Metody Inteligencji Obliczeniowej

Dokumentacja projektu

Temat projektu:

Dostrajanie klasyfikatora FUZZY LOGIC z użyciem procedury PSO

Kateryna Andrusiak, Vadym Semkovych, Hleb Shypula

Informatyka Stosowana

Wydział Fizyki i Informatyki Stosowanej

Akademia Górniczo-Hutnicza w Krakowie

Opis projektu

Celem projektu jest wykorzystanie układu FUZZY LOGIC, którego zadaniem byłaby klasyfikacja danych (IRIS, WINE oraz SEEDS) z zbioru benchmarkowego UCI MLR http://archive.ics.uci.edu/ml/datasets.php.

Należy podzielić dane z powyższych zbiorów na uczące i testujące, dalej zbudować układ FUZZY LOGIC, a następnie wykorzystać algorytm optymalizacji PSO.

Realizacja projektu

Project jest realizowany w środowisku MATLAB.

Link do repozytorium:
https://github.com/hlebshypulahub/MIO-Project-WFiIS-2021/blob/master/project.m

1. Pierwszym krokiem jest inicjalizacja zbiorów, w naszym przypadku to są zbiory:
   - IRIS – każdą próbkę opisują cztery cechy, czyli nasz układ FL będzie miał 4 wejścia.
   - WINE – każdą próbkę opisują 13 cech, w naszym projekcie wykorzystujemy tylko 7.
   - SEEDS – każdą próbkę opisują 7 cech (7 wejść).

2. Następnie losowo mieszamy próbki zbioru.

3. Dzielimy nasze dane ze zbioru na uczące (80% zbioru) i testujące (20%).

4. Dalej inicjalizujemy FIS (fuzzy inference system) za pomocą funkcji:
   ```
   fis = genfis(inputData,outputData,options)
   ```
   która zwraca FIS wygenerowany przy użyciu określonych danych wejściowych, wyjściowych i opcji. W naszym przypadku ustawiamy opcję `'SubtractiveClustering'`.

5. Ewaluujemy nasz wygenerowany FIS z użyciem funkcji:
   ```
   output = evalfis(fis,input)
   ```
   która zwraca jednowyjściowy Sugeno FIS przy użyciu podziału siatki (grid partititon) danych wejściowych i wyjściowych.

6. Po ewaluacji pobieramy z FIS dane potrzebne dla PSO (Particie Swarm Optimization) za pomocą funkcji:
   ```
   [in,out,rule] = getTunableSettings(fis)
   ```
   zwraca przestrajalne ustawienia wejść, wyjść i reguł systemu rozmytego FIS.

   ```
   paramvals = getTunableValues(fis,paramset)
   ```
   zwraca przestrajalne wartości parametrów systemu wnioskowania rozmytego FIS.

7. Definiujemy granice danych wejściowych określone jako wektor na potrzeby PSO. lb reprezentuje granice dolną, ub - górną ($lb \leq x \leq ub$).

8. Definiujemy funkcje przystosowania, która pobiera x – parametry funkcji przynależności oraz zwraca błąd przystosowania.

```
%%--%%--%% Funkcja Fitness
function fitness = fun(x)
%%% Funkcje przynależności są typu Gaussa, nie mogą przyjmować 0 jako
%%% parameter odchylenia std, dlatego jeżeli jakaś wartość wektora jest 0,
%%% randomizujemy wartość bliską do 0 i zmieniamy x(i)
for i = 1:size(x, 2)
    if x(i) == 0
        x(i) = 0.001 + rand * (0.05 - 0.001);
    end
end

fis_test = getGlobalfis;
in = getTunableSettings(fis_test);

paramVals = x;
fis_test = setTunableValues(fis_test, in, paramVals);

y_pso = evalfis(fis_test, getGlobal_x_u);

for i = 1:size(y_pso, 1)
    y_pso(i) = round(y_pso(i));
end

temp = y_pso - getGlobal_y_u;
q = find(temp == 0);

fitness = (size(y_pso, 1) - size(q, 1)) / size(y_pso, 1);

end
```

9. Zatem możemy wykonać PSO za pomocą funkcji:

x = particleswarm(fun,nvars,lb,ub,options)

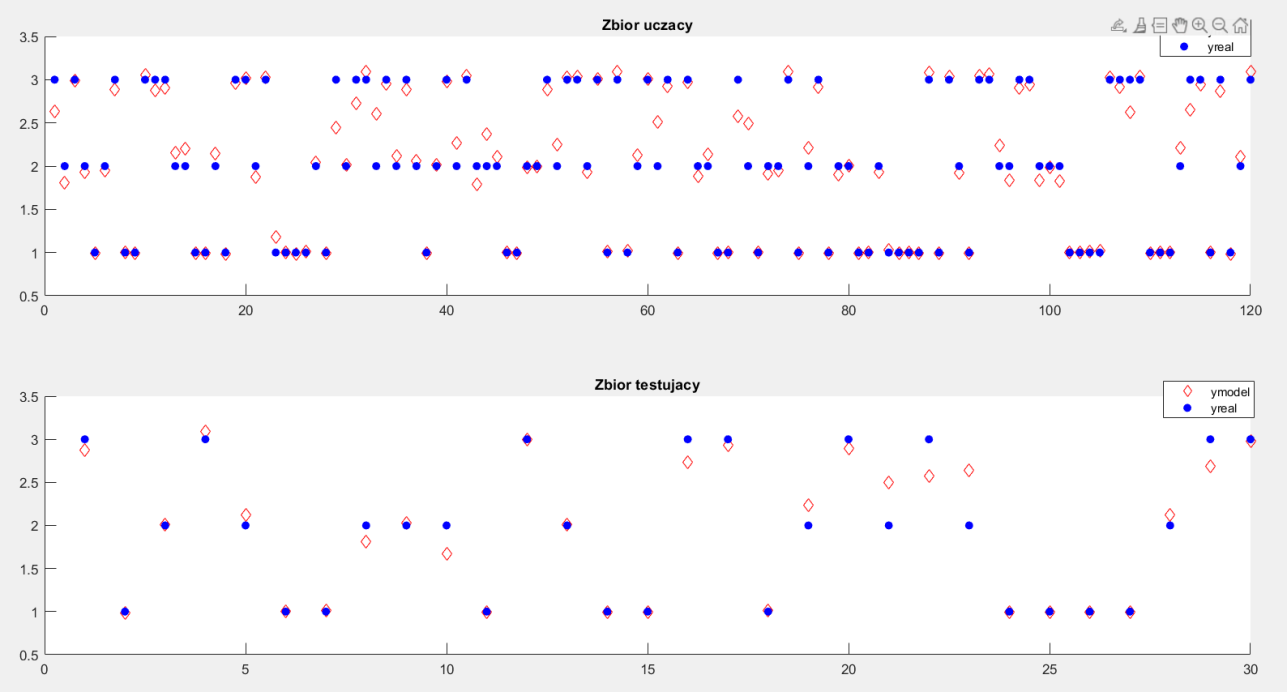
gdzie **fun** – funkcja przystosowania (fitness) (p. 8),
nvars – ilość parametrów,
lb, ub – granice (p.7)
options – opcje wykonywania optymizacji

```
options = optimoptions('particleswarm', 'PlotFcns', @pswplotbestf, ...
    'SwarmSize', 20, 'MaxIterations', 1000*(sum(bound_in))/20, 'MaxStallIterations', 15);
```

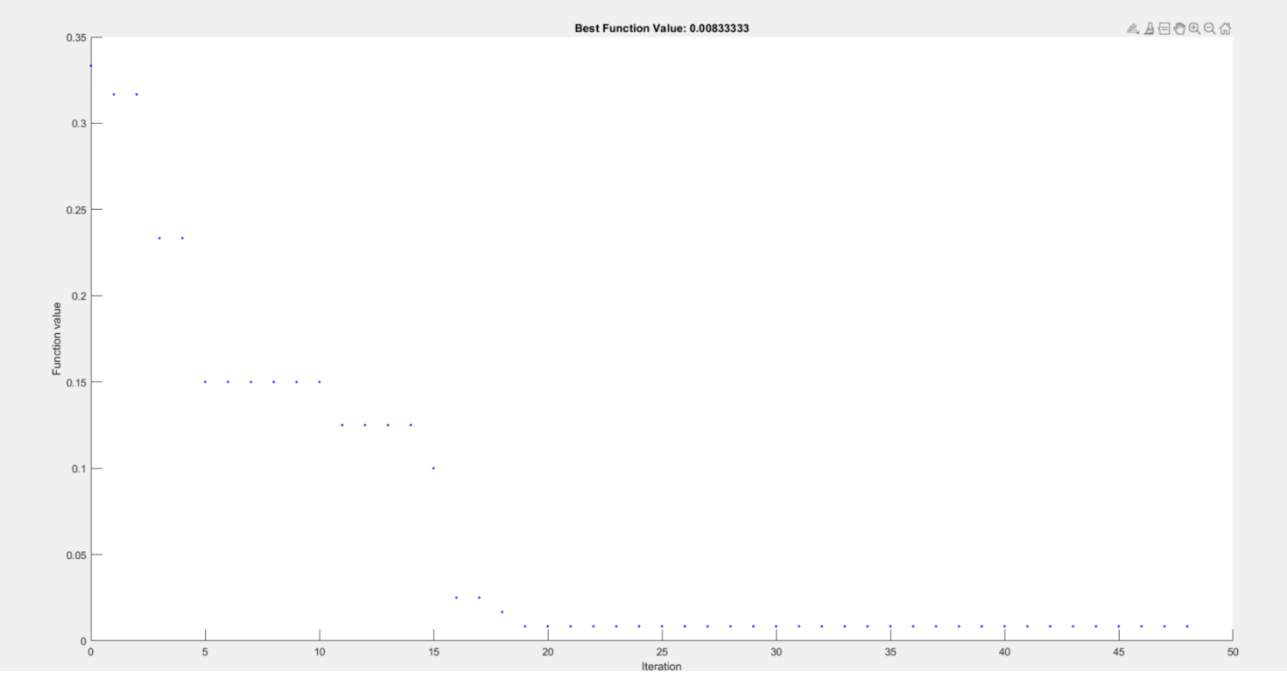
10. Analiza wyników.

Analiza wyników

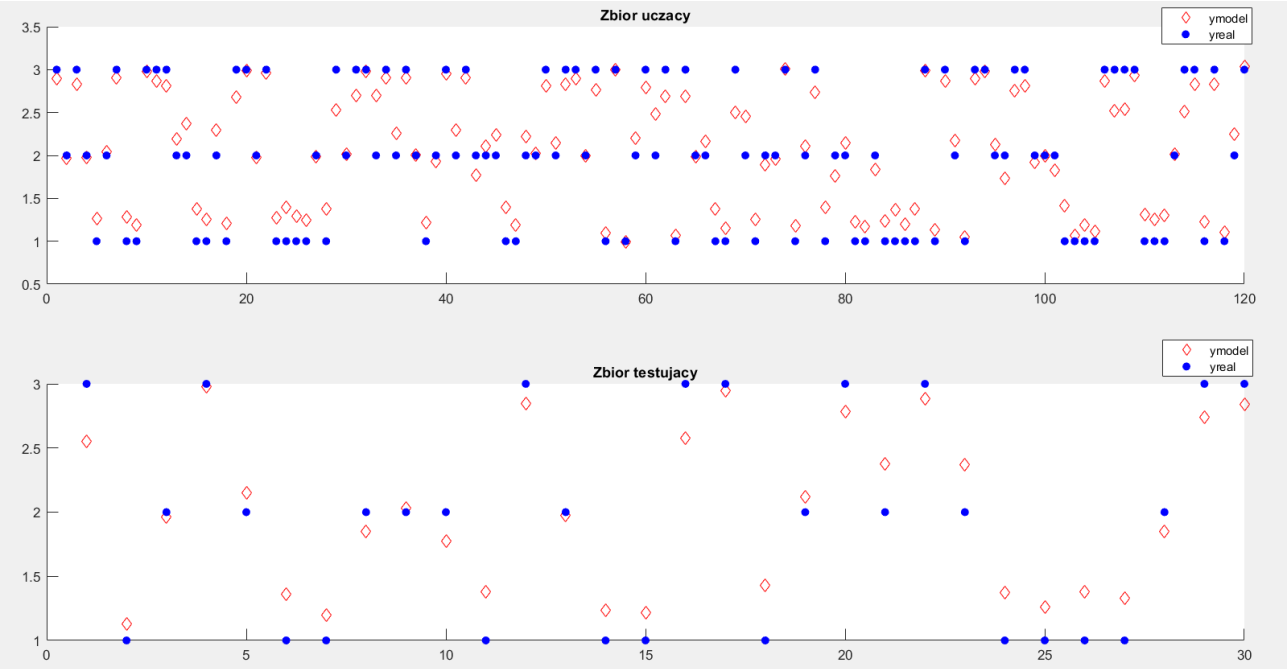
- **IRIS**



Rysunek 1. Wykres próbek zbiorów uczącego oraz testującego po 'Subtractive Clustering'



Rysunek 2. Przebieg PSO (MaxStallIterations = 30)



Rysunek 3. Wykres próbek zbiorów uczącego oraz testującego po PSO

Liczba dobrze zakwalifikowanych przypadków (SubtractiveClustering FIS)

- set uczący: 117 - set testujący: 28

Liczba dobrze zakwalifikowanych przypadków (PSO FIS)

- set uczący: 119 - set testujący: 30

W tym przypadku po użyciu PSO otrzymaliśmy trochę lepsze wyniki - więcej zaakceptowanych próbek.

Aby lepiej prześledzić działanie naszego algorytmu implementujemy 10 iteracji dla klasyfikacji danych.

Result:

Description	SC_u	PSO_u	SC_t	PSO_t
"Iteracja: 1"	117	119	29	27
"Iteracja: 2"	116	117	28	29
"Iteracja: 3"	118	118	30	29
"Iteracja: 4"	117	119	29	28
"Iteracja: 5"	117	118	29	29
"Iteracja: 6"	118	119	29	23
"Iteracja: 7"	117	118	29	29
"Iteracja: 8"	117	118	30	28
"Iteracja: 9"	118	118	27	30
"Iteracja: 10"	117	117	30	28
"Suma"	1172	1181	290	280
"Średnie"	117.2	118.1	29	28
"Odchylenie std"	0.63246	0.73786	0.94281	1.9437

Rysunek 4.1. Wyniki otrzymane po wykonaniu klasyfikacji danych zbioru IRIS (10 iteracji).

Result:

Description	SC_u	PSO_u	SC_t	PSO_t
"Iteracja: 1"	116	118	29	30
"Iteracja: 2"	116	118	29	28
"Iteracja: 3"	115	119	30	29
"Iteracja: 4"	116	117	30	29
"Iteracja: 5"	116	116	29	29
"Iteracja: 6"	120	119	26	28
"Iteracja: 7"	118	119	29	30
"Iteracja: 8"	116	118	30	30
"Iteracja: 9"	117	118	30	29
"Iteracja: 10"	119	119	28	29
"Suma"	1169	1181	290	291
"Średnie"	116.9	118.1	29	29.1
"Odchylenie std"	1.5951	0.99443	1.2472	0.73786

Rysunek 4.2. Wyniki otrzymane po wykonaniu klasyfikacji danych zbioru IRIS (10 iteracji).

Result:

Description	SC_u	PSO_u	SC_t	PSO_t
"Iteracja: 1"	119	117	29	28
"Iteracja: 2"	117	119	29	30
"Iteracja: 3"	116	116	30	29
"Iteracja: 4"	119	117	29	29
"Iteracja: 5"	117	118	29	28
"Iteracja: 6"	117	117	30	29
"Iteracja: 7"	118	117	28	29
"Iteracja: 8"	116	117	30	29
"Iteracja: 9"	118	119	28	29
"Iteracja: 10"	115	119	30	30
"Suma"	1172	1176	292	290
"Średnie"	117.2	117.6	29.2	29
"Odchylenie std"	1.3166	1.075	0.78881	0.66667

Rysunek 4.3. Wyniki otrzymane po wykonaniu klasyfikacji danych zbioru IRIS (10 iteracji).

Po przebadaniu działania naszego algorytmu dla zbioru danych IRIS, można wywnioskować, że w większości przypadków PSO poprawia nam pewne wyniki (rys. 4.2, 4.3), ale czasem one stają się gorsze (rys. 4.1), może to być wytłumaczone nie doskonale dopasowanymi parametrami PSO albo dużym rozrzutem granic poszczególnych wejść.

- **WINE**

Zbiór danych WINE ma 13 cech, czyli przy budowaniu FIS otrzymalibyśmy 13 wejść, ale dla takich klasyfikacji to jest dużo, więc stwierdziliśmy, że wykorzystamy tylko 7 cech.

Result:

Description	SC_u	PSO_u	SC_t	PSO_t
"Iteracja: 1"	141	89	34	24
"Iteracja: 2"	142	131	29	24
"Iteracja: 3"	142	129	33	26
"Iteracja: 4"	141	129	35	29
"Iteracja: 5"	142	98	32	17
"Iteracja: 6"	142	132	27	31
"Iteracja: 7"	142	127	32	29
"Iteracja: 8"	142	132	31	31
"Iteracja: 9"	142	120	33	23
"Iteracja: 10"	141	125	34	26
"Suma"	1417	1212	320	260
"Średnie"	141.7	121.2	32	26
"Odchylenie std"	0.48305	15.186	2.4495	4.2947

Rysunek 5.1. Wyniki otrzymane po wykonaniu klasyfikacji danych zbioru WINE (10 iteracji).

Widzimy z rys 5.1, że po PSO otrzymaliśmy wiele gorsze wyniki, może to być związane z dużą ilością wejść oraz dużym rozrzutem granic poszczególnych wejść.

Result:

Description	SC_u	PSO_u	SC_t	PSO_t
"Iteracja: 1"	140	134	34	32
"Iteracja: 2"	141	127	29	30
"Iteracja: 3"	141	130	28	30
"Iteracja: 4"	142	129	32	29
"Iteracja: 5"	142	137	30	28
"Iteracja: 6"	142	133	30	27
"Iteracja: 7"	142	123	32	33
"Iteracja: 8"	141	129	30	31
"Iteracja: 9"	142	132	28	31
"Iteracja: 10"	142	132	30	32
"Suma"	1415	1306	303	303
"Średnie"	141.5	130.6	30.3	30.3
"Odchylenie std"	0.70711	3.9215	1.8886	1.8886

Rysunek 5.2. Wyniki otrzymane po wykonaniu klasyfikacji danych zbioru WINE (10 iteracji).

Wyniki z rys. 5.2 otrzymane są na innym komputerze po 15 godzinach działania, w przypadku zbioru testującego w sumie wszystkich iteracji po PSO otrzymaliśmy takie same wyniki .

- **SEEDS**

Result:

Description	SC_u	PSO_u	SC_t	PSO_t
"Iteracja: 1"	160	148	37	34
"Iteracja: 2"	157	136	39	28
"Iteracja: 3"	164	128	36	33
"Iteracja: 4"	159	109	34	22
"Iteracja: 5"	155	125	36	31
"Iteracja: 6"	153	117	36	33
"Iteracja: 7"	157	138	37	32
"Iteracja: 8"	160	141	35	32
"Iteracja: 9"	153	123	37	23
"Iteracja: 10"	150	117	38	29
"Suma"	1568	1282	365	297
"Średnie"	156.8	128.2	36.5	29.7
"Odchylenie std"	4.158	12.336	1.4337	4.2177

Rysunek 6. Wyniki otrzymane po wykonaniu klasyfikacji danych zbioru SEEDS (10 iteracji).

Klasyfikując zbiór SEEDS po PSO otrzymaliśmy wiele gorsze wyniki, tak samo jak w WINE może to być związane z dużą ilością wejść, rozrzutem granic oraz niedoskonałym dopasowaniem argumentów optymalizacji.

Wnioski

Optymalizując wektor parametrów funkcji przynależności FIS, zobaczyliśmy, że algorytm PSO czasami potrafi dać dobre wyniki, ale niekoniecznie wygra u Substractive Clustering.

Najlepiej to widać na zbiorze IRIS: liczba inputów jest 4, dlatego wektor parametrów ma w typowym przypadku 24 składowe. PSO radzi z tym, ale jest mu ciężko. Liczba dobrze zakwalifikowanych przypadków jest prawie na takim samym poziomie jak dla Substractive Clustering, ale wartość odchylenia standardowego jest większa, co świadczy o mniejszej stabilności FISu zbudowanego na podstawie wektora parametrów wyliczonego w PSO. Ale mimo tego uznaliśmy, że wyniki na takim poziomie są bardzo dobre dla PSO.

Natomiast dla SEEDS oraz WINE różnica w jakości pomiędzy tymi dwoma podejściami jest bardziej widoczna. Ale na przykład na rysunku 5.2 widać, że PSO udało się wyjść na poziom Substractive Clustering i dostarczyć identyczne wyniki.

Patrząc na te tabeli wynikowe można wywnioskować, że PSO nie jest lepszym podejściem do optymalizacji parametrów funkcji przynależności. Mamy tam dużo parametrów, a taki PSO nie lubi przestrzeni wieloparametrycznej. A drugie podejście związane z PSO polegające na klasteryzacji oraz robieniu reguł na podstawie tych klastrów jest bardziej złożone.

Ogólnie podsumowując pracę nad projektem, można powiedzieć, że PSO jest przyjemnym algorytmem i nie wymaga wysokiego poziomu wiedzy na opanowanie, ponieważ opiera się na metaheurystykę inspirowaną naturą.

Możemy powiedzieć, że wiedza otrzymana w czasie robienia projektu jest bardzo przydatna i może być zastosowana przy napotkaniu zagadnień inteligencji obliczeniowej w przyszłości.