UMA - dokumentacja końcowa

Piotr Jabłoński (325163) i Paweł Wysocki (325248)

Spis treści

1 Założenia

1.1 Temat projektu

Celem naszego projektu jest implementacja algorytmu konstruującego drzewo klasyfikujące, w którym test dla danego węzła jest wybierany za pomocą ruletki, tj. każdy test ma szansę na wybór proporcjonalną do swojej jakości.

1.2 Drzewo klasyfikacyjne

1.2.1 Ogólna zasada działania

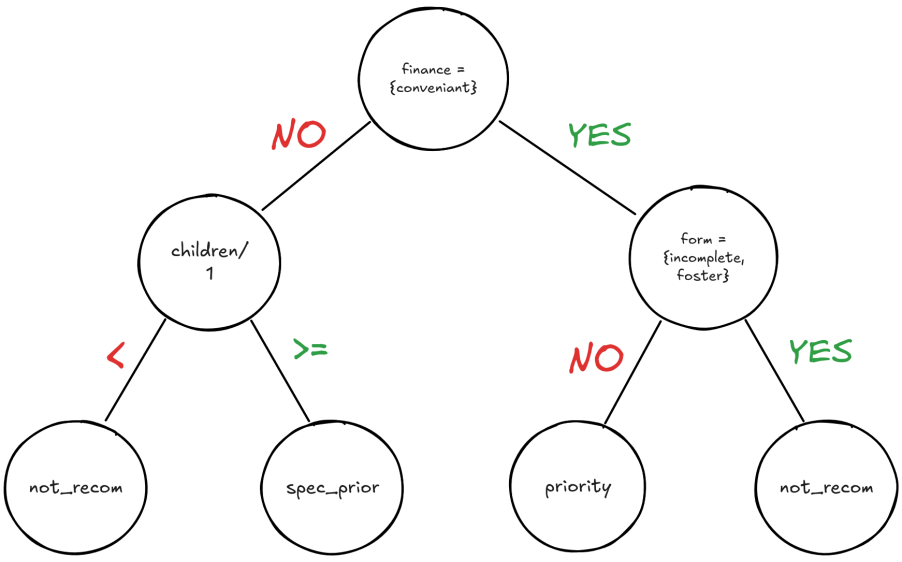
Drzewo klasyfikacyjne to relatywnie prosty model uczenia maszynowego. Polega on na konstrukcji drzewa binarnego, gdzie:

- **węzeł** - atrybut, na podstawie którego dzielimy klasy na podzbiory (tzw. "test")
- **liść** - klasa lub predykcja klasy

Drzewo można traktować jako wielką instrukcję "if-else żeby dojść do liścia, trzeba przejść przez wiele węzłów warunkowych. Kolejne testy w drzewie są wybierane automatycznie, wg wskaźnika **zysku informacji** (Information Gain) dla danego atrybutu (w każdym węźle wybieramy atrybut, dla którego jest on największy).

W podstawowej, nieograniczonej wersji drzewa klasyfikacyjnego, dążymy do momentu, gdy wszystkie dane w liściu należą do jednej klasy. Im lepszy test zostanie wybrany do podziału danych, tym precyzyjniejsze będzie nasze drzewo - oznacza to, że celem jest znalezienie takiego testu, dla którego entropia ("nieczystość") zbioru zmniejszy się najbardziej. Jest to ryzykowne podejście, ponieważ czyni ono drzewo bardzo wrażliwym na dane - niewielka zmiana danych wejściowych może znacznie zmienić konstrukcję całego drzewa. Dodatkowo, entropia jest wyliczana dla danych uczących, więc istnieje duże ryzyko przeuczenia drzewa, szczególnie dla niereprezentatywnych zbiorów danych.

Przykładowe drzewo dla zbioru danych "Nursery"



1.2.2 Entropia

Entropia to miara niejednorodności podziału klas. Im większą ma wartość, tym klasy są bardziej wymieszane. Jest obliczana wg wzoru:

$$E(X) = \sum_{d \in C} -P(c = d) \ \log P(c = d)$$

gdzie:

- X - zbiór przykładów w danym węźle
- $P(c = d)$ - p-stwo wystąpienia klasy d w zbiorze ($\frac{\text{liczba wystąpień klasy } d \text{ w zbiorze}}{\text{liczba wszystkich elementów w zbiorze}}$)

Entropia przyjmuje wartość z przedziału $[0, 1]$, dążymy do jej minimalizacji.

Entropia warunkowa określa entropię zbioru po jego podziale danym testem. Oblicza się ją wg wzoru:

$$E(X|T) = \sum_{t} \frac{|X_t|}{|X|} E(X_t)$$

gdzie:

- t - konkretna wartość dla danego testu T
- $|X_t|$ - liczba elementów zbioru, dla których testowany atrybut przyjmuje wartość t
- $|X|$ - liczba wszystkich elementów zbioru X
- $E(X_t)$ - entropia podzbioru X_t zbioru X, w którym wszystkie przykłady przyjmują wartość t dla testowanego atrybutu

Jest to efektywnie średnia ważona entropii podzbiorów podzielonych testem T.

1.2.3 Kryteria podziału

Przyjmujemy następujące kryteria podziału w węzłach:

- **atrybuty ciągłe**: binarny na podstawie nierówności $a_i \leq t$
- **atrybuty dyskretne**: binarny na podstawie przynależności do zbioru $a_i \in T$
  - wprowadzamy ograniczenie, że zbiór T może zawierać najwyżej połowę wartości atrybutu
  - w teście bierzemy pod uwagę tylko kombinację wartości zawartych w zbiorze T, która daje najwyższy przyrost informacji

1.3 Selekcja ruletkowa

Zwykłe drzewa decyzyjne są zachłanne, tzn. wybierają ten test, który ma największą jakość. Takie podejście jest proste w realizacji i bardzo efektywne, jednakże czyni drzewo bardzo podatnym na przeuczenie. W naszym przypadku selekcja będzie odbywała się dla każdego testu z zastosowaniem ruletki wg prawdopodobieństwa:

$$P(a_i) = \frac{IG(a_i)}{\sum_j^n IG(a_j)}$$

gdzie

- $P(a_i)$ - prawdopodobieństwo wyboru atrybutu a_i
- $IG(a_i)$ - zysk informacji dla atrybutu a_i
- $\sum_j^n IG(a_j)$ - suma zysków informacji dla wszystkich atrybutów możliwych do wyboru w danym węźle

Takie podejście sprawia, że prawdopodobieństwo wybrania testu dla atrybutu a_i jest wprost proporcjonalne do zysku informacji, dzięki czemu drzewa powinny mieć mniejszą podatność na przeuczenie. Problem wrażliwości na dane wejściowe również zostanie zredukowany - zmianie będą ulegały prawdopodobieństwa wyboru testu, lecz nie musi to oznaczać zmiany budowy drzewa.

1.4 Plan eksperymentów

W celu przeprowadzenia odpowiednich testów statystycznych, eksperymenty będą prowadzone na różnych zbiorach danych. Początkowo, uzyskane wyniki miały być porównane z klasyfikatorem DecisionTreeClassifer z pakietu naukowego scikit-learn, jednakże okazał się on niekompatybilny z naszą metodą wyznaczania testów dla danych kategorycznych, w związku z czym zdecydowaliśmy się sami zaimplementować klasyczne drzewo.

Macierz błędów (tablica pomyłek) posłuży nam do zwizualizowania i zweryfikowania skuteczności klasyfikacji. Będziemy skupiać się na miarach: **PPV**, **Recall** i **F1**.

2 Algorytmy

Do skonstruowania drzewa klasyfikacyjnego niezbędne są następujące algorytmy:

- Algorytm budowania drzewa
- Algorytm wyboru testu z ruletką
- Algorytm obliczania zysku informacji (**IG**)

Poniżej przedstawiamy pseudokody tych algorytmów w języku Pythono-podobnym w celu lepszego zwizualizowania ich działania:

2.1 Algorytm budowania drzewa

```
def build_tree(attrs, data, classes, max_depth) -> DecisionTree:
    if max_depth == 0 or len(attrs) == 0:
        # w przypadku danych niejednorodnych wybieramy najczęstszą klasę
        return most_common(data)
    tree = DecisionTree()

    # przeprowadzamy test z ruletką
    tree.attr, tree.threshold = test(attrs, data, classes)
    new_attr = attrs - tree.attr

    # dzielimy dane na postawie testu
    left_data, right_data = [...]
    tree.left = build_tree(new_attr, left_data, classes, max_depth - 1)
    tree.right = build_tree(new_attr, right_data, classes, max_depth - 1)

    return tree
```

2.2 Algorytm wyboru testu z ruletką

```
def test(attrs, data, classes):
    IGs = []
    for a in attrs:
        IGs.append(IG(a, data, classes))

    # ruletkowy wybór zysku informacji
    total = sum([ig.gain for ig in IGs])
    probabilities = [ig.gain / total for ig in IGs]
    return numpy.random.choice(IGs, 1, p = probabilities)
```

2.3 Algorytm obliczania zysku informacji (Information Gain)

Information Gain stosowany jest do mierzenia zmiany entropii dla danego testu. Jest obliczany wg wzoru:

$$IG(X,T) = E(X) - E(X|T)$$

gdzie:

- $E(X)$ - entropia zbioru X
- $E(X|T)$ - entropia warunkowa zbioru X po podziale testem T

Wskaźnik ten pozwala nam w łatwy sposób określić, jak bardzo dany test wpływa na nasz zbiór danych, więc dążymy do jego maksymalizacji.

3 Zbiory danych

Przygotowaliśmy 4 zbiory danych, na których będziemy prowadzić eksperymenty.

3.1 Red Wine Quality

Zawiera 11 fizykochemicznych atrybutów win:

1. Kwasowość stała
2. Kwasowość lotna
3. Kwas cytrynowy
4. Cukier pozostały po fermentacji
5. Chlorki
6. Dwutlenek siarki wolny
7. Dwutlenek siarki całkowity
8. Gęstość
9. pH
10. Siarczany
11. Procent alkoholu

Wszystkie atrybuty są ciągłe, więc zastosowany będzie wyłącznie podział nierównościowy.

Zadanie klasyfikacji - **określenie jakości wina w skali całkowitoliczbowej (1-10)**.

3.2 Loan Approval Classification

Zawiera 9 atrybutów o osobie składającej wniosek o pożyczkę oraz 4 atrybuty o samej pożyczce - łącznie 13 atrybutów, na podstawie których należy sklasyfikować stan wniosku (zaakceptowany bądź odrzucony). Atrybuty:

1. Wiek
2. Płeć
3. Wykształcenie
4. Dochód roczny
5. Liczba lat doświadczenia zawodowego
6. Stan posiadania domu (wynajem, na własność, hipoteka)
7. Kwota pożyczki
8. Cel pożyczki
9. Oprocentowanie pożyczki
10. Wysokość wypożyczenia w relacji do dochodu rocznego (%)
11. Zdolność kredytowa
12. Długość historii kredytowej w latach

13. Indykator wcześniejszych niespłaconych wypożyczeń

W tym zbiorze danych występują atrybuty zarówno ciągłe, jak i dyskretne, w związku z czym zostaną wykorzystane oba typy podziałów.

Zadanie klasyfikacji - **akceptacja wniosku o pożyczkę (prawda/fałsz)**.

3.3 Nursery

Zawiera 8 atrybutów dotyczących rodziny:

1. Zawód rodziców
2. Jakość zapewnionej opieki nad dzieckiem
3. Struktura rodziny (kompletna, rozbita, itp.)
4. Liczba dzieci
5. Warunki zamieszkania
6. Sytuacja finansowa
7. Sytuacja społeczna
8. Sytuacja zdrowotna

W tym zbiorze występują atrybuty dyskretne, jednakże zostaną zastosowane oba typy podziałów, ponieważ liczbę dzieci da się podzielić nierównościowo.

Zadanie klasyfikacji - **ocena aplikacji do przedszkola (wieloklasowa)**.

3.4 Mobile Device Usage and User Behavior

Zawiera 10 atrybutów (korzystamy z 9), które opisują parametry urządzeń mobilnych oraz aktywności użytkowników:

1. Id użytkownika - nieużywane, bo nieistotne
2. Model urządzenia
3. System operacyjny
4. Czas używania aplikacji
5. SOT (Screen On Time)
6. Codzienne zużycie baterii (mAh)
7. Liczba zainstalowanych aplikacji
8. Codzienne zużycie danych (MB)
9. Wiek
10. Płeć (M/K)

W tym zbiorze występują atrybuty ciągłe i dyskretne, więc zostaną zastosowane oba typy podziałów.

Zadanie klasyfikacji - **ocena zachowania użytkownika (od lekkiego do ekstremalnego użycia w skali całkowitoliczbowej 1-5)**.

4 Opis funkcjonalny

4.1 Dane wejściowe i wyjściowe

4.1.1 Budowanie drzewa

Podczas budowy drzewa, dane wejściowe stanowi zbiór próbek testujących w postaci:

- Tablica atrybutów próbek (zarówno numerycznych, jak i kategorycznych)
- Wektor klas wyjściowych (zazwyczaj kategorycznych)

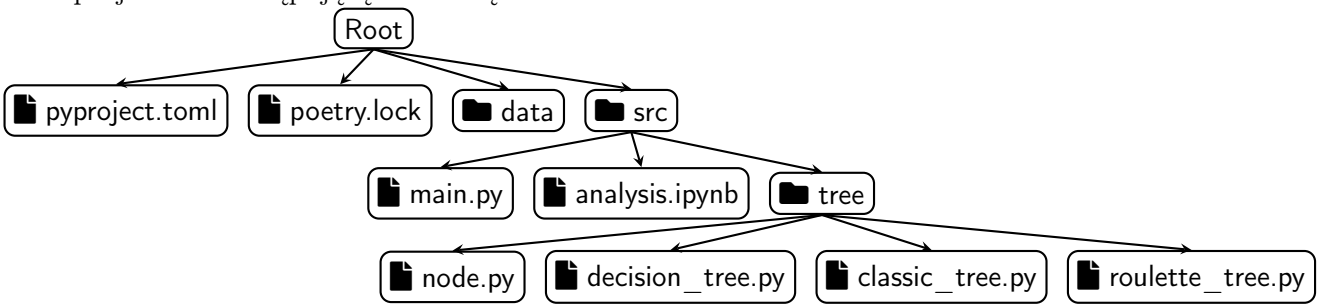
Na ich podstawie tworzona jest struktura drzewiasta, która pozwala na przeprowadzenie predykcji.

4.1.2 Predykcja

W przypadku predykcji, dane wejściowe stanowią wyłącznie atrybuty próbki (ewentualnie tablica, jeżeli generujemy predykcję dla wielu próbek jednocześnie). Natomiast w wyniku predykcji otrzymujemy przewidzianą klasę wyjściową dla danej próbki (ewentualnie wektor klas).

4.2 Pliki programu

Nasz projekt ma następującą strukturę:



Mniej znaczące pliki to:

- Folder *data* - znajdują się tu zbiory danych wykorzystywane w testach.
- Pliki *poetry.lock* oraz *pyproject.toml* - pliki zawierające informacje o używanych pakietach, pozwalające uruchomić projekt.
- Plik *main.py* - prosty plik pozwalający na szybkie uruchomienie modelu w celu demonstracji / sprawdzenia jego działania.
- Plik *analysis.ipynb* - plik wykorzystywany do przeprowadzania testów i generowania wykresów wykorzystywanych w dokumentacji.

Pliki zawierające implementację drzewa znajdują się w folderze *tree*.

4.2.1 Plik *node.py*

W pliku tym znajduje się implementacja węzła decyzyjnego drzewa klasyfikującego. Klasa *Node* składa się on z pól:

- *value* - w przypadku liścia jest to przewidywana klasa, natomiast w przypadku węzła nim niebędącego - test w postaci: (indeks atrybutu, wartość graniczna / zbiór wartości dla danych kategorycznych)
- Pola *left* i *right* - pola wykorzystywane przez węzły niebędące liśćmi jako wskaźniki na kolejne węzły drzewa

Jedyną metodą zdefiniowaną dla tej klasy jest *predict* - pozwala ona przejść rekursywnie przez drzewo w celu określenia klasy, do której przynależy dana próbka.

4.2.2 Plik *decision_tree.py*

Plik ten zawiera główną implementację drzewa decyzyjnego. Klasa *DecisionTree* zawiera następujące pola:

- *features* - lista atrybutów zbioru danych, do którego przystosowane będzie drzewo
- *max_depth* - maksymalna głębokość, jaką może osiągnąć drzewo decyzyjne. Parametr opcjonalny - nieokreślenie głębokości spowoduje, że drzewo będzie rosło dopóki dane treningowe nie zostaną całkowicie podzielone
- *min_samples_split* - minimalna liczba próbek znajdujących się w danym węźle, które można podzielić. Domyślnie wartość wynosi 2
- *min_samples_leaf* - jeżeli po podziale danych po którejś stronie zostanie mniej próbek niż ta wartość, podział nie będzie przeprowadzony i węzeł zostanie uznany za liść. Domyślnie wartość wynosi 5
- *num_thresholds* - liczba rozpatrywanych wartości progowych przy podziale wg atrybutu ciągłego (równe odstępy od wartości minimalnej do maksymalnej). Domyślnie wartość wynosi 20
- *root* - korzeń drzewa decyzyjnego

Metody zdefiniowane dla tej klasy, które mają znaczenie z perspektywy użytkownika to:

- *fit* - metoda wykorzystywana do budowy drzewa decyzyjnego. Jako argumenty wejściowe przyjmuje tablicę atrybutów próbek oraz wektor odpowiadających im klasom. Po jej wywołaniu, możliwe jest przewidywanie klas dla innych próbek
- *predict* - metoda wykorzystywana do przewidywania do klasy dla danej próbki. Jako argument wejściowy przyjmowany jest wektor atrybutów danej próbki. Możliwe jest również przewidywanie dla większej ilości próbek naraz

Pozostałe metody zdefiniowane dla tej klasy, ale nieużywane przez użytkownika to:

- *test* - metoda wykorzystywana do określenia testu dla danego węzła
- *choose_test* - metoda abstrakcyjna, odpowiedzialna za wybór testu na podstawie Information Gain. Implementacja różni się w zależności od typu drzewa decyzyjnego, dlatego jest zdefiniowana w pozostałych klasach

- *info_ gain* - metoda wykorzystywana do obliczania zysku informacyjnego
- *entropy* - metoda wykorzystywana do obliczania entropii zbioru
- *split_ data* - metoda wykorzystywana do podziału zbioru na podstawie testu

4.2.3 Plik *classic_ tree.py*

Początkowo, planowaliśmy do porównania wykorzystać klasyfikatore DecisionTreeClassifer z pakietu naukowego scikit-learn, jednakże okazał się on niekompatybilny z naszą metodą określania testów dla atrybutów kategorycznych. W związku z tym, zaimplementowaliśmy klasyczne drzewo własnoręcznie. Jedyną zaimplementowaną rzeczą jest definicja wspomnianej wcześniej metody abstrakcyjnej *choose_ test* - po prostu wybieramy test, dla którego Information Gain jest największy.

4.2.4 Plik *roulette_ tree.py*

W pliku tym zaimplementowana jest podklasa drzewa decyzyjnego, w której wybór testu jest dokonywany przez selekcję ruletkową.

5 Testy

Poniżej przedstawiamy analizę wyników uzyskanych dla wspomnianych wcześniej zbiorów danych. Testy zostały przeprowadzone dla różnych maksymalnych głębokości drzew. Klasyczna implementacja drzewa nie zawiera losowości, w związku z czym budujemy tylko jedno drzewo. Natomiast dla naszej implementacji budujemy 10 drzew z ruletką, a na wykresach przedstawiamy wartości metryk: uśrednioną, minimalną oraz maksymalną.

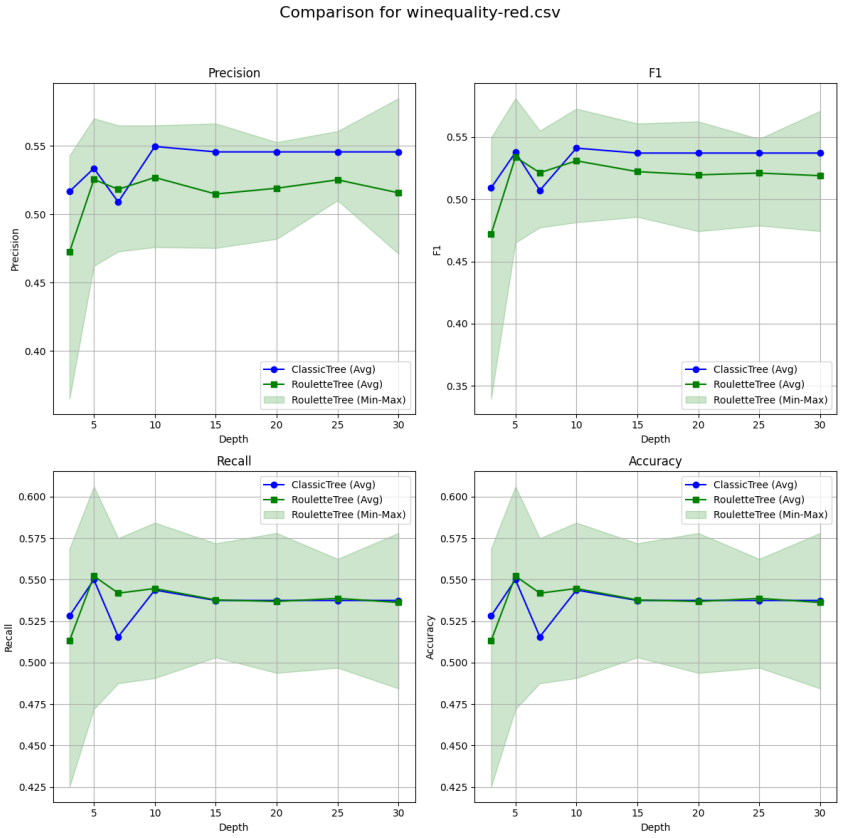
Wszelkie testy zostały przeprowadzone dla domyślnych wartości pozostałych parametrów, ponieważ nie wpływają one na wyniki.

5.1 Metryki

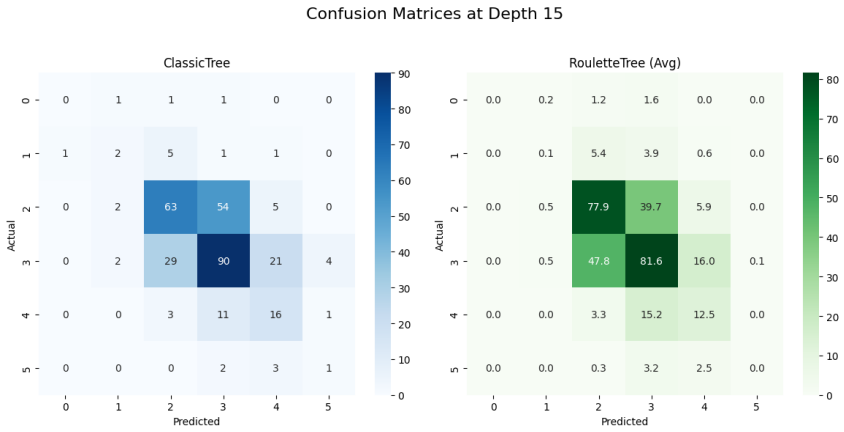
W analizie wzięliśmy pod uwagę następujące miary:

- **Macierz błędów** - macierz pozwalająca w prosty sposób określić skuteczność predykcji dla konkretnych klas. Wiersze przedstawiają rzeczywiste klasy próbek, a kolumny - przewidziane przez algorytm
- **Accuracy** - proporcja poprawnie sklasyfikowanych przypadków do wszystkich przypadków
- **Precision** - średnia ważona proporcji prawidłowych przewidywań pozytywnych wśród wszystkich przypadków uznanych za pozytywne (także błędnie) obliczonych dla każdej klasy oddzielnie
- **Recall** - średnia ważona proporcji przypadków pozytywnych sklasyfikowanych poprawnie do wszystkich przypadków pozytywnych (także błędnie sklasyfikowanych jako negatywne) obliczonych dla każdej klasy oddzielnie
- **F1** - średnia harmoniczna precision i recall ($\mathrm{F1} = 2\frac{\mathrm{Precision}\cdot\mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}}$), zapewnia bardziej zrównoważoną ocenę, szczególnie w przypadku niezbalansowanych danych

5.2 Zbiór Red Wine Quality



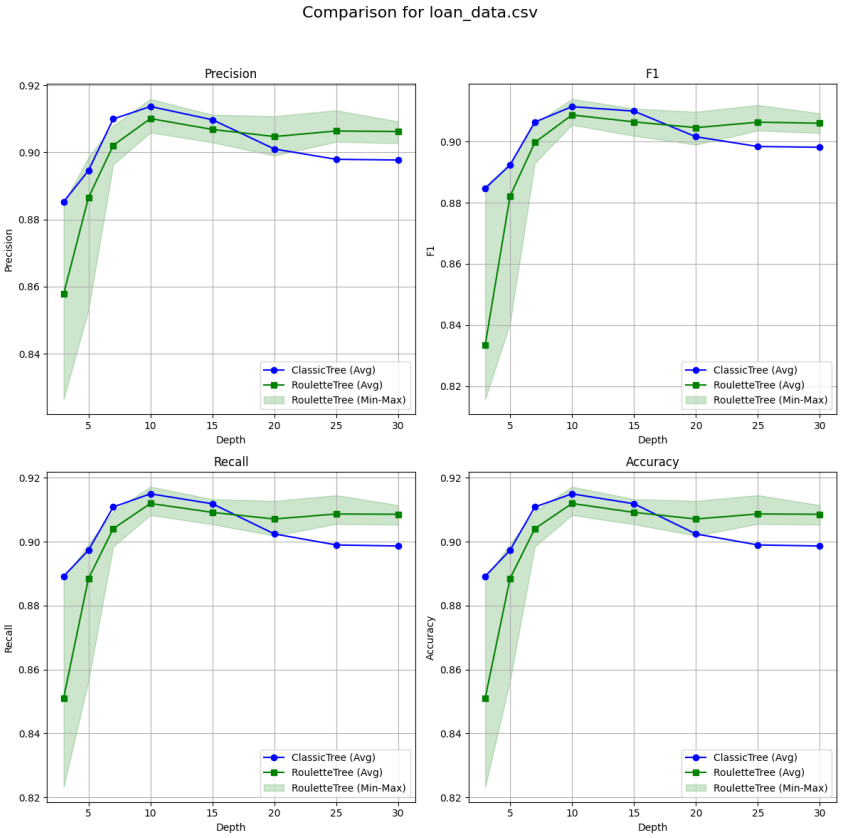
Rysunek 1: Metryki obliczone dla zbioru **Red Wine Quality**



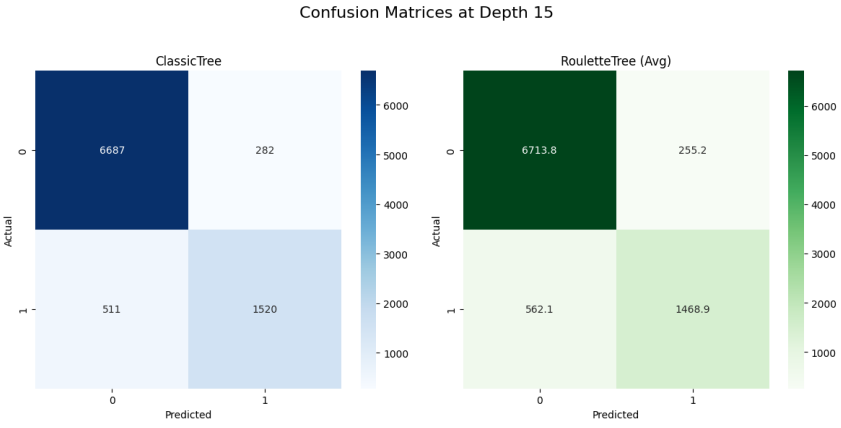
Rysunek 2: Macierz błędów dla zbioru **Red Wine Quality**

Jak widać na powyższych wykresach, zbiór ten nie nadaje się do klasyfikacji za pomocą drzew - dokładność jest na poziomie ok. 55%. Pokazuje to nam jednak ciekawą zależność - w takim przypadku możliwe jest osiągnięcie lepszych wyników przy zastosowaniu ruletki. Niestety, jest to jednak sytuacja rzadka - zazwyczaj wynik będzie gorszy niż uzyskany dla klasycznego drzewa, co widać szczególnie na wykresach precyzji i F1. Sytuację tą potwierdza macierz pomyłek wyznaczona dla drzew o głębokości 15 - błędne przewidywania są bardziej "rozproszone"w przypadku drzewa ruletkowego.

5.3 Zbiór Loan Approval Classification



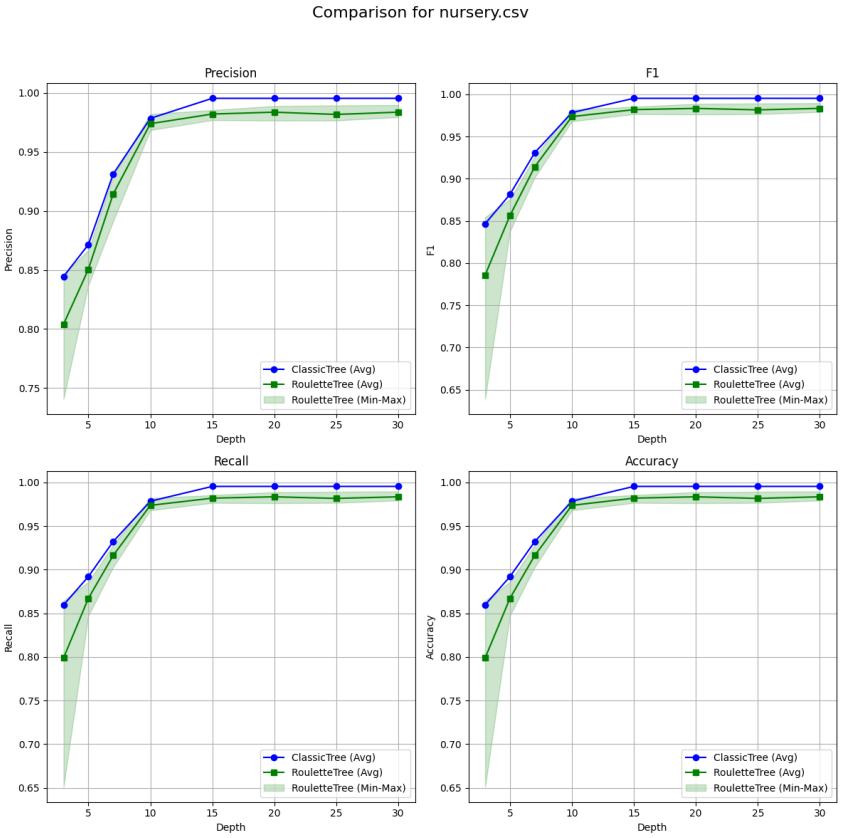
Rysunek 3: Metryki obliczone dla zbioru **Loan Approval Classification**



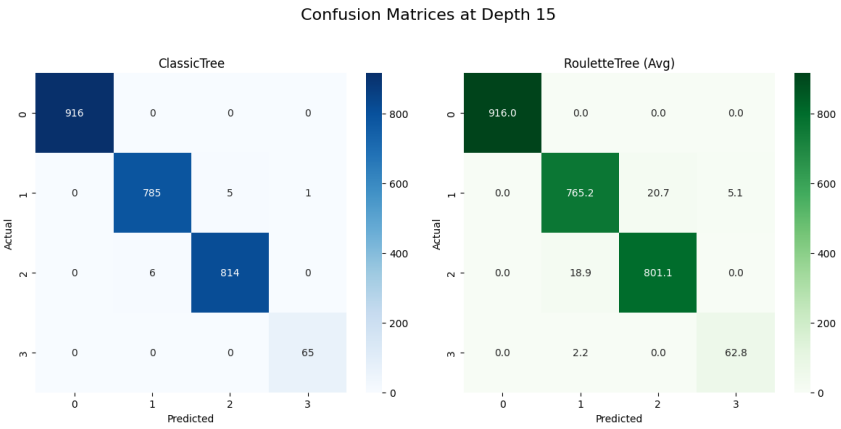
Rysunek 4: Macierz błędów dla zbioru **Loan Approval Classification**

Jak widać, zbiór ten jest dobrze przystosowany do klasyfikacji za pomocą drzew - metryki są na poziomie ok. 90%. Niemniej jednak, ukazuje się tu jedna z przewag drzewa ruletkowego - jest ono znacznie odporniejsze na przeuczenie. Widać to w szczególności dla głębokości drzew 20-30 - klasyczne drzewo ulega tu silnemu przeuczeniu, dając coraz gorsze wyniki, natomiast drzewo ruletkowe utrzymuje swoją jakość niezależnie od głębokości. Wynika to z faktu, że w końcowych fazach budowy drzewa żaden test nie niesie już dużych zysków, w związku z czym każdy z nich ma podobną szansę na zostanie wybranym - losowość ta chroni drzewo przed przeuczeniem.

5.4 Zbiór Nursery



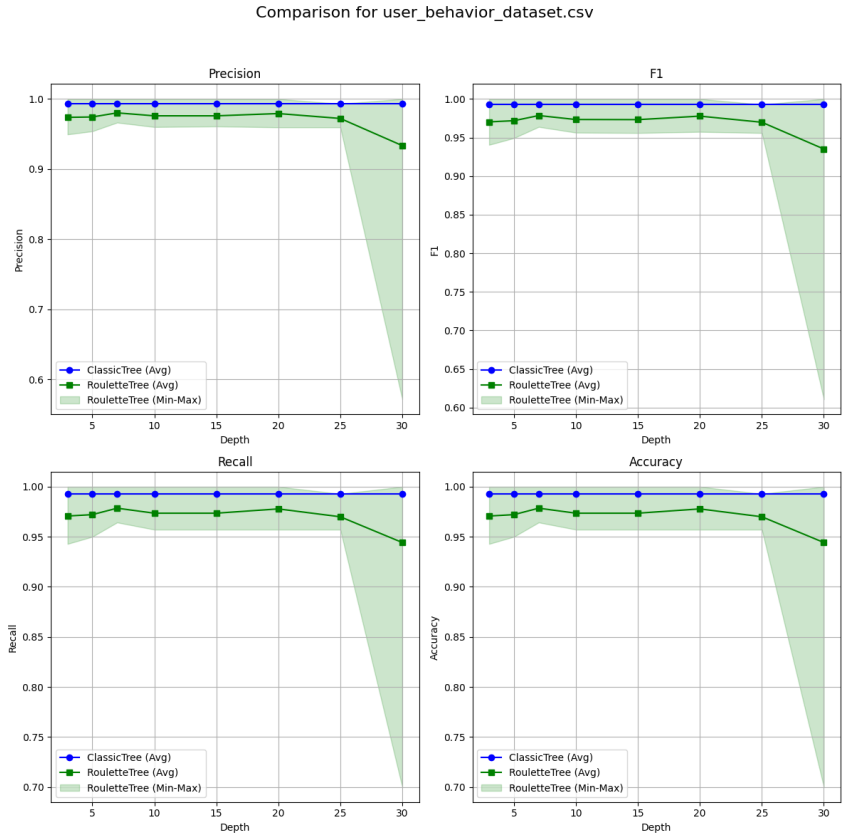
Rysunek 5: Metryki obliczone dla zbioru **Nursery**



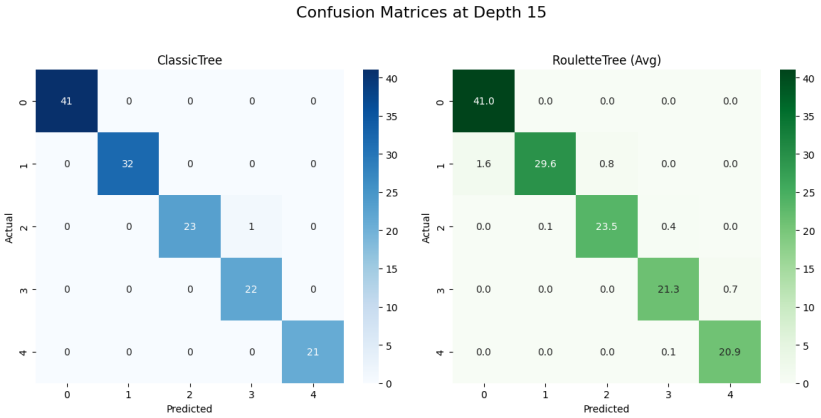
Rysunek 6: Macierz błędów dla zbioru **Nursery**

Zbiór ten jest bardzo dobrze przystosowany do klasyfikacji za pomocą drzew - klasyczne drzewo uzyskuje dla odpowiedniej głębokości skuteczność 100%. Dobrze wizualizuje on wadę drzew ruletkowych - dla takich zbiorów daje gorsze wyniki. Najprawdopodobniej wynika to z losowości - blisko 100% skuteczności, testy mają mały przyrost informacji, w związku z czym szansa na wybranie każdego z nich jest podobna, co działa na niekorzyść jakości predykcji (uniemożliwia wręcz uzyskanie 100% jakości). Widać to także na macierzy pomyłek - drzewo ruletkowe średnio myli się znacznie częściej od klasycznego.

5.5 Zbiór Mobile Device Usage and User Behavior



Rysunek 7: Metryki obliczone dla zbioru **Mobile Device Usage and User Behavior**



Rysunek 8: Macierz błędów dla zbioru **Mobile Device Usage and User Behavior**

Na zbiorze tym klasyczne drzewo uzyskuje skuteczność 99% niezależnie od głębokości, natomiast drzewo ruletkowe w niektórych przypadkach jest w stanie uzyskać 100% skuteczności, jednakże jest to sytuacja rzadka - w większości przypadków uzyskany wynik jest gorszy. Dodatkowo, uwidacznia się tutaj kolejne ryzyko związane z losowością - dla głębokości 25-30 drzewo znacznie traci na swojej jakości. Wynika to z wspomnianego wcześniej dobierania testów przy bardzo niskich wzrostach informacji - wybór nieodpowiedniego testu może znacznie obniżyć jakość całego modelu (np. dla głębokości 30 jakość spadła nawet do 70%).

5.6 Podsumowanie

Wykorzystanie ruletki przy wyborze testów w procesie budowy drzewa decyzyjnego zdecydowanie ma zastosowanie dla określonych zbiorów danych. W przypadkach, gdy dla określonych danych klasyczne drzewo nie daje zadowalających wyników (np. Red Wine Quality), warto zbudować kilka drzew ruletkowych - niewykluczone, że któreś z nich okaże się bardziej efektywne. Ponadto, dla zbiorów, do których model klasyczny nie dopasowuje się idealnie (ok. 90% skuteczności, np. Loan Approval Classification), opłacalne może być skorzystanie z modelu ruletkowego, ponieważ osiąga on niegorsze wyniki, a dodatkowo gwarantuje odporność na przeuczenie. Warto również stosować je w sytuacji, gdy nie mamy pewności czy zbiory treningowy i testowy dobrze reprezentują domenę problemu - jakość nieznacznie się pogorszy (np. Nursery), ponieważ dopasowanie do zbioru zmniejszy się, ale dzięki temu zwiększy się szansa na uzyskanie lepszych wyników na danych rzeczywistych.

Należy jednak pamiętać, że losowy wybór testów może także wpływać negatywnie na działanie modelu - dobór nieodpowiedniego testu może znacznie pogorszyć jakość modelu. Jest to szczególnie widoczne w sytuacji, gdy klasyczny model dopasowuje się praktycznie idealnie do zbioru - wtedy testy nie niosą praktycznie żadnego zysku informacji, przez co ich wybór staje się kompletnie losowy. To sprawia, że przy zbyt dużej głębokości drzewa bardzo prawdopodobne staje się pojawienie testów, które będą miały bardzo negatywny wpływ na działanie modelu.

6 Narzędzia i biblioteki

Wykorzystane przez nas biblioteki to:

- numpy - biblioteka przydatna w wykonywaniu skomplikowanych obliczeń.
- pandas - biblioteka wykorzystywana do analizy danych i manipulacji nimi.
- sklearn.metrics - biblioteka używana do obliczania metryk dla uzyskanych wyników.
- matplotlib - biblioteka do tworzenia wykresów w celu obrazowania danych.