WSPÓŁCZESNE METODY HEURYSTYCZNE - SPRAWOZDANIE CZĄSTKOWE
AGNIESZKA CZAPLICKA, BARTOSZ SOWUL

PB14 - Wykorzystanie algorytmu ewolucyjnego do doboru parametrów SVM do zadania klasyfikacji

1 OPIS PROBLEMU

Celem naszego projektu jest napisanie programu w języku Python, który wykorzystuje algorytm ewolucyjny do optymalizacji parametrów *C* i *gamma SVM* do zadania klasyfikacji na zbiorach danych o 3 różnych stopniach trudności. *C* oznacza dopuszczalny koszt pomyłki modelu na danych trenujących podczas procesu uczenia, a więc pozwala kontrolować liczbę źle zaklasyfikowanych przykładów. Natomiast *gamma* jest szerokością radialnej funkcji bazowej, używanej jako jądro sieci *SVM*. Para dobieranych parametrów, czyli aktualne rozwiązanie, będzie przedstawione jako genotyp jednego osobnika w populacji.

Dane są podzielone na zbiory trenujące, stosowane do stworzenia klasyfikatorów oraz testujące, przeznaczone do oceny uzyskanego modelu. Każdy zbiór składa się ze 102 przykładów, opisanych przez 36 atrybutów o charakterze liczbowym i z przypisaną kategorią, reprezentowaną przez liczbę całkowitą z zakresu {-62, -61, -52, -51, -42, -41, -32, -31, -22, -21, -12, -11, 0, 11, 12, 21, 22, 31, 32, 41, 42, 51, 52, 61, 62}. W związku z tym, że mamy do czynienia z klasyfikacją wieloklasową, ostateczny model powstanie jako połączenie $n*(n-1)/2$ binarnych *SVM* z kodowaniem *one-versus-one*, gdzie *n* oznacza liczbę kategorii, co będzie odpowiadało wprost liczebności populacji.

Opis zbiorów danych:

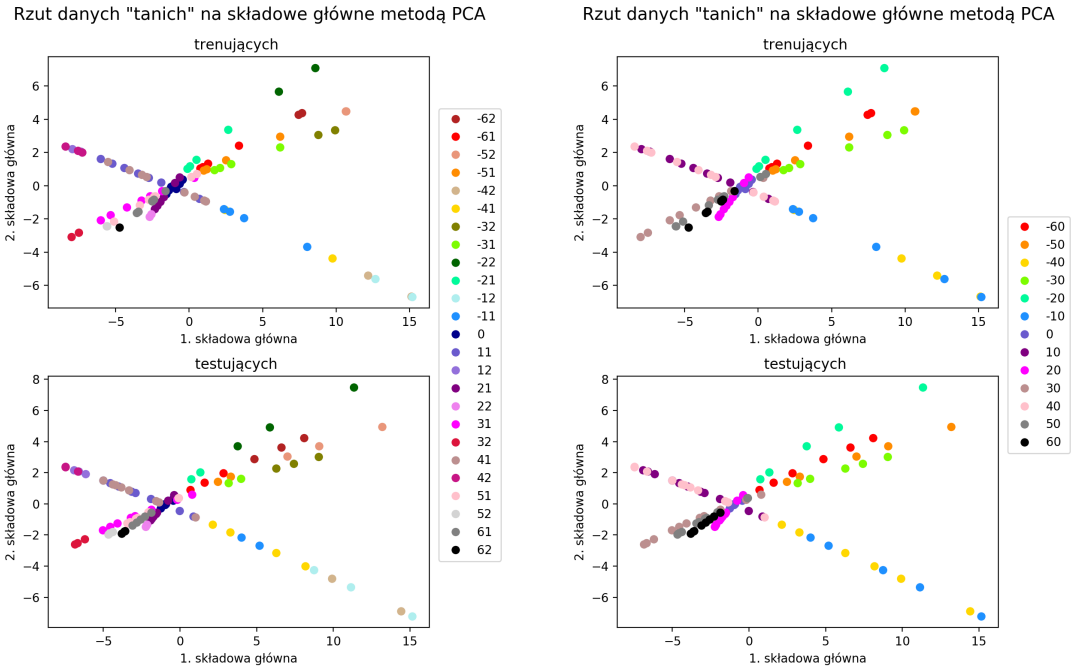
- zbiór "tani"
  Rozkład przykładów w klasach w zbiorze trenującym:

klasa	-62	-61	-52	-51	-42	-41	-32	-31	-22	-21	-12	-11
train	2	4	2	4	2	3	2	4	2	4	2	4
test	3	3	3	2	2	4	3	2	3	2	3	2

0	11	12	21	22	31	32	41	42	51	52	61	62
13	8	2	6	2	7	2	8	3	7	2	6	1
9	8	3	7	3	8	3	8	3	7	3	5	3

  Na rys. 1.1 przedstawiony został rzut danych "tanich"trenujących i testujących na składowe główne, wyznaczone przy użyciu metody PCA przy rozróżnieniu danych w poszczególnych klasach. Na wykres a) każda klasa odpowiada innej kategorii, natomiast na

wykresie b) zbliżone kategorie zostały złączone w jedną klasę (np. -60 reprezentuje -62 i -61).



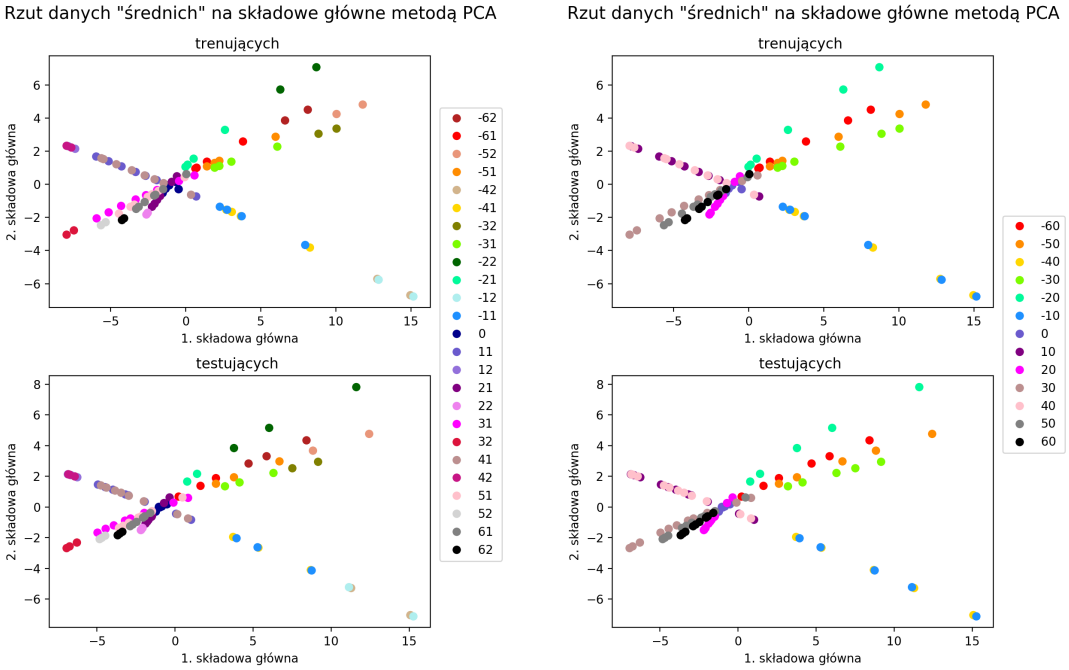
Rysunek 1.1: Zbiór tani

- zbiór "średni"
  Rozkład przykładów w klasach:

klasa	-62	-61	-52	-51	-42	-41	-32	-31	-22	-21	-12	-11
train	2	4	2	4	2	4	2	4	2	4	2	4
test	3	3	2	3	2	3	2	3	3	2	2	3

0	11	12	21	22	31	32	41	42	51	52	61	62
11	7	2	6	2	8	2	8	2	7	2	7	2
9	8	3	7	3	8	3	8	3	7	3	6	3

Na rys. 1.2 przedstawiony został rzut danych "średnich"trenujących i testujących na składowe główne, wyznaczone przy użyciu metody PCA przy rozróżnieniu danych w poszczególnych klasach. Na wykres a) każda klasa odpowiada innej kategorii, natomiast na wykresie b) zbliżone kategorie zostały złączone w jedną klasę (np. -60 reprezentuje -62 i -61).



Rysunek 1.2: Zbiór średni

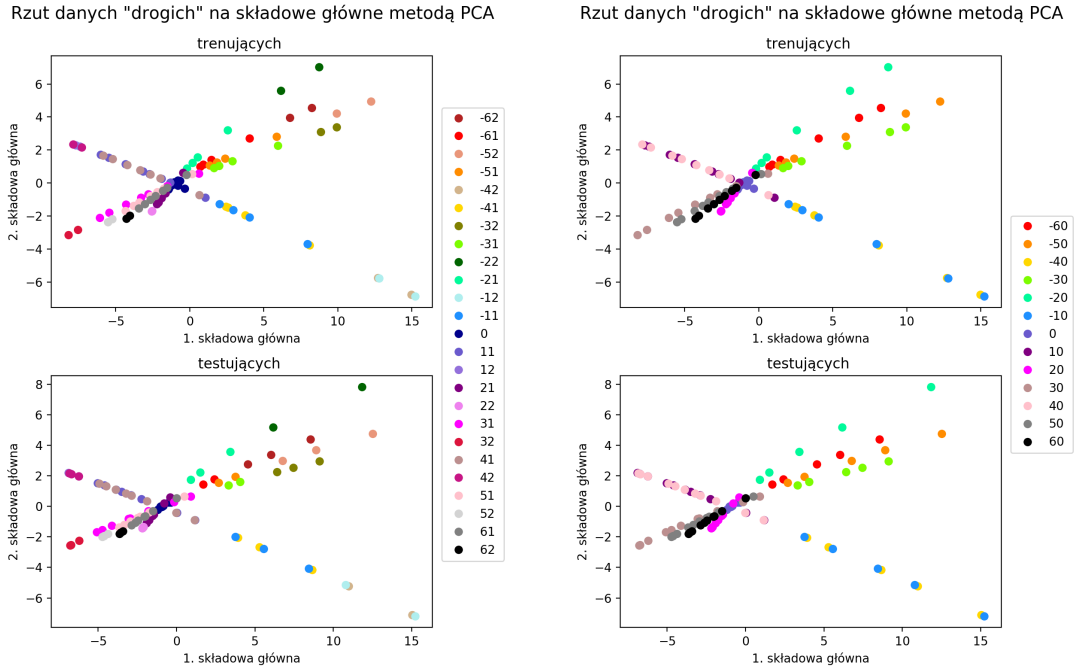
- zbiór "drogi
  Rozkład przykładów w klasach:

klasa	-62	-61	-52	-51	-42	-41	-32	-31	-22	-21	-12	-11
train	2	4	2	4	2	4	2	4	2	4	2	4
test	3	2	3	2	2	3	3	2	2	3	2	3

0	11	12	21	22	31	32	41	42	51	52	61	62
11	7	2	6	2	8	2	8	2	7	2	7	2
9	8	3	7	3	8	3	8	3	7	3	7	3

  Na rys. 1.3 przedstawiony został rzut danych "drogich"trenujących i testujących na składowe główne, wyznaczone przy użyciu metody PCA przy rozróżnieniu danych w poszczególnych klasach. Na wykres a) każda klasa odpowiada innej kategorii, natomiast na wykresie b) zbliżone kategorie zostały złączone w jedną klasę (np. -60 reprezentuje -62 i -61).

Dla każdego zbioru pierwsza składowa zawiera ok. 76% całkowitej wariancji, a druga ok. 15%. Analizując zobrazowane dane można zauważyć, że są one w większości rozróżnialne przy wykorzystaniu tych informacji, a najtrudniej separowalne pozostają przykłady należące do zbliżonych kategorii np. -62 i -61. Jeśli dane okażą się zbyt skomplikowane dla naszego programu, a przykładów zbyt mało dla stworzenia SVM, to posłużymy się zmniejszoną o połowę liczbą klas, jak przedstawiono na wykresach b).



Rysunek 1.3: Zbiór drogi

2 Projekt algorytmu ewolucyjnego

Definiując algorytm ewolucyjny należy wybrać:

- kodowanie, czyli sposób reprezentacji problemu,
- funkcję dopasowania,
- metodę inicjacji populacji,
- operatory ewolucyjne,
- rodzaj selekcji (reprodukcji i sukcesji),
- kryterium stopu.

Z uwagi na fakt, że optymalizowane parametry C i *gamma* są liczbami rzeczywistymi, naturalnym sposobem kodowania jest reprezentacja zmiennoprzecinkowa. Przykładowy osobnik to wektor liczb, na przykład $\begin{bmatrix}0.87 & 0.44\end{bmatrix}$, gdzie pierwsza wartość odpowiada parametrowi C, a druga parameterowi *gamma*.

Wyznacznikiem jakości klasyfikatora może być dowolna metryka zaimplementowana w klasie *sklearn.metrics*. W przytoczonym niżej przykładzie użycia algorytmu na zbiorze *iris* użyto funkcji *neg log loss*. Stanowi ona funkcję oceny dopasowania danego osobnika.

Zdecydowano się na selekcję turniejową z liczbą uczestników $q = 4$. Taka reprodukcja przebiega dwustopniowo. W każdym kroku wybierana jest najpierw podpopulacja zawierająca q osobników z populacji P^t. Wybrano wariant losowania osobników bez zwracania. Oznacza to, że $q-1$ najgorzej przystosowanych osobników nie będzie miało szansy na reprodukcję. Wszystkie q-elementowe kombinacje z P^t są jednakowo prawdopodobne. Następnym etapem jest przeprowadzenie turnieju, którego zwycięzcą zostaje osobnik najlepiej przystosowany. Osobnik kopiowany jest do populacji potomnej. Proces losowania osobników i rozgrywania turnieju wykonywany jest wielokrotnie, aż do zapełnienia populacji potomnej.

Zaimplementowano krzyżowanie uśredniające w wariancie alternatywnym. Uśrednianie przeprowadzane jest według schematu:

$$Y_i = X_i^1 + \xi_{U(0,1),i}(X_i^2 - X_i^1),$$

gdzie:

- X_i^1 - i-ty gen chromosomu X^1 (pierwszego rodzica),
- X_i^2 - i-ty gen chromosomu X^2 (drugiego rodzica),
- Y_i - i-ty gen chromosomu Y (osobnika potomnego),
- $\xi_{U(0,1),i}$ - zmienna losowa o rozkładzie jednostajnym.

W tym wariancie realizacja zmiennej losowej o rozkładzie jednostajnym następuje osobno dla każdego genu. Drugi osobnik potomny powstaje w następujący sposób:

$$\mathbf{Z} = \mathbf{X}^2 + \mathbf{X}^1 - \mathbf{Y}.$$

Kolejnym wybranym operatorem ewolucyjnym jest operator mutacji. Mutacja przeprowadzana jest poprzez dodanie do wektora osobnika wektora będącego n-wymiarową realizacją zmiennej losowej o rozkładzie Cauchy'ego, który, w przeciwieństwie do rozkładu normalnego, pozwala na generowanie dużych wartości.

Po operacjach krzyżowania i mutacji populacja potomna uzupełniana jest o dwóch najlepszych osobników z poprzedniej generacji. Jest to sukcesja elitarna.

Kryterium stopu jest maksymalna dopuszczalna liczba iteracji. Wybór tego parametru pozostawiony jest użytkownikowi.

3 Założenia implementacyjne

Algorytm zaimplementowany jest w języku Python w postaci samodzielnej klasy (*class EvoAlgo*). Moduł jest kompatybilny z wersjami języka 2.7, 3.5 i nowszymi.
Do budowy algorytmu wykorzystano następujące (kolejność alfabetyczna), ogólnodostępne moduły Pythona:

- *matplotlib* - biblioteka do tworzenia wykresów,
- *NumPy* - biblioteka numeryczna,
- *pathos* - framework ułatwiający obliczenia równoległe,
- *scikit-learn* - biblioteka do uczenia maszynowego,
- *tqdm* - moduł wyświetlający pasek postępu w terminalu.

Oprócz tego wykorzystano bibliotekę standardową języka Python, głównie strukturę danych *list* oraz kolekcję *namedtuple*.

4 Opis implementacji

Plik *evoalgo_svm.py* zawiera implementację algorytmu ewolucyjnego do doboru parametrów `SVM` wraz z przykładowym wywołaniem dla znanego (i niewielkiego) zbioru danych *iris*.

Zaprojektowany algorytm składa się z następujących metod:

- *__init__* - konstruktor, przyjmujący jako parametry: klasyfikator bazowy, rodzaj walidacji krzyżowej, ilość osobników w populacji, ilość iteracji (kryterium stopu),
- *_create_individual* - zwraca osobnika, czyli wektor dwóch liczb zmiennoprzecinkowych,
- *_create_population* - tworzy populację poprzez zapełnienie macierzy wektorami zwracanymi przez wywołanie metody *_create_individual*,
- *_get_fitness* i *_score_ind* - ocenia przystosowanie danego osobnika jako średnią wyników z *k-fold CV*,
- *_select_parents* - przeprowadza selekcję turniejową, czyli wybiera rodziców do reprodukcji,
- *_crossover* - wykonuje krzyżowanie uśredniające w wariancie alternatywnym,
- *_mutate* - wykonuje mutację poprzez dodanie wektora, który jest realizacją zmiennej losowej o rozkładzie Cauchy'ego,
- *_create_next_generation* - tworzy nową populację, korzysta z wcześniej wymienionych metod,
- *fit* - "serce" algorytmu, jedyna metoda publiczna w interfejsie klasy (w języku Python metody rozpoczynające się od znaku "_" umownie traktowane są jako prywatne, choć sam język tego nie wymusza), szuka najlepszych parametrów dla optymalizowanego algorytmu `SVM`.

Dodatkowo dostępne są funkcje pomocnicze:

- *get_params* - zwraca wartości parametrów *C* i *gamma* najlepszego osobnika oraz wartość jego przystosowania,
- *plot* - wyświetla wykres wyników w funkcji numeru generacji przedstawiający: wynik bazowy klasyfikatora, wynik średni danej generacji i wynik najlepszego osobnika w danej generacji, przy czym wynikiem jest wartość wybranej wcześniej metryki oceniającej jakość klasyfikacji,
- *plot_gen* - umożliwia zobaczenie wygenerowanych osobników na wykresie dla każdej generacji. Przykładowe działanie algorytmu przedstawia załączony plik *animation.gif*, stworzony na podstawie uzyskanych (i zapisanych) wykresów dla poszczególnych generacji za pomocą polecenia *convert -delay 10 -loop 0 *.png animation.gif* w systemie `Linux`.