Metody Inteligencji Obliczeniowej w Analizie Danych 2021

SPRAWOZDANIE Z LABORATORIÓW O ALGORYTMACH GENETYCZNYCH

9 czerwca 2021

Elżbieta Jowik, 298821

1 Cele

Przedmiotem rozważań w bieżącej części laboratorium było zastosowanie algorytmów genetycznych w problemach optymalizacyjnych.
W ramach pierwszych trzech etapów należało zaimplementować algorytmy genetyczne optymalizujące wartość funkcji w przestrzeni $\mathbf{R}^n$.

2 Krótkie wprowadzenie teoretyczne

Algorytmy genetyczne (genetic algorithm) należą do klasy metod ewolucyjnych proponujących alternatywne podejście do problemu optymalizacji. Dzięki ich zastosowaniu możliwe jest szybsze przeszukiwanie przestrzeni rozwiązań przy jednoczesnej minimalizacji ryzyka wpadnięcia w pułapkę minimum lokalnego. Algorytmy genetyczne zaliczane są do klasy algorytmów heurystycznych, w których potencjalne rozwiązania traktowane są jako osobniki populacji a proces poszukiwania rozwiązania odbywa się na drodze symulacji sekwencji: naturalnej selekcji, krzyżowania ze sobą osobników najsilniejszych (względem ustalonej miary) i samoistnej, losowej mutacji. Konsekwencją działania algorytmu genetycznego jest populacja najlepiej przystosowanych osobników, wśród których może znajdować się najlepsze rozwiązanie a ostateczny wynik jest w ogólności jedynie przybliżeniem najlepszego rozwiązania.
NEAT (NeuroEvolution of Augmenting Topologies) jest metodą ewaluacji (znajdowania topologii oraz wag) sztucznych sieci neuronowych za pomocą algorytmu genetycznego. Jest to metoda ze zmienną topologią, w której ewolucji podlega zarówno struktura sieci, jak i wagi połączeń. NEAT realizuje koncepcję rozpoczynania ewolucji od małych, prostych sieci i ich stopniowej kompleksyfikacji. Zgodnie z tym podejściem populacja początkowa składa się z możliwie najmniej złożonych sieci (jednakowych pod względem struktury) o losowo zainicjowanych wagach a z biegiem pokoleń dodefiniowywane są nowe neurony i połączenia.

3 Etap 1.

3.1 Opis wykonanych prac

W ramach pierwszego etapu przeprowadzonych rozważań należało zaimplementować podstawowy algorytm genetyczny z mutacją gaussowską i krzyżowaniem jednopunktowym a następnie przetestować przygotowaną implementację na funkcji $x^2 + y^2 + 2z^2$ oraz na pięciowymiarowej funkcji Rastrigina.
Na potrzeby realizacji powyższych założeń, z zachowaniem możliwości uogólnienia rozwiązania szerszej klasy problemów optymalizacji, w przygotowanej implementacji zawarto trzy niezależne klasy - dwie z nich (*Multinomial, Rastrigin*) odpowiadają optymalizowanym funkcjom celu a najbardziej złożona *GeneticAlgorithm* stanowi realizację zadanej implementacji algorytmu genetycznego. Klasy *Multinomial* i *Rastrigin* składają się z metod, które stanowią jedynie wdrożenie powszechnie dostępnych matematycznych formuł, w związku z czym dalsza część bieżącej sekcji niniejszego sprawozdania będzie zorientowana na opis struktury klasy symulującej działanie algorytmu genetycznego.
Klasa *GeneticAlgorithm* będąca krytycznym komponentem opracowanej implementacji jest parametryzowana trzema zmiennymi tożsamościowo równymi odpowiednio: funkcji celu, wymiarowi danych, liczności populacji i składa się z pięciu metod, których funkcjonalności zestawiono w poniższej tabeli:

Metody klasy *GeneticAlgorithm*	
Nazwa metody	**Opis działania**
generate_data	Metoda, która na podstawie parametryzacji obiektu bieżącej klasy generujące dane tak, aby strukturalnie odpowiadały one optymalizowanej funkcji celu.
RouletteSelection	Metoda selekcji ruletkowej przyjmująca na wejściu: populację, wektor wartości funkcji celu dla poszczególnych osobników i liczbę całkowitą k niosącą informację o liczbie osobników poddawanych jednorazowemu krzyżowaniu. W procesie selekcji wybierane są osobniki najlepiej przystosowane, które zostaną włączone do grupy rozrodczej. Dla każdego osobnika oblicza się, proporcjonalne do jego jakości, prawdopodobieństwo wejścia do grupy reprodukcyjnej a następnie z tymi niezależnymi prawdopodobieństwami losuje się indeksy osobników wejściowej populacji, które mają zostać poddane krzyżowaniu. Zwracana jest lista indeksów wylosowanych osobników.
BestSelection	Rankingowa metoda selekcji, będąca alternatywą dla selekcji ruletkowej. W metodzie tej również wybierane są osobniki najlepiej przystosowane, które zostaną włączone do grupy rozrodczej, przy czym osobniki populacji są sortowane według ich jakości: od najlepszego do najgorszego a do dalszego rozrodu przechodzi tylko k najlepiej przystosowanych osobników
OnePointCrossover	Funkcja symulująca proces generowania nowej populacji z osobników, które weszły do grupy rozrodczej. Krzyżowanie odbywa się z podanym na wejściu prawdopodobieństwem poprzez rozcięcie chromosomów rodziców w losowym punkcie *crossover_point* i przydzielenie chromosomów indeksowanych liczbami z przedziału $[0, crossover\_point)$ jednemu potomkowi, a reszty drugiemu. Obydwa osobniki są modyfikowane na miejscu a ich wynikowa długość będzie równa pierwotnej.
mutGaussian	Metoda symulująca samoistne, losowe zmiane w genotypach podanych na wejściu osobników, które mają być zmutowane. Tak jak w przypadku krzyżowania mutacja zachodzi z zadanym prawdopodobieństwem. Opisywana funkcja jest wdrożeniem mutacji gaussowskiej modyfikującej wybrane geny w ten sposób, że do jego wartości dodawana jest losowa wartość z rozkładu Gaussa o podanych na wejściu wartości oczekiwanej i wariancji.
evaluate	Metoda, która poprzez wywołanie w zadanej sekwencji wszystkich wyżej wymienionych funkcji, reprodukuje zachodzący w przyrodzie proces ewolucji. Zasada działania: 1. inicjalizacja początkowej populacji osobników, 2. poddanie każdego z nich ocenie, 3. selekcja osobników najlepiej przystosowanych do rozważanego problemu, 4. stworzenie nowego pokolenia za pomocą genetycznych operacji mutacji i krzyżowania, 5. powrót do 2 (tyle razy, ile pokoleń podano na wejściu)

Tabela 1

3.2 Otrzymane rezultaty

Przygotowaną implementację przetestowano na funkcji $x^2 + y^2 + 2z^2$ oraz na pięciowymiarowej funkcji Rastrigina. Dla zasadności wniosków dla każdej z wymienionych funkcji eksperyment powtórzono dziesięciokrotnie. Otrzymane rezultaty zagregowano i umieszczono w poniższej tabeli.

Zestawienie wyników dla przeprowadzonych eksperymentów								
Metoda selekcji	Minimum wartości funkcji	Maksimum wartości funkcji	Średnia wartość funkcji	Odchylenie standard. wartości funkcji	Minimum liczby pokoleń	Maksimum liczby pokoleń	Średnia liczba pokoleń	Odchylenie standard. liczby pokoleń
Statystyki dla problemu optymalizacji funkcji $x^2 + y^2 + 2z^2$								
Roulette	0.004344	0.077788	0.133736	0.044211	1	1.6	4	0.916515
Ranking	0.000009	0.000061	0.000231	0.000060	55	80.5	99	12.579746
Statystyki dla problemu optymalizacji funkcji Rastrigina								
Roulette	33.893293	45.131318	54.783526	7.378357	1	2.0	6	1.673320
Ranking	0.002637	0.036207	0.112761	0.037088	111	133.4	147	12.240915

3.3 Wnioski końcowe

W problemach minimalizacji obydwu funkcji rankingowa metoda selekcji statystycznie wykonywała znacznie więcej iteracji. Biorąc pod uwagę, że faktyczne minima obydwy funkcji są jednakowe i równe: $f(0, 0, ..., 0) = 0$ uzaję, że jednocześnie znajdowała ona wyraźnie lepsze przybliżenia rzeczywistych rozwiązań. Zatem zasadny wydaje się być wniosek, że selekcja ruletkowa nie sprawdza się dla problemów minimalizacji.

4 Etap 2.

4.1 Opis wykonanych prac

Założeniem prac przeprowadzonych w kolejnym tygodniu było rozwiązanie wariantu problemu znanego w literaturze jako cutting stock problem, w którym dla danego koła o promieniu r oraz zbioru dostępnych prostokątów zadanych przez trzy liczby: wysokość, szerokość i wartość należy znaleźć takie ułożenie prostokątów w kole tak, aby zmaksymalizować sumę ich wartości, przy następujących ograniczeniach:

→ boki wszystkich prostokątów mają być równoległe do osi układu,

→ wnętrza prostokątów nie powinny mieć części wspólnej (prostokąty mają na siebie nie nachodzić, ale mogą stykać się bokami),

→ każdy prostokąt można wstawić dowolnie wiele razy.

Celem zwiększenia czytelności kodu, w zastosowanym podejściu uwzględniono następujące klasy pomocnicze:

→ *Objective*, która analogicznie jak w minionym tygodniu odpowiadała funkcji celu, którą tym razem była suma wartości prostokątów.

→ *Rectangle*, czyli klasę obiektów typu prostokąt parametryzowanych przez rzędną i odciętą lewego dolnego wierzchołka oraz długość i szerokość boków.

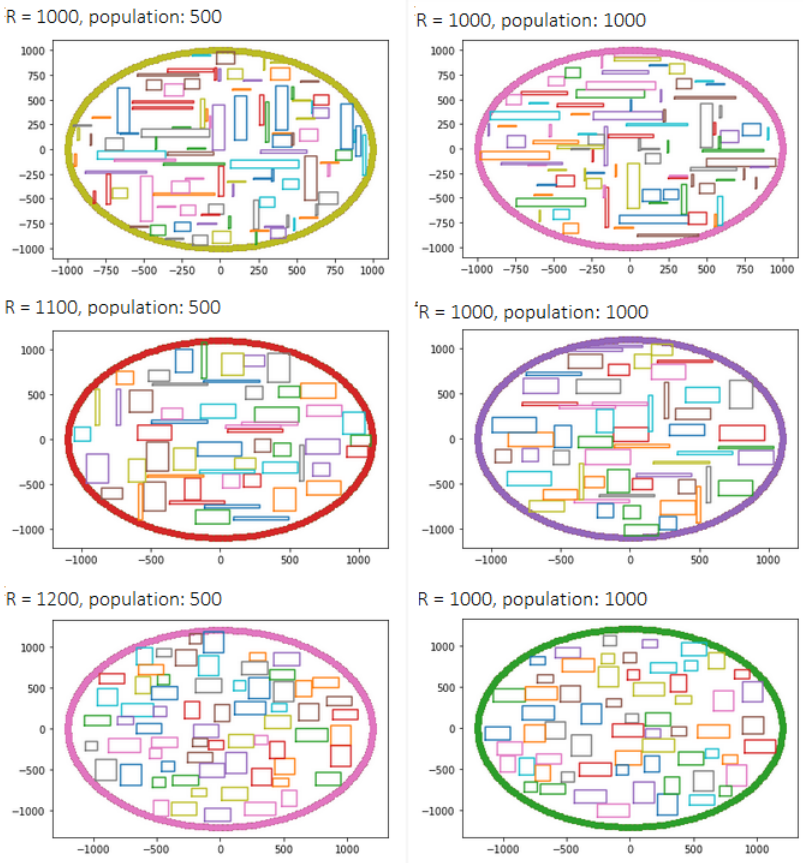
Komponentem krytycznym z punktu widzenia realizacji założeń bieżącego przyrostu była klasa *CuttingStock*, której obekty były parametryzowane przez: wielkość populacji, długość promienia koła w danym problemie oraz ścieżkę do pliku, w którym miały być przechowywane wymiary i wartości prostokątów. Szczegółowe specyfikacje metod związanych tą klasą umieszczono w poniższej tabeli:

Metody klasy *CuttingStock*	
Nazwa metody	**Opis działania**
do_overlap	Metoda, która na podstawie parametrów dwóch podanych na wejściu obiektów klasy *Rectangle* rozstrzyga, czy prostokąty mają niepuste przecięcie. Zwraca *FALSE* jeśli zadane obiekty są rozłączne lub stykają się krawędziami oraz *TRUE* w przeciwnym przypadku.
check_constraints	Funkcja odpowiedzialna za zbadanie spełnienia geometrycznych ograniczeń problemów, która z wykorzystaniem funkcji *do_overlap* bada czy w bieżącym rozwiązaniu wnętrza prostokątów nie mają części wspólnej oraz dodatkowo czy każdy prostokąt zawiera się w kole o zadanym promieniu. Badanie spełnienia drugiego z wymienionych ograniczeń odbywa się na podstawie obliczenia euklidesowej odległości współrzędnych wierzchołków od środka koła, który arbitralnie umieszczono w punkcie $(0, 0)$ układu współrzędnych.
generate_random_rectangle	Działanie funkcji polega na generowaniu obiektów klasy *Rectangle* i sprawdzeniu czy dodanie do osobnika danego prostokąta nie spowoduje złamania przez niego geometrycznych ograniczeń zadania. Ulokowanie lewego dolnego wierzchołka nowego prostokąta odbywa się na drodze losowania punktu z koła. Działanie funkcji jest przerywane w momencie, gdy potencjalny nowy osobnik nie spełnia założeń zadania. Wówczas zwracana jest dwuelementowa lista, której składowymi są obiekt klasy prostokąt parametryzowany przez wylosowane koordynaty i zczytane z pliku wymiary oraz, również zczytana pliku, wartość wylosowanego prostokąta.
generate_population	Funkcja, która docelowo ma być wykorzystywana do inicjalizacji i rozszerzania populacji na drodze umiejscowiania w bieżącym osobniku (tj. kole w jakimś stopniu wypełnionego prostokątami) prostokątów losowo wybieranych z pliku. Immanentnym komponentem tej funkcji jest metoda *generate_random_rectangle*.
evaluate	Metoda reprodukujący zachodzący w przyrodzie proces ewolucji. Zasada działania: 1. wylosowanie, z utworzonej uprzednio populacji początkowej, osobników macierzystych, 1. stworzenie jednostek potomnych na drodze, przeprowadzanych z odpowiednimi prawdopodobieństwami, genetycznych operacji krzyżowania i mutacji (w której losowe prostokąty osobnika macierzystego zastępowano nowymi, stworzonymi za pomocą funkcji *generate_random_rectangle*), 2. selekcja, tych osobników, które są najlepiej przystosowane z punktu widzenia rozważanego problemu, 5. stworzenie nowej populacji na drodze nadpisania krzyżowanych osobników populacji wyjściowej i powrót do 2 (tyle razy, ile pokoleń podano na wejściu)
calculate	Funkcja obliczająca sumę wartości prostokątów dla zbioru prostokątów stworzona na potrzeby wizualizacji.

Tabela 2

4.2 Otrzymane rezultaty

Rozwiązanie przetestowano za dostarczonych danych, takich, że w nazwie danego pliku w podany był promień koła a szerokość, wysokość i wartość protokąta były zapisane odpowiednio w jego pierwszej, drugiej i trzeciej kolumnie. Poniżej umieszczono wizualizacje otrzymanych rezultatów dla kół o promieniach kolejno: $1000, 1100, 1200$ oraz populacji liczności odpowiednio: 500 i 1000.



Rysunek 1

4.3 Wnioski końcowe

Ze zróżnicowania wyników dla poszczególnych parametryzacji można wywnioskować, że w istocie rezultatem działania algorytmu genetycznego jest populacja najlepiej przystosowanych osobników, wśród których może znajdować się najlepsze rozwiązanie a ostateczny wynik jest w ogólności jedynie przybliżeniem najlepszego rozwiązania.

5 Etap 3.

5.1 Opis wykonanych prac

Celem realizowanym w trzeciej fazie rozważań była optymalizacja wag w sieci MLP z użyciem algorytmu genetycznego, tzn. implementacja prostego uczenia MLP z użyciem algorytmu genetycznego z zastosowaniem standardowych operatorów krzyżowania i mutacji. Na wejściu należało przyjąć strukturę sieci neuronowej i dane uczące a za optymalizowaną funkcję przekształcenie wektora wag sieci na błąd na zbiorze uczącym.

Zaimplementowane klasy pomocnicze:
→ *Activations* - zawarte w niej metody odpowiadały dopuszczalnym funkcjom aktywacji,
→ *Transformations* służąca wzajemnym przekształceniom macierzy i wektorów.

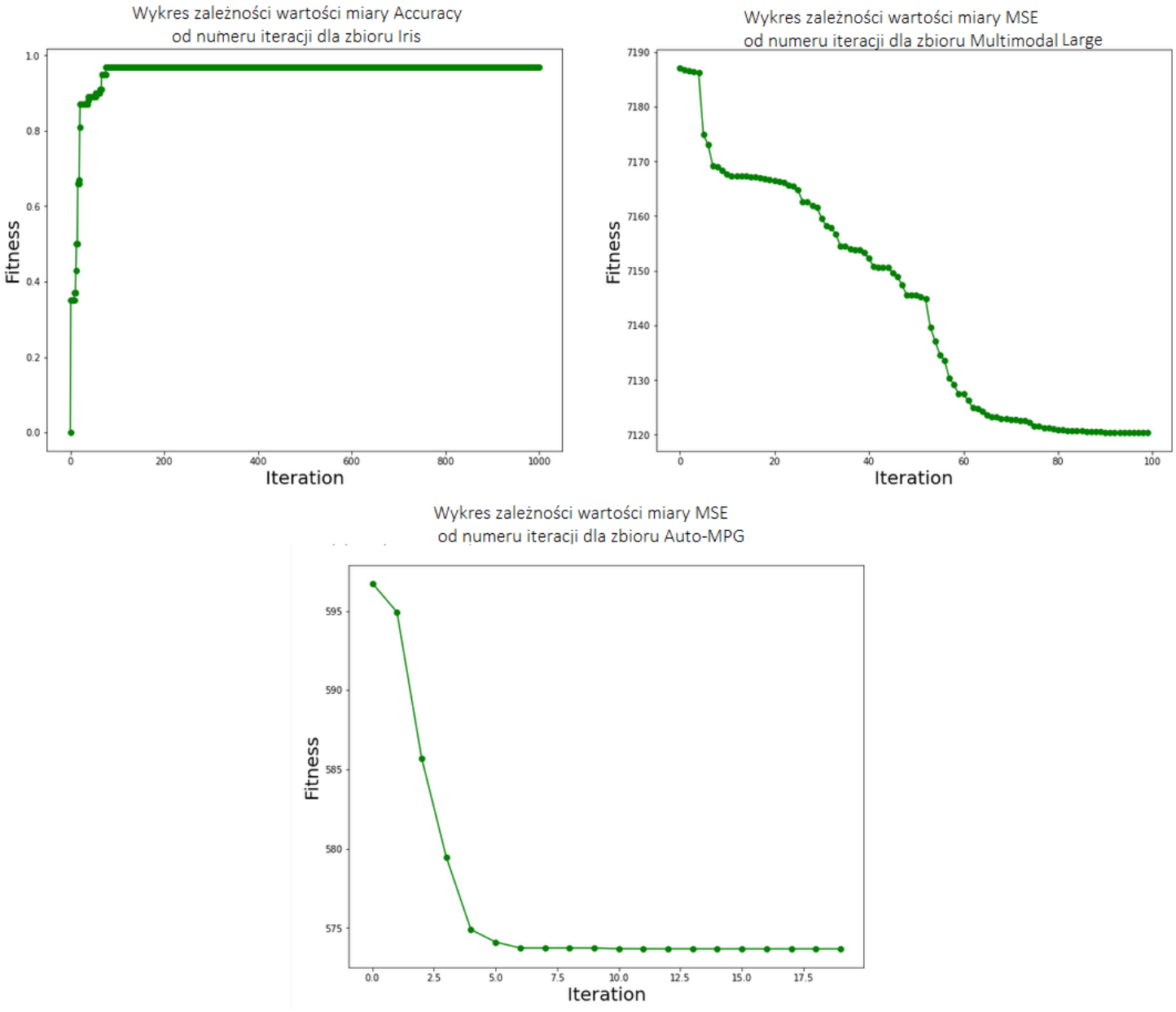
Założeniem towarzyszącym procesowi implementacji, podyktowanym naturą zbiorów testowych, było, że docelowa sieć ma być stosowana zarówno w problemach klasyfikacji jak i regresji. Podstawą implementacji sieć neuronowej zamkniętej w klasie *MLP_with_GA* była sieć neuronowa typu MLP zaimplementowana w pierwszej części semestru. Obiekty tej klasy są parametryzowane przez: liczbę neuronów w każdej z warstw, flagę, pozwalającą na przekazanie sieci informacji czy przedmiotem rozważań będzie problem klasyfikacyjny czy regresyjny a także prawdopodobieństwo zajścia mutacji oraz trzy liczby całkowite - odpowiadające rozmiarowi populacji, liczbie pokoleń, liczbie rodziców, którzy ulegają krzyżowaniu w procesie ewolucji. W szczególności liczby neuronów w kolejnych warstwach miały być podawane w formie tablicy, której długość powinna odpowiadać liczbie warstw w żądanej architekturze sieci, pierwszy element musiał być tożsamy z wymiarem danych wejsciowych, a ostatni z oczekiwaną liczbą wyjść sieci. Oprócz wymienionych atrybutów, obiekty klasy będącej krytycznym komponentem opracowanej implementacji zostały "wyposażone" w 7 metod, których funkcjonalności zestawiono w poniższej tabeli.

Metody klasy *MLP_with_GA*	
Nazwa metody	**Opis działania**
initialize_weights	Funkcja, która dla architektury zadanej przez podaną na wejściu listę liczności neuronów w poszczególnych warstwach, inicjalizuje losowe wagi połączeń pochodzące z rozkładu jednostajnego.
selection	Implementacja rankingowej metody selekcji, której założenia zostały opisane w sekcji 4. niniejszego sprawozdania.
crossover	Realizacja krzyżowania jednopunktowego, w której punkt rozcięcia chromosomów osobników macierzystych nie jest całkowicie losowy, ale podyktowany ilości rodziców poddawanych krzyżowaniu.
mutation	Metoda mutacji, która z określonym na etapie inicjalizacji obiektu prawdopodobieństwem, modyfikuje wybrany losowo pojedynczy gen osobnika potomnego, poprzez dodanie do jego wartości liczby z rozkładu jednostajnego na przedziale $[-1, 1]$
fit	Funkcja zwracająca liczbę, będącą oceną jakości przystosowania osobnika. Przystosowanie to związane jest z jakością danego rozwiązania. W zależności od tego czy rozważany jest problem klasyfikacji czy regresji, za funkcję przystosowania przyjmuje się odpowiednio *Accuracy* lub *MSE*
predict	Funkcja przyjmuje na wejściu wagi (pojedynczego osobnika), dane wejściowe i wyjścowe oraz opcjonalny parametr określający, której funkcji aktywacji należy użyć. Zwraca, obliczoną na podstawie danych wejściowych i wag predykcję oraz odpowiadającą jej wartość funkcji dopasowania.
train	Metoda, która poprzez wywołanie w zadanej sekwencji wszystkich wyżej wymienionych funkcji, reprodukuje zachodzący w przyrodzie proces ewolucji. Sekwencja działania: 1. inicjalizacja wag początkowych → 2. ocena przystosowania → 3. stworzenie nowego pokolenia za pomocą genetycznych operacji selekcji, mutacji i krzyżowania → 4. powrót do 2 (tyle razy, ile pokoleń podano na wejściu)

Tabela 3

5.2 Otrzymane rezultaty

Rozwiązanie przetestowano za dostarczonych danych, otrzymane rezultaty,mierzone względem arbitralnie dobranych miar dopasowania, zakregowano a wizualizacje ich zmienności umieszczono poniżej.



Rysunek 2

5.3 Wnioski końcowe

W procesie optymalizacji wag w sieci MLP z użyciem algorytmu genetycznego, tempo poprawy wyników z iteracji na iterację wydaje się być odwrotnie proporcjonalna do wolumenu danych w zbiorze. Dla najbardziej złożonego zbioru *Mulimodal Large* zmiany wartości funkcji przystosowania są znikome, podczas gdy dla pozostałych zbiorów można zaobserwować ich gwałtowniejsze skoki.