Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Podstawy Sztucznej Inteligencji I Uczenia Głębokiego

Inżynieria Mechatroniczna

Instrukcja 10:

Płytkie klasyfikatory do rozpoznawania obiektów

Nauczysz się: jak wybrać cechy obrazowe na potrzeby klasyfikacji, jak zbudować swój własny klasyfikator bazujący na drzewie decyzyjnym, jak porównać jego wyniki z modelami kNN oraz ANN. Sprawdzimy też które kroki procedury mają największe znaczenie ze względu na finalną skuteczność systemu.

Dodatkowe materiały:

- Wykłady

Koordynator przedmiotu:
Krzysztof Holak, holak@agh.edu.pl

Autor instrukcji:
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');
```

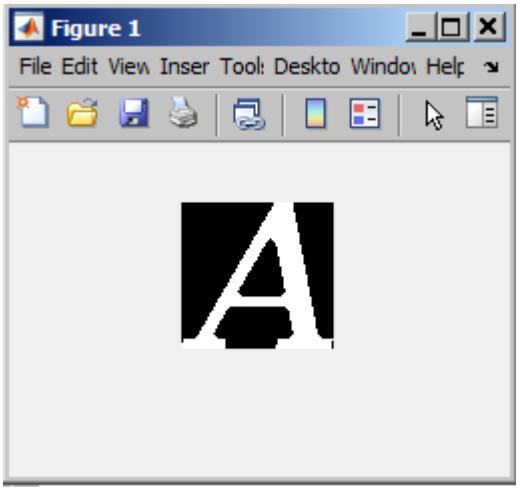
W przestrzeni roboczej pojawiły się dwie struktury. Aby dostać się do nich by np. podejrzeć konkretny obiekt należy posłużyć się np. takim kodem:

```
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" może wyglądać jak na Rys 1.



Rys 1 – przykładowy obiekt z bazy do klasyfikacji

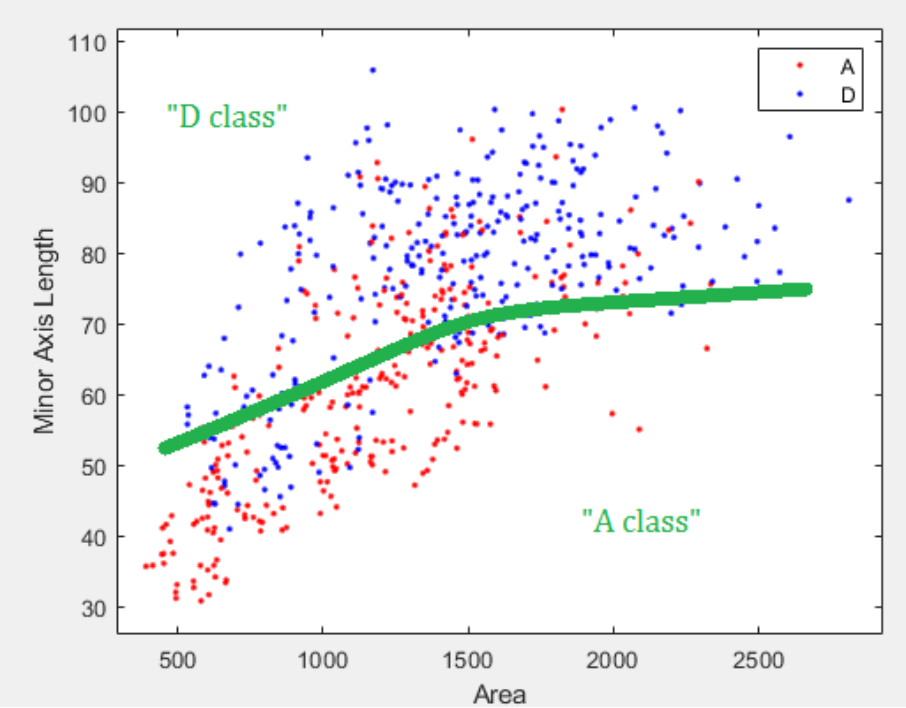
Zadanie 10.1: Zobacz jak wyglądają przykładowe obiekty w Twoim **własnym** zadaniu klasyfikacyjnym. Poprzez podstawianie kilkunastu losowo wybranych wartości w miejsce zmiennej *ObjectNumber* zidentyfikuj przykłady obiektów w.g. Ciebie łatwych oraz trudnych w klasyfikacji. Przygotuj się do uzasadnienia 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');
```

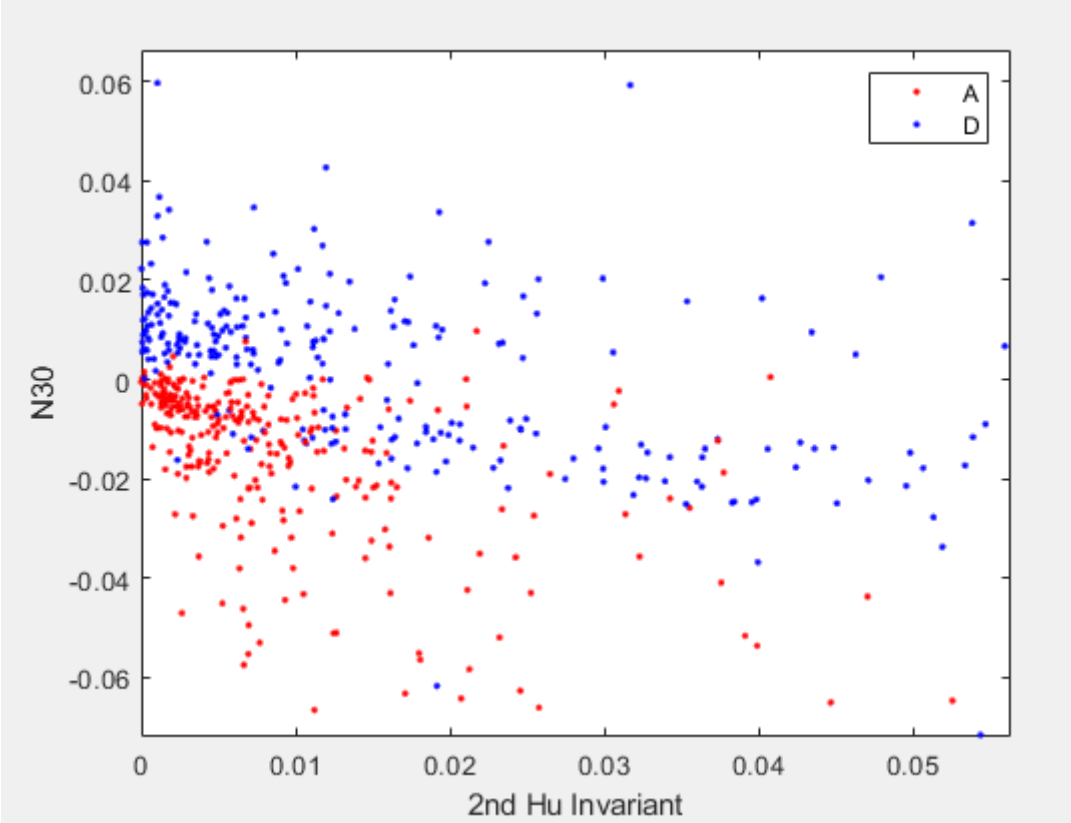
Uwaga! Wyświetlamy tutaj wyłącznie 300 pierwszych obiektów a nie wszystkie dostępne. Robimy to celowo - dlaczego, wytłumaczymy za chwilę. Na razie proszę o trzymanie się tej procedury i **NIE** zmienianie wyświetlania na "600" obiektów!

Otrzymany rezultat przedstawiłem na Rys 2.



Rys 2 – Przestrzeń cech przedstawiająca dwie wstępnie wybrane cechy obiektów

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 reguły zwizualizowanej za pomocą zielonej linii a skuteczność klasyfikacji byłaby lepsza od losowej, ale celem tego etapu powinno być znalezienie takich cech, które pozwolą uzyskać jak najlepszą separację klas. Po kilku próbach oraz przetestowaniu cech z grupy momentów i niezmienników momentowych udało mi się znaleźć przekrój przez przestrzeń cech, który przedstawiłem na Rys 3. Tutaj wyniki są już znacznie bliższe oczekiwanych, tylko w kilku obszarach klasy nakładają się na siebie.



Rys 3 – Dwie cechy wyselekcjonowane do klasyfikacji

Zadanie 10.2: 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

ErrorsA	18
ErrorsD	126

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:

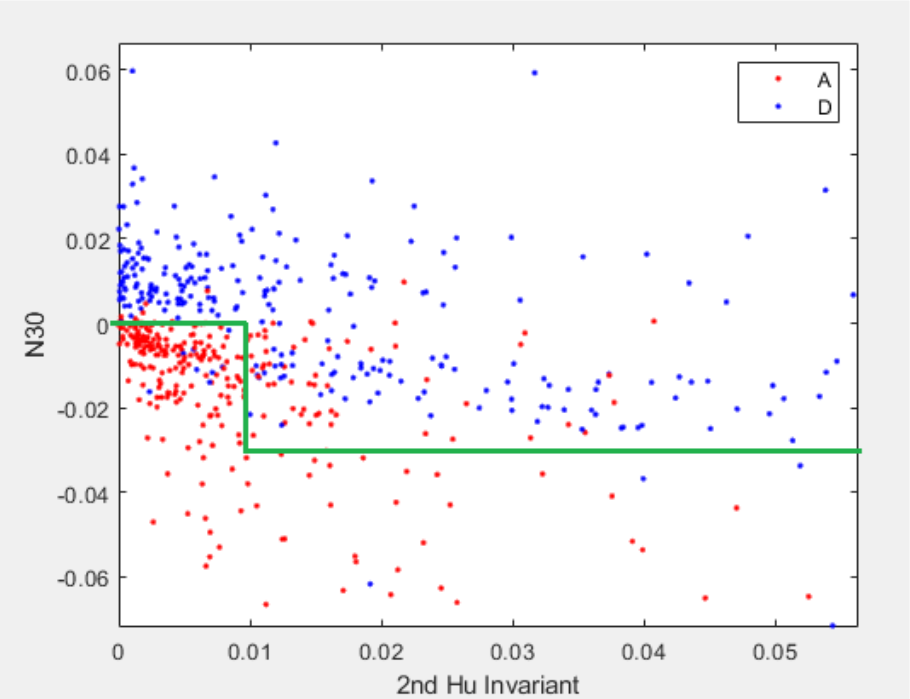
```
ErrorsA = 0;
ErrorsD = 0;
for k = 301:600
    if(ClassifierDT(A.Data(k).Normal) == 1)  % Detected class "1", (our "D")
        ErrorsA = ErrorsA + 1;
    end
    if(ClassifierDT(D.Data(k).Normal) == 0)  % Detected class "0", (our "A")
        ErrorsD = ErrorsD + 1;
    end
end
```

W przypadku testowym uzyskałem 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, który dzieli przestrzeń w sposób pokazany na Rys. 4 zbudowany może być 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
```

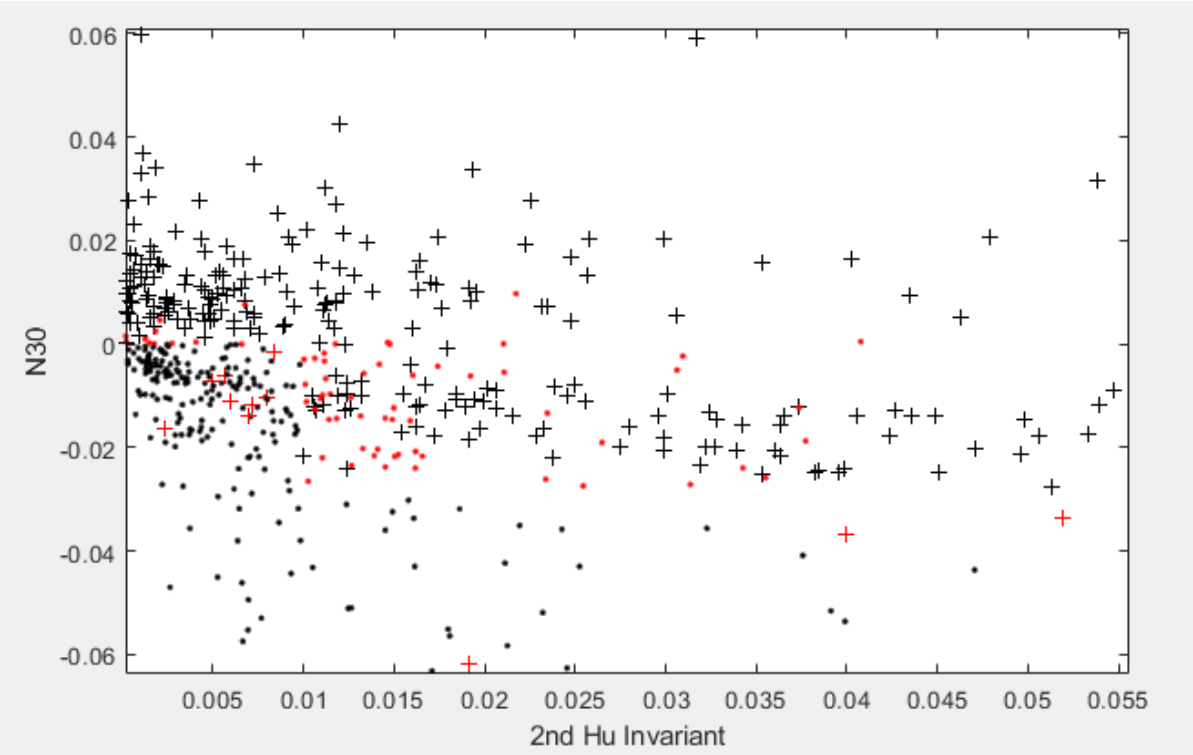


Rys 4 – Nowy, lepszy przepis na klasyfikację

Klasyfikator ten 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 (u mnie na Rys. 5) w której klasie i w którym obszarze przestrzeni cech występują błędy.



Rys 5 – Wykres błędów w wybranej przestrzeni cech

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 10.3: 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. Zapisz wyniki w tabeli znajdującej się na końcu instrukcji.

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 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 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 10.4: 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 10.5: Znormalizuj swoje dane. Czy skuteczność klasyfikacji klasyfikatora kNN wzrosła? Zapisz program aby móc zaprezentować wyniki. Zapisz uzyskane wyniki w tabeli znajdującej się na końcu instrukcji.

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 10.6: 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 tabeli znajdującej się na końcu instrukcji. Jak dobrze klasyfikator kNN daje się skalować do wielowymiarowych przestrzeni? Czy zwiększanie wartości k ma pozytywny czy negatywny wpływ na skuteczność klasyfikacji?

Wielowarstwowy perceptron

Kolejnym narzędziem do klasyfikacji danych wizyjnych które wykorzystamy w tym ćwiczeniu będzie znana nam już sieć neuronowa. Warto zauważyć, że struktura kodu którą przygotowaliśmy testując klasyfikator kNN nadaje się w zasadzie bez większych zmian do zastosowania sieci neuronowej. Musimy jedynie podjąć decyzję, czy chcemy przekazywać dane do klasyfikacji punkt-po-punkcie, czy (lepiej!) zbudować najpierw macierz testową zawierającą cechy ułożone analogicznie jak w macierzy treningowej.

Zadanie 10.7: W oparciu o wybrane zestawy cech użyte w zadaniu 10.6 przetestuj swój klasyfikator MLP. Uruchom kod kilka razy. Czy wyniki są takie same, czy się różnią? Zapisz swój program, aby móc zaprezentować go prowadzącemu, oraz zapisz uzyskane wyniki w tabeli znajdującej się na końcu instrukcji.

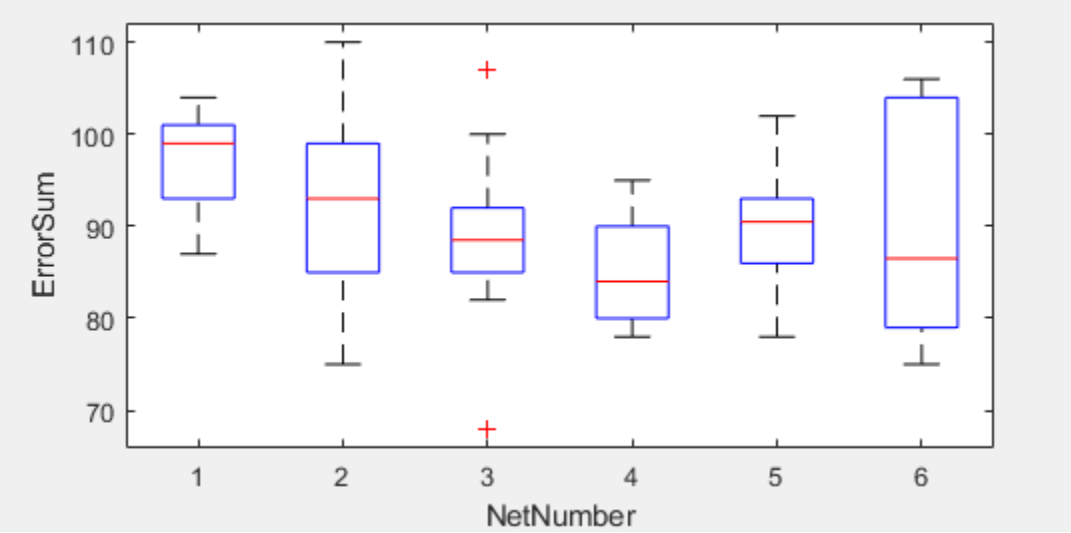
Optymalizacja metaparametrów sieci neuronowej

Dotarliśmy teraz do końcowego etapu naszego procesu. Choć z praktycznego punktu widzenia należałoby uruchomić jedną z wcześniej omawianych procedur optymalizacji metaparametrów, aby zaoszczędzić czas i zwiększyć szanse na poprawę wyników, tym razem podejdziemy do problemu inaczej. Przeanalizujemy statystycznie, które metaparametry rzeczywiście wpływają na jakość działania klasyfikatora. Wymaga to przeprowadzenia znacznie większej liczby testów i obliczeń niż byłoby to konieczne do znalezienia optymalnej struktury sieci, ale dzięki temu dowiemy się, które elementy są naprawdę istotne, a które można w przyszłości pominąć. Do wizualizacji wyników użyjemy wykresów pudełkowych (boxplotów). Załóżmy, że chcemy zbadać wpływ rozmiaru sieci —

i chcemy to zrobić w sposób statystyczny. Poniżej przedstawiono szkielet kodu do tego zadania. Zwróć uwagę na wyróżnione komentarze wskazujące, jakie elementy należy w tych miejscach umieścić:

```
for TestSerie = 1:6
   NetworkSize = [2*TestSerie 2*TestSerie]; % Number of neurons in all the hidden layers
   for Test = 1:10

   % Here we should have net initialization with defined network size (to reinitialize
internal weights and start training from scratch)

   % Here we should have the actual training of our net on the features from training
dataset

   % Here we simulate our net on the new (testing) data and calculate sum of errors for
a given net run and configuration...

     ErrorSum(Test,TestSerie) = % ...which we then store here.
   end
end

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

Jeśli wszystko przebiegło pomyślnie, po wykonaniu kodu powinieneś otrzymać macierz ErrorSum. Użycie polecenia boxplot na tej macierzy wygeneruje wykres podobny do tego przedstawionego na Rysunku 6. Na wykresie można zaobserwować wartości mediany (czerwone linie), pierwszy i trzeci kwartyl oraz wąsy rozciągające się od wartości minimalnych do maksymalnych — z wyłączeniem wartości odstających. Wartości odstające są oznaczone osobno jako czerwone znaki "+".



Rys 6 – Wykresy pudełkowe przedstawiające wyniki uzyskane dla 6 różnych sieci neuronowych. Możemy porównać, jak różne konfiguracje wpływają na rozrzut wyników oraz jak różnią się wartości mediany — mieszczące się w przedziale od 85 do niemal 100

Zadanie 10.8: Użyjmy teraz tej struktury kodu do statystycznej oceny wpływu różnych metaparametrów. Sprawdź następujące metaparametry, oceniając ich wpływ na średnią oraz rozrzut wyników:

- Szerokość sieci (liczba neuronów w warstwach: od 2 do 40)
- Głębokość sieci (1–3 warstwy ukryte, przy zachowaniu zbliżonej liczby wag)
- różne konfiguracje cech wejściowych — dwóch (różne konfiguracje), 4 oraz 6

Wyniki przedstaw na trzech wykresach pudełkowych, z wyrównaną osią Y — aby umożliwić lepsze porównanie. Na koniec sformułuj wnioski dotyczące twojej roli jako projektanta systemu decyzyjnego.

Zadanie 10.9: Zmodyfikuj zadanie 8 tak, aby uwzględniało również wyniki zadań 3 i 6 w formie graficznej. Zwróć uwagę, że te wcześniejsze zadania nie powinny być oceniane statystycznie (w tej konfiguracji są deterministyczne!) — a wszystkie potrzebne wartości masz już obliczone. Twoim zadaniem jest teraz zaprojektowanie wizualizacji, która umożliwi łatwe porównanie wszystkich metod.

Litery do klasyfikacji:		
1 para cech		
2 para cech		
3 para cech		
	Trening:	Walidacja:
Drzewo dec. 1 próba		
Drzewo dec. finalne		
kNN – 2 wymiary	-	
kNN – 4 wymiary	-	
kNN – 2 wymiary po normalizacji	-	
kNN – 4 wymiary po normalizacji	-	

kNN: wpływ ilości cech i sąsiadów:			
	2 cechy	4 cechy	6 cech
k = 1			
k = 3			
k = 5			
k = 9			

MLP: 3 uruchomienia:			
Próba 1			
Próba 2			
Próba 3			

Zadania rozszerzające:

Zadanie 10.10: 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 10.11: 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 10.12: 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 10.13: 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?

Zadanie 10.14: Porównaj naszą implementację klasyfikatora kNN oraz funkcję matlaba *fitcknn*. Zwróć uwagę zarówno na skuteczność jak I efektywność obliczeniową obu rozwiązań.