Sprawozdanie z eksperymentu z przedmiotu Metaheurystyki i Obliczenia Inspirowane Biologicznie

3 grudnia 2012

Prowadzący: dr inż. Maciej Komosiński

Autorzy: **Tomasz Ziętkiewicz** inf84914 ISWD tomek.zietkiewicz@gmail.com

Zajęcia poniedziałkowe, 15:10.

Streszczenie

Użycie klasyfikatora *SVM* wymaga doboru odpowiedniej do danego problemu funkcji jądrowej. Jej wybór nie jest zadaniem trywialnym i może decydować o sprawności klasyfikatora. Postanowiono zbadać możliwość użycia programowania genetycznego w celu wygenerowania optymalnych funkcji jądrowych. Z użyciem biblioteki *ECJ* stworzono algorytm programowania genetycznego, który generował poprawne funkcje jądrowe. Wartość ich przystosowania była sprawdzana za pomocą biblioteki *LibSVM*. Wyniki pokazują, że w przypadku pewnych zbiorów danych optymalizacja funkcji jądrowej ma znaczący wpływ na trafność klasyfikacji. Optymalne wyniki uzyskiwano już dla przebiegu trwającego jedną generację co sugeruje możliwość dalszej optymalizacji algorytmu.

1 Wstęp

1.1 Opis eksperymentu

Celem eksperymentu było użycie programowania genetycznego do wyewoluowania optymalnych funkcji jądrowych dla klasyfikatora *SVM*.

Klasyfikator SVM dokonuje klasyfikacji binarnej oddzielając od siebie dwie grupy przykładów hiperpłaszczyzną przebiegającą w przestrzeni atrybutów opisujących przykłady. Najczęściej grupy te nie są liniowo separowalne i trzeba dokonać transformacji cech opisujących przykłady do przestrzeni o większej liczbie wymiarów tak, żeby były w niej liniowo separowalne. Funkcje używane do dokonania tej transformacji to funkcje jądrowe (inczej kernele - ang. kernel functions). Wybór odpowiedniej funkcji zależy od rozwiązywanego problemu i zazwyczaj opiera się na doświadczeniu osoby używającej klasyfikator, posiadanej przez nią wiedzy dziedzinowej. W przypadku nie znanych *apriori* danych wejściowych i braku doświadczenia w wyborze optymalnej funkcji jądrowej można posłużyć się automatycznymi metodami optymalizacji. Wśród nich idalną do tego zadania wydaje się *programowanie geetyczne* (*GP* - ang. *Genetic Programming*). Jest to szczególny rodzaj *obliczeń ewolucyjnych* (*EC* - ang. *Evolutionary Computing*), w którym ewoluowane osobniki to funkcje reprezentowane za pomocą struktur drzewiastych.

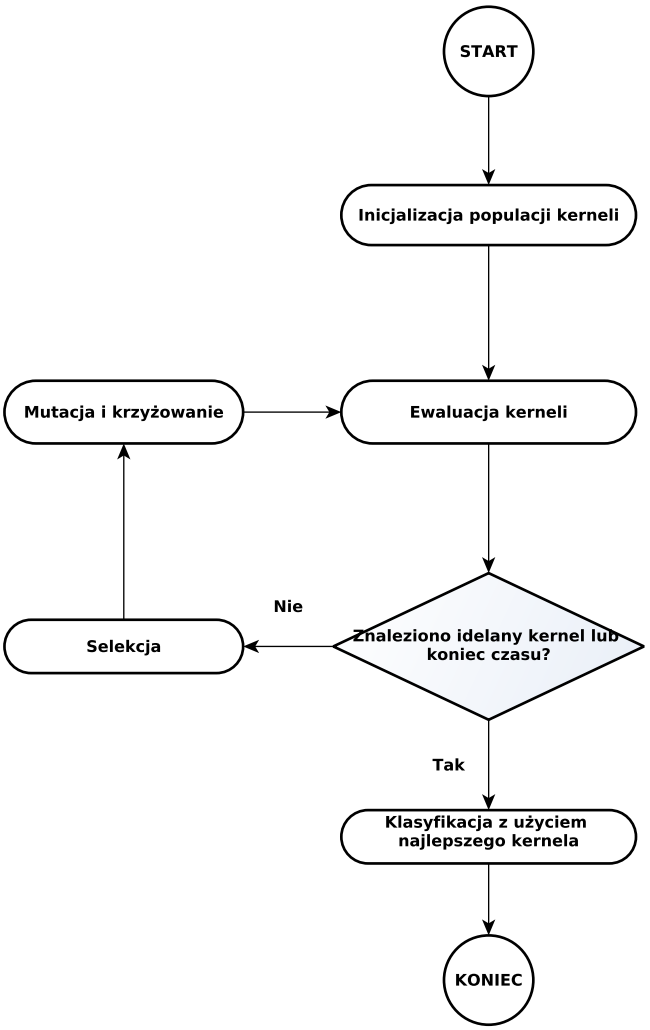
Na potrzeby eksperymentu zaprojektowano algorytm programowania genetycznego, który ewoluował funkcje jądrowe poprzez łączenie ze sobą prostych funkcji jądrowych w funkcje złożone za pomocą funkcji łączących. Co ważne taki sposób generowania funkcji jądrowych zapewnia, że będą one poprawnymi funkcjami jądrowymi [3]. Jest to własność domknięcia (ang. *closure*) zbioru funkcji jądrowych ze względu na pewne operacje (funkcje łączące).

1.2 Opis algorytmu

Przebieg algorytmu jest typowy dla algorytmów genetycznych:

1. Utwórz początkową populację kerneli
2. Oblicz wartość *funkcji dopasowania* każdego z kerneli: dokładność klasyfikacji SVM z użyciem tego kernela
3. Jeśli znaleziono idealny kernel (dokładność klasyfikacji 100) lub skończył się czas, użyj tego kernela do klasyfikacji zbioru walidującego, zwróć wyniki klasyfikacji i zakończ algorytm.
4. Dokonaj selekcji najlepszych funkcji z populacji
5. Utwórz nową populację poprzez mutację i krzyżowanie wybranych w poprzednim kroku funkcji

6. Wróć do punktu 2



Rysunek 1: Diagram przepływu algorytmu Kernel GP.

Algorytm pokazano również na diagramie przepływu na rycinie 1. Poszczególne kroki algorytmu zostaną opisane poniżej.

1.3 Inicjalizacja populacji

Podczas inicjalizacji początkowo pusta populacja jest zapełniana przez generowane w sposób losowy drzewa reprezentujące funkcje. Generowane drzewa muszą być poprawne, czyli spełniać narzucone ograniczenia na głębokość drzewa, liczbę węzłów, typ wartości zwracanych przez drzewo. Wielkość populacji jest jednym z parametrów algorytmu. Zbyt mała populacja powoduje losowe zawężenie przeszukiwanej przestrzeni i zmniejsza prawdopodobieństwo znalezienia optymalnej funkcji. Z drugiej strony zbyt duża wielkość populacji upodabnia

algorytm genetyczny do pełnego przeszukiwania, co oczywiście zwiększa szanse znalezienia optymalnego kernela, ale wydłuża czas działania algorytmu.

1.3.1 Generowanie funkcji

Generowanie drzew reprezentujących funkcje jądrowe polega na łączeniu ze sobą funkcji elementarnych zgodnie z przypisanymi im ograniczeniami. Funkcje elementarne wraz z ograniczeniami zdefiniowane w algorytmie:

- Funkcje łączące - jako argument przyjmują wynik dwóch lub jednej funkcji jądrowej i ewentualnie stałą *ERC*. Zwracają wartość rzeczywistą. Dzięki właściwości domknięcia zbioru kerneli ze względu na operacje wykonywane przez te funkcje funkcja powstała przez połączenie dwóch kerneli funkcją łączącą jest również poprawnym kernelem [3].
  - Dodawanie: $k(x,z) = k_1(x,z) + k_2(x,z)$
  - Mnożenie: $k(x,z) = k_1(x,z) * k_2(x,z)$
  - Mnożenie przez stałą: $k(x,z) = a * k_1(x,z)$
  - Funkcja wykładnicza: $k(x,z) = e^{k_1(x,z)}$

  Gdzie a to stała rzeczywista generowana jako stała *ERC*.
- Podstawowe funkcje jądrowe - jako argument przyjmują odpowiednią do funkcji liczbę stałych ERC. Zwracają wartość rzeczywistą.
  - Liniowa: $k(x,z) = \langle x,z \rangle$
  - Wielomianowa: $k(x,z) = \langle x,z \rangle^d$
  - Gausowska: $e^{-\gamma * ||x-z||^2}$
  - Sigmoidalna: $k(x,z) = \text{tgh}(\gamma \langle x,z \rangle + \tau)$

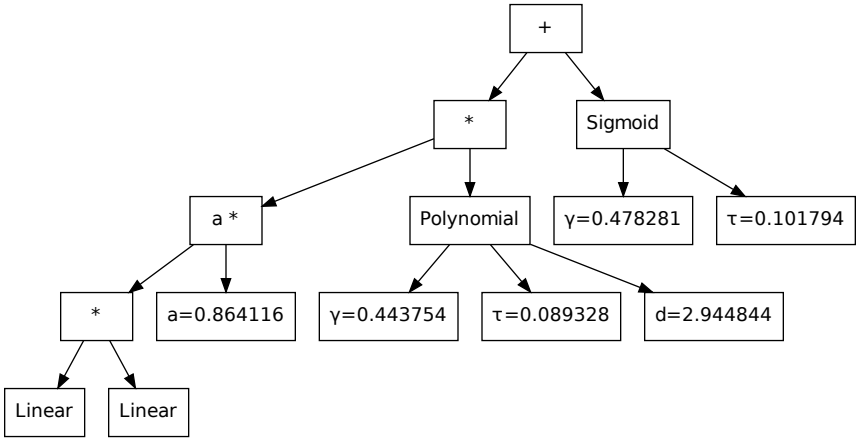
  Gdzie γ, τ oraz d to wartości stałe generowane jako stałe *ERC*. a $\langle x,y \rangle$ to iloczyn skalarny wektorów x i y.
- Stałe *ERC* (ang. *Ephemeral Random Constant*) liczby rzeczywiste lub całkowite, które służą jako parametry innych funkcji. Są one liściami w drzewie, nie przyjmują żadnych argumentów. Mogą losowo zmieniać swoją wartość podczas mutacji.
  - γ: liczba rzeczywista z zakresu
  - τ: liczba rzeczywista z zakresu
  - d: liczba całkowita z zakresu
  - a: liczba rzeczywista z zakresu

Przykładowe drzewo wygenerowane przez algorytm pokazana na ryc.2.

Wektory cech będące najważniejszymi argumentami funkcji jądrowych nie są wyodrębnione jako osobne węzły budujące drzewo, ponieważ są one danymi wejściowymi ewoluowanych funkcji i dla każdej ewoluowanej funkcji są takie same.

1.4 Ewaluacja kerneli

Ewaluacja funkcji jądrowej polega na jej zastosowaniu w algorytmie SVM do klasyfikacji zbioru testowego. Otrzymana trafność stanowi miarę przystosowania (ang. *fitness*) danej funkcji.



Rysunek 2: Przykładowe drzewo generowane przez algorytm.

1.5 Selekcja

Jednym z problemów programowania genetycznego jest to, że drzewa powstałe w wyniku procesu ewolucyjnego mogą być bardzo duże, co nie jest pożądaną cechą - większe drzewo dłużej oblicza zwracaną wartość, zajmuje więcej miejsc w pamięci. Dlatego wielkość drzew należy ograniczać, jeśli wzrost drzewa nie prowadzi do zwiększenia wartości funkcji dopasowania. Wielkość generowanych drzew jest regulowana przez dwa mechanizmy. Pierwszy to proste ograniczenie na maksymalną głębokość drzewa. Wartość tę ustawiono na 6 - drzewa o większej głebokości nie zostaną w ogóle wygenerowane przez podczas inicajlizacji populacji czy podczas krzyżowania i mutacji. Drugi mechanizm, o angielskiej nazwie *parsimony pressure*, promuje mniejsze drzewa podczas selekcji. W tym celu stosowany jest algorytm selekcji turniejowej leksykograficznej z koszykami (ang. Bucket Lexicographic Tournament Selection). Algorytm ten sortuje populację według przystosowania osobników, następnie grupuje je w N "koszyki". Następnie selekcja przebiega według zasad selekcji turniejowej, z tym, że porównuje się nie przystosowanie osobników, ale koszyk, do którego są przypisane. W przypadku gdy w turnieju porównywane są dwa osobniki z tego samego koszyka wygrywa ten, który jest mniejszy.

1.6 Krzyżowanie i mutacja

Krzyżowanie polega na odcięciu dwóch losowych poddrzew z dwóch różnych osobników i zamianie ich miejscami. Wygenerowane w ten sposób drzewo musi spełniać narzucone na drzewo ograniczenia dotyczące typów i wielkości. Mutacja drzew polega na zamianie losowo wybranego poddrzewa przez losowo wygenerowane drzewo. Dodatkowo mutowane są również węzły ERC. Ich mutacja polega na dodaniu losowej wartości o rozkładzie normalnym do wartości przechowywanej w węźle. Wartość ta może być ujemna lub dodatnia. Mutacji podlega 90% drzew, pozostałe 10% jest pozostawiane bez zmian. Wszystkie drzewa podlegające mutacji mają mutowane węzły ERC. Ponadto 70% z nich podlega krzyżowaniu, 20% mutacji polegającej na generowaniu losowych poddrzew a 10% ma mutowane jedynie węzły ERC.

1.7 Walidacja rozwiązania

Walidacja polega na użyciu najlepszego znalezionego kernela do klasyfikacji przykładów ze zbioru walidującego, który nie był używany podczas uczenia klasyfikatora SVM ani podczas ewaluacji kerneli. Najpierw algorytm SVM jest uczony na połączonych zbiorach trenującym i walidującym, przy pomocy tej funkcji jądrowej. Następnie dokonywana jest klasyfikacja zbioru walidującego. Otrzymana w wyniku tej klasyfikacji trafność jest miarą oceny całego algorytmu Kernel GP.

2 Implementacja

Algorytm został napisany w języku Java z użyciem bibliotek *ECJ (Evolutionary Computing in Java)* [2] oraz LibSVM [1]. Pierwsza z nich dostarcza mechanizmy *obliczeń ewolucyjnych* w tym *programowania genetycznego*. *LibSVM* to klasyfikator SVM napisany oryginalnie w języku C z dostępną implementacją w Javie. Mechanizmy *ECJ* stanowią trzon algorytmu zapewniając tworzenie populacji funkcji, ich selekcję, mutację oraz krzyżowanie. *LibSVM* został użyty na etapie ewaluacji wygenerowanych przez *ECJ* funkcji.

3 Wyniki

3.1 Zbiory danych

Do oceny pracy algorytmu użyto standardowych zbiorów danych służących do testowania systemów maszynowego uczenia się, dostępnych na stronie biblioteki *LIBSVM* [1]. Zbiory zostały opisane w tabelce 1.

Nazwa zbioru	Liczba klas	Liczba atrybutów	Wielkość zbioru uczącego	Wielkość zbioru testującego	Wielkość zbioru walidującego
Iris	3	4	68	33	49
Letter	26	16	9000	4400	6600
DNA	3	180	1435	700	1051
Vowel	11	10	447	217	326

Tablica 1: Zbiory danych użyte do testowania systemu.

3.2 Metodologia pomiarów

Żeby oszacować trafność klasyfikacji osiąganą przez skonstruowany system konieczne było podzielenie zbioru danych na zbiór uczący i walidujący, a w przypadku algorytmu Kernel-GP również wydzielenie ze zbioru uczącego podzbioru testującego, używanego do obliczania miary przystosowania (fitness) podczas przebiegu algorytmu genetycznego. Ponieważ sposób podziału zbioru danych ma wpływ na osiąganą trafność klasyfikacji, dokonywano 5 takich podziałów a następnie wyciągano średnią oraz odchylenie standardowe z wyników otrzymanych dla tych podziałów. Ta procedura dotyczyła zarówno testowania algorytmu *Kernel-GP* jak i porównawczych testów klasyfikatora SVM z biblioteki *LibSVM*. Dla obu algorytmów stosowano te same podziały danych, przy czym w przypadku klasyfikatora *LibSVM* nie dzielono zbioru uczącego na trenujący i testujący.

Jeśli nie zaznaczono inaczej, w poniższym omówieniu wyników słowa "dokładność"lub "trafność"klasyfikacji odnoszą się do trafności klasyfikacji danych ze zbioru walidującego.

Algorytm genetyczny jest w swej naturze stochastyczny, korzysta więc z funkcji generujących liczby pseudolosowe. Aby zapewnić powtarzalność wyników i umożliwić ich porównanie ziarno generatora liczb pseudolosowych ustawiono na stałą wartość.

Aby ocenić skuteczność algorytmu genetycznego w poszukiwaniu optymalnych funkcji jądrowych oraz oszacować optymalną wielkość populacji oraz czas trwania (liczbę ewaluowanych generacji) algorytmu przeprowadzono szereg eksperymentów obliczeniowych, w których uruchamiano algorytm dla coraz to większych wartości tych parametrów. Dla każdego przebiegu algorytmu zapisywano trafność klasyfikacji zbioru walidującego.

Analizując tak zebrane dane można przeanalizować na ile poszukiwanie funkcji jądrowej przez algorytm genetyczny było podobne do losowego przeszukiwania a na ile było ono zbieżne. W pierwszym przypadku na wyniki osiągane przez algorytm powinna mieć wpływ przede wszystkim wielkość populacji, w drugim również liczba generacji przez które poszukiwano rozwiązania. W szczególności ciekawym przypadkiem jest ten, gdy liczba generacji wynosi 1, czyli cały algorytm ogranicza się do wygenerowania populacji losowych osobników i wybrania jednego z nich - w tym przypadku algorytm genetyczny sprowadza się do losowego poszukiwania rozwiązania. Porównując różnicę w trafności osiąganej w trakcie jednej generacji i coraz większej ich liczby można ocenić czy proces ewolucyjny przebiega poprawnie.

3.3 Trafność klasyfikacji

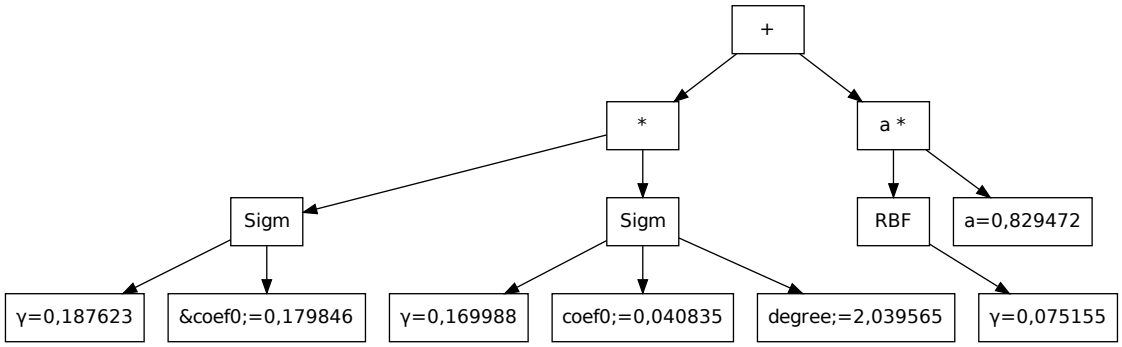
Jak widać na rysunkach 8-17 sprawność algorytmu zależy mocno od konkretnego zbioru danych. Na przykład zbiór IRIS jest na tyle łatwy w klasyfikacji, że ciężko jest poprzez dobranie optymalnej funkcji jądrowej polepszyć znacznie wyniki klasyfikacji. Na osiąganą trafność klasyfikacji wpływ ma też podział zbioru danych na testujący, uczący i walidujący - widać to po odchyleniu standardowym widocznym na wykresach.

3.3.1 Monotoniczność funkcji trafności

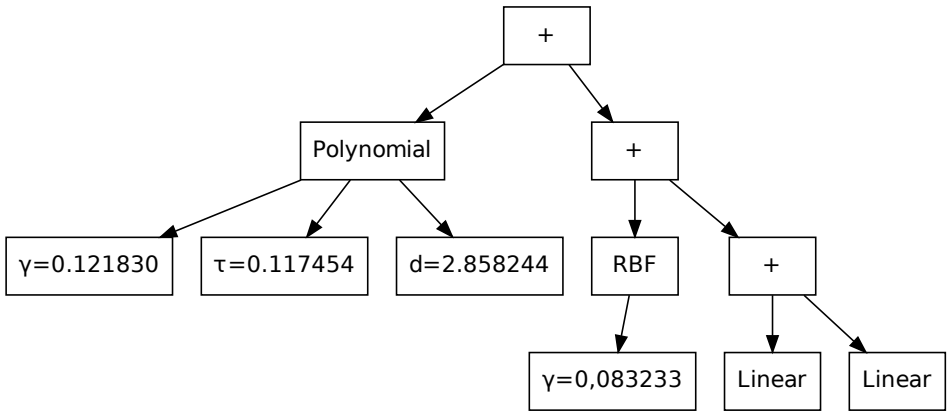
Miejscami funkcja trafności nie jest monotoniczna, a ściślej niemalejąca, względem liczby generacji oraz wielkości populacji (co widać np. na wykresach 9 i 14). Wydawałoby się, że tak być nie powinno (algorytm genetyczny zwraca najlepszego osobnika z całego swojego przebiegu, więc wszystkie osobniki, które pojawiły się podczas przebiegu z 5 generacjami pojawią się podczas przebiegu z 7 generacjami, więc trafność dla przebiegu z 7 generacjami powinna być co najmniej tak dobra jak dla przebiegu z 5 generacjami). Jednak może się tak zdarzyć ze względu na to, że trafność pokazana na wykresach to trafność klasyfikacji zbioru walidującego, natomiast trafność użyta przez algorytm genetyczny jako miara dostosowania (ang. *fitness*) to trafność klasyfikacji zbioru testującego. Widać to na wykresie 10, który przedstawia wartość przystosowania dla tych samych danych, dla których na wykresie 9 jest pokazana trafność klasyfikacji na zbiorze walidującym - tutaj funkcja wykazuje mniej braku monotoniczności.

Zatem przynajmniej część braku monotoniczności funkcji trafności na zbiorze walidującym wynika z przeuczenia algorytmu - znaleziona przez algorytm genetyczny funkcja jądrowa lepiej sprawdza się przy klasyfikacji zbioru testującego niż walidującego. Nie jest to jednak jedyna przyczyna braku monotoniczności - widać to na wykresie 15 przedstawiającym wartość funkcji przystosowania dla zbioru *vowel* - jej przebieg jest bardzo podobny do przebiegu ukazanej na rys. 14 funkcji trafności klasyfikacji zbioru walidującego na tym samym zbiorze. Co więc jest przyczyną braku monotoniczności? Warto zauważyć, że funkcja jest niemalejąca ze względu na ilość generacji oraz że dla jednej generacji funkcja jest monotoniczna. Sugeruje

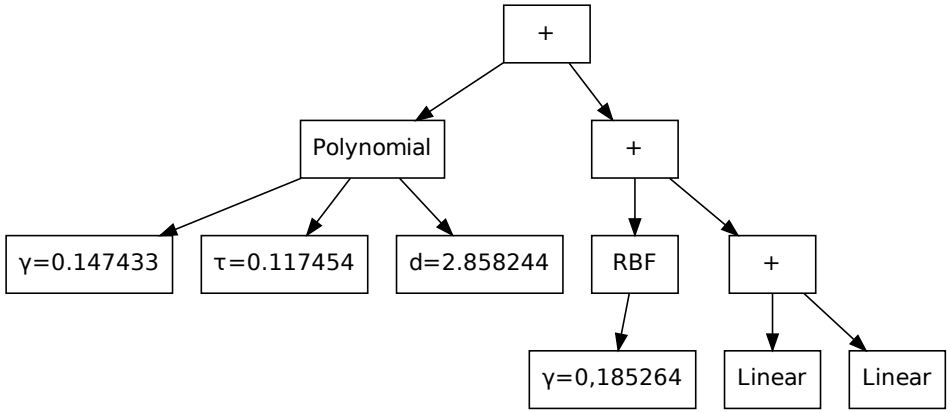
to, że "winnym"może być selekcja - w praktyce nie zachodzi ono w przypadku gdy algorytm genetyczny działa przez jedną generację.



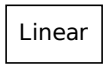
Rysunek 3: Funkcja z pierwszej generacji, która w przebiego z wielkością populacji 4 osiągnęła fitness 0.4242424. Przodek funkcji z rys.4



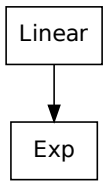
Rysunek 4: Funkcja z trzeciej generacji, która w przebiego z wielkością populacji 4 osiągnęła fitness 0.78030306. Potomek funkcji z rys.3, przodek zwycięskiej funkcji (rys.5) z przebiegu z populacją o wielkości 4.



Rysunek 5: Funkcja z ostatniej generacji w przebiegu z wielkością populacji 4 osiągnęła fitness 0.8333333. Przodek zwycięskiej funkcji z rys.7



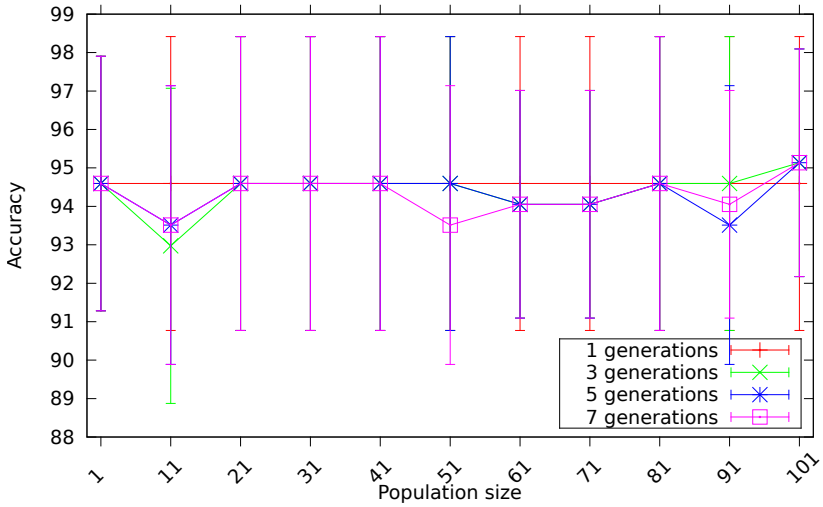
Rysunek 6: Funkcja z trzeciej generacji, która w przebiegu z wielkością populacji 3 i 4 osiągnęła fitness 0.6515151. Przodek zwycięskiej funkcji z rys.7



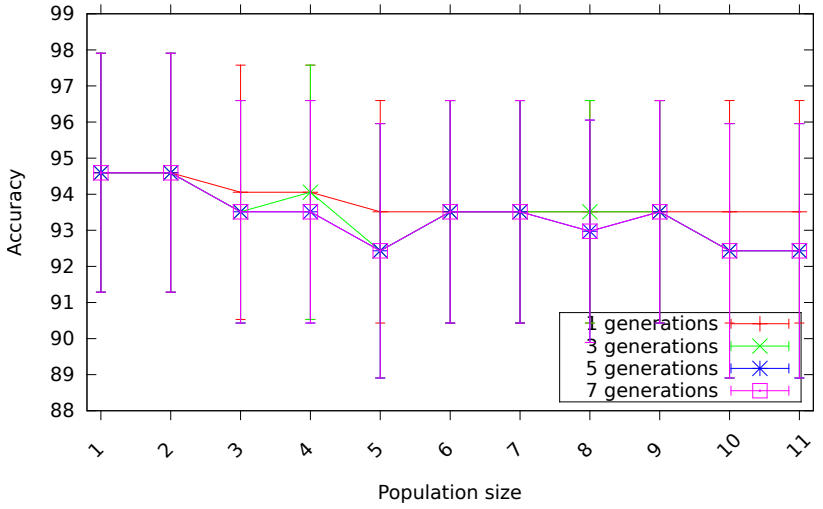
Rysunek 7: Zwycięska funkcja w przebiegu z populacją wielkości 3, potomek funkcji z rys.6. Osiągnęła fitness 0.9015151.

Gdy przyjrzeć się dokładnie przebiegowi ewolucji widać, że rzeczywiści tak jest. Dodatkowy osobnik (rys. 3), który odróżnia w generacji pierwszej populacje o wielkości 3 i 4 jest przodkiem innego osobnika (rys. 4), który w trzeciej generacji osiąga fitness większy niż osobnik (rys. 6), który w przypadku populacji wielkości 3 był przodkiem osobnika (rys.

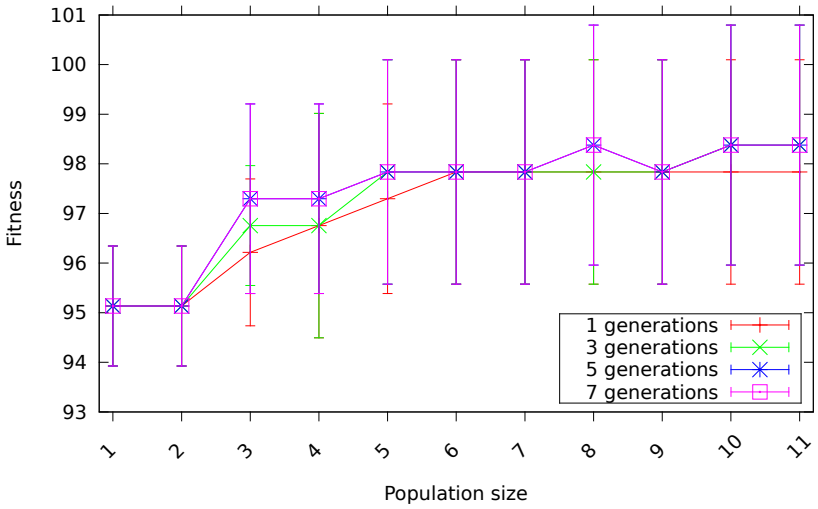
7), który to okazał się najlepszym podczas przebiegu całego algorytmu. W rezultacie tego "geny"potencjalnego zwycięzcy nie przetrwały w przebiegu algorytmu z populacją liczącą 4 osobników. Jak widać osobnik najlepszy we wszystkich generacjach nie musi być wcale potomkiem osobników najlepszych w poszczególnych generacjach - czasem połączenie dwóch osobników przeciętnych może dać osobnika bardzo dobrego.



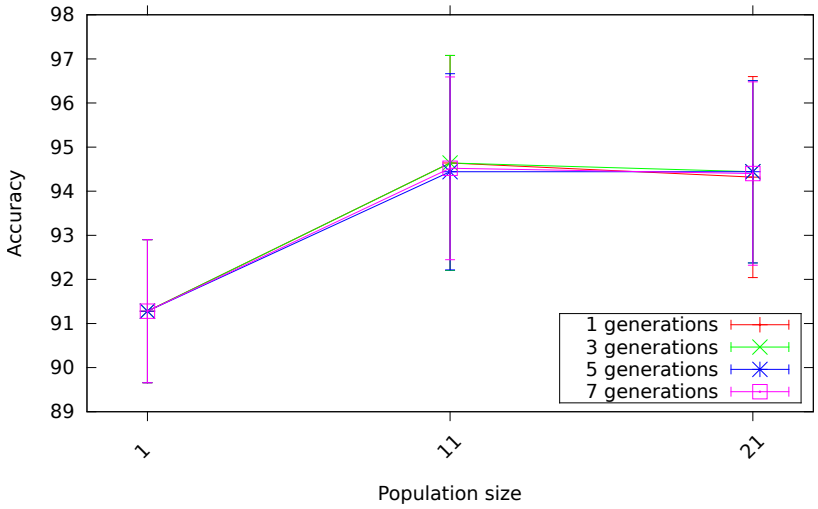
Rysunek 8: Dokładność klasyfikacji dla zbioru *iris* w funkcji rozmiaru populacji dla róznych ilości generacji.



Rysunek 9: Dokładność klasyfikacji dla zbioru *iris* w funkcji rozmiaru populacji dla róznych ilości generacji, dla małych populacji.



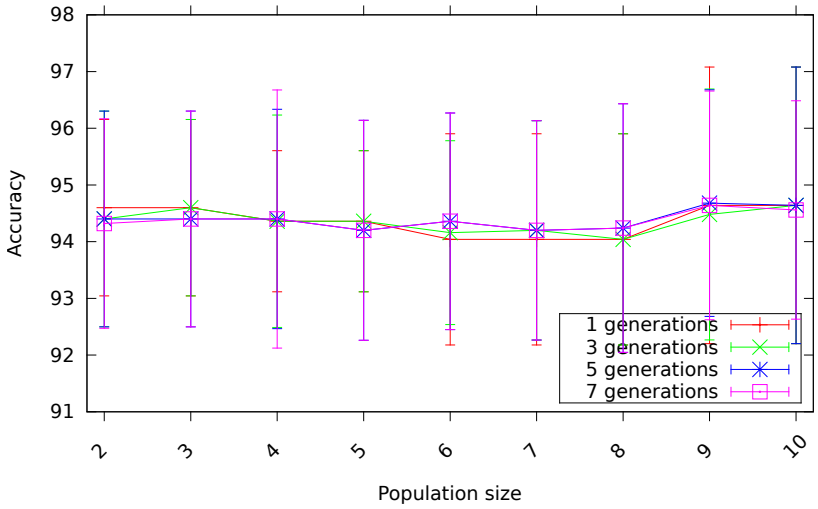
Rysunek 10: Najlepsza wartość funkcji przystosowania (ang. *fitness*) *iris* w funkcji rozmiaru populacji dla róznych ilości generacji, dla małych populacji.



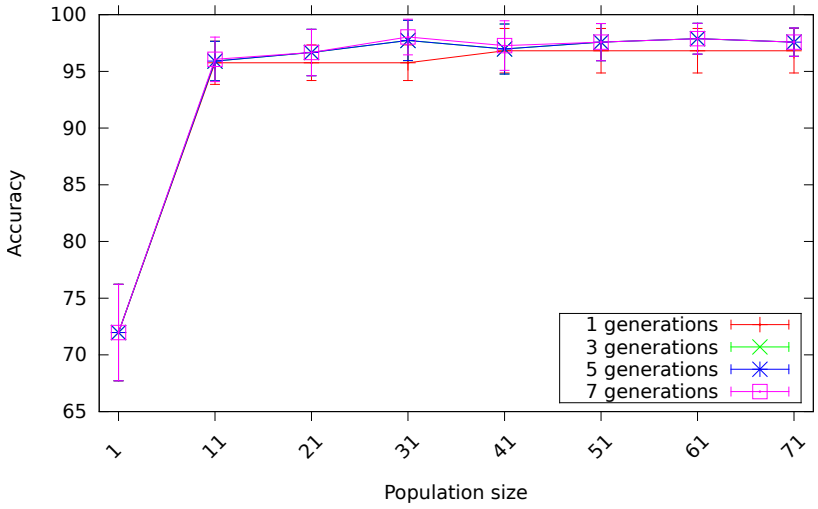
Rysunek 11: Dokładność klasyfikacji dla zbioru *DNA* w funkcji rozmiaru populacji dla róznych ilości generacji.

3.4 Porównanie z tradycyjnym algorytmem SVM

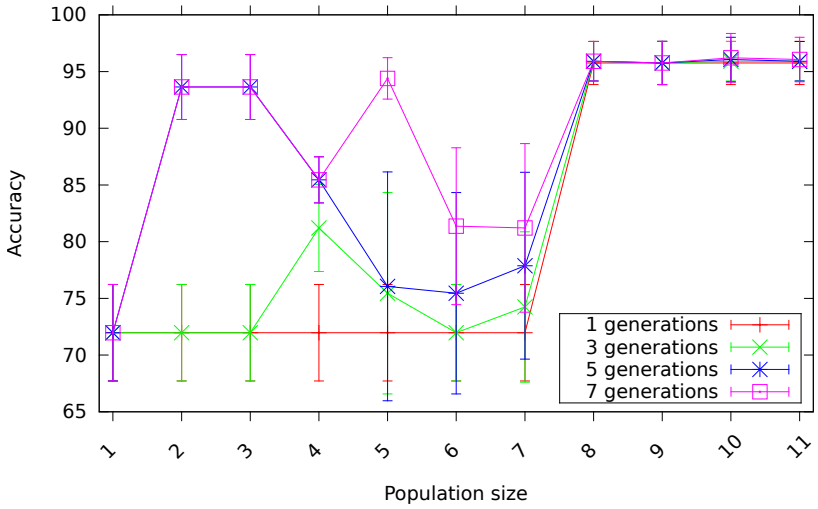
Na wykresach 18-21 przedstawiono porównanie trafności klasyfikacji zbioru walidującego przez algorytm SVM z użyciem czterech podstawowych funkcji jądrowych (liniowej, wielomianowej, RBF i sigmoidalnej) i przez stworzony algorytm Kernel-GP. W przypadku dwóch zbiorów: *vowel* i *letter* udało się uzyskać polepszenie trafności klasyfikacji względem standardowych funkcji jądrowych. Są to zbiory, dla których standardoy SVM osiąga słabe wyniki - ok. $70 W przypadku dwch pozostaych zbiorw nie zalenie od uytej funkcji jdrowej osiga nat rafno klasyfikacji jest bardzo wys$ *ok*. 95



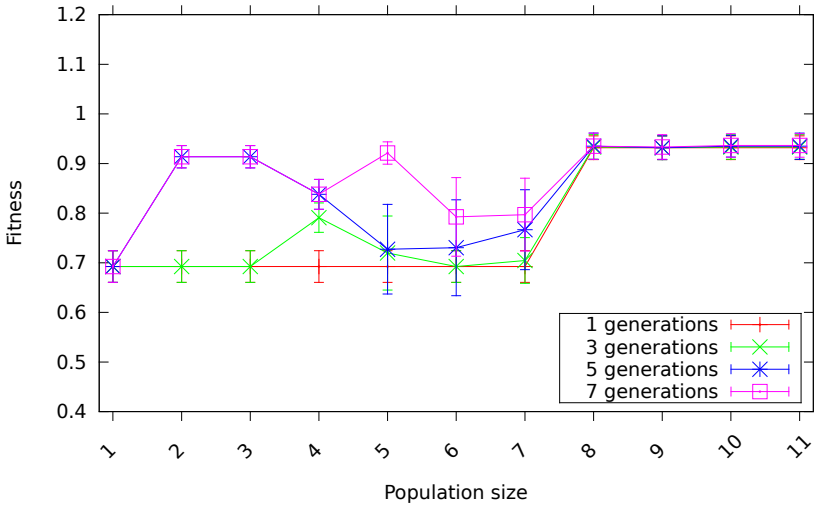
Rysunek 12: Dokładność klasyfikacji dla zbioru *DNA* w funkcji rozmiaru populacji dla róznych ilości generacji, dla małych populacji.



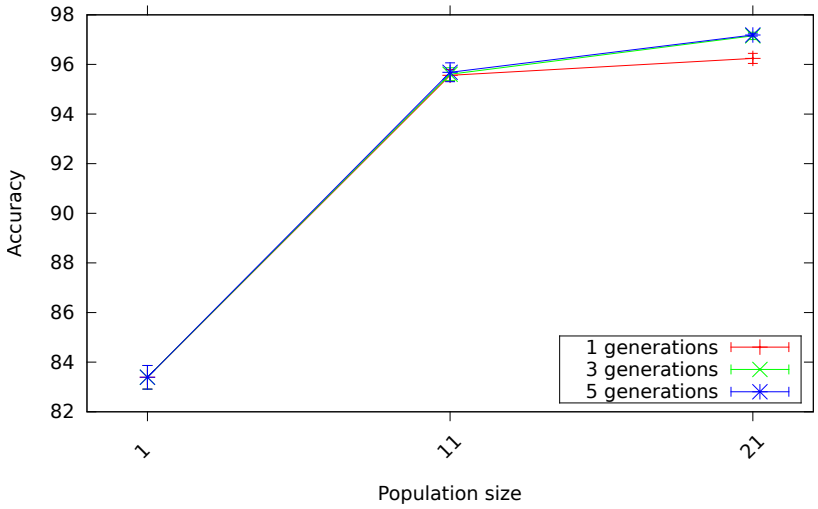
Rysunek 13: Dokładność klasyfikacji dla zbioru *vowel* w funkcji rozmiaru populacji dla róznych ilości generacji.



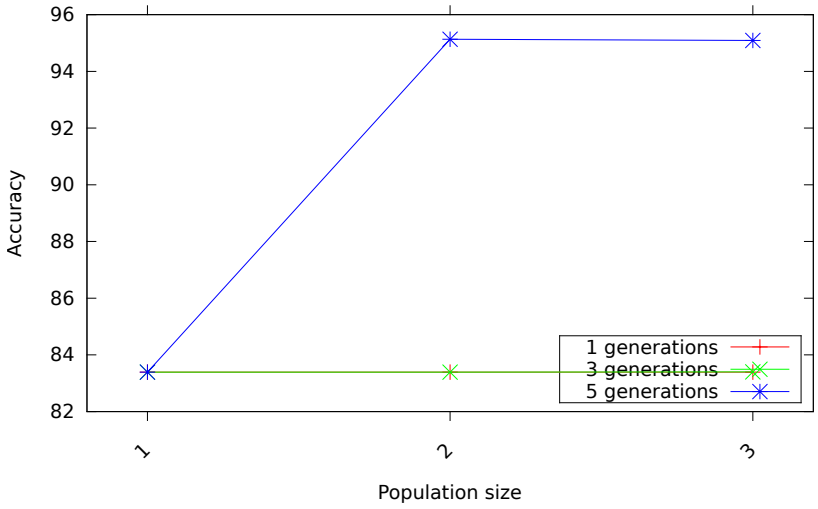
Rysunek 14: Dokładność klasyfikacji dla zbioru *vowel* w funkcji rozmiaru populacji dla różnych ilości generacji, dla małych populacji.



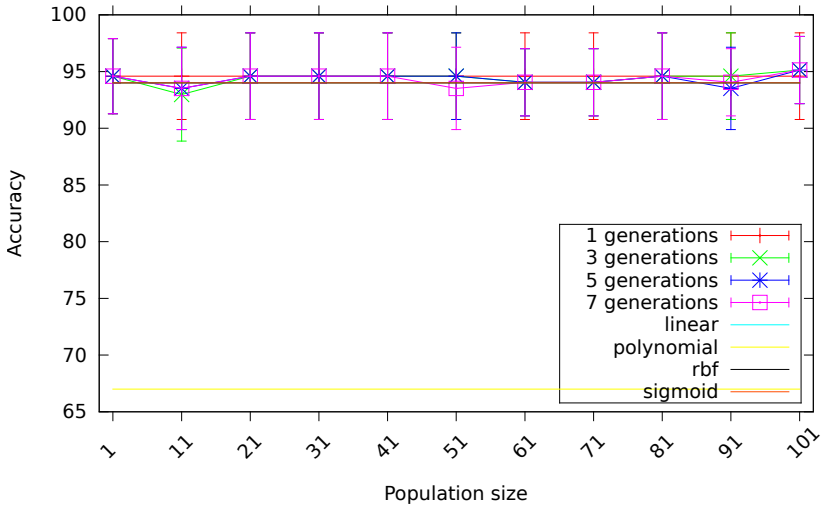
Rysunek 15: Dokładność klasyfikacji dla zbioru *vowel* w funkcji rozmiaru populacji dla różnych ilości generacji, dla małych populacji.



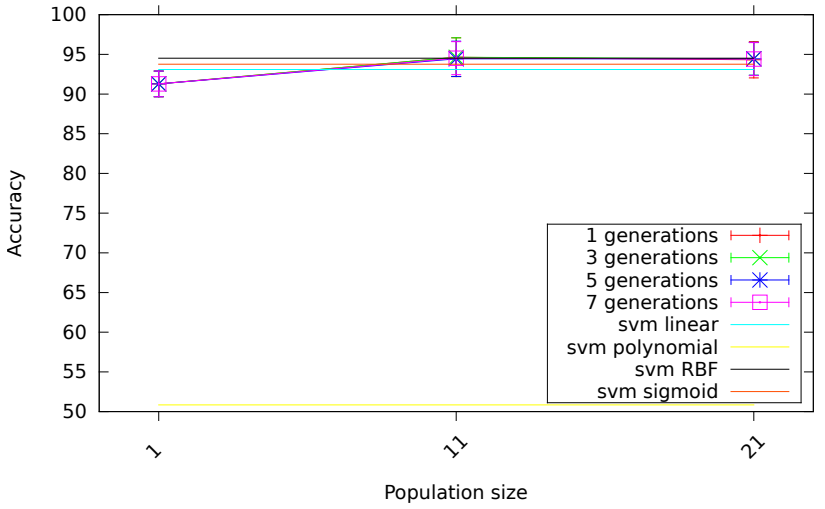
Rysunek 16: Dokładność klasyfikacji dla zbioru *letter* w funkcji rozmiaru populacji dla różnych ilości generacji.



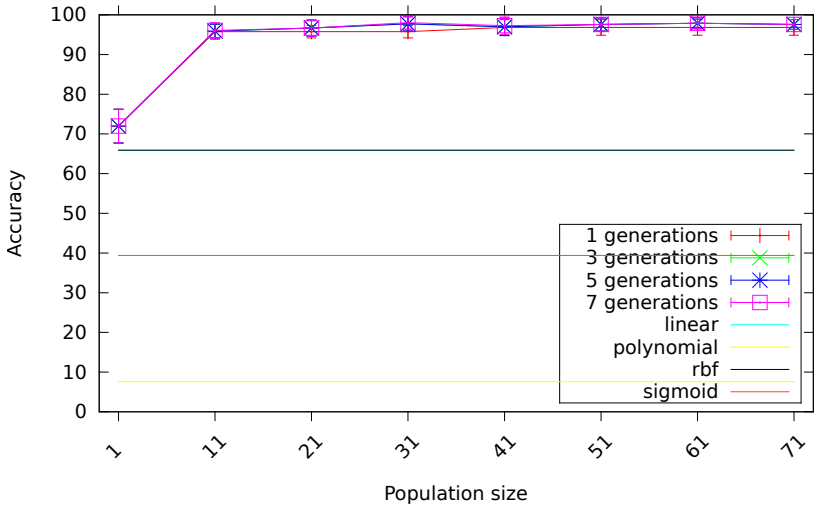
Rysunek 17: Dokładność klasyfikacji dla zbioru *letter* w funkcji rozmiaru populacji dla różnych ilości generacji, dla małych populacji.



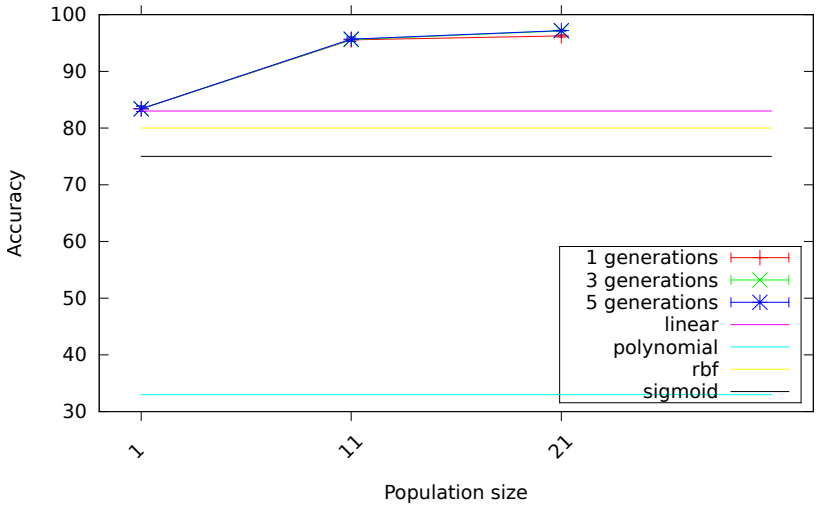
Rysunek 18: Porównanie dokładności klasyfikacji dla zbioru *iris* przez algorytm SVM z różnymi funkcjami jądrowymi i algorytm Kernel-GP



Rysunek 19: Porównanie dokładności klasyfikacji dla zbioru *dna* przez algorytm SVM z różnymi funkcjami jądrowymi i algorytm Kernel-GP.



Rysunek 20: Porównanie dokładności klasyfikacji dla zbioru *vowel* przez algorytm SVM z różnymi funkcjami jądrowymi i algorytm Kernel-GP



Rysunek 21: Porównanie dokładności klasyfikacji dla zbioru *letter* przez algorytm SVM z różnymi funkcjami jądrowymi i algorytm Kernel-GP

4 Podsumowanie

4.1 Wnioski

Algorytm programowania genetycznego pozwala znaleźć optymalne funkcje jądrowe dla klasyfikatora SVM. W przypadku trudnych zbiorów różnice w trafności klasyfikacji mogą być znaczące. Kosztem jaki trzeba zapłacić za polepszenie jakości klasyfikacji jest czas obliczeń, który wzrasta wielokrotnie, choć należy pamiętać, że raz wyewoluowanej funkcji można używać wielokrotnie dla nowych danych.

Rozważane w rozdziale 3.3 różnice między trafnością osiągana dla różnych czasów trwania algorytmu (ilości ewoluowanych generacji) są mało znaczące i widoczne tylko dla bardzo małych populacji. Dla większych populacji okazuje się, że wyniki są tak samo dobre dla jednej jak i dla wielu generacji. To niestety pokazuje, że algorytm programowania genetycznego w tym przypadku upodobnił się do losowego przeszukiwania - skoro przy odpowiednio dużej (nawet tak mało jak 4-8) wielkości populacji liczba generacji nie ma znaczenia dla osiąganego wyniku - nawet dla jednej generacji wyniki są takie same jak dla większej ich liczby - oznacza to, że mechanizm krzyżowania nie spełnia kluczowej roli w poszukiwaniu rozwiązań - wystarczy wygenerować odpowiednio dużą liczbę losowych funkcji jądrowych. Może być to spowodowane tym, że przeszukiwany krajobraz rozwiązań (funkcji jądrowych) jest dość jednolity, cechuje się małą zmiennością - choć przestrzeń rozwiązań jest bardzo duża, to są one zgrupowane w zbiory funkcji nie różniących się od siebie znacznie pod względem trafności klasyfikacji. Możliwe, że należałoby zmienić parametry procesu ewolucyjnego tak, żeby poszerzyć zakres generowanych funkcji, np. poluzować ograniczenie na głębokość funkcji. Mała liczba przebadanych zbiorów danych również może być przyczyną - możliwe, że dla innych zbiorów, mniej standardowych, które trudniej rozdzielić przy pomocy standardowych funkcji SVM, wyniki byłyby inne. Wreszcie można by poszerzyć zbiór funkcji bazowych, z których algorytm genetyczny generuje funkcje bardziej złożone - należałoby jednak zadbać o zachowanie ich poprawności.

Literatura

[1] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[2] Luke Sean. *The ECJ Owner's Manual.* Yale Univ Pr, October 2010.

[3] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004.