Las losowy w zadaniu klasyfikacji z selekcją progową oraz ruletką

Autor: Rafał Lewanczyk

1. Cele projektu:
    a. Implementacja w języku Python lasu losowego, w którym podczas każdego z drzew testy wybierane są za pomocą selekcji progowej w połączeniu z ruletką (odrzucamy testy o prawdopodobieństwie wybrania mniejszym od zadanego progu następnie losujemy z prawdopodobieństwem wyboru zależnego od jakości testu).
    b. Ocena zaimplementowanego modelu.
    c. Porównanie zaimplementowanego modelu ze standardowym algorytmem lasu losowego używającego algorytmu ID-3 do budowy pojedynczego drzewa

2. Zastosowane algorytmy :
    a. Las losowy:
    Do budowy lasu losowego został zastosowany algorytm zaproponowany przez Breimana z uwzględnieniem ruletki. Drzewa składające się na las uzyskujemy w następujący sposób:

    1. Losujemy ze zwracaniem podzbiór danych z dostępnej próby uczącej
    2. Utwórz drzewo z wylosowanego podzbioru
    - Sprawdzamy, czy dzielony zbiór jest jednorodny lub zbyt mały
    - Losujemy pewną liczbę zmiennych, na podstawie których będziemy szukali
    kandydata do podziału
    - Z wylosowanego zbioru obliczamy informację uzyskaną przez podział dla
    danego atrybutu, wyznaczamy prawdopodobieństwo wyboru, usuwamy
    atrybuty z prawdopodobieństwem mniejszym od progu odcięcia,
    wybieramy kandydata przy pomocy ruletki.
    3. Powtarzamy punkty 1,2 aż uzyskamy żądaną liczbę drzew
    4. Dodatkowe założenia przy budowie lasu:

- Przy braku możliwości podziału według atrybutu oraz sprzecznych klasach
przykładów w liściu jest umieszczana najczęściej powtarzająca się klasa.
Jeśli nie ma jednej najczęściej powtarzającej się klasy zostaje wybrana
losowa klasa spośród najczęściej powtarzających się.

Klasyfikacja jest wykonywana na podstawie głosowania wszystkich drzew, w
przypadku równego rozdziału głosów wybierana jest losowa klasa spośród
klas o największym numerze głosów.

b. K-Krotna walidacja krzyżowa
Ocena działania algorytmu zostanie wykonana przy pomocy K-krotnej walidacji krzyżowej.

1. Oryginalną próbę dzielimy na K podzbiorów. Następnie każdy z nich bierzemy jako zbiór testowy, a pozostałe razem jako zbiór uczący oraz wykonujemy analizę. Błąd jest średnim błędem z uzyskanych K-błędów

3. Zbiory danych
   a. Do testów algorytmu zostaną użyte 3 zbiory danych:
      i. *Cars.data* największy zbiór danych zawierający 1729 przykładów, 6 atrybutów oraz 9 klas
      ii. *Qualitive_Bankruptcy.data* średni zbiór zawierający 250 przykładów, 6 atrybutów oraz 2 klasy
      iii. *Lenses.data mały zbiór zawierający 23 przykłady, 4 atrybuty oraz 3 klasy*
      iv. *Primary-tumor.data* średni zbiór zawierający 338 przykładów, 17 atrybutów i 22 klasy
      v. *Golf.data* najmniejszy zbiór zawierający 14 przykładów, 4 atrybutów oraz 2 klasy

Zbiory danych zostały dobrane tak aby sprawdzić działanie algorytmu w zależności od liczby przykładów oraz liczby dostępnych klas. Rozmiar największej z klas został ograniczony koniecznością przeprowadzenia wielu testów na każdym ze zbiorów.

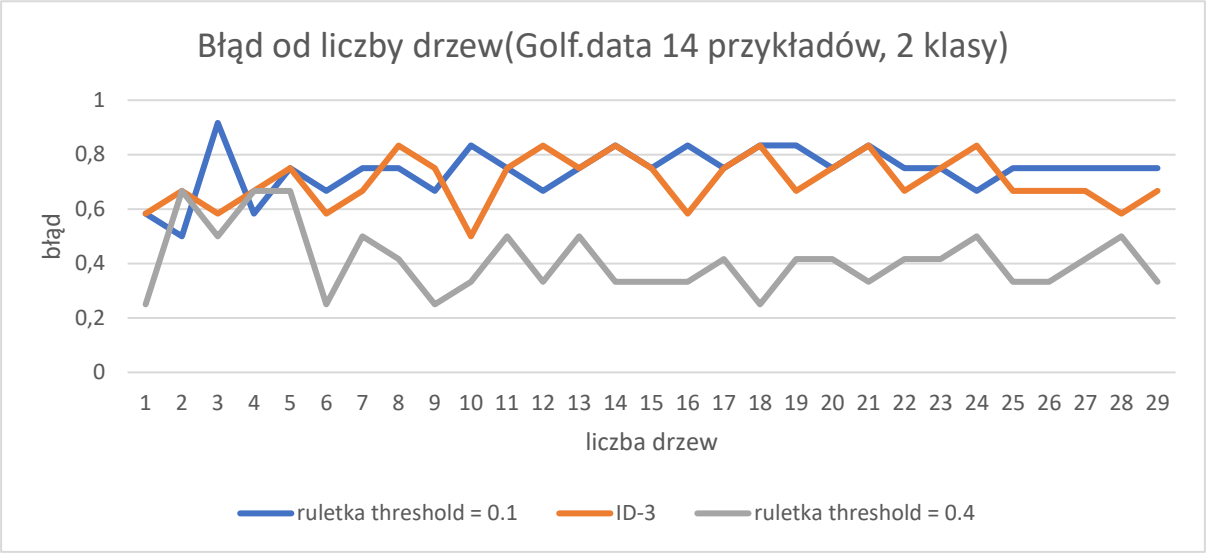
Zbiór danych musi być w formacie:
- pierwszy rząd nazwy atrybutów oddzielone przecinkami gdzie ostatni atrybut to klasa
- pozostałe rzędy to wartości atrybutów poooddzielane przecinkami

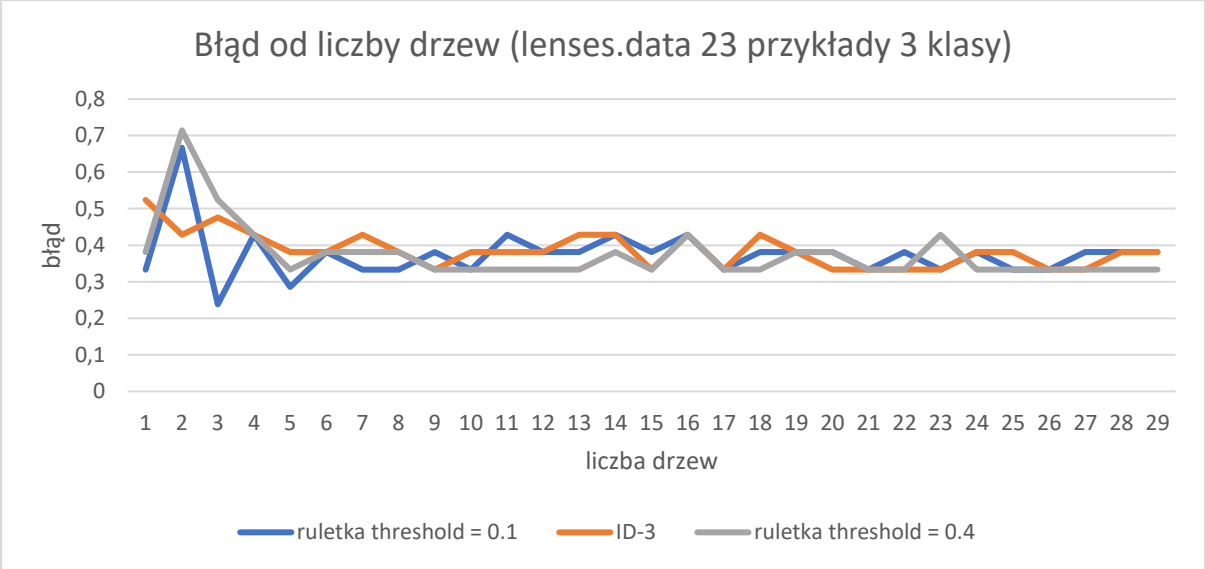
4. Przeprowadzone eksperymenty
    a. Na każdym ze zbiorów zostanie przeprowadzone badanie dla 2 lasów z ruletką dla (dużego 0.4 i małego 0.1) punktu odcięcia, standardowego algorytmu ID-3 oraz liczby drzew od 1 do 30
    b. Algorytmy będą oceniane przez 3-Krotną walidację krzyżową

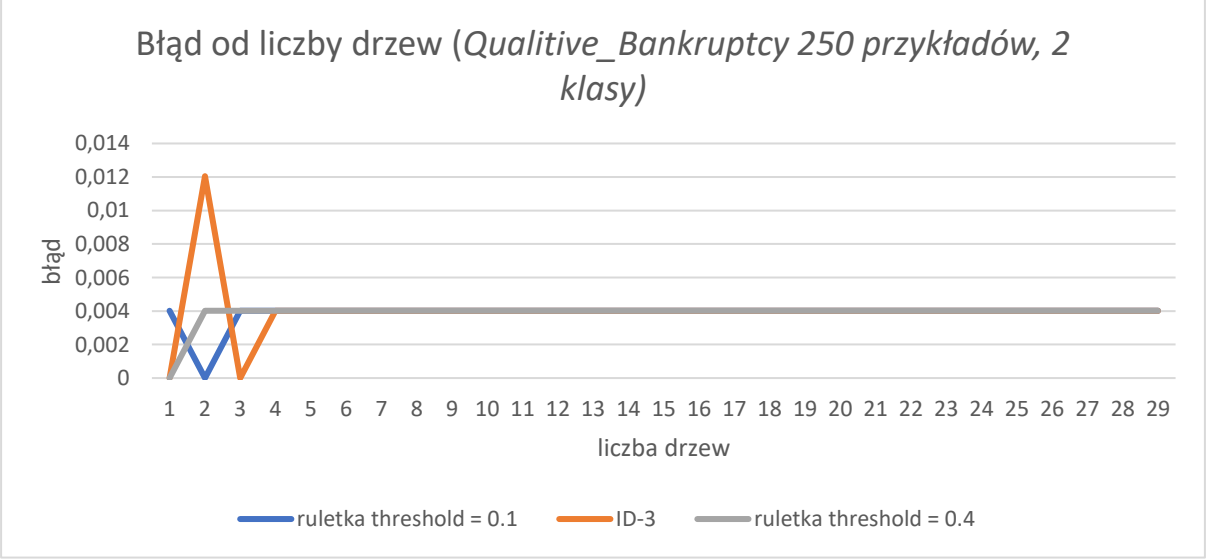
5. Uruchomienie oraz wyjście programu
    a. Program przy uruchomieniu przyjmuje dwa argumenty
        i. -d : ścieżka do pliku ze zbiorem danych
        ii. -k : współczynnik K w K-krotnej walidacji krzyżowej

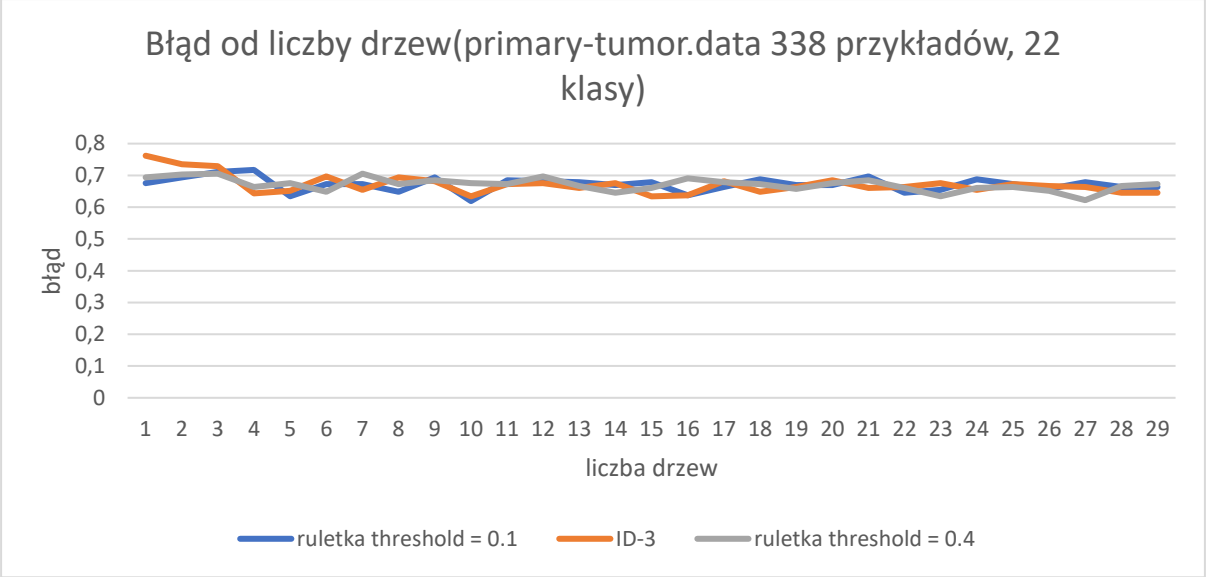
        Np. *python main.py -d data/Golf.data -k 3*
    b. Program na wyjściu wypisuje 3 wektory długości 30 zawierające średnie błędy dla odpowiednio 1 … 30 drzew w lesie dla ruletki z w wartościami punktu odcięcia 0.1 i 0.4 oraz dla standardowego algorytmu ID-3

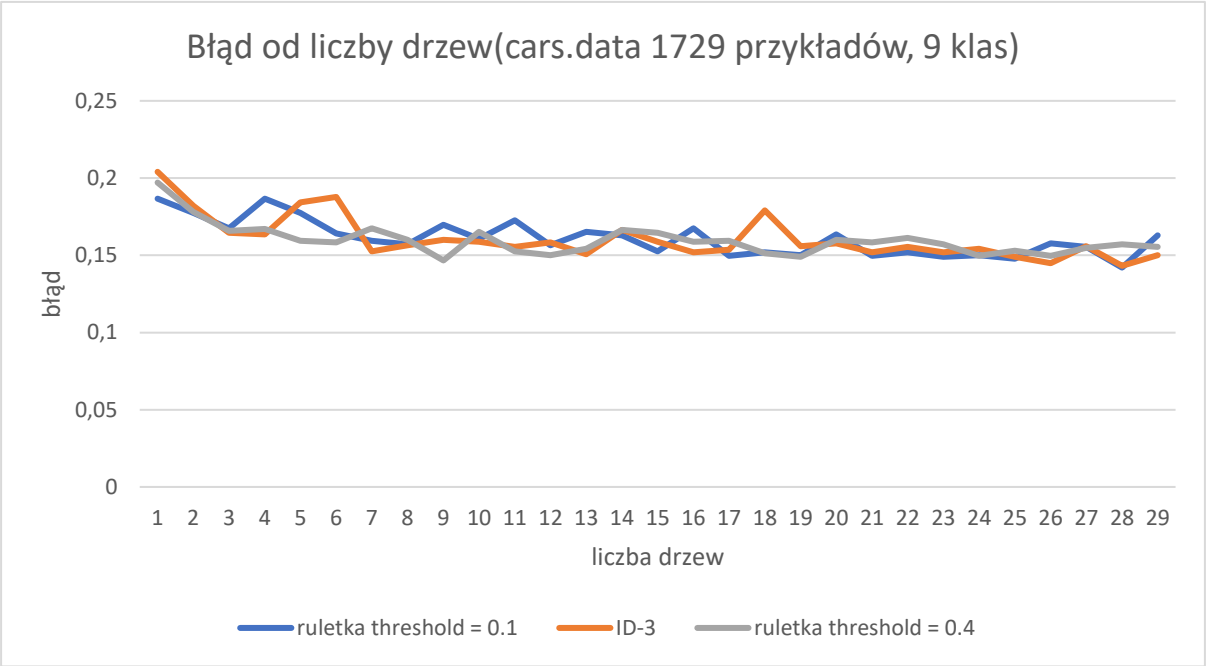
6. Wyniki eksperymentów











7. Wnioski
    a. Jak można zauważyć z wykresów skuteczność naszego klasyfikatora w bardzo dużej mierze zależy od zbioru danych na jakim zostanie uruchomiony. Jendak oceniam klasyfikator za skuteczny.
    b. Wysokie wyniki błędów na zboirze *primary-tumor.data* wynikaja z bardzo dużej ilości klas (znacznie większa szansa że klasyfikator popełni pomyłkę), dlatego średnią wielkość błedu w okolicach ~65% uważam za zadawalającą, pokazuje ona że udało się klasyfikatorowi znaleźć zależności między danymi i nie działa w sposób losowy.

c. Na wykresach możemy zauważyć, że niezależnie od użytego algorytmu budowy drzewa, błąd osiąga swoje minimum w okolicach lasu składającego się z 10 drzew a następnie oscyluje wokół tej wartości.
d. Standardow algorytm ID-3 budowy drzewa porównywalnie z algorytmami ruletkowymi w niektórych przypadkach np. cars.data działa minimalnie lepiej od innch natomiast sa przypadki dla których działa znacznie gorzej np. Golf.data. Wynika to z tego że w niektórych zbiorach istnieją silne zależności pomiędzy konkretnymi atrybutami, które odzielnie oceniane są słabo jednak w zbiorze (na jednej ścieżce) niosą dużo informacj. Standardowy algorytm ID-3 nie ma możliwości skonstrułowania takiej ścieżki, natomiast algorytmy z ruletką mają szansę taką scieżkę wylosować.
e. Wartość punktu odcięcia 0.1 na wszystkich pokrywa się z algorytmem ruletkowym o punkcie odcięcia 0.4 oprócz na danych Golf.data oraz Lenses.data gdzie wrtość 0.4 uzyskuje lepsze wyniki, dla danych Golf.data nawet o 40 punktów procentowych. Zauważmy równie że dla tych zbiorów ruletka o punkcie odcięcia 0.1 wypda najgorzej zatem atrybuty o małym przyroście informacji sa faktycznie najgorszym wyborem ale za to losowość wprowadzona przez ruletke o duzej wartości odcięcia poprawia działanie standardowego algorytmu.

8. Użyte biblioteki
    a. Wszystkie użyte biblioteki zostały zawarte w pliku *requirements.txt*