniższy kod korzysta z uprzednio przygotowanej struktury *Clusters* i za jej pomocą generuje dane do klasyfikacji oraz wyświetla je na obrazie:

```
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
```

Zwróćmy uwagę na to, że parametr *DataDivision* określa jak dużo danych "wyląduje" w każdej klasie a parametr *v* określa "ciężar ogonów" rozkładu T-Studenta (im mniejszy - tym cięższe ogony). Powyższe parametry pozwoliły na wygenerowanie zestawu danych przedstawionego na Rys 2. Warto zauważyć, że zastosowanie rozkładu "T Studenta" pozwoliło na "uzyskanie" kilku outlierów oraz doprowadziło do "sklejenia ze sobą" poszczególnych klastrów.



Rys 2 - Wygenerowane przykładowe dane

Zbiór uczący i testowy

Nasze dane należy teraz przygotować do klasyfikacji poprzez wykonanie losowego podziału naszego zbioru na podzbiory uczące, walidacyjne i testowe. Przydzielimy do tych danych odpowiednio 50%, 25% i 25% danych, tj. 500, 250 oraz 250 próbek. Najprościej można to zrobić za pomocą poniższego skryptu:

```
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
```

Zadanie 3.2: W oparciu o swoją strukturę *Clusters* proszę wygenerować zbiór 1000 punktów danych podzielonych równomiernie pomiędzy dwie klasy - z zastosowaniem parametru $v=2$. Tak zapisany zbiór danych proszę zapisać na dysku - na potrzeby przyszłej klasyfikacji - zarówno w pełnej wersji (DATA) jak i w podziale na zbiory uczący, walidacyjny i testowy. Proszę zapisać skrypt do generowania danych aby był łatwo dostępny w kolejnych zadaniach.

Budowa prostego systemu klasyfikacji liniowej

Postaraliśmy się o to, aby wygenerowane powyżej dane były nieseparowalne liniowo. Mimo wszystko spróbujemy jednak podzielić je linią prostą. W przypadku każdego punktu danych zadawać będziemy pytanie "czy punkt położony jest powyżej zdefiniowanej prostej?"

Musimy zatem zbudować warunek klasyfikacji:

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

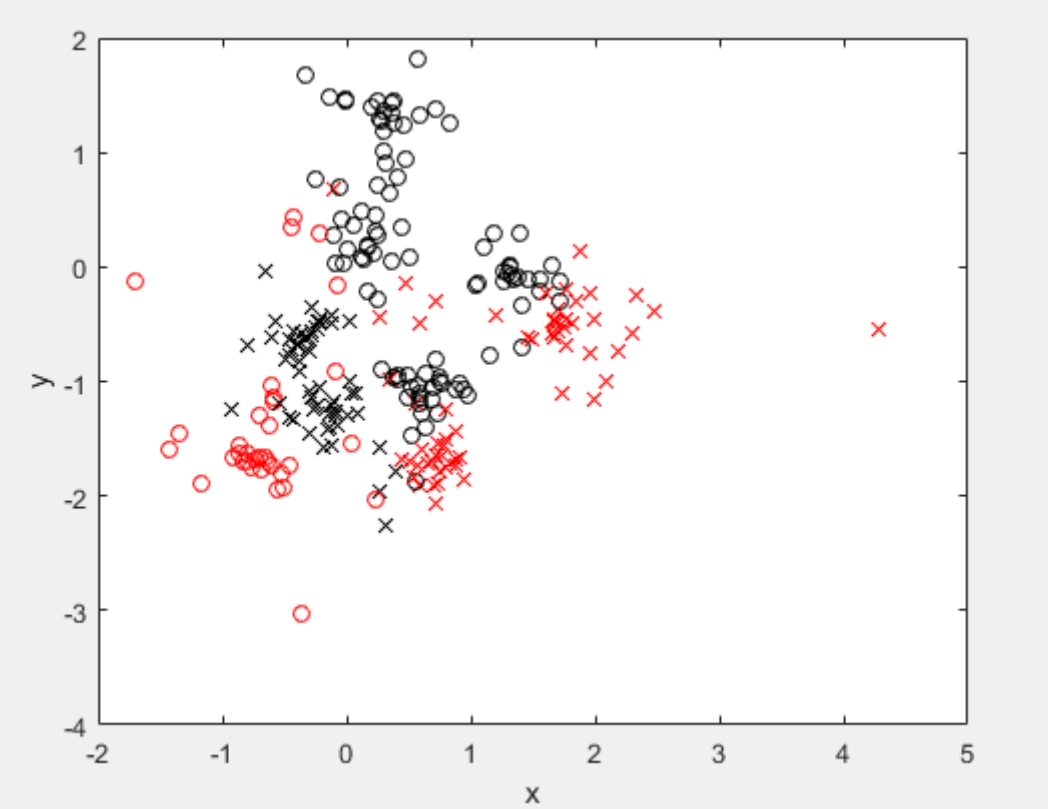
A następnie spróbujemy znaleźć takie parametry *W1* oraz *W2,* aby zmaksymalizować skuteczność klasyfikacji. Będziemy potrzebowali klasyfikatora, który może wyglądać np. tak:

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

Taki klasyfikator można następnie zapisać i wykorzystać do klasyfikowania naszych danych w ten sposób:

```
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
```

Powyższy skrypt w naszym problemie przykładowym pozwolił na uzyskanie wyniku przedstawionego na Rys. 3. Otrzymaliśmy 34 błędy w klasie A i 61 błędów w klasie B - co wynika z przypadkowego doboru prostej według której dzielimy dane.



Rys 3 - Wynik klasyfikacji. Błędy zaznaczono kolorem czerwonym

Czy ten wynik mógłby być lepszy? Zwróćmy uwagę, że mamy tutaj do czynienia z problemem optymalizacyjnym, dwuparametrycznym, jednokryterialnym. Kryterium optymalizacji będzie tutaj suma błędów (powinna być jak najmniejsza).

Mamy już zbudowane narzędzia do rozwiązywania tej klasy problemów - tzn. algorytmy optymalizacyjne opracowywane w ramach pierwszego i drugiego laboratorium. Zwróćmy uwagę na fakt, że o ile problem jest *ciągły* - tzn. możemy wybrać wartości parametrów ***W1*** oraz ***W2*** z dowolną dokładnością, zastosowanie algorytmu gradientowego nie będzie miało sensu, bowiem wartość funkcji celu będzie się zmieniała wyłącznie wtedy, gdy poruszenie prostą pozwoli na "przeskoczenie" któregoś punktu z jednej do drugiej klasy. Bardzo mała zmiana wartości któregokolwiek z parametrów najprawdopodobniej nie spowoduje więc zmiany wartości funkcji celu.

Aby móc zastosować w prosty sposób używany wcześniej algorytm optymalizacyjny, poprzedni skrypt trzeba będzie zamknąć w funkcji przyjmującej jako argument wartości parametrów W1 oraz W2 (zaznaczone w skrypcie na kolor zielony), zwracającej sumę błędów (zaznaczoną na błękitno) oraz zakomentować generowanie wykresu (zaznaczone na szaro)

Zadanie 3.3: W oparciu o skrypty stworzone na zajęciach 1 i 2 zoptymalizuj położenie prostej danej parametrami *Parameters.W1* oraz *Parameters.W2*. Zastosuj algorytm grid search. Do uczenia klasyfikatora zastosuj zbiór treningowy (tj. **TR_DATA**). Po zakończeniu optymalizacji przetestuj otrzymany klasyfikator na zbiorze walidacyjnym (tj. **VA_DATA**). Zapisz skrypt do późniejszej oceny przez prowadzącego.

Zadanie 3.4: W oparciu o skrypty stworzone na zajęciach 1 i 2 zoptymalizuj położenie prostej danej parametrami *Parameters.W1* oraz *Parameters.W2*. Do uczenia klasyfikatora zastosuj zbiór treningowy (tj. **TR_DATA**). Zastosuj algorytm genetyczny. Po zakończeniu optymalizacji przetestuj otrzymany klasyfikator na zbiorze walidacyjnym (tj. **VA_DATA**). Zapisz skrypt do późniejszej oceny przez prowadzącego.

Rozbudowa klasyfikatora o więcej stopni swobody

Nasz klasyfikator nie pozwoli na poprawne sklasyfikowanie danych, ani nawet na osiągnięcie niewielkiego błędu - ze względu na brak możliwości liniowej separacji danych zapewniony na etapie ich generowania. Spróbujmy zatem dodać mu więcej *stopni swobody* - aby był w stanie zaproponować bardziej złożony mechanizm klasyfikacji.

Zmodyfikujmy funkcję *InitialClassifier* w ten sposób, aby dodać do niej zmienną *ClassLikelihood* - która będzie inkrementowana za każdym razem gdy klasyfikowany punkt danych spełnia warunek **(1)** dla którejś prostej oraz dekrementowany gdy warunek nie jest spełniony.

Ilość stopni swobody definiować będziemy poprzez długość wektorów parametrów, tzn. dla przykładowych dziewięciu stopni swobody struktura *Parameters* może wyglądać następująco:

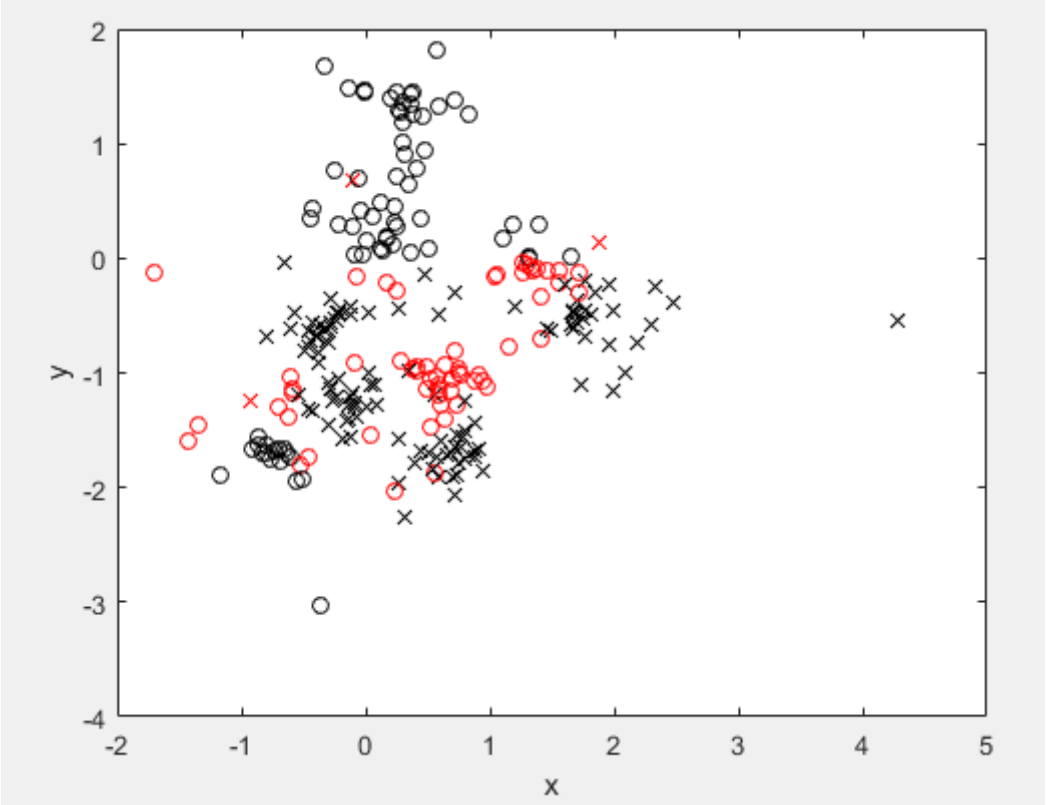
```
Parameters =

  struct with fields:

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

Wewnątrz klasyfikatora będziemy więc potrzebowali przejścia po wszystkich *liniach,* np. taką pętlą *for*:

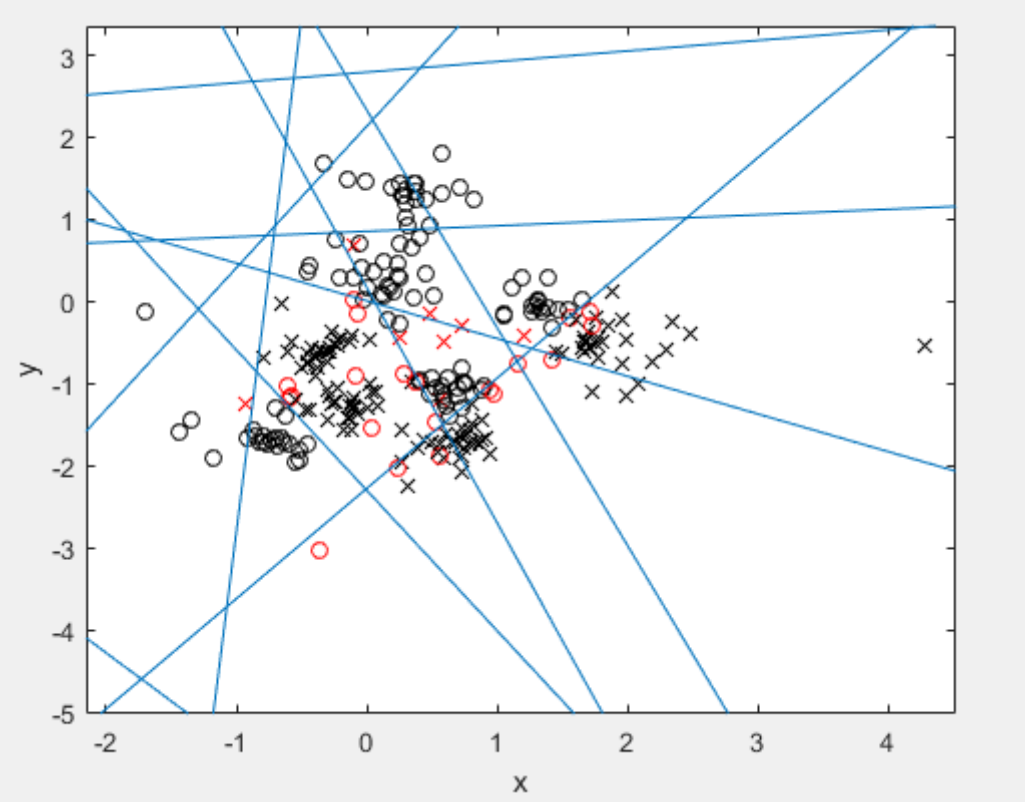
```
for k = 1:length(Parameters.W1)
      …
end
```

a następnie w każdym obiegu pętli realizujemy klasyfikację za pomocą linii prostej, w zależności od wyniku inkrementując bądź dekrementując zmienną *ClassLikelihood.* Jeśli na koniec działania funkcji wartość będzie większa od zera, jako *ClassLabel* zwracamy 1, w przeciwnym wypadku 0. Przykładowy wynik takiego klasyfikatora może wyglądać jak na rysunku 4. Jak widać, klasyfikator nabrał możliwości klasyfikowania w sposób nieliniowy i zredukował błąd do wartości 63 błędów w sumie.

Po wykonaniu optymalizacji klasyfikator o 10 liniach na zaprezentowanym problemie osiąga 28 błędów. Efekt klasyfikacji przedstawiono na rys. 5.



Rys 4 - Wynik klasyfikacji uzyskany przez przykładowy klasyfikator używający trzech linii



Rys 5 - Klasyfikator zoptymalizowany algorytmem genetycznym, o 10 liniach (30 stopniach swobody). Ilość błędów wynosi 28

Zadanie 3.5: W oparciu o skrypty stworzone na zajęciach 1 i 2 zoptymalizuj parametry klasyfikatora o 3, 5 i 7 liniach. Do uczenia klasyfikatora zastosuj zbiór treningowy (tj. **TR_DATA**). Zastosuj algorytm genetyczny. Po zakończeniu optymalizacji przetestuj otrzymany klasyfikator na zbiorze walidacyjnym (tj. **VA_DATA**). Zapisz skrypt do późniejszej oceny przez prowadzącego.

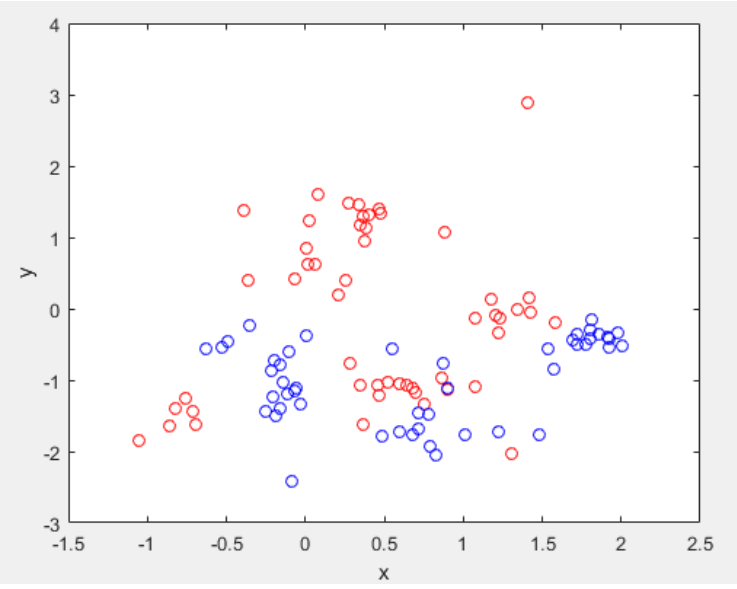
Trening w warunkach przeuczenia

Czy maksymalna skuteczność klasyfikacji osiągana na zbiorze uczącym (na zakończenie optymalizacji) była jak do tej pory zbliżona do skuteczności klasyfikacji osiąganej na zbiorze walidacyjnym? Być może tak, bowiem stosujemy jak do tej pory klasyfikatory o bardzo małej ilości stopni swobody w stosunku do ilości punktów danych co w praktyce uniemożliwia im przeuczenie. W miarę wzrostu złożoności klasyfikatora oraz w przypadku posiadania niedostatecznej ilości danych uczących problem przeuczenia jednak się pojawi - musimy się zatem zabezpieczyć poprzez wykorzystanie zbioru walidacyjnego w trakcie optymalizacji.

Postarajmy się o problem w którym przeuczenie może się pojawić. Niech zbiór danych zawiera zamiast 1000 jedynie 100 próbek (w tym 50 treningowych, 25 walidacyjnych i 25 testowych). Przykładowy zbiór może wyglądać np. tak, jak na Rys. 5.

Teraz optymalizacja będzie wyglądała następująco:

1) Na zbiorze uczącym (TR_DATA) wykonujemy optymalizację tak samo jak poprzednio - tj. w przypadku stosowania algorytmu genetycznego oceniamy przystosowanie osobników. Osobniki lepiej przystosowane przekazują geny dalej, osobniki gorzej przystosowane są odrzucane.
2) W każdej generacji osobnik elitarny jest poddawany ocenie ze względu na zbiór **VA_DATA**. Skuteczność na tym ostatnim zbiorze jest podstawą do przerwania treningu klasyfikatora: brak poprawy skuteczności na zbiorze walidacyjnym przez wybraną ilość iteracji algorytmu (np. 5) powinien spowodować zatrzymanie nauki i zwrócenie osobnika historycznie najlepiej przystosowanego.



Rys 5 - Zbiór danych o znacznie mniejszej liczności

Zadanie 3.6: Zoptymalizuj parametry klasyfikatora o 30 stopniach swobody (10 linii) działającego na **małym** zbiorze uczącym (zawierającym 50 próbek). Jako warunek zatrzymania treningu zastosuj skuteczność na zbiorze walidacyjnym (**VA_DATA**). Następnie przetestuj klasyfikator uczony nową metodą z poprzednim (10-liniowy klasyfikator mający ilość iteracji jako jedyny warunek zatrzymania) używając zbioru testowego (**TE_DATA).**

Zapisz skrypty do późniejszej oceny przez prowadzącego.

Zadania dodatkowe i rozszerzające:

Zadanie 3.7: Do tej pory rzadko korzystaliśmy z finalnego zbioru testowego (**TE_DATA)**. Wykonaj optymalizację metaparametrów algorytmu genetycznego do rozwiązywania początkowego problemu klasyfikacyjnego (tj. wygenerowanego w postaci 1000 punktów danych). Każdorazowo oceniaj działanie AG na podstawie zbioru walidacyjnego, po zoptymalizowaniu metaparametrów wykonaj test na zbiorze **TE_DATA**. Zadbaj o statystykę w każdym z przypadków. Czy skuteczność na zbiorze testowym jest porównywalna ze skutecznością na zbiorze walidacyjnym?

Zadanie 3.8: Zdefiniuj problem o nierównomiernym rozkładzie danych pomiędzy klasy. Niech *DataDivision* = 0.1. Czy wpływa to znacząco na otrzymywane wyniki sumaryczne? Czy wpływa to znacząco na procentową skuteczność klasyfikacji w każdej klasie z osobna? Co można zrobić w sytuacji, gdy danych z jednej z klas mamy wprawdzie mało, ale ich skuteczność klasyfikacji jest dla nas równie ważna? Rozwiąż ten problem na dwa sposoby: (1) Poprzez sztuczną redukcję ilości danych w klasie liczniejszej tak, aby dane w klasach uczących (wyłącznie w zbiorze TR_DATA) były równoliczne, oraz (2) poprzez uwzględnienie w funkcji celu obu kryteriów z różnymi wagami - waga dla skuteczności w mniej licznej klasie powinna być większa. Który sposób pozwolił na poprawę wyniku w klasie mniej licznej? Który sposób pozwolił na zachowanie wyższej skuteczności sumarycznej?

Zadanie 3.9: Zdefiniuj klasyfikator nie za pomocą linii prostych, lecz za pomocą okręgów (Zmienna *ClassLikelihood* jest odpowiednio inkrementowana lub dekrementowana w zależności od tego, czy punkt danych leży wewnątrz czy na zewnątrz okręgu. Okręgi powinny być dane za pomocą współrzędnych ich środków oraz promieni, przy czym promień ujemny oznacza, że okrąg "klasyfikuje" odwrotnie, tzn. klasa B wewnątrz, klasa A na zewnątrz okręgu. Wykonaj trening klasyfikatora z zastosowaniem AG - czy problem jest łatwiejszy czy trudniejszy w porównaniu do klasyfikatorów bazujących na prostych? Dlaczego?

Zadanie 3.10: Jak dużo stopni swobody jest potrzeba naszemu klasyfikatorowi, aby uzyskał maksymalną skuteczność? Sprawdź to, używając dwóch podejść: (1) - wytrenuj klasyfikatory posiadające różne ilości stopni swobody zaczynając od 5 linii oraz inkrementując ten parametr o 3 dopóki efektywność nie zacznie spadać. Jaka jest przyczyna tego spadku? (2) - uwzględnij ilość linii w genomie AG (algorytm może zmutować geny osobnika tak, że dodaje lub usuwa z nich linię). Czy rezultaty w obu podejściach są konsekwentne? Czy jesteśmy w stanie korzystając z podejścia (2) uzyskać klasyfikator optymalny, znaleziony w podejściu (1)?