POP - Projekt

Sushko Nikita, Będkowski Patryk

Semestr zimowy 2023

1 Opis projektu

1.1 Temat

Optymalizacja atrybutów dla zadania klasyfikacji binarnej.

Dla problemu klasyfikacji binarnej zbioru o ogromnej liczbie atrybutów (np. Dorothea lub Farm ads) zaproponuj algorytm heurystyczny który wybierał będzie możliwie najlepszy podzbiór atrybutów które poprawią wyniki klasyfikacji. Jako klasyfikatora użyj dowolnego prostego algorytmu (kNN, regresja liniowa).

1.2 Opis projektu

Celem projektu jest porównanie algorytmów wybierających najlepszy podzbiór atrybutów poprawiających wyniki dla problemu klasyfikacji binarnej.
Zdecydowaliśmy się wybrać i zaimplementować następujące algorytmy heurystyczne do wyboru najlepszego podzbioru atrybutów:

- Algorytmy:
  - Sequential Forward Selection (SFS)
  - Simulated Annealing (SA)
  - Genetic Algorithm (GA)

2 Dane wejściowe

2.1 Opis danych wejściowych

Zdecydowaliśmy użyć zbioru **Farm Ads** dla porównania skuteczności wymienionych wyżej algorytmów heurystycznych.

Zbiór danych **Farm Ads** zawiera informacje z tekstowych reklam pochodzących z dwunastu witryn, które dotyczą różnych aspektów związanych z hodowlą zwierząt gospodarskich. Dane te są zebrane zarówno z treści reklam, jak i z witryn docelowych reklam. Etykiety binarne wskazują, czy reklama została zatwierdzona czy odrzucona przez właściciela.

W ramach naszego projektu wykorzystamy reprezentację bag-of-words, która jest dostępna w formie wektorów rzadkich SVMLight. W tej reprezentacji pierwsza wartość to etykieta, a następnie każdy niezerowy atrybut jest zakodowany jako indeks:wartość.

Liczba przykładów: **4143**.
Liczba atrybutów: **54877**.

3 Opis zadania

3.1 Zadanie klasyfikacji

Problem klasyfikacji dotyczy zbioru danych składający się z par (x,y), gdzie x to wektor cech opisujący obserwację, a y to przypisana do niej klasa. Naszym celem jest znalezienie funkcji $f(x)$, która przypisuje nowe obserwacje do określonych klas na podstawie ich cech.

Rozważamy problem uczenia modelu klasyfikacji binarnej na zbiorze danych $D = (x_i, y_i)_{i=1}^n$, gdzie $x_i \in R^d$, a $y_i \in C_1, C_2$ są odpowiednio zmiennymi niezależnymi i zależną.

Naszym celem jest znalezienie funkcji $y_n \approx f(x_n)$ najlepiej odwzorowującej zależność między zmiennymi wejściowymi a zmienną wyjściową. Do rozwiązania tego zadania posłużymy się algorytmem k-najbliższych sąsiadów (kNN).

3.2 K-najbliższych sąsiadów

K-najbliższych sąsiadów (kNN) to prosty i intuicyjny algorytm klasyfikacji, który może być efektywny zwłaszcza w przypadku prostych problemów binarnej klasyfikacji. W tym kontekście zakładamy, że każdy obiekt w zbiorze danych należy do jednej z dwóch klas, oznaczanych jako C_0 i C_1. Celem algorytmu jest przewidzenie klasy nowego obiektu na podstawie klasy większości spośród jego k najbliższych sąsiadów.

Możemy zdefiniować funkcję celu:
Dla zadania klasyfikacji binarnej, funkcja celu w przypadku kNN może być zdefiniowana jako minimalizacja błędu klasyfikacji:

$$J(w) = \sum_{i=1}^{n} I(y_i \neq C(x_i))$$

gdzie:

- w to parametry modelu (w przypadku kNN jest to liczba sąsiadów k),
- $C(x_i)$ to przypisana klasa obiektu x_i na podstawie k najbliższych sąsiadów,
- $I(.)$ to funkcja wskaźnikowa (1, jeśli obserwacja jest źle sklasyfikowana, 0 w przeciwnym razie).

Celem jest minimalizacja liczby błędnie sklasyfikowanych obserwacji.

4 Opis algorytmów i eksperymentów

4.1 Sequential Forward Selection (SFS)

SFS zaczyna od pustego zbioru atrybutów. W każdej kolejnej iteracji dodaje najlepsze atrybuty spośród pozostałych.

Algorithm 1 Sequential Forward Selection

$X_0 \leftarrow \emptyset$
$k \leftarrow 0$
$p \leftarrow$ final number of features
while $k < p$ **do**
  $x^+ \leftarrow argmax(J(X_k + x)), x \in Y - X_k$
  $X_{k+1} \leftarrow X_k + x^+$
  $k \leftarrow k + 1$
end while

SFS to prosty, zachłanny algorytm przeszukujący całą przestrzeń rozwiązań i który nie wymaga dostosowywania parametrów. Postanowiliśmy wykorzystać go jako punkt odniesienia do oceny efektywności bardziej zaawansowanych algorytmów przedstawionych niżej.

4.2 Simulated Annealing (SA)

Reprezentacja rozwiązania: wektor binarny. 1 na pozycji *i* oznacza, że $atrybut_i$ jest częścią rozwiązania. Natomiast, 0 oznacza, że nie należy do rozwiązania.

Algorithm 2 Simulated Annealing

$T \leftarrow T_{max}$
$x \leftarrow$ initial solution
$E \leftarrow E(x)$
while $(T > T_{min})$ and $(E > E_{th})$ **do**
  $x \leftarrow$ new candidate solution
  $E_{new} \leftarrow E(x_{new})$
  $\Delta E \leftarrow E_{new} - E$
  if $Accept(\Delta E, T)$ **then**
    $x \leftarrow x_{new}$
    $E \leftarrow E_{new}$
  end if
  $T \leftarrow T/\alpha$
end while
return x

Funkcja akceptacji nowego rozwiązania:

Algorithm 3 Accept

Input: T - temperature, ΔE - the energy variation between the new candidate solution and the current one
Output: Boolean value, indicating whether the new solution is accepted or rejected

if $(\Delta E < 0)$ **then**
  return $True$
else
  $r \leftarrow$ random value in the range [0,1)
  return $(r < \exp(-\Delta E/T))$
end if

Jeśli chodzi o eksperymenty z algorytmem Simulated Annealing, to skupimy się głównie na nastawieniu parametrów temperatury (T) oraz zmiany temperatury (α). Dzięki temu będziemy w stanie uzyskać równowagę między eksploracją a eksploatacją przestrzeni danych. Dodatkowo, w trakcie eksperymentów istotny będzie parametr T_{min}, reprezentujący minimalną temperaturę. Sterując tym parametrem będziemy w stanie kontrolować zbieżność algorytmu oraz, co za tym idzie, liczbę wybieranych atrybutów.

4.3 Genetic Algorithm (GA)

Reprezentacja rozwiązania: taka sama jak w algorytmie SA.

Algorithm 4 Genetic Algorithm

$P \leftarrow$ initialize population
$t \leftarrow 0$
while $(t < t_{max})$ **do**
  $P_{new} \leftarrow select(P)$
  $crossover(P_{new})$
  $mutate(P_{new})$
  $evaluate(P_{new})$
  $P \leftarrow P_{new}$
  $t \leftarrow t + 1$
end while
return P

Algorytm genetyczny daje dużą przestrzeń dla eksperymentów (wybór odpowiedniej selekcji, krzyżowania...). Planujemy rozpocząć od algorytmu, który korzysta z selekcji elitarnej, mutacji bitowej, krzyżowania jednorodnego (uniform crossover). Ale,

oczywiście, konieczne będzie przeprowadzenie eksperymentów na konkretnym zbiorze danych w celu wybrania najlepszych parametrów, np:

- rozmiar selekcji elitarnej
- selekcja elitarna vs selekcja turniejowa
- mutacja jednopunktowa vs wielopunktowa (większa eksploracja danych)
- sterowanie rozmiarem populacji

5 Metryki klasyfikacji

W zadaniu posługujemy się następującymi metrykami do oceny jakości modeli klasyfikacji:

Dokładność (Accuracy):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Współczynnik F1 Score:

$$F1 \text{ Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

gdzie:

- TP (True Positives) - liczba przypadków, w których model poprawnie zidentyfikował pozytywną klasę.
- FP (False Positives) - liczba przypadków, w których model błędnie zidentyfikował negatywną klasę jako pozytywną.
- FN (False Negatives) - liczba przypadków, w których model błędnie zidentyfikował pozytywną klasę jako negatywną.
- TN (True Negatives) - liczba przypadków, w których model poprawnie zidentyfikował negatywną klasę.

6 Metodologia eksperymentów

Wykonane eksperymenty będą przedstawiać porównanie wyników wyżej opisanych metryk dla zadania klasyfikacji przy użyciu zbiorów danych utworzonych przez metody doboru N najlepszych atrybutów. Zostaną przedstawione 2 wykresy, dla wszystkich metryk. Każdy wykres będzie przedstawiał stosunek wartości metryki (oś OY) do ilości wybranych atrybutów (oś OX). Dodatkowo każdy wykres będzie posiadał 3 krzywe odpowiadające metodom. Testowane ilości atrybutów zostaną doprecyzowane później (mogą to być wartości: 2, 4, 8, 16, 32, 64, 128).

Z reguły na testowanie również niedeterministycznych algorytmów, na wykresach zostaną przedstawione również średnie wartości i odchylenia standardowe metryk.

Ponadto wybrany zbiór danych *Farm Ads* posiada znaczną ilość atrybutów równą *54877*. Przeprowadzanie obliczeń na kompletnym zbiorze danych bez znacznych zasobów obliczeniowych będzie niemożliwym do spełnienia zadaniem w określonym czasie. Zatem postanowimy wybrać podzbiór zbiory danych, na którym będziemy przeprowadzać eksperymenty.

7 Narzędzia i biblioteki

7.1 Środowisko

Kod zostanie przez nas zaimplementowany w języku Python3 i testowany na wersjach 3.11.3. Do prowadzenia eksperymentów wykorzystaliśmy również notatnik Jupyter. Korzystaliśmy ze środowisk Microsoft Visual Studio Code IDE oraz PyCharm IDE. Najważniejsze biblioteki użyte w projekcie to:

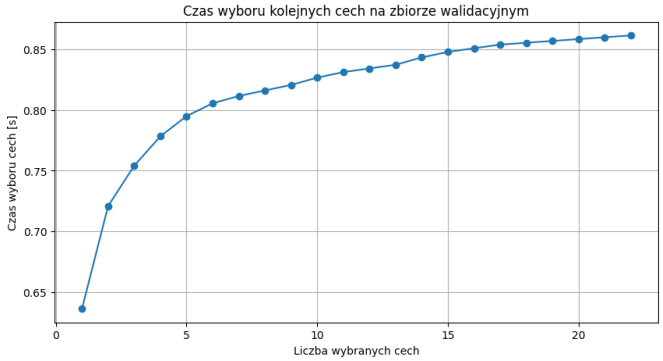
- NumPy - zarządzanie i przetwarzanie zbiorów danych
- Pandas - Wczytywanie zbiorów danych
- SkiKit Learn - Pobieranie zbiorów danych oraz implementacja metryk funkcji kosztów
- Matplotlib - Wizualizacja wyników iteracji algorytmów uczenia

8 Testy

W tej sekcji przedstawione zostaną wyniki przeprowadzonych eksperymentów dotyczących wybranych algorytmów. Dokonana zostanie także szczegółowa analiza istotnych obserwacji i wniosków wynikających z charakterystyki oraz zachowania omawianych algorytmów. Przeprowadzona zostanie również optymalizacja parametrów algorytmów (które tego wymagają) osiągających najwyższe wartości wykorzystywanych metryk.

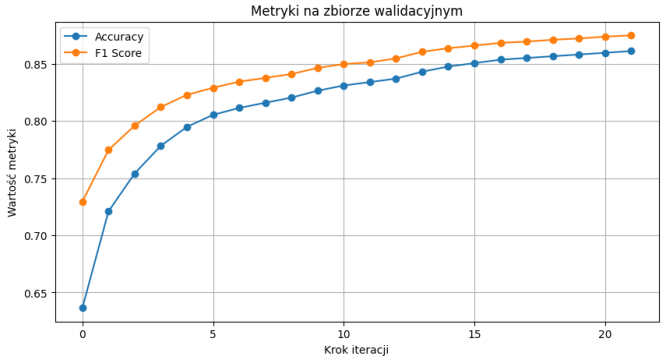
8.1 Sekwencyjny Dobór Atrybutów Do Przodu

Z reguły na naiwny charakter algorytmu sterowanie jego metrykami opiera się na wyborze wartości parametru ilości cech, którą chcemy uzyskać. Algorytm iteruje po wszystkie dostępnych atrybutach w zbiorze danych, poddaje je zadaniu klasyfikacji i ocenia czy dana wybrana cecha, polepsza wyniki metryk. Jako kryterium sukcesu w doborze kolejnej cechy do zbioru atrybutów wybraliśmy, aby obie metryki klasyfikacji (accuracy i f1 score) osiągnęły wyższy wynik niż z poprzednim zbiorem.



Rysunek 1: Czas wyboru kolejnych cech przez SFS.

Czas wyboru kolejnych cech podczas działania algorytmu SFS jest rosnący zgodnie z przewidywaniami (Rysunek 1.). W kolejnych iteracjach, algorytm musi iterować po większej ilości atrybutów, aby wybrać tę wartość, która podnosi wartość obu metryk.



Rysunek 2: Wartość metryk na zbiorze walidacyjnym dla wybranych cech.

Przetestowano jakość metryk na zbiorze walidacyjnym dla odpowiednio wybranych cech w kolejnych iteracjach. Stwierdzamy, że dodawanie kolejnych cech do zbioru atrybutów polepsza wyniki obu metryk względem wybranego kryterium wyboru. Wzrost metryk jest nierównomierny. Podczas początkowej fazy doboru kolejnych atrybutów następuje gwałtowny wzrost metryk. Dodawanie kolejnych najlepszych atrybutów stabilizuje jakość wyników.

8.2 Symulowane wyżarzanie

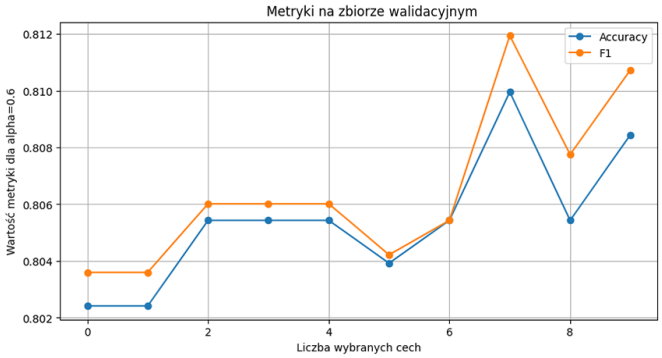
Zaimplementowaliśmy kilka modyfikacji obecnie iterowanego zbioru atrybutów, które polegają na:

- dodaniu losowego atrybutu
- usunięciu losowego atrybutu

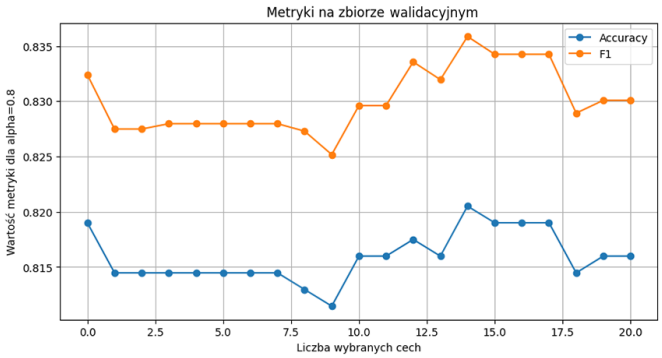
- zamiany losowego atrybutu z obecnego zbioru z atrybutem będącym w zbiorze dostępnych atrybutów

Wybór poszczególnych operacji jest losowy.
Jako kryterium stopu, oprócz standardowego polegającego na osiągnięciu temperatury niższej bądź równej ustalonej minimalnej, wybraliśmy maksymalną ilość iteracji, którą wykona algorytm.

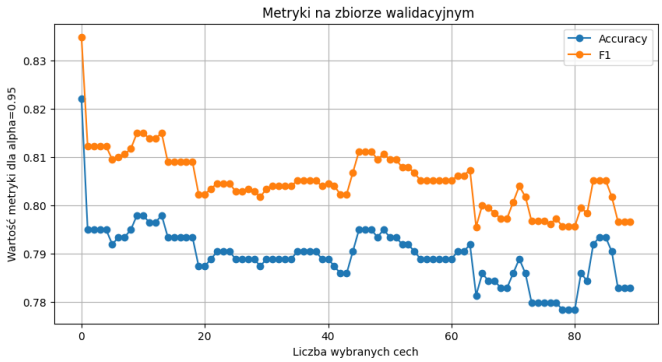
8.2.1 Wpływ parametru α na rozwiązanie



Rysunek 3: Wartość metryk dla parametru α=0.6



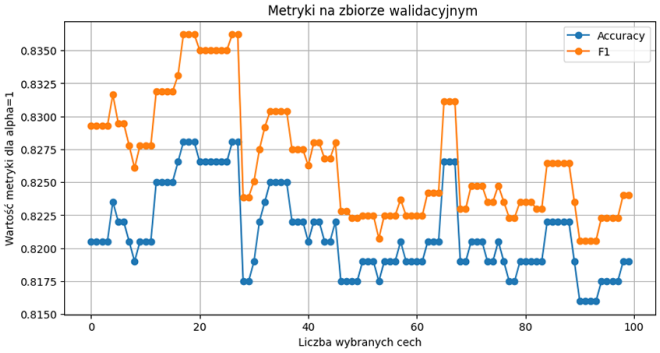
Rysunek 4: Wartość metryk dla parametru α=0.8



Rysunek 5: Wartość metryk dla parametru α=0.95

Powyższe wykresy przedstawiają wpływ parametru α na ilość iteracji algorytmu.
Wysokie wartości α oznaczają powolne obniżanie temperatury. Algorytm będzie dłużej eksplorował przestrzeń rozwiązań i mniej prawdopodobne jest, że szybko utknie w minimum lokalnym. To może prowadzić do lepszego rozwiązania końcowego, ale zwiększa czas działania algorytmu.

Niższe wartości α prowadzą do szybkiego obniżania temperatury. Algorytm szybciej skoncentruje się na lokalnym przeszukiwaniu przestrzeni rozwiązań, co może prowadzić do szybszego znalezienia rozwiązania, ale zwiększa ryzyko utknięcia w minimum lokalnym i może skutkować gorszym rozwiązaniem końcowym.

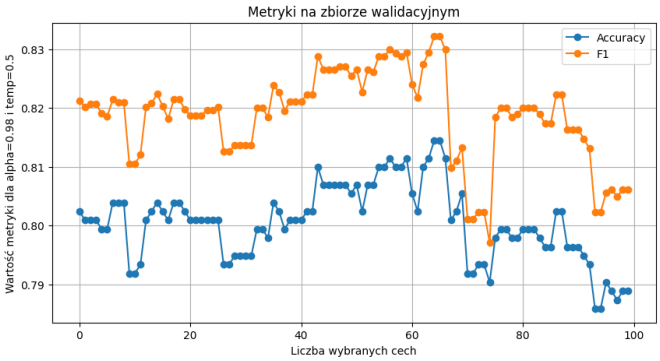


Rysunek 6: Wartość metryk dla parametru α=1

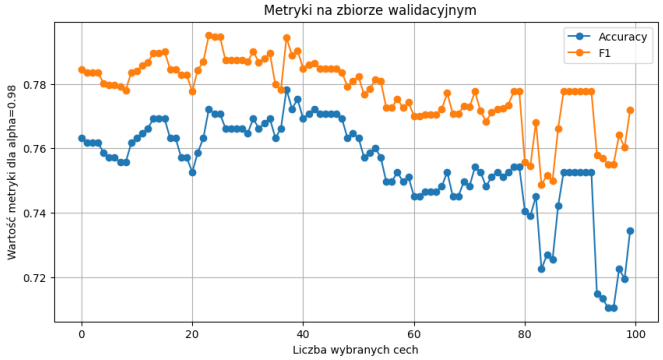
Jeśli parametr α w algorytmie symulowanego wyżarzania będzie miał wartość 1, oznacza to, że temperatura w systemie nie będzie się zmniejszać pomiędzy kolejnymi iteracjami algorytmu.
Algorytm stale będzie zachowywać swoją początkową zdolność do eksploracji przestrzeni rozwiązań bez koncentrowania się na optymalizacji lokalnej. W takim przypadku algorytm może nigdy nie zbiegać do rozwiązania, ponieważ stale akceptuje ruchy, które pogarszają funkcję celu z takim samym prawdopodobieństwem jak na początku procesu wyżarzania. Jest to równoznaczne z przeszukiwaniem losowym bez konkretnego kierunku w kierunku minimum.
Utrzymanie stałej temperatury znosi główną zaletę symulowanego wyżarzania, czyli zdolność do "ochładzania"rozwiązania i skupiania się na jego stopniowym poprawianiu.

8.2.2 Wpływ parametru temperatury na rozwiązanie

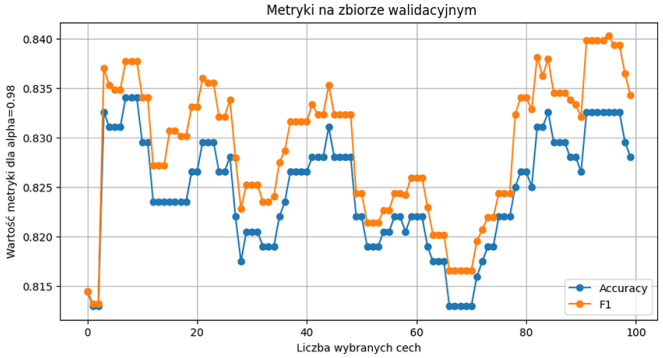
Początkowa i minimalna temperatura w algorytmie symulowanego wyżarzania są kluczowymi parametrami, które wpływają na jego zachowanie i skuteczność w poszukiwaniu optymalnego rozwiązania. Dla wszystkich poniższych eksperymentów jako minimalną temperaturę określającą koniec iteracji ustalono wartość 0.01. Gdy obecna temperatura jest niższa niż minimalna bądź gdy przekroczono maksymalną ilość iteracji, algorytm kończy działanie.
Wybrano wartość parametru α na 0.98 co oznacza, że temperatura będzie obniżana powoli. Umożliwia to dokładniejsze przeszukiwanie przestrzeni rozwiązań na każdym etapie.



Rysunek 7: Wartość metryk dla parametru α=0.98 oraz $temp = 0.5$

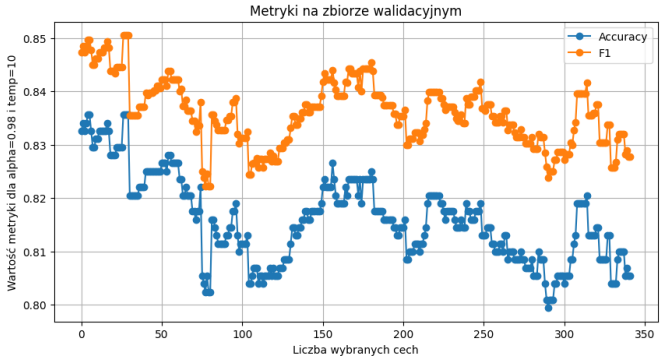


Rysunek 8: Wartość metryk dla parametru α=0.98 oraz $temp = 1$



Rysunek 9: Wartość metryk dla parametru α=0.98 oraz $temp = 5$

Zbyt niska początkowa temperatura może spowodować, że algorytm szybko zbiegnie do najbliższego minimum lokalnego bez odpowiedniej eksploracji przestrzeni rozwiązań, co może skutkować znalezieniem suboptymalnego rozwiązania.
Wysoka początkowa temperatura przedstawiona na rysunku 9 zwiększa prawdopodobieństwo akceptacji gorszych rozwiązań na początku procesu wyżarzania, co pozwala algorytmowi na eksplorację szerokiego zakresu przestrzeni rozwiązań i pomaga uniknąć utknięcia w minimum lokalnym.



Rysunek 10: Wartość metryk dla parametru α=0.98 oraz $temp = 10$

Jeśli początkowa temperatura w algorytmie symulowanego wyżarzania jest ustawiona na bardzo wysoki poziom (Rysunek 10) w stosunku do temperatury minimalnej, spowoduje to dłuższy czas dojścia do optymalnego rozwiązania.

8.2.3 Wybór najlepszych parametrów.

Bazując na powyższych testach stwierdzamy, iż wartość parametru α równa 0.98 . Taki wybór współczynnika pozwala na skuteczne zbalansowanie procesu eksploracji przestrzeni rozwiązań z koniecznością osiągnięcia konwergencji algorytmu. Dzięki temu, algorytm zachowuje większą elastyczność w początkowych fazach poszukiwań, co zwiększa prawdopodobieństwo znalezienia optymalnego globalnie rozwiązania.

Z kolei ustalenie wartości początkowej temperatury na 5, przy minimalnej wartości 0.01, pozwala na wystarczającą eksplorację przestrzeni rozwiązań, jednocześnie zapewniając efektywne przechodzenie do fazy eksploracji poszukiwań do fazy eksploatacji w miarę zbliżania się do optymalnego rozwiązania.

8.3 Algorytm genetyczny

Zaimplementowaliśmy kilka różnych wersji algorytmu genetycznego, uwzględniając 2 rodzaje selekcji (turniejową i ruletkową), 2 rodzaje krzyżowania (jednopunktowe i jednorodne) oraz 2 rodzaje mutacji (jednopunktową i wielopunktową mutację "bitflip"). W celu porównania efektywności różnych kombinacji algorytmu, postanowiliśmy skorzystać z podzbioru oryginalnego zbioru danych o rozmiarze 300 atrybutów. Ten krok pozwoli ocenić wydajność różnych wariantów algorytmu genetycznego i wybrać najbardziej obiecującą kombinację do dalszych testów.

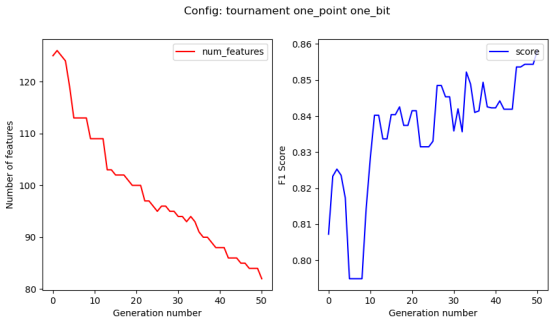
Aby zachęcić algorytm do generowania rozwiązań o mniejszej liczbie atrybutów, zdecydowaliśmy się na zastosowanie zmodyfikowanej funkcji celu do oceny jakości rozwiązań. Funkcja ta jest przedstawiona poniższym wzorem:

$$f_{new} = \alpha * (1 - f_{orig}) + (1 - \alpha) * numfeatures$$

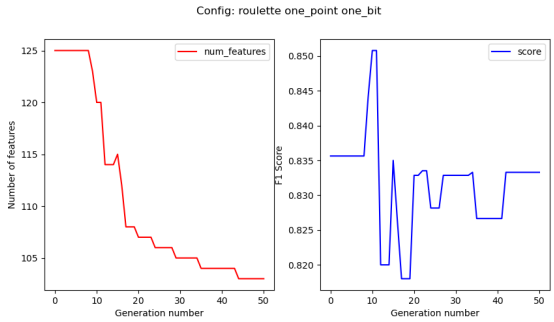
gdzie $\alpha = 0.99$, $numfeatures$ - liczba atrybutów wziętych do rozwiązania, f_{orig} - F1 Score/Accuracy.

Więc, algorytm generyczny ma za zadanie **zminimalizowanie** tak zdefiniowanej funkcji celu.

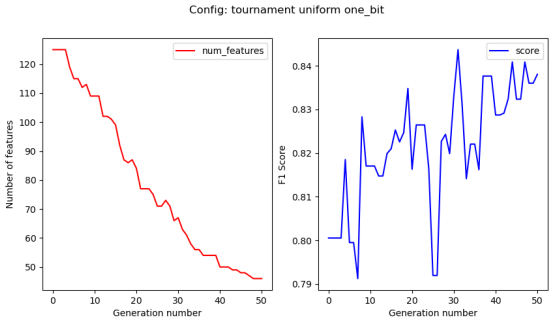
Parametry podczas testowania: rozmiar populacji: 50, liczba iteracji: 50, prawdopodobieństwo krzyżowania: 0.8, prawdopodobieństwo mutacji: 0.5 (one bit) i 0.05 (multi bit), rozmiar elity: 3. Również, dla wszystkich uruchomień w tym teście ustawione zostało ziarno generatora (np.random.seed = 42).



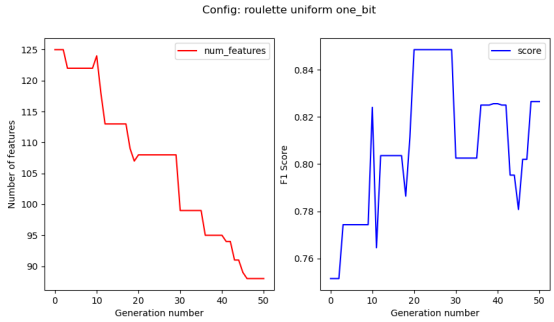
(a) Selekcja turniejowa, krzyżowanie jednopunktowe, mutacja jednopunktowa



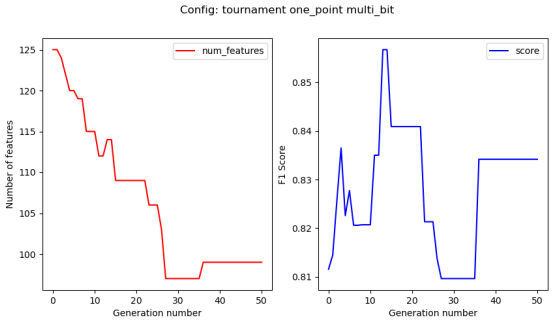
(b) Selekcja ruletkowa, krzyżowanie jednopunktowe, mutacja jednopunktowa



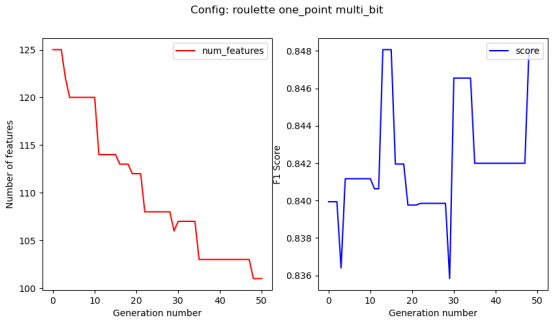
(c) Selekcja turniejowa, krzyżowanie jednorodne, mutacja jednopunktowa



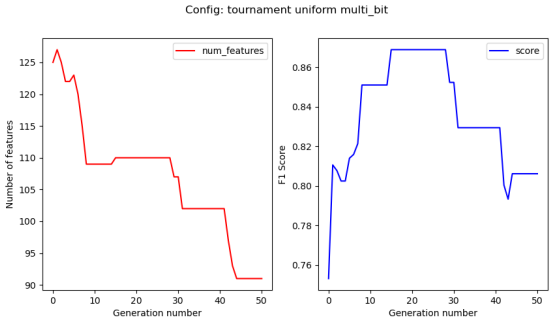
(d) Selekcja ruletkowa, krzyżowanie jednorodne, mutacja jednopunktowa



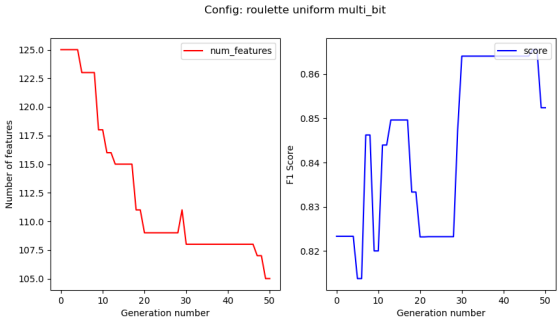
(e) Selekcja turniejowa, krzyżowanie jednopunktowe, mutacja wielopunktowa



(f) Selekcja ruletkowa, krzyżowanie jednopunktowe, mutacja wielopunktowa



(g) Selekcja turniejowa, krzyżowanie jednorodne, mutacja wielopunktowa



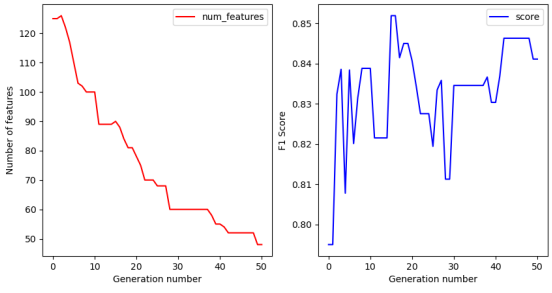
(h) Selekcja ruletkowa, krzyżowanie jednorodne, mutacja wielopunktowa

Rysunek 11: Porównanie różnych kombinacji algorytmu genetycznego

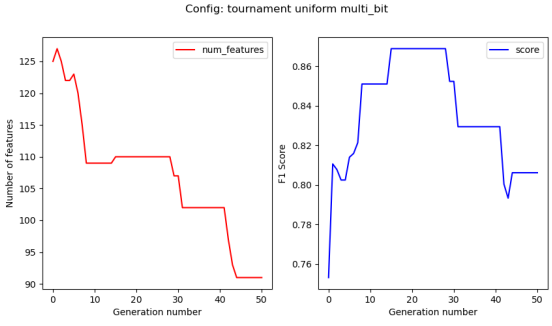
Jak widać z powyższych wykresów, najlepszą konfiguracją jest algorytm genetyczny z selekcją turniejową, krzyżowaniem jednorodnym oraz mutacją jednopunktową.

Kombinacje z mutacją wielopunktową wykazują nieco słabsze wyniki, co prawdopodobnie związane jest z redukcją liczby cech wziętych do rozwiązania. W trakcie mutacji algorytm preferuje wybieranie większej liczby cech spoza rozwiązania (co może skutkować zbyt dużą eksploracją). W celu zweryfikowania tej hipotezy, przeprowadziliśmy dwa testy dla najlepszej konfiguracji z mutacją wielopunktową (patrz wykres (g)). Pierwszy z nich skoncentrował się na zbadaniu wpływu prawdopodobieństwa mutacji. Natomiast drugi na podziale wszystkich mutacji między atrybutami należącymi do rozwiązania oraz atrybutami spoza rozwiązania (50% ogólnej liczby mutacji przypada na jedną grupę, a drugie 50% na drugą grupę).

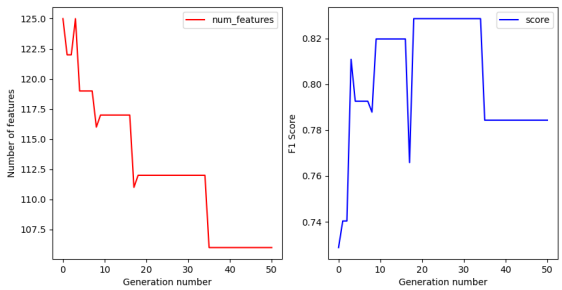
Test 1



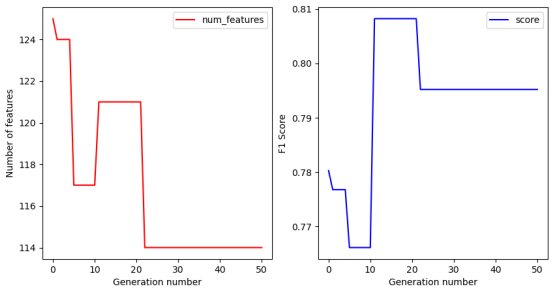
(a) Prawdopodobieństwo mutacji: 0.01



(b) Prawdopodobieństwo mutacji: 0.05



(c) Prawdopodobieństwo mutacji: 0.1

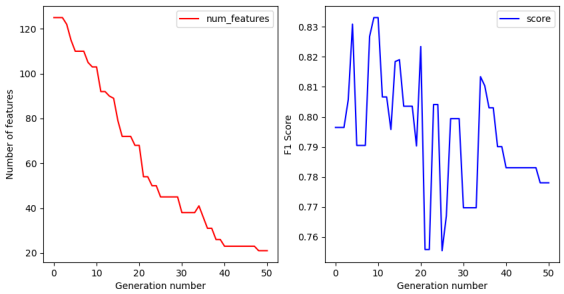


(d) Prawdopodobieństwo mutacji: 0.3

Rysunek 12: Wpływ prawdopodobieństwa mutacji

Wartości F1 Score dla zbioru testującego: (a) 0.796, (b) 0.818, (c) 0.80, (d) 0.749.

Test 2



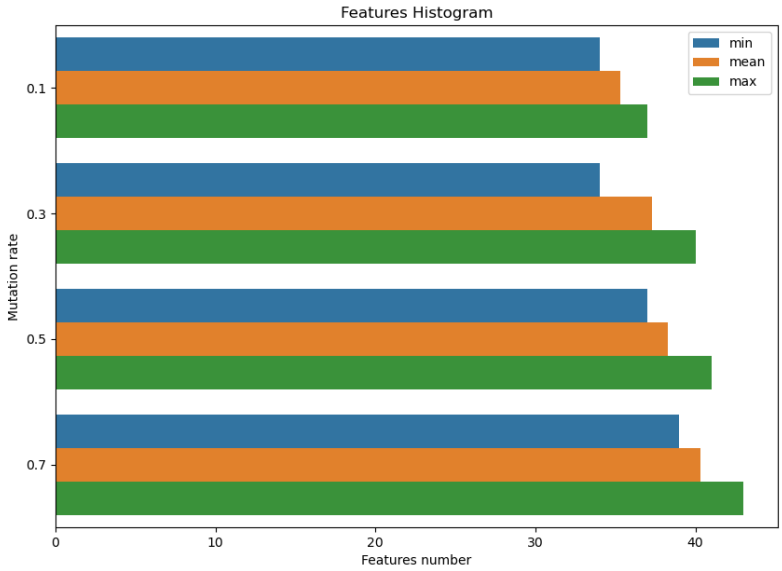
Rysunek 13: Prawdopodobieństwo mutacji: 0.1. Podział liczby mutacji: 50/50.

Wartość F1 Score dla zbioru testującego: 0.788.

Z pierwszego testu można wnioskować, że większe prawdopodobieństwo mutacji rzeczywiście negatywnie wpływa na jakość generowanych rozwiązań. Zauważamy, że dla większych wartości prawdopodobieństw algorytm jest w stanie na początku generować nowe rozwiązania. Jednak po osiągnięciu określonego progu cech, które są brane pod uwagę w rozwiązaniu, jakość rozwiązań przestaje się zmieniać (w przypadku zbyt dużej eksploracji, jakość nie spada, a jedynie utrzymuje się dzięki osobnikom z elity).

Z drugiego testu można wywnioskować, że podział liczby mutacji na poziomie 50/50 istotnie poprawia wyniki, zwiększając eksploatację kosztem eksploracji. Jednakże algorytm z taką konfiguracją staje się bardziej podatny na wpadanie w ekstrema lokalne. Aby przeciwdziałać temu problemowi, skutecznym rozwiązaniem może być zastosowanie strategii "agresywnej mutacji". Strategia ta polega na szybkim zmniejszeniu liczby atrybutów, a następnie stopniowym zwiększaniu eksploracji wokół znalezionego obszaru i kontynuowaniu poszukiwań.

Przed testem na większym podzbiorze, postanowiliśmy znaleźć najlepszą wartość mutacji dla konfiguracji (selekcja turniejowa, krzyżowanie jednorodne, mutacja jednopunktowa). Zbadaliśmy wartości 0.1, 0.3, 0.5, 0.7. Aby zwiększyć precyzję, uruchomiliśmy każdy algorytm 3 razy i na wykresie przedstawiliśmy wartości średnie/minimalne/maksymalne ze wszystkich uruchomień.



Rysunek 14: Wpływ parametru mutacji

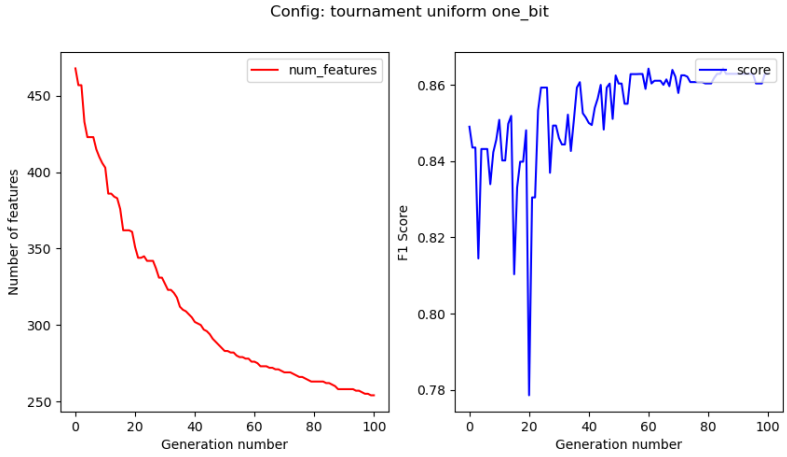
Wartości funkcji celu (F1 Score) na zbiorze testującym:

- 0.1: **0.7986**
- 0.3: **0.7888**
- 0.5: **0.8043**
- 0.7: **0.8096**

Jak widać w tym przypadku (mutacja jednopunktowa), główną siłą, która kieruje algorytm ewolucyjny w kierunku rozwiązania jest krzyżowanie. Mutacja odgrywa rolę uzupełnienia brakujących genów i zwiększenia lokalności poszukiwań. Wpływ parametru mutacji nie jest aż tak zauważalny, jednak widzimy, że im większe prawdopodobieństwo mutacji, tym większa liczba atrybutów jest wybierana na końcu.

Dla dalszych testów przyjmiemy wartość prawdopodobieństwa mutacji równą 0.1.

Dalej dla najlepszej konfiguracji (tournament, uniform, one-bit), przeprowadziliśmy testy dla podzbioru docelowego zbioru o rozmiarze 1000 cech (aby móc efektywnie go porównać z algorytmem Simulated Annealing). W tym celu zwiększyliśmy liczbę iteracji do 100. Pozostałe parametry zostali takie same.



Rysunek 15: Uruchomienie dla 1000 atrybutów

Wyniki na zbiorze testującym są odpowiednio: F1 Score - 0.8557 i Accuracy - 0.8299.

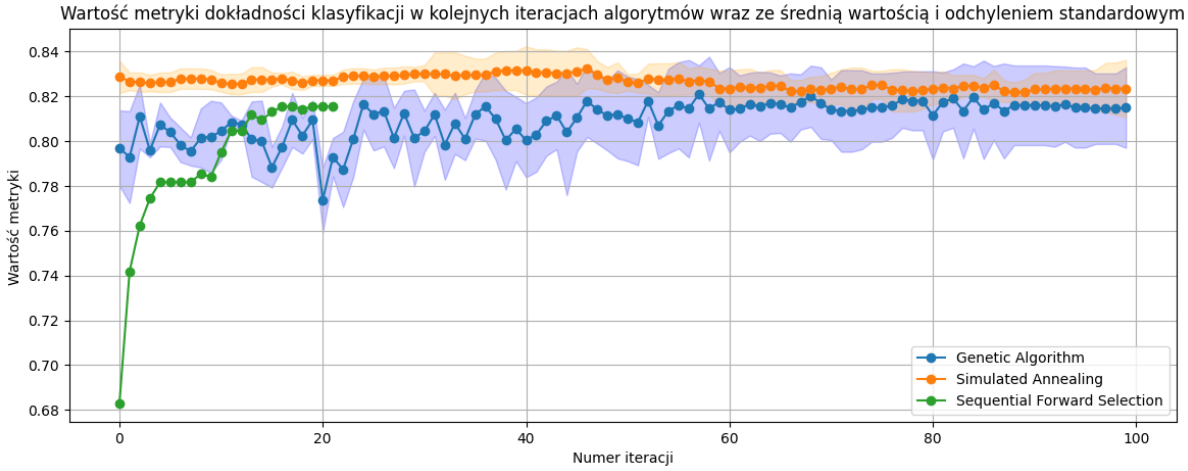
Aby zmniejszyć liczbę atrybutów, można byłoby zwiększyć rozmiar populacji oraz liczbę iteracji. Jednak, spowodowałoby wzrost kosztu obliczeniowego i algorytm z taką konfiguracją wykonywałby się bardzo długo.

9 Porównanie metod

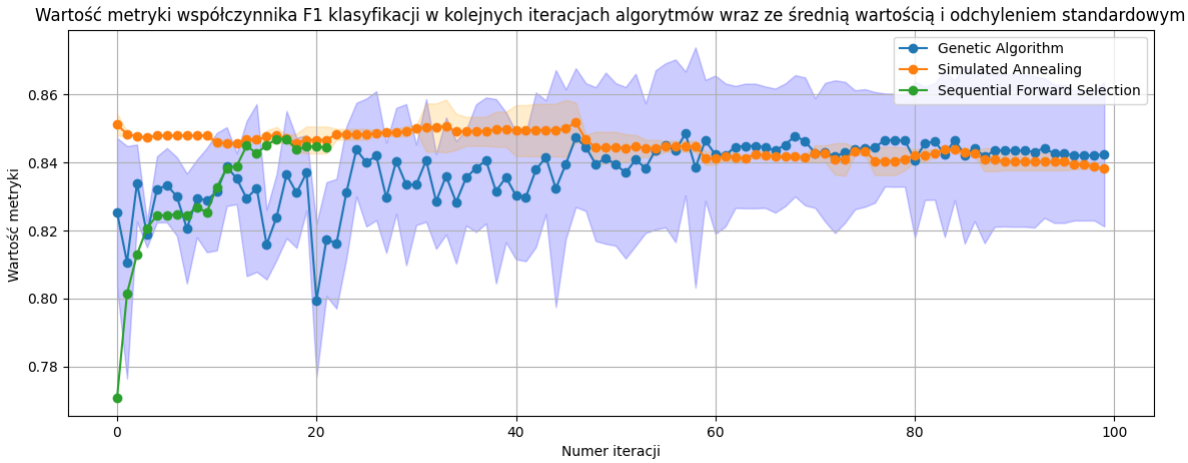
W niniejszej sekcji przeprowadzone zostanie porównanie algorytmów w oparciu o parametry wybrane podczas optymalizacji w poprzedniej części pracy. Optymalizacja przebiegła na zbiorze walidacyjnym zredukowanym do 1000 atrybutów. Analiza skupi się na ocenie wydajności tych metod, z uwzględnieniem ich stabilności wyników, ilustrowanej poprzez odchylenia standardowe oraz wartości minimalne i maksymalne.

Algorytmy zostały uruchomione dla maksymalnej ilości iteracji równej 100. Ze względu na niedeterministyczny charakter algorytmu symulowanego wyżarzania (Simulated Annealing) oraz genetycznego (Genetic Algorithm), ilość uruchomień wyniosła 3. Następnie obliczono średnie i odchylenia standardowe wartości metryk spośród uruchomień, które przedstawione są na wykresach.

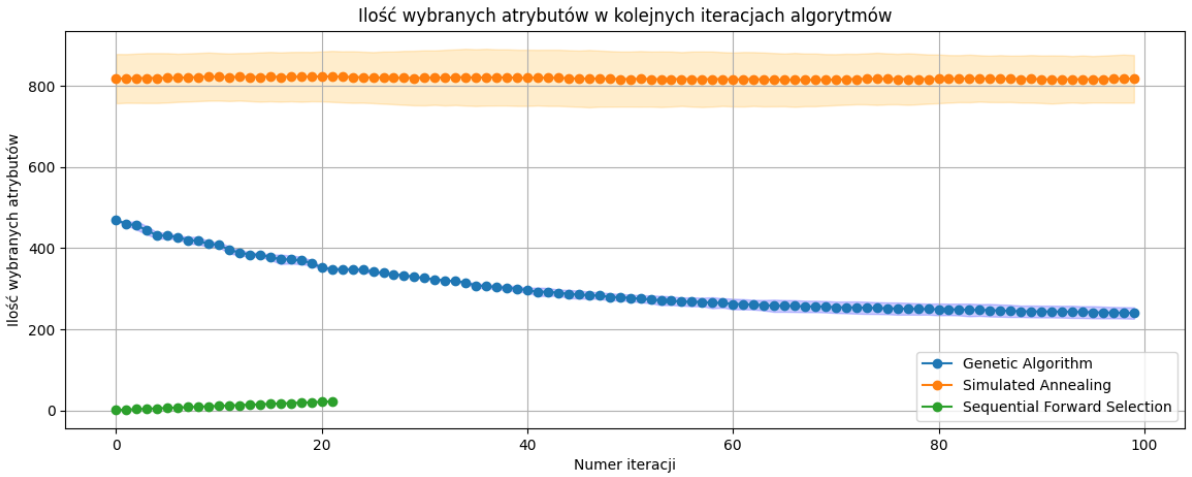
Ze względu na deterministyczny charakter algorytmu sekwencyjnego wybierania najlepszego atrybutu (SFS) uruchomiono go 1 raz aby wybrał 22 najlepsze atrybuty spośród zbioru testowego. Owy algorytm nie wybrał większej ilości atrybutów, ponieważ kolejne nie zwiększyły wartości metryk.



Rysunek 16: Porównanie metryki dokładności (accuracy) w kolejnych iteracjach algorytmów



Rysunek 17: Porównanie metryki współczynnika F1 (f1 score) w kolejnych iteracjach algorytmów



Rysunek 18: Porównanie ilości atrybutów wybranych przez algorytmy w kolejnych iteracjach

Warto jednak zaznaczyć, że chociaż algorytm SFS jest zachłanny, to zupełnie nie gwarantuje znalezienia najlepszego zestawu atrybutów. Łatwo wyobrazić sobie sytuację, w której bardziej się opłacało by wzięcie dwóch atrybutów gorszych niż jednego lepszego. SFS szuka tylko najlepszego lokalnego rozwiązania, przez co nie znajduje globalnie optymalnego rozwiązania.

Testowane miary jakości algorytmu symulowanego wyżarzania ulegają drobnej zmianie w kolejnych iteracjach ze względu na charakter funkcji zmiany, która opiera się na modyfikacji tylko jednego atrybutu w iteracji. Z tego powodu ilość atrybutów w kolejnych iteracjach ulega drobnej zmianie, co zostało przedstawione na rysunku 18. Powoduje to, iż otrzymujemy względnie stałe wartości metryk w kolejnych iteracjach (Rysunek 16 i 17).

Dodatkowo na wykresie 18 przedstawiono, że algorytm genetyczny sukcesywnie minimalizuje ilość wybranych atrybutów nie pogarszając jednocześnie wartości metryk na wykresach 16 i 17.

Algorytm	**F1 Score**	**Accuracy**	**Iteracja**
Sequential Forward Selection	0.8468	0.8154	16
Simulated Annealing	0.8562	0.8408	33
Genetic Algorithm	0.8643	0.8371	85

Tabela 1: Najlepsze wyniki metryk i ich iteracje.

Algorytm genetyczny osiągnął najwyższą wartość metryki f1 score, natomiast Simulated Annealing najwyższą wartość metryki dokładności (Tabela 1).

Można stwierdzić, iż algorytm genetyczny najlepiej sobie poradził ze znalezieniem optymalnego zbioru atrybutów sukcesywnie minimalizując zbiór atrybutów (Rysunek 18) osiągając tym samym najwyższą wartość metryki f1 score.