Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Radioelektroniki
i Technik Multimedialnych
Zakład Elektroniki Jądrowej i Medycznej

Praca dyplomowa inżynierska

na kierunku Elektronika
w specjalności Elektronika i Informatyka w Medycynie

Algorytm genetyczny tworzący nowe przestrzenie cech w zadaniach klasyfikacji

Bartosz Sowul
nr albumu 269059

promotor
dr inż. Piotr Płoński

WARSZAWA 2018

Algorytm genetyczny tworzący nowe przestrzenie cech w zadaniach klasyfikacji

Streszczenie

W pracy zbadano zagadnienie tworzenia nowych przestrzeni cech za pomocą algorytmu genetycznego. Tworzenie nowych cech jest jednym z kluczowych elementów współczesnego uczenia maszynowego, w skład którego wchodzi zadanie klasyfikacji. W pracy zilustrowano działanie dwóch algorytmów klasyfikujących: lasu losowego i k najbliższych sąsiadów. Przedstawiono metody oceny jakości klasyfikatora: metrykę log loss i metodę sprawdzianu krzyżowego, która pomaga w poprawnej ocenie trenowanego modelu. Omówiono możliwe warianty algorytmu genetycznego. Algorytm genetyczny zaimplementowano w języku Python. Skuteczność algorytmu została zweryfikowana eksperymentalnie na przykładowych zestawach danych medycznych. Porównano jakość modeli uczonych na danych oryginalnych, na zbiorze danych złożonym z cech podstawowych i nowych, oraz tylko na cechach stworzonych przez algorytm.

Słowa kluczowe: algorytm genetyczny, klasyfikacja, tworzenie nowych cech

Genetic algorithm for feature engineering in classification tasks

Abstract

The problem of feature engineering using genetic algorithm is considered. Feature creation is one of the key elements of the contemporary machine learning which includes classification task. Two classification algorithms, random forest and k nearest neighbors, are discussed. Logistic loss metric and cross-validation technique for measuring classification performance are described. Genetic algorithm and its possible variants are presented and then implemented in Python. Usability of the implementation is experimentally verified using a sample group of medical data sets. Comparison of models fitted on raw data set, on data set composed of base features and new ones, and only on newly created features is made.

Keywords: genetic algorithm, classification, feature engineering

załącznik do zarządzenia nr 28/2016 r.
Rektora PW

Politechnika Warszawska

załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

Warszawa, 26.01.2018
(miejscowość i data)

Bartosz Sowul
(imię i nazwisko studenta)

269059
(numer albumu)

Elektronika
(kierunek studiów)

OŚWIADCZENIE

Świadomy odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

Rozdział 1

Wstęp

1.1 Cel pracy

Tematem niniejszej pracy jest *Algorytm genetyczny tworzący nowe przestrzenie cech w zadaniach klasyfikacji*. Celem pracy inżynierskiej jest zaprojektowanie oraz implementacja algorytmu genetycznego kreującego nowe cechy na podstawie zadanego zbioru danych. Efektem działania modułu powinien być nowy zbiór danych, który pozwoli na osiągnięcie lepszej jakości klasyfikacji. Moduł powinien oferować użycie dowolnego algorytmu klasyfikacji zgodnego z interfejsem *scikit-learn* [23].

Do testów algorytmu wykorzystano niektóre z dostępnych na stronie *UC Irvine Machine Learning Repository* [33] zbiory danych.

1.2 Plan pracy

Rozdział 1 wyjaśnia cel i motywację pracy. W rozdziale 2 opisano zadanie klasyfikacji, omówiono użyte w pracy algorytmy klasyfikacji oraz metodę oceny ich wyników. Rozdział 3 przedstawia krótką historię algorytmu genetycznego i jego przykładowe zastosowania. Implementację modułu tworzącego nowe cechy pokazano w rozdziale 4. Wyniki działania algorytmu zgromadzono w rozdziale 5. Rozdział 6 zawiera podsumowanie pracy oraz przedstawia możliwości jej dalszego rozwoju.

Rozdział 2

Klasyfikacja

Klasyfikacja (ang. classification) oznacza podział podmiotów na kategorie, pogrupowanie ich w użyteczny sposób [26]. W uczeniu maszynowym (ang. machine learning, ML) grupy takie nazywane są klasami, a członkowie jednej klasy mają wspólne cechy. Nauka algorytmu polega na podaniu mu poprawnie sklasyfikowanych próbek treningowych, na podstawie których, przez analizę wartości cech tych próbek, powstają klasy decyzyjne. Poprawnie zbudowany klasifikator (ang. classifier) umożliwia generalizację, to znaczy, dokonuje prawidłowego przyporządkowania nowych, wcześniej nieznanych przykładów do odpowiedniej grupy.

2.1 Zadanie klasyfikacji

Dany jest:

- zbiór n atrybutów,
- zbiór k klas,
- zbiór próbek treningowych, to jest par (wektor atrybutów, etykieta) takich, że:

$$(i_i, l_i) \ldots (i_j, l_j)$$

$$i = (v_1, \ldots, v_n)$$

$$l \in c_1, \ldots, c_k$$

Problem klasyfikacji polega na wyznaczeniu reguły klasyfikacji (ang. classification rule) na podstawie wartości atrybutów wektorów zbioru treningowego, która pozwoli przyporządkować klasę dla dowolnego elementu dziedziny zadania.

2.2 Przegląd algorytmów klasyfikacji

Zgodnie z twierdzeniem *no free lunch* [35] najlepszy klasyfikator nie będzie taki sam dla różnych zbiorów danych. Wybór odpowiedniego algorytmu klasyfikującego nie jest jednak prostym zadaniem ze względu na ogromną ilość algorytmów i ich dostępnych implementacji [9]. Do zaprezentowania działania modułu będącego tematem pracy wybrano algortymy z dwóch różnych rodzin: najbliższych sąsiadów i lasów losowych.

2.2.1 K najbliższych sąsiadów

Najbliżsi sąsiedzi (ang. nearest neighbors, `NN`) to jeden z najprostszych algorytmów klasyfikujących, użyteczny w przypadkach problemów takich jak: wykrywanie anomalii (ang. anomaly detection), korekcja pisowni (ang. spell checking) czy rekomendowanie (ang. recommender system).

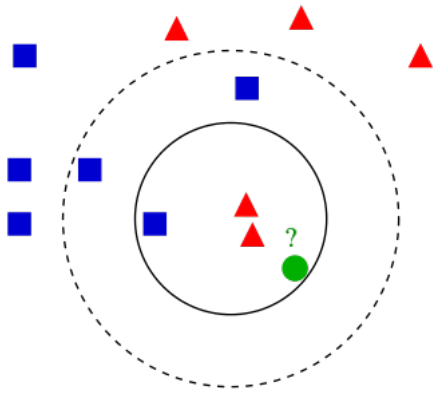
W zbiorze danych M, *najbliższym sąsiadem* obiektu q jest obiekt M_i, który minimalizuje funkcję $dist(q, M_i)$, której wynikiem jest odległość między dwoma obiektami. Warto zauważyć, że jeśli obiekt M_i jest najbliższym sąsiadem obiektu q, to obiekt q nie musi być najbliższym sąsiadem obiektu M_i (Rys.2.1).

Naiwna metoda szukania najbliższego sąsiada polega na liniowym przeszukaniu wszystkich obiektów z przestrzeni M. Szybkość działania algorytmu zależy między innymi od:

- wielkości zbioru danych, który jest przeszukiwany,

- kosztu obliczenia wybranej miary długości,
- użytej reprezentacji (struktury) danych.

Wciąż poszukiwane są nowe metody przyspieszające szukanie sąsiadów.



Rysunek 2.1: Prosty przykład poszukiwania najbliższego sąsiada.

Źródło: https://en.wikipedia.org/wiki/File:KnnClassification.svg

W klasyfikacji z wykorzystaniem k najbliższych sąsiadów (ang. k-nearest neighbors, `k-NN`) przyporządkowanie do klasy odbywa się przez głosowanie większościowe (ang. majority voting), obiekt jest przypisany do klasy, której największa liczba reprezentantów znajduje się wśród k najbliższych sąsiadów [36]:

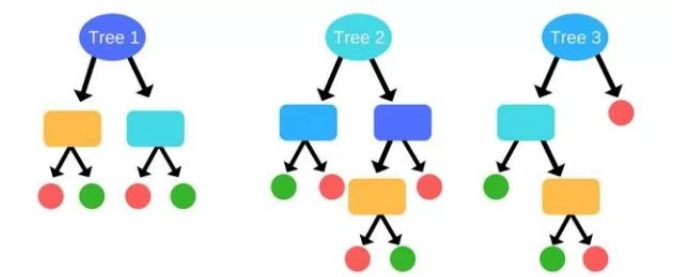
$$Majority\ Voting: \ y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i,y_i)\in M_z} I(v = y_i), \tag{2.1}$$

gdzie v jest etykietą klasy, y_i to etykieta klasy i-tego najbliższego sąsiada (x_i), M_z to lista najbliższych sąsiadów $(M_z \subseteq M)$, a $I(\cdot)$ jest funkcją charakterystyczną (indykatorem) zbioru:

$$I(v, y) = \begin{cases} 1 & \text{jeżeli } v = y \\ 0 & \text{jeżeli } v \neq y \end{cases} \tag{2.2}$$

2.2.2 Las losowy

Las losowy to przedstawiciel modeli złożonych (ang. ensemble learning). Jest połączeniem techniki *bagging* (ang. **b**ootstrap **agg**regat**ing**) i metody losowych podprzestrzeni (ang. random subspace method, `RSM`).



Rysunek 2.2: Procedura uczenie lasu losowego.

Źródło: https://i1.wp.com/dataaspirant.com/wp-content/uploads/2017/04/Random-Forest-Introduction.jpg

Bagging został zaproponowany przez Leo Breimana [6] i polega na losowym generowaniu podzbiorów zbioru uczącego, na których trenowane są następnie słabe klasyfikatory (ang. weak learners). Skuteczność słabego klasyfikatora jest niewiele lepsza niż losowe zgadywanie. Zbiór słabych klasyfikatorów pozwala jednak na stworzenie skutecznego silnego ucznia [27] (ang. strong learner).

`RSM` [11] to metoda łączenia modeli polegająca na konstruowaniu drzew decyzyjnych w losowo wybranej podprzestrzeni oryginalnej dziedziny zadania. Oznacza to, że każdy model jest uczony na zestawie danych treningowych spróbkowanym w przestrzeni cech. Technika ta jest chętnie stosowana w zadaniach, gdzie liczba cech znacząco przekracza liczbę próbek treningowych, na przykład przy danych z funkcjonalnego obrazowania metodą rezonansu magnetycznego [15] (ang. functional magnetic resonance imaging, `fMRI`).

Dany jest zestaw treningowy $X = x_1, \ldots, x_n$ i zestaw etykiet klas $Y = y_1, \ldots, y_n$. Algorytm uczenie lasu losowego (Rys.2.2) dla każdego z T drzew można opisać następująco:

1. Ze zbioru *(X, Y)* losowany jest ze zwracaniem podzbiór zawierający n próbek treningowych z X i Y.

2. Dla każdego węzła w drzewie (drzewo decyzyjne jest grafem):

   (a) losowane jest m cech,

   (b) cecha, która zapewnia najlepszy podział, jest wykorzystana do podziału binarnego na danym węźle.

Po treningu, klasyfikacja wcześniej niewidzianego przykładu odbywa się w drodze głosowania większościowego.

2.3 Metody oceny jakości klasyfikatora

Przy ocenianiu jakości zbudowanego modelu należy wziąć pod uwagę kryteria takie jak:

- trafność klasyfikacji,
- szybkość i skalowalność,
- odporność na szum i brakujące wartości.

Istnieje wiele sposobów analizy trafności klasyfikacji, najpopularniejsze to:

- skuteczność klasyfikacji (ang. classification accuracy),
- macierz pomyłek (ang. confusion matrix),
- krzywa `ROC` i pole pod nią (ang. receiver operating characteristic curve, area under the curve, `AUC`).

Podczas projektowania algorytmu genetycznego zdecydowano się jednak na użycie metryki liczbowej *logistic loss*.

2.3.1 Logistic loss

Logistic loss (znany również jako *log loss* lub *cross-entropy loss*) to miara zdefiniowana jako ujemne prawdopodobieństwo logarytmiczne prawidłowego przyporządkowania etykiety na podstawie danego przez klasyfikator prawdopodobieństwa przyporządkowania.
Log loss wyrażony jest wzorem:

$$log\ loss = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \ln p_{ij} \ , \tag{2.3}$$

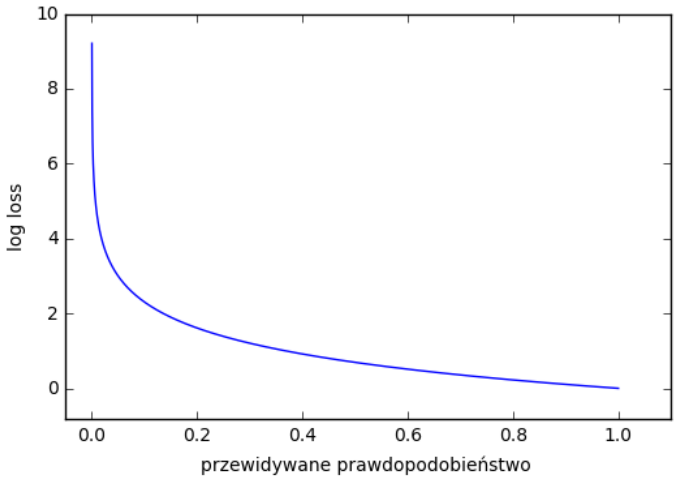
gdzie:

- N - liczba próbek,
- M - liczba klas,
- y_{ij} - binarny indykator mówiący, czy klasa j została przyporządkowana poprawnie próbce i,
- p_{ij} - przewidywane prawdopodobieństwo, że próbka i należy do klasy j.

Dla zadania klasyfikacji binarnej wzór 2.3 upraszcza się do postaci:

$$log\ loss = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \ln p_i + (1-y_i)\ln\left(1-p_i\right)\right] \tag{2.4}$$

Rysunek 2.3 przedstawia wykres funkcji *logistic loss*. Jest to funkcja malejąca, w związku z czym perfekcyjny klasyfikator (czyli model przyporządkowujący poprawnie wszystkie próbki z prawdopodobieństwem 1) będzie miał miarę *log loss* równą 0.



Rysunek 2.3: Wykres funkcji log loss.

2.3.2 Sprawdzian krzyżowy

Sprawdzian krzyżowy (również walidacja krzyżowa, kroswalidacja, ang. cross-validation, `CV`) to metoda statystyczna polegająca na podziale zbioru na podzbiory, a następnie przeprowadzeniu uczenia modelu na niektórych podzbiorach (zbiorach uczących), podczas gdy pozostałe służą do oceny jakości modelu (zbiór testowy). `CV` używana jest, gdy zbiór testowy dla danego zadania nie jest dostępny lub jest zbyt mały, co uniemożliwia poprawną ocenę trenowanego modelu [14].

Najprostsza strategia zakłada losowy podział zbioru danych na dwie części: zestaw treningowy i zestaw walidacyjny. Algorytm uczony jest na pierwszym zbiorze, a następnie jest użyty do predykcji na próbkach ze zbioru testowego. Takie podejście nie jest pozbawione wad. Trening przeprowadzany jest na znacząco mniejszym zbiorze danych niż oryginalny, przez co obliczony błąd klasyfikacji będzie przeszacowany. Błąd ten będzie również wahał się w zależności od podziału zbioru obserwacji na podzbiory.

Rozwiązaniem próbującym rozwiązać problemy poprzedniej strategii jest wariant `CV` nazwany *leave-one-out cross-validation* (`LOOCV`). Dany jest zbiór danych składający się z n obserwacji. Zamiast jego podziału na 2 równe podzbiory, `LOOCV` zakłada, że zbiorem walidacyjnym jest jedna próbka (x_1, y_1), a zbiorem uczącym pozostałe $\{(x_2, y_2), \ldots, (x_n, y_n)\}$. Głównym minusem tej metody jest koszt jej implementacji, ponieważ algorytm musi być uczony n razy. Dlatego też głównym przeciwwskazaniem do użycia `LOOCV` jest duże n lub długi czas nauki algorytmu.

Alternatywą dla `LOOCV` jest k-krotna walidacja (ang. k-fold cross-validation, `k-fold CV`). Polega ona na podziale całego zbioru na k podzbiorów, w przybliżeniu o takim samym rozmiarze. Następnie pierwszy podzbiór jest traktowany jako zbiór testowy, a pozostałe wykorzystane są w procesie nauki modelu. Proces ten powtarzany jest k razy, za każdym razem z innym podzbiorem jako zestawem walidującym.

Rozdział 3

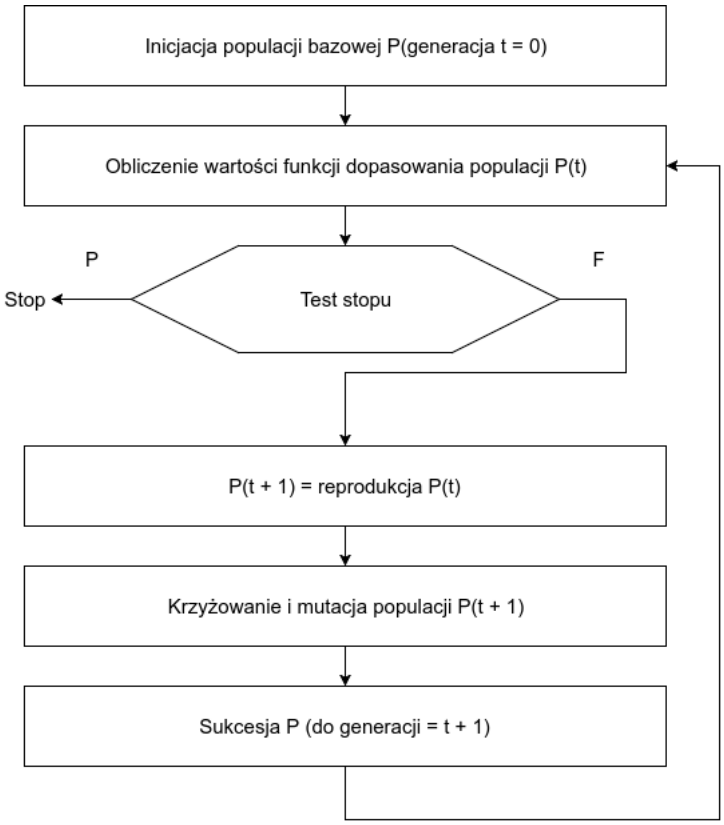
Algorytm genetyczny

Biologiczna ewolucja zakłada, że przetrwanie gatunku zależy od tego, jak dobrze potrafi się on zaadoptować do panujących w środowisku warunków, a następnie wydać na świat potomstwo. W kolejnych pokoleniach członkowie danej populacji zastępowani są przez potomstwo rodziców lepiej przystosowanych do przetrwania i rozrodu w środowisku, w którym zachodzi dobór naturalny. Biologia zainspirowała wiele algorytmów, między innymi sztuczne sieci neuronowe czy sztuczne układy odpornościowe. Jednym z takich algorytmów jest, zaproponowany w 1975 roku przez Johna Hollanda [12], algorytm genetyczny, którego zadaniem było modelowanie procesu ewolucji.

3.1 Prosty algorytm genetyczny

Działanie algorytm Hollanda [3] (ang. Simple Genetic Algorithm, `SGA`) przedstawia rysunek 3.1. Algorytm przetwarza populację bazową $P(t)$. Inicjowana jest populacja bazowa $P(t = 0)$. Następnie dla każdego z losowo wygenerowanych osobników jest obliczana wartość funkcji przystosowania. Funkcja przystosowania (inaczej funkcja oceny) $\Phi(\mathbf{X})$ to miara określająca, jak dobrze dany osobnik jest dopasowany do środowiska. Kolejnym krokiem jest reprodukcja - do $P(t+1)$ kopiowane są losowane wybrane osobniki z populacji bazowej (prawdopodobieństwo wylosowania jest wyższe dla osobników o większej wartości funkcji przystosowania). Osobniki z $P(t+1)$ poddawane są operacjom genetycznym. Pierwszą z nich jest krzyżowanie. Losowana są pary osobników, dla każdej pary jest podejmowana, z prawdopodobieństwem

p_c, decyzja o krzyżowaniu. Jeśli wystąpi krzyżowanie, to osobniki potomne zastępują swoich rodziców, w przypadku decyzji negatywnej para pozostaje bez zmian. Po krzyżowaniu dochodzi do mutacji - dla każdego genu każdego osobnika, z prawdopodobieństwem p_m, wartość genu zmienia się na przeciwną. Następuje sukcesja do generacji $P(t+1)$. Pętla jest wykonywana do spełnienia warunku zatrzymania.



Rysunek 3.1: Schemat `SGA`.

3.2 Elementy algorytmu

Definiując `GA` należy wybrać:

- kodowanie, czyli sposób reprezentacji problemu,

- funkcję dopasowania,
- metodę inicjacji populacji,
- operatory ewolucyjne,
- rodzaj selekcji (reprodukcji i sukcesji),
- kryterium stopu.

3.2.1 Kodowanie

Dwie najpopularniejsze formy kodowania to reprezentacja binarna i zmiennoprzecinkowa. Tradycyjnie w `GA` była używana reprezentacja binarna, ale sprawia ona, że już przy niewielkiej liczbie zmiennych, długość słowa binarnego jest duża, co prowadzi do olbrzymiej przestrzeni przeszukiwań. Dlatego też preferowana jest reprezentacja zmiennoprzecinkowa, która jest szybsza i prowadzi do lepszej dokładności [19]. Niewątpliwą zaletą `GA` jest możliwość dobrania reprezentacji odpowiedniej dla danego problemu, dlatego też spotyka się również kodowanie całkowitoliczbowe czy drzewiaste.

3.2.2 Krzyżowanie

Operatory krzyżowania (ang. crossover) można podzielić na operatory wymieniające i uśredniające [3]. Z pary rodziców może powstać jeden osobnik potomny lub para potomków.

3.2.2.1 Krzyżowanie wymieniające

Można rozróżnić krzyżowanie jednopunktowe, dwupunktowe lub równomierne. W pierwszym przypadku, dla kodu genetycznego o długości n, wybiera się (z rozkładem jednostajnym) liczbę c ze zbioru $\{1, 2, \ldots, n-1\}$, która stanowi tzw. punkt rozcięcia. Część pierwszą chromosomu X^1 łączy się z drugą częścią X^2, a pierwszą część X^2 z drugą częścią X^1. Powstaje nowy osobnik $Y = [X^1_1, \ldots, X^1_c, X^2_{c+1}, \ldots, X^2_n]$.

Krzyżowanie dwupunktowe przeprowadza się na podobnej zasadzie, z tym, że losowane są dwa punkty rozcięcia. Następnie, krzyżowane chromosomy X_1, X_2, dzieli się na trzy części i zamienia środkowy fragment, co daje chromosom potomny $Y = [X_1^1, \ldots, X_{c_1}^1, X_{c_1+1}^2, \ldots, X_{c_2}^2, X_{c_2+1}^1, \ldots, X_n^1]$.

Osobnik potomny przy zastosowaniu krzyżowania równomiernego jest tworzony w następujący sposób:

$$Y_i = \begin{cases} X_i^1 & \text{jeżeli } \xi_{U(0,1),i} < p_e \\ X_i^2 & \text{w przeciwnym przypadku.} \end{cases} \tag{3.1}$$

Typowo $p_e = \frac{1}{2}$. W wariancie z dwoma potomkami drugi chromosom tworzony jest analogicznie.

3.2.2.2 Krzyżowanie uśredniające

Krzyżowanie uśredniające stosowane jest przy reprezentacji zmiennoprzecinkowej. Generuje się wartości $\xi_{U(0,1)}$ i uśrednia wartości genów chromosomów rodzicielskich:

$$\mathbf{Y} = \mathbf{X^1} + \xi_{U(0,1)}(\mathbf{X^2} - \mathbf{X^1}) \tag{3.2}$$

3.2.3 Mutacja

3.2.3.1 Mutacja dla reprezentacji binarnej

Dla każdego genu X_i podejmowana jest, z prawdopodobieństwem p_m, decyzja o zmianie bitu. W przypadku zmiany następuje negacja bitu lub zamiana bitu na losowy.

3.2.3.2 Mutacja dla reprezentacji zmiennoprzecinkowej

Najczęściej korzysta się z mutacji z rozkładów nieskorelowanych, wyrażonej wzorem:

$$Y_i = X_i + \xi_{r,i} \tag{3.3}$$

gdzie i oznacza numer genu, $\xi_{r,i}$ jest realizacją zmiennej losowej o rozkładzie r, definiowanej indywidualnie dla każdego genu.

3.2.4 Reprodukcja

3.2.4.1 Reprodukcja proporcjonalna

Prawdopodobieństwo wylosowania osobnika jest wprost proporcjonalne do jego wartości funkcji przystosowania i wyraża się wzorem:

$$p_r(\mathbf{X}) = \frac{\Phi(\mathbf{X})}{\sum_{\mathbf{Y} \in \mathbf{P}^t} \Phi(\mathbf{Y})} \tag{3.4}$$

3.2.4.2 Reprodukcja rangowa

Ranga to liczba charakteryzująca jakość danego osobnika na tle populacji. Na jej podstawie podane jest prawdopodobieństwo reprodukcji każdego osobnika. Rangę nadaje się sortując nierosnąco populację według wartości funkcji przystosowania osobników - ranga odpowiada numerowi osobnika w tym szeregu. W tej sytuacji osobniki o takiej samej wartości funkcji przystosowania będą mieli różne rangi. Aby uniknąć takiej sytuacji można nadać im taką samą rangę. Następnie należy zdefiniować zmienna losową, przypisując każdemu osobnikowi prawdopodobieństwo reprodukcji na podstawie jego rangi. Funkcja ta musi być nierosnąca z numerem rangi.

3.2.4.3 Reprodukcja progowa

Reprodukcja progowa jest szczególnym przypadkiem reprodukckji rangowej. Funkcja $p_r(\mathbf{X})$ ma postać progu, powyżej którego osobnik jest dopuszczony do reprodukcji.

3.2.4.4 Reprodukcja turniejowa

Najpierw wybierana jest część populacji biorąca udział w turnieju. Część ta liczy q osobników. Wszystkie q-elementowe kombinacje z obecnie przetwarzanej populacji są jednakowo prawdopodobne. Możliwe są dwa warianty: losowanie ze zwracaniem i bez zwracania. Następnie wśród wylosowanej populacji $\mathbf{Q}$ przeprowadza się turniej. Zwycięzcą zostaje osobnik o największej wartości funkcji przystosowania. Jest on kopiowany do populacji potomnej.

Proces wyboru uczestników i przeprowadzania turnieju jest wykonywany wielokrotnie, aż do uzyskania wymaganego rozmiaru populacji potomnej.

3.2.5 Sukcesja

3.2.5.1 Sukcesja z całkowitym zastępowaniem

Podstawowy wariant sukcesji, w którym populacja potomna staje się nową populacją bazową.

3.2.5.2 Sukcesja z częściowym zastępowaniem

W sukcesji z częściowym zastępowaniem nowa populacja bazowa może zawierać osobniki ze starej populacji bazowej i z populacji potomnej. Część osobników starej populacji bazowej należy usunąć, na przykład poprzez:

- usunięcie osobników o najgorszej wartości funkcji przystosowania,
- usunięcie losowo wybranych osobników.

3.2.5.3 Sukcesja elitarna

Z populacji P^t wybieranych jest η najlepszych osobników, ich populacja to P^t_η, a populacja potomstwa to O^t. Następnie tworzy się populację P^{t+1}, wybierając μ najlepszych osobników z $P^t_\eta \cup O^t$.

Poważną wadą sukcesji elitarnej jest osiadanie w obszarach przyciągania maksimów lokalnych (wpadanie w pułapki ewolucyjne). Aby wydobyć się z takiej pułapki należy zastosować mutację o bardzo dużym zasięgu.

3.3 Zastosowania

Algorytmy genetyczne mają wiele zastosowań praktycznych. Jest to przede wszystkim znajdowanie przybliżonych rozwiązań problemów optymalizacyj-

nych. Znane przykłady to problem plecakowy i problem komiwojażera. `GA` można również wykorzystać do budowy struktury sztucznej sieci neuronowej i wyboru jej wag [5]. W zastosowaniach medycznych algorytmu użyto do obróbki obrazów w cyfrowej angiografii różnicowej [10].

Tworzenie nowych cech (ang. feature creation lub feature engineering) przy pomocy algorytmu genetycznego nie jest zagadnieniem nowym [29]. *Feature engineering* jest być może najważniejszym elementem współczesnego *data science* [30], czyli interdyscyplinarnej dziedziny łączącej statystykę, analizę danych i uczenie maszynowe. Celem tej pracy było uproszczenie tego procesu wykorzystując `GA`.

Rozdział 4

Implementacja algorytmu

W tym rozdziale opisano wykorzystane podczas pisania modułu narzędzia oraz przedstawiono podjęte decyzji projektowe.

4.1 Opis wykorzystanych narzędzi

Algorytm zaimplementowano w języku Python [24]. Moduł jest kompatybilny z wersjami języka 2.7, 3.5 i nowszymi.
W pracy wykorzystano następujące (kolejność alfabetyczna), ogólnodostępne moduły Pythona:

- *matplotlib* [13] - biblioteka do tworzenia wykresów,
- *NumExpr* [20] - szybki ewaluator wyrażeń algebraicznych,
- *NumPy* [21] - biblioteka numeryczna,
- *pandas* [22] - zestaw stuktur i narzędzi do analizy danych,
- *pathos* [17] [18] - framework ułatwiający obliczenia równoległe,
- *six* [28] - biblioteka wspomagająca moduły działające jednocześnie z Pythonem 2 i 3,

- *scikit-learn* [23] - biblioteka do uczenia maszynowego,
- *tqdm* [32] - moduł wyświetlający pasek postępu w terminalu.

4.2 Projekt algorytmu genetycznego

Implementując algorytm zdecydowano się na kodowanie całkowitoliczbowe. Liczby całkowite odpowiadają indeksom tablicy operatorów matematycznych, za pomocą których przekształcane są zbiory danych. Przykładowy osobnik to wektor liczb, na przykład $\begin{bmatrix}21 & 1 & 19\end{bmatrix}$. Populacja zawiera 100 osobników.

Wyznacznikiem jakości klasyfikatora jest funkcja *log loss*, która jest funkcją malejącą. Z tego powodu oraz dla zachowania zgodności interfejsu pakiet *scikit-learn* wykorzystuje funkcję *neg log loss*, która jest rosnąca. Jest ona użyta do oceny jakości przystosowania danego osobnika.

Zdecydowano się na selekcję turniejową z $q = 4$. Proces wyboru rodziców przedstawia listing 4.1.

```
def _select_parents(self, q=4):
    parents = np.empty((0, q), dtype=np.int8)
    for i in range(self.pop_members-self.migrants-self.elite):
        parent = np.random.choice(
                    self.pop_members, size=(1, q), replace=False)
        parents = np.append(parents, parent, axis=0)
    parents = np.amin(parents, axis=1)
    np.random.shuffle(parents)
    return np.reshape(parents, (len(parents)//2, 2))
```

Listing 4.1: Selekcja turniejowa.

Zaimplementowano krzyżowanie wymieniające równomierne z $p_e = \frac{1}{2}$, w wyniku którego z 2 rodziców powstają 2 osobniki potomne. Mutacja przeprowadzana jest globalnie i polega na dodaniu do każdego chromosomu wartości z przedziału $(-int(std) - 1,\ int(std) + 1)$, gdzie int to funkcja zaokrąglająca liczbę zmiennoprzecinkową w stronę 0, a std to wartość odchylenia standardowego danej populacji. W praktyce polega to na dodaniu do siebie dwóch macierzy (Listing 4.2) - macierzy zawierającej chromosomy osobników danej populacji i wygenerowanej macierzy wartości mutujących. Dla wartości,

które przekroczą wartość długości tablicy operatorów matematycznych przeprowadzana jest naprawa chromosomu za pomocą operacji modulo.

```
def _mutate(self, new_population, std):
    mutation = np.random.randint(
               -int(std)-1, int(std)+1, size=new_population.shape)
    new_population = (new_population + mutation) %\
                      self.n_operators
    return new_population
```

Listing 4.2: Mutacja i naprawa chromosomów.

Nowa generacja składa się z 90 potomków, 4 najlepiej przystosowanych osobników z populacji przetwarzanej (sukcesja elitarna) oraz 6 migrantów, którzy w tym przypadku są nowymi, losowo powstałymi osobnikami.

4.3 Interfejs modułu

Interfejs modułu był wzorowany na powszechnie używanych bibliotekach do uczenia maszynowego. Konstruktor klasy *GeneticAlgorithm* (Listing 4.3) wymaga od użytkownika zainicjalizowania:

- zmiennej *clf*, czyli klasyfikatora zgodnego z interfejsem modułu *scikit-learn* (implementującego metodę *fit* na zbiorach X i y),
- zmiennej *cv*, czyli sposobu przeprowadzenia `CV` (domyślnie 5-fold `CV`),
- jednej ze zmiennych *duration* lub *max_iter*, czyli wybrania warunku stopu algorytmu - czas działania programu (w minutach) lub ilość przetworzonych generacji,
- zmiennej *base_included*, która określa, czy do stworzenia nowego zbioru danych wykorzystywany jest zbiór oryginalny (baza).

```
def __init__(self, clf, cv=5,
             duration=None, max_iter=None, base_included=True)
```

Listing 4.3: Konstruktor klasy *GeneticAlgorithm*.

Fragment kodu odpowiadający za inicjalizację tylko jednej ze zmiennych *duration* i *max_iter* przedstawia Listing 4.4.

```
1  if all(var is not None for var in (duration, max_iter)):
2      raise ValueError(
3          'Duration and max_iter variables are both not None.
4          One of them should be None.')
5  if all(var is None for var in (duration, max_iter)):
6      raise ValueError(
7          'Duration and max_iter variables are both None.
8          Only one of them should be None.')
9  if duration is not None:
10     if isinstance(duration, (integer_types, float)):
11         self.duration = duration
12         self.max_iter = max_iter
13     else:
14         raise ValueError(
15             'Duration value must be a float or an integer.')
16 else:
17     if isinstance(max_iter, integer_types):
18         self.max_iter = max_iter
19         self.duration = duration
20     else:
21         raise ValueError('Max_iter value must be an integer.')
```

Listing 4.4: Inicjalizacja zmiennych *duration* i *max_iter*.

Za tworzenie nowych cech odpowiada metoda *fit*, która wymaga podania 2 zbiorów: zbioru danych i zbioru etykiet. Metoda *transform* pozwala na przekształcenie oryginalnych danych według zadanych przez danego osobnika cech. Metoda *get_params* zwraca wektor przekształceń najlepszego lub najczęściej najlepszego osobnika. Funkcje *save* i *load* pozwalają odpowiednio na zapisanie i załadowanie wybranego osobnika razem z jego wektorem przekształceń. Wykres pokazujący zmianę średniego przystosowania populacji oraz najlepszego osobnika można wywołać funkcją *plot*.

4.4 Przykład użycia

Listing 4.5 pokazuje, jak łatwe jest zastosowanie modułu w praktyce. Linie 4-7 importują potrzebne moduły: zbiór danych *iris*, algorytm `KNN` oraz algorytm genetyczny. W linii 9 inicjalizowany jest `KNN`, w 10 przekazywany jest

on do konstruktora `GA`, razem z 5-fold `CV` i czasem trwania 1 minuta. W linii 11 ładowany jest zbiór danych, który następnie jest przekazywany do funkcji *fit* (linia 13). Następuje tworznie nowych cech. Linie 15 i 16 wyświetlają odpowiednio najczęściej najlepszego i najlepszego osobnika. Polecenie w linii 17 pokazuje wykres wartości funkcji przystosowania w funkcji numeru generacji.

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  from sklearn.datasets import load_iris
5  from sklearn.neighbors import KNeighborsClassifier
6
7  from ga import GeneticAlgorithm
8
9  clf = KNeighborsClassifier()
10 ga = GeneticAlgorithm(clf, 5, duration=1)
11 iris = load_iris()
12
13 ga.fit(iris.data, iris.target)
14
15 ga.get_params('most_freq')
16 ga.get_params('best')
17 ga.plot()
```

Listing 4.5: Minimalny program wykorzystujący zbudowany moduł.

Rozdział 5

Wyniki

5.1 Wykorzystane zbiory danych

Testy algorytmu zostały przeprowadzone przy użyciu następujących zbiorów danych:

- *Acute Inflammations* [7], to zbiór danych do diagnozy ostrego zapalenia pęcherza moczowego. Zapalenie dolnych dróg moczowych wywołuje pałeczka okrężnicy, która dostaje się do układu moczowego przez cewkę moczową. Właściwie zdiagnozowana i leczona choroba nie jest groźna, jednak niewyleczona może nawracać i w konsekwencji prowadzić nawet do zapalenia nerek.

- *Arrhythmia* [2], to zestaw danych do diagnozy obecności i rodzaju arytmii. Arytmia serca polega na zaburzeniach rytmu serca objawiających się przyspieszeniem, zwolnieniem albo nieregularnościami pojawiającymi się bez wyraźnej przyczyny.
  Z uwagi na bardzo nierównomierną liczność poszczególnych klas, uproszczono zadanie do zadania klasyfikacji binarnej. Etykiety klas różnych rodzajów arytmii zostały zastąpione jedną etykietą oznaczającą występowanie choroby. Etykiety próbek, w których nie stwierdzono arytmii, pozostawiono bez zmian.

- *Breast Cancer Wisconsin* [31], to zbiór danych obliczony na podstawie obrazu materiału uzyskanego z biopsji aspiracyjnej cienkoigłowej (ang.

fine needle aspirate, FNA). Zaletą tej metody jest możliwość uzyskania materiału z guzów położonych głęboko wśród tkanek w sposób mało inwazyjny, bez znieczulenia ogólnego.

- *Cardiotocography* [4], to zbiór cech uzyskanych z przetworzonych kardiotokogramów. Kardiotokografia (KTG) to monitorowanie czynności serca płodu z jednoczesnym zapisem czynności skurczowej macicy. Badanie jest wykonywane pod koniec i w trakcie porodu, pozwala wykryć sytuacje zagrożenia życia płodu.
  W tym zadaniu algorytm klasyfikuje stan płodu (normalny, podejrzany, patologiczny).

- *Heart Disease* [1], to zestaw wyników pacjentów z Cleveland z chorobą niedokrwienną serca, która, ze wszystkimi jej podtypami, jest najczęstszą przyczyną śmierci w większości państw zachodnich [34].

- *Mammographic Mass* [8], to zbiór atrybutów z BI-RADS (ang. Breast Imaging-Reporting and Data System, system stworzony w celu standaryzacji opisów mammograficznych), na podstawie którego można próbować przewidzieć, czy nowotwór będzie łagodny czy złośliwy.

- *SPECT Heart* [16], to zestaw cech binarnych stworzonych na podstawie obrazów z badania tomografii emisyjnej pojedynczych fotonów (ang. Single Proton Emission Computed Tomography, SPECT). Źródłem promieniowania w SPECT są organy badanego pacjenta, w których zgromadzony jest promieniotwórczy izotop. Gammakamera jest wykorzystywana do obrazowania kolejnych warstw, na podstawie których jest rekonstruowana macierz aktywności izotopu. Wartości macierzy kodowane są kolorami, dając w ten sposób obraz.

- *Thyroid Disease* [25], to zbiór danych, na podstawie którego można stwierdzić stan tarczycy pacjenta (niedoczynność, norma, poniżej normy). Niedoczynność tarczycy może być spowodowana uszkodzeniem gruczołu tarczowego lub niedoborem hormonu tyreotropowego (ang. thyroid-stimulating hormone, TSH).

5.2 Przeprowadzone testy

Przed przeprowadzeniem testów algorytmu każdy z zestawów danych został wstępnie przetworzony:

- cechy jakościowe zostały zakodowane do wartości liczbowych,
- brakujące dane (ang. missing values lub missing data) zostały zastąpione wartościami spoza zbioru wartości danej cechy,
- wszystkie wartości zostały przeskalowane do zakresu (0, 1).

Następnie, na tak przetworzonych danych oryginalnych, uruchomiono algorytmy `RFC` i `KNN` w celu uzyskania wyniku bazowego na danym zbiorze.

Dla każdego przetworzonego zbioru algorytm genetyczny został uruchomiony 4 razy:

- z klasyfikatorem las losowy (`RFC`), gdzie do oceny klasyfikatora wykorzystywane były:
  - cechy z oryginalnego zbioru razem z nowo powstałymi cechami,
  - tylko nowe cechy,
- z klasyfikatorem `KNN`, gdzie do oceny klasyfikatora wykorzystywane były:
  - cechy z oryginalnego zbioru razem z nowo powstałymi cechami,
  - tylko nowe cechy.

Las losowy w `RFC` składa się z 10 drzew, a maksymalna głębokość pojedynczego drzewa została ustawiona na 3.
W `KNN` klasyfikacja jest dokonywana na podstawie 5 najbliższych sąsiadów, a odległość liczona jest za pomocą standardowej metryki euklidesowej.

5.3 Analiza wyników

		RFC			
dane	wymiar (wiersze, kolumny)	wynik bazowy	wynik po 1. generacji	wynik po 25. generacjach	wynik po 100. generacjach
acute	(120, 6)	-0.151	-0.034	-0.004	-0.001
arrhytmia	(430, 279)	-0.564	-0.566	-0.557	-0.553
wisconsin	(699, 9)	-0.133	-0.089	-0.080	-0.079
ctg	(2126, 31)	-0.275	-0.228	-0.222	-0.217
heart_disease	(303, 13)	-0.407	-0.396	-0.382	-0.380
mammo_masses	(961, 5)	-0.393	-0.394	-0.388	**-0.388**
spect	(80, 22)	-1.021	-0.562	-0.543	-0.543
thyroid_disease	(3772, 21)	-0.116	-0.061	-0.052	-0.047

		KNN			
dane	wymiar (wiersze, kolumny)	wynik bazowy	wynik po 1. generacji	wynik po 25. generacjach	wynik po 100. generacjach
acute	(120, 6)	-0.010	0.000	0.000	**0.000**
arrhytmia	(430, 279)	-3.212	-2.149	-1.845	-1.845
wisconsin	(699, 9)	-0.359	-0.260	-0.168	-0.168
ctg	(2126, 31)	-0.308	-0.262	-0.248	**-0.247**
heart_disease	(303, 13)	-2.047	-1.607	-1.167	-1.034
mammo_masses	(961, 5)	-2.164	-1.820	-1.692	-1.623
spect	(80, 22)	-3.941	-2.252	-1.924	-1.472
thyroid_disease	(3772, 21)	-1.340	-0.797	-0.645	-0.645

Tabela 5.1: Wyniki RFC i KNN na zbiorach danych złożonych z oryginalnych i stworzonych cech.

Tabela 5.1 przedstawia wyniki działania algorytmu genetycznego na zbiorach danych złożonych ze starych i nowo stworzonych cech. Poszczególne wyniki są średnią wyników uzyskanych z 5-fold `CV`. Pierwsza kolumna zawiera nazwę wykorzystanego zestawu danych, w drugiej natomiast podany jest rozmiar danego zbioru w formacie $(wiersze, kolumny)$. Trzecia kolumna to wynik klasyfikatora uzyskany na podstawowym zbiorze danych. Następne trzy kolumny zawierają wyniki algorytmu klasyfikującego uzyskane na zbiorze danych składającym się z cech oryginalnych i stworzonych przez algorytm po 1, 25 i 100 generacjach. Wymiar nowego zbioru danych to $(wiersze, kolumny + n)$, gdzie n to liczba nowych cech zaproponowanych przez `GA`. Dla ustalonego ziarna (ang. seed) generatora liczb, $n = 5$. Przypadki, w których jakość kla-

syfikatora poprawiła się względem wyniku bazowego, zaznaczone są zielonym kolorem, w przeciwnym przypadku jest to kolor różowy. Tabela 5.2 przedstawia analogiczne wyniki klasyfikatorów na zbiorach danych stworzonych tylko z nowych cech zaproponowanych przez `GA`.

		RFC			
dane	wymiar (wiersze, kolumny)	wynik bazowy	wynik po 1. generacji	wynik po 25. generacjach	wynik po 100. generacjach
acute	(120, 6)	-0.151	-0.044	-0.003	**0.000**
arrhytmia	(430, 279)	-0.564	-0.605	-0.573	**-0.527**
wisconsin	(699, 9)	-0.133	-0.088	-0.079	**-0.076**
ctg	(2126, 31)	-0.275	-0.248	-0.201	**-0.179**
heart_disease	(303, 13)	-0.407	-0.419	-0.395	**-0.378**
mammo_masses	(961, 5)	-0.393	-0.401	-0.391	-0.389
spect	(80, 22)	-1.021	-0.554	-0.518	**-0.513**
thyroid_disease	(3772, 21)	-0.116	-0.088	-0.033	**-0.033**

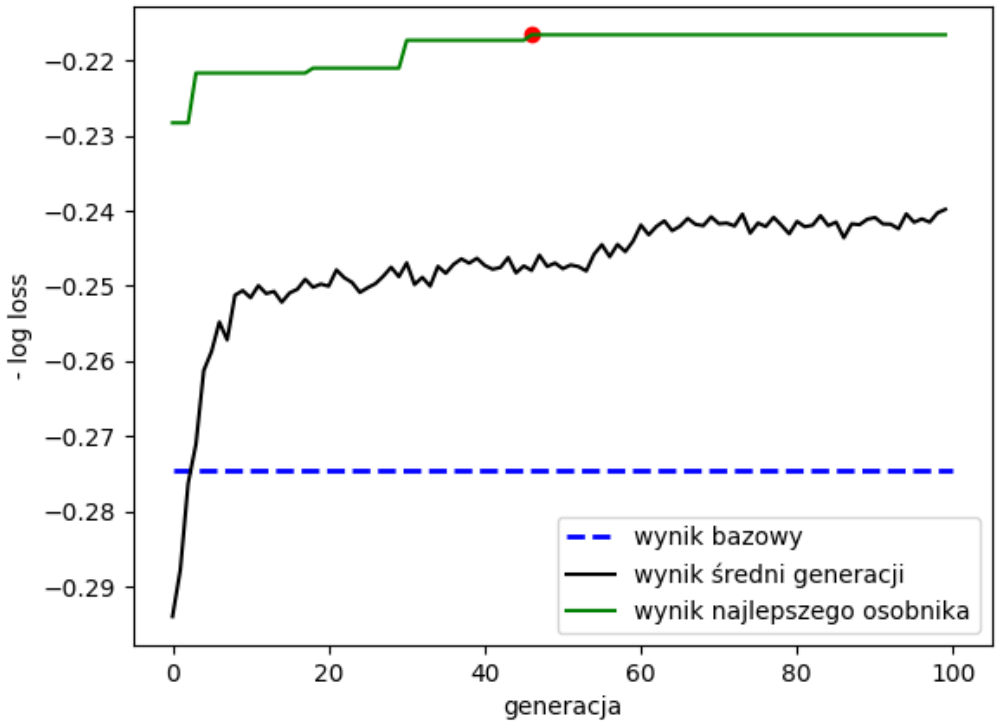
		KNN			
dane	wymiar (wiersze, kolumny)	wynik bazowy	wynik po 1. generacji	wynik po 25. generacjach	wynik po 100. generacjach
acute	(120, 6)	-0.010	0.000	0.000	0.000
arrhytmia	(430, 279)	-3.212	-1.218	-0.624	**-0.601**
wisconsin	(699, 9)	-0.359	-0.264	-0.158	**-0.158**
ctg	(2126, 31)	-0.308	-1.064	-0.572	-0.419
heart_disease	(303, 13)	-2.047	-0.903	-0.844	**-0.844**
mammo_masses	(961, 5)	-2.164	-1.541	-1.277	**-0.698**
spect	(80, 22)	-3.941	-1.200	-0.698	**-0.629**
thyroid_disease	(3772, 21)	-1.340	-0.293	-0.171	**-0.103**

Tabela 5.2: Wyniki RFC i KNN na zbiorach danych złożonych tylko z cech zaproponowanych przez algorytm genetyczny.

Porównując tabele 5.1 i 5.2 warto zwrócić uwagę na pogrubione wartości w ostatnich kolumnach. W ten sposób zaznaczono najlepsze wyniki uzyskane na danym zbiorze danych. Wyraźnie widać, że cechy stworzone przez `GA` i samodzielnie wykorzystane w zadaniach klasyfikacji dają klasyfikatory o lepszej jakości niż algorytmy uczone na danych oryginalnych, a nawet na zbiorach łączących dane pierwotne z nowymi.

W tabeli 5.2 widać, że w przypadku algorytmu `KNN` i zestawu cech stworzonych na podstawie zbioru *Cardiotocography* nie nastąpiła poprawa jako-

ści klasyfikatora. Można przypuszczać, że redukcja wymiaru zadania z 31 cech do 5 spowodowała utratę wartościowych informacji z tego zbioru. Chociaż wyniki w kolejnych kolumnach są coraz lepsze, to niewielka poprawa na przestrzeni 75 generacji nie pozwala zakładać, że wraz ze zwiększeniem ilości iteracji następny wynik będzie lepszy od bazowego.

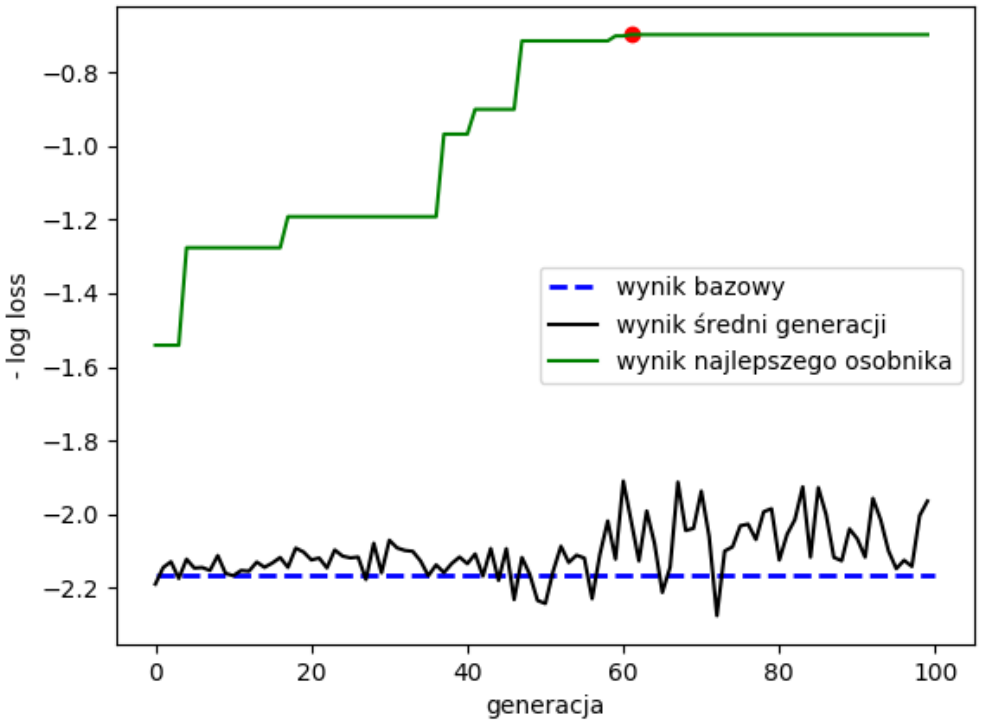


Rysunek 5.1: Wykres wartości funkcji przystosowania populacji dla zbioru *Cardiotocography* z nowymi cechami i algorytmu `RFC`.

Podczas przeprowadzania testów zwrócono uwagę na 2 wykresy. Pierwszy z nich (zestaw danych z kardiotokografii i algorytm `RFC`, Rys. 5.1) przedstawia w zasadzie standardowy rozwój populacji. Średnia wartość funkcji przystosowania w populacji dość szybko i znacząco przekracza wartość *log lossa* uzyskaną na podstawowym zbiorze danych (na wykresie oznaczona przerywaną niebieską linią), można nawet zauważyć powolny trend wzrostowy, ale wartość funkcji dla najlepszego osobnika nie zmienia się od około 50 generacji.

Można przypuszczać, że zostało osiągnięte maksimum (lokalne lub globalne), jeśli to maksimum lokalne, to mutacja jest zbyt słaba, aby znaleźć lepszego osobnika.

Drugi przypadek (`KNN` i dane z mammografii, Rys. 5.2) jest całkowicie odmienny. Wartość średnia funkcji przystosowania danej generacji oscyluje wokół wyniku uzyskanego na zbiorze bez nowych cech, przypomina błądzenia losowe. Bardzo ciekawie wygląda natomiast wykres wartości funkcji przystosowania najlepszego osobnika danej generacji. Krótkie okresy stagnacji przerywane są gwałtownymi skokami. Na tle niezbyt wybitnych populacji co jakiś czas pojawia się osobnik wybitny, jednak jest on za słaby, aby wpłynąć na wynik całej generacji.



Rysunek 5.2: Wykres wartości funkcji przystosowania populacji dla zbioru nowych cech zbudowanych na podstawie *Mammographic Mass* oraz algorytmu `KNN`.

Rozdział 6

Podsumowanie

Celem pracy było zaprojektowanie i zaimplementowanie algorytmu genetycznego ułatwiającego tworzenie nowych cech na podstawie zadanego zbioru danych. Przedstawiono problem jakim jest zadanie klasyfikacji, omówiono wykorzystane w pracy algorytmy klasyfikujące, opisano działanie i możliwe warianty algorytmu genetycznego.

Język Python jako narzędzie do implementacji algorytmu genetycznego spełnił swoją rolę. Nie pozwolił on jednak na stworzenie prawdziwie wydajnego programu. Każdy z osobników danej populacji może być przetwarzany i oceniany niezależnie, dlatego też próbowano zrównoleglić ten proces. Niestety, interpreter języka Python uniemożliwia rozwiązanie tego problemu w pełni satysfakcjonujący sposób z uwagi na tzw. *Global Interpreter Lock* (`GIL`), który nie pozwala na swobodne wykorzystanie wielu rdzeni procesora przez wiele wątków. Jest to jeden z minusów użytej implementacji języka Python, który mimo tego zapewnia dobry kompromis między szybkością pisania programu a jego wydajnością.

Jedną z możliwości dalszego rozwoju modułu jest przepisanie go z wykorzystaniem innego, szybszego języka programowania. Wprowadzenie bardziej adaptacyjnego zasięgu operatora mutacji powinno pozwolić na lepszą optymalizację lokalną przestrzeni rozwiązań, a dodanie osobnikom zmiennego parametru żywotności zapobiegłoby potencjalnie nieskończonemu czasowi życia najlepszych osobników w sukcesji elitarnej.

Cel pracy został zrealizowany, a zaproponowana metoda tworzenia nowych cech może być z powodzeniem stosowana w praktyce.

Spis listingów

Spis rysunków

Spis tabel

Bibliografia

[1] Aha, D. W., Kibler, D., *Instance-based prediction of heart-disease presence with the Cleveland database*, University of California, 1980, `https://archive.ics.uci.edu/ml/datasets/Heart+Disease`.

[2] Altay Guvenir et al., *A Supervised Machine Learning Algorithm for Arrhythmia Analysis*, Proceedings of the Computers in Cardiology Conference, Lund, Sweden, 1997, `https://archive.ics.uci.edu/ml/datasets/Arrhythmia`.

[3] Arabas J., *Wykłady z algorytmów ewolucyjnych*, Wydawnictwo Naukowo-Techniczne, 65-192, 2004.

[4] Ayres de Campos et al., *SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms*, J Matern Fetal Med 5, 311-318, 2000, `https://archive.ics.uci.edu/ml/datasets/Cardiotocography`.

[5] Brabazon, A., O'Neill, M., *Biologically Inspired Algorithms for Financial Modelling*, Springer-Verlag New York, Inc., 175-182, 2006.

[6] Breiman, L., *Bagging predictors*, Machine Learning 24, 123-140, 1996.

[7] Czerniak, J., Zarzycki, H., *Application of rough sets in the presumptive diagnosis of urinary system diseases*, Artifical Inteligence and Security in Computing Systems, ACS'2002 9th International Conference Proceedings, Kluwer Academic Publishers, 41-51, 2003, `https://archive.ics.uci.edu/ml/datasets/Acute+Inflammations`.

[8] Elter, M., Schulz-Wendtland, R., Wittenberg, T., *The prediction of breast cancer biopsy outcomes using two CAD approaches that both emphasize an intelligible decision process*, Medical Physics 34(11), 4164-4172, 2007, `https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass`.

[9] Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., *Do we need hundreds of classifiers to solve real world classification problems?*, Journal of Machine Learning Research 15, 3133-3181, 2014.

[10] Goldberg D., *Algorytmy genetyczne i ich zastosowania*, Wydawnictwo Naukowo-Techniczne, 104-158, 2003.

[11] Ho, TK, *The Random Subspace Method for Constructing Decision Forests*, IEEE Transactions on Pattern Analysis and Machine Intelligence Volume 20, 832-844, 1998.

[12] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[13] Hunter, J. D., *Matplotlib: A 2D graphics environment*, Computing In Science & Engineering, 90-95, 2007, `https://matplotlib.org/`.

[14] James et al., *An Introduction to Statistical Learning: With Applications in R*, Springer Publishing Company, Incorporated, 176-186, 2014.

[15] Kuncheva et al., *Random Subspace Ensembles for fMRI Classification*, IEEE Transactions on Medical Imaging 29, 531–542, 2010.

[16] Kurgan et al., *Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis*, Artificial Intelligence in Medicine, 149-169, 2001, `https://archive.ics.uci.edu/ml/datasets/SPECT+Heart`.

[17] McKerns et al., *Building a framework for predictive science*, Proceedings of the 10th Python in Science Conference, 2011, `http://arxiv.org/pdf/1202.1056`.

[18] McKerns, M., Aivazis, M., *pathos: a framework for heterogeneous computing*, 2010, `http://trac.mystic.cacr.caltech.edu/project/pathos`.

[19] Michalewicz, Z., *Algorytmy genetyczne + struktury danych = programy ewolucyjne*, Wydawnictwo Naukowo-Techniczne, 127-137, 2003.

[20] *NumExpr Documentation Reference*, `https://numexpr.readthedocs.io/en/latest/index.html`.

[21] *NumPy Reference*, `https://docs.scipy.org/doc/numpy/reference/`.

[22] *Python Data Analysis Library*, `https://pandas.pydata.org/`.

[23] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research 12, 2825-2830, 2011, `scikit-learn.org`.

[24] The official home of the Python Programming Language, `https://www.python.org`.

[25] Quinlan, J. R., *Induction of decision trees*, Machine Learning, 1, 81-106, 1986, `https://archive.ics.uci.edu/ml/datasets/Thyroid+Disease`.

[26] Sammut, C., Webb, G. I., *Encyclopedia of Machine Learning*, Springer Publishing Company, Incorporated, 168-170, 2011.

[27] Schapire, R. E., *The Strength of Weak Learnability*, Machine Learning 5, 197-227, 1990.

[28] *Six: Python 2 and 3 Compatibility Library*, `https://pythonhosted.org/six/`.

[29] Smith M.G., Bull L., *Feature Construction and Selection Using Genetic Programming and a Genetic Algorithm*, Genetic Programming. EuroGP 2003. Lecture Notes in Computer Science, 2003.

[30] Smith, M., VeeramachaneniK., *FeatureHub: Towards Collaborative Data Science*, MIT, 2017.

[31] Street, W. N., Wolberg, W. H., Mangasarian, O. L., *Nuclear feature extraction for breast tumor diagnosis*, IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, 861-870, 1993, `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`.

[32] *tqdm*, `https://github.com/noamraph/tqdm`.

[33] UC Irvine Machine Learning Repository, `https://archive.ics.uci.edu/ml/index.php`.

[34] *Top 10 causes of death worldwide*, `http://who.int/mediacentre/factsheets/fs310/en/`.

[35] Wolpert, D. H., *The Lack of A Priori Distinctions Between Learning Algorithms*, Neural Computation 8, 1341-1390, 1996.

[36] Wu et al., *Top 10 algorithms in data mining*, Knowledge and Information Systems 14, 22-24, 2007.