1 Treść zadania

Las losowy w zadaniu klasyfikacji. Podczas budowy każdego z drzew testy (z pełnego zbioru testów) wybieramy za pomocą selekcji progowej w połączeniu z ruletką, czyli odrzucamy część najgorszych, po czym każdy z pozostałych testów ma proporcjonalną do swojej jakości szansę na bycie wybranym.
Nasza interpretacja treści zadania:
Naszym celem jest napisanie programu, który na podstawie zadanego zbioru danych testowych tworzy las losowy drzew decyzyjnych. Następnie, z jego wykorzystaniem, jest w stanie klasyfikować podany przykład. Ruletkę zaimplementujemy na poziomie doboru warunku dla kolejnych węzłów drzew.

2 Wstęp teoretyczny

2.1 Drzewa decyzyjne

Struktura danych złożona z węzłów, połączonych krawędziami (gałęzie), prowadzącymi do kolejnych poziomów drzewa - bezpośrednio do liści, w taki sposób że dwa dowolne węzły połączone są tylko jedną ścieżką. Macierzysty węzeł (bez rodzica) jest korzeniem, natomiast ostatni poziom węzłów nazywany jest liśćmi. W kontekście algorytmów drzewa można interpretować w następujący sposób: węzły - testy przeprowadzone na wartościach atrybutów przykładów, gałęzie - możliwe wyniki przeprowadzonych testów, liście - etykiety kategorii. Dzięki temu pozwalają klasyfikować przykłady o zadanych atrybutach. Pojedyncze drzewa są podatne na nadmierne dopasowanie, dlatego są bardzo dobrym przykładem do wykorzystania lasów losowych. Dodatkowo korzystne jest stosowanie warunków ograniczających tj. maksymalna głębokość drzewa (liczba poziomów drzewa). [1]

2.2 Lasy losowe

Stanowią kolekcję modeli, w tym przypadku utworzonych drzew decyzyjnych, i wykorzystywane są w koncepcji **Ensemble learning**. Takie podejście pozwala uzyskać zamiast kilku klasyfikatorów, jeden dokładniejszy (czyli minimalizuje niedokładność pojedynczych modeli). W celu utworzenia lasu wykorzystywana jest metoda *bagging'u*.

2.2.1 Bagging

Metoda generuje nowe modele drzew decyzyjnych, które następnie agreguje, aby uzyskać jeden klasyfikator. Pierwszym jej etapem jest *Bootstraping*. Generowane są nowe zestawy danych (z możliwymi powtórzeniami) bazując na danych trenujących. Następnie na podstawie tak utworzonych zbiorów uczone są nowe modele (nowe drzewa decyzyjne dla T_i zestawu, uczony jest M_i model). Końcowym etapem jest *aggregation*, gdzie w celu sklasyfikowania zadanego przykładu wyniki z wszystkich modeli w lasie są 'agregowane' i finalnie przypisywana przykładowi jest najliczniejsza klasa. [2]

2.2.2 Gini index

Wskaźnik używany przy budowie drzew decyzyjnych, pozwala na obliczenie jakości, a więc na wybranie optymalnego warunku, sprawdzanego w kolejnym węźle modelu po podziale. Wyraża się wzorem:

$$G = 1 - \sum_{i=1}^{n} p_i^2 \tag{1}$$

gdzie p_i oznacza prawdopodobieństwo wybrania elementu $i-tej$ klasy. W tym celu można wykorzystać też entropię, jednak użyjemy wyżej opisanego index'u z uwagi na lepszą optymalizację - uproszczone obliczenia (nie zawiera logarytmów jak entropia). [3] W dalszej części posługujemy się także określeniem całkowitej wartości wskaźnika Gini:

$$Gw = \frac{a}{a+b} \cdot G_1 + \frac{b}{a+b} \cdot G_0$$

gdzie: Gw oznacza całkowitą wartość wskaźnika Gini, a oznacza liczbę elementów spełniających dany warunek, b oznacza liczbę elementów niespełniających danego warunku, G_1 oznacza wskaźnik Gini dla podzbioru spełniającego dany warunek, G_0 oznacza wskaźnik Gini dla podzbioru niespełniającego dany warunek. Posługujemy się także terminem zmiany wskaźnika Gini:

$$\Delta G = G_r - Gw$$

gdzie: ΔG oznacza zmianę wskaźnika Gini, G_r oznacza wskaźnik Gini dla całego zbioru danych, Gw oznacza całkowitą wartość wskaźnika Gini dla danego wyboru.

3 Implementacja

3.1 Planowana implementacja - narzędzia

Poniżej opisaną implementację algorytmu chcielibyśmy wykonać w języku Python, z wykorzystaniem bibliotek do manipulacji danymi i ich prezentacji, tj. NumPy, Pandas, Matplotlib/Seaborn.
Kod będziemy przechowywać/wersjonować na wydziałowym GitLab'ie. Wyniki naszej pracy porównamy z rozwiązaniem przykładowym dla algorytmu CART wybierającego najlepsze atrybutu z biblioteki $scikit - learn\_1.3.2$.

3.2 Drzewa decyzyjne

Po przeanalizowaniu używanych algorytmów drzew decyzyjnych zdecydowaliśmy się oprzeć nasze rozwiązanie na popularnym algorytmie CART. Oryginalna implementacja algorytmu zakłada podział danych testowych na węźle decyzyjnym na dwa podzbiory wedle warunku wskazanego przez wskaźnik Gini. Ze wszystkich wartości wskaźnika obliczanych dla każdego atrybuty wybierany jest ten z najmniejszą wartością. W naszym wariancie algorytmu, dla każdego atrybutu rozdzielamy nasz zbiór testowy na dwa podzbiory wedle warunku utworzonego na podstawie wartości atrybutu, następnie dla każdego takiego podziału, wyliczana jest wartość wskaźnika Gini, kolejno wyliczana jest wartość całkowitej wartości wskaźnika Gini poprzez sumę iloczynów wag wskaźnika Gini oraz wskaźnika Gini. Po tym kroku usuwane są te warunki, których wskaźnik Gini znajduje się ponad progiem, a z pozostałych warunków losowo wybierany (ruletka) jest ostateczny warunek z prawdopodobieństwem równym proporcji wartości różnicy wskaźnika Gini dla całego zbioru oraz wskaźnika Gini dla danego wyboru. Następnie, wedle tego wyboru dane są rozdzielane na dwa podzbiory, spełniające dany warunek i niespełniające. Kolejno, przekazywane są rekurencyjnie do następnych węzłów w celu utworzenia drzewa. W procesie tworzenia drzewa decyzyjnego istnieje kilka warunków zakończenia algorytmu: przekroczenie przez wartość głębokości maksymalnej wartości głębokości drzewa, osiągnięcie przez liczność podzbioru minimalnej wartości liczby elementów, utworzenie przez wybór podzbioru pustego. Wystąpienie, któregokolwiek z tych warunków, powoduje zakończenie dalszej generacji węzłów decyzyjnych i ustalenie tego węzła jako liść z etykietą wybieraną większościowo z etykiet w zbiorze, lub losowo w momencie braku możliwości wyboru większościowego. Kolejnym warunkiem zakończenia procesu generacji węzłów decyzyjnych jest utworzenie podzbioru danych z tylko jedną etykietą, w tym przypadku węzeł przerabiany jest na liść z wartością etykiety równą wartości jedynej występującej klasy w podzbiorze.

Algorithm 1 Pseudokod budowy drzewa decyzyjnego

```
1: function CREATEDECISIONTREE(T,X,d) return root
2:     decisionTreeNode = new Node
3:     if all elements of T are of some class c then
4:         Set decisionTreeNode value = c
5:         return decisionTreeNode
6:     else if number of elements of T < min_elements OR d > max_depth then
7:         Set DecisionTreeNode value = selectBestClass(T)
8:         return decisionTreeNode
9:     end if
10:    dataBaseIndex = calculateGeneralGiniIndex(T)
11:    for all elementType in X do
12:        for all element in elementType do
13:            temp = calculateGiniWeightedIndex(element,T)
14:            indexChangeCollection = dataBaseIndex - temp
15:        end for
16:    end for
17:    dataBaseIndexPruned = threshold(indexChangeCollection)
18:    if dataBaseIndexPruned is empty then
19:        Set decisionTreeNode value = selectBestClass(T)
20:    end if
21:    selection = rouletteSelection(dataBaseIndexPruned)
22:    for all element value x of selection type do
23:        if x = selection then
24:            T_1 add x
25:        else
26:            T_0 add x
27:        end if
28:    end for
29:    if T_1 is empty OR T_0 is empty then
30:        Set DecisionTreeNode value = selectBestClass(T)
31:        return decisionTreeNode
32:    end if
33:    Set decisionTreeNode leftNode = createDecisionTree(T_1,X,d+1)
34:    Set decisionTreeNode rightNode = createDecisionTree(T_0,X,d+1)
35:    return decisionTreeNode
36: end function
```

gdzie:

- T - zestaw danych, X - zbiór atrybutów, d - depth (liczba poziomów drzewa)
- selectBestClass(T): Wybiera najczęściej występującą klasę w zbiorze, jeżeli nastąpi remis wybiera losowo
- calculateGeneralGiniIndex(T) - oblicza Gini index dla całego zbioru danych
- calculateGiniWeightedIndex(element,T): Oblicza Gini weighted Index dla każdego elementu, ustalonego typu
- indexChangeCollection - zbiór różnic bieżącego indexu Gini z wartością ogólną
- dataBaseIndexPruned - zbiór wyboru po zastosowaniu selekcji progowej
- rouletteSelection() - funkcja wybierająca warunek z użyciem ruletki
- selection - wybrany warunek za pomocą rouletteSelection()
- T_1, T_0 - podzbiory danych odpowiednio spełniające i nie spełniające danego warunku węzła

Atrybuty mogą być dyskretne lub ciągłe, wpływa to na sposób dla nich wartości gini index'u. W przypadku dyskretnych podejście jest jedno, ale dla atrybutów ciągłych spotkaliśmy się z kilkoma sposobami tj. liczenie średnich lub mediany. Zdecydowaliśmy się na opcję ze średnimi, dwóch kolejnych wartości, wykorzystywaną w algorytmie C4.5.

3.3 Las losowy

Pierwszym etapem budowy lasu losowego jest utworzenie N modeli. Dla każdego drzewa, tworzymy osobny zbiór treningowy (bootstrapping) oraz wybieramy losowe atrybuty i ich liczbę. Warty zaznaczenia jest fakt, że wszelkie losowania odbywają się ze zwracaniem, tzn. w drzewie możemy mieć np. kilka warunków z wykorzystaniem tego samego atrybutu. Następnie, gdy mamy stworzony las, każdy przykład ze zbioru K klasyfikujemy za pomocą wszystkich drzew. Finalną klasą przypisywaną do przykładu k, jest etykieta najczęściej występująca w zbiorze po agregacji odpowiedzi ze wszystkich modeli.

Algorithm 2 Pseudokod budowy lasu losowego

```
1:  function RANDOMFOREST(T, N, K) return finalClasses
2:      listOfModels = empty array
3:      for i in 1...N do
4:          T_i = bootstrap(T)
5:          choosenAtributes = random(T_i, m)
6:          add createDecisionTree(T_i, choosenAtributes, 0) to listOfModels
7:      end for
8:      finalClasses = empty array
9:      for all k in K do
10:         classes = empty array
11:         for all model in listOfModels do
12:             class <- classification of k of one model
13:             add class value to classes
14:         end for
15:         finalClass = getTheMostCommonValue(classes)
16:         add finalClass to finalClasses
17:     end for
18:     return finalClasses
19: end function
```

gdzie:

- T - zbiór trenujący, N - liczba drzew do utworzenia, K - przykłady do sklasyfikowania
- listOfModels - struktura danych przechowująca wszystkie utworzone modele w ramach lasu
- bootstrap(T) - metoda tworząca nowy zestaw danych w każdej iteracji, na podstawie wejściowego zestawu danych - z możliwością powtarzania
- m - liczba atrybutów do wylosowania
- choosenAtributes - losowo wybrane atrybuty ze zbioru T_i przez funkcję random(), przekazane jako parametr do utworzenia drzewa
- classes - struktura danych przechowująca wszystkie etykiety klas, które wystąpiły
- getTheMostCommonValue() - funkcja agregująca te same wartości i zwracająca liczbę wystąpień najczęstszej. W przypadku takiej samej wartości, finalna wybierana jest w sposób losowy.
- finalClasses - zwraca wszystkie klasy wyznaczone przez algorytm

3.4 Przykład obliczeniowy

Najlepszym opisem sposobu działania algorytmu budowy lasu losowego, będzie prześledzenie procesu budowy jednego z jego drzew decyzyjnych. Za zbiór danych treningowych posłuży nam lekko zmodyfikowany zbiór Zbiór danych, zamienione w nim zostały wartości temperatury z wartości dyskretnych na wartości ciągłe.
Atrybuty:

- Outlook $\in$ { Sunny, Overcast, Rain }
- Temperature $\in$ [5,20]
- Humidity $\in$ { High, Normal }

- Windy $\in$ { Weak, Strong }

Etykiety:

- Play $\in$ { Yes, No }

Nr.	Outlook	Temperature	Humidity	Windy	Play
1	Sunny	19	High	Weak	No
2	Sunny	15	High	Strong	No
3	Overcast	11	High	Weak	Yes
4	Rain	9	High	Weak	Yes
5	Rain	7	Normal	Weak	Yes

Idąc zgodnie z założeniami algorytmu, początkowo losowo wybieramy $\sqrt{4} = 2$ atrybutów dla danego drzewa. Załóżmy, że wybrane zostały atrybuty: outlook oraz temperaturę.Najpierw wyliczmy wskaźnik Gini dla całego zbioru danych

$$G(T) = 1 - (\frac{2}{5})^2 - (\frac{3}{5})^2 = 0,48$$

Wyliczamy teraz całkowity wskaźnik Gini, który jest średnią ważoną ilości elementów zbioru danych oraz wskaźnika Gini, dla każdej z dyskretnych wartości atrybutu Outlook. Jako że jest to algorytm drzewa binarnego, przyjmujemy prosty warunek, albo wartość jest równa wartości argumentu, albo nie (Sunny == Sunny, Rain != Sunny):

$$G(Sunny, T) = (1 - (\frac{2}{2})^2) \cdot \frac{2}{5} + (1 - (\frac{3}{3})^2) \cdot \frac{3}{5} = 0,00$$
$$G(Overcast, T) = (1 - (\frac{1}{1})^2) \cdot \frac{1}{5} + (1 - (\frac{2}{4})^2 - (\frac{2}{4})^2) \cdot \frac{4}{5} = 0,40$$
$$G(Rain, T) = (1 - (\frac{2}{2})^2) \cdot \frac{2}{5} + (1 - (\frac{2}{3})^2 - (\frac{1}{3})^2) \cdot \frac{3}{5} = 0,27$$

Następnie dla atrybutu o wartości ciągłej sortujemy dane rosnąco: 7, 9, 11, 15, 19 oraz wyliczamy średnie arytmetyczne dwóch kolejnych temperatur $t_1 = 8, t_2 = 10, t_3 = 13, t_4 = 17$ (jest to przykład jednego ze sposobów podejścia do tego typu danych, może być to jeden z kilku trybów pracy algorytmu) i następnie dla każdej z wartości t_i wyliczamy całkowity wskaźnik Gini:

$$G(temperature < t_1, T) = (1 - (\frac{1}{1})^2) \cdot \frac{1}{5} + (1 - (\frac{2}{4})^2 - (\frac{2}{4})^2) \cdot \frac{4}{5} = 0,40$$
$$G(temperature < t_2, T) = (1 - (\frac{2}{2})^2) \cdot \frac{2}{5} + (1 - (\frac{1}{3})^2 - (\frac{2}{3})^2) \cdot \frac{3}{5} = 0,27$$
$$G(temperature < t_3, T) = (1 - (\frac{3}{3})^2) \cdot \frac{3}{5} + (1 - (\frac{2}{2})^2) \cdot \frac{2}{5} = 0,00$$
$$G(temperature < t_4, T) = (1 - (\frac{1}{4})^2 - (\frac{3}{4})^2) \cdot \frac{4}{5} + (1 - (\frac{1}{1})^2) \cdot \frac{1}{5} = 0,30$$

Następnie wyliczamy zmianę wskaźnika Gini dla każdego warunku:

$$\Delta G(Sunny, T) = 0,48 - 0,00 = 0,48$$
$$\Delta G(Overcast, T) = 0,48 - 0,40 = 0,08$$
$$\Delta G(Rain, T) = 0,48 - 0,27 = 0,21$$
$$\Delta G(temperature < t_1, T) = 0,48 - 0,40 = 0,08$$
$$\Delta G(temperature < t_2, T) = 0,48 - 0,27 = 0,21$$
$$\Delta G(temperature < t_3, T) = 0,48 - 0,00 = 0,48$$
$$\Delta G(temperature < t_4, T) = 0,48 - 0,30 = 0,18$$

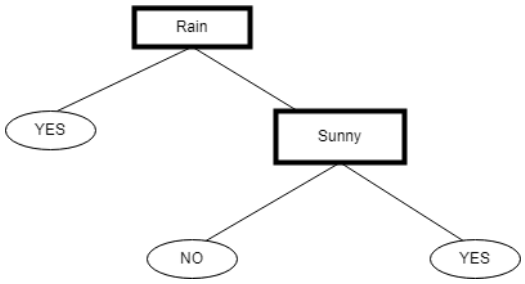
W tym momencie zachodzi selekcja wedle ustawionego progu np. usunięcie najgorszych 40% warunków, a następnie na zasadzie ruletki wybór ostatecznego warunku. Przykładowy wzór na wyliczenie prawdopodobieństwa wyboru danego warunku:

$$p(x) = \frac{\Delta G(x, T)}{\sum_{y \in X} \Delta G(y, T)}$$

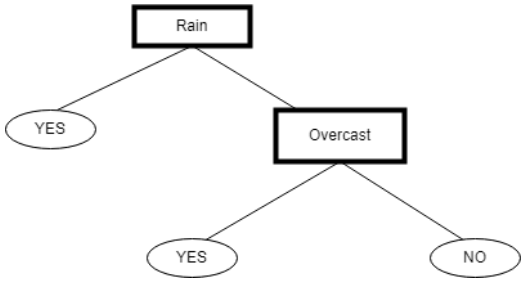
W naszym przypadku usunięcie 40% warunków oznacza usunięcie ≈ 3 najgorszych warunków. w ten sposób pozostaną nam następujące warunki z wyliczonymi szansami na zostanie wybranym:

$$p(Sunny) = \frac{0,48}{0,48+0,21+0,21+0,48} = 35\%$$
$$p(Rain) = \frac{0,21}{0,48+0,21+0,21+0,48} = 15\%$$
$$p(temperature < t_2) = \frac{0,21}{0,48+0,21+0,21+0,48} = 15\%$$
$$p(temperature < t_3) = \frac{0,48}{0,48+0,21+0,21+0,48} = 35\%$$

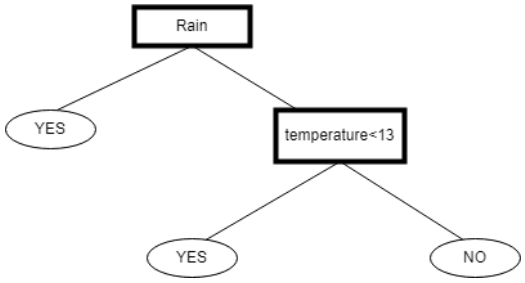
Na podstawie tych wartości prawdopodobieństwa losujemy, który warunek zostanie wybrany do węzła decyzyjnego. Następnie, z utworzonych przez kryterium wyboru dwóch podzestawów danych rekurencyjnie wybierane są następne warunki z wylosowanego wcześniej zestawu atrybutów, aż do momentu, w którym wszystkie elementy w zbiorze mają tę samą etykietę, wtedy węzeł taki staje się liściem o danej etykiecie, lub do momentu osiągnięcia maksymalnej, wcześniej ustalonej, głębokości drzewa lub osiągnięcia przez podzbiór danych minimalnej liczności elementów. W momencie osiągnięcia kryterium liczności węzeł taki staje się liściem, gdzie etykieta wybierana jest według większości elementów danego typu w podzbiorze. W przypadku równej liczby takich elementów etykieta wybierana jest losowo.
W wyżej przedstawionym przypadku mamy łącznie 70% (podział ze względu na Sunny oraz ze względu na t_3) na to, że zostanie wybrany warunek dzielący zestaw treningowy na dwa podzbiory z tylko jedną klasą etykiet w danym zbiorze, oraz 30% (podział ze względu na Rain oraz ze względu na t_2). Załóżmy, że wybrany został warunek ze względu na wartość atrybutu 'Rain'. Możemy otrzymać wtedy następujące drzewo:



Lub takie:



Lub takie:



To jakie drzewo ostatecznie otrzymalibyśmy po wywołaniu algorytmu zależy od ruletki.

4 Plan eksperymentów

W celu zbadania wpływu różnych parametrów na przebieg oraz wydajność działania naszego algorytmu stworzonego na bazie algorytmu CART ze zmienionym warunkiem doboru wyborów w węzłach decyzyjnych opartych na ruletce. Nasze pomysły na przeprowadzenie eksperymentów:

- zmiana liczby modeli drzew wykorzystanych do budowy lasu losowego w danej iteracji,
- zmiana liczby atrybutów dostępnych dla danego modelu w procesie budowy drzewa
- wpływ parametru depth przy budowie drzewa
- wpływ różnych metod wyliczania warunków dla węzłów decyzyjnych dla atrybutów ciągłych
- wpływ zmiany minimalnej liczby elementów w podzbiorze danych wymaganej do przeprowadzenia procesu tworzenia nowego węzła decyzyjnego

Dodatkowo porównamy wynik działania naszego algorytmu z algorytmem typu CART wybierającym wybory w węzłach decyzyjnych na podstawie najmniejszej wartości wskaźnika Gini. Ostatecznie wyniki będziemy porównywać według metryk jakości opisanych w kolejnym rozdziale.

5 Miary jakości

5.1 Tabela pomyłek i dokładność algorytmu

Do oceny jakości, a właściwie dokładności modelu będziemy posługiwać się *confusion matrix*, czyli tabelą pomyłek. [5] Taka tabela składa się z następujących pól:

Wynik algorytmu	Klasa rzeczywista	
	Pozytywna	Negatywna
Pozytywny	TP	FN
Negatywny	FP	TN

Gdzie:

- `TP` - true positive - przykłady prawdziwie pozytywne
- `TN` - true negative - przykłady prawdziwie negatywne
- `FP` - false positive - przykłady fałszywie pozytywne
- `FN` - false negative - przykłady fałszywie negatywne

Z wykorzystaniem tych wartości możemy obliczyć następujące wskaźniki:

- Accuracy/Dokładność (ACC): $\frac{TP+TN}{TP+TN+FP+FN}$
- True positive rate/Czułość/recall (TPR): $\frac{TP}{TP+FN}$
- Precision/Precyzja (PPV):$\frac{TP}{TP+FP}$
- True negative rate (TNR): $\frac{TN}{FP+TN}$
- false alarm rate (FAR): $\frac{FP}{TN+FP}$

Poza FAR oczekujemy od dobrego algorytmu jak najwyższych wartości. Dodatkowo, możemy wyznaczyć:

F_1: $2 \cdot \frac{precision \cdot recall}{precision+recall}$

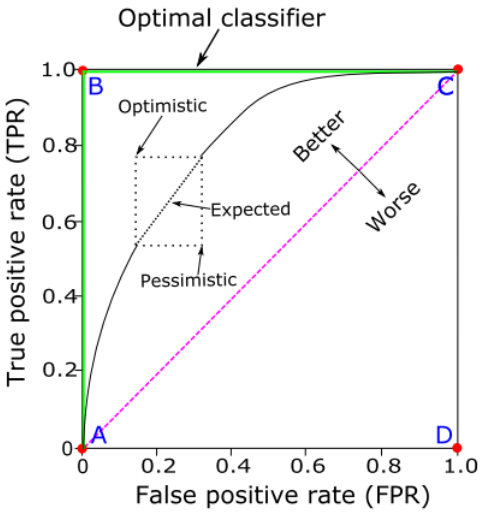
$F_1 - score$ oznacza średnią harmoniczną TPR i PPV; jej zakres wynosi od 0 do 1; Wysokie wartości wskazują na wysoką jakość klasyfikacji algorytmu.

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

Matthews correlation coefficient (MCC) oznacza: +1 perfekcyjną predykcję, -1 kompletny brak zgodnych wartości przewidzianych z rzeczywistymi, 0 wynik nie lepszy niż wartości losowe. [4]

5.2 Krzywa ROC

Receiver operating characteristics jest dwu-wymiarowym wykresem, gdzie na osi y zareprezentowane są wartości TPR, a na osi x wartości FPR, gdzie $FPR = 1 - TNR = \frac{FP}{FP+TN}$. Pozwala oszacować równowagę pomiędzy benefitami (TPR) i kosztami (FPR). Wartości szybciej zbiegające do punktu (0,1) oznaczają wyższą jakość.
Drzewa decyzyjne zwracają jedną etykietę dla każdego testowanego przypadku, co tworzy jedną tabelę pomyłek, a w rezultacie daje tylko jeden punkt w przestrzeni ROC. Dlatego w celu ich porównania, konieczne jest ich umieszczenie na jednym wykresie. [4]



6 Zbiory danych

Do wykonania projektu wybraliśmy trzy zestawy danych, które zapewnią klasy: binarne, wieloklasowe jak i dziedziny atrybutów: dyskretne oraz ciągłe. Ważnym aspektem przy wyborze była liczba rekordów - staraliśmy się rozważać tylko zbiory danych zawierające powyżej 1000 wpisów.

6.1 Gender Classification Dataset

(url) - binarna klasa, atrybuty dyskretne i ciągłe, zbiór zblasansowany; rekordów: 5001
Atrybuty:

- $long\_hair$ - atrybut binarny; 1 - ma długie włosy, 0 - nie ma długich włosów
- $forehead\_width\_cm$ - atrybut ciągły [cm]
- $forehead\_height\_cm$ - atrybut ciągły [cm]
- $nose\_wide$ - atrybut binarny; 1 - ma szeroki nos , 0 - nie ma szerokiego nosa
- $nose\_long$ atrybut binarny; 1 - ma długi nos , 0 - nie ma długiego nosa
- $lips\_thin$ - atrybut binarny; 1 - ma wąskie usta , 0 - nie ma wąskich ust
- $distance\_nose\_to\_lip\_long$ - atrybut binarny; 1 - ma duży dystans pomiędzy nosem i ustami , 0 - nie ma dużego dystansu między nosem i ustami

Klasa:

- gender - binarna; Male (50%) or Female (50%)

6.2 Red Wine Quality

(url) - wieloklasowy, atrybuty ciągłe, zbiór niezbalansowany; rekordów: 1599.
Atrybuty:

- $fixedacidity$ - atrybut ciągły

- $volatileacidity$ - atrybut ciągły
- $citricacid$ - atrybut ciągły
- $residualsugar$ - atrybut ciągły
- $chlorides$ - atrybut ciągły
- $freesulfurdioxide$ - atrybut ciągły
- $totalsulfurdioxide$ - atrybut ciągły
- $density$ - atrybut ciągły
- pH - atrybut ciągły
- $sulphates$ - atrybut ciągły
- $alcohol$ - atrybut ciągły

Klasa:

- quality - klasa wielowartościowa; 3 (0,63%), 4 (3,31%), 5 (42,59%), 6 (39,9%), 7 (12,45%), 8 (1,13%)

6.3 Heart Disease Classification Datase

(url) - binarna klasa, atrybuty dyskretne i ciągłe, zbiór niezbalansowany, rekordów: 1319.
Atrybuty:

- age - atrybut ciągły
- $gender$ - atrybut binarny; 0 - kobieta, 1 - Mężczyzna
- $impulse$ - atrybut ciągły
- $pressurehight$ - atrybut ciągły
- $pressurelow$ atrybut ciągły
- $glucose$ - atrybut ciągły
- kcm - atrybut ciągły
- $troponin$ - atrybut ciągły

Klasa:

- class - binarna; positive (61%) lub negative (39%)

7 Wstęp techniczny

7.1 Zmiany względem wstępnej dokumentacji

Po uwzględnieniu uwag prowadzącego po pierwszym etapie, zmieniliśmy system wyboru klasy w przypadku remisu w głosowaniu większościowym w węźle drzewa. Zamiast wybierać klasę losowo, wybieramy wartość węzła nadrzędnego (rodzica).

7.2 Struktura folderów

Projekt ma dwa główne katalogi: Data - zawiera pliki .csv ze zbiorami danych oraz src. Katalog src podzielony jest na następujące foldery:

- `Comparison_Algorithm` - część odpowiedzialna za generowanie wyników algorytmem porównawczym, w tym celu należy uruchomić plik `Comparison_Algorithm_runner.py` - wszelka konfiguracja odbywa się z wykorzystaniem pliku `conf.ini` - o tym w sekcji 7.3. Wyniki przechowywane są w katalogu `Logs`, dla każdego zbioru danych zostaje utworzony osobny podfolder (z odpowiednią nazwą).

- `Our_Algorithm` - analogicznie część odpowiedzialna za nasz algorytm. Uruchamiany jest z pliku `Our_Algorithm_runner.py` (konfiguracja ponownie z pliku `conf.ini`. Wyniki przechowywane są w ten sam sposób. Dodatkowymi elementami są pliki z naszym kodem, tzn.:
  - W katalogu `Forest`: plik `Forest.py` odpowiedzialny za generowanie lasu losowego.
  - W katalogu `Tree`: `Tree_node.py` zawiera definicję pojedynczego węzła w drzewie, a `Tree.py` odpowiada za budowanie drzewa.
- `Quality_tool` - zawiera "nadmiarowy, niewymagany" plik `Quality_tool.py` przetwarzający wyliczone przez oba algorytmy dane. Klasa następnie tworzy dla każdego zbioru danych jeden dokument pdf zawierający wszystkie tabele, confusion matrix, a także wykresy porównawcze ROC oraz ACC. W folderze `Measures` przechowywane są wyniki procesowania (tabela oraz macierz pomyłek) dla każdej testowanej wartości każdego parametru (z podziałem na nasz algorytm i porównawczy). Finalny raport dla aktualnego zbioru danych ma nazwę `final_raport.pdf`.

Ostatnim elementem jest plik `Utility.py` zawierający utworzone przez nas metody pomocnicze (jak tworzenie folderów, ładowanie ustawień z pliku konfiguracyjnego itp.)

7.3 Konfiguracja

Cała konfiguracja uruchamianego algorytm odbywa się w pliku `conf.ini`. Pierwszą sekcją jest `Test_settings`, w której ustawiamy zakresy testowanych wartości dla każdego osobnego eksperymentu. Drugą sekcją jest `Base_settings`, zawiera wartości bazowe dla każdego testu (zmieniany jest tylko aktualnie testowany parametr). Ostatnia sekcja `System_settings` zawiera: nazwę pliku testowanego zbioru danych, listę typów atrybutów (0 - dyskretny, 1-ciągły), liczbę próbek (mnożone razy 5 przez Cross-validation o czym więcej w sekcji 8.2) oraz liczbę wykorzystanych logicznych procesorów (multithreading).

7.4 Instrukcja

Po wpisaniu odpowiednich ustawień w pliku `conf.ini` uruchamiamy odpowiedniego runner'a (odpowiedni plik .py). W celu procesowania wyników i wygenerowania dokumentów pdf (po wygenerowaniu wszystkich wyników) należy, uruchomić klasę `Quality_tool.py`. Domyślną lokalizacją uruchamiania plików jest folder bazowy projektu, np. dla Quality_tool.py wykonalibyśmy następującą komendę: `python src\Quality_tool\Quality_tool.py`

7.5 Załączone pliki z wynikami eksperymentów

Napisaliśmy dodatek generujący wyniki, tzn. finalny raport dla każdego zbioru danych, zawierający porównanie obu algorytmów dla każdego przeprowadzonego testu: wykresy dokładności i ROC, tabele z miarami jakości oraz macierze pomyłek. Raporty odpowiadające danemu zbiorowi danych zawierają go w nazwie. Postanowiliśmy nie wstawiać wszystkich elementów, a odwoływać się do rozdziału i strony odpowiedniego dokumentu, żeby zachować czytelność sprawozdania.
Załączone dodatkowo pliki:

Plik raport_klasyfikacja_plci.pdf

Zawiera wszystkie wyniki dla zbioru danych: Klasyfikacja płci.

Plik raport_klasyfikacja_choroby_serca.pdf

Zawiera wszystkie wyniki dla zbioru danych: Klasyfikacja choroby serca

Plik raport_ocenianie_jakosci_wina.pdf

Zawiera wszystkie wyniki dla zbioru danych: Określanie jakości wina.

8 Założenia wstępne

8.1 Metodyka podziału danych

Z powodu braku jasno wyznaczonych danych testowych oraz treningowych, wprowadziliśmy mechanizm Cross-validation. Zbiór danych za każdym razem dzielony jest na pięć części, z czego każda z nich zachowuje proporcję

klas bazowego pliku. Następnie dane te grupowane są w dane: treningowe (80% - cztery części oryginalnego zbioru) i testowe (20% - jedna część oryginalnego zbioru). Za każdym razem $\frac{1}{5}$ zbioru - dane testowe, są zmieniane (każda grupa po kolei, w przypadku naszych testów - łącznie do trzech razy). Rezultatem jest średnia arytmetyczna uzyskanych wyników. Dzięki temu algorytm uczy się na odpowiednio dużej próbce zawierającej odpowiednią charakterystykę, a następnie zostaje sprawdzony na różnych zbiorach testowych zawierających te samą specyfikę klas.

8.2 Metodyka zbierania danych

Efektem każdego z pojedynczych testów jest wpis w pliku o nazwie odpowiadającej aktualnie testowanej wartości. Wpis ten składa się z informacji o wszystkich parametrach lasu, dla których przeprowadzony został eksperyment oraz ze sprowadzonej do postaci listy list macierzy pomyłek, zawierającej informację ile razy las sklasyfikował daną klasę w określony sposób np. wartość `Male:Female` oznacza liczbę sklasyfikowanych przez las wartości jako klasa `"Female"`, których rzeczywistą klasą jest `"Male"`.

8.3 Metodyka testowania

Testowanie naszego algorytmu możemy podzielić na 3 podstawowe etapy:
Etap weryfikacji poprawności działania algorytmu: Etap ten polegał na sprawdzeniu poprawności działania algorytmu. Testy wykonywane były na małych próbkach danych, bez wyliczania średnich ani bez dużego zakresu danych testowych. Głównym celem weryfikacji była ocena poprawności działania kodu pod względem błędów.
Etap estymacji wartości bazowych oraz zakresu wartości testowych:
Etap ten polegał na pełnym uruchomieniu całego algorytmu dla każdego z testowanych zbiorów danych. Dzięki niemu ustaliliśmy dane bazowe oraz zakresy testów. Dane bazowe są stałe dla każdego wykonywanego eksperymentu i zmienia się tylko wartość testowanego parametru. Zakres wartości testowych to właściwa cześć testu polegająca na badaniu zmian w wynikach algorytmu przy zmianie tylko jednego z parametrów. Po uzyskaniu plików z raportami z testów oraz przeanalizowaniu wykresów doszliśmy do następujących wniosków:

- należy zwiększyć liczbę próbek, ponieważ wyniki dla wartości niemających wpływu na algorytm porównawczy (nie są przez niego obsługiwane) potrafiły wahać się o kilka punktów procentowych,
- ustawiliśmy opisane poniżej wartości bazowe 8.4 na najniższe wartości przy których oba algorytmy otrzymywały podobne wartości oraz oba zaczynały się stabilizować,
- ze względu na czasochłonność procesu, zmniejszyliśmy ilość wykonywanych testów dla niektórych wartości w momencie w którym zauważyliśmy, że algorytmy się ustabilizowały.

Wyniki tego etapu w postaci raportów dla każdego ze zbiorów danych zostaną dołączone do tego sprawozdania.
Etap przeprowadzenia faktycznych testów
Etap, którego wynikiem stały się raporty dla każdego z badanych zbiorów danych na, których podstawie zostały wyciągnięte wnioski oraz obserwacje w dalszych częściach tego raportu.

8.4 Wybrane dane bazowe

- Maksymalna głębokość drzewa (liczba poziomów bez liczenia korzenia) = 10
- Minimalna liczba elementów w podzbiorze danych wymagana do przeprowadzenia procesu tworzenia nowego węzła decyzyjnego = 10
- Wartość procentowa warunków do odrzucenia przed ruletką = 0.5
- Procent wykorzystanych atrybutów = 0.5
- Liczba drzew w lesie losowym = 50
- Tryb traktowania parametrów ciągłych = 3

Warto wyjaśnić znaczenia parametru `mode`. Mianowicie oznacza sposób traktowania wartości ciągłych, gdzie:

- 0 - wartość średnia dla dwóch sąsiednich wartości
- 1 - wartość średnia dla wszystkich wartości
- 2 - mediana wszystkich wartości
- 3 - podział danych na kubły o równym rozmiarze (funkcja `qcut`)

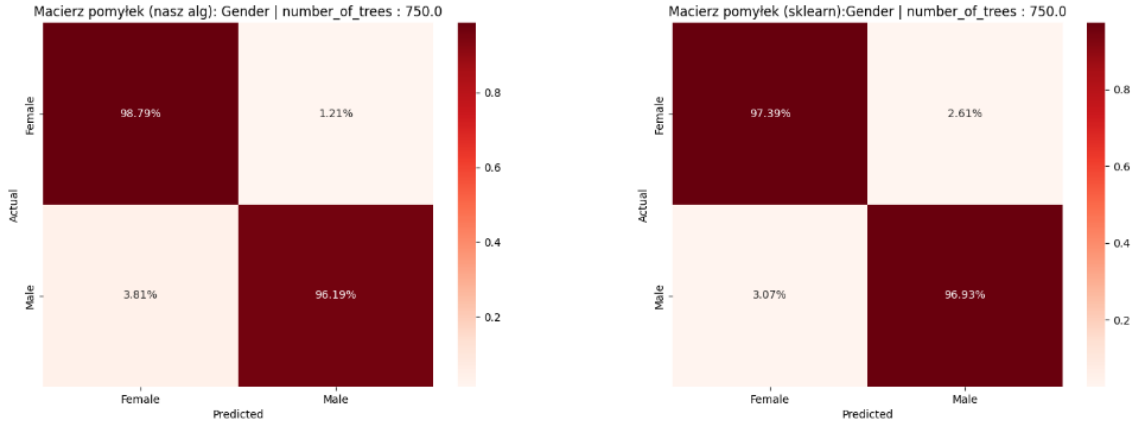
8.5 Wybrane zakresy danych testowych

Algorytm wykona proces klasyfikacji danych po kolei dla każdego parametru z listy zmieniając jego wartość na podaną (zachowując dla innych wartości bazowe).

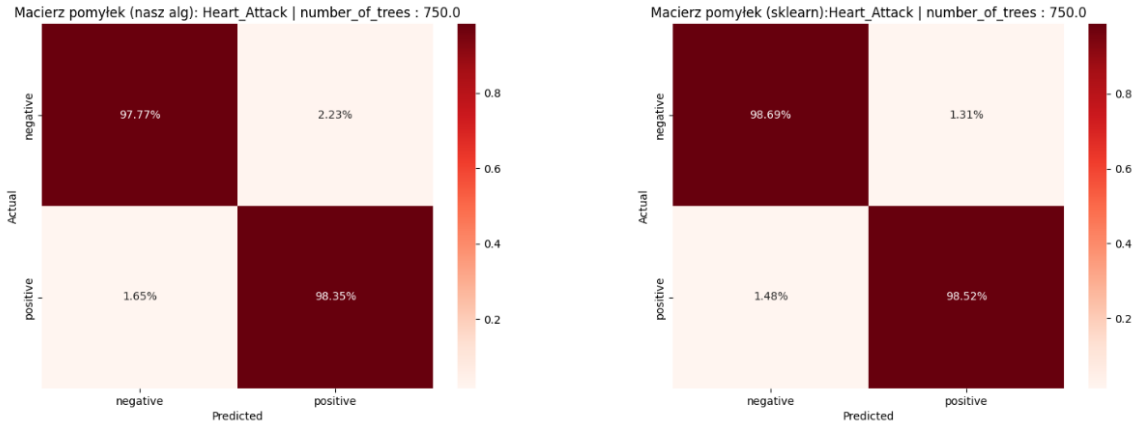
```
Maksymalna głębokość drzewa = 1,2,5,10,15,25,50,100
Minimalna liczba elementów w podzbiorze danych = 5,10,15,25,50,75,125,250,500
Wartość progu odrzucenia najsłabszych warunków = 0,0.25,0.5,0.75,1.0
Procent wykorzystanych atrybutów = 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0
Liczba drzew w lesie =  1,3,5,10,15,25,50,100,200,500,750
Tryb traktowania parametrów ciągłych = 0,1,2,3
```

9 Eksperymenty - zmiany parametrów

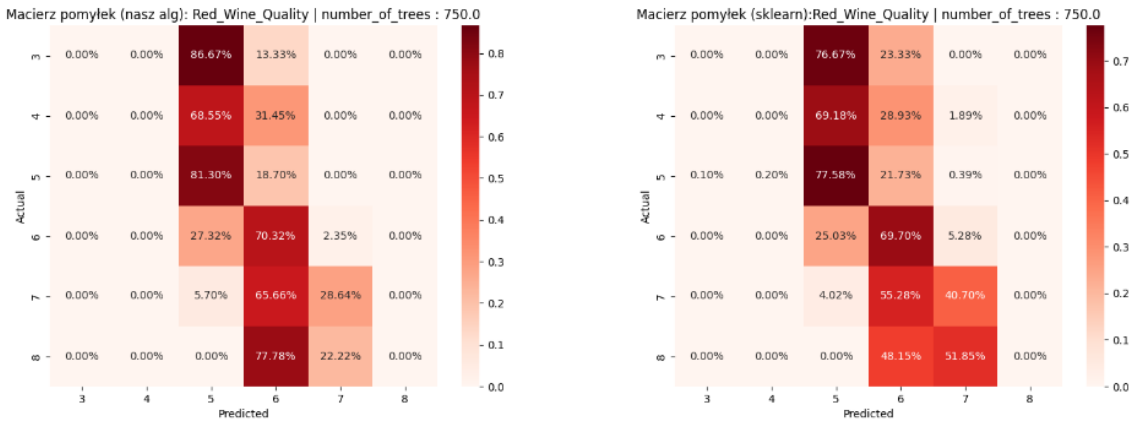
Na początku chcielibyśmy wstawić wybrane heatmap'y (po jednej dla każdego zbioru danych) w celu zaprezentowania, że napisany algorytm osiąga podobne wyniki do algorytmu porównawczego (sklearn).



Rys. 1: Macierz pomyłek dla zbioru danych: Klasyfikacja płci



Rys. 2: Macierz pomyłek dla zbioru danych: Klasyfikacja choroby serca



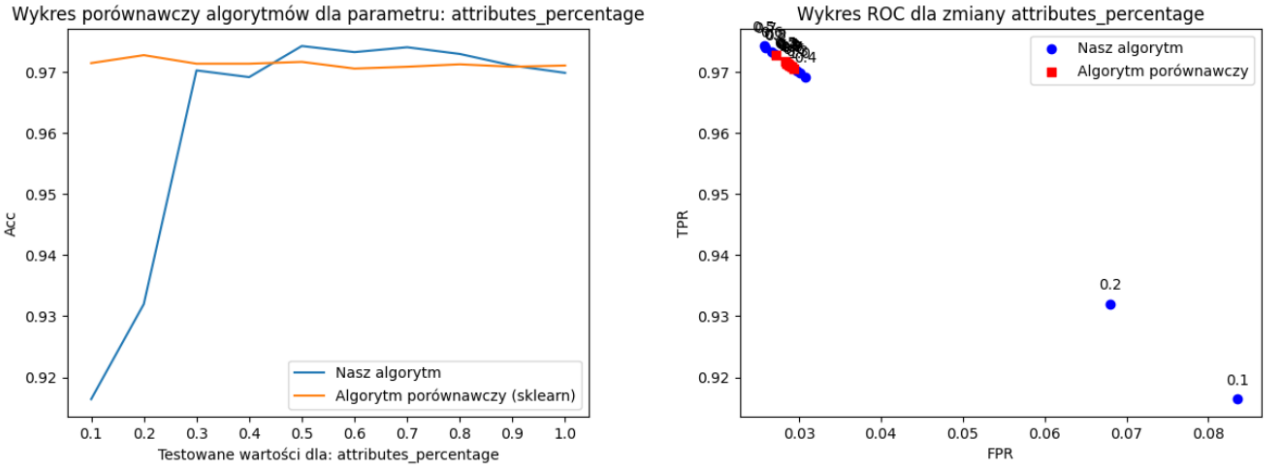
Rys. 3: Macierz pomyłek dla zbioru danych: Określanie jakości wina

Wyniki potwierdzają wstępną weryfikację - napisany algorytm osiąga zbliżone wyniki do algorytmu porównawczego, w szczególności w przypadkach 1 i 2.

9.1 Procent wybranych atrybutów

W pierwszym eksperymencie sprawdziliśmy wpływ liczby atrybutów dostępnych dla danego modelu podczas budowy drzewa na dokładność algorytmu. Testy zostały przeprowadzone zgodnie z założeniami i parametrami opisanymi w 8.4, 8.

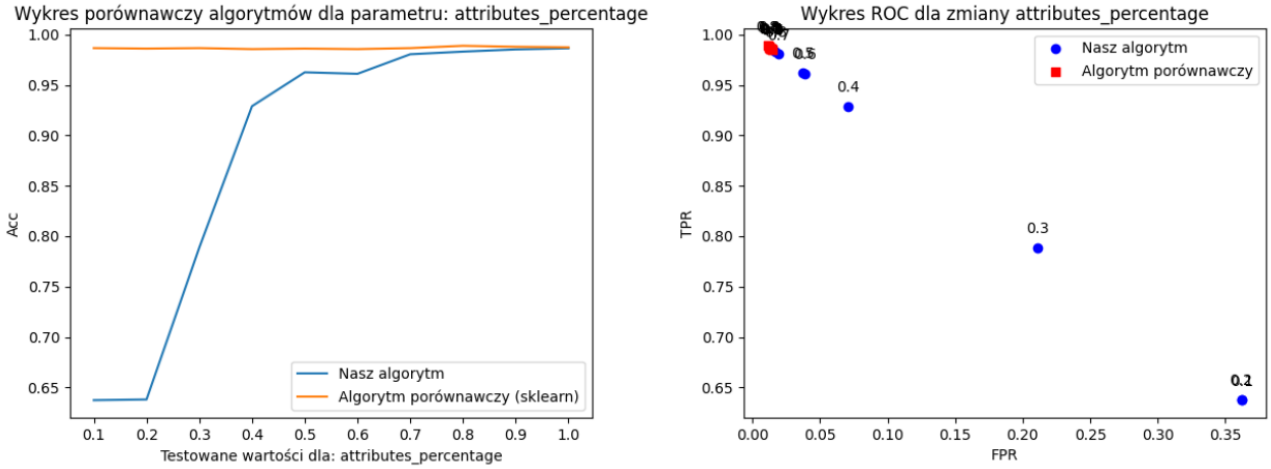
9.1.1 Zbiór danych: Klasyfikacja płci



Rys. 4

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_plci.pdf w rozdziale 1 (od strony 1).

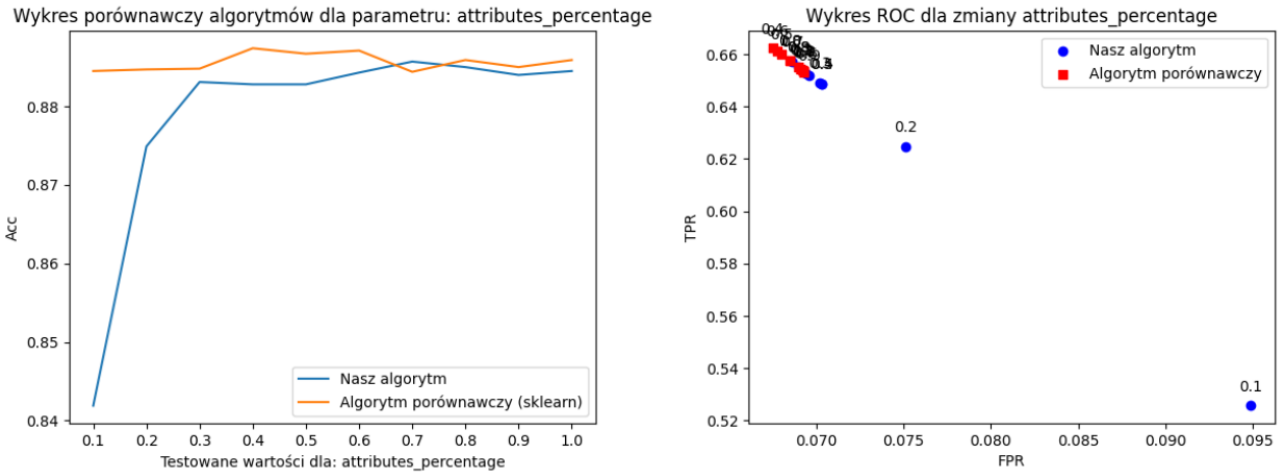
9.1.2 Zbiór danych: Klasyfikacja choroby serca



Rys. 5

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_choroby_serca.pdf w rozdziale 1 (od strony 1).

9.1.3 Zbiór danych: Określanie jakości wina



Rys. 6

Wszystkie wyniki dostępne są w pliku Plik raport_ocenianie_jakosci_wina.pdf w rozdziale 1 (od strony 1).

9.1.4 Wnioski z eksperymentu

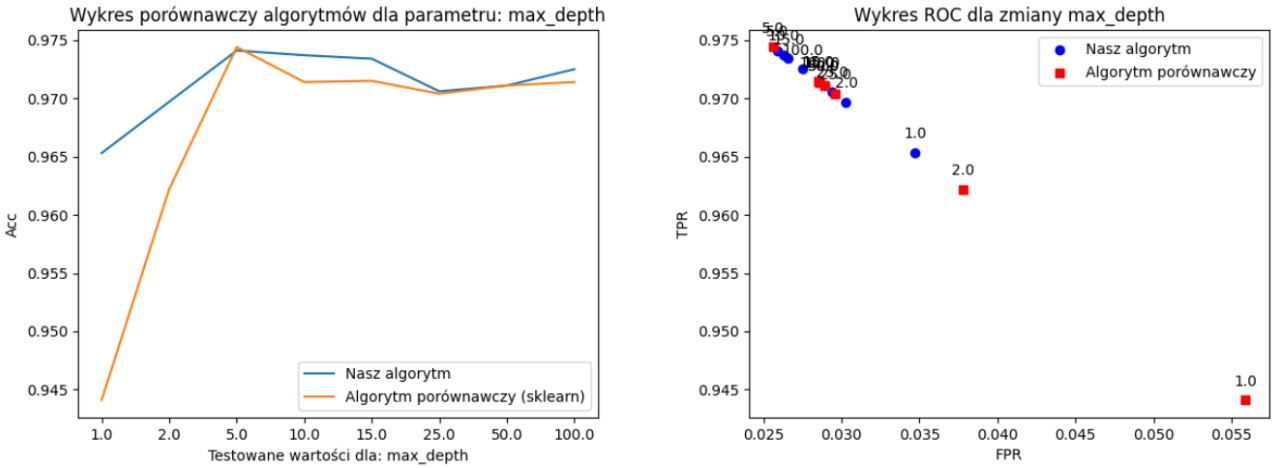
Dla każdego zbioru zachodzi ogólna zależność, tzn. im wyższy procent/wyższa liczba użytych atrybutów do budowy drzewa, tym dokładniejsze wyniki. Najszybszy przyrost dla naszego algorytmu możemy zaobserwować dla początkowych wartości testowanego parametru. Po osiągnięciu pewnego poziomu wzrost spowalnia aż do ustabilizowania tendencji. Należy zaznaczyć, że jedynie dla zbioru klasyfikacji płci obserwujemy dla pewnych wartości (od 0.7) efekt nadmiernego dopasowania. Może to wynikać z faktu, że zbiór jest prosty, tzn. większość atrybutów jest typu dyskretnego i klasy są binarne. Dla dwóch pozostałych zbiorów danych efekt nie zachodzi, ponieważ mają atrybuty typu ciągłego (dodatkowo zbiór win jest wieloklasowy). W rezultacie ciężej dopasować wyniki co skutkuje obserwacją stabilizacji dokładności.
Dla algorytmu porównawczego obserwowana dokładność jest praktycznie stała, niezależnie od ustawionej wartości parametru. Powyższe obserwacje potwierdza krzywa ROC, dla algorytmu porównawczego wszystkie wartości znajdują się blisko punktu (1,0). Dla naszego algorytmu dopiero dla wartości 0,2 w Zbiór danych: Klasyfikacja płci i w Zbiór danych: Określanie jakości wina, a dla 0,3 w Zbiór danych: Klasyfikacja choroby serca punkty zbiegają do punktu (1,0).

Większa liczba atrybutów zwiększa przestrzeń danych, dla której algorytm się uczy, dzięki temu wyniki są dokładniejsze. Stałość dokładności algorytmu porównawczego może wynikać z zastosowania metody .forest zamiast .tree - o tym napiszemy dokładniej w podsumowaniu eksperymentów. Jednakże patrząc na wyniki testów podejrzewamy, że Sklearn używa tylko wszystkich atrybutów (wartość 1.0), dlatego możemy stwierdzić że algorytmy nasz i porównawczy działają z podobną dokładnością.

9.2 Maksymalna głębokość drzewa

W tym eksperymencie został zbadany wpływ parametru głębokości drzewa (korzeń ma indeks równy 0) na dokładność algorytmu. Testy zostały przeprowadzone zgodnie z założeniami i parametrami opisanymi w 8.4, 8.

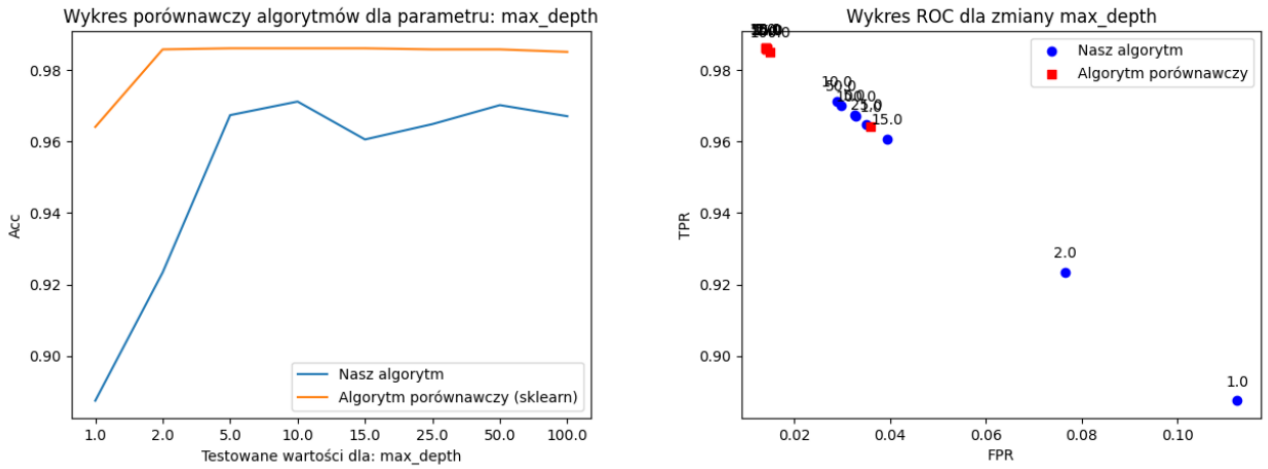
9.2.1 Zbiór danych: Klasyfikacja płci



Rys. 7

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_plci.pdf w rozdziale 2 (od strony 12).

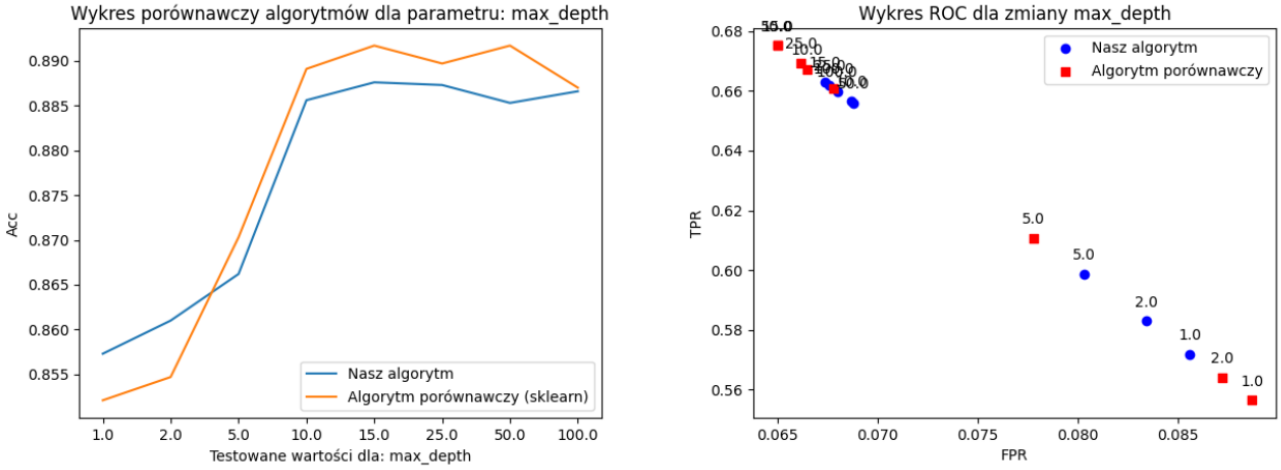
9.2.2 Zbiór danych: Klasyfikacja choroby serca



Rys. 8

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_choroby_serca.pdf w rozdziale 2 (od strony 12).

9.2.3 Zbiór danych: Określanie jakości wina



Rys. 9

Wszystkie wyniki dostępne są w pliku Plik raport_ocenianie_jakosci_wina.pdf w rozdziale 2 (od strony 12).

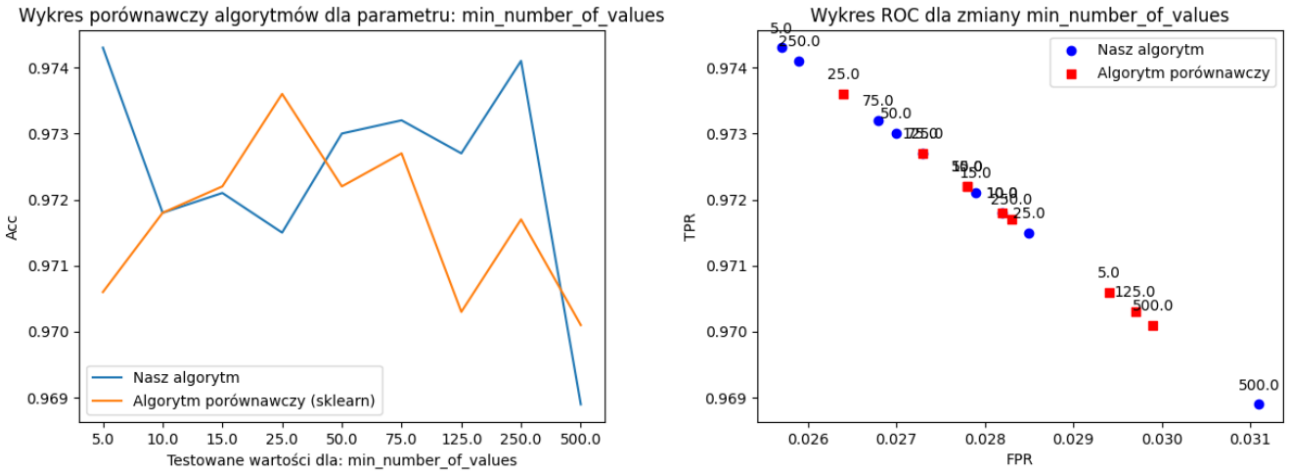
9.2.4 Wnioski z eksperymentu

W przypadku tego testu wniosek jest wspólny dla wszystkich zbiorów danych - wraz ze wzrostem głębokości budowanego drzewa rośnie dokładność algorytmu. Wynika to z faktu, że większa głębokość oznacza więcej węzłów decyzyjnych (warunków), a to prowadzi do dokładniejszej selekcji kolejnych podzbiorów danych. Krzywe ROC to potwierdzają, im większa głębokość, tym wyniki są bliżej punktu (1,0). W przypadku Zbiór danych: Klasyfikacja płci oba algorytmy na początku mają tendencję rosnącą, jednak nasz ma większą dokładność co może wynikać z bardziej precyzyjnego rozpatrywania możliwości dokonania podziału względem algorytmu porównawczego. Wraz ze wzrostem parametru oba algorytmy zachowują się wręcz jednakowo, zarówno pod względem stabilności jak i skuteczności. W przypadku zbioru Zbiór danych: Klasyfikacja choroby serca nasz algorytm okazuje się być gorszy od porównawczego (bazując na wynikach uzyskanych w trakcie drugiego etapu testów opisanego w sekcji drugiej rozdziału 8.3, wynika to z obniżenia ilości drzew w lesie do wartości faworyzującej algorytm porównawczy, ale za to wydajniejszej czasowo). Dodatkowo algorytm traci na stabilności co, według naszych przewidywań, jest spowodowane implementacją ruletki. W przypadku zbioru Zbiór danych: Określanie jakości wina algorytmy radzą sobie bardzo porównywalnie, jednak w przypadku algorytmu Sklearn od wartości głębokości równej 50 możemy zaobserwować coś, co może przypominać przetrenowanie. W naszym algorytmie efekt ten nie występuje. Przypuszczalnie ruletka, która w przypadku zbioru o klasyfikacji binarnej Zbiór danych: Klasyfikacja choroby serca spowodowała niestabilność rozwiązania, w tym przypadku uchroniła nasz algorytm przed przetrenowaniem do jednej z wielu klas zbioru wieloklasowego. Różnice między zbiorem Zbiór danych: Klasyfikacja płci a Zbiór danych: Klasyfikacja choroby serca i Zbiór danych: Określanie jakości wina, polegające na przewadze algorytmu porównawczego względem naszego rozwiązania, prawdopodobnie wynikają ze skomplikowania zbiorów danych, a dokładniej ilości atrybutów dyskretnych w zbiorze.

9.3 Minimalna liczba elementów wymagana do wykonania podziału

W tym teście został zbadany wpływ parametru minimalnej liczby elementów (min_number_of_values) w uzyskanym podzbiorze danych, wymaganej do utworzenia nowego węzła decyzyjnego. Testy zostały wykonane z założeniami i wartościami opisanymi w 8.4, 8.

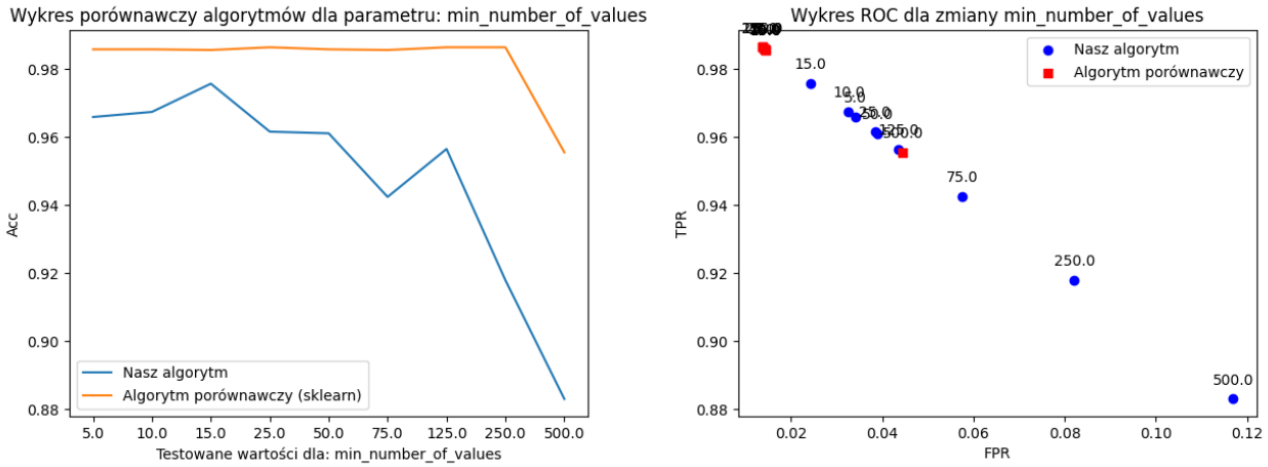
9.3.1 Zbiór danych: Klasyfikacja płci



Rys. 10

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_plci.pdf w rozdziale 3 (od strony 21).

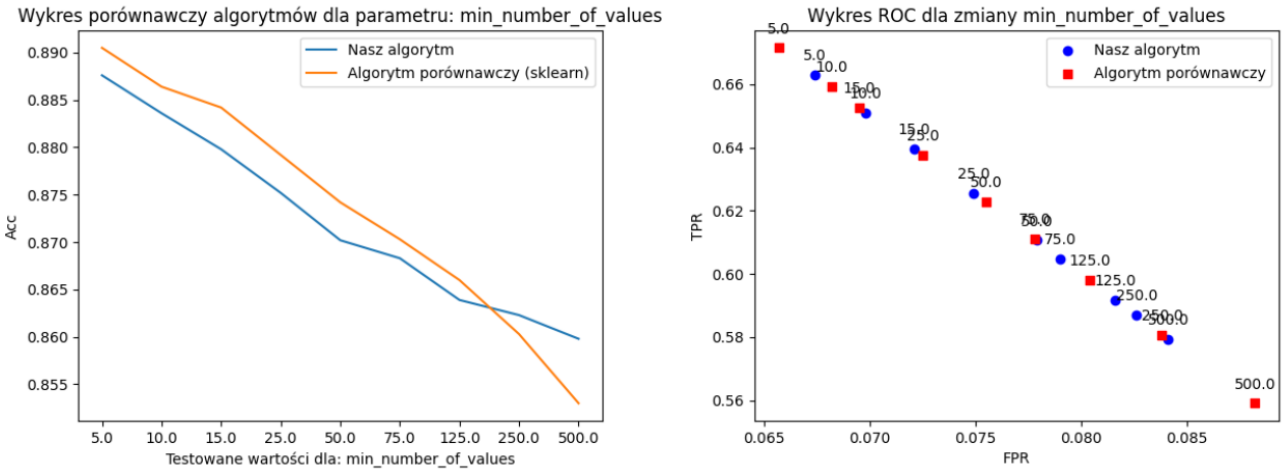
9.3.2 Zbiór danych: Klasyfikacja choroby serca



Rys. 11

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_choroby_serca.pdf w rozdziale 3 (od strony 21).

9.3.3 Zbiór danych: Określanie jakości wina



Rys. 12

Wszystkie wyniki dostępne są w pliku Plik raport_ocenianie_jakosci_wina.pdf w rozdziale 3 (od strony 21).

9.3.4 Wnioski z eksperymentu

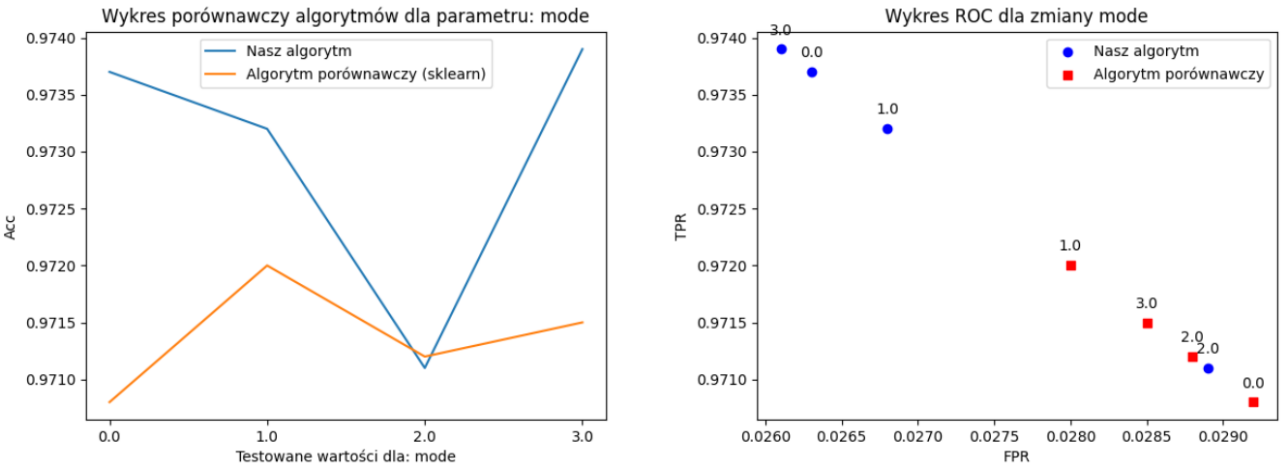
Pierwsza, rzucająca się w oczy obserwacja to ogólna tendencja - wraz ze wzrostem testowanego parametru spada dokładność algorytmu.

W przypadku Zbiór danych: Klasyfikacja płci obserwujemy liczne wahania występujące z powodu prostszego i większego, względem dwóch pozostałych, zbioru danych (więcej rekordów), co oznacza więcej możliwych podziałów. Po osiągnięciu wartości 250 następuje znaczący spadek dokładności spowodowany dużym ograniczeniem liczby możliwych podziałów przez testowany parametr, czyli nadmierne dopasowanie.
W przypadku Zbiór danych: Klasyfikacja choroby serca wyżej opisane zachowanie obserwujemy dopiero po osiągnięciu określonej wartości, dla naszego algorytmu wynosi ona 125). Oba algorytmy zachowują się podobnie, jednak to Sklearn zachowuje nieco wyższą dokładność, stabilność oraz ma opóźniony punkt załamania trendu (spadek dokładności). W tym przypadku opóźnione pojawienie się nadmiernego dopasowania w przypadku algorytmu Sklearn wynika z połączenia użycia w naszym algorytmie ruletki, wycinania najgorszych wyników oraz sposobu traktowania wartości ciągłych.
Inaczej sytuacja wygląda w przypadku wykresu Zbiór danych: Określanie jakości wina, dokładność cały czas dla obu algorytmów maleje. Patrząc, na otrzymane tablice pomyłek możemy wywnioskować, że nasz algorytm dopasowuje się do najczęściej występującej klasy, co sprawia, że przy zmniejszonej ilości podziałów zachowuje on względną skuteczność.
Wykresy ROC (w szczególności Zbiór danych: Określanie jakości wina) potwierdzają obserwacje - większe wartości oddalają się od punktu (1,0), gdzie dla każdego zbioru danych maksymalna wartość parametru jest najdalej.
Konkludując, im większa wartość parametru min_number_of_values, tym mniejsza liczba podziałów zachodzących przy budowie drzewa, co implikuje mniejszą dokładność algorytmu.

9.4 Tryby pracy algorytmu

W tym eksperymencie został zbadany wpływ metody obsługi atrybutów ciągłych na dokładność wyników. Zastosowane wartości są zgodne z opisem z 8.4, 8.

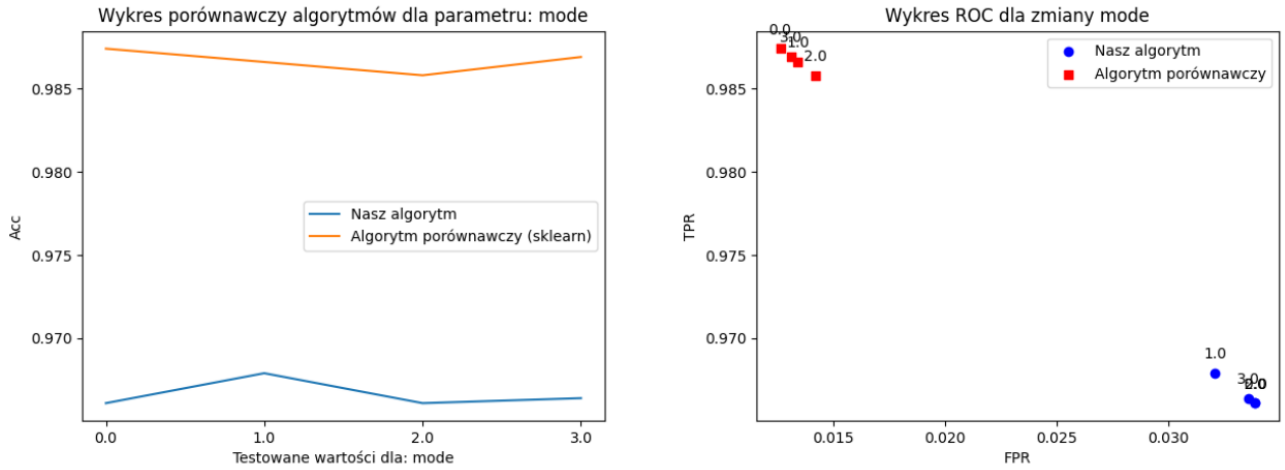
9.4.1 Zbiór danych: Klasyfikacja płci



Rys. 13

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_plci.pdf w rozdziale 4 (od strony 31).

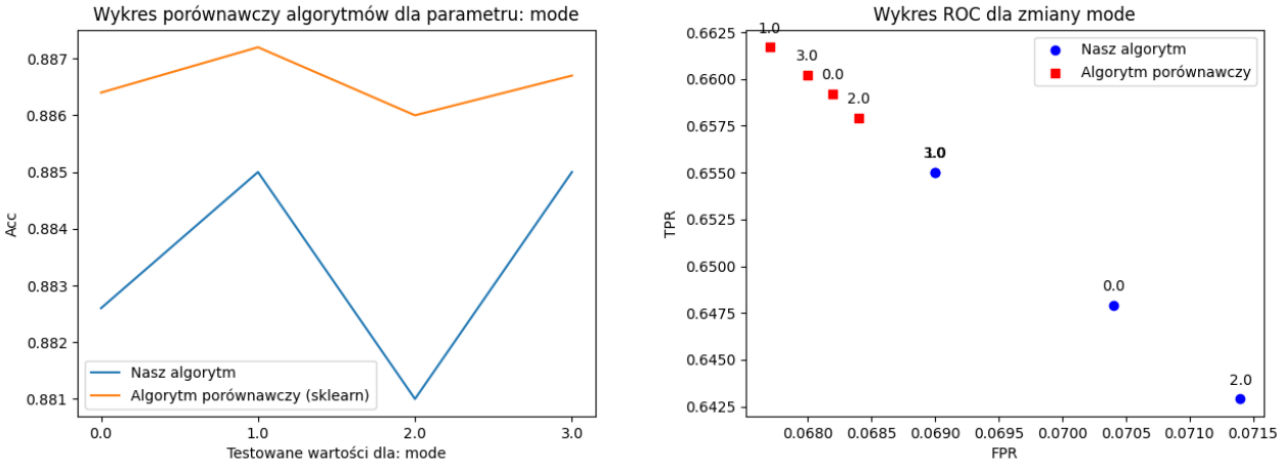
9.4.2 Zbiór danych: Klasyfikacja choroby serca



Rys. 14

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_choroby_serca.pdf w rozdziale 4 (od strony 31).

9.4.3 Zbiór danych: Określanie jakości wina



Rys. 15

Wszystkie wyniki dostępne są w pliku Plik raport_ocenianie_jakosci_wina.pdf w rozdziale 4 (od strony 31).

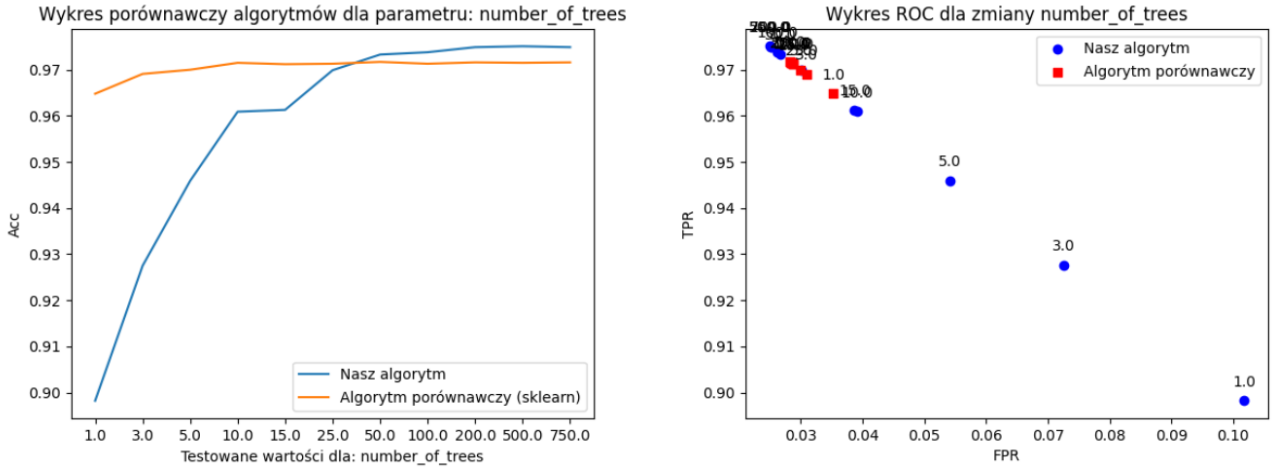
9.4.4 Wnioski z eksperymentu

Dla Zbiór danych: Klasyfikacja płci, możemy zauważyć, że teoretycznie najlepszą celnością cechuje się tryb dzielący dane na kubełki jednakże, zwracając uwagę na skalę, możemy zauważyć, że różnice są zbyt małe, aby wyciągnąć jednoznaczne wnioski (co potwierdza wykres ROC). Podobnie dla Zbiór danych: Klasyfikacja choroby serca, różnice między pojedynczymi trybami dla danego algorytmu są zbyt małe, żeby przeprowadzić analizę (co potwierdza wykres ROC), jednakże tak jak w poprzednich przypadkach, możemy zauważyć, że dla średniej ze wszystkich wartości dokładność jest minimalnie lepsza w przypadku algorytmu porównawczego. Dopiero wyniki otrzymane dla Zbiór danych: Określanie jakości wina dają nam możliwość wyciągnięcia wniosków. Ewidentnie można stwierdzić, że najsłabsze rezultaty daje mediana wszystkich wartości tryb = 2.0 (co widać na wykresie dokładności i ROC). Dobre wyniki osiąga podział danych na kubły (opcja 3) oraz wartość średnia dla wszystkich wartości. Różnica w tendencji zachodzi jedynie w przypadku wartości średniej dla dwóch sąsiednich wartości. Może wynikać to z faktu, że w zbiorze danych jakości wina jest więcej atrybutów ciągłych, co w rezultacie może prowadzić do nadmiernego dopasowania. Wiąże się to, z większym obciążeniem obliczeniowym co przekłada się na zwiększoną długość obliczeń. Ponadto porównywanie w tym przypadku zachowania algorytmów, nie ma większego sensu, ponieważ algorytm Sklearn nie obsługuje trybów innych niż trzeci, dlatego został on wybrany jako bazowy. Różnice w wynikach to czysty błąd statystyczny. Możemy się przez to pokusić o stwierdzenie, że przy odpowiedniej ilości próbek wybór ten nie ma wpływu na skuteczność rozwiązania.

9.5 Liczba drzew w lesie losowym

W tym teście zbadaliśmy wpływ liczby modeli drzew w lesie na jakość generowanych wyników zgodnie z 8.4, 8.

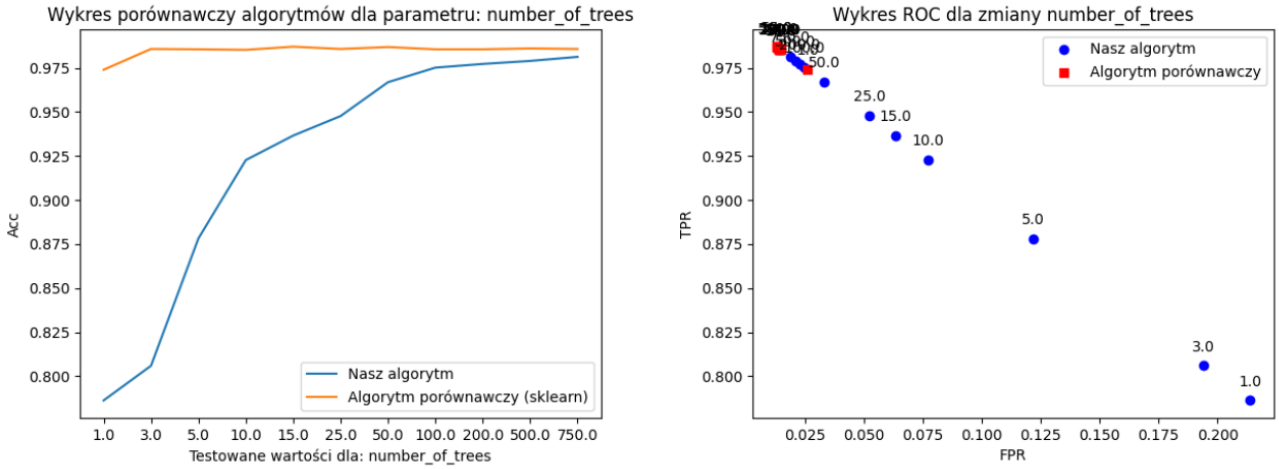
9.5.1 Zbiór danych: Klasyfikacja płci



Rys. 16

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_plci.pdf w rozdziale 5 (od strony 36).

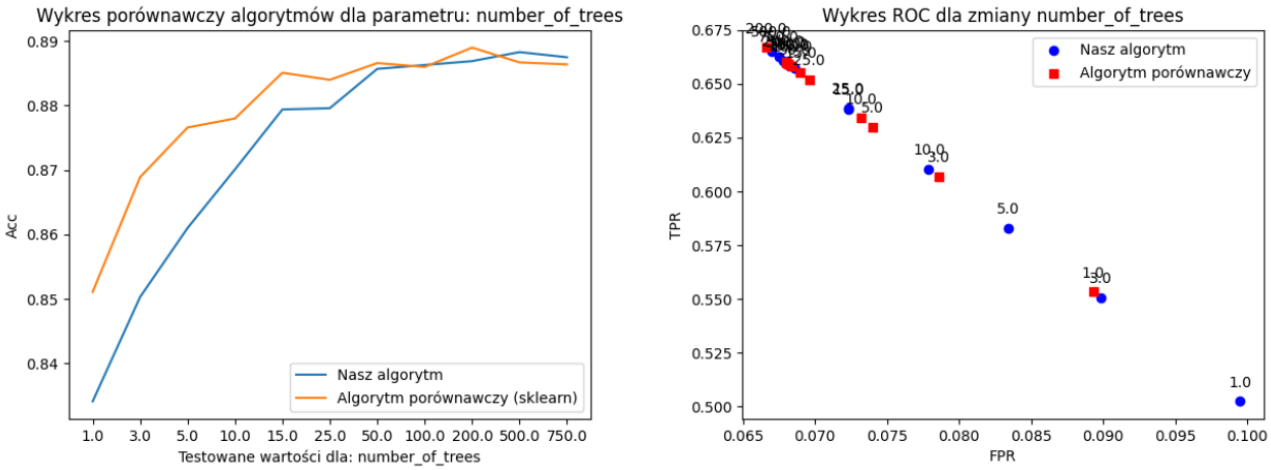
9.5.2 Zbiór danych: Klasyfikacja choroby serca



Rys. 17

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_choroby_serca.pdf w rozdziale 5 (od strony 36).

9.5.3 Zbiór danych: Określanie jakości wina



Rys. 18

Wszystkie wyniki dostępne są w pliku Plik raport_ocenianie_jakosci_wina.pdf w rozdziale 5 (od strony 36).

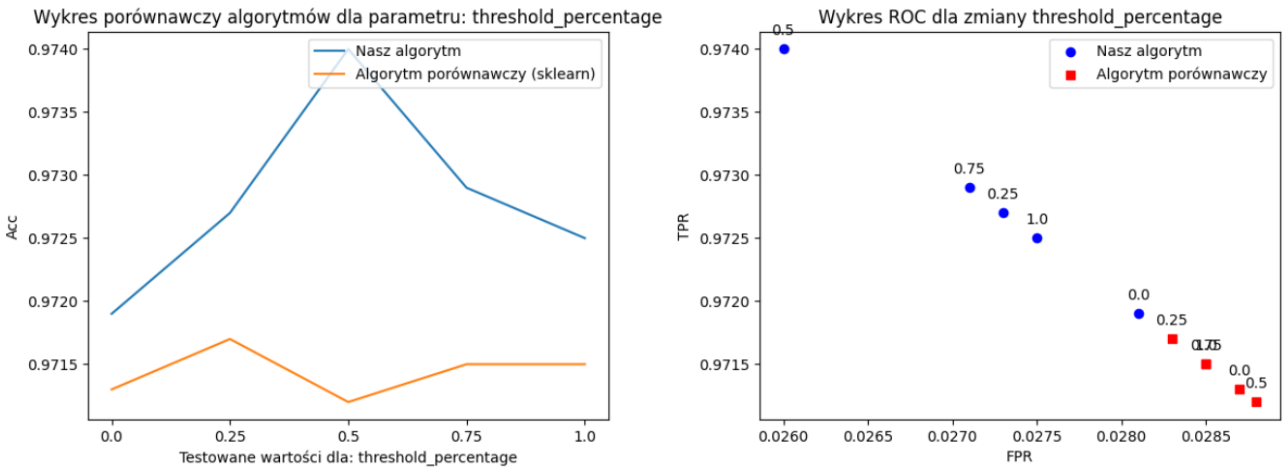
9.5.4 Wnioski z eksperymentu

Z powyższych wykresów na pierwszy rzut oka rysuje się oczywista zależność dla naszego algorytmu - im więcej drzew w lesie losowym tym dokładniejszy jest wynik. Największe przyrosty są dla początkowych wartości testu, dopiero przy liczbie drzew wynoszącej 50 (lub ok. 100 w przypadku Zbiór danych: Określanie jakości wina) dokładność się stabilizuje.
Analizując wykres ROC widzimy, że w przypadku naszego algorytmu dopiero od 5 drzew w Zbiór danych: Klasyfikacja płci, a dla Zbiór danych: Klasyfikacja choroby serca i Zbiór danych: Określanie jakości wina otrzymujemy wyniki zbiegające do punktu (1,0).
Algorytm porównawczy natomiast, dla zbiorów danych z klasami binarnymi nawet dla najmniejszych ilości drzew osiąga dokładność na praktycznie równym poziomie. Zachowanie to zmienia się w przypadku wieloklasowego zbioru, gdzie Sklearn prezentuje tendencję podobną do naszego algorytmu. Potwierdza to wykres ROC, gdzie dla Zbiór danych: Klasyfikacja płci i Zbiór danych: Klasyfikacja choroby serca wartości od początku są w okolicy punktu (1,0), a w Zbiór danych: Określanie jakości wina do tego punktu zbiegają dopiero od 3 drzew.
Dla niskich wartości, ilości drzew w lesie, algorytm Sklearn okazuje się być bardziej skuteczny, jednakże wraz ze wzrostem ilości drzew oba algorytmy coraz bardziej się zrównują. Może wynikać to z faktu zastosowania w naszym algorytmie parametrów threshold, ruletki oraz faktu, że w przeciwieństwie do naszego algorytmu parametr ilości atrybutów nie działa dla algorytmu porównawczego. Zjawisko nadmiernego dopasowania w tym przypadku nie zachodzi, ponieważ im więcej drzew tym więcej możliwości różnych wyborów klas co ma za zadanie wyeliminowanie tego efektu.

9.6 Warunek brzegowy ruletki

W tym eksperymencie zmieniamy wartość parametru threshold (zgodnie z opisem w 8.4, 8), czyli procentową wartość warunków do odrzucenia przed ruletką (dla wartości 1,0 = 100% zostaje tylko najlepszy wynik).

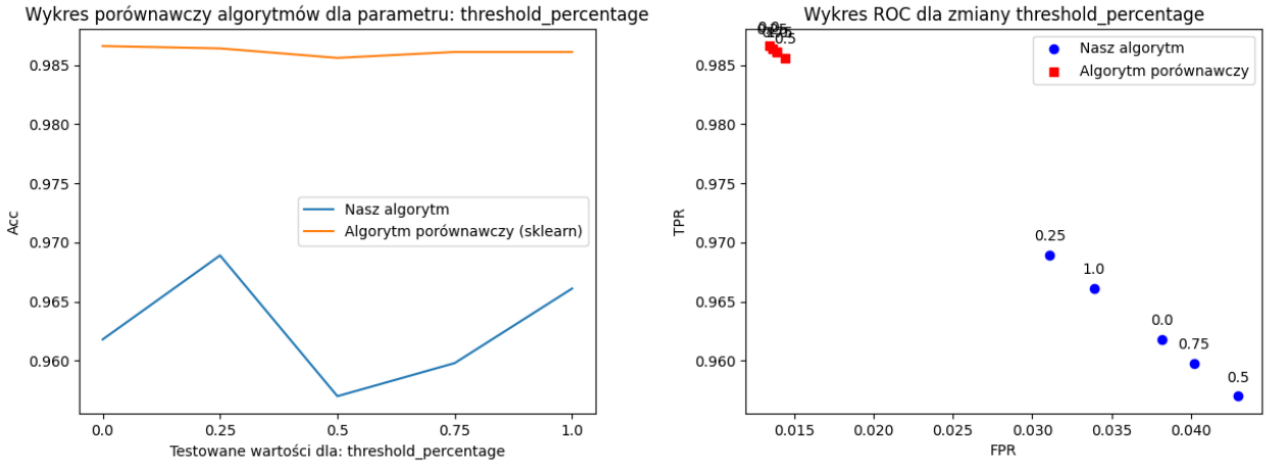
9.6.1 Zbiór danych: Klasyfikacja płci



Rys. 19

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_plci.pdf w rozdziale 6 (od strony 48).

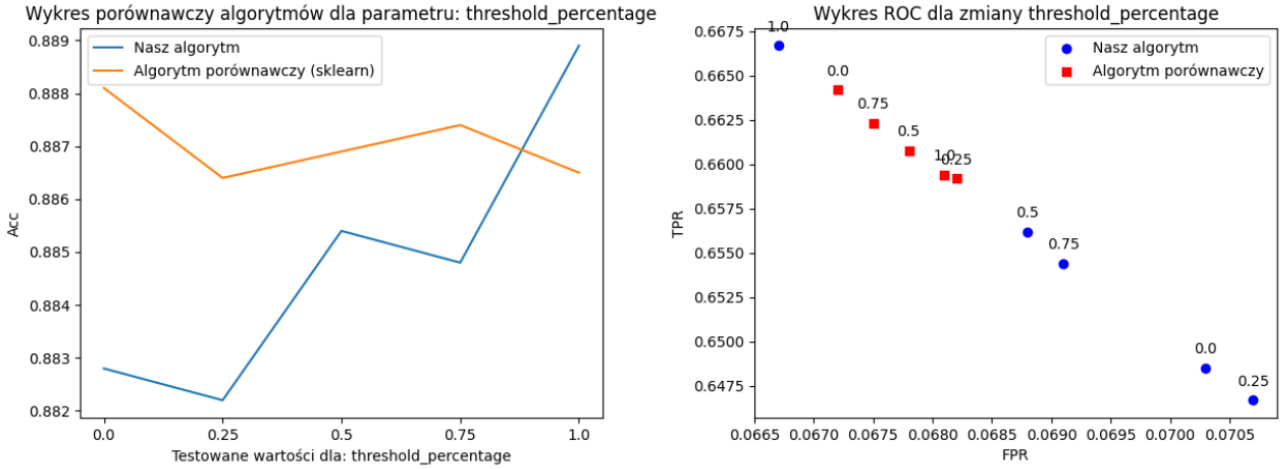
9.6.2 Zbiór danych: Klasyfikacja choroby serca



Rys. 20

Wszystkie wyniki dostępne są w pliku Plik raport_klasyfikacja_choroby_serca.pdf w rozdziale 6 (od strony 48).

9.6.3 Zbiór danych: Określanie jakości wina



Rys. 21

Wszystkie wyniki dostępne są w pliku Plik raport_ocenianie_jakosci_wina.pdf w rozdziale 6 (od strony 48).

9.6.4 Wnioski z eksperymentu

Przed analizą wyników warto zwrócić uwagę na skalę osi - z pozoru kompletnie różne wyniki w Zbiór danych: Klasyfikacja płci różnią się tysięcznymi. Dla pierwszego z wyników Zbiór danych: Klasyfikacja płci różnice są za małe, żeby wyciągnąć daleko idące wnioski. Dla zbioru Zbiór danych: Klasyfikacja choroby serca możemy zauważyć niestabilność rozwiązania. Wydaje się, że wraz ze wzrostem wartości progowej wyniki stają się coraz lepsze, zaburza to wynik dla wartości 25%, może to wynikać zarówno ze sposobu działania ruletki jak i z powodu nadmiernego dopasowania. Tej drugiej hipotezie przeczy fakt, że dla wyboru najlepszego podziału uzyskaliśmy podobny, najwyższy wynik. Dopiero dla zbioru danych określanie jakości wina obserwujemy znaczny wzrost dokładności wraz ze zwiększającą się liczbą usuwanych warunków. Tendencja ta się utrzymuje aż do wartości threshold = 1, gdzie zostają usunięte wszystkie warunki poza najlepszym.
Wartość threshold nie wpływa na dokładność w kontekście pierwszego prostego zbioru danych z binarnymi klasami. W tym przypadku prawdopodobieństwa dobrych warunków po obliczeniu wartości gini są na tyle wysokie, że losowanie (ruletka) nie zmienia wyniku. Dopiero przy zbiorach z wartościami ciągłymi oraz zbiorami wieloklasowymi, gdzie jest więcej opcji do wyboru obserwowalna staje się korelacja dokładności i wartości granicznej. Warto zaznaczyć jednak, że w przypadku zbiorów z klasami binarnymi, algorytm jest bardziej podatny na wybór optymalniejszej opcji za pomocą ruletki, pomimo mniejszej wartości granicznej.

10 Podsumowanie eksperymentów

Powyżej opisane eksperymenty pokazały wpływ wartości poszczególnych parametrów na dokładność i jakość algorytmów. Wśród nich są takie, których zwiększanie jednoznacznie poprawia te wartości: procent wybranych atrybutów, maksymalna głębokość drzewa, liczba drzew w lesie losowym. Spodziewaliśmy się, że efekt przetrenowania często będzie występował dla odpowiednio wysokich wartości tych parametrów. Jednak testy pokazały, że implementacja ruletki przeciwdziałała temu zjawisku, choć były też sytuacje, w których powodowała wahanie się dokładności. W sposób niejednoznaczny na wyniki wpływało manipulowanie minimalną liczbą elementów wymaganych do podziału. Tutaj w zależności od złożoności i rozmiaru zbioru danych uzyskiwaliśmy zróżnicowane rezultaty, które ostatecznie po osiągnięciu pewnego progu prowadziły do przetrenowania obu algorytmów. Parametrami niemającymi wpływu na wyniki algorytmów są: tryb pracy algorytmu oraz warunek brzegowy ruletki. Ten drugi zależy ponownie od złożoności i rozmiaru zbioru danych, tzn. zależność wzrostu dokładności przy wzroście warunku brzegowego wystąpiła tylko w sytuacji gdy zbiór miał atrybuty ciągłe oraz był wieloklasowy. Pewnym utrudnieniem w porównywaniu algorytmów, okazał się sposób działania wyboru liczby atrybutów w algorytmie Sklearn. Z naszych doświadczeń wynika, że algorytm, w przeciwieństwie do informacji zawartych w dokumentacji, zawsze bierze pod uwagę wszystkie możliwości. Sztuczne ograniczenie ilości kolumn przekazywanych do nauki algorytmu pokazały, że przypadku braku wszystkich opcji jego skuteczność znacząco spada. Biorąc to pod uwagę oraz analizując wartości eksperymentów otrzymane we wcześniejszych, bardziej przekrojowych, ale mniej dokładnych testach, przeprowadzonych w etapie drugim, możemy być zadowoleni z uzyskanych przez nasz algorytm wyników. Przy zachowaniu najbardziej optymalnych warunków dla obu algorytmów tzn.

- ustawieniu liczby drzew na wartość większą od 200 (im więcej tym lepiej, nie zachodzi zjawisko przetrenowania, ogranicza nas tylko moc procesora oraz czas),
- wybranie około 80%-90% atrybutów (w zależności od typu danych, w przypadku danych o mało złożonej charakterystyce może to prowadzić do przetrenowania - podana wartość jest najbardziej optymalna dla naszych danych),
- określenie maksymalnej głębokości drzewa na 50 (kompromis między: czasem, a skutecznością obu algorytmów),
- wybraniem trzeciego trybu pracy (domyślny dla rozwiązania Sklearn oraz optymalny dla naszego rozwiązania),
- ustalenie minimalnej liczby elementów do dokonania podziału na: 5-15 (w zależności od liczności zbioru danych)
- określenie wartości granicznej na ok. 75% (wspomaga najlepszy wybór, jednocześnie unikając przetrenowania)

stworzony przez nas algorytm cechuje się podobną, a zazwyczaj lepszą dokładnością w porównaniu do algorytmu Sklearn.

11 Wnioski

Zadanie projektowe uznajemy za ekstremalnie ciekawe oraz rozwijające. Możliwość stworzenia oraz eksperymentowania na algorytmie uczenia maszynowego jest fascynującym zjawiskiem. Możliwość porównywania oraz ulepszania naszego rozwiązania względem algorytmu porównawczego Sklearn pod pewnymi względami przypominało grę w szachy, w której należało poświęcać czas na rzecz skuteczności, skuteczność na rzecz powtarzalności, powtarzalność na rzecz czasu. Otrzymane przez nas wyniki okazały się pod pewnymi względami zaskakujące oraz niespodziewane. Nie spodziewaliśmy się jak niski wpływ na skuteczność może mieć sposób pracy z argumentami ciągłymi oraz jak dużą rolę w naszym rozwiązaniu odegra ruletka. Nauczyliśmy się także, jak dużą wagę ma charakterystyka zbioru. Niespodziewane okazało się również, jak łatwo algorytmy uczenia maszynowego są w stanie poradzić sobie z danymi o prostej charakterystyce oraz jak po pewnych usprawnieniach są w stanie zrobić to samo z wieloklasowymi danymi o ciągłych argumentach. Ciekawym dodatkowym zagadnieniem okazała się automatyzacja raportowania zebranych danych. Dzięki stworzeniu dodatkowego rozwiązania byliśmy w stanie przeprowadzać kilkunastogodzinne testy, ponieważ otrzymane w ten sposób dane przetwarzane były automatycznie co pozwalało na prostszą ich analizę. Jest to w pewien sposób ciekawe, że do właśnie tych celów coraz częściej stosuje się sztuczną inteligencję chociażby w zagadnieniach związanych z cyberbezpieczeństwem (np. wykrywanie anomalii w ruchu sieciowym). Wiedza zdobyta w trakcie tego projektu na pewno nam się przyda w dzisiejszych realiach.

References

[1] Paweł Cichosz. “Systemy uczące się”. In: Warszawa: Wydawnictwa Naukowo-Techniczne, 2000, p. 139. ISBN: 83-204-2544-1.

[2] Yaliang Li et al. “Ensemble Learning”. In: *Handbook of Data Mining and Knowledge Discovery*. Ed. by Charu C. Aggarwal. CRC Press, Taylor Francis Group, 2015. Chap. 19, pp. 484–494. ISBN: 978-1-4665-8675-8.

[3] Ayush Singhal Gaurav Shukla. “Decision Trees : classification and regression trees (CART)”. In: p. 7. URL: https://www.niser.ac.in/~smishra/teach/cs460/23cs460/lectures/lec9.pdf.

[4] Alaa Tharwat. “Classification assessment methods”. In: 2018, pp. 170–190. URL: https://www.emerald.com/insight/content/doi/10.1016/j.aci.2018.08.003/full/pdf?title=classification-assessment-methods.

[5] Nele Verbiest, Karel Vermeulen, and Ankur Teredesai. “Evaluation of Classification Methods”. In: *Handbook of Data Mining and Knowledge Discovery*. Ed. by Charu C. Aggarwal. CRC Press, Taylor Francis Group, 2015. Chap. 24, pp. 636–642. ISBN: 978-1-4665-8675-8.