Sprawozdanie z eksperymentu z przedmiotu Metaheurystyki i Obliczenia Inspirowane Biologicznie

7 października 2012

Prowadzący: dr inż. Maciej Komosiński

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

Zajęcia poniedziałkowe, 15:10.

Streszczenie

SVM

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.

1.2 Opis algorytmu

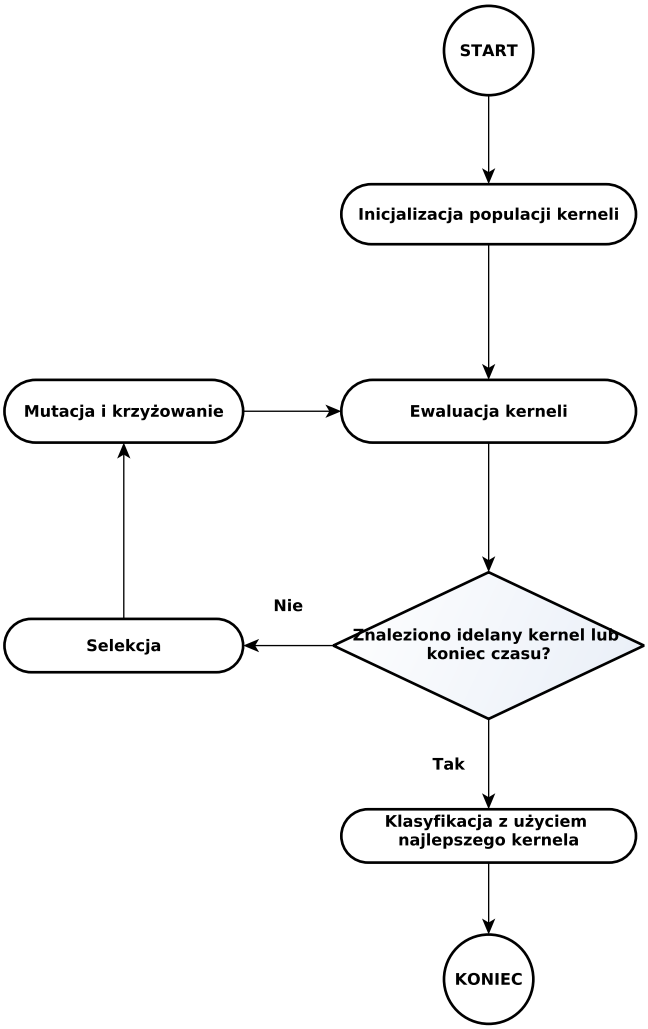
Przebieg algorytmu jest typowy dla algorytmów genetycznych:

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

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



Rysunek 1: Diagram przepływu algorytmu Kernel GP.

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 [**?**].

- 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.

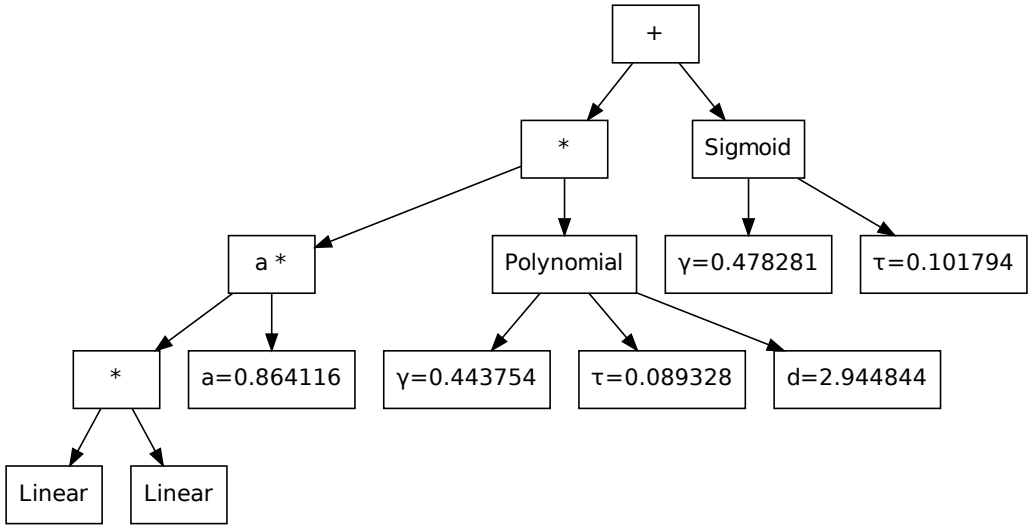
- Stałe *ERC* (ang. *Ephemeral Random Constant*) liczby rzeczywiste lub całkowite, które służą jako parametry innych funkcji. Są one liściami w drzewie, nie przyjmują żadnych argumentów. Mogą losowo zmieniać swoją wartość podczas mutacji.
  - γ: liczba rzeczywista z zakresu
  - τ: liczba rzeczywista 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 funkcje budujące drzewo.

1.4 Ewaluacja kerneli

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. Trafność 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.



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.

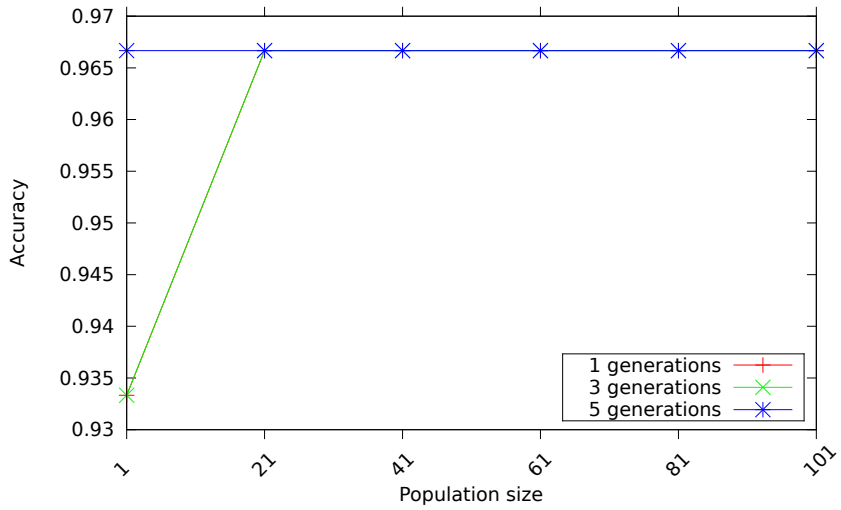
1.7 Walidacja rozwiązania

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. 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)* [**?**] 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 Wyniki

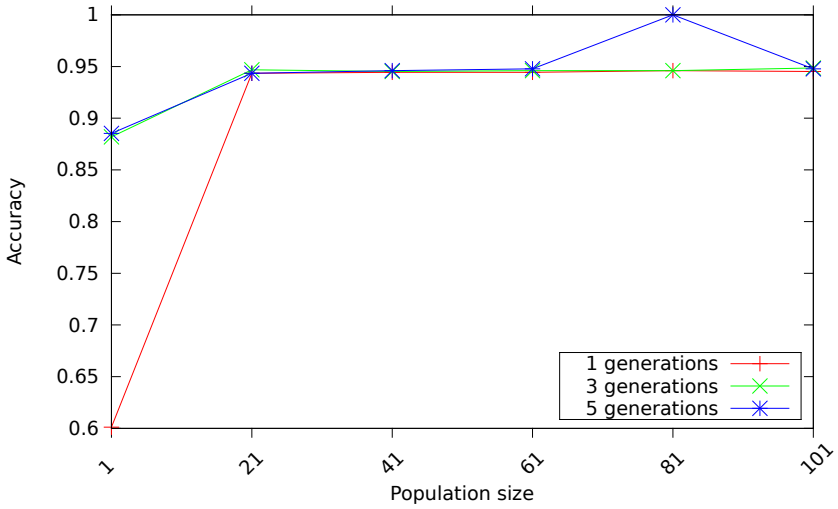


Rysunek 3: Dokładność klasyfikacji dla zbioru *iris* w funkcji rozmiaru populacji dla róznych ilości generacji.

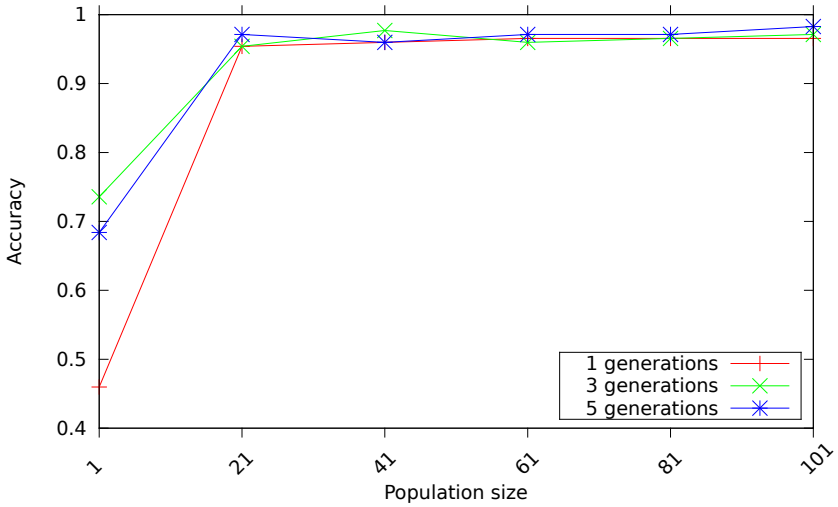
4 Podsumowanie

4.1 Wnioski

-



Rysunek 4: Dokładność klasyfikacji dla zbioru *DNA* w funkcji rozmiaru populacji dla róznych ilości generacji.



Rysunek 5: Dokładność klasyfikacji dla zbioru *vowel* w funkcji rozmiaru populacji dla róznych ilości generacji.