Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

OPTYMALIZACJA KLASYFIKATORA SVM ZA POMOCĄ PROGRAMOWANIA GENETYCZNEGO

Tomasz Ziętkiewicz

Promotor
dr hab. Krzysztof Krawiec

Poznań, 2013

Spis treści

Rozdział 1

Wprowadzenie

1.1 Cel i zakres pracy

Niniejsza praca ma dwa podstawowe cele:

- Stworzenie algorytmu programowania genetycznego optymalizującego parametry klasyfikatora SVM
- Zastosowanie stworzonego algorytmu do klasyfikacji danych ze zbioru ADHD-200

Realizacja drugiego z powyższych celów służyć ma przede wszystkim sprawdzeniu efektywności stworzonego algorytmu, ale jest też wyzwaniem samym w sobie. Zbiór danych ADHD-200 nie poddaje się łatwo klasyfikacji za pomocą metod uczenia maszynowego, dlatego każda poprawa wyników klasyfikacji względem wyników dotychczas osiąganych będzie sporym sukcesem.

1.2 Struktura pracy

Struktura pracy jest następująca: rozdział drugi przedstawia ważniejsze zagadnienia teoretyczne związane z pracą oraz zawiera przegląd literatury. W rozdziale trzecim opisano zaimplementowany algorytm Kernel GP oraz przedstawiono sposób jego implementacji. Rozdział czwarty przedstawia wyniki działania algorytmu na standardowych zbiorach danych używanych do testowania algorytmów maszynowego uczenia. W rozdziale piątym prezentowane są wyniki działania algorytmu na zbiorze ADHD-200. Rozdział szósty zawiera podsumowanie.

Rozdział 2

Podstawy teoretyczne

2.1 Uczenie maszynowe

Uczenie maszynowe (ang. *Machine Learning*) to dziedzina informatyki zajmująca się konstruowaniem *systemów uczących się* [**?**]. Podstawową cechą takich systemów jest to, że potrafią one zmieniać sposób swojego działania w miarę jak napływają do nich kolejne dane. Zmiana działania systemu może mieć różną skalę - od zmiany pojedynczych parametrów programu, przez zapamiętywanie danych wejściowych po całkowitą zmianę wykonywanego algorytmu. Niezależnie od skali każda taka zmiana powinna mieć wpływ na jego przyszłe działanie i powinna mieć na celu uzyskanie jak najwyższej *oceny* pracy systemu. Jak ujmuje to Tom Mitchell [**?**]:

> System uczy się z doświadczenia E ze względu na pewną klasę zadań T i ocenę wykonania P jeśli ocena wykonania zadań należących do klasy T rośnie wraz z doświadczeniem E.

Systemy uczące się mają wiele zastosowań, między innymi:

- Rozpoznawanie mowy ludzkiej
- Rozpoznawanie tekstu pisanego (*OCR*, ang. *Optical Character Recognition*
- Diagnostyka medyczna
- Klasyfikacja tekstów, np. na potrzeby filtrowania niechcianych wiadomości
- Automatyczna identyfikacja zagrożeń na podstawie obrazu z kamer przemysłowych
- Kierowanie autonomicznymi pojazdami
- Prognozowanie pogody
- Prognozowanie zmian kursów akcji na giełdzie
- Wykrywanie podejrzanych transakcji finansowych
- Biometria - identyfikacja ludzi na podstawie cech takich jak głos, wygląd twarzy, odciski palców, sposób chodzenia
- Wspomaganie podejmowania decyzji

2.1.1 Systemy klasyfikujące

Jednym z typów systemów uczących się są *systemy klasyfikujące* (inaczej *klasyfikatory*). Operują one na zbiorach *przykładów* opisanych za pomocą pewnego zbioru *atrybutów*. *Przykłady* (zwane też *obserwacjami*) reprezentują pewne obiekty, które różnią się od siebie wartościami atrybutów. Każdy przykład jest całkowicie scharakteryzowany przez swoje wartości atrybutów, co oznacza, że dwa przykłady o identycznych wartościach atrybutów są z punktu widzenia systemu klasyfikującego nieodróżnialne. Przykładem zbioru przykładów może być np. zbiór pacjentów, zaś zbiorem atrybutów zbiór cech takich jak wiek, płeć, wzrost, wyniki testów laboratoryjnych. Wśród zbioru atrybutów wyróżnia się jeden specjalny atrybut zwany atrybutem decyzyjnym (w odróżnieniu od pozostałych - atrybutów warunkowych) zwany też klasą lub etykietą obiektu. Zazwyczaj wartość tego atrybutu nie jest znana bezpośrednio, jej zdobycie stanowi pewną wartość. W przytoczonym przypadku pacjentów takim atrybutem może być na przykład diagnoza choroby. Uczenie systemu klasyfikującego polega na dostarczeniu do systemu zbioru przykładów z przypisanymi etykietami. Zbiór taki nazywamy *zbiorem trenującym* / *uczącym*. Na podstawie przykładów ze zbioru uczącego system wytwarza wewnętrzną reprezentację, która następnie umożliwia przypisanie nieznanych klasyfikatorowi etykiet/klas nowym przykładom, które nie występowały w zbiorze uczącym.

Formalizacja problemu klasyfikacji

W celu uściślenia dalszych rozważań konieczne jest wprowadzenie notacji formalnej opisującej problem klasyfikacji [**?**]. Zbiór wszystkich możliwych obiektów x, których dotyczy dany problem klasyfikacji, nazywany jest dziedziną i jest oznaczany przez U. Atrybut $a_i(x) : U \mapsto V_{ai}$ to dowolna funkcja określona na dziedzinie U z przeciwdziedziną A_i. Zbiór wszystkich atrybutów oznaczamy przez $A = \{a_1, a_2, ..., a_n\}$.

Każdy przykład $x \in X$ można opisać jako wektor w n-wymiarowej przestrzeni atrybutów Ω, czyli $\langle a_1(x), a_2(x), ..., a_n(x)\rangle$.

Problem klasyfikacji polega na znalezieniu odwzorowania, które każdemu $x_i \in D$ (gdzie D to zbiór danych wejściowych) przypisuje jego klasę c_i. W przypadku klasyfikacji binarnej $c_i \in \{-1, 1\}$.

Metody oceny skuteczności klasyfikacji

W celu oceny skuteczności systemu należy za jego pomocą dokonać klasyfikacji przypadków, które nie były użyte podczas jego uczenia i których etykiety są znane (choć nie dostępne klasyfikatorowi). Wyniki klasyfikacji porównuje się z właściwymi etykietami i w ten sposób szacuje skuteczność klasyfikacji. W tym celu można wydzielić ze zbioru przykładów specjalny podzbiór, zwany *zbiorem testującym*, który jest używany do testowania a w fazie uczenia klasyfikator nie ma do niego dostępu. Czasami wydziela się też *zbiór walidujący*, który jest używany w trakcie uczenia w celu optymalizacji parametrów algorytmu. Stałego podziału zbioru przykładów na zbiór trenujący, testujący i walidujący można dokonać tylko wtedy, gdy zbiory te są wystarczająco liczne. W przeciwnym przypadku może okazać się, że nie są one wystarczająco reprezentatywne i na przykład rozkład przykładów z poszczególnych klasy jest mocno skrzywiony w którymś ze zbiorów. Aby tego uniknąć można posłużyć się metodą *k-krotnej walidacji krzyżowej*. Polega ona na podzieleniu zbioru na k podzbiorów i następnie powtarzanych k-razy fazach uczenia i testowania klasyfikatora, przy czym za każdym razem k-ty podzbiór służy jako zbiór testujący/walidujący a pozostałem podzbiory jak zbiór uczący. Skuteczności klasyfikacji oblicza się wtedy jako średnią sprawność osiąganą we wszystkich k testach.

2.1.2 Miary skuteczności klasyfikacji

Do oceny jakości klasyfikacji można używać różnych miar. W przypadku klasyfikacji binarnej (czyli kiedy rozróżniamy tylko dwie klasy przykładów) większość z nich można wyrazić za pomocą stosunku kilku z czterech wartości wyrażających liczbę przypadków klasyfikowanych w określony sposób. Wartości te są odnoszą się zawsze do jednej z klas, która jest w pewien sposób wyróżniona. Na przykład w przypadku diagnozy medycznej zazwyczaj taką klasą jest grupa osób chorych na jakąś chorobę. Przypadki zaklasyfikowane jako należące do tej klasy określane są jako zaklasyfikowane *pozytywnie* (ang. *positive*) natomiast przypadki zaklasyfikowane jako do niej nienależące jako zaklasyfikowane negatywnie (ang. *negative*). Słowa ang. *"True"* oraz ang. *"False"* odnoszą się odpowiednio do przypadków zaklasyfikowanych prawidłowo i nieprawidłowo:

- ***True Positive (*TP*)*** - liczba przypadków **poprawnie** zaklasyfikowanych jako **należące** do wyróżnionej klasy,
- ***True Negative (*TN*)*** - liczba przypadków **poprawnie** zaklasyfikowanych jako **nienależące** do wyróżnionej klasy,
- ***False Positive (*FP*)*** - liczba przypadków **niepoprawnie** zaklasyfikowanych jako **należące** do wyróżnionej klasy (inaczej błąd pierwszego rodzaju)
- ***False Negative (*FN*)*** - liczba przypadków **niepoprawnie** zaklasyfikowanych jako **nienależące** do wyróżnionej klasy (inaczej błąd drugiego rodzaju)

Poniżej zostały opisane miary, o których będzie mowa w dalszej części pracy. Wszystkie one zawierają się w przedziale $\langle 0, 1 \rangle$.

- *Precyzja (ang.* precision*)* - określa jaka część przypadków zaklasyfikowanych jako należące do wyróżnionej klasy rzeczywiście do niej należy. Dana jest wzorem:

$$precision = \frac{TP}{TP + FP}$$

- *Kompletność (ang.* recall*)* - określa jaka część przypadków należących do wyróżnionej klasy została prawidłowo zaklasyfikowana jako należące do niej. Dana jest wzorem:

$$recall = TPTP + FN$$

- *Trafność (lub dokładność)* (ang. *Accuracy*) - stosunek liczby przypadków ze zbioru walidującego, które zostały zaklasyfikowane poprawnie do liczby wszystkich przepadków w zbiorze testującym. Może być wyrażona jako:

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

- *Miara* F_1 (ang. F_1 *measure*) - miara uwzględniająca zarówno precyzję (ang. *precision*) jak i *kompletność* (ang. *recall*). Miara ta nie uwzględnia wartości *TN*. Jej wartość jest dana wzorem:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

- MCC *ang.* Matthews correlation coefficient - miara, która w przeciwieństwie do miary F_1 bierze pod uwagę wszystkie cztery wartości (*TP, TN, FP* i *FN*). Dana wzorem:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- *Średnie prawdopodobieństwo wyboru właściwej klasy* - niektóre klasyfikatory zamiast przypisywać każdemu z przykładów jedną z klas potrafią zwrócić dla każdego przykładu rozkład przynależności do wszystkich rozważanych klas. Jakość klasyfikacji można wtedy obliczyć jako uśrednioną po wszystkich przykładach wartość prawdopodobieństwa przypisanego klasie, do której przykład należy. Wartość taka może wahać się od wartości 0 (kiedy dla każdego przykładu do jego właściwej klasy zostało przypisane prawdopodobieństwo 0) do wartości 1 (kiedy dla każdego przykładu do jego właściwej klasy zostało przypisane prawdopodobieństwo 1).

W przypadku problemów, w których wyróżnia się $k > 2$ klas miara korzystająca z wartości TP, TN, FP i FN jest obliczana jako średnia wartość tej miary dla k problemów binarnych polegających na zaklasyfikowaniu przykładów jako należących lub nienależących do wybranej klasy.

2.1.3 SVM

Maszyna wektorów wspierających (*SVM*, ang. *Support Vector Machine*) to rodzaj klasyfikatora binarnego. Stanowi on rozszerzenie *klasyfikatora liniowego*, lecz w przeciwieństwie do niego jest w stanie poprawnie klasyfikować dane *nieseparowalne liniowo*. Jest to możliwe dzięki dokonywanej przez SVM transformacji danych do wyższych wymiarów za pomocą *funkcji jądrowych*.

Klasyfikator liniowy

Jednym z najprostszych klasyfikatorów jest *klasyfikator liniowy*. Rozwiązuje on problem klasyfikacji binarnej poprzez znalezienie w przestrzeni atrybutów Ω hiperpłaszczyzny, która dzieli ją na dwie części odpowiadające dwóm klasom decyzyjnym: $\{-1, 1\}$.

Definicja 2.1.1 *Hiperpłaszczyzna w przestrzeni* Ω *to zbiór:*

$$\{x \in \Omega | \langle w, x \rangle + b = 0\}, w \in \Omega, b \in R \tag{2.1}$$

$\langle x, y \rangle$ *oznacza iloczyn skalarny wektorów* x *i* y*:*

$$\langle x, y \rangle = \sum_{i=1}^{N} [x]_i [y]_i$$

gdzie $[x]_i$ *to i-ta wartość wektora* x*.*

Wektor w we wzorze 2.1 to wektor wag, normalny do hiperpłaszczyzny, $||w||$ to norma euklidesowa tego wektora, czyli jego długość, a $b/||w||$ to odległość płaszczyzny od początku układu współrzędnych. Oba te parametry można razem dowolnie przeskalowywać, to jest pomnożyć w i b przez tę samą stałą zachowując tę samą hiperpłaszczyznę. Dlatego wprowadza się ograniczenie, po którego zastosowaniu otrzymujemy tak zwaną postać kanoniczną hiperpłaszczyzny:

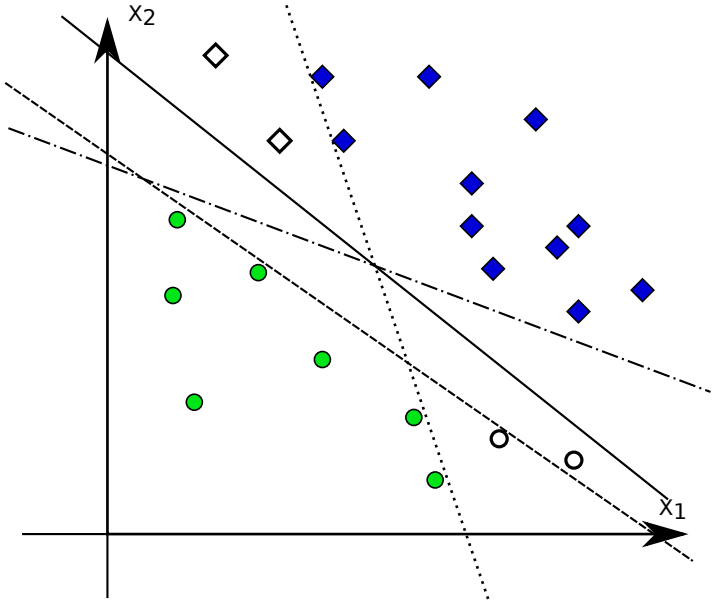
Definicja 2.1.2 *Dla danego zbioru obserwacji* $x_1, x_2, ..., x_m \in \Omega$ *wektor* w *i parametr* b *wyznaczają* ***kanoniczną postać hiperpłaszczyzny*** *jeśli:*

$$\min_{i=1..m} |\langle w, x_i \rangle + b| = 1 \tag{2.2}$$

Hiperpłaszczyzna dana wzorem 2.1 definiuje funkcję decyzyjną, która każdemu przypadkowi z Ω przypisuje klasę decyzyjną:

$$\begin{aligned} &f_{w,b} : \Omega \to \{\pm 1\} \\ &x \mapsto f_{w,b}(x) = sgn(\langle w, x \rangle + b) \end{aligned} \tag{2.3}$$

Wynikiem uczenia klasyfikatora liniowego jest znalezienie hiperpłaszczyzny i odpowiadającej jej funkcji decyzyjnej, która przykładom ze zbioru uczącego $(x_i, y_i) \in \Omega$ przypisuje prawidłowe etykiety,



RYSUNEK 2.1: Hiperpłaszczyzny separujące dwa zbiory punktów w przestrzeni dwówymiarowej. Każda z z nich poprawnie separuje punkty ze zbioru uczącego - zielone koła i niebieskie kwadraty. Przykłady ze zbioru testowego (puste kwadraty i kółka) są poprawnie separowane jedynie przez dwie proste (prosta narysowana linią ciągłą i prosta narysowana kropkami i keskami).

czyli dla każdego (jeśli zbiór jest liniowo separowalny), lub dla jak największej liczby przypadków x_i zachodzi $f_{w,b}(x_i) = y_i$. Zazwyczaj kilka hiperpłaszczyzn równie dobrze rozdziela przypadki ze zbioru uczącego, mogą się jednak one różnić zdolnością do klasyfikacji zbioru testowego, co pokazano na rysunku 2.1.3. Optymalna hiperpłaszczyzna separująca to taka, która charakteryzuje się największym marginesem, czyli odległością hiperpłaszczyzny do najbliżej położonych obserwacji [**?**].

Definicja 2.1.3 *Dla hiperpłaszczyzny danej wzorem* $x \in \Omega | \langle w, x \rangle + b = 0$ *oraz zbioru obserwacji* $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ *marginesem tego zbioru od hiperpłaszczyzny nazywamy minimalną odległość hiperpłaszczyzny od punktów z tego zbioru:*

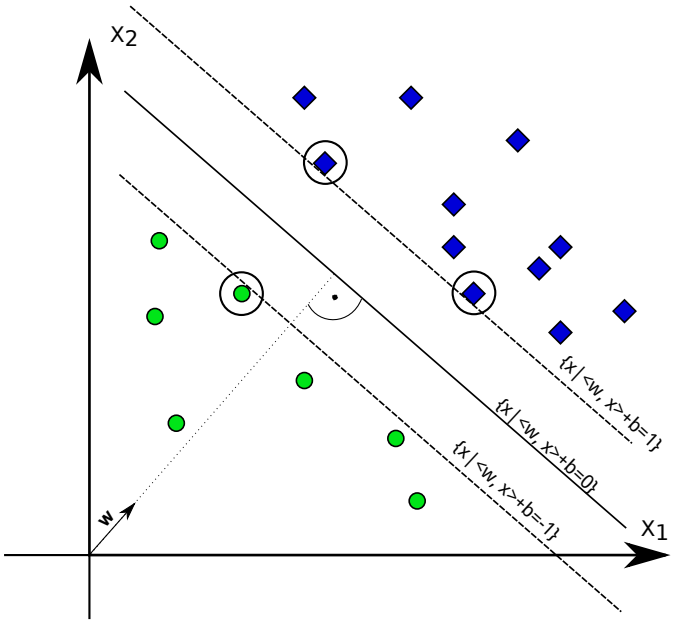
$$\rho_{w,b} := \min_{i=1..m} y \times (\langle w, x \rangle + b)/||w|| \tag{2.4}$$

Żeby maksymalizować margines powinniśmy minimalizować $||w||$, zachowując warunek 2.2. Problem znalezienia optymalnej hiperpłaszczyzny separującej zbiór przykładów $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ można zatem zapisać jako problem optymalizacyjny:

$$\begin{aligned} &\min_{w \in \Omega, b \in \Re} && \tau(w) = \tfrac{1}{2}||w||^2 \\ &\text{p.o.} && y_i(\langle w, x \rangle + b) \geq 1 \quad \text{dla } i = 1..m \end{aligned} \tag{2.5}$$

Ograniczenia w powyższym problemie zapewniają, że wartość funkcji decyzyjnej $f_{w,b}(x_i)$ będzie równa y_i, czyli, że hiperpłaszczyzna poprawnie odseparuje przykłady z dwóch grup. Osiągnięcie celu optymalizacji zapewnia znalezienie hiperpłaszczyzny o maksymalnym marginesie.

Powyższy problem programowania matematycznego jest podany w tak zwanej *formie prymalnej*. W praktyce rozwiązuje się wersję *dualną* problemu, która ma przyjmuje następującą postać:



RYSUNEK 2.2: Hiperpłaszczyzna separująca dwa zbiory punktów w przestrzeni dwuwymiarowej wraz z marginesami. Przykłady w kółku to wektory podpierające.

Definicja 2.1.4 *Wersja dualna problemu znalezienia optymalnej hiperpłaszczyzny:*

$$\begin{aligned} \max_{\alpha \in R^m} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ p.o. \quad & \alpha_i \geq 0, \; dla \; i = 1..m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{2.6}$$

Dla formy dualnej problemu optymalizacji funkcja decyzyjna przyjmuje postać:

$$f(x) = sgn\left(\sum_{i=1}^{m} y_i a_i \langle x, x_i \rangle + b\right) \tag{2.7}$$

W przypadku, w którym dane nie są separowalne liniowo, tzn. nie istnieje taka hiperpłaszczyzna, która dla danego zbioru przykładów spełnia ograniczenia ze wzoru 2.8, do problemu optymalizacyjnego wprowadza się tak zwane *zmienne osłabiające* (ang. *slack variables*) $\zeta_i, i = 1..m$. Wzór 2.8 przyjmuje wówczas postać:

$$\begin{aligned} \min_{w \in \Omega, b \in \Re} \quad & \tau(w) = \tfrac{1}{2}||w||^2 + \tfrac{C}{m}\sum_{i=1}^{m} \zeta_i \\ \text{p.o.} \quad & y_i(\langle w, x \rangle + b) \geq 1 - \zeta_i \qquad \text{dla } i = 1..m \\ & \zeta_i \geq 0 \qquad \text{dla } i = 1..m \end{aligned} \tag{2.8}$$

Natomiast wzór 2.6:

$$\begin{aligned} \max_{\alpha \in R^m} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{p.o.} \quad & 0 \leq \alpha_i \leq \tfrac{C}{m}, \text{ dla } i = 1..m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{2.9}$$

Wektory wspierające

Maszyny wektorów wspierających

- *Transformacja*
- *Kernel Trick*

2.2 Obliczenia ewolucyjne

Obliczenia ewolucyjne to ...

- *osobnik*
- *populacja*
- *pokolenie*
- *mutacja*
- *krzyżowanie*
- *selekcja*
- *funkcja przystosowania* (ang. *fitness*)

2.2.1 Programowanie genetyczne

Programowanie genetyczne (*GP*, ang. *Genetic Programming*) to

Funkcje, które generuje algorytm programowania genetycznego są w nim reprezentowane w postaci drzew. Węzłami takiego drzewa są elementarne funkcje zadeklarowane w kodzie programu. Każda z takich funkcji ma przypisane pewne ograniczenia co do ilości i typu argumentów, które przyjmuje oraz co do typu wartości, który zwraca. Drzewo jako całość również ma zadeklarowany typ zwracanej wartości.

2.3 Ewolucja kerneli

Rozdział 3

Algorytm Kernel GP

3.1 Opis algorytmu

Jedną z trudności, która wiąże się z używaniem klasyfikatora SVM jest dobór odpowiedniej do zbioru danych *funkcji jądrowej*. Wymaga to doświadczenia lub przebiega na zasadzie prób i błędów. Ponadto zbiór powszechnie używanych funkcji jest ubogi - zazwyczaj ogranicza się do trzech podstawowych funkcji. Oprócz wyboru funkcji konieczne jest również ustawienie odpowiednich wartości ich parametrów.

Celem algorytmu Kernel GP jest odnalezienie optymalnej dla danego problemu funkcji jądrowej wraz z jej parametrami. Dzięki opisanej w poprzednim rozdziale własności domknięcia zbioru kerneli ze względu na pewne operacje arytmetyczne możliwe jest tworzenie nieograniczonej ilości dowolnie złożonych funkcji na podstawie kilku podstawowych kerneli. Opisywany algorytm przeszukuje przestrzeń takich funkcji za pomocą *programowania genetycznego*. Szukana jest taka funkcja, przy której użyciu klasyfikator SVM osiągnie największą *trafność* (*accuracy*) klasyfikacji.

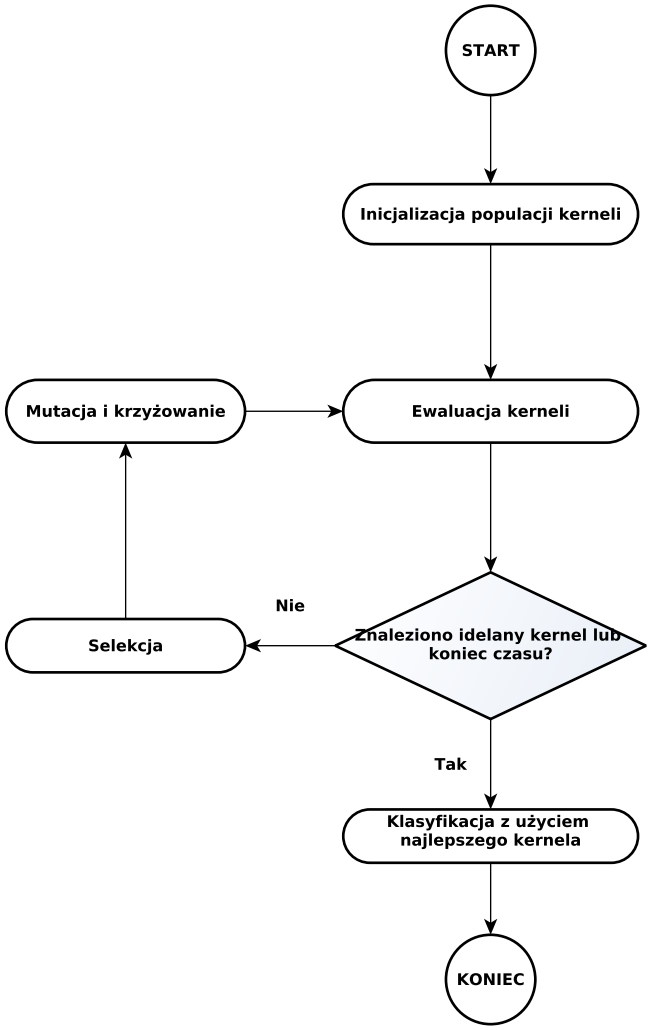
Przebieg algorytmu jest typowy dla algorytmów genetycznych:

1. Utwórz początkową populację kerneli
2. Oblicz wartość *funkcji dopasowania* każdego z kerneli: trafność klasyfikacji SVM z użyciem tego kernela
3. Jeśli znaleziono idealny kernel (wartość funkcji fitness wyniosła 1) 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

Algorytm pokazano również na diagramie przepływu na rycinie 3.1. Poszczególne kroki algorytmu zostaną opisane poniżej.

3.1.1 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



RYSUNEK 3.1: Diagram przepływu algorytmu Kernel GP.

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.

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 [**?**].
  - 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: $k(x,z) = \tanh(\gamma\langle x,z\rangle + \tau)$
  - Logarytmiczna: $k(x,z) = -log(\|x-y\|^d + 1)$
  - Potęgowa: $k(x,z) = (\alpha x^T z + c)^d$
  - Cauchego: $k(x,y) = \frac{1}{1+\frac{\|x-y\|^2}{\sigma}}$
  - Wykładnicza: $k(x,y) = \exp\left(-\frac{\|x-y\|}{2\sigma^2}\right)$

  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 $\langle 0.1, 2.0\rangle$
  - τ: liczba rzeczywista z zakresu $\langle 0.1, 1.0\rangle$
  - d: liczba całkowita z zakresu $\langle 1.0, 10.0\rangle$
  - a: liczba rzeczywista z zakresu $\langle -10.0, 10.0\rangle$

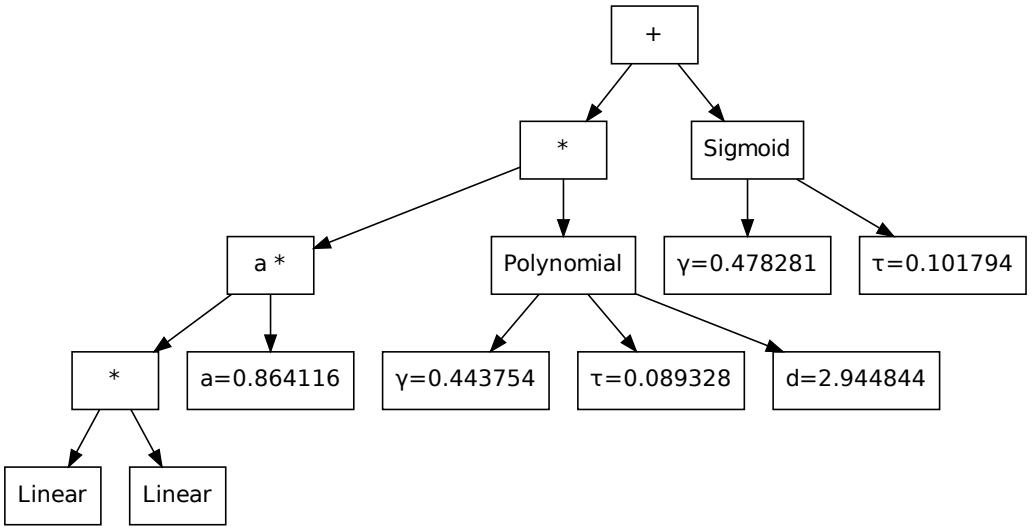
Przykładowe drzewo wygenerowane przez algorytm pokazana na ryc.3.2.

Wektory cech będące najważniejszymi argumentami funkcji jądrowych nie są wyodrębnione jako osobne funkcje budujące drzewo.

3.1.2 Ewaluacja kerneli

Każda wygenerowana przez algorytm GP funkcja zostaje poddana ocenie, w wyniku której zostaje jej przypisana wartość funkcji przystosowania (ang. *fitness*). W tym celu funkcja ta jest wykorzystywana przez algorytm SVM jako funkcja jądrowa a jakość wyników klasyfikacji stanowi ocenę funkcji jądrowej. Ewaluacja funkcji jądrowej może odbywać się na jeden z dwóch sposobów. Jeśli w zbiorze danych oprócz zbioru uczącego wydzielono zbiory testowy i walidujący, to sprawdzany kernel jest używany do klasyfikacji danych ze zbioru testującego. Ocena jakości klasyfikacji zostaje przeliczona na wartość *funkcji przystosowania* ewaluowanej funkcji jądrowej. Jeśli w zbiorze danych wydzielono tylko dwa podzbiory: uczący i walidujący, to zdolność klasyfikacji przez kernel jest oceniana za pomocą *walidacji krzyżowej (ang. cross-validation.* Walidacja krzyżowa pozwala użyć więcej danych podczas fazy uczenia, jednak wiąże się ze znacznym wzrostem złożoności obliczeniowej - zamiast jednej klasyfikacji musimy przeprowadzić k procesów uczenia i k klasyfikacji.

Do oceny jakości wyników klasyfikacji używana jest jedna z miar opisanych w części 2.1.2. Ponieważ wszystkie te miary należą do zakresu $\langle 0,1\rangle$, to mogą być bezpośrednio użyte jako wartość fitness ewaluowanego kernela.



RYSUNEK 3.2: Przykładowe drzewo generowane przez algorytm.

3.1.3 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 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.

3.1.4 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.

3.1.5 Walidacja rozwiązania

Walidacja polega na użyciu najlepszego znalezionego kernela do klasyfikacji przykładów ze zbioru walidującego, które nie były używane podczas uczenia klasyfikatora SVM ani podczas ewaluacji ker-

neli. Najpierw algorytm SVM jest uczony na połączonych zbiorach trenującym i testującym, przy pomocy tej funkcji jądrowej. Następnie dokonywana jest klasyfikacja zbioru walidującego. Otrzymane w wyniku tej klasyfikacji miary jakości klasyfikacji (opisane w część 2.1.2 są miarą oceny całego algorytmu.

3.2 Implementacja

Algorytm został napisany w języku Java z użyciem bibliotek *ECJ (Evolutionary Computing in Java)* [**?**] oraz LibSVM [**?**]. 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.

3.3 Złożoność obliczeniowa

Rozdział 4

Wyniki działania algorytmu na popularnych zbiorach danych

4.1 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* ze zbioru uczącego nie wydzielano zbioru testującego.

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, liczbę ewaluowanych pokoleń oraz najlepszą funkcję przystosowania przeprowadzono szereg eksperymentów obliczeniowych, w których uruchamiano algorytm dla coraz to większych wartości tych parametrów. Dla każdego przebiegu algorytmu obliczano i zapisywano kilka miar trafność klasyfikacji zbioru *walidującego* (miary te zostały opisane w części 2.1.2).

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 populacji przez które poszukiwano rozwiązania. W szczególności ciekawym przypadkiem jest ten, gdy liczba populacji 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 jednego pokolenia i coraz większej ich liczby można ocenić czy proces ewolucyjny przebiega poprawnie.

4.2 Opis zbiorów danych

Do oceny pracy algorytmu użyto standardowych zbiorów danych służących do testowania systemów maszynowego uczenia się, dostępnych na stronie biblioteki *LIBSVM* [**?**] oraz w repozytorium UCI [D]. Zbiory zostały opisane w tabelce 4.1. Użyte nazwy zbiorów są zgodne z tymi ze strony libsvm [C].

TABLICA 4.1: Zbiory danych użyte do testowania systemu.

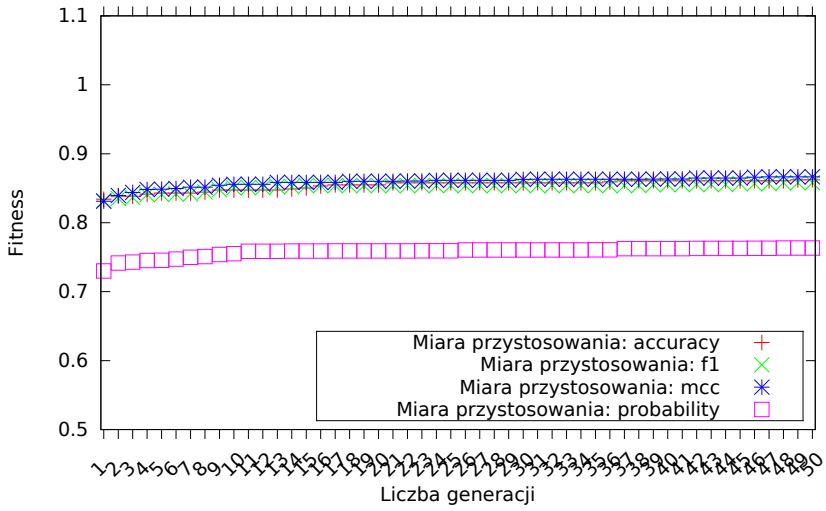
Nazwa zbioru	Liczba klas	Liczba atrybutów	Wielkość zbioru	Wielkość zbioru uczącego	Wielkość zbioru testującego	Wielkość zbioru walidującego
Iris	3	4	150	68	33	49
Letter	26	16	15000	9000	4400	6600
DNA	3	180	2000	1435	700	1051
Vowel	11	10	528	447	217	326
Breast cancer	2	10	683	343	170	170
Heart	2	13	270	136	67	67

TABLICA 4.2: Zbiory danych użyte do testowania systemu.

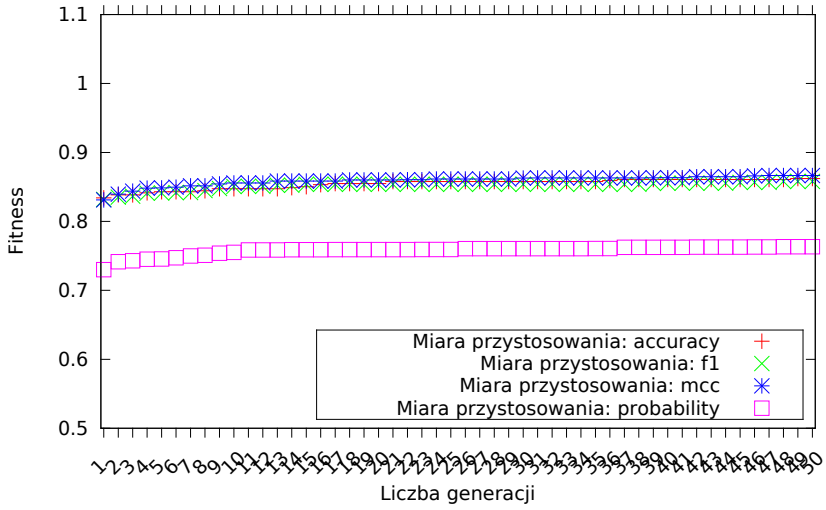
Nazwa zbioru	Liczba atrybutów ciągłych	Liczba atrybutów nominalnych	Liczba klas	Proporcje klas
Breast Cancer	10	0	2	239/444
Heart	7	6	2	150/120
DNA	0	180	3	464/485/1051
Vowel	10	0	11	Każda klasa 48 razy

4.3 Fitness

Aby ocenić dobór parametrów procesu ewolucyjnego zapisywano wartości fitness osiągane przez najlepszego osobnika w każdym pokoleniu. Wartości te zostały przedstawione na wykresach przedstawiających wartość fitness najlepszego osobnika w funkcji czasu trwania algorytmu (ilości dotychczas wygenerowanych populacji), czyli tak zwanych ang. *Fitness Graphs*. Na wykresash 4.1 - 4.2 widać jak zmienia się wartość funkcji fitness wraz z kolejnymi pokoleniami dla różnych miar jakości klasyfikacji użytych jako funkcja fitness (miary te zostały opisane w części 2.1.2 a ich użycie w części 3.1.2).



RYSUNEK 4.1: Najlepsza wartość funkcji przystosowania dla kolejnych pokoleń dla zbioru *heart*.

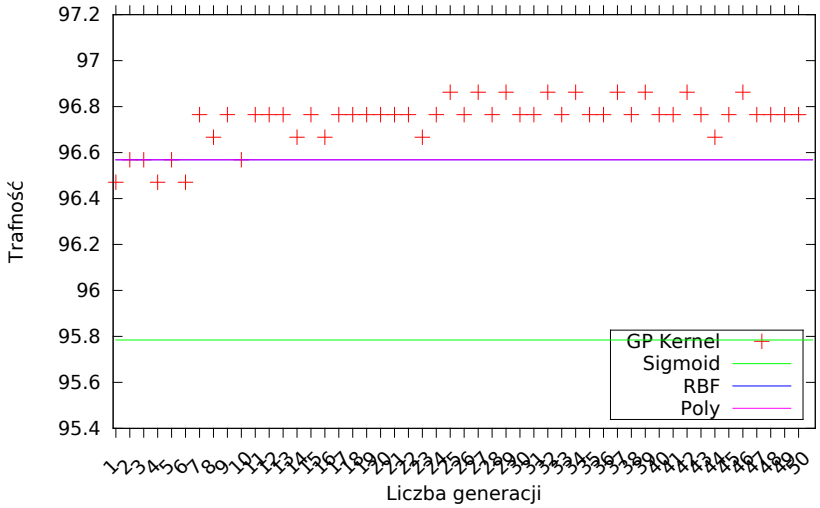


RYSUNEK 4.2: Najlepsza wartość funkcji przystosowania dla kolejnych pokoleń dla zbioru *breast*.

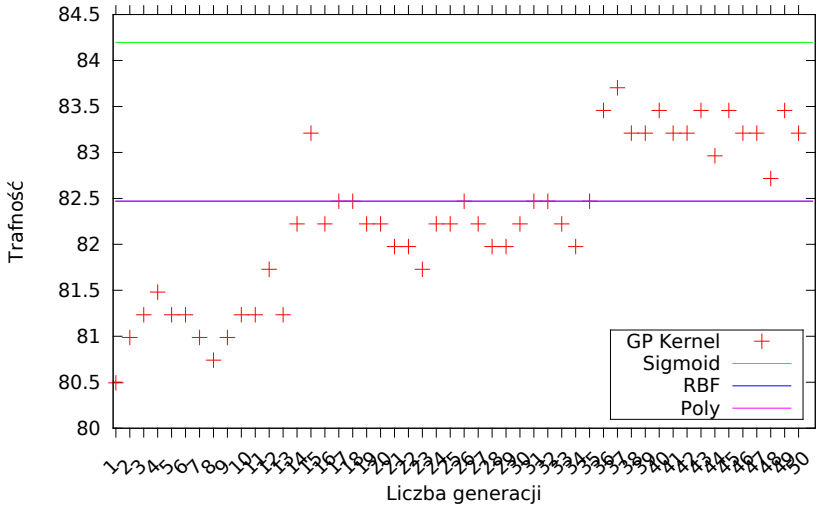
4.4 Wyniki klasyfikacji zbioru walidującego

Wyniki klasyfikacji zostały ocenione za pomocą miar opisanych w części 2.1.2. Dla każdej z tych miar przedstawiono jej wartości dla przebiegów algorytmu, w których jako funkcja fitness była wybrana właśnie ta miara. Dodatkowo na wykresach ukazano wartości danych miar uzyskane w wyniku klasyfikacji za pomocą trzech standardowych funkcji jądrowych (*Wielomianowej*, *Sigmoidalnej* i *RBF*).

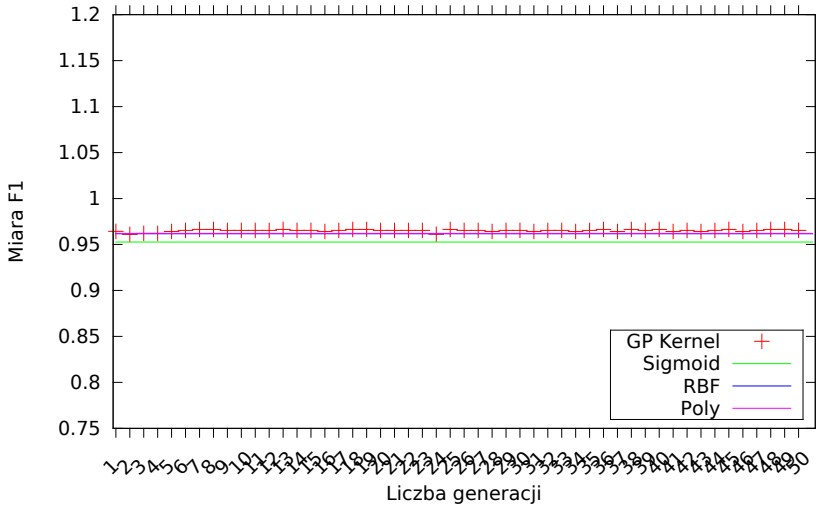
Jak widać na rysunkach 4.3-4.10 sprawność algorytmu zależy zarówno od zbioru danych jak i od wybranej miary jakości.



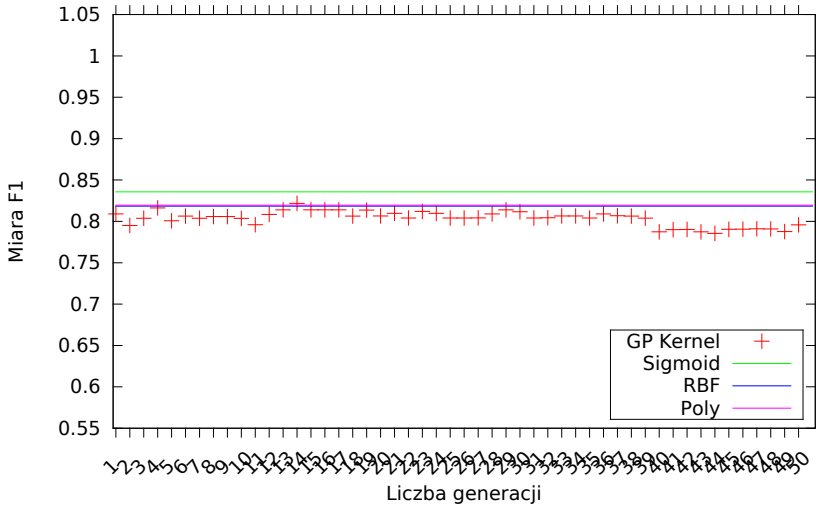
RYSUNEK 4.3: Tragność (ang. *accuracy*) klasyfikacji dla zbioru *breast* w funkcji czasu wykonania (ilości pokoleń).



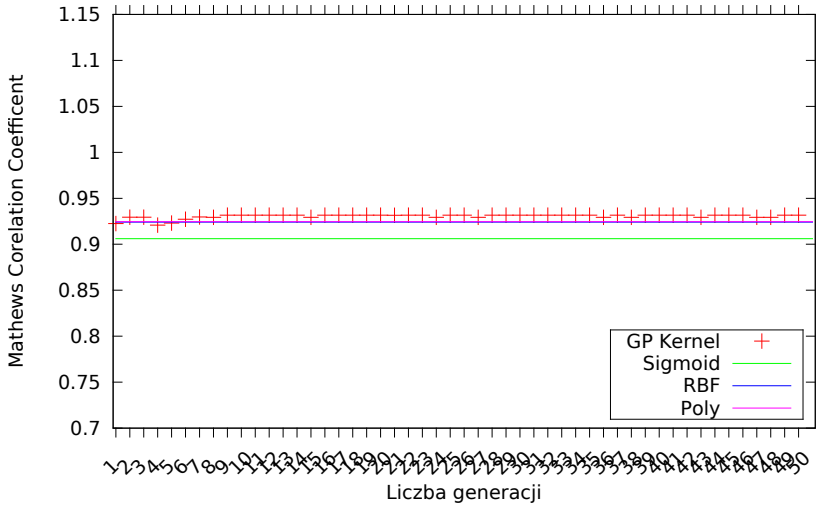
RYSUNEK 4.4: Tragność (ang. *accuracy*) klasyfikacji dla zbioru *heart* w funkcji czasu wykonania (ilości pokoleń).



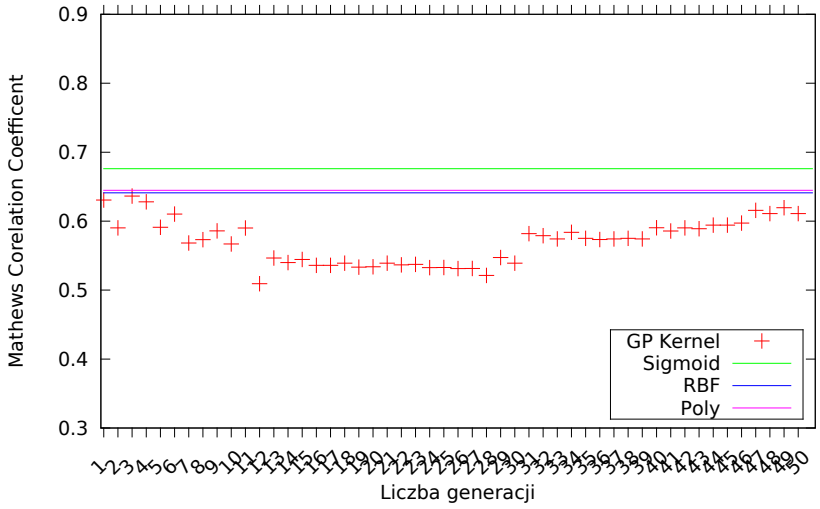
RYSUNEK 4.5: Wartość miary F1 dla wyników klasyfikacji zbioru *breast* w funkcji czasu wykonania (ilości pokoleń).



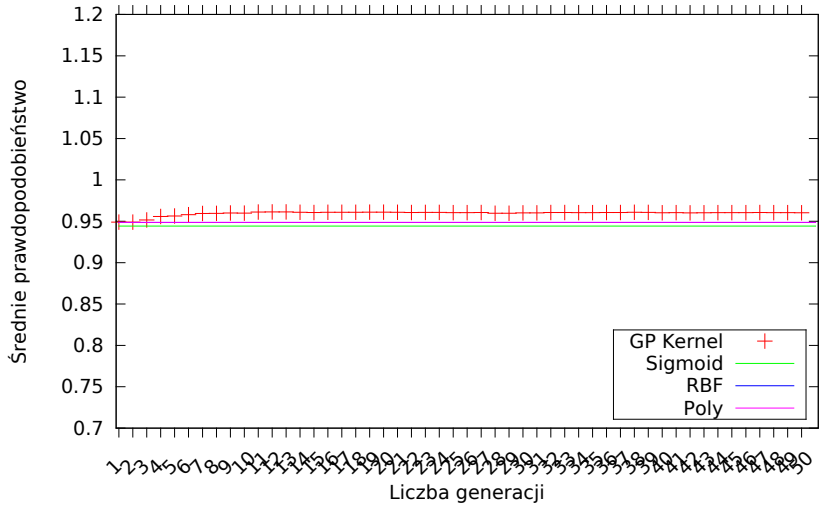
RYSUNEK 4.6: Wartość miary F1 dla wyników klasyfikacji zbioru *heart* w funkcji czasu wykonania (ilości pokoleń).



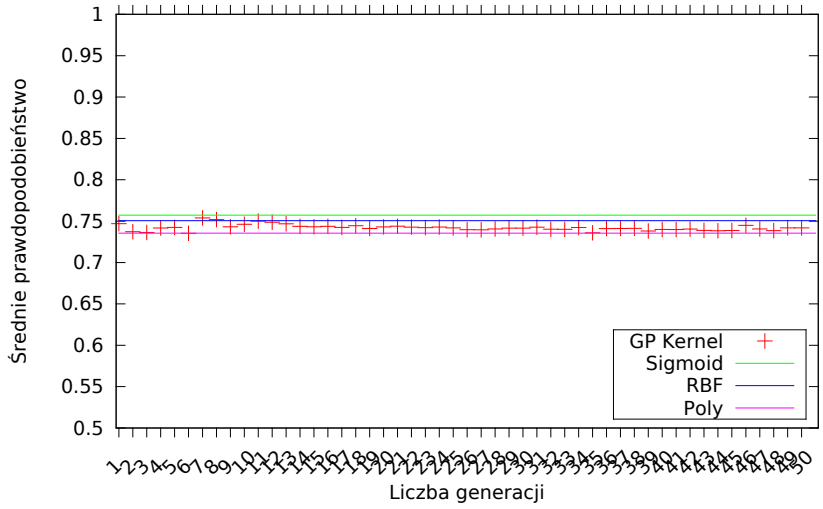
RYSUNEK 4.7: Wartość miary *Mathews Correlation Coefficent* (*MCC*) dla wyników klasyfikacji zbioru *breast* w funkcji czasu wykonania (ilości pokoleń).



RYSUNEK 4.8: Wartość miary *Mathews Correlation Coefficent* dla wyników klasyfikacji zbioru *heart* w funkcji czasu wykonania (ilości pokoleń).



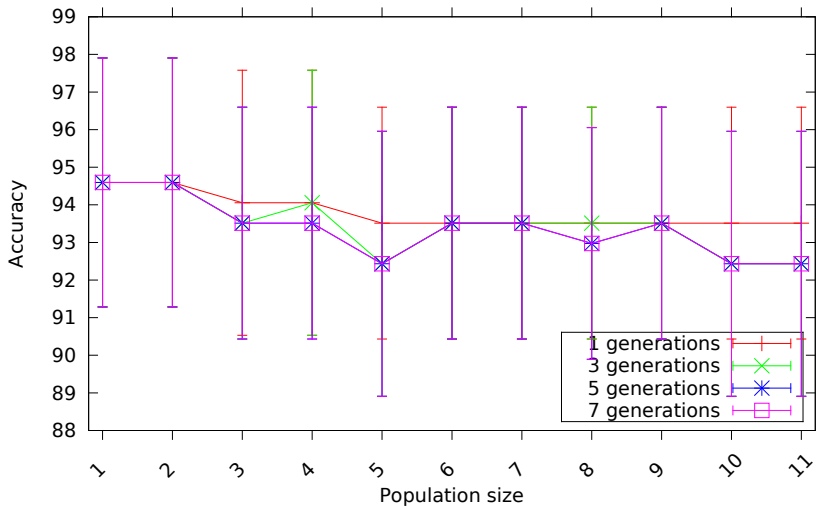
RYSUNEK 4.9: Średnia wartość prawdopodobieństwa przypisywanego przez SVM właściwej dla klasyfikowanego przykłau klasie w funkcji czasu wykonania (ilości pokoleń). Zbiór *breast*



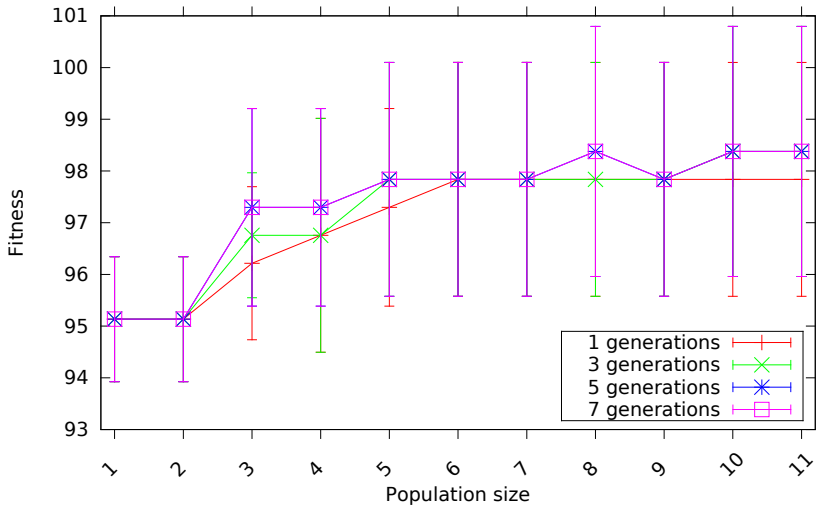
RYSUNEK 4.10: Średnia wartość prawdopodobieństwa przypisywanego przez SVM właściwej dla klasyfikowanego przykłau klasie w funkcji czasu wykonania (ilości pokoleń). Zbiór *heart*.

4.4.1 Monotoniczność funkcji trafności

Miejscami funkcja trafności nie jest monotoniczna, a ściślej niemalejąca, względem liczby pokoleń oraz wielkości populacji (co widać np. na wykresach 4.11 i 4.13). Wydawałoby się, że tak być nie powinno (algorytm genetyczny zwraca najlepszego osobnika z całego swojego przebiegu, więc wszystkie osobniki, które pojawiły się w pierwszych 5 pokoleniach pojawią się w pierwszych 7 pokoleniach, więc trafność dla po 7 pokoleniach powinna być co najmniej tak dobra jak po 5). 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 4.12, który przedstawia wartość przystosowania dla tych samych danych, dla których na wykresie 4.11 jest pokazana trafność klasyfikacji na zbiorze walidującym - tutaj funkcja wykazuje mniej braku monotoniczności.

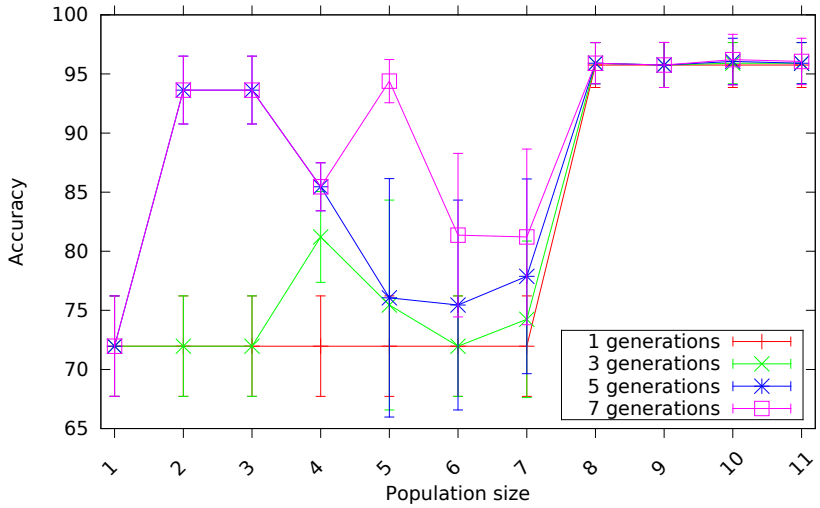


RYSUNEK 4.11: Trafność klasyfikacji dla zbioru *iris* w funkcji rozmiaru populacji dla róznych ilości pokoleń, dla małych populacji.

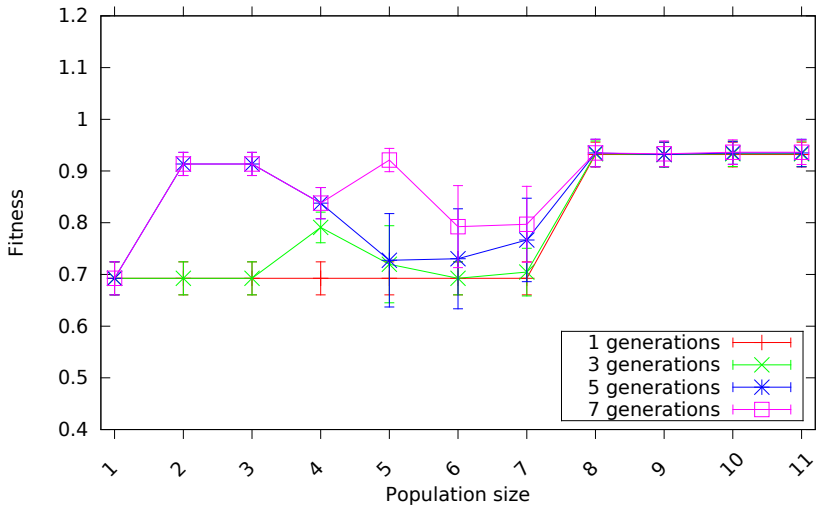


RYSUNEK 4.12: Najlepsza wartość funkcji przystosowania (ang. *fitness*) *iris* w funkcji rozmiaru populacji dla róznych ilości pokoleń, dla małych populacji.

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 4.14 przedstawiającym wartość funkcji przystosowania dla zbioru *vowel* - jej przebieg jest bardzo podobny do przebiegu ukazanej na rys. 4.13 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 pokoleń oraz że dla jednego pokolenia funkcja jest monotoniczna. Sugeruje to, że "winnym"może być selekcja - w praktyce nie zachodzi ono w przypadku gdy algorytm genetyczny działa przez jedno pokolenie.

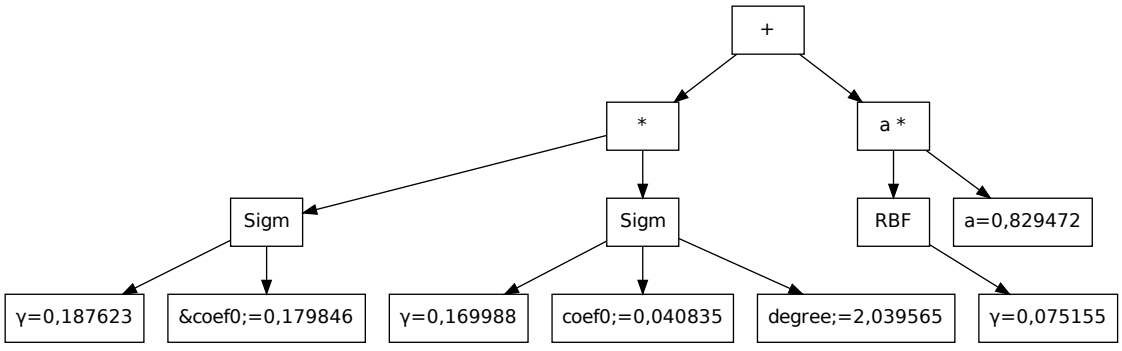


RYSUNEK 4.13: Trafność klasyfikacji dla zbioru *vowel* w funkcji rozmiaru populacji dla róznych ilości pokoleń, dla małych populacji.

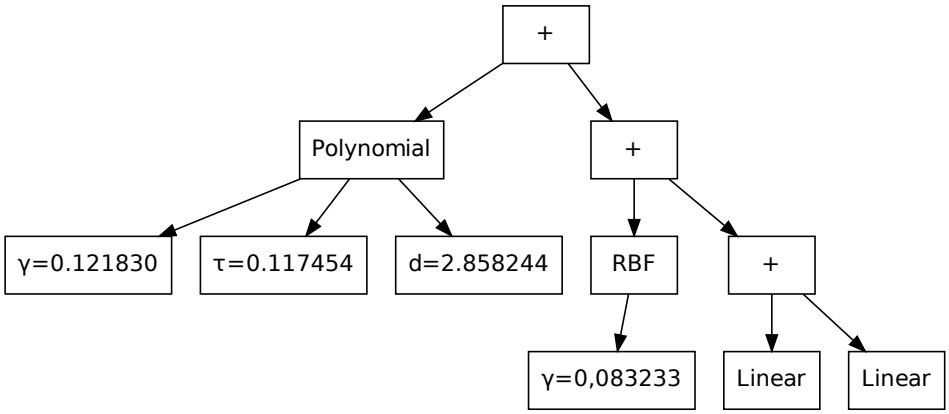


RYSUNEK 4.14: Trafność klasyfikacji dla zbioru *vowel* w funkcji rozmiaru populacji dla różnych ilości pokoleń, dla małych populacji.

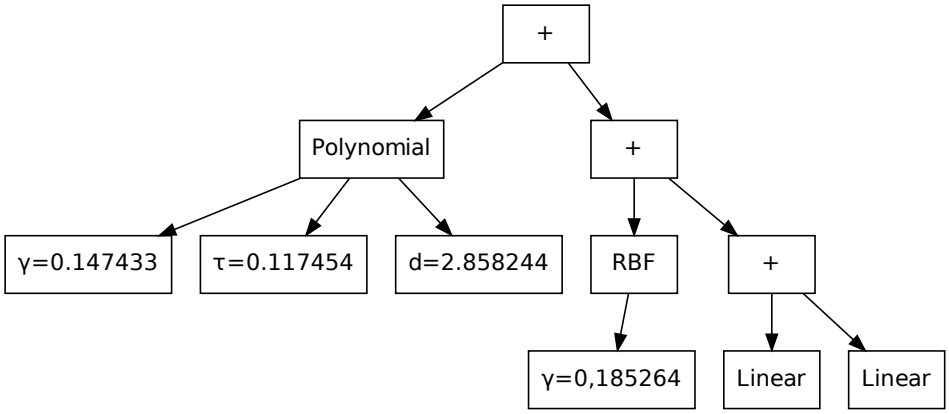
Gdy przyjrzeć się dokładnie przebiegowi ewolucji widać, że rzeczywiści tak jest. Dodatkowy osobnik (rys. 4.15), który odróżnia w pokoleniu pierwszym populacje o wielkości 3 i 4 jest przodkiem innego osobnika (rys. 4.16), który w trzecim pokoleniu osiąga fitness większy niż osobnik (rys. 4.18), który w przypadku populacji wielkości 3 był przodkiem osobnika (rys. 4.19), 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 pokoleniach nie musi być wcale potomkiem osobników najlepszych w poszczególnych pokoleniach - czasem połączenie dwóch osobników przeciętnych może dać osobnika bardzo dobrego.



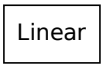
RYSUNEK 4.15: Funkcja z pierwszego pokolenia, która w przebiego z wielkością populacji 4 osiągnęła fitness 0.4242424. Przodek funkcji z rys.4.16



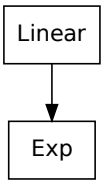
RYSUNEK 4.16: Funkcja z trzeciego pokolenia, która w przebiego z wielkością populacji 4 osiągnęła fitness 0.78030306. Potomek funkcji z rys.4.15, przodek zwycięskiej funkcji (rys.4.17) z przebiegu z populacją o wielkości 4.



RYSUNEK 4.17: Funkcja z ostatniego pokolenia w przebiegu z wielkością populacji 4 osiągnęła fitness 0.8333333. Przodek zwycięskiej funkcji z rys.4.19



RYSUNEK 4.18: Funkcja z trzeciego pokolenia, która w przebiegu z wielkością populacji 3 i 4 osiągnęła fitness 0.6515151. Przodek zwycięskiej funkcji z rys.4.19



RYSUNEK 4.19: Zwycięska funkcja w przebiegu z populacją wielkości 3, potomek funkcji z rys.4.18. Osiągnęła fitness 0.9015151.

4.5 Czas wykonania

4.6 Użycie pamięci

4.7 Posdumowanie wyników

Rozdział 5

Case study - klasyfikacja danych ADHD 200

ADHD 200 był międzynarodowym konkursem, który zakończył się we wrześniu 2011 roku. Dzięki współpracy ośmiu szpitali i ośrodków naukowych z całego świata w ramach konkursu udostępniono zbiór zawierające dane medyczne 776 dzieci, z czego 285 z ADHD. Zadaniem uczestników konkursu było skonstruowanie klasyfikatora diagnozującego ADHD na podstawie tych danych. Zbiór testowy zawierał dane xxx dzieci, których danych nie było w zbiorze uczącym, bez podanej diagnozy. Celem skonstruowanego klasyfikatora było przypisanie diagnozy do przykładów ze zbioru testującego.

Wyniki konkursu pokazały, że zbiór danych był trudny w klasyfikacji. Największa osiągnięta trafność klasyfikacji wyniosła 60.51% (szczegółowe wyniki dostępne na stronie konkursu: [B].

5.1 Opis zbioru danych

5.1.1 Surowe dane

Dane dostarczone przez organizatorów konkursu składają się z:

- Danych klinicznych:
  - Płeć
  - Wiek
  - Współczynnik IQ
  - Prawo/lewo ręczność
- Danych obrazowych:
  - Strukturalnych - dane pochodzące z obrazowania rezonansu magnetycznego (*MRI*, ang. *Magnetic Resonance Imaging*). Są to trójwymiarowe obrazy o rozdzielczości ok. 256x254x160 punktów, obrazujące strukturę mózgu osoby badanej
  - Funkcjonalnych - dane pochodzące z obrazowania funkcjonalnego rezonansu magnetycznego (*FMRI*, ang. *functional Magnetic Resonance Imaging*) będące sekwencją ok 120 trójwymiarowych obrazów o rozdzielczości ok 250x250x250 punktów, obrazującą aktywność mózgu osoby badanej rejestrowaną przez ok 6 minut.

5.1.2 Preprocessing

5.2 Konstrukcja i selekcja cech

5.3 Wyniki klasyfikacji

5.3.1 Kernel GP

5.3.2 Porównanie z innymi algorytmami

SVM

Inne klasyfikatory

Rozdział 6

Podsumowanie

Zasoby internetowe

[A] ECJ
http://cs.gmu.edu/~eclab/projects/ecj/

[B] ADHD 200
http://fcon_1000.projects.nitrc.org/indi/adhd200/

[C] Libsvm datasets
http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[D] UCI Machine Learning Repository
http://archive.ics.uci.edu/ml/datasets/

Instytut Informatyki, Wydział Informatyki i Zarządzania
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ mnowak-masterthesis,
    author = "Tomasz Ziętkiewicz",
    title = "{Optymalizacja klasyfikatora SVM za pomocą programowania genetycznego}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2013",
}
```