Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Przetwarzanie sygnałów i identyfikacja w sterowaniu urządzeń mechatronicznych

Temat: Zastosowanie naturalnych metod obliczeniowych w klasyfikacji danych wizyjnych

Cel ćwiczenia: Selekcja cech i klasyfikacja obiektów w przygotowanej przestrzeni cech

Zagadnienia: Wizualizacja przestrzeni cech, selekcja cech na potrzeby klasyfikacji, Klasyfikacja liter, Klasyfikator oparty na drzewie decyzyjnym, algorytmie kNN, sieci neuronowej. Optymalizacja metaparametrów klasyfikatorów, testowanie klasyfikatorów, przeuczenie, transfer wiedzy

Prowadzący: dr inż. Ziemowit Dworakowski zdw@agh.edu.pl

Wprowadzenie

Pierwszym z ćwiczonych na laboratoriach zagadnień będzie klasyfikacja danych wizyjnych. Na potrzeby ćwiczenia wykorzystane zostaną obrazy liter. Zbiór danych uwzględnia następujące litery pogrupowane w trzy zestawy:

Zestaw 1: Litery F H K L M N T oraz S
Zestaw 2: Litery A oraz D
Zestaw 3: Litery E G oraz R (nie udostępniane w wersji podstawowej zestawu)

Z liter zestawu 1 zbudowane zostaną tzw. **własne** zadania klasyfikacji. Aby ustalić zadanie własne należy podzielić liczby liter imienia i nazwiska przez 8, reszty z tych dzieleń określą numery klas liter, które należy nauczyć się klasyfikować, przy czym 1 = F, 2 = H itd, aż do 0 = S.

Przykładowo *Jan Kowalski* otrzyma do klasyfikacji litery **K** (reszta z dzielenia 3 ("Jan") przez 8 to 3, 3 litera z 1 zestawu to K) oraz **S** (reszta z dzielenia 8 ("Kowalski") to 0 co oznacza ostatnią literę z zestawu. W przypadku takiej samej ilości liter imienia i nazwiska druga klasa otrzymywana jest przez dodanie 1 do jednej z reszt, np. *Maria Nowak* otrzyma do klasyfikacji litery **M** (reszta 5) oraz **N** (reszta 5 + 1).

Wszystkie dane znajdują się w strukturach opisanych określonymi nazwami, tzn. litery "A" wraz z wstępnie wyznaczonymi cechami znajdują się w strukturze o nazwie "FeaturesA". Zbiór danych zawiera litery zapisane różnymi czcionkami, kursywą, boldem oraz kursywą i boldem. Dodatkowo występują trzy podzbiory danych, zbiór *Normal* zawierający dane podstawowe i zbiory *W1* oraz W2 zawierające zniekształcone wersje liter.

Cechy wstępnie wyznaczone dla wszystkich obiektów wszystkich klas to:

Podstawowe cechy geometryczne z funkcji regionprops:

Area, MajorAxisLength, MinorAxisLength, Eccentricity, Orientation, ConvexArea, Circularity, EulerNumber, EquivDiameter, Solidity, Extent oraz Perimeter

Obraz obiektu (może być wykorzystany do wyznaczania własnych cech lub wzorców):

Image

Cechy bazujące na momentach:

Moments: Struktura z momentami centralnymi (M...) i znormalizowanymi (N...)
HuInvariants: Struktura z 6 pierwszymi niezmiennikami momentowymi Hu (I...)

Wizualizacja cech w przestrzeniach 2D

Na potrzeby wyjaśnienia zadania do wykonania wyobraźmy sobie, że stoimy przed zadaniem klasyfikacji liter "A" oraz "D". Pierwszym krokiem oczywiście będzie zapoznanie się ze zbiorem.

Z wykorzystaniem następującego kodu wczytamy nasze dane:

```
A = load('StudentData/FeaturesA');
D = load('StudentData/FeaturesD');
```

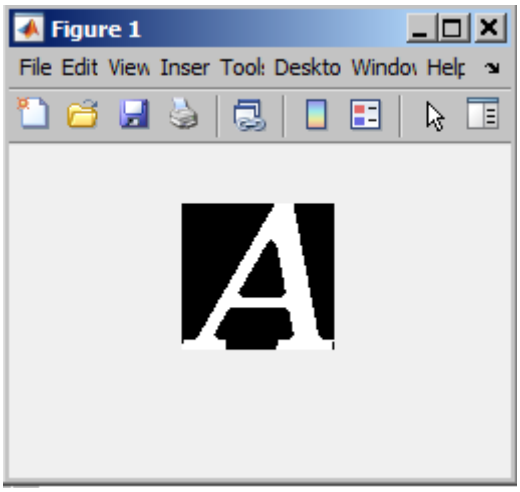
W przestrzeni roboczej pojawiły się dwie struktury. Aby dostać się do nich by np. podejrzeć konkretny obiekt należy posłużyć się np. takim kodem:

```
ObjectNumber = 1;
imshow(A.Data(ObjectNumber).Normal.Image)
```

Używając go wyświetliliśmy pierwszy obiekt klasy "A". Odpowiadające mu dane zniekształcone znajdują się odpowiednio w poniższych miejscach, ale nie będziemy z nich na razie korzystać:

```
imshow(A.Data(ObjectNumber).W1.Image)
imshow(A.Data(ObjectNumber).W2.Image)
```

Wyświetlony obraz z podzbioru "Normalnego" powinien wyglądać jak na tym rysunku:



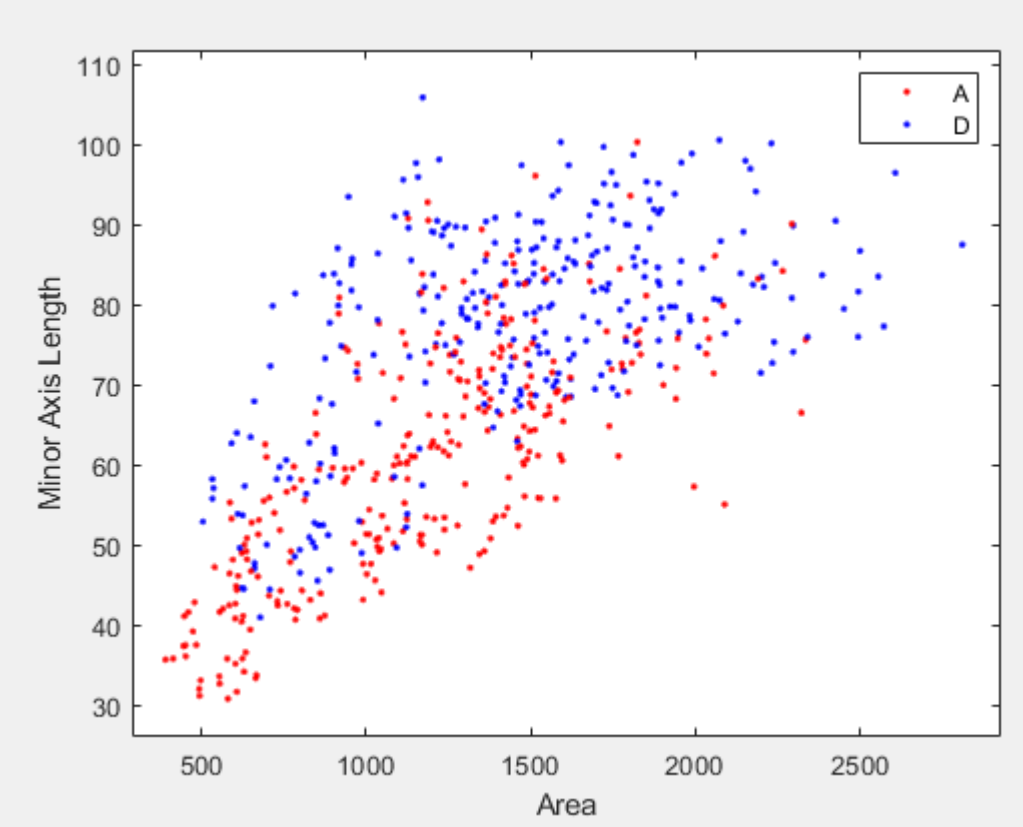
Zadanie 1: Zobacz jak wyglądają przykładowe obiekty w Twoim **własnym** zadaniu klasyfikacyjnym. Poprzez podstawianie kilkunastu losowo wybranych wartości w miejsce zmiennej *ObjectNumber* zidentyfikuj przykłady obiektów w.g. Ciebie łatwych oraz trudnych w klasyfikacji. Przygotuj się do uzasadnienia swojego wyboru oraz zaprezentowania obiektów prowadzącemu (np. w formie zestawu screenów lub programu który pozwala wyświetlić kilka obiektów w jednym oknie)

Ponieważ zbiór zawiera aż 600 obiektów każdej klasy, nie jest możliwe ręczne "przejrzenie" wszystkich. Z konieczności musimy więc przejść do przestrzeni cech i od samego początku opierać się na tym, co w takiej przestrzeni cech można zobaczyć. Za pomocą poniższego kodu jesteśmy w stanie wyświetlić pole powierzchni oraz orientację pierwszych 300 obiektów:

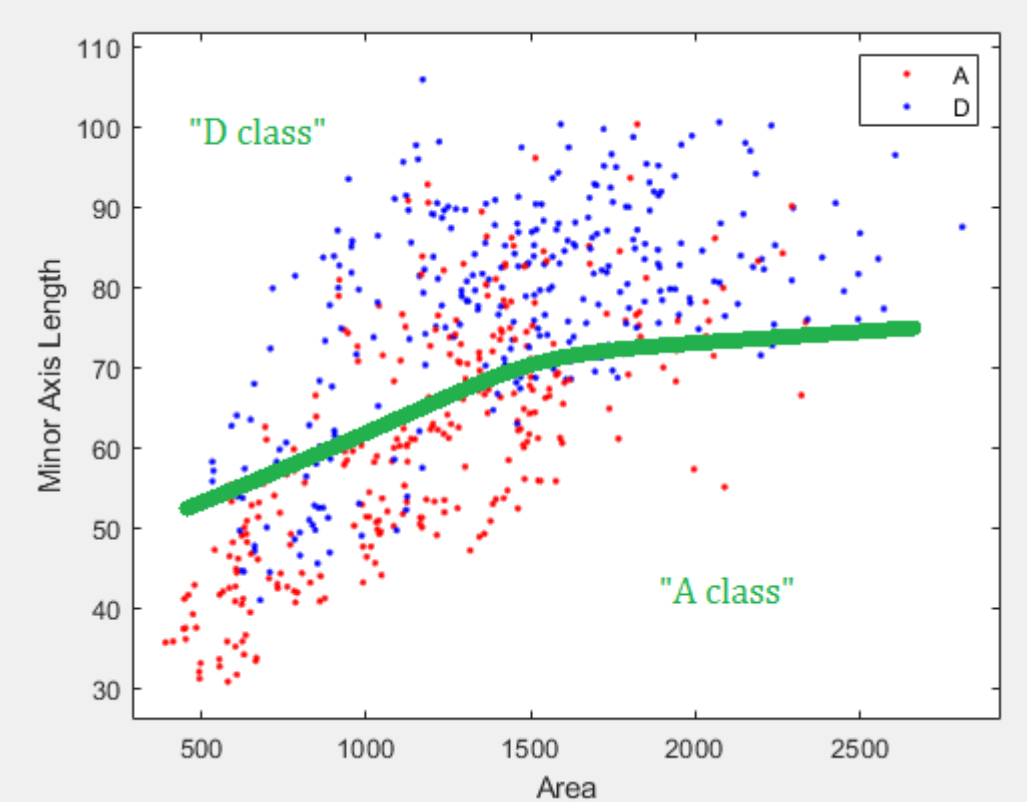
```
figure;
for k = 1:300
   plot(A.Data(k).Normal.Area,A.Data(k).Normal.Orientation,('.r')); hold on
   plot(D.Data(k).Normal.Area,D.Data(k).Normal.Orientation,('.b')); hold on
end
xlabel('Area');
ylabel('Orientation');
legend('A','D');
```

Uwaga! Wyświetlamy tutaj wyłącznie 300 pierwszych obiektów a nie wszystkie dostępne. Robimy to celowo - dlaczego, wytłumaczymy za chwilę. Na razie proszę o trzymanie się tej procedury i **NIE** zmienianie wyświetlania na "600" obiektów!

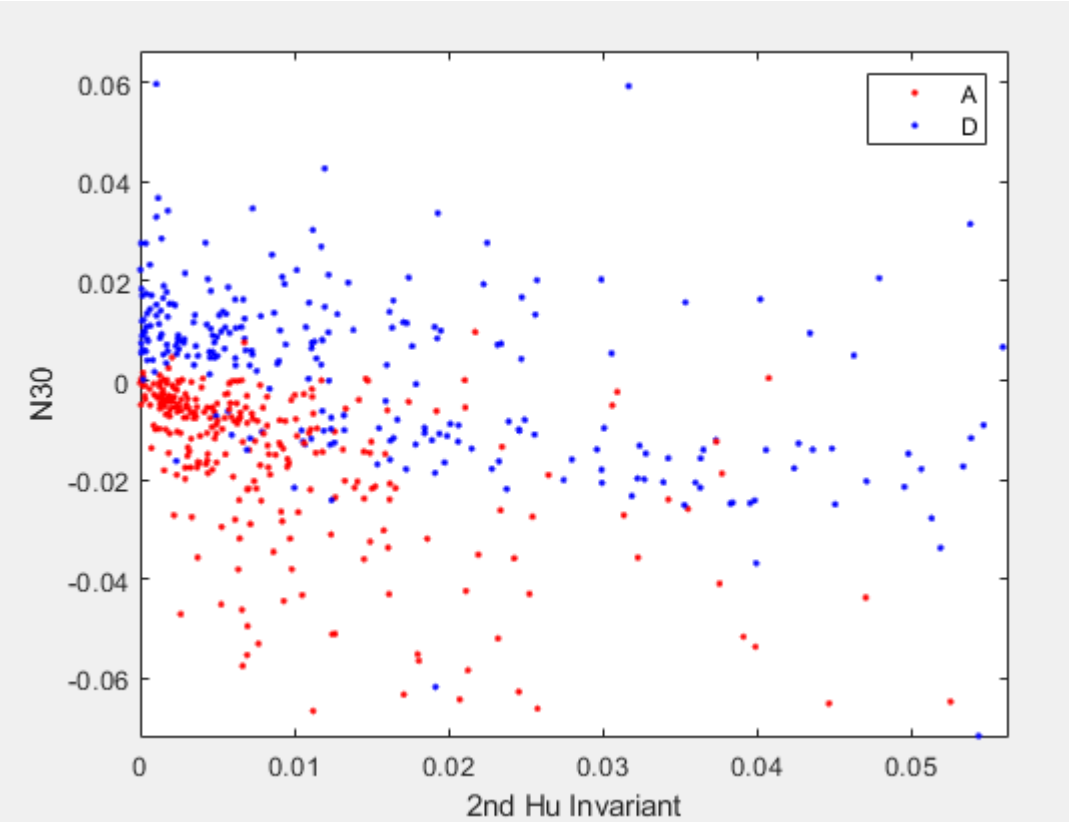
Otrzymany rezultat wygląda następująco:



Widzimy, że obiekty klasy "A" oraz "D" nie zajmują dokładnie tej samej przestrzeni, klaster zawierający litery "D" wydaje się być nieco przesunięty ku prawej i szerszy. Być może moglibyśmy sklasyfikować obiekty w tej przestrzeni z zastosowaniem następującej reguły a skuteczność klasyfikacji byłaby lepsza od losowej:



Ale celem tego etapu musi być znalezienie takich cech, które pozwolą uzyskać jak najlepszą separację klas. Po kilku próbach oraz przetestowaniu cech z grupy momentów i niezmienników momentowych udało się znaleźć następujący przekrój przez przestrzeń cech:



Tutaj wyniki są już znacznie bliższe oczekiwanych, tylko w kilku obszarach cechy nakładają się na siebie.

Zadanie 2: Oglądając różne przestrzenie cech dla 300 pierwszych obiektów każdej klasy Twojego **własnego** zadania klasyfikacji zidentyfikuj przestrzeń w której klasy są jak najwyraźniej rozseparowane (ideałem jest uzyskanie liniowej separacji z szerokim marginesem bezpieczeństwa). Zapisuj cechy, które przetestowałeś. oraz zapisz 2 - 3 zrzuty ekranu przestrzeni cech które są według Ciebie najlepsze.

Uwaga! Nie sugeruj się tym co widzisz na sąsiednich ekranach. Każde zadanie klasyfikacji jest inne, nie w każdym uda się uzyskać dobrą i wyraźną separację klastrów.

Klasyfikacja z wykorzystaniem drzewa decyzyjnego:

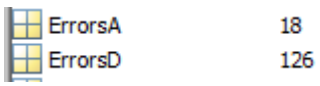
Na podstawie znalezionej przestrzeni cech zaproponujemy teraz reguły klasyfikacyjne pozwalające na jak najlepsze odseparowanie klas. Tutaj możemy to zrobić w wersji podstawowej korzystając z faktu, że centralny znormalizowany moment N30 wydaje się przyjmować wartości dodatnie dla klasy "D" oraz ujemne dla klasy "A". Przygotujmy prosty klasyfikator i zapiszmy go jako osobną funkcję (zapisujemy jako nowy skrypt):

```
function[Class] = ClassifierDT(Object)
    if(Object.Moments.N30 > 0)
        Class = 1;
    else
        Class = 0;
    end
end
```

Teraz w naszym podstawowym kodzie możemy go "nakarmić" danymi i zobaczyć jak dużo błędów popełni:

```
ErrorsA = 0;
ErrorsD = 0;
for k = 1:300
    if(ClassifierDT(A.Data(k).Normal) == 1)  % Detected class "1", (our "D")
        ErrorsA = ErrorsA + 1;
    end
    if(ClassifierDT(D.Data(k).Normal) == 0)  % Detected class "0", (our "A")
        ErrorsD = ErrorsD + 1;
    end
end
```

W naszym przypadku rezultat wygląda następująco



Czyli klasa A jest rozpoznawana zupełnie dobrze, w klasie D występuje ok. 42% błędów. Po zsumowaniu mamy więc skuteczność na poziomie ok. 76% (144 błędy na 600 próbek).

Do tej pory pracowaliśmy wyłącznie na połowie dostępnych danych. Wynika to z faktu, że rzeczywisty test powinien być zawsze wykonywany na danych nie wykorzystywanych do konfiguracji metaparametrów metod. W naszym przypadku zbiorem testowym którego do tej pory nie widzieliśmy jest druga połowa naszych danych. Wykonajmy więc test na tym zbiorze:

```
ErrorsA = 0;
ErrorsD = 0;
for k = 301:600
    if(ClassifierDT(A.Data(k).Normal) == 1)   % Detected class "1", (our "D")
        ErrorsA = ErrorsA + 1;
    end
    if(ClassifierDT(D.Data(k).Normal) == 0)   % Detected class "0", (our "A")
        ErrorsD = ErrorsD + 1;
    end
end
```

W przypadku testowym uzyskano następujący wynik:

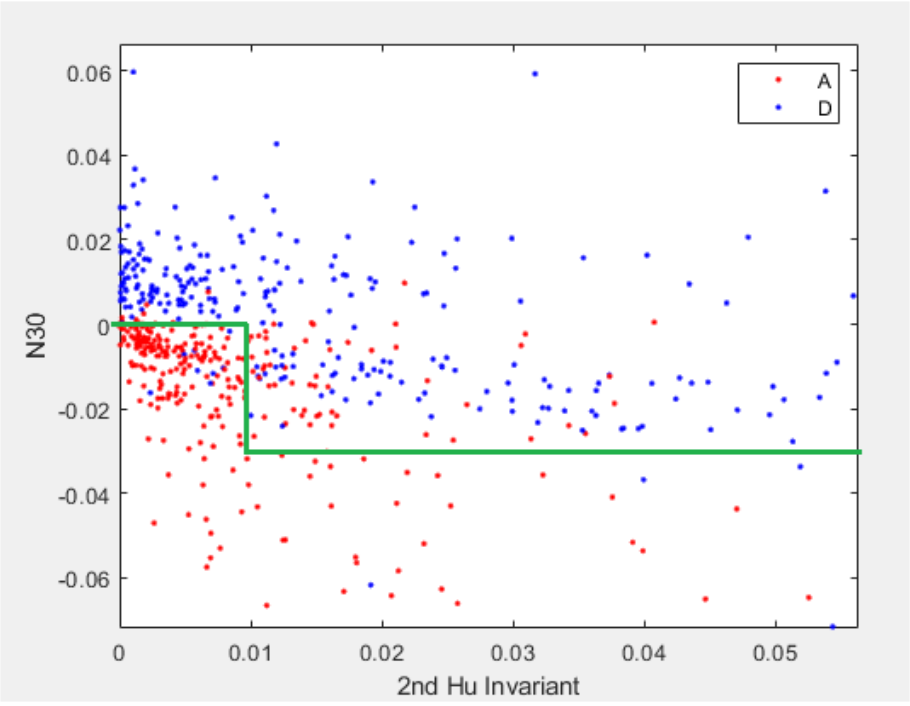
ErrorsA	14
ErrorsD	159

Widać więc, że rozwiązanie jest dość ogólne (poziom błędu jest podobnego rzędu jak w przypadku etapu konfiguracji)

Oczywiście powyższy kod jest jedynie demonstracją sposobu myślenia. W klasyfikatorze można wykorzystywać więcej niż jedną cechę oraz ich złożenia. Przykładowo, klasyfikator zbudowany następująco:

```
function[Class] = ClassifierDT2(Object)
    if(Object.Moments.N30 > 0)
        Class = 1;
    elseif(Object.Moments.N30 < -0.03)
        Class = 0;
    elseif(Object.HuInvariants.I1 < 0.01)
        Class = 0;
    else
        Class = 1;
    end
end
```

Który klasyfikuje przestrzeń w następujący sposób:

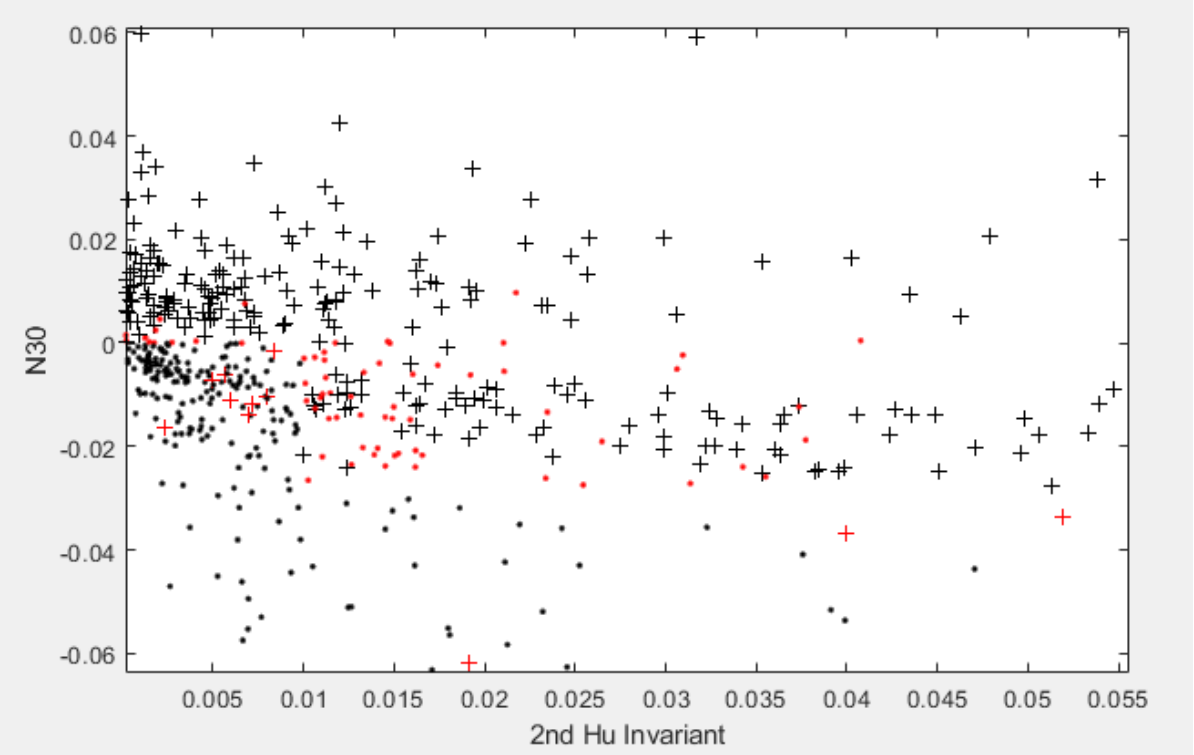


Osiąga w sumie tylko 93 błędy na zbiorze uczącym i 115 błędów na zbiorze testowym.

Aby zwizualizować działanie klasyfikatora i pokazać miejsca, w których występują błędy można skorzystać z następującego wzbogacenia kodu:

```
ErrorsA = 0;
ErrorsD = 0;
figure;
for k = 1:300
    if(ClassifierDT2(A.Data(k).Normal) == 1)  % Detected class "1", (our "D")
        ErrorsA = ErrorsA + 1;
        plot(A.Data(k).Normal.HuInvariants.I1,A.Data(k).Normal.Moments.N30,'.r'); hold on
    else
        plot(A.Data(k).Normal.HuInvariants.I1,A.Data(k).Normal.Moments.N30,'.k'); hold on
    end
    if(ClassifierDT2(D.Data(k).Normal) == 0)  % Detected class "0", (our "A")
        ErrorsD = ErrorsD + 1;
        plot(D.Data(k).Normal.HuInvariants.I1,D.Data(k).Normal.Moments.N30,'+r'); hold on
    else
        plot(D.Data(k).Normal.HuInvariants.I1,D.Data(k).Normal.Moments.N30,'+k'); hold on
    end
end
xlabel('2nd Hu Invariant');
ylabel('N30');
```

Który pozwala zobaczyć w której klasie w których miejscach występują błędy:



Można teraz starać się tak przesuwać reguły klasyfikacyjne, aby ilość błędów na zbiorze uczącym maksymalnie zmalała. Po krótkiej zabawie regułami dla niniejszego zadania ilość błędów zmalała do 77 na zbiorze uczącym i 99 na testowym, dla poniższego klasyfikatora:

```
function[Class] = ClassifierDT(Object)
    if(Object.Moments.N30 > 0);                 Class = 1;
    elseif(Object.Moments.N30 < -0.035);        Class = 0;
    elseif(Object.HuInvariants.I1 < 0.016);     Class = 0;
    else                                        Class = 1;
    end
end
```

Zadanie 3: Na podstawie wybranego zestawu cech zaproponuj klasyfikator oparty na drzewie decyzyjnym. Skonfiguruj parametry bazując na pierwszych 300 obiektach każdej klasy a następnie przetestuj go używając drugiej połowy zbioru. Zapisz otrzymaną skuteczność uczącą i testową otrzymaną w pierwszej próbie uruchomienia klasyfikatora oraz po wykonaniu kilku kroków "poprawek" parametrów klasyfikatora.

Klasyfikator k-najbliższych sąsiadów

Aby sklasyfikować obiekty można oprzeć się na tym jak blisko w przestrzeni cech znajdują się inne obiekty tej samej klasy. Kolejnym zastosowanym klasyfikatorem będzie klasyfikator k-najbliższych sąsiadów. Aby go przygotować najpierw dostarczyć musimy zbiór z którego kNN będzie wybierał sąsiadów. Należy zwrócić uwagę, że ponownie korzystamy wyłącznie z pierwszej połowy pełnego zbioru:

```
for sample = 1:300
   TrainingDataClass0(:,sample) = ...
[A.Data(sample).Normal.Moments.N30,A.Data(sample).Normal.HuInvariants.I1];
   TrainingDataClass1(:,sample) = ...
[D.Data(sample).Normal.Moments.N30,D.Data(sample).Normal.HuInvariants.I1];
end
```

Następnie, mając tak przygotowany zbiór budujemy klasyfikator bazujący na pojedynczym sąsiedzie (zapisując go jako osobną funkcję!):

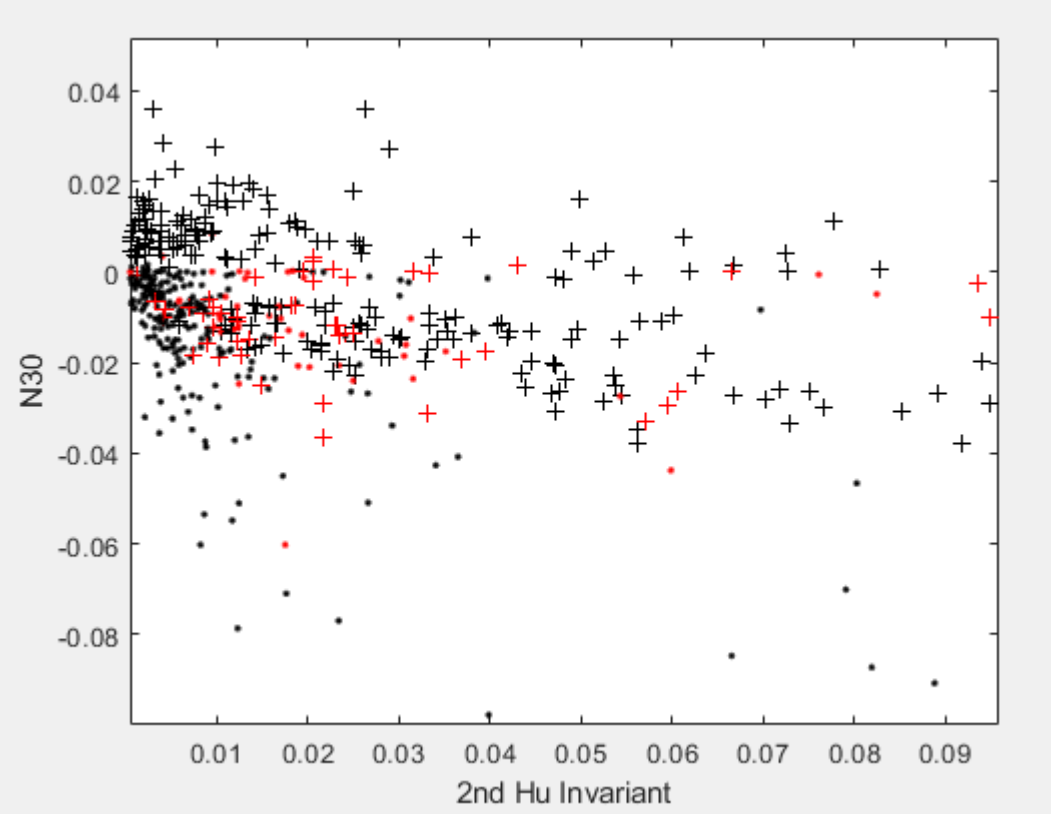
```
function[Class] = ClassifierKNN(Features, TrainingDataClass0,TrainingDataClass1)
    distancesC1 = dist(Features,TrainingDataClass0);
    distancesC2 = dist(Features,TrainingDataClass1);
    if(min(distancesC1) < min(distancesC2))
        Class = 0;
    else
        Class = 1;
    end
end
```

I finalnie możemy go przetestować w naszym zadaniu:

```
figure;
for k = 301:600
    F1 = A.Data(k).Normal.Moments.N30;
    F2 = A.Data(k).Normal.HuInvariants.I1;

    if(ClassifierKNN([F1,F2],TrainingDataClass0,TrainingDataClass1) == 1)  % Detected class "1"
       ErrorsA = ErrorsA + 1;
       plot(A.Data(k).Normal.HuInvariants.I1,A.Data(k).Normal.Moments.N30,'.r'); hold on
    else
       plot(A.Data(k).Normal.HuInvariants.I1,A.Data(k).Normal.Moments.N30,'.k'); hold on
    end

    F1 = D.Data(k).Normal.Moments.N30;
    F2 = D.Data(k).Normal.HuInvariants.I1;

    if(ClassifierKNN([F1,F2],TrainingDataClass0,TrainingDataClass1) == 0)  % Detected class "0
ErrorsD = ErrorsD + 1;
       plot(D.Data(k).Normal.HuInvariants.I1,D.Data(k).Normal.Moments.N30,'+r'); hold on
    else
       plot(D.Data(k).Normal.HuInvariants.I1,D.Data(k).Normal.Moments.N30,'+k'); hold on
    end
end
xlabel('2nd Hu Invariant');
ylabel('N30');
```

Zwróćmy uwagę, że tak zbudowany klasyfikator zawsze będzie miał na naszym zbiorze uczącym skuteczność 100%. W tym wypadku na zbiorze testowym uzyskaliśmy w tym przypadku skuteczność ok. 65%

(104 błędy):



Zadanie 4: Na podstawie wybranego zestawu cech przetestuj klasyfikator kNN z k = 1. Dołóż do klasyfikatora więcej cech: niech dystans znajdowany będzie w przestrzeni 4 wymiarowej korzystając z dwóch alternatywnych cech wybranych w zadaniu 1. Czy udało się podnieść skuteczność? Zapisz program w obu wersjach aby była możliwość szybkiego zaprezentowania wyników.

Uwaga! Nasz klasyfikator już przystosowany jest do pracy w przestrzeni cech o dowolnej wymiarowości. Należy jedynie dopisać dodatkowe cechy do kolejnych kolumn macierzy TrainingDataClass0 oraz TrainingDataClass1 i dodać odpowiednie cechy (np. F3, F4) w wywoływaniu klasyfikatora

W przypadku dostarczania danych do klasyfikatorów bazujących na dystansach w przestrzeni cech niezbędna jest zazwyczaj ich **normalizacja**. W naszym przypadku większość danych zarówno na osi X jak i na osi Y ląduje w zakresie o szerokości ok. 0.04 jednostek ale możliwe są przypadki, w których te wartości będą zupełnie różne - np. pierwsze dwie przetestowane cechy tj. MinorAxisLength oraz Area różnią się swoim zakresem o dwa rzędy wielkości. W takim przypadku cecha bardziej "rozrzucona" będzie się w znacznie większym stopniu przyczyniała do otrzymywanego wyniku zamieniając efektywnie przestrzeń w jednowymiarową. Najprostszym sposobem likwidującym problem jest zastosowanie podstawowej normalizacji polegającej na podzieleniu otrzymywanych wartości przez maksimum lub przez wartość średnią (mniej wrażliwe na outliery). Oczywiście stosując tą procedurę do zbioru uczącego konieczne będzie konsekwentne dzielenie przez te same wartości w zbiorze testowym (tzn. np. średnie wyliczone dla zbioru uczącego a nie testowego!)

Procedura normalizacji naszych danych możliwa jest do wykonania w następującym kroku:

```
% Finding of the means for the whole dataset
Means = abs(mean([TrainingDataClass0,TrainingDataClass1]'));
% Dividing the data:
TrainingDataClass0 = TrainingDataClass0./Means'
TrainingDataClass1 = TrainingDataClass1./Means'
```

A następnie wydobywanie cech do klasyfikacji może wyglądać następująco:

```
F1 = A.Data(k).Normal.Moments.N30/Means(1);
F2 = A.Data(k).Normal.HuInvariants.I1/Means(2);
```

Zadanie 5: Znormalizuj swoje dane. Czy skuteczność klasyfikacji klasyfikatora kNN wzrosła? Zapisz program aby móc zaprezentować wyniki.

Jak sama nazwa wskazuje, klasyfikator "k" najbliższych sąsiadów może mieć parametr "k" o wartości większej niż 1. Aby zaimplementować taki klasyfikator zmodyfikujemy jego kod w następujący sposób:

```
function[Class] = ClassifierKNN(Features, TrainingDataClass0,TrainingDataClass1)

    K = 3;    % Metaparameter: How many neighbors do we want to take?

    distancesC1 = dist(Features,TrainingDataClass0);
    distancesC2 = dist(Features,TrainingDataClass1);
    % We store all the calculated distances into one matrix
    TestingMatrix(:,1) = [distancesC1,distancesC2];
    % We add information from which class do rows come from
    TestingMatrix(:,2) = [zeros(1,length(distancesC1)),ones(1,length(distancesC2))];
    % We sort the matrix so the closest distances are in the top
    SortedMatrix = sortrows(TestingMatrix,1)
    % We take K top rows and sum class indications
    CompoundNeighborClasses = sum(SortedMatrix(1:K,2));
    % if the sum is higher than half of the K - we have class 1
    if(CompoundNeighborClasses < K/2)
        Class = 0;
    else
        Class = 1;
    end
end
```

Zadanie 6: Przetestuj klasyfikator kNN dla k = 1, k = 3, k = 5 oraz k = 9 w przestrzeni dwu, cztero i sześciowymiarowej. Zapisz otrzymane wyniki (ilość błędów) w formie tabeli. Jak dobrze klasyfikator kNN daje się skalować do wielowymiarowych przestrzeni? Czy zwiększanie wartości k ma pozytywny czy negatywny wpływ na skuteczność klasyfikacji?

	k = 1	k = 3	k = 5	k = 9
2 cechy				
4 cechy				
6 cech				

Wielowarstwowy perceptron

Kolejnym narzędziem do klasyfikacji danych wizyjnych które wykorzystamy w tym ćwiczeniu będzie prosta sieć neuronowa. Warto zauważyć, że struktura kodu którą przygotowaliśmy testując klasyfikator kNN nadaje się w zasadzie bez zmian do zastosowania sieci neuronowej. Będziemy potrzebowali zbioru uczącego, który zbudujemy z już przygotowanych macierzy *TrainingDataClass0* oraz *TrainingDataClass1:*

```
TrainingData = [TrainingDataClass0,TrainingDataClass1];
TargetTrain = [ones(1,300),zeros(1,300)];
```

poprawny rozmiar macierzy TrainingData to 2 x 600 (jeśli używamy 2 cech) lub 4 x 600 (jeśli używamy 4). Poprawny rozmiar wektora TargetTrain to 1x600.

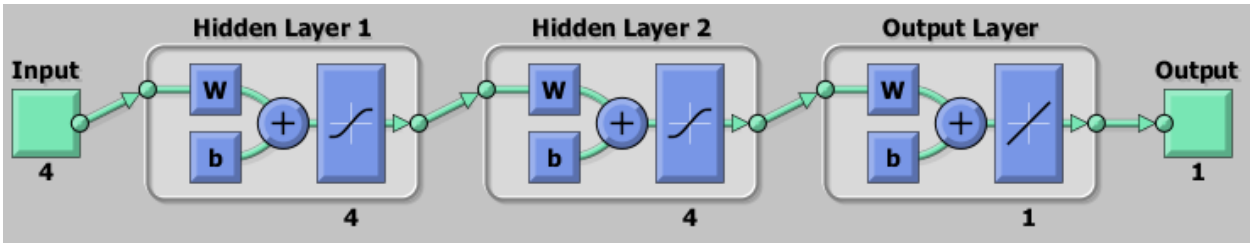
Następnie musimy dobrać wartości metaparametrów sieci neuronowej i skonfigurować samą sieć:

```
NetworkSize = [4 4];        % Number of neurons in all the hidden layers
OurNet = newff(TrainingData,TargetTrain,NetworkSize); % Net initialization
OurNet.TrainParam.time = 10;     % Max time allowed [in sec]
```

Można zobaczyć jak zbudowana sieć wygląda używając komendy

```
view(OurNet)
```

Powinniśmy zobaczyć coś takiego:



Inna, podejrzanie duża ilość wejść prawdopodobnie oznacza, że dane uczące mają postać macierzy "pionowej" (wiersze zawierają cechy dla kolejnych liter) co jest błędem.

I finalnie można naszą sieć wytrenować komendą:

```
OurNet = train(OurNet,TrainingData,TargetTrain);
```

Na tym etapie można powrócić do kodu uruchamiającego kNN i wprowadzić w nim następujące zmiany (zaznaczone na żółto):

```
for k = 301:600
    F1 = A.Data(k).Normal.Moments.N30/Means(1);
    F2 = A.Data(k).Normal.HuInvariants.I1/Means(2);

    if(sim(OurNet,[F1,F2]') > 0.5)  % Detected class "1", (our "D")
        ErrorsA = ErrorsA + 1;
        plot(A.Data(k).Normal.HuInvariants.I1,A.Data(k).Normal.Moments.N30,'.r'); hold on
    else
        plot(A.Data(k).Normal.HuInvariants.I1,A.Data(k).Normal.Moments.N30,'.k'); hold on
    end

    F1 = D.Data(k).Normal.Moments.N30/Means(1);
    F2 = D.Data(k).Normal.HuInvariants.I1/Means(2);

    if(sim(OurNet,[F1,F2]') <= 0.5)  % Detected class "0", (our "A")
        ErrorsD = ErrorsD + 1;
```

```
        plot(D.Data(k).Normal.HuInvariants.I1,D.Data(k).Normal.Moments.N30,'+r'); hold on
    else
        plot(D.Data(k).Normal.HuInvariants.I1,D.Data(k).Normal.Moments.N30,'+k'); hold on
    end
end
xlabel('2nd Hu Invariant');
ylabel('N30');
```

Zadanie 7: Na podstawie wybranego zestawu cech (wybrane na początku 2 najlepsze cechy) przetestuj wielowarstwowy perceptron. Uruchom swój program kilka razy. Czy za każdym razem otrzymujesz ten sam wynik? Zapisz program pod osobną nazwą aby być gotowym do zaprezentowania jego działania prowadzącemu.

Wiemy już, że wielowarstwowy perceptron musi być trenowany i testowany wielokrotnie aby możliwa była rzetelna ocena jego potencjału. Zmodyfikujemy więc poprzedni kod, aby pętla odpowiadająca za inicjalizację, uczenie i testowanie klasyfikatora zamknięta była w wykonującym się dziesięciokrotnie bloku, Usuniemy również wizualizację wyników, nie będzie nam już potrzebna - a skoro nie będziemy potrzebowali wizualizować wyników, wykonamy komplet obliczeń w wersji macierzowej by przyspieszyć ich wykonywanie.

Zaczniemy od przygotowania macierzy testowej tak samo, jak przygotowaliśmy treningową. Uwaga! Należy zwrócić uwagę, że dane testowe znajdują się pod indeksami 301:600, stąd fragment "300+" w kodzie.

```
for sample = 1:300
   TestingDataClass0(:,sample) = ...
[A.Data(300+sample).Normal.Moments.N30,A.Data(300+sample).Normal.HuInvariants.I1];
   TestingDataClass1(:,sample) = ...
[D.Data(300+sample).Normal.Moments.N30,D.Data(300+sample).Normal.HuInvariants.I1];
end

TestingDataClass0 = TestingDataClass0./Means'
TestingDataClass1 = TestingDataClass1./Means'
```

A następnie wykonujemy pełen cykl 10 inicjalizacji i testów sieci:

```
NetworkSize = [4 4];        % Number of neurons in all the hidden layers
for Test = 1:10
    OurNet = newff(TrainingData,TargetTrain,NetworkSize); % Net initialization
    OurNet.TrainParam.time = 20;   % Max time allowed
    OurNet = train(OurNet,TrainingData,TargetTrain);

    Results0 = sim(OurNet,TestingDataClass0);
    Results1 = sim(OurNet,TestingDataClass1);

    ErrorsA(Test) = sum([Results0>0.5]);
    ErrorsD(Test) = sum([Results1<=0.5]);
    ErrorSum(Test) = ErrorsD(Test)+ErrorsA(Test)
end
[mean(ErrorSum) std(ErrorSum)]
```

Zadanie 8: Przetestuj 10-krotnie swoją sieć neuronową dla standardowych parametrów, dla większej sieci (np. *NetworkSize = [10 10]*), dla większej ilości cech (np. 4), dla sieci głębszej (np. *NetworkSize = [4 4 4]*) lub płytszej (np. *NetworkSize = [4]*). W którym z tych przypadków udało się uzyskać najwyższą średnią skuteczność?

Dotarliśmy do optymalizacji metaparametrów sieci. Pytanie brzmi: jaki zestaw wartości metaparametrów pozwala w najlepszym stopniu wykonywać zadania tej klasy? Pamiętając o tym, że pojedynczy test nie daje sensownego wyniku konieczne będzie wielokrotne przetestowanie wielu różnych wartości metaparametrów. Dodatkowo, niezbędne będzie podzielenie danych testowych na dwie kategorie, te do optymalizacji metaparametrów oraz te do finalnego testu.

Podział danych może zostać wykonany losowo:

```
Indices = randperm(300); % Here we randomly permute data indices
for sample = 1:150
   TestingDataClass0(:,sample) =  ...
      [A.Data(300+Indices(sample)).Normal.Moments.N30,...
      A.Data(300+ Indices(sample)).Normal.HuInvariants.I1];
   TestingDataClass1(:,sample) =  ...
      [D.Data(300+Indices(sample)).Normal.Moments.N30,...
      D.Data(300+Indices(sample)).Normal.HuInvariants.I1];
end
for sample = 1:150
   FinalTestClass0(:,sample) = ...
      [A.Data(300+Indices(sample)).Normal.Moments.N30,...
      A.Data(300+Indices(sample)).Normal.HuInvariants.I1];
   FinalTestClass1(:,sample) = ...
      [D.Data(300+Indices(sample)).Normal.Moments.N30,...
      D.Data(300+Indices(sample)).Normal.HuInvariants.I1];
end
```

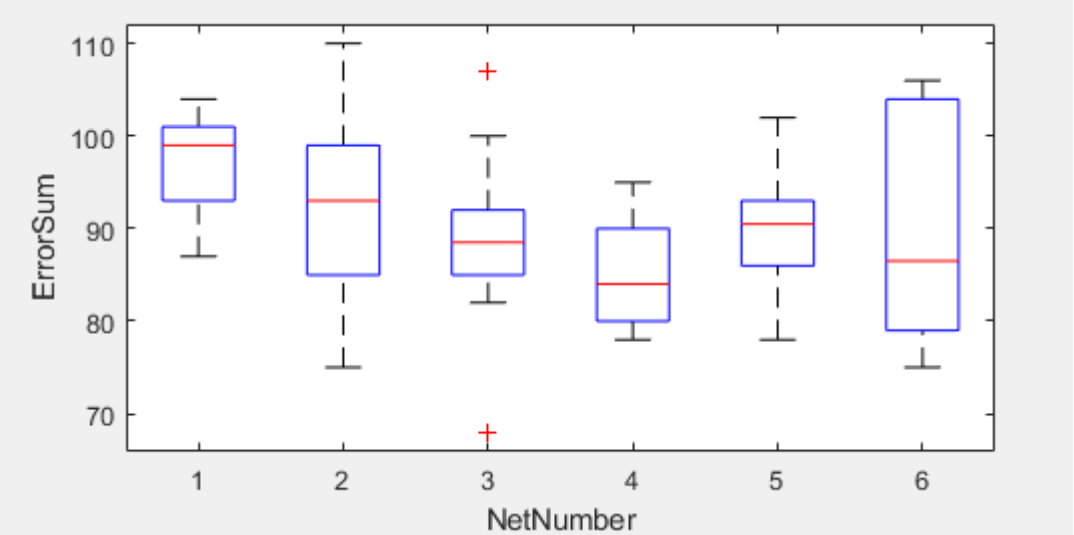
Samą optymalizację metaparametrów najprościej zrobić zamykając kod testujący sieci w nadrzędnej pętli

```
for TestSerie = 1:6
NetworkSize = [2*TestSerie 2*TestSerie];    % Number of neurons in all the hidden layers
    for Test = 1:10
        OurNet = newff(TrainingData,TargetTrain,NetworkSize); % Net initialization
        OurNet.TrainParam.time = 20;    % Max time allowed
        OurNet = train(OurNet,TrainingData,TargetTrain);

        Results0 = sim(OurNet,TestingDataClass0);
        Results1 = sim(OurNet,TestingDataClass1);

        ErrorsA(Test,TestSerie) = sum([Results0>0.5]);
        ErrorsD(Test,TestSerie) = sum([Results1<=0.5]);
        ErrorSum(Test,TestSerie) = ErrorsD(Test,TestSerie)+ErrorsA(Test,TestSerie)
    end
end
```

Następnie, tak przygotowane dane można wyplotować używając tzw. wykresu pudełkowego:

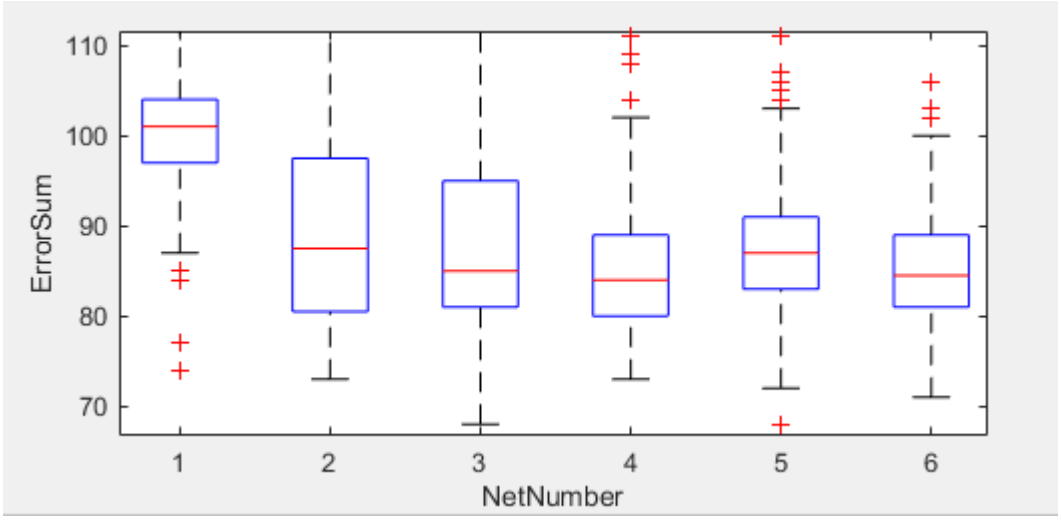
```
figure;   boxplot(ErrorSum);   ylabel('ErrorSum');   xlabel('NetNumber');
```

Który powinien wyglądać np. tak:



Na wykresie przedstawiono mediany (czerwone linie), 1 i 3 kwartyl (niebieskie poziome linie), "wąsy" rozciągają się na całym zakresie danych (tzn. od min do max) z wyłączeniem outlierów. Outliery zaznaczane są osobno w formie czerwonych plusów.

Widać, że w przykładowym zadaniu dla badanych cech minimum ilości błędów i zarazem maksimum powtarzalności sieci znajduje się mniej więcej w okolicach 4 sieci, tzn. sieci o strukturze [8 8]. Oczywiście nawet te wyniki powinny być traktowane z dużą rezerwą, bowiem 10 powtórzeń nie pozwala na wykonanie dokładnej statystyki. Przykładowo, dla 100 powtórzeń testu każdej sieci fragment powyższego wykresu (dla tego samego zakresu na osi Y) wyglądać będzie następująco:



Jak widać, poprzednio zakresy zmienności zostały niedoszacowane, choć konsekwentnie sieć o nr 4 charakteryzuje się najmniejszą medianą oraz dość wysoką powtarzalnością. Na sam koniec należy przetestować najlepszą uzyskaną sieć na finalnym zbiorze testowym. Nałożenie się (mniej więcej) 'pudełek' na siebie oznacza dobrze wykonaną optymalizację. Znaczna różnica pomiędzy 'pudełkami' może wskazywać na "przeoptymalizowanie" metaparametrów (tj. przeuczenie 2 rzędu).

Zadanie 9: Wykonaj optymalizację ilości neuronów w warstwach dla 1, 2 i trójwarstwowej sieci (licząc ilość warstw ukrytych). Dla każdej rozważanej głębokości sieci przetestuj zakres od 2 do 20 neuronów, co 2 neurony. Dla każdej konfiguracji wykonaj 10 powtórzeń. Zaproponuj sieć najlepszą do rozważanego przez siebie przypadku dla dwuwymiarowej przestrzeni cech.

Zadanie 10: Powtórz zadanie 9 odpowiednio dla cztero i sześciowymiarowej przestrzeni cech, zapisz komplet otrzymanych danych, porównaj sieci najlepsze w każdej kategorii ze sobą na jednym wykresie pudełkowym (tzn. pierwsze pudełko: najlepsza sieć jednowarstwowa w przestrzeni 2D. Drugie pudełko, najlepsza sieć dwuwarstwowa w przestrzeni 2D, ... , Ostatnie pudełko, najlepsza sieć trójwarstwowa w przestrzeni 6D.

Zadania rozszerzające:

Zadanie 11: Sprawdź jak dobrze nasza najlepsza nauczona sieć jest w stanie wykonać *transfer wiedzy* - przetestuj ją na danych z kategorii W1 (600 próbek) - jaką skuteczność uzyskała? Czy dołożenie do zbioru uczącego większej ilości danych pozwoli podnieść ten wynik? Co w przypadku, jeśli zbiór uczący będzie zawierał komplet danych normalnych (Normal) oraz komplet danych z kategorii W2 (razem 1200 przykładów w każdej klasie) - czy to podniesie skuteczność sieci?

Zadanie 12: Analogicznie do optymalizacji metaparametrów sieci wykonaj (dla jednej określonej struktury sieci) optymalizację wejść - tzn. przygotuj program, który w automatyczny sposób będzie podstawiał różne zestawy cech do budowy zbiorów uczących i testowych. Czy znaleziony na początku "ręcznie" zbiór cech był bliski optymalnemu?

Zadanie 13: Z zastosowaniem sieci neuronowej wykonaj klasyfikację trzech liter, dwóch uzyskanych w zadaniu **własnym** oraz trzeciej wybranej losowo. Wykonaj to zadanie poprzez przekazanie do sieci etykiety klasy w postaci wartości 0, 1 lub -1 (odpowiednio dla litery nr 1, 2 i 3), w postaci trzech niezależnych wyjść z sieci oraz w postaci trzech sieci niezależnie klasyfikujących litery: Sieć pierwsza odróżnia literę 1 od pozostałych, sieć 2 odróżnia literę 2 od pozostałych, sieć 3 odróżnia literę 3 od pozostałych. W którym z przypadków uzyskano najwyższą skuteczność potwierdzoną statystycznie?

Zadanie 14: Jak dużo danych potrzeba do skutecznego nauczenia sieci? Jak dużo danych potrzeba do skutecznego (ze statystycznego punktu widzenia) oszacowania skuteczności? Załóżmy, że testujemy sieć na 300 przypadkach a uczymy na 300, 200, 100, 50 i 10. Jak bardzo będą się zmieniały wyniki? Załóżmy, że uczymy sieć na 300 punktach danych a testujemy na 300, 200, 100, 50 i 10. Która wartość - błąd wynikający ze zbyt ubogiego zbioru danych uczących czy niepewność testu narasta szybciej?