Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Podstawy Sztucznej Inteligencji I Uczenia Głębokiego

Inżynieria Mechatroniczna

Instrukcja 4:

Praktyczny System Decyzyjny

Nauczysz się: jak przeprowadzić typową procedurę obejmującą pozyskiwanie sygnału → obliczanie cech → wybór cech → trening modelu → optymalizację modelu, na podstawie symulowanych sygnałów drganiowych.

Dodatkowe materiały:

- Wykłady 3 - 5

Ta instrukcja została przetłumaczona przez ChatGPT na podstawie angielskiego oryginału – materiałów z przedmiotu *Basic of Artificial Intelligence and Deep Learning*

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

Autor instrukcji:
Ziemowit Dworakowski, zdw@agh.edu.pl

Wprowadzenie

Do tej pory rozwiązywaliśmy problemy tworzone sztucznie (czyli generowaliśmy punkty bez fizycznego znaczenia, a następnie na ich podstawie próbowaliśmy zbudować uczące się modele do regresji lub klasyfikacji). Choć ta zasada pozostanie dziś nadal aktualna, zrobimy krok wstecz, jeśli chodzi o sposób przetwarzania danych, i założymy, że rozwiązujemy rzeczywisty problem inżynierski na podstawie sygnałów surowych.

Wyobraź sobie, że otrzymujesz zadanie interpretacji danych z maszyny wirnikowej. Dane te mogą mieć postać sygnałów drganiowych w dziedzinie czasu. Na ich podstawie oczekuje się od Ciebie wydobycia informacji istotnych z punktu widzenia utrzymania ruchu.
Być może maszyna jest uszkodzona, a Twoim zadaniem jest to wykryć? A może masz rozpoznać stany robocze maszyny albo wywnioskować zależności między zarejestrowanymi wartościami? Niezależnie od przypadku, procedura przetwarzania informacji będzie podobna: zbierzesz dane, przetworzysz je w celu wyodrębnienia cech informacyjnych, zbudujesz system rozpoznawania wzorców i spróbujesz zoptymalizować cały proces modyfikując jego komponenty. Właśnie to będzie symulować to laboratorium.

Choć sygnały, z którymi będziesz pracować, są również generowane sztucznie (aby były przewidywalne i ciekawe z dydaktycznego punktu widzenia, a zarazem nie zajmowały dużo miejsca na dysku), są dość podobne do tych, które można zaobserwować w rzeczywistości.

Jeśli chcesz podejść do problemu z wykorzystaniem wiedzy dziedzinowej, warto wiedzieć, że:
– uszkodzenia łożysk będą przejawiać się głównie w wysokoczęstotliwościowych składnikach sygnału,
– niewyosiowanie pojawi się jako zwiększenie amplitudy częstotliwości wału,
– uszkodzenie zęba przekładni wywoła szpilki w sygnale, gdy uszkodzony ząb wchodzi w zazębienie,
– zmiany obciążenia spowodują ogólne zmiany amplitudy sygnału w określonych częstotliwościach.

Aby zadanie nie było zbyt proste (aby nie dało się łatwo odróżnić sygnałów uszkodzonych od nieuszkodzonych), wszystkie sygnały będą symulowały wyniki pomiarów uzyskanych dla różnych (nieznanych) prędkości obrotowych.

Przetwarzanie bazy sygnałów – tworzenie danych do własnych zadań

Zaczniemy od etapu pozyskiwania sygnałów. W rzeczywistości oznaczałoby to podłączenie czujników do maszyny, skonfigurowanie urządzeń akwizycji danych, a następnie pobranie z nich zarejestrowanych sygnałów. W naszym przypadku po prostu pobierzemy strukturę *AcquiredSignals* z naszego archiwum *StudentToolbox*.
Potrzebne będą jednak również etykiety dla danych – i to właśnie te etykiety będą definiować poszczególne zadania, które masz rozwiązać. Oznacza to, że w zależności od przydzielonego zadania ten sam zbiór danych będzie miał inne etykiety. Odpowiednie wektory etykiet znajdziesz w strukturze AcquiredSignals. Lista zadań znajduje się w Tabeli 5.1.

*Tabela 5.1 – Opis **indywidualnych*** zadań realizowanych w zakresie niniejszej instrukcji*

Kod zadania	Wymagane struktury	Symulowany problem
0	*AcquiredSignalsA, CodeVectorA1*	Wykrywanie rodzaju obciążenia
1	*AcquiredSignalsA, CodeVectorA2*	Wykrywanie uszkodzenia łożyska
2	*AcquiredSignalsA, CodeVectorA3*	Niewyosiowanie
3	*AcquiredSignalsA, CodeVectorA4*	Uszkodzony ząb przekładni
4	*AcquiredSignalsB, CodeVectorB1*	Wykrywanie rodzaju obciążenia
5	*AcquiredSignalsB, CodeVectorB2*	Wykrywanie uszkodzenia łożyska
6	*AcquiredSignalsB, CodeVectorB3*	Niewyosiowanie
7	*AcquiredSignalsB, CodeVectorB4*	Uszkodzony ząb przekładni

Pierwszym krokiem w każdej analizie tego typu jest podział danych na podzbiory treningowy, walidacyjny i testowy. Powinniśmy to zrobić zanim w ogóle spojrzymy na dane – aby uniknąć uprzedzeń przy formułowaniu wniosków na podstawie zbioru treningowego. Użyjmy więc naszych wektorów kodów, aby oznaczyć dane, a następnie podzielmy zbiór danych na podzbiory.

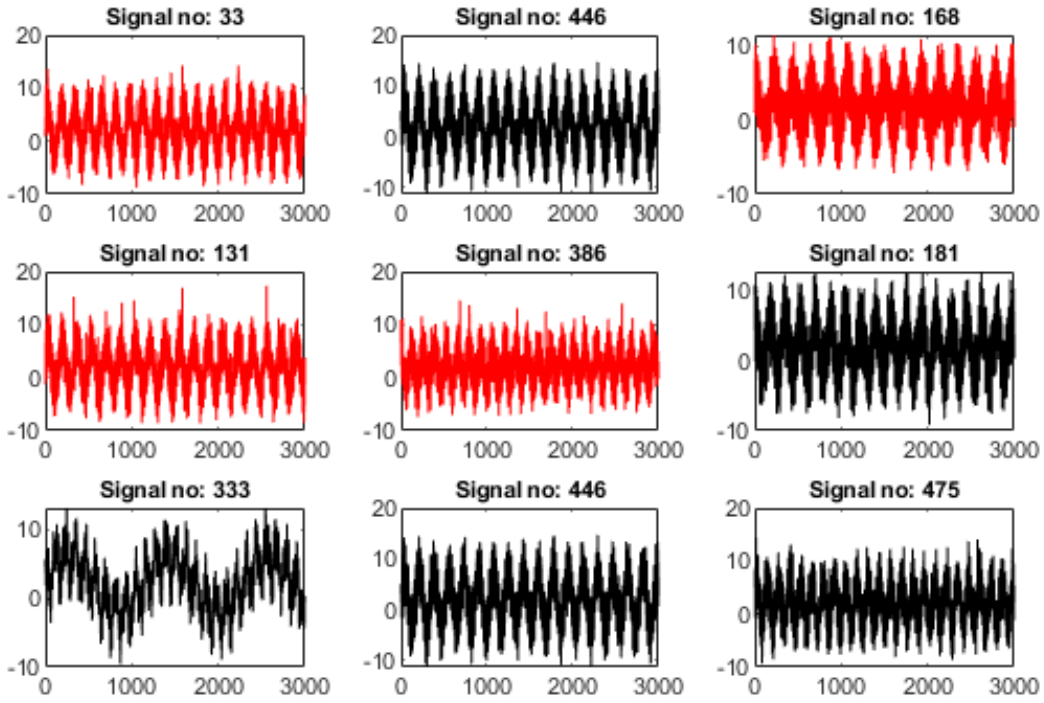
Zwróć uwagę na pomarańczowe fragmenty kodu – będziesz musiał dostosować nazwę *CodeVector* oraz plik *AcquiredSignal* tak, aby odpowiadały Twojemu indywidualnemu zadaniu z Tabeli 5.1:

```
load AcquiredSignalsA

% We'll create a randomly permutated vector of indices to keep track of
% where our data belongs.
RP = randperm(1000);
TrainIndices = RP(1:500);
ValIndices = RP(501:750);
TestIndices = RP(751:1000);

% Now we'll divide data into subsets:
TrainSignals = AcquiredSignalsA(TrainIndices);
TrainTargets = CodeVectorA1(TrainIndices);
ValSignals = AcquiredSignalsA(ValIndices);
ValTargets = CodeVectorA1(ValIndices);
TestSignals = AcquiredSignalsA(TestIndices);
TestTargets = CodeVectorA1(TestIndices);

% And now, similarily as before, we'll save our data subsets:
save TrainData TrainSignals TrainTargets
save ValData ValSignals ValTargets
save TestData TestSignals TestTargets
```

Zwróć uwagę, że po zapisaniu danych ten skrypt nie powinien być już więcej uruchamiany, ponieważ każde uruchomienie inaczej podzieli dane pomiędzy podzbiory.
Na koniec użyjemy poniższego kodu, aby wyświetlić 9 losowych sygnałów (zob. rys. 1):

```
figure(1);
for k = 1:9
    subplot(3,3,k);
    no = randi(500); % Random signal number
    if(TrainTargets(no)==1)
        plot(TrainSignals(no).sig,'k')
    else
        plot(TrainSignals(no).sig,'r')
    end
    title(['Signal no: ',num2str(no)])
end
```

Zadanie 5.1: Pobierz strukturę danych i oetykietuj ją, korzystając z kodów przypisanych do Twojego indywidualnego zadania (zobacz Tabelę 5.1). Wyświetl przykłady sygnałów należących do tej samej klasy oraz do przeciwnych klas. Spróbuj znaleźć jakiekolwiek cechy, które pozwoliłyby Ci „ręcznie" rozpoznawać te sygnały.



Rys 1 – Przykłady 9 losowych sygnałów z naszej bazy. Kolor odnosi się do etykiety sygnału dla danego zadania. Zauważ jak duża jest zmienność sygnałów w obrębie tej samej klasy – nie jesteśmy w stanie łatwo zaproponować reguły pozwalającej odróżnić próbki czerwone od czarnych.

Zwróć uwagę, że mimo iż każdy sygnał jest powiązany z wartością docelową, nie możemy jeszcze obejrzeć żadnej przestrzeni cech – ponieważ cechy nie zostały jeszcze wyodrębnione.
Na tym etapie nie mamy pewności, które cechy faktycznie pomogą nam rozwiązać problem, zacznijmy zatem od wyznaczenia cech ogólnych i zobaczmy, co się stanie.

Zwróć uwagę, że sposób przekazywania danych wejściowych i zwracania wyników powinien być podobny dla wszystkich funkcji ekstrakcji cech: każda z funkcji powinna przyjmować sygnał i zwracać jedną wartość, którą zapisujemy w naszej macierzy *Features*:

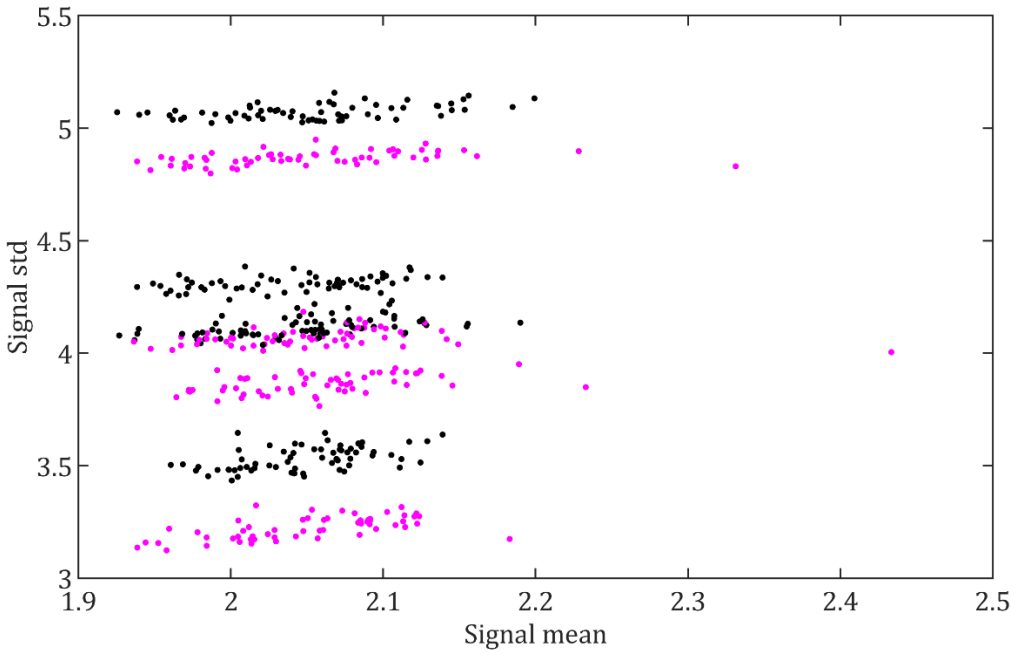
```
for k = 1:length(TrainSignals)
    Features_train(k,1) = mean(TrainSignals(k).sig);
    Features_train(k,2) = std(TrainSignals(k).sig);
    Features_train(k,3) = MyFeature(TrainSignals(k).sig);
    % ...
    % And here we'll want to have many different features of your choice
end
```

Możesz tworzyć własne funkcje oraz korzystać z funkcji wbudowanych, takich jak *kurtosis*, *skewness*, *peak2peak*, *rms* i inne. Ja zdecydowałem się użyć jednej cechy mojego własnego autorstwa (którą nazwałem *MyFeature*) – nie dlatego, że jest to konieczne (w rzeczywistości jest ona identyczna z wbudowaną funkcją MATLAB-a *peak2rms*), ale dlatego, że chciałem pokazać Ci, jak się to robi:

```
function[feature] = MyFeature(signal)
   m = max(abs(signal)); % We take the maximum absolute of a signal
   feature = m/rms(signal); % And then divide it by rms of the signal
end
```

Teraz możemy wyodrębnić cechy – i w końcu zobrazować dowolne dwuwymiarowe podprzestrzenie przestrzeni cech. Skojarzmy wartości docelowe z odpowiednimi kolorami:

```
figure(2);
for no = 1:length(TrainSignals)
    if(TrainTargets(no) == 1)
        plot(Features_train(no,1),Features_train(no,2),'.k'); hold on
    else
        plot(Features_train(no,1),Features_train(no,2),'.m'); hold on
    end
    xlabel('Signal mean')
    ylabel('Signal std')
end
```



Rys 2 – Wykres punktowy przestrzeni cech zbudowanej z dwóch cech które wyznaczyliśmy dla naszych sygnałów – z klasami reprezentowanymi poprzez kolor czarny i różowy.

Na Rys. 2 widzimy przestrzeń klasyfikacyjną opartą na dwóch cechach. Dane pojawiają się w klastrach – co możne oznaczać różne stany operacyjne lub warunki środowiskowe. Ta przestrzeń daje niezłe wyniki (istnieją klastry zawierające wyłącznie próbki „różowe" lub wyłącznie „czarne"), ale nadal jest daleka od ideału: klasy częściowo się pokrywają a cecha *Signal mean* wydaje się nie wnosić zbyt wiele do rozwiązania naszego problemu.

Zadanie 5.2: Oblicz co najmniej trzy dodatkowe cechy (oprócz tych, które zostały już obliczone w naszym kodzie), a następnie przedstaw wszystkie sześć cech dla całego podzbioru treningowego za pomocą dwuwymiarowych wykresów punktowych. Postaraj się znaleźć taką parę cech, dla której dane będą najłatwiejsze do oddzielenia. Zapisz wykres, który to pokazuje – aby móc przedstawić go prowadzącemu.

Wstępny system decyzyjny

Skoro mamy już cechy, które mogą posłużyć jako wejścia do naszego systemu, przepuśćmy dane przez sieć neuronową i sprawdźmy, czy jesteśmy w stanie zmusić ją do poprawnego rozpoznawania wzorców w zbiorze treningowym. Na tym etapie nie wiemy jeszcze, czy nasze cechy są wystarczająco informatywne, ile z nich powinniśmy użyć ani jak duży jest rzeczywisty poziom nakładania się klas, ale wykonajmy ten krok, aby mieć punkt odniesienia do dalszego doskonalenia. Musisz przygotować dane w taki sposób, aby pasowały do sieci neuronowej, którą projektowaliśmy w ramach Laboratorium 4.

Zadanie 5.3: Wytrenuj model sieci neuronowej (ANN) używając podzbioru treningowego oraz dwóch wybranych przez siebie cech, a następnie wykorzystaj podzbiór walidacyjny, aby oszacować, jak bardzo ogólne rozwiązanie otrzymałeś. Zapisz konfigurację sieci oraz uzyskany wynik walidacji w Tabeli 5.2 na końcu instrukcji. Ta sieć stanie się naszą bazową siecią odniesienia do dalszej oceny.

Uwaga! *Musisz obliczyć cechy dla zbioru walidacyjnego w taki sam sposób jak dla zbioru treningowego i zapisać je w macierzy Features val, w tych samych kolumnach, co w macierzy Features train. W przeciwnym razie Twoja sieć nie będzie w stanie rozpoznać nowych danych!*

Zadanie 5.4: Oceń znaczenie poszczególnych metaparametrów w naszej konfiguracji. Nie próbuj ich jeszcze optymalizować, ale sprawdź, jaki będzie wynik walidacji, jeśli zmodyfikujesz swój system decyzyjny w następujący sposób:

a) użyjesz znacznie szerszego modelu sieci ANN (więcej neuronów w warstwie),
b) użyjesz modelu o innej głębokości (więcej warstw ukrytych),
c) zastosujesz inny algorytm treningowy dla swojej sieci,
d) użyjesz innej pary cech,
e) użyjesz wszystkich obliczonych cech.

Następnie porównaj te wyniki z konfiguracją z zadania 5.3 (bazową siecią), ale ocenioną statystycznie – tak, aby sprawdzić, czy któraś z prób (a)–(e) daje istotną różnicę w stosunku do naturalnej zmienności procesu trenowania sieci neuronowej. Zapisz wyniki w Tabeli 5.2 i przygotuj się do omówienia ich z prowadzącym.

Uwaga*: punkty (a)–(e) to wyniki tylko jednej próby (bez powtórzeń). Robimy to w celu oszczędności czasu i uproszczenia obliczeń, ponieważ jesteśmy jeszcze w początkowej fazie strojenia modelu.*

Optymalizacja metaparametrów

Istnieją cztery podstawowe podejścia do ustawiania metaparametrów modelu:

Podejście 1: Dbamy jedynie o to, aby model był wystarczająco złożony, by uchwycić dane, i korzystamy z „wartości domyślnych" – lub kopiujemy wartości metaparametrów z podobnej konfiguracji. Takie podejście oszczędza czas i dane – poza kryterium zatrzymania (np. w treningu sieci neuronowej), nie trzeba przeznaczać dużych zbiorów danych na optymalizację, ani też rezerwować czasu na „próby konfiguracyjne", w których model jest trenowany tylko po to, by jego wydajność mogła zostać oceniona dla potrzeb strojenia metaparametrów, a nie dla ostatecznego celu zadania. Tego podejścia nie będziemy stosować w ramach naszej instrukcji.

Podejście 2: Wybieramy rozsądny punkt startowy – często oparty na wartościach domyślnych modelu lub podobnej konfiguracji znalezionej w literaturze – a następnie modyfikujemy pojedynczy metaparametr, sprawdzając, czy zmiana przynosi statystycznie istotną poprawę – i kontynuujemy ten proces, dopóki starczy nam czasu. W praktyce podejście to przypomina optymalizację 1+1. Czas alokowany jest w miarę potrzeb – dopóki widać poprawę. Jeśli chcielibyśmy zastosować to podejście do naszego modelu, możemy postępować zgodnie z poniższą procedurą:

- **A)** Wybierz dobry punkt startowy – np. na podstawie wyników z Zadania 5.4
- **B)** Oceń ten punkt statystycznie (zapisując np. średnią i odchylenie standardowe z 10–30 uruchomień, w zależności od tego, jak bardzo wyniki są spójne)
- **C)** Wybierz metaparametr i zmodyfikuj go (znacząco, ale nie skrajnie– np. o 20–30% wcześniejszej wartości)
- **D)** Powtórz krok B i na podstawie porównania obu serii zdecyduj, czy kontynuować w tym kierunku, cofnąć się, czy zmienić inny metaparametr
- **E)** Powtarzaj kroki B–D, aż poprawa przestanie być zauważalna w kolejnych próbach.

Podejście 3: Wybieramy te metaparametry, które naszym zdaniem mają największy wpływ na wynik, i przeprowadzamy pełną procedurę optymalizacji tylko dla nich (np. przy użyciu metody *grid search*). Zwróć uwagę, że ponieważ zakładamy, że funkcja celu jest relatywnie gładka, nie potrzebujemy aż tylu powtórzeń dla każdego punktu – bo możemy uśredniać wyniki w sąsiedztwie punktów siatki. Choć ryzykujemy wykonanie znacznie większej liczby obliczeń niż w innych podejściach, to zyskujemy dokładniejszy obraz kształtu funkcji celu. Jeśli chcesz zastosować to podejście, możesz kierować się następującą procedurą:

- **A)** Wybierz zestaw dwóch do trzech metaparametrów, które wydają się najistotniejsze (na podstawie wyników zadania 5.4)
- **B)** Zdefiniuj siatkę punktów w przestrzeni określonej przez te metaparametry. W każdym punkcie uruchom ocenę kilka razy. Dobierz gęstość siatki oraz liczbę uruchomień w każdym węźle, tak aby cała procedura nie trwała zbyt długo (załóż sobie rozsądne ograniczenia czasowe). Jeśli stosujesz gęstą siatkę i niewiele powtórzeń na węzeł, rozważ wygładzenie wyników – np. przez uśrednianie sąsiednich punktów.

Podejście 4: Przeprowadzamy jednowymiarową optymalizację dla każdego metaparametru osobno. Zaczynając od pewnych wybranych wartości, każdorazowo zmieniamy tylko jeden metaparametr, przechodząc do znalezionego minimum przed optymalizacją kolejnego. Choć procedura ta jest prosta i nie wymaga dużej mocy obliczeniowej, bardzo łatwo może utknąć w minimum lokalnym – ponieważ metaparametry często silnie wpływają na siebie nawzajem.

Jeśli chcesz zastosować to podejście, możesz postępować według poniższych kroków:

- **A)** Rozpocznij od punktu startowego – np. wybranego na podstawie wyników zadania 5.4
- **B)** Wybierz metaparametr, który wydaje się mieć największy wpływ na wynik. Optymalizuj tylko ten parametr, zamrażając pozostałe. Możesz tutaj zastosować przeszukiwanie losowe (*random search*) albo podejście siatkowe (*grid search*) – w tym przypadku siatka będzie jednowymiarowa.
- **C)** Powtarzaj krok B kolejno dla wszystkich jeszcze nieoptymalizowanych metaparametrów.

Zadanie 5.5: Wybierz dowolne z podejść do optymalizacji metaparametrów (z wyłączeniem oczywiście podejścia 1 „bez optymalizacji, zastosuj wartości domyślne”) i realizuj je tak długo, aż nie uda Ci się już bardziej poprawić systemu decyzyjnego. Sprawdź, czy uzyskane wyniki są statystycznie istotne – w porównaniu do wyników z zadania 5.3.
Na koniec oceń zoptymalizowane rozwiązanie na podstawie zbioru testowego, aby oszacować przyszłą skuteczność Twojego systemu. Zapisz wyniki w Tabeli 5.2.

Tabela 5.2 – Zebranie wyników instrukcji 5

Definicja zadania:	Zbiór sygnałów: Wektor kodowy: Rodzaj zadania:	
Wyniki sieci referencyjnej (5.3):	**Liczba błędów, trening:**	
	Liczba błędów, walidacja:	
Konfiguracja sieci referencyjnej: **(struktura i algorytm uczący)**		
Zastosowane cechy:		
Skuteczność statystyczna sieci referencyjnej (5.4):	**Średnia ilość błędów z 20 prób:**	
	Odchylenie standardowe w.w.:	
Skuteczność walidacyjna dla różnych konfiguracji sieci (5.4):	**Inna ilość neuronów w warstwie:**	**Inna ilość warstw:**
	Inny algorytm uczący:	**Inna para cech:**
	Wszystkie cechy:	
Resultat optymalizacji metaparametrów (5.5):	**Finalna konfiguracja:**	
	Błąd walidacyjny (średni i najmniejszy):	
	Bład testowy:	

Zadania dodatkowe:

Zadanie 5.6: Oblicz widmo częstotliwościowe sygnału, a następnie przefiltruj je, aby wyodrębnić kilka pasm częstotliwości. Dla każdego z tych pasm oblicz wartość skuteczną (RMS) – osobno dla każdego, tworząc w ten sposób zbiór cech opartych na RMS. Następnie wykorzystaj te cechy, aby sprawdzić, czy pozwalają one poprawić skuteczność systemu decyzyjnego, który zoptymalizowałeś w zadaniu 5.5.

Zadanie 5.7: Wybierz inne podejście do optymalizacji metaparametrów niż to, które zastosowane zostało w zadaniu 5.5 (czyli np. jeśli użyto *grid search*, zastosuj teraz podejście 1+1). Przeprowadź pełną procedurę optymalizacji metaparametrów zgodnie z alternatywnym podejściem, następnie porównaj oba uzyskane wyniki statystycznie – sprawdzając, czy różnice między nimi są istotne, i które podejście okazało się bardziej skuteczne lub stabilne.

Zadanie 5.8: Wykorzystaj inny wektor kodowy (ten, którego nie używałeś w Zadaniu 5.1) jako dodatkową cechę i sprawdź, czy jego użycie pozwala na statystycznie istotną poprawę wyników. Przetestuj to kilkukrotnie, używając różnych dostępnych wektorów kodowych.
Spróbuj wyjaśnić uzyskane wyniki: Czy dodanie nowej cechy pomogło? Dlaczego tak się stało?

Zadanie 5.9: Zwróć uwagę, że nasze dane są wyraźnie skupione w klastrach. Możliwe więc, że klasyfikację dałoby się wykonać głównie na podstawie etykiet klastrów.
Wybierz dwuwymiarową przestrzeń cech, w której widać wyraźne klastry danych (podobnie jak na rysunku 2), a następnie:
a) Przeczytaj dokumentację metody k-means w MATLAB-ie (*doc kmeans*)
b) Uruchom kmeans na danych treningowych i walidacyjnych razem (czyli na zbiorze będącym połączeniem obu podzbiorów), aby uzyskać indeksy klastrów
c) Przypisz etykiety klas do klastrów na podstawie najczęściej występującej etykiety treningowej w danym klastrze (czyli np. jeśli ponad 50% punktów treningowych w „klastrze o numerze 3" ma etykietę 1 – przypisz temu klastrowi etykietę 1)
d) Sklasyfikuj dane walidacyjne na podstawie przypisanych w punkcie (c) informacji o klasach
e) Porównaj uzyskane wyniki z wynikami zadań 5.5 oraz 5.3.