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.
Interpretacja: celem projektu jest stworzenie programu budującego decyzyjny las losowy. W tracie budowy drzew testy są wybierane przez selekcję progową z ruletką bazującą na funkcji jakości podziału przykładów przez dany test.
Założenia i wymagania: implementacja w R, Python, C++ lub C. Algorytm ma być niezależny od zbioru danych. Program musi działać pod kontrolą Ubuntu Linux 22.04.

2 Podłoże teoretyczne

2.1 Metodyka lasów losowych

Metoda lasów losowych bazuje na koncepcji modelowania zespołowego (ang. *ensemble learning*). Polega ono na wykorzystaniu wielu modeli w celu wzmocnienia siły predykcyjnej do poziomu przekraczającego ten, który każdy z tych modeli osiągnąłby osobno. Jej celem jest zmniejszenie niedoskonałości wykorzystywanych modeli.

W przypadku lasów losowych ww. modelami (nazywanymi bazowymi) są drzewa (tu: decyzyjne), skąd algorytm bierze swoją nazwę. Stosowaną metodą modelowania zespołowego jest tzw. bagging.

2.2 Bagging

Bagging to metoda polegająca na generowaniu wielu wersji modelu predykcyjnego, a następnie wykorzystania ich do stworzenia modelu agregowanego.

Jej pierwsza faza polega na niezależnym wylosowaniu ze zwracaniem podzbiorów T_i ze zbioru treningowego T. Takie próby nazywa się *boostrapowymi*. Każdy model M_i jest potem uczony zbiorem T_i (stąd niezależność modeli). W zadaniu klasyfikacji danego przykładu wyniki poszczególnych modeli są agregowane poprzez głosowanie większościowe, aby dać wspólny rezultat.

Bagging może wprowadzić znaczną poprawę predykcji niestabilnych modeli, dla których zmiany w zbiorze trenującym znacząco wpływają na sposób ich budowy. Z tego powodu jest to odpowiednia metoda do wykorzystania przy lasach losowych, których modelami bazowymi są niestabilne drzewa [1].

2.3 Drzewa decyzyjne

Drzewo jest strukturą danych składającą się z węzłów i łączących ich krawędzi. Krawędzie łączą węzły w taki sposób, że pomiędzy dwoma dowolnymi węzłami istnieje dokładnie jedna ścieżka. Węzeł, od którego nie odchodzą gałęzie, jest nazywany liściem.

Drzewo decyzyjne ma na celu klasyfikacje przykładów o pewnych atrybutach. Jego kolejne węzły odpowiadają podziałom (testom) przykładów. Gałęzie prowadzą przykład po kolejnych węzłach aż do liścia, który odpowiada klasie przykładu.

Schemat ich budowy to sekwencja decyzji (tzw. kryteriów stopu) w kolejnych węzłach: czy przykładom docierającym do danego węzła nadać klasę, czy podzielić je dalej. Celem tego działania jest zagwarantowanie minimalizacji błędu na zbiorze trenującym.

Cechuje je względnie dobra jakość predykcji. Są jednak podatne na nadmierne dopasowanie, dlatego konieczne jest stosowanie odpowiednich kryteriów stopu lub innych metod kontroli (np. przycinania) [2] [3].

3 Opis algorytmu

3.1 Drzewa decyzyjne

Zdecydowaliśmy się na oparcie naszego rozwiązania o istniejące już algorytmy takie jak ID3 i C4.5. Ich standardowe implementacje budowy drzewa decyzyjnego bazują na wybieraniu najlepszego (niewykorzystanego) atrybutu w każdej iteracji, na bazie którego w węzłach dokonywany jest podział na poszczególne gałęzie [2]. W naszym rozwiązaniu miarą jakości podziału będzie zysk informacyjny ozn. $\mathrm{IG}(x, S)$. Dla podziału zbioru S atrybutem $x \in X$ określa go wzór:

$$\mathrm{IG}(x, S) = H(S) - H(S|x)$$

przy czym:

- $H(A) = -\sum_{c \in C} p(c) \log_2 p(c)$ to entropia zbioru A, gdzie $p(c)$ to liczba przykładów klasy c w zbiorze A podzielona przez moc zbioru A,
- $H(S|x) = \sum_{v \in V_x} \frac{|S_{x=v}|}{|S|} H(S_{x=v})$ to entropia warunkowa, gdzie V_x to zbiór wartości atrybutu x, a $S_{x=v}$ to przykłady z S, dla których $x = v$.

W naszym rozwiązaniu nie będziemy wybierać jednak atrybutów o największym zysku informacji jak robi to algorytm ID3 lub C4.5. Po obliczeniu IG dla wszystkich niewykorzystanych wcześniej atrybutów najpierw odrzucimy te, których IG będzie poniżej określonego progu, a pozostałe atrybuty będą miały szansę bycia wybranym proporcjonalną do swojej jakości. Prawdopodobieństwo wyboru danego atrybutu $x \in X$ to:

$$p(x) = \frac{\mathrm{IG}(x, S)}{\sum_{y \in X} \mathrm{IG}(y, S)}$$

gdzie S to zbiór przykładów rozważany (i dzielony) w danym węźle, a X to zbiór niewykorzystanych atrybutów. Atrybut podziału A będzie zatem wynikiem losowania ze zbioru atrybutów $\{x_1, x_2, ..., x_n\}$, gdzie $P(A = x_i) = p(x_i)$ zgodnie z powyższą definicją.

Jesteśmy zatem w stanie sformułować algorytm budowy modeli bazowych w postaci pseudokodu [4][5]:

Algorytm 1 Pseudokod budowy modeli bazowych

```
1:  function DecisionTree(S, X) returns tree
2:      Let T be a new tree
3:      if all elements of S are of some class c then
4:          addLeaf(T, c); return T
5:      end if
6:      X' ← threshold(S, X)
7:      if X' is empty then
8:          addLeaf(T, bestClass(S)); return T
9:      end if
10:     A ← sample(X')
11:     addNode(T, A)
12:     for each value x of A do
13:         S_x ← elements of S with A = x
14:         if S_x is empty then
15:             Let T_x be a new tree
16:             addLeaf(T_x, bestClass(S))
17:         else
18:             T_x ← DecisionTree(S_x, X − {A})
19:         end if
20:         addBranch(T, T_x, x)
21:     end for
22:     return T
23: end function
```

gdzie:

- `addLeaf`(T,c): dodaje do drzewa T liść o klasie c,
- `addNode`(T,A): dodaje do drzewa T nowy węzeł z podziałem względem atrybutu A,
- `addBranch`(T, T_x, x): dodaje gałąź pomiędzy drzewem T i T_x opisaną przez wartość x atrybutu A,
- `bestClass`(S): zwraca klasę c, która pokrywa maksymalną liczbę przykładów w zbiorze S,
- `threshold`(S,X): zwraca zbiór par (atrybut-zysk) dla atrybutów z X, które przy podziale zbioru S dają zysk informacyjny wyższy od określonego progu (mechanizm selekcji progowej),
- `sample`(X'): losuje atrybut ze zbioru X' par (atrybut-zysk). Prawdopodobieństwo wylosowania jest proporcjonalne do zysku informacyjnego.

W powyższym pseudokodzie w pętli `for each` istotne staje się rozważenie przypadku atrybutów ciągłych. Można je rozpatrzeć na dwa sposoby:

- podział na przedziały i dyskretyzację.
- wyznaczenie wartości atrybutu, dla której uzyskamy największy zysk informacyjny przy podziale. Może być zrealizowane przez binarny warunek $A \leqslant t$. Wartość progową t można szukać jako średnią arytmetyczną dwóch wartości x_i, x_{i+1}, będących wartościami atrybutu A dla kolejnych przykładów posortowanego względem A zbioru S. Metoda wykorzystywana w algorytmie C4.5 [6].

W naszym rozwiązaniu skupimy się na drugim sposobie ze względu na jego sprawdzoną efektywność. W związku z tym pętla `for each` w algorytmie 1. zostanie wykorzystana jedynie w przypadku atrybutów dyskretnych. Dla atrybutów ciągłych wykorzystamy przywołany powyżej algorytm. Przykład jego wykonania przedstawiamy w sekcji 4.

3.2 Las losowy

Budowę lasu losowego w zadaniu klasyfikacji rozpoczyna się od budowy N drzew decyzyjnych. Zazwyczaj stosuje się co najmniej kilkaset modeli bazowych. Ich tworzenie można podzielić na dwa etapy:

- dla każdego drzewa niezależnie losuje się pewną listę atrybutów z pełnego zbioru m atrybutów. Zazwyczaj losuje się ich $\lfloor\sqrt{m}\rfloor$.
- dla każdego drzewa niezależnie losuje się ze zwracaniem zbiory (z dopuszczonymi duplikatami) ze zbioru przykładów treningowych. Na ich podstawie trenuje się drzewa, rozważając podział jedynie względem wylosowanych atrybutów.

W wyniku otrzymujemy N klasyfikatorów postaci $h(x)$, gdzie x jest wektorem (przykładem) wejściowym. Klasyfikacja przykładu x polega na wprowadzeniu go do każdego z wygenerowanych klasyfikatorów, a następnie na zagłosowaniu na najczęściej występującą klasę [7]. Możemy zatem zapisać algorytm budowy decyzyjnego lasu losowego w postaci pseudokodu:

Algorytm 2 Pseudokod decyzyjnego lasu losowego

```
1:  function RandomForest(TS, N, S) returns classifications
2:      Let Trees be an empty array
3:      A ← attributes(TS)
4:      for i=1,...,N do
5:          T ← sampleR(TS, |TS|)
6:          X ← sampleNR(A, ⌊√|A|⌋)
7:          Add DecisionTree(T, X) to Trees
8:      end for
9:
10:     Let P be an empty list of classes
11:     for each value s of S do
12:         Let C be an array of counters for all classes
13:         Set all C's counters to 0
14:         for each tree of Trees do
15:             class ← tree's classification of s
16:             Increment counter of class in C
17:         end for
18:         Add class with the biggest C counter to P
19:         or select one at random if there are > 1
20:     end for
21:     return P
22: end function
```

gdzie:

- *TS*: zbiór treningowy.
- *N*: liczba drzew decyzyjnych.

- *S*: przykłady do sklasyfikowania.
- `attributes`(*TS*): zwraca listę atrybutów przykładów ze zbioru *TS*.
- `sampleR`(*Y*,*n*): losuje ze zwracaniem *n* przykładów ze zbioru *Y* zgodnie z rozkładem jednostajnym.
- `sampleNR`(*Y*,*n*): losuje bez zwracania *n* przykładów ze zbioru *Y* zgodnie z rozkładem jednostajnym. Nie modyfikuje *Y*.

Główną wadą algorytmów bazujących na zysku informacji jest preferencja atrybutów, które mają więcej unikalnych wartości. Zazwyczaj cechują się one nadmiernie wysokim zyskiem informacji. Dzięki selekcji progowej z ruletką faworyzowanie tych atrybutów powinno zostać zredukowane. Nasze rozwiązanie może mieć zatem większą jakość predykcji niż rozwiązania takie jak ID3 czy C4.5 dla zbiorów danych o wielu wielowartościowych atrybutach.

4 Przykładowe obliczenia

Prześledźmy algorytm budowy jednego z drzew decyzyjnych lasu losowego. Rozważmy przykładowy zbiór treningowy (wzorzec - [link]) w tabeli 1, którego użyjemy do budowy drzewa. Kolejne atrybuty:

- pogoda $\in$ {słonecznie, zachmurzenie, deszcz},
- temperatura $\in [0, 20]$,
- wilgotność $\in$ {wysoka, niska},
- wiatr $\in$ {silny, słaby}

determinują klasę danego przykładu oznaczoną przez pole *dobra* $\in$ {tak, nie}.

x	pogoda	temperatura	wilgotność	wiatr	dobra
1	słonecznie	20	niska	słaby	tak
2	słonecznie	17	wysoka	silny	nie
3	zachmurzenie	13	wysoka	słaby	tak
4	deszcz	9	wysoka	słaby	nie

Tabela 1: *Zbiór danych treningowych*

Wylosujmy 2 z 4 atrybutów - niech będzie to *wiatr* oraz *temperatura*. Tworzymy nowe drzewo $T1$. Policzmy entropię źródła:

$$\begin{aligned} H(S) &= -p(\text{tak}) \log_2 p(\text{tak}) - p(\text{nie}) \log_2 p(\text{nie}) \\ &= -\frac{1}{2} \log_2 \left(\frac{1}{2}\right) - \frac{1}{2} \log_2 \left(\frac{1}{2}\right) = 1 \end{aligned}$$

Następnie zysk podziału przykładów *wiatrem*:

$$\begin{aligned} \text{IG}(\text{wiatr}, S) &= H(S) - H(S|\text{wiatr}) \\ &= 1 - \frac{3}{4} H(S_{\text{w=słaby}}) - \frac{1}{4} H(S_{\text{w=silny}}) \\ &= 1 - \frac{3}{4} \cdot 0,92 - \frac{1}{4} \cdot 0 = 0,31 \end{aligned}$$

Posortujmy dane rosnąco względem *temperatury*. Kolejność przykładów to: 4,3,2,1 (identyfikacja po x). Średnie arytmetyczne dwóch kolejnych temperatur to $t_1 = 11$, $t_2 = 15$ i $t_3 = 18,5$. Zbiór przykładów spełniających nierówność *temperatura* $\leqslant t_i$ oznaczmy jako $S_{t \leqslant t_i}$. Obliczmy entropię warunkową dla każdego t_i:

$$H(S|\text{temp}_i) = \frac{|S_{t \leqslant t_i}|}{|S|} H(S_{t \leqslant t_i}) + \frac{|S_{t > t_i}|}{|S|} H(S_{t > t_i})$$

- dla t_1: $H(S|\text{temp}_1) = 0,25 \cdot 0 + 0,75 \cdot 0,92 = 0,69$
- dla t_2: $H(S|\text{temp}_2) = 0,5 \cdot 1 + 0,5 \cdot 1 = 1$
- dla t_3: $H(S|\text{temp}_3) = 0,75 \cdot 0,92 + 0,25 \cdot 0 = 0,69$

Największy zysk informacyjny uzyskujemy dla t_1 (wybieramy pierwszą największą wartość):

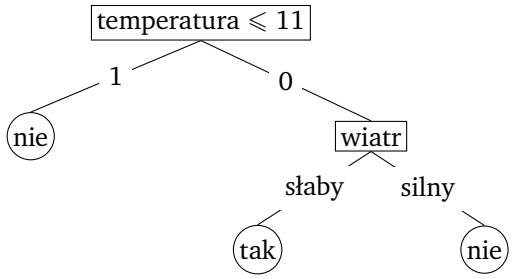
$$\text{IG}(\text{temp}_1, S) = 0,31$$

Ze względu na ograniczony przykład nie eliminujemy w tym miejscu żadnego atrybutu przez selekcję progową. Podziały względem *wiatru* i *temperatury* dają taki sam zysk informacyjny, więc losowo wybieramy jeden z nich. Niech będzie to temperatura z warunkiem $\leqslant 11$. Do drzewa $T1$ dodajemy węzeł odpowiadający ww. podziałowi. Węzeł dzieli przykłady na dwa podzbiory, które rozpatrujemy dalej:
1. Tworzymy nowe drzewo $T2$. Wszystkie przykłady zbioru $S_{t \leqslant 11} = \{4\}$ (skrócone do podania x) są klasy *nie*. Do $T2$ dodajemy liść o klasie *nie*. Łączymy $T2$ gałęzią o etykiecie "1" z drzewem $T1$.
2. Tworzymy nowe drzewo $T3$. Przykłady zbioru $S_{t > 11} = \{1, 2, 3\}$ są różnych klas. Niewykorzystanym atrybutem jest *wiatr*, więc to on zostaje wybrany. Do drzewa $T3$ dodajemy odpowiadający mu węzeł. Atrybut dzieli przykłady $S_{t > 11}$ na dwa zbiory (*silny*, *słaby*):

2.1 Dla wartości *słaby* tworzymy nowe drzewo $T4$. Wszystkie przykłady zbioru $S = \{1, 3\}$ są klasy *tak*. Dodajemy do $T4$ liść o klasie *tak*. Łączymy $T4$ gałęzią o etykiecie "słaby" z drzewem $T3$.
2.2 Dla wartości *silny* tworzymy nowe drzewo $T5$. Wszystkie przykłady zbioru $S = \{2\}$ są klasy *nie*. Dodajemy do drzewa liść o klasie *nie*. Łączymy $T5$ gałęzią o etykiecie "silny" z drzewem $T3$.

Łączymy $T3$ gałęzią o etykiecie "0" z drzewem $T1$.

Ostatecznie po wykonaniu algorytmu uzyskujemy poniższe drzewo decyzyjne:



Zauważmy, że utworzone drzewo ma zerowy błąd na zbiorze treningowym.

Pozostałe części przedstawionych algorytmów uważamy za intuicyjne i niewymagające tłumaczenia.

5 Plan eksperymentów

Planujemy wykonać szereg eksperymentów, aby zbadać wpływ określonych parametrów na jakość uzyskiwanego modelu. Takimi eksperymentami będą:

- Manipulowanie wielkością listy losowanych atrybutów podczas budowy drzew decyzyjnych. Porównanie wpływu tego parametru dla zbiorów danych o różnej złożoności.
- Manipulowanie ilością modeli bazowych, jaką wykorzystujemy do budowy lasu losowego.

Planujemy również porównać jakość predykcji lasu losowego uzyskanego w wyniku naszej implementacji z rozwiązaniem, w którym podczas budowy modeli bazowych wybierane są atrybuty o największym zysku informacji (ID3, C4.5). Szczególną uwagę zwrócimy na zbiory z wieloma atrybutami wielowartościowymi. Przypuszczamy, że nasze rozwiązanie może sprawdzić się dla nich najlepiej.

Do określenia jakości i porównania uzyskanych modeli wykorzystamy miary jakości, które opisaliśmy szerzej w następnej sekcji.

6 Miary jakości

6.1 Tablica pomyłek

Podstawową miarą jakości, którą wykorzystamy podczas ewaluacji uzyskanych modeli, będzie tablica pomyłek (ang. *confusion matrix*). Tablice pomyłek dla danych klasyfikowanych binarnie pozwalają na odczytanie 4 podstawowych wielkości badanego modelu:

- TP: liczba przykładów prawdziwie pozytywnych,
- FP: liczba przykładów fałszywie pozytywnych,
- TN: liczba przykładów prawdziwie negatywnych,
- FN: liczba przykładów fałszywie negatywnych.

Tablica pomyłek dla klasyfikacji binarnej ma następującą postać:

		Klasa rzeczywista	
		Pozytywna	Negatywna
Predykcja	Pozytywna	**TP**	**FP**
	Negatywna	**FN**	**TN**

Na podstawie powyższej tablicy oprócz wcześniej wspomnianych wielkości jesteśmy w stanie obliczyć następujące miary:

Czułość (TPR)	$\frac{TP}{TP+FN}$
Swoistość (TNR)	$\frac{TN}{FP+TN}$
Precyzja (PPV)	$\frac{TP}{TP+FP}$
Dokładność (ACC)	$\frac{TP+TN}{TP+FP+TN+FN}$
Miara F1	$\frac{2TP}{2TP+FP+FN}$

Wszystkie z tych metryk powinny być jak największe. W przypadku modeli cechujących się małą precyzją i dużą czułością (lub na odwrót) lepiej posługiwać się miarą F1.

Tablica pomyłek dla klasyfikacji wieloklasowej nieznacznie różni się od tej dla klasyfikacji binarnej. Umożliwia ona jednak przedstawienie i wyliczenie tych samych miar. Przykładową tablicę pomyłek dla klasyfikacji wieloklasowej przedstawiamy poniżej:

		Klasa rzeczywista				
		Klasa 1	Klasa 2	Klasa 3	Klasa 4	Klasa 5
Predykcja	Klasa 1	7	4	5	6	2
	Klasa 2	4	6	9	1	3
	Klasa 3	7	2	8	5	4
	Klasa 4	4	7	1	5	1
	Klasa 5	4	1	2	0	9

Z powyższej tabeli jesteśmy w stanie odczytać TP, TN, FP i FN dla każdej klasy w sposób analogiczny jak dla dwuwymiarowej tablicy pomyłek. Na ich podstawie jesteśmy w stanie wyliczyć wymienione metryki zdefiniowane identycznie jak w przypadku klasyfikacji binarnej. Wprowadza się natomiast nowe miary bazujące na definicji miary F1:

- Micro F1: wyliczane dla zsumowanych wartości TP, FP oraz FN wszystkich klas.
- Macro F1: wyliczane jako średnia arytmetyczna miar F1 wszystkich klas.
- Ważone F1: wyliczane jako średnia ważona miar F1 dla wszystkich możliwych klas, gdzie wagą jest liczba wystąpień instancji danej klasy.

Wybór prawidłowej metryki zależy ściśle od badanego zestawu danych. Dla niezbalansowanych danych najlepszym wyborem będzie macro F1 (ponieważ metryka każdej klasy jest równie ważna), dzięki czemu błąd na klasie o małej liczbie instancji nie zostanie "zaciemniony". Kiedy mamy do czynienia z zestawem danych, w którym chcemy nadać większą wagę klasom o większej liczbie instancji, powinniśmy skorzystać z ważonego F1. Jeśli mamy do czynienia ze zbalansowanym zbiorem danych, to optymalnym wyborem jest micro F1 [8].

6.2 Krzywa ROC

Do porównania modeli użyjemy również krzywej ROC. Jest ona wykresem TPR od FPR:$= 1 - \text{TNR}$ (tzw. przestrzeń ROC) wyznaczonych dla różnych wartości progu

klasyfikacji. Im szybciej do (0,1) zbliża się dana krzywa, tym lepsza jest jakość odpowiadającego jej modelu. Ponieważ model w postaci decyzyjnego lasu losowego zwraca jedynie etykiety klasy, to jego reprezentacją w przestrzeni ROC jest jeden punkt (FPR, TPR) [9]. Zestawienie wielu modeli w postaci punktów na jednej płaszczyźnie pozwala na przejrzyste przedstawienie i porównanie ich jakości.

6.3 Sposób oceny jakości

Dla każdego typu modelu lasu (tj. konkretnych parametrów) przeprowadzimy wielokrotne próby w celu oceny jego jakości. W każdej iteracji zbadamy odpowiednie metryki, a na końcu je uśrednimy. Szczególną uwagę zwrócimy na dokładność ACC.

Dla każdej próby stworzymy punkt w przestrzeni ROC, aby spróbować określić obszar, na który mapuje się dany typ modelu. Do porównania różnych typów modeli na wykresie w przestrzeni ROC naniesiemy odpowiadające im punkty, będące wynikiem uśrednienia poszczególnych prób.

Aby w sposób poprawny ocenić i porównać wpływ konkretnych parametrów na jakość modelu, zawsze będziemy modyfikować tylko jeden parametr.

7 Zbiory danych

Zdecydowaliśmy się na dwa zbiory danych, aby wykorzystać algorytm zarówno do klasyfikacji binarnej jak i do wieloklasowej. Ważna jest dla nas również dziedzina atrybutów. Szukaliśmy zbiorów, które zawierają zarówno dane ciagłe jak i dyskretne. Wybranymi przez nas zbiorami są:

- Mushroom Data Set (źródło) - atrybuty o wartościach dyskretnych, klasa binarna, występują w nim braki danych, zbalansowany.
- Red Wine Quality (źródło) - atrybuty o wartościach ciągłych, wiele klas, niezbalansowany.

7.1 Mushroom Data Set

Zbiór danych składa się z opisów 23 gatunków grzybów, którym przypisywana jest klasa `edible` lub `poisonous`. Wszystkie atrybuty są dyskretne, większość z nich jest wielowartościowa. Zbiór składa się z 8124 instancji i zawiera puste wartości.
Atrybuty wraz z ilością możliwych wartości (podaną w nawiasie) przedstawiamy poniżej:

- cap-shape: kształt kapelusza (6)
- cap-surface: tekstura kapelusza (4)
- cap-color: kolor kapelusza (10)
- bruises?: czy stłuczenia (2)
- odor: zapach (9)
- gill-attachment: szkrzela (4)
- gill-spacing: rozstaw skrzel (3)
- gill-size: rozmiar skrzeli (2)
- gill-color: kolor skrzeli (12)
- stalk-shape: kształt łodygi (2)
- stalk-root: korzeń łodygi (7, możliwe braki)
- stalk-surface-above-ring: powierzchnia łodygi powyżej pierścienia (4)
- stalk-surface-below-ring: powierzchnia łodygi poniżej pierścienia (4)
- stalk-color-above-ring: kolor łodygi powyżej pierścienia (9)
- stalk-color-below-ring: kolor łodygi poniżej pierścienia (9)
- veil-type: rodzaj welonu (2)
- veil-color: kolor welonu (4)
- ring-number: ilość pierścieni (3)
- ring-type: rodzaj pierścieni (8)
- spore-print-color: kolor odcisków zarodników (9)
- population: populacja (6)
- habitat: siedlisko (7)

Ilość wystąpień instancji poszczególnych klas:

Klasa	N	N[%]
edible	4208	51,8%
poisonous	3916	48,2%
total	8124	

7.2 Red Wine Quality

Zbiór danych, który zawiera analizę chemiczną różnych wariantów tego samego portugalskiego wina. Wszystkie atrybuty zawierają wartości ciągłe, a wynikowy atrybut to ocena jakości w skali od 0 do 10. W zbiorze danych występują jedynie wina o jakości od 3 do 8, więc tylko one będą istotne w zadaniu klasyfikacji. Zbiór składa się z 1599 przykładów i jest niezbalansowany - najwięcej jest win o jakości 5 i 6.
Atrybuty przedstawiamy poniżej:

- fixed acidity: ustalona kwasowość
- volatile acidity: lotna kwasowość
- citric acid: kwas cytrynowy
- residual sugar: cukier resztkowy
- chlorides: chlorki
- free sulfur dioxide: wolny dwutlenek siarki
- total sulfur dioxide: ogólny dwutlenek siarki
- density: gęstość
- pH: skala pH
- sulphates: siarczany
- alcohol: zawartość alkoholu

Ilość wystąpień instancji poszczególnych klas:

Klasa	N	N[%]
3	10	0,6%
4	53	3,3%
5	681	42,6%
6	638	40%
7	199	12,4%
8	18	1,1%
total	1599	

7.3 Zbiór trenujący i testowy

W celu zapewnienia niezależności programu od dostarczanych zbiorów danych zdecydowaliśmy się na wprowadzenie uniwersalnych reguł jego podziału na zbiór trenujący i testowy. Ważne jest zachowanie odpowiedniego rozmiaru obu zbiorów, aby mierzone miary jakości utworzonego modelu były wiarygodne. Dodatkowo należy zadbać o ich identyczne cechy, w szczególności rozkład klas. Z tego powodu dostarczony zbiór najpierw podzielimy na podzbiory przykładów o identycznych klasach, a następnie te podzbiory rozdzielimy w proporcji 7:3 na zbiór treningowy i testowy. Taki podział spełni nasze warunki.

8 Wykorzystywane narzędzia

Implementacja algorytmu zostanie wykonana w języku Python z wykorzystaniem poniższych bibliotek:

- `numpy` - operacje na dużych zbiorach danych oraz wiele przydatnych funkcji matematycznych,
- `pandas` - obsługa i formatowanie danych zestawu danych (podział na zbiory, obsługa wybrakowanych danych itd.),
- `matplotlib` - wizualizacja danych na wykresach,
- `scikit-learn 1.2.2` - porównanie naszej implementacji z rozwiązaniami, które wybierają najlepsze atrybuty.

Oprócz tego wykorzystamy narzędzie `Git` do wersjonowania naszego rozwiązania, co znacząco ułatwi nam pracę i pozwoli śledzić jej postęp w czasie.

Bibliografia

[1] Leo Breiman. “Bagging predictors”. In: *Machine learning* 24 (1996), pp. 123–140.

[2] Mridula Batra and Rashmi Agrawal. “Comparative Analysis of Decision Tree Algorithms”. In: *Nature Inspired Computing*. Ed. by Bijaya Ketan Panigrahi et al. Singapore: Springer Singapore, 2018, pp. 31–36. ISBN: 978-981-10-6747-1.

[3] Barry De Ville. “Decision trees”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 5.6 (2013), pp. 448–455.

[4] Nicha Kaewrod and Kietikul Jearanaitanakij. “Improving ID3 Algorithm by Ignoring Minor Instances”. In: *2018 22nd International Computer Science and Engineering Conference (ICSEC)*. 2018, p. 2. DOI: 10.1109/ICSEC.2018.8712762.

[5] Salvatore Ruggieri. “Efficient C4. 5 [classification algorithm]”. In: *IEEE transactions on knowledge and data engineering* 14.2 (2002), p. 438.

[6] J Ross Quinlan. “Improved use of continuous attributes in C4. 5”. In: *Journal of artificial intelligence research* 4 (1996), p. 78.

[7] Leo Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–6.

[8] Juri Opitz and Sebastian Burst. “Macro f1 and macro f1”. In: *arXiv preprint arXiv:1911.03347* (2019).

[9] César Ferri, Peter Flach, and José Hernández-Orallo. “Learning decision trees using the area under the ROC curve”. In: *Icml*. Vol. 2. 2002, p. 142.