Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Podstawy Sztucznej Inteligencji I Uczenia Głębokiego

Inżynieria Mechatroniczna

Instrukcja 2:

Algorytmy regresji liniowej i wielomianowej, przechowywanie i prezentacja wyników

Nauczysz się: Jak napisać od podstaw i skonfigurować podstawowe algorytmy regresyjne: regresję liniową i wielomianową – w dwuwymiarowej przestrzeni cech. Dodatkowo nauczysz się również w jaki sposób przechowywać wyniki generowane przez systemy decyzyjne - co przyda się w trakcie kolejnych instrukcji przedmiotowych.

Dodatkowe materiały:

- Wykład nr 1 [*obowiązkowy*]
- Wykład nr 2 [obowiązkowy]

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

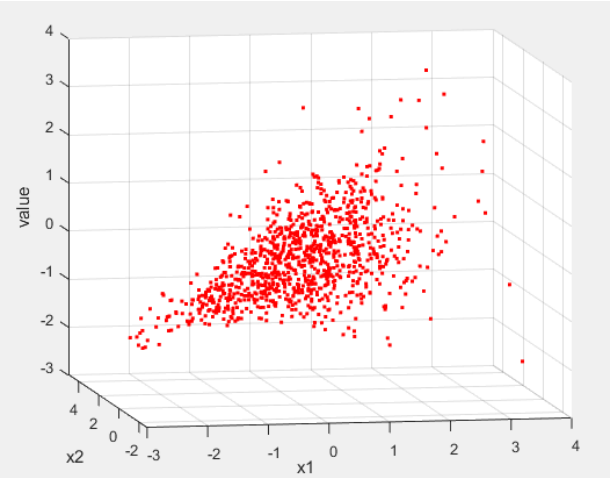
Koordynator przedmiotu:
Krzysztof Holak, holak@agh.edu.pl

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

Regresja liniowa i wielomianowa

Zacznijmy od wczytania i wyświetlenia zbioru danych dla dwuwymiarowego problemu regresji. Zbiory danych, na których będziemy pracować w tym zadaniu, znajdują się w folderze *RegressionData*.
Na początek użyję zbioru *DataForDemonstration*, który po wczytaniu i wyświetleniu za pomocą poniższego kodu wygląda jak na rys. 1:

```
load RegressionTrainingData


figure;
plot3(RegressionTrainingData(:,1), RegressionTrainingData(:,2), RegressionTrainingData(:,3),'.r')
xlabel('x1')
ylabel('x2')
zlabel('value')
grid on
```



Rys 1 – Przykładowy zbiór danych

Trzecią kolumnę danych potraktujemy jako wartość docelową, którą powinniśmy nauczyć się przewidywać. Naszym celem jest zbudowanie regresora. Musimy w jakiś sposób znaleźć współczynniki płaszczyzny regresji – modelu, który umożliwi przewidywanie nowych próbek. Dokonamy tego za pomocą optymalizacji. Naszą funkcją celu będzie błąd między przewidywanymi a rzeczywistymi wartościami, a rozwiązanie znajdziemy przy użyciu algorytmów opracowanych ostatnim razem.

Zacznijmy od tego prostego modelu:

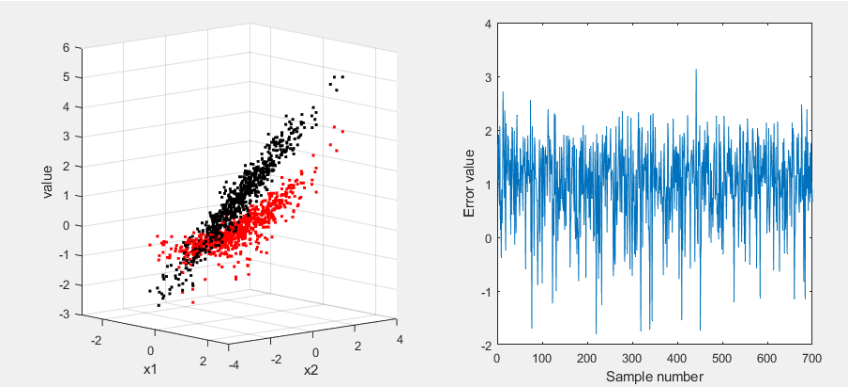
```
function [PredictedValue] = LinearRegressionModel(X,W)
        PredictedValue = W(1)*X(1) + W(2)*X(2) + W(3);
end
```

Zwróć uwagę, że jest to funkcja (zapisz ją jako osobny plik), która przyjmuje wektor cech X=[x1,x2] oraz wektor parametrów W=[W1,W2,W3] który określa sposób działania modelu. Ustalmy wartości wektora parametrów dowolnie i przepuśćmy wszystkie dane treningowe przez model, aby zobaczyć, co się stanie:

```
W1 = 1;   % Since we don't know what the model parameters should be – lets start with ones.
W2 = 1;
W3 = 1;

for k = 1:length(RegressionTrainingData)
    PredValue(k) = LinearRegressionModel([RegressionTrainingData(k,1),...
RegressionTrainingData(k,2)],[W1,W2,W3]);
    % The third column of TrainingData is our target
    Error(k) = PredValue(k) - RegressionTrainingData(k,3);

end

MSE = mse(RegressionTrainingData(:,3),PredValue');
```

Teraz możemy zobrazować wartości błędu lub pokazać, jak bardzo nasze przewidywania odbiegały od rzeczywistych punktów – zobacz wynik na rys. 2:

```
figure;
subplot(1,2,1)
  plot3(RegressionTrainingData(:,1),RegressionTrainingData(:,2),RegressionTrainingData(:,3),'.r'); hold on
  plot3(RegressionTrainingData(:,1),RegressionTrainingData(:,2),PredValue,'.k');
  xlabel('x1');  ylabel('x2')
  zlabel('value')
  grid on
subplot(1,2,2)
  plot(Error)
  xlabel('Sample number');  ylabel('Error value')
```



Rys 2 – Błąd regresji w przestrzeni cech (z lewej) oraz wyświetlony dla poszczególnych próbek (z prawej)

To jest dobry punkt wyjścia do budowy optymalizatora. Wystarczy, że umieścimy wszystkie wcześniejsze fragmenty kodu w funkcji, która stanie się naszą nową *FunctionForOptimization*. Jeśli *FunctionForOptimization* przyjmie wektor jako wejście a wartość MSE zwróci na wyjściu, będzie można łatwo użyć jej w algorytmach optymalizacyjnych opracowanych podczas laboratorium 1.

Uwaga: Jeśli funkcja ma działać szybko, należy wyłączyć wszystkie wizualizacje (*czyli wszystkie wykresy*).

Zadanie 2.1: Wczytaj swój **indywidualny** zbiór danych treningowych (*RegressionData_n*, gdzie n to numer Twojego zadania). Utwórz nową funkcję, nazwij ją np. *LinearRegressorCheck*. Niech funkcja przyjmuje wektor parametrów W = [W1, W2, W3] i zwraca wartość MSE dla wszystkich punktów w zbiorze *RegressionTrainingData*. Oceń funkcję dla kilku różnych wartości W1, W2 i W3, spróbuj znaleźć taki zestaw, który daje rozsądnie dobrą wartość. Zapisz wyniki w tabeli 2.1.

Zadanie 2.2: Użyj funkcji *LinearRegressorCheck* jako funkcji do optymalizacji w swoim algorytmie przeszukiwania siatkowego (grid search) opracowanym w ramach laboratorium 1. Znajdź optimum i zapisz je w Tabeli 2.1.

Zadanie 2.3: Użyj funkcji *LinearRegressorCheck* jako funkcji do optymalizacji w swoim algorytmie 1+1 z adaptacyjnym krokiem opracowanym w ramach laboratorium 1. Znajdź optimum i zapisz je w Tabeli 2.1.

Zadanie 2.4: Użyj funkcji *LinearRegressorCheck* jako funkcji do optymalizacji w swoim gradientowym wielostartowym algorytmie optymalizacyjnym z adaptacyjnym krokiem opracowanym w ramach laboratorium 1. Znajdź optimum i zapisz je w Tabeli 2.1.

Zadanie 2.5: Zmodyfikuj swój model regresyjny by umożliwić korzystanie z bardziej złożonej geometrii. Zastosuj model wielomianowy dany równaniem:

$$y = w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2 + w_6$$

Zoptymalizuj ten 6-wymiarowy model z zastosowaniem algorytmu 1+1 z adaptacyjnym krokiem. Zanim zapiszesz rezultaty w tabeli 2.1, dokonaj konfiguracji algorytmu by konsekwentnie odnajdywał minimum lokalne. Dobierz samodzielnie ilość iteracji i metaparametry metody, przygotuj się do obrony (uzasadnienia) swoich decyzji.

Zadanie 2.6: Powtórz zadanie 2.5, tym razem używając algorytmu gradientowego, wielostartowego. Zastosuj ilość sprawdzeń funkcji celu równą tej wykorzystanej w zadaniu 2.5 (by zapewnić uczciwe porównanie metod optymalizacyjnych). Skonfiguruj ilość startów algorytmu wielostartowego i pozostałe metaparametry związane z adaptacyjnym krokiem, by zapewnić jak najwyższą powtarzalność wyniku i jak najmniejszą wartość MSE. Zapisz finalne rezultaty w tabeli 2.1

Zadanie 2.7: Jak dotąd, używaliśmy wyłącznie zbioru treningowego. Czas na wykorzystanie drugiego, do tej pory nieużywanego podzbioru danych. Użyj odnalezionych w zadaniach 2.2 – 2.6 parametrów rozwiązania aby sprawdzić jego skuteczność na podzbiorze walidacyjnym naszego zbioru danych. Zapisz wyniki w tabeli 2.1.

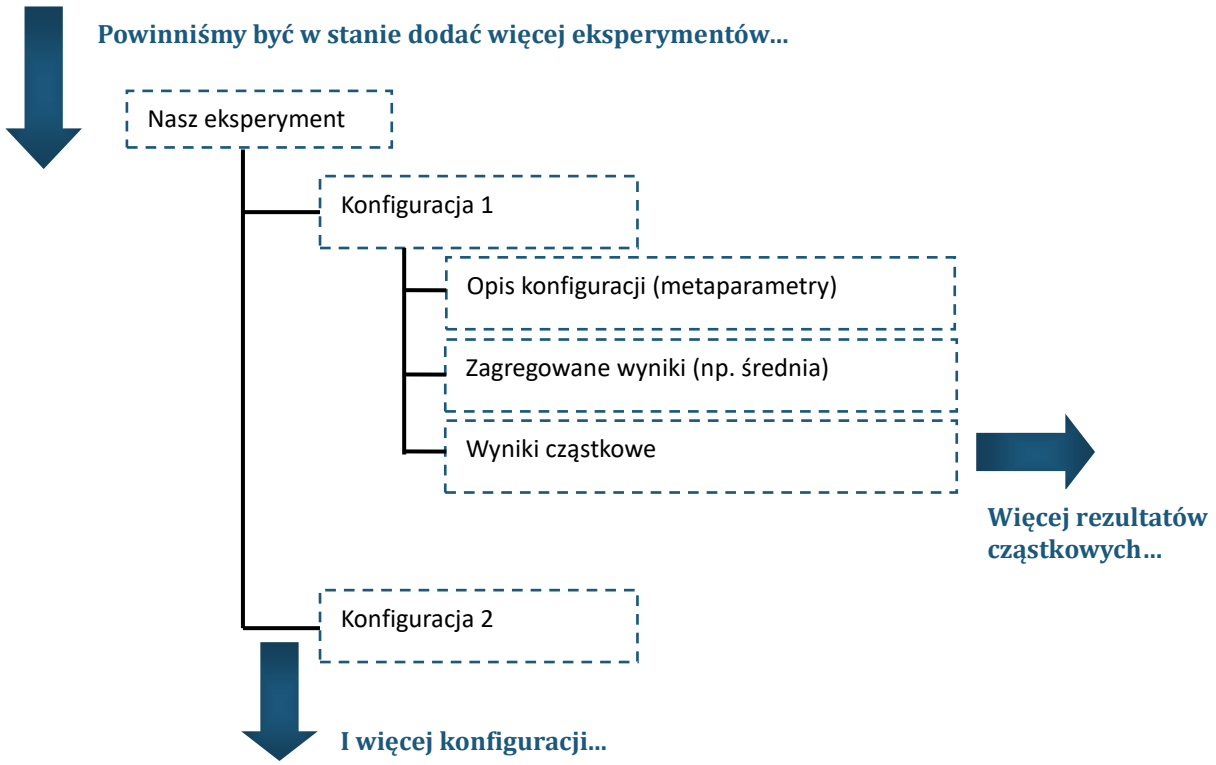
Przechowywanie i prezentacja statystycznych rezultatów testów systemów decyzyjnych

Do tej pory zapisywaliśmy wyniki w tabelach (prawdopodobnie ręcznie w notesie lub ręcznie adnotując plik instrukcji PDF). Nie jest to jednak optymalne rozwiązanie. Od teraz używajmy do tego struktur danych.

Załóżmy, że przeprowadzamy eksperyment, w którym wielokrotnie testujemy pewną konfigurację algorytmu, a następnie wykonujemy podobne testy dla innej konfiguracji. Chcielibyśmy przechowywać te wyniki w strukturze drzewa, która wyglądałaby jak na rys. 6.

Tabela 2.1: Rezultaty regresji

	W1	W2	W3	MSE			MSE walidacja
Ręczna konfiguracja 1							-
Ręczna konfiguracja 2							-
Ręczna konfiguracja 3							-
Grid search							
Algorytm 1+1				Try 1:	Try 2:	Try 3:	
Metoda gradientowa				Try 1:	Try 2:	Try 3:	
1+1: model wielomianowy	-	-	-	Try 1:	Try 2:	Try 3:	
gradient: model wielomianowy	-	-	-	Try 1:	Try 2:	Try 3:	



Rys 6 – Struktura danych do przechowywania wyników eksperymentalnych

Na potrzeby wyjaśnienia wyobraźmy sobie, że ponownie pracujemy z pierwszą instrukcją i testujemy algorytmy optymalizacji na naszych sztucznych funkcjach 2D i 4D. Pamiętasz, jak w zadaniu 1.8 otrzymaliśmy wektor wyników (*Results*), średnią i odchylenie standardowe? Teraz wykorzystajmy to do zilustrowania, jak możemy przechowywać te wyniki.

Wiemy, że mieliśmy dwa algorytmy do oceny (*wielostartowy gradient* i *1+1*) oraz trzy funkcje, na których przeprowadzaliśmy testy (*fewminima*, *manyminima*, *multidimensional*). Powinniśmy także mieć możliwość testowania różnych konfiguracji algorytmów, czyli różnych wartości metaparametrów. Najprostszym sposobem na zdefiniowanie zagnieżdżonej struktury danych do przechowywania tych wyników jest umieszczenie danych na poziomie, na którym chcemy je przechowywać – a reszta struktury zostanie zorganizowana odpowiednio:

```
for repetition = 1:20

   %% Here we have a test which end in one Result that is stored in the Results vector:
   %% ...
   Results(repetition) = BestHistory(end)

end
% Here we calculate statistics from our run:
mean(Results)
std(Results)

% Here we store all the necessary information into the data structure:

AggregatedResults.MultistartGradient.Fewminima(1).Results = Results;
AggregatedResults.MultistartGradient.Fewminima(1).Mean = mean(Results);
AggregatedResults.MultistartGradient.Fewminima(1).Std = std(Results);
AggregatedResults.MultistartGradient.Fewminima(1).Config.Starts = 5;
AggregatedResults.MultistartGradient.Fewminima(1).Config.MaxSteps = 100;
...
```

Jeśli następnie chcielibyśmy przetestować kolejną metodę optymalizacji, na przykład: *1+1*, wystarczyłoby zmienić odpowiedni poziom struktury:

```
AggregatedResults.OnePlusOne.Fewminima(1).Results = Results;
AggregatedResults.OnePlusOne.Fewminima(1).Mean = mean(Results);
AggregatedResults.OnePlusOne.Fewminima(1).Std = std(Results);
AggregatedResults.OnePlusOne.Fewminima(1).Config.InitialStep = 7;
AggregatedResults.OnePlusOne.Fewminima(1).Config.ReductionFactor = 0.8;
AggregatedResults.OnePlusOne.Fewminima(1).Config.AdaptationTrigger = 3;
```

Jeśli zdecydujemy się uruchomić ten sam algorytm dla tego samego problemu, ale z inną konfiguracją (różnymi wartościami metaparametrów), możemy po prostu dodać kolejne pole w wektorze *Fewminima*:

```
AggregatedResults.OnePlusOne.Fewminima(2).Results = Results;
AggregatedResults.OnePlusOne.Fewminima(2).Mean = mean(Results);
AggregatedResults.OnePlusOne.Fewminima(2).Std = std(Results);
AggregatedResults.OnePlusOne.Fewminima(2).Config.InitialStep = 4;
AggregatedResults.OnePlusOne.Fewminima(2).Config.ReductionFactor = 0.6;
AggregatedResults.OnePlusOne.Fewminima(2).Config.AdaptationTrigger = 6;
```

Możesz także łączyć części struktury. Poniższy kod jest więc równoważny poprzedniemu:

```
Config.InitialStep = 4;
Config.ReductionFactor = 0.6;
Config.AdaptationTrigger = 6;
AggregatedResults.OnePlusOne.Fewminima(2).Results = Results;
AggregatedResults.OnePlusOne.Fewminima(2).Mean = mean(Results);
AggregatedResults.OnePlusOne.Fewminima(2).Std = std(Results);
AggregatedResults.OnePlusOne.Fewminima(2).Config = Config;
```

Jedynym problemem, który nam umyka, jest fakt, że nasza struktura zostanie utracona po zamknięciu kodu – a chcemy mieć jedną wspólną strukturę danych dla wszystkich uruchamianych kodów. Z tego powodu potrzebujemy sposobu na wczytanie istniejącej struktury danych oraz jej zapis po zakończeniu pracy:

```
if(exist('AggregatedResults'))
    load AggregatedResults
else  % If the structure does not exist - it will just be generated and saved at the end of our code
end


%% Here we have an actual method (maybe some tests, repetitions, etc.)
% ...
% ...
% ...



% Here we are storing the results of our code:
Config.InitialStep = 4;
Config.ReductionFactor = 0.6;
Config.AdaptationTrigger = 6;
AggregatedResults.OnePlusOne.Fewminima(2).Results = Results;
AggregatedResults.OnePlusOne.Fewminima(2).Mean = mean(Results);
AggregatedResults.OnePlusOne.Fewminima(2).Std = std(Results);
AggregatedResults.OnePlusOne.Fewminima(2).Config = Config;

% Here we save the data structure with new contents:
save AggregatedResults AggregatedResults
```

Jak pokazać wyniki zawarte w strukturze danych?

Jeśli dane są już zapisane w Twojej strukturze, możesz łatwo wyświetlić jej części, np. w taki sposób:

```
plot(AggregatedResults.OnePlusOne.Fewminima(1).Results,'r'); hold on
plot(AggregatedResults.MultistartGradient.Fewminima(1).Results,'b'); hold on
legend('OnePlusOne','MultistartGradient')
xlabel('Run number')
ylabel('Result')
```

Lub w ten sposób:

```
VectorOfResults = [AggregatedResults.OnePlusOne.Fewminima(1).Mean,…
AggregatedResults.OnePlusOne.Fewminima(2).Mean,…
AggregatedResults.OnePlusOne.Fewminima(3).Mean,…
AggregatedResults.OnePlusOne.Fewminima(4).Mean]
bar(VectorOfResults); hold on
```

Lub po prostu wyświetl je w linii komend:

```
AggregatedResults.OnePlusOne.Fewminima(1).Mean
AggregatedResults.MultistartGradient.Fewminima(1).Mean
```

Zadanie 2.8: Przejrzyj ponownie niniejszą instrukcję i przygotuj sposób przechowywania danych z wielu eksperymentów. Następnie przeprowadź ocenę statystyczną modeli liniowych i wielomianowych dla regresji (zadania 2.3, 2.4, 2.5 i 2.6). Każde zadanie należy uruchomić co najmniej 20 razy.

Twoja struktura danych powinna zawierać:
- Informacje o rodzaju rozwiązywanego zadania (regresja)
- Informacje o użytym klasyfikatorze
- Informacje o metaparametrach użytego klasyfikatora
- Oszacowania statystyczne uzyskane na podstawie wielu uruchomień (średnia i odchylenie standardowe)
- Wyniki źródłowe (wyniki każdego uruchomienia)
- Informacje o efektywności walidacji obliczonej w zadaniu 2.6 dla modeli regresyjnych

Graficzna prezentacja danych statystycznych

Gdy Twoje dane będą już przygotowane, nadszedł czas, aby przedstawić je w sposób łatwy do zrozumienia. Tym razem nie dostaniesz dokładnego kodu do wykonania tego zadania – konieczne będzie rozwiązanie go, korzystając z dokumentacji MATLAB.
Rozpocznij od struktury danych przygotowanej w ramach zadania 2.7, a następnie porównaj różne algorytmy i ich konfiguracje, używając wykresów i diagramów słupkowych. Krótki opis składni funkcji można znaleźć za pomocą polecenia *help*, np.: „*help bar*”. Pełna dokumentacja, z uwzględnieniem przykładów użcia, dostępna jest za pomocą komendy *doc,* np. „*doc bar*”. Poniżej przedstawiłem listę funkcji, które warto by było poznać:

bar pozwala na ryzowanie diagramów słupkowych (również wielokolorowych – do porównywania różnych wektorów)
subplot pozwala na umieszczenie wielu wykresów na jednym obrazie
line pozwala na rysowanie linii na diagramach I wykresach – np. Do oznaczania poziomu jakiejś zmiennej w porównaniu do pozostałych wyświetlanych zmiennych.

Zadanie 2.9: Przedstaw dane z laboratorium 2 za pomocą zaprojektowanych przez siebie wizualizacji. Twoje wykresy powinny zawierać legendy, etykiety osi oraz różne kolory, aby wyraźnie pokazać różne wartości. Postaraj się, aby wyglądały profesjonalnie i były jak najbardziej dopracowane.

Wykresy powinny zawierać co najmniej:

- Porównanie średnich wyników walidacyjnych i treningowych z zadania 2.7 za pomocą diagramów słupkowych dla trzech różnych modeli – na trzech pod-wykresach w jednym oknie.

- Wizualizację wyników źródłowych dla statystyki (czyli wyników każdego z 20 uruchomień modelu) wraz z linią oznaczającą średnią wartość oraz przerywanymi liniami wskazującymi odchylenie standardowe wyników.

Zadania dodatkowe:

Zadanie 2.10: Zmodyfikuj zadanie 2.5, tak aby zamiast modelu wielomianowego używało estymatora lokalnie liniowego (locally weighted regression). Następnie porównaj wyniki uzyskane na zbiorze walidacyjnym z odpowiednimi wynikami z poprzednich zadań. Zwróć uwagę, że aby uzyskać predykcję punktów w zbiorze walidacyjnym, powinieneś używać danych treningowych (zbiór walidacyjny dostarcza jedynie punktów do oceny, natomiast punkty, w które dopasowujesz krzywe, nadal powinny pochodzić ze zbioru treningowego).

Zadanie 2.11: Zmodyfikuj zadanie 2.6, aby używało wielomianu wyższego rzędu. Skonfiguruj metodę treningową, dopasuj ten wielomian do danych treningowych, a następnie oceń jego skuteczność na zbiorze walidacyjnym. Porównaj rozwiązania *1+1* oraz gradientowe jako algorytmy optymalizacyjne. Czy różnica między nimi jest podobna jak wcześniej, czy też nastąpiła zmiana w relatywnych wynikach?