POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Sztuczna Inteligencja

Temat: **Realizacja klasyfikatora przy użyciu drzewa decyzyjnego uczącego się rozpoznawania rodzajów kwiatu Irysa.**

Projekt opracował
Eryk Delikat
Nr albumu: 168139
Kierunek: Informatyka
Kod: 3EF-ZI
Grupa projektowa: P01

Rzeszów, czerwiec 2023

Spis treści

1. Opis problemu

Celem tego projektu jest stworzenie drzewa decyzyjnego do klasyfikacji danych zbioru Iris. Zbiór Iris zawiera informacje dotyczące trzech gatunków irysów, a zadaniem jest przewidzenie do jakiego gatunku irysa należy dany rekord na podstawie cech takich jak długość i szerokość płatków oraz długość i szerokość kielicha.

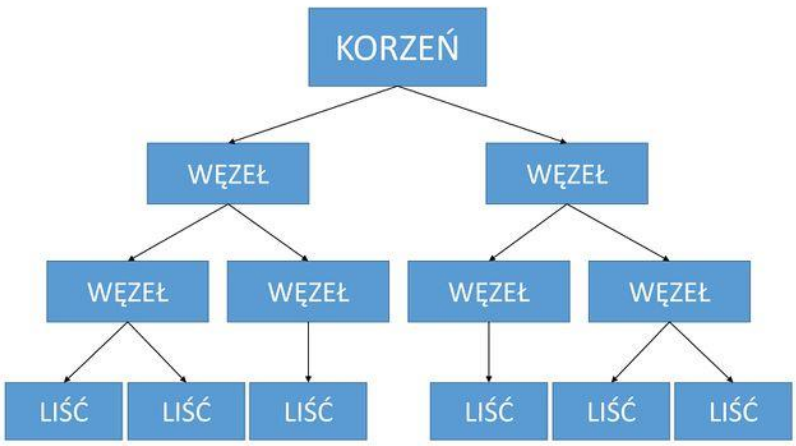
Do rozwiązania tego problemu używamy języka programowania Python wraz z kilkoma bibliotekami do uczenia maszynowego. Główne biblioteki wykorzystane w projekcie to NumPy, scikit-learn oraz hickle. Dzięki wykorzystaniu tych bibliotek możemy efektywnie przetwarzać i analizować dane, tworzyć model drzewa decyzyjnego oraz oceniać jego jakość klasyfikacji.

2. Część teoretyczna

2.1. Budowa drzewa decyzyjnego

Drzewo decyzyjne jest grafem drzewa składającym się z korzenia, węzłów, krawędzi i liści. Liście reprezentują końcowe klasyfikacje, natomiast korzeń jest tworzony na podstawie wybranego atrybutu. Każda gałąź drzewa reprezentuje wartości tego atrybutu. Drzewa decyzyjne, zbudowane na podstawie danych empirycznych, umożliwiają klasyfikację nowych obiektów, które nie były uwzględnione podczas tworzenia drzewa.

Charakterystyczną cechą drzew decyzyjnych jest ich hierarchiczna struktura. W kolejnych krokach dzieli się zbiór obiektów, odpowiadając na pytania dotyczące wartości wybranych cech lub ich kombinacji liniowych. Ostateczna decyzja jest podejmowana na podstawie odpowiedzi na wszystkie pytania.



Rys. 1 Schemat budowy drzewa decyzyjnego

2.2. Opis matematyczny drzewa decyzyjnego

Drzewo decyzyjne można matematycznie opisać jako skierowany graf acykliczny. Niech V będzie zbiorem węzłów, a E będzie zbiorem skierowanych krawędzi. Niech v_0 będzie korzeniem drzewa decyzyjnego, czyli początkowym węzłem, a v_1, v_2, ..., v_n będą węzłami wewnętrznymi lub liśćmi. Każdy węzeł v_i, różny od korzenia, jest połączony skierowaną krawędzią z węzłem rodzica (v_i). Węzeł korzenia v_0 nie ma rodzica i stanowi początek drzewa.

Niech a(v_i) będzie atrybutem testowanym w węźle v_i. Jeśli v_i jest węzłem wewnętrznym, to a(v_i) reprezentuje wartość atrybutu, na podstawie którego dokonywany jest podział. Jeśli v_i jest liściem, a(v_i) reprezentuje przypisaną klasę lub prognozę.

Formalnie, drzewo decyzyjne można zdefiniować jako uporządkowaną parę (V, E), gdzie V reprezentuje zbiór węzłów drzewa, a E reprezentuje zbiór skierowanych krawędzi, które łączą te węzły.

Ten opis matematyczny drzewa decyzyjnego jest ogólnym sposobem przedstawienia struktury drzewa w kontekście teorii grafów i algorytmów. Implementacja drzewa decyzyjnego w konkretnym języku programowania i bibliotece może różnić się szczegółami, ale ogólna struktura i zasady działania pozostają zgodne z opisanym modelem.

2.3. Tworzenie drzewa decyzyjnego przy pomocy funkcji DecisionTreeClassifier

Algorytm budowy drzewa decyzyjnego przy użyciu funkcji DecisionTreeClassifier składa się z kilku kroków. Na początku inicjalizuje się parametry budowy drzewa, takie jak:

criterion - Określa kryterium używane do podziału węzłów drzewa decyzyjnego. Dwie najczęściej używane wartości to **gini** oraz **entropy**. Parametr ten wpływa na wybór optymalnego podziału węzła podczas tworzenia drzewa decyzyjnego.

Domyślnie używane jest kryterium gini, które jest stosowane to wyznaczania nieczystości. Nieczystość (impurity) odzwierciedla stopień mieszania różnych klas w danym węźle. Dla problemu klasyfikacji trzech klas, indeks gini jest obliczany jako suma kwadratów proporcji klas, które są różne od zera w danym węźle. Minimalizacja indeksu gini dąży do tworzenia bardziej jednorodnych podgrup klasowych w drzewie decyzyjnym, gdzie podgrupy zawierają przeważająco jedną klasę.

Indeks Gini można obliczyć za pomocą wzoru:

$$gini(p) = 1 - \sum_{i=1}^{K} {p_i}^2 \quad (1)$$

gdzie:

- $gini(p)$to indeks gini dla danego węzła
- p to wektor proporcji klas w danym węźle, gdzie p_i to proporcja klasy i
- K to liczba klas

Drugim kryterium miary nieczystości jest entropia, często wykorzystywana w drzewach decyzyjnych. Entropia odzwierciedla stopień nieuporządkowania w danym węźle. Dla problemu klasyfikacji wieloklasowej entropia jest obliczana jako suma przeciwnych wartości logarytmów prawdopodobieństw klas w danym węźle. Minimalizacja entropii dąży do tworzenia bardziej jednorodnych podgrup klasowych w drzewie decyzyjnym, gdzie podgrupy składają się głównie z jednej klasy.

$$entropia(p) = -\sum_{i=1}^{K} p_i \log_2(p_i) \tag{2}$$

gdzie:

- $entropia(p)$ to entropia dla danego węzła
- p to wektor proporcji klas w danym węźle, gdzie p_i to proporcja klasy i
- K to liczba klas

Oba kryteria, gini i entropia, są używane w przypadku klasyfikacji wieloklasowej i różnią się miarą nieczystości, której minimalizacji dążą. Wybór między nimi zależy od specyfiki problemu. W praktyce oba kryteria mogą prowadzić do podobnych wyników, a wybór jednego z nich ma niewielki wpływ na działanie algorytmu, szczególnie w przypadku dobrze zbalansowanych danych.

max_depth - Maksymalna głębokość drzewa decyzyjnego. Określa maksymalną liczbę poziomów w drzewie. Wyższa wartość może prowadzić do bardziej skomplikowanych modeli, które mogą dopasowywać się zbyt dokładnie do danych uczących (overfitting), dlatego istotne jest odpowiednie dostosowanie tego parametru, aby uniknąć nadmiernego dopasowania.

splitter - Strategia podziału węzłów. Może przyjąć dwie wartości: best lub random. Best wybiera najlepszy podział na podstawie określonego kryterium, takiego jak gini lub entropia. Random wybiera losowy podział spośród najlepszych.

min_samples_split - Minimalna liczba próbek wymagana do podziału węzła. Określa minimalną liczbę próbek, które muszą być obecne w węźle, aby podział był dokonywany. W przypadku węzłów o liczności mniejszej niż min_samples_split, podział nie będzie kontynuowany.

min_samples_leaf - Minimalna liczba próbek wymagana w liściu. Określa minimalną liczbę próbek, które muszą być obecne w liściu, aby został utworzony. Jeśli podział prowadzi do utworzenia liścia o liczności mniejszej niż min_samples_leaf, podział zostanie odrzucony.

min_weight_fraction_leaf - Minimalna frakcja sumy wag próbek wymagana w liściu. Określa minimalny udział wag próbek, które muszą być obecne w liściu, aby został utworzony. Może być użyteczny w przypadku, gdy próbki mają różne wagi.

max_features - Odzwierciedla liczbę cech branych pod uwagę podczas procesu poszukiwania optymalnego podziału w węzłach. Jego wartość ma kluczowe znaczenie dla efektywności i dokładności modelu. Może przyjmować wartość **sqrt**, która jest równa pierwiastkowi kwadratowemu liczby wszystkich cech. Jeśli dane są 4 cechy, to max_features wynosi: $\sqrt{4} = 2$. Ta metoda ma na celu ograniczenie liczby cech branych pod uwagę, jednocześnie zachowując

pewną różnorodność.
Kolejną możliwą wartością jest **log2** logarytm o podstawie 2. W takim przypadku wartość max_features jest równa logarytmowi o podstawie 2 liczby wszystkich cech. Jeśli dane są 4 cechy, to wartość parametru wynosi $\log_2 4 = 2$. Podobnie jak w przypadku "sqrt", ta metoda ma na celu kontrolowanie liczby cech branych pod uwagę.

max_leaf_nodes - Maksymalna liczba liści. Określa maksymalną liczbę liści, które mogą być utworzone w drzewie. Ustawienie tego parametru może prowadzić do wcześniejszego zatrzymania tworzenia drzewa.

min_impurity_decrease - Minimalny spadek czystości wymagany do podziału węzła. Określa minimalną wartość, o jaką indeks gini lub entropia muszą się zmniejszyć, aby podział był dokonywany. Wartość większa niż zero promuje bardziej znaczące podziały.

ccp_alpha - Parametr przycinania Cost-Complexity Pruning (CCP). Określa parametr regularyzacji, który kontroluje kompromis między prostotą drzewa a jego dopasowaniem do danych uczących. Wyższe wartości ccp_alpha prowadzą do znaczącego przycinania drzewa.

Następnie dane wejściowe i docelowe są odpowiednio przygotowywane, na przykład poprzez przekształcenie i normalizację. Zbiór danych zostaje podzielony na zbiór treningowy i testowy, zwykle przy użyciu metody walidacji krzyżowej lub podziału na zestawy treningowe i testowe.

Kolejnym krokiem jest tworzenie instancji klasy DecisionTreeClassifier z odpowiednimi parametrami.
Ta instancja reprezentuje drzewo decyzyjne. Następnie drzewo jest dopasowywane do danych treningowych za pomocą metody fit(), gdzie analizuje cechy i etykiety treningowe w celu nauczenia się reguł decyzyjnych.

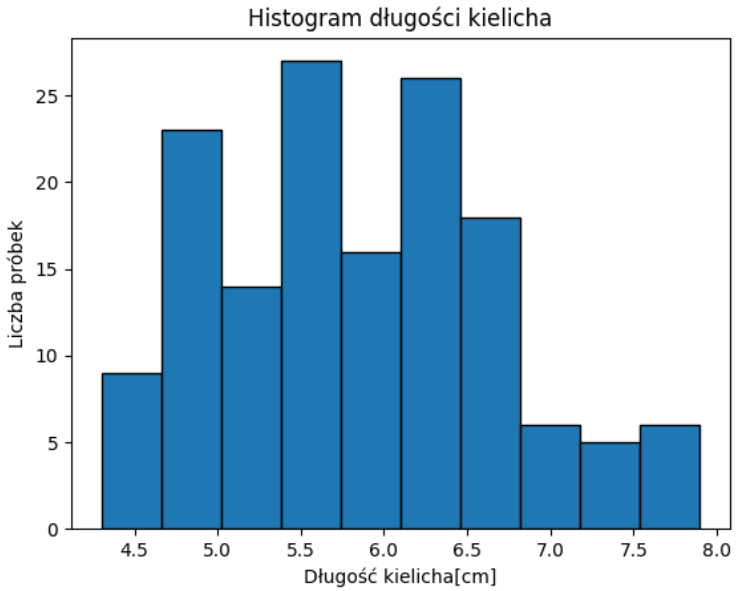
Po dopasowaniu drzewa, można go wykorzystać do predykcji na danych testowych za pomocą metody predict(). Drzewo decyzyjne dokonuje klasyfikacji lub prognozowania na podstawie wcześniej nauczonych reguł decyzyjnych.

3. Analiza danych

Dane, które zostały użyte do utworzenia drzewa decyzyjnego pochodzą ze strony http://archive.ics.uci.edu/ml/datasets/Iris.

1. Wielkość danych „Iris”:
    - Liczba rekordów: 150
    - Liczba cech: 4
    - Liczba klas: 3 (Iris setosa, Iris versicolor, Iris virginica)
2. Statystyki podsumowujące: (wartości podane w [cm])

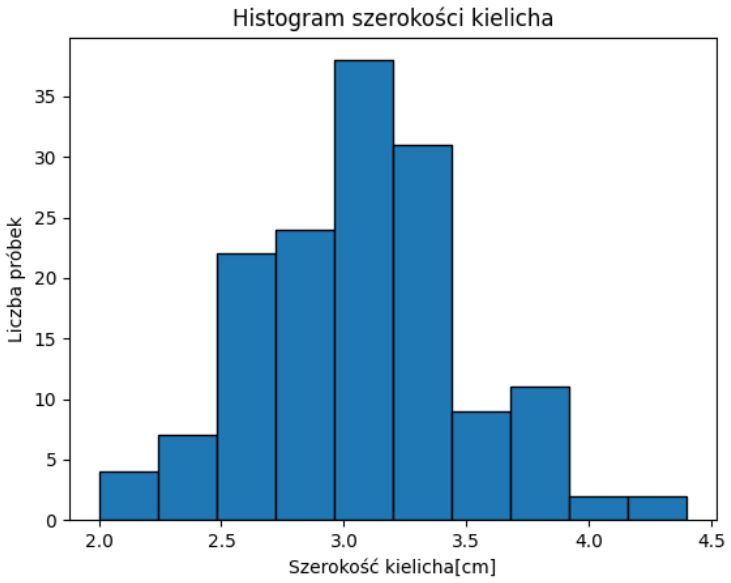
    a) Długość kielicha:
    - Minimum: 4.3
    - Maksimum: 7.9
    - Średnia: 5.84
    - Odchylenie standardowe: 0.83



Rys. 3.1 Histogram długości kielicha

b) Szerokość kielicha:

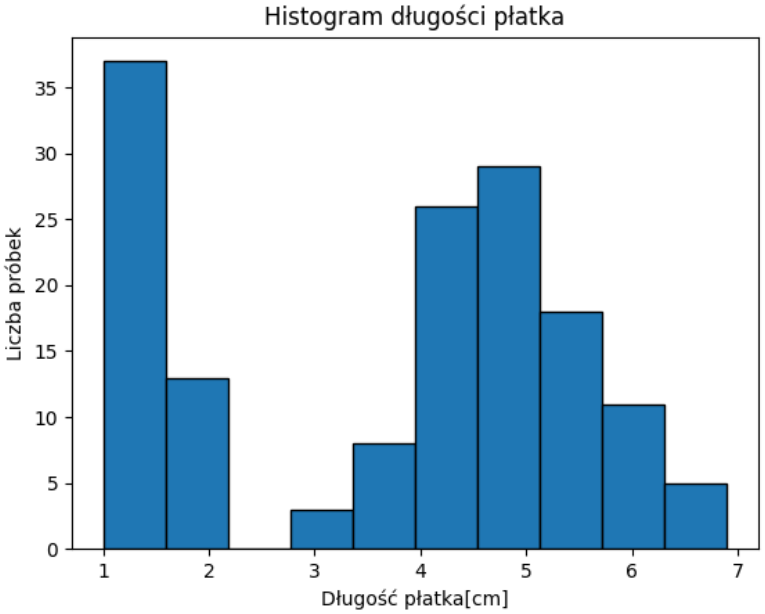
- Minimum: 2.0
- Maksimum: 4.4
- Średnia: 3.05
- Odchylenie standardowe: 0.43



Rys. 2.2 Histogram szerokości kielicha

c) Długość płatka:

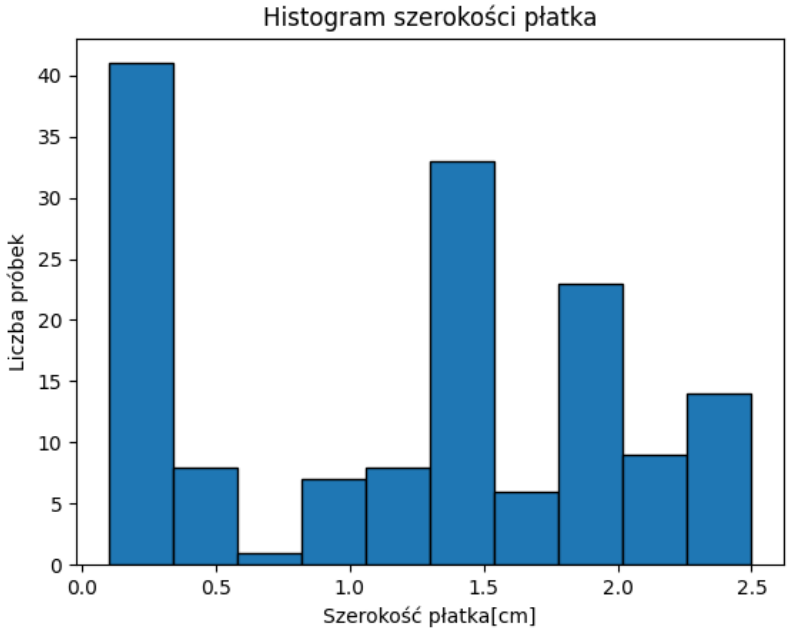
- Minimum: 1.0
- Maksimum: 6.9
- Średnia: 3.76
- Odchylenie standardowe: 1.76



Rys. 3.3 Histogram długości płatka

d) Szerokość płatka:

- Minimum: 0.1
- Maksimum: 2.5
- Średnia: 1.20
- Odchylenie standardowe: 0.76



Rys. 3.4 Histogram szerokości płatka

4. Skrypt programu

Kod programu tworzącego drzewo decyzyjne Rys. 5.3.2 oparte na optymalnych argumentach funkcji DecisionTreeClassifier. Poniższy kod uzyskuje **96.66%** trafności na danych zbioru Iris.

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
import hickle as hkl
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt

# Wczytanie danych z pliku 'iris.hkl'
x, y_t, x_norm, x_n_s, y_t_s = hkl.load('iris.hkl')
# Odejmowanie 1 od wartości w y_t
y_t -= 1
# Transpozycja macierzy x
x = x.T
# Usunięcie dodatkowych wymiarów w y_t
y_t = np.squeeze(y_t)
# Zdefiniowanie parametrów dla drzewa decyzyjnego
criterion = "entropy"
max_depth = 3
splitter = "random"
min_samples_split = 5
min_samples_leaf = 2
min_weight_fraction_leaf = 0
max_leaf_nodes = 5

# Przypisanie danych i celu
data = x
target = y_t

# Liczba podziałów w walidacji krzyżowej
CVN = 10
# Inicjalizacja walidacji krzyżowej
skfold = StratifiedKFold(n_splits=CVN)

# tworzy tablicę zerową o długości 10
PK_vec = np.zeros(CVN)

# Pętla walidacji krzyżowej
for i, (train, test) in enumerate(skfold.split(data, target),
start=0):
    x_train, x_test = data[train], data[test]
    y_train, y_test = target[train], target[test]

    # Inicjalizacja i dopasowanie modelu drzewa decyzyjnego
    decision_tree = DecisionTreeClassifier
      (
        random_state=0,
        max_depth=max_depth,
        criterion=criterion,
        splitter=splitter,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        min_weight_fraction_leaf=min_weight_fraction_leaf,
        max_leaf_nodes=max_leaf_nodes
      )
    decision_tree = decision_tree.fit(x_train, y_train)

    # Przewidywanie wyników na danych testowych
    result = decision_tree.predict(x_test)

    # Obliczanie miary PK
```

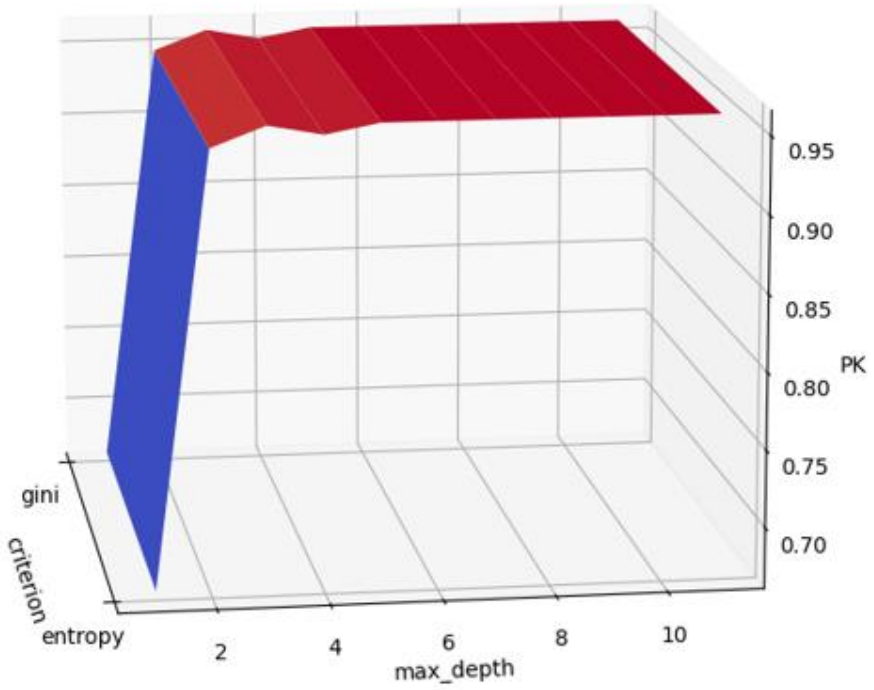
```
    n_test_samples = test.size
    PK_vec[i] = np.sum(result == y_test) / n_test_samples

# Obliczanie średniej wartości PK
PK_mean = np.mean(PK_vec)

# Wyświetlanie wartości PK
print("PK: {}".format(PK_mean))
```

5. Eksperymenty

5.1. Implementacja walidacji krzyżowej dla drzewa decyzyjnego realizowana dla par parametrów

Celem eksperymentu było wyznaczenie optymalnych wartości (Tab. 5.2.) dla argumentów funkcji DecisionTreeClassifier. Szczegółowe dane liczbowe, które zostały użyte do generowania poniższych wykresów znajdują się w dodatku 8.5.

5.1.1 Walidacja krzyżowa dla criterion oraz max_depth



Rys. 5.1.1 Wykres powierzchniowy przedstawiający zależność criterion, max_depth i PK

Przeprowadzając analizę zamieszczonych wyników, można zauważyć, że miara dokładności (PK) jest podatna na zmiany w dwóch kluczowych parametrach: criterion oraz max_depth.

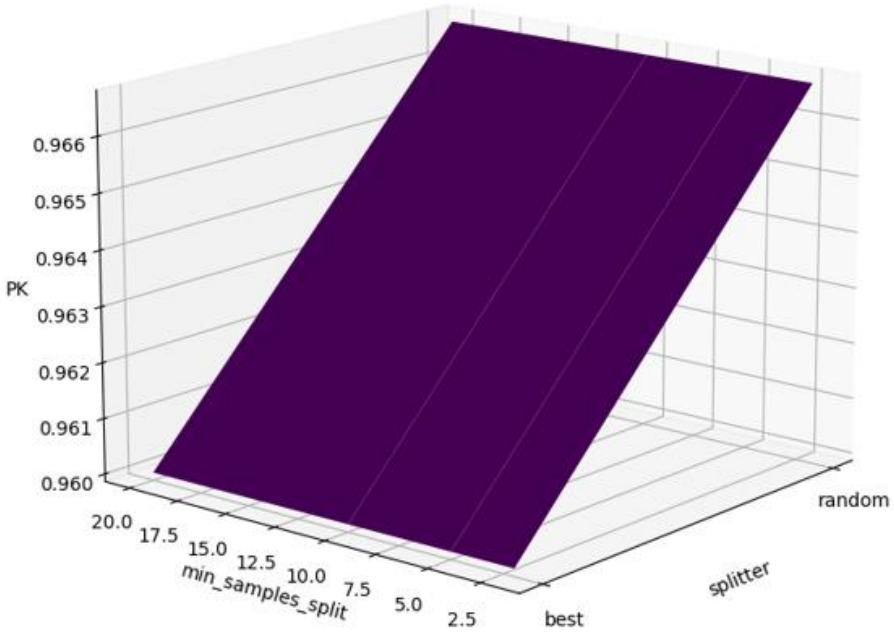
Najmniej satysfakcjonujące wyniki osiągamy dla następujących kombinacji: criterion "gini" przy max_depth równym 1, co przekłada się na PK: 0.6666. Ta wartość dokładności jest najniższa w porównaniu z innymi przypadkami.

Najlepsze wyniki odnotowujemy dla kombinacji: criterion "gini" przy max_depth równej 3, 5, 6, 7, 8, 9, 10, 11 oraz criterion "entropy" przy max_depth równej 3, 5, 6, 7, 8, 9, 10, 11. W tych przypadkach osiągamy PK: 0.96. Jest to najwyższy poziom dokładności, jaki można osiągnąć. W

zakresie max_depth od 2 do 4, niezależnie od kryterium (gini lub entropy), uzyskujemy wysoką dokładność na poziomie około 0.95-0.9533.

Warto zaznaczyć, że dla criterion "gini" i max_depth równego 1 lub 2, oraz dla criterion "entropy" i max_depth równego 1, obserwujemy niższą dokładność PK, która wynosi 0.6666. Te rezultaty wskazują, że drzewo decyzyjne o maksymalnej głębokości 1 nie jest wystarczająco złożone, by adekwatnie modelować dane.

5.1.2. Walidacja krzyżowa dla splitter oraz min_samples_split



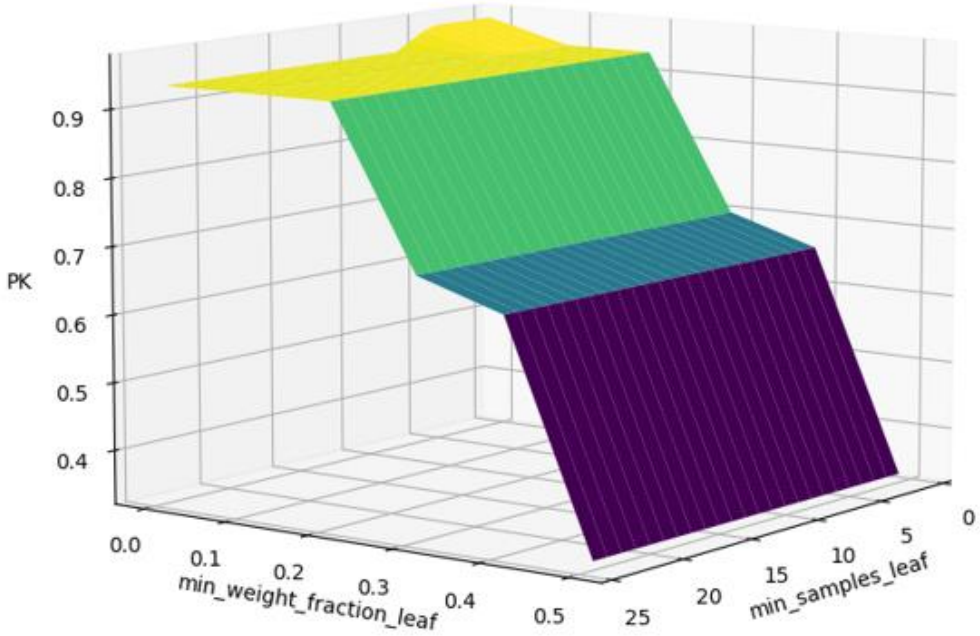
Rys. 5.1.2 Wykres powierzchniowy przedstawiający zależność splitter, min_samples_split i PK

Na podstawie powyższego wykresu powierzchniowego można wyznaczyć najwyższą wartość miary PK, która wynosi 0.9666. Ta wartość została osiągnięta dla wszystkich kombinacji parametrów splitter i min_samples_split. W przypadku tych parametrów, żadna kombinacja nie przekroczyła tej wartości, co oznacza, że model jest dość stabilny i skuteczny w przewidywaniu.

Najniższa wartość miary PK w danych jest równa 0.96. Ta wartość również została osiągnięta dla różnych kombinacji parametrów splitter i min_samples_split. Podobnie jak w przypadku najwyższej wartości, żadna kombinacja nie osiągnęła niższej wartości 0.96.

Wartości te są relatywnie bliskie i różnica między najwyższą a najniższą wartością wynosi jedynie 0.0066. To sugeruje, że testowane kombinacje parametrów były dość podobne pod względem skuteczności, a dokładność modelu była wysoka i nieznacznie się różniła.

5.1.3. Walidacja krzyżowa dla min_samples_leaf oraz min_weight_fraction_leaf



Rys. 5.1.3 Wykres powierzchniowy przedstawiający zależność min_samples_leaf, min_weight_fraction_leaf i PK

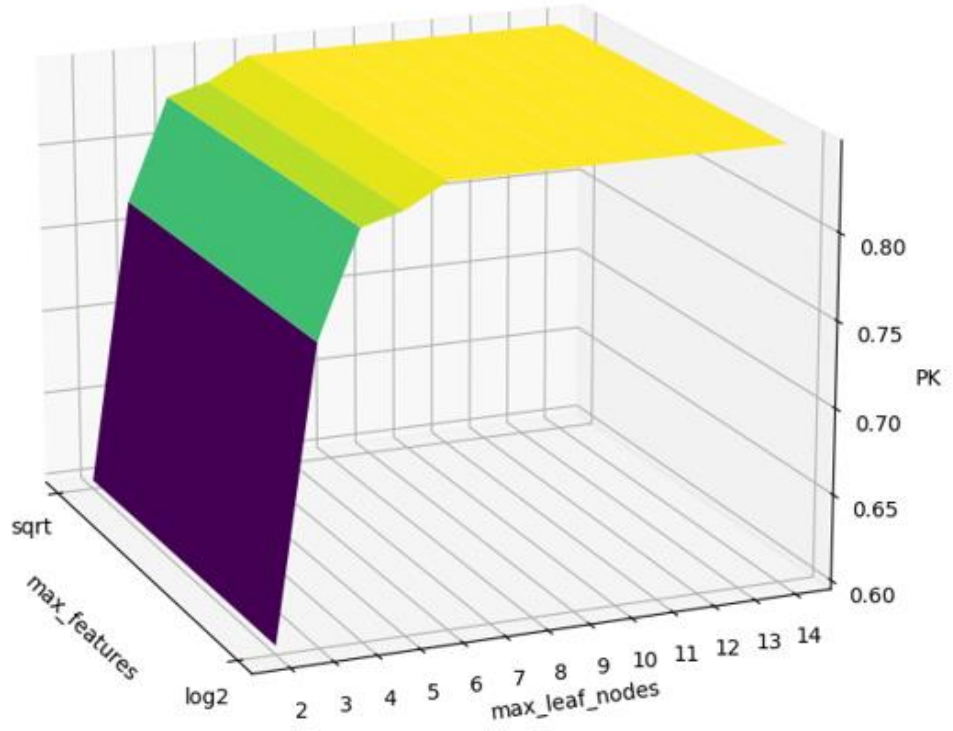
Na podstawie powyższych danych można zauważyć kilka istotnych obserwacji dotyczących miary PK dla różnych kombinacji parametrów min_samples_leaf i min_weight_fraction_leaf. Najwyższa wartość miary PK wynosi 0.9667 i została osiągnięta dla kombinacji min_samples_leaf: 1,2,3,4,5 oraz min_weight_fraction_leaf: 0.0. Wartości miary PK dla innych kombinacji tych parametrów są nieco niższe, ale wciąż utrzymują się na wysokim poziomie. Ilustruje to żółta płaszczyzna w górnej części wykreu.

Najniższa wartość miary PK wynosi 0.3333 i występuje dla kombinacji niezależnie od min_samples_leaf, natomiast dla min_weight_fraction_leaf wynoszącego 0.5. Dla pozostałych kombinacji parametrów miara PK wynosi 0.7 lub 0.6599, co potwierdza stabilność wyników.

Warto zauważyć, że zmiana wartości parametru min_samples_leaf w wyznaczonym zakresie nie ma znaczącego wpływu na wartość miary PK. Dla wartości min_samples_leaf większych niż 1, miara PK utrzymuje się stabilnie, co wskazuje na to, że minimalna liczba próbek wymaganych w liściu nie ma znaczącego wpływu na jakość modelu.

Natomiast zmiana wartości parametru min_weight_fraction_leaf ma wpływ na wartość miary PK. Im większa wartość min_weight_fraction_leaf, tym niższa wartość miary PK. Dla wartości min_weight_fraction_leaf większych niż 0.3, miara PK spada do poziomu 0.7 lub 0.66. Oznacza to, że zwiększenie powyższego parametru prowadzi do utraty informacji i pogorszenia jakości modelu.

5.1.4. Walidacja krzyżowa dla max_features oraz max_leaf_node



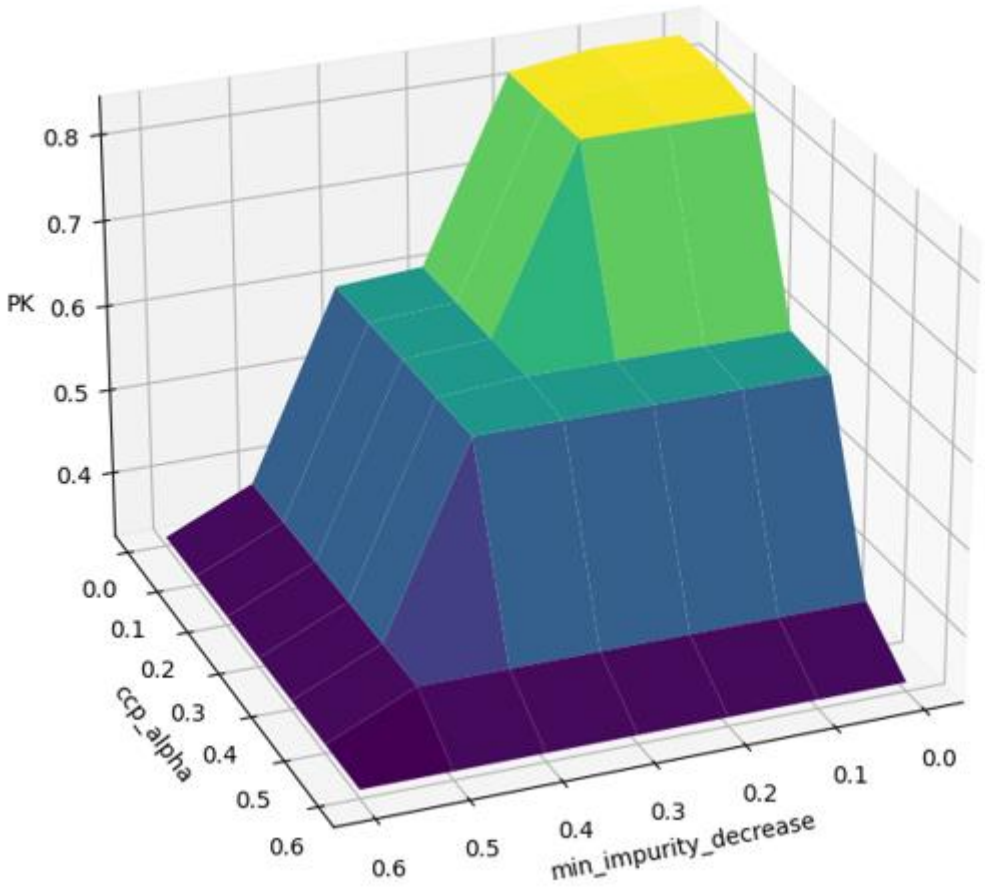
Rys. 5.1.4 Wykres powierzchniowy przedstawiający zależność, max_features, max_leaf_nodes i PK

Interpretując podane wyniki dla różnych kombinacji parametrów max_features i max_leaf_nodes, można zauważyć kilka ważnych obserwacji dotyczących miary PK.

Dla parametru max_features, który określa liczbę cech branych pod uwagę przy poszukiwaniu najlepszego podziału, zastosowano dwie opcje: sqrt oraz log2. W obu przypadkach miara PK utrzymuje się na wysokim poziomie, ponieważ w przypadku danych Iris oba parametry zwrócą tę samą wartość równa 2. Wartości miary PK wahają się w przedziale od 0.6 do 0.8467. Można zauważyć, że wyższe wartości max_leaf_nodes mają tendencję do zwiększania PK.

Analizując parametr max_leaf_nodes, który reguluje maksymalną liczbę liści w drzewie, widzimy, że wartości miary PK wzrastają wraz ze zwiększaniem tej liczby, aż osiągają stabilność w okolicy 0.8467. Oznacza to, że zwiększanie liczby maksymalnych liści pozwala na lepsze dopasowanie modelu do danych treningowych, jednak po pewnym punkcie nie przynosi już znaczących korzyści i model stabilizuje się na danym poziomie wydajności.

5.1.5. Walidacja krzyżowa dla min_impurity_decrease oraz ccp_alpha



Rys. 5.1.5 Wykres powierzchniowy przedstawiający zależność, min_impurity_decrease, ccp_alpha i PK

Badając wyniki dla różnych kombinacji parametrów min_impurity_decrease i ccp_alpha, można wyróżnić kilka istotnych obserwacji dotyczących miary PK.

Dla parametru min_impurity_decrease wartości miary PK majeją wraz ze wzrostem tego parametru. Dla wartości 0.0, miara PK wynosi 0.8333. Dla wartości od 0.3 do 0.4, miara PK utrzymuje się na poziomie niemal 0.60. Dla wartości powyżej 0.4, miara PK znacząco spada do 0.33. Można zauważyć, że badane parametry nie równoważą się, kiedy zwiększymy min_impurity_decrease to niezależnie od ccp_alpha wartość PK spadnie.

Wraz ze wzrostem ccp_alpha wartości miary PK maleją dla wszystkich wartości min_impurity_decrease. Najwyższa wartość miary PK 0.8333 osiągana jest dla wartości bliskich 0 dla ccp_alpha, również dla min_impurity_decrease wartości bliskie 0 są najkorzystniejsze dla wartości PK. Podsumowując dla wybranego zbioru danych optymalną wartością analizowanych parametrów używanych do tworzenia drzewa jest 0.

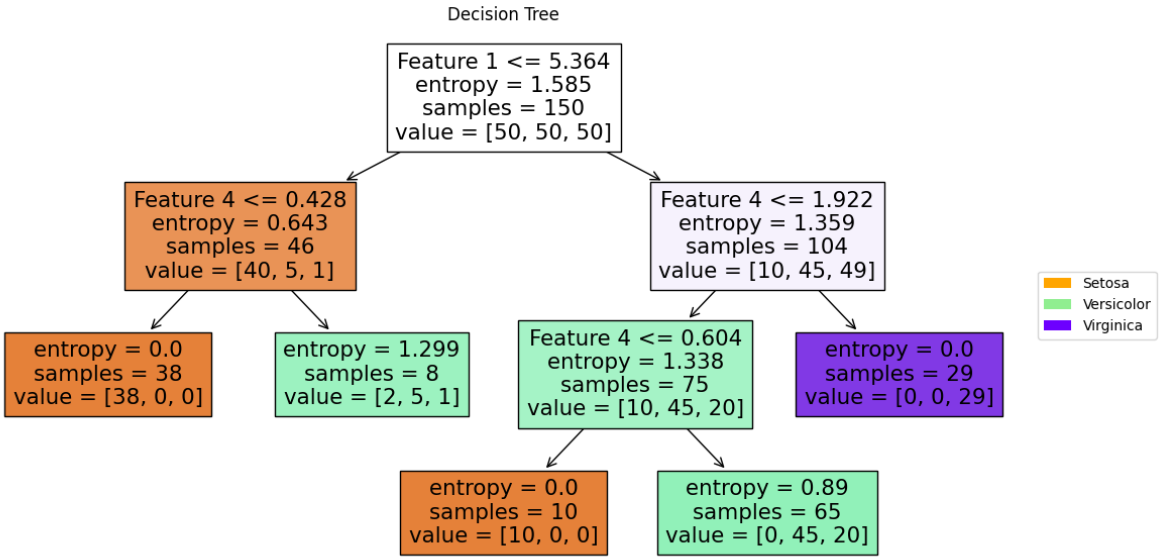
5.2. Wyznaczenie optymalnych wartości dla argumentów funkcji DecisionTreeClassifier

Przeanalizowanie wyników walidacji krzyżowej doprowadziło do otrzymania optymalnych[1] parametrów dla algorytmu tworzącego drzewo decyzyjne.

Trafność klasyfikatora utworzonego na poniższych wartościach wynosi 84,6666%

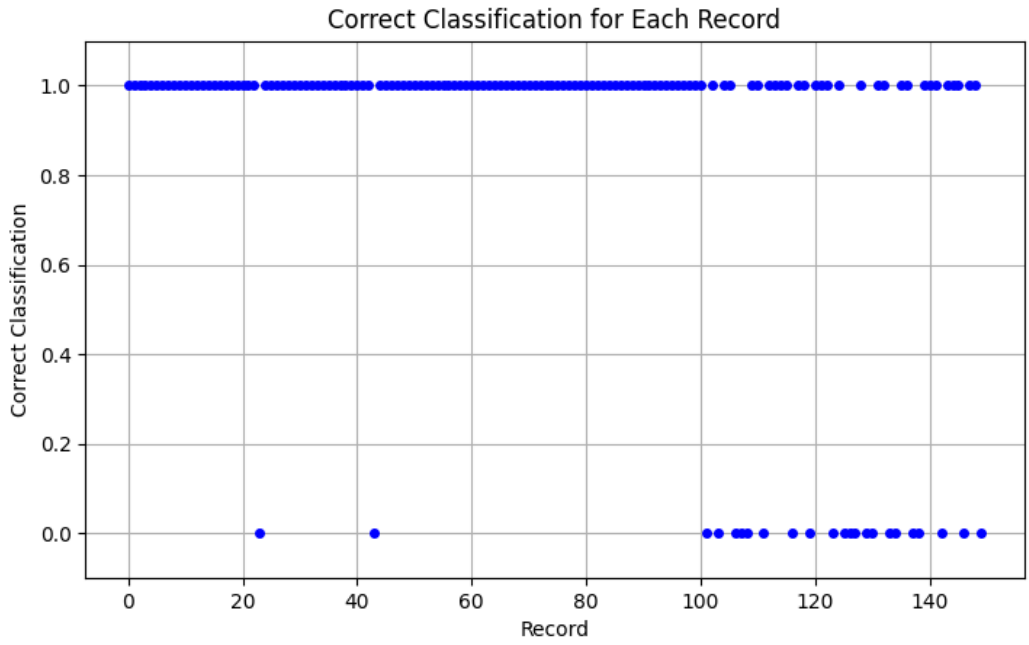
Tab. 5.2 Wybrane wartości parametrów na podstawie przeprowadzonej walidacji krzyżowej

criterion	entropy
max_depth	3
splitter	random
min_samples_split	5
min_samples_leaf	2
min_weight_fraction_leaf	0
max_features	log2
max_leaf_nodes	5
min_impurity_decrease	0
ccp_alpha	0



Rys. 5.2.1 Schemat drzewa decyzyjnego otrzymanego w eksperymencie 5.1

[1] Optymalne parametry tylko dla eksperymentu 5.1, po pewnych zamianach uzyskuje się wyższą trafność.



Rys. 5.2.2 Poprawność klasyfikacji dla każdego rekordu

Zostało zauważone, w eksperymencie oznaczonym jako 5.1.4. parametr max_features przyjmujący wartości log2 lub sqrt powoduje około 12 punktów procentowych straty trafności.

5.3 Korekta optymalnych wartości dla argumentów funkcji DecisionTreeClassifier

Okazuje się, że parametr max_features może przyjmować jeszcze jedną wartość jaką jest None, która jest domyślną wartością argumentu funkcji. Kiedy argument został ręcznie ustawiony na None, biblioteka matplotlib.pyplot jest w stanie generować wykresy.

max_features : *int, float or {"auto", "sqrt", "log2"}, default=None*

The number of features to consider when looking for the best split:

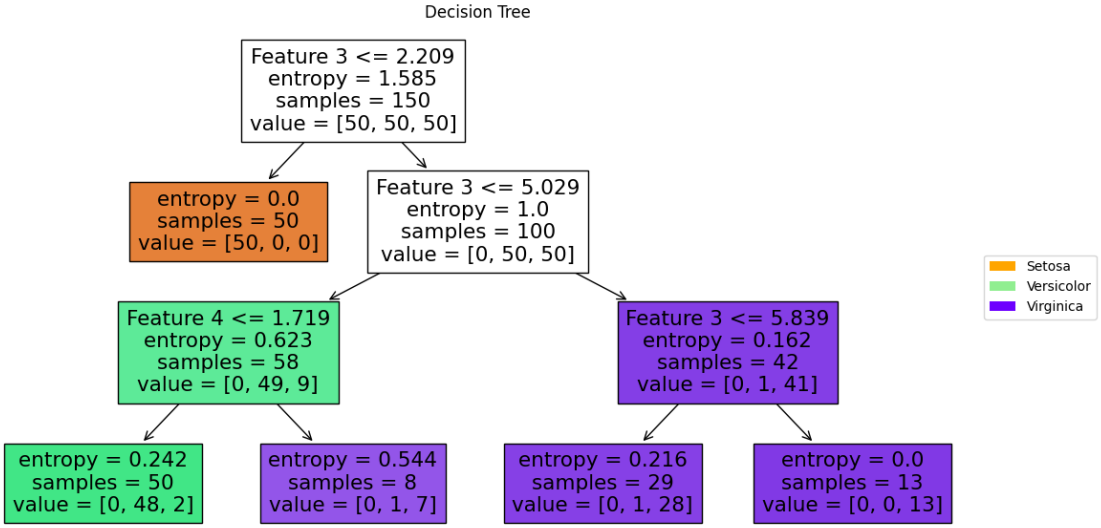
- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `max(1, int(max_features * n_features_in_))` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Rys. 5.3.1 Wycinek z dokumentacji opisujący jakie wartości może przyjąć parametr max_features

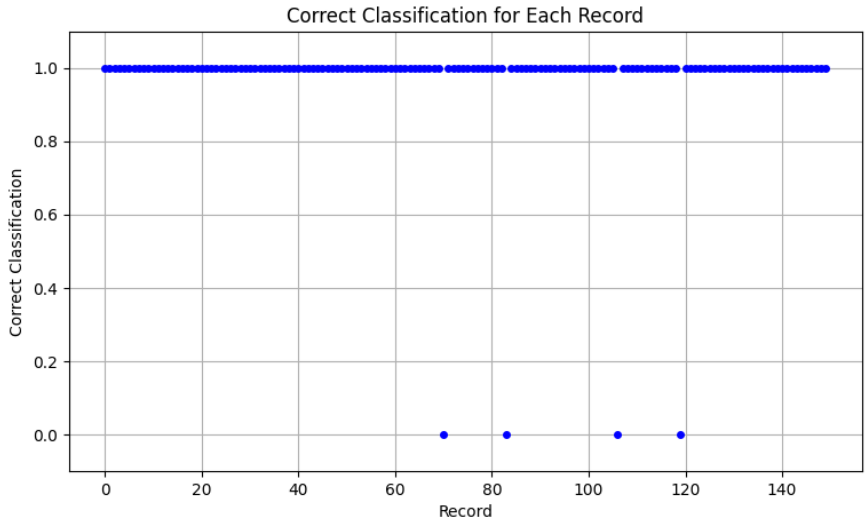
Otrzymanie trafności na poziomie **96.66%** wymaga usunięcia inicjalizacji parametru max_features, w takim przypadku funkcja użyje domyślnej wartości None oraz będzie możliwość utworzenia wykresu.

Tab. 5.3 Wartości parametrów po optymalizacji

criterion	entropy
max_depth	3
splitter	random
min_samples_split	5
min_samples_leaf	2
min_weight_fraction_leaf	0
max_features	**None (domyślny)**
max_leaf_nodes	5
min_impurity_decrease	0
ccp_alpha	0



Rys. 5.3.2 Schemat drzewa decyzyjnego po ustawieniu max_features na wartość None



Rys. 5.3.3 Poprawność klasyfikacji dla każdego rekordu po ustawieniu max_features na wartość None

6. Podsumowanie i wnioski

W ramach projektu przeprowadzono proces przygotowania danych, tworzenia modelu drzewa decyzyjnego, walidacji krzyżowej i optymalizacji parametrów w celu klasyfikacji gatunków irysów na podstawie ich cech.

W pierwszym etapie, dane zostały wczytane z pliku tekstowego i poddane odpowiednim operacjom przetwarzania. W celu uzyskania wartości numerycznych zaczynających się od zera, etykiety klas zostały przeskalowane poprzez odjęcie jedynki. Ponadto, dane zostały przekształcone w taki sposób, aby cechy były reprezentowane jako wiersze, a próbki jako kolumny. Taka reprezentacja danych jest wymagana przez model drzewa decyzyjnego.

Następnie, skorzystano z klasy DecisionTreeClassifier dostępnej w bibliotece scikit-learn, aby utworzyć model drzewa decyzyjnego. W celu zoptymalizowania wydajności modelu, dostosowano różne parametry, takie jak maksymalna głębokość drzewa (max_depth), minimalna liczba próbek wymagana do podziału w węźle (min_samples_split), minimalna liczba próbek wymagana w liściu (min_samples_leaf), itd. Dobór optymalnych parametrów ma na celu zwiększenie trafności klasyfikacji modelu.

Aby ocenić wydajność modelu na różnych podzbiorach danych, zastosowano technikę walidacji krzyżowej. Zastosowano StratifiedKFold z 10 podziałami, co zapewnia równomierne rozłożenie klas w każdym podzbiorze. Dla każdego podziału, model został trenowany na zbiorze treningowym, a następnie oceniany na zbiorze testowym. Ostatecznie, przeprowadzono optymalizację parametrów modelu drzewa decyzyjnego. Testowano różne kombinacje parametrów, aż osiągnięto najwyższą trafność klasyfikacji. Po ustawieniu parametru max_features na wartość domyślną osiągnięto wynik trafności na poziomie 96% wskazuje na dobrą zdolność modelu do poprawnego rozpoznawania gatunków irysów na podstawie ich cech.

Podsumowując, przeprowadzony projekt skupiał się na analizie danych irysów, tworzeniu i optymalizacji modelu drzewa decyzyjnego oraz ocenie jego wydajności przy użyciu walidacji krzyżowej. Uzyskane wyniki wskazują na potencjał drzewa decyzyjnego w klasyfikacji gatunków irysów na podstawie ich cech.

7. Bibliografia

Cichosz, P. (2000). *Systemy uczące się*. Wydawnictwo Naukowo-Techniczne.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Education.

https://mfiles.pl/pl/index.php/Drzewo_decyzyjne

https://gdudek.el.pcz.pl/files/SUS/SUS_wyklad3.pdf

Dokumentacja biblioteki scikit-learn
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

8. Dodatki

8.1. Kod przeprowadzający walidację krzyżowa dla criterion oraz max_depth

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
import hickle as hkl
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt

x, y_t, x_norm, x_n_s, y_t_s = hkl.load('iris.hkl')
y_t -= 1
x = x.T
y_t = np.squeeze(y_t)

criterion_vec = ["gini", "entropy"]
criterion_num_vec = np.array(range(len(criterion_vec)))
for ind_criterion in range(len(criterion_vec)):
    print(criterion_vec[ind_criterion])

max_depth_vec = np.array(range(1, 12, 1))
data = x
target = y_t
CVN = 10
skfold = StratifiedKFold(n_splits=CVN)
PK_cr_md_vec = np.zeros([len(criterion_vec), len(max_depth_vec)])
for criterion_ind in range(len(criterion_vec)):
    for max_depth_ind in range(len(max_depth_vec)):
        PK_vec = np.zeros(CVN)
        for i, (train, test) in enumerate(skfold.split(data, target),
start=0):
            x_train, x_test = data[train], data[test]
            y_train, y_test = target[train], target[test]

            decision_tree = DecisionTreeClassifier(random_state=0, \

max_depth=max_depth_vec[max_depth_ind], \

criterion=criterion_vec[criterion_ind])
            decision_tree = decision_tree.fit(x_train, y_train)
            result = decision_tree.predict(x_test)

            n_test_samples = test.size
            PK_vec[i] = np.sum(result == y_test) / n_test_samples

        PK_cr_md_vec[criterion_ind, max_depth_ind] = np.mean(PK_vec)
        print("criterion: {} | max_depth: {} | PK:
{}".format(criterion_vec[criterion_ind], \

max_depth_vec[max_depth_ind], \
PK_cr_md_vec[criterion_ind, max_depth_ind]))
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
X, Y = np.meshgrid(criterion_num_vec, max_depth_vec)
surf = ax.plot_surface(X, Y, PK_cr_md_vec.T, cmap='coolwarm')
ax.set_xlabel('criterion')
ax.set_xticks(np.array(range(len(criterion_vec))))
ax.set_xticklabels(criterion_vec)
ax.set_ylabel('max_depth')
ax.set_zlabel('PK')
ax.view_init(30, 200)
plt.show()
```

8.2. Kod tworzący histogramy

```
import numpy as np
import matplotlib.pyplot as plt

# Wczytanie danych z pliku txt
data = np.genfromtxt('histogramy.txt', delimiter=',')

# Podział danych na poszczególne cechy
sepal_length = data[:, 0]
sepal_width = data[:, 1]
petal_length = data[:, 2]
petal_width = data[:, 3]

# Histogram długości kielicha
plt.hist(sepal_length, bins=10, edgecolor='black')
plt.xlabel('Długość kielicha[cm]')
plt.ylabel('Liczba próbek')
plt.title('Histogram długości kielicha')
plt.show()

# Histogram szerokości kielicha
plt.hist(sepal_width, bins=10, edgecolor='black')
plt.xlabel('Szerokość kielicha[cm]')
plt.ylabel('Liczba próbek')
plt.title('Histogram szerokości kielicha')
plt.show()

# Histogram długości płatka
plt.hist(petal_length, bins=10, edgecolor='black')
plt.xlabel('Długość płatka[cm]')
plt.ylabel('Liczba próbek')
plt.title('Histogram długości płatka')
plt.show()

# Histogram szerokości płatka
plt.hist(petal_width, bins=10, edgecolor='black')
plt.xlabel('Szerokość płatka[cm]')
plt.ylabel('Liczba próbek')
plt.title('Histogram szerokości płatka')
plt.show()
```

8.3 Kod tworzący schemat drzewa decyzyjnego z legenda

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
import hickle as hkl
import matplotlib.pyplot as plt

x, y_t, x_norm, x_n_s, y_t_s = hkl.load('iris.hkl')
y_t -= 1
x = x.T
y_t = np.squeeze(y_t)

criterion = "entropy"
max_depth = 3
splitter = "random"
min_samples_split = 5
min_samples_leaf = 2
min_weight_fraction_leaf = 0
max_features = "log2"
max_leaf_nodes = 5
min_impurity_decrease = 0
ccp_alpha = 0
```

```
decision_tree = DecisionTreeClassifier(random_state=0,
max_depth=max_depth,
                                       criterion=criterion,
                                       splitter=splitter,

min_samples_split=min_samples_split,

min_samples_leaf=min_samples_leaf,

min_weight_fraction_leaf=min_weight_fraction_leaf,
                                       max_features=max_features,
                                       max_leaf_nodes=max_leaf_nodes,

min_impurity_decrease=min_impurity_decrease,
                                       ccp_alpha=ccp_alpha)
decision_tree.fit(x, y_t)

# Plotting the decision tree
plt.figure(figsize=(12, 6))
plot_tree(decision_tree, feature_names=["Feature 1", "Feature 2",
"Feature 3", "Feature 4"], filled=True)

# Create legend
class_names = ["Setosa", "Versicolor", "Virginica"]
class_colors = ["orange", "lightgreen", "#6d00ff"]
patches = [plt.Rectangle((0, 0), 1, 1, fc=color) for color in
class_colors]
plt.legend(patches, class_names, loc="center left", bbox_to_anchor=(1,
0.5))

plt.title("Decision Tree")
plt.tight_layout()
plt.show()
```

8.4. Kod przedstawiający wykres trafności klasyfikatora

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
import hickle as hkl
import matplotlib.pyplot as plt

x, y_t, x_norm, x_n_s, y_t_s = hkl.load('iris.hkl')
y_t -= 1
x = x.T
y_t = np.squeeze(y_t)

criterion = "entropy"
max_depth = 3
splitter = "random"
min_samples_split = 5
min_samples_leaf = 2
min_weight_fraction_leaf = 0
max_features = "log2"
max_leaf_nodes = 5
min_impurity_decrease = 0
ccp_alpha = 0

decision_tree = DecisionTreeClassifier(random_state=0,
max_depth=max_depth,
                                       criterion=criterion,
                                       splitter=splitter,

min_samples_split=min_samples_split,
```

```
min_samples_leaf=min_samples_leaf,

min_weight_fraction_leaf=min_weight_fraction_leaf,
                                        max_features=max_features,
                                        max_leaf_nodes=max_leaf_nodes,

min_impurity_decrease=min_impurity_decrease,
                                        ccp_alpha=ccp_alpha)
decision_tree.fit(x, y_t)
result = decision_tree.predict(x)

classification_vector = (result == y_t).astype(int)

plt.figure()
plt.plot(range(len(classification_vector)), classification_vector,
'bo', markersize=4)
plt.xlabel('Record')
plt.ylabel('Correct Classification')
plt.title('Correct Classification for Each Record')
plt.ylim([-0.1, 1.1])
plt.grid(True)
plt.show()
```

8.5. Szczegółowe wyniki eksperymentu 5.1.

```
criterion: gini | max_depth: 1 | PK: 0.6666666666666667
criterion: gini | max_depth: 2 | PK: 0.9466666666666667
criterion: gini | max_depth: 3 | PK: 0.96
criterion: gini | max_depth: 4 | PK: 0.9533333333333334
criterion: gini | max_depth: 5 | PK: 0.96
criterion: gini | max_depth: 6 | PK: 0.96
criterion: gini | max_depth: 7 | PK: 0.96
criterion: gini | max_depth: 8 | PK: 0.96
criterion: gini | max_depth: 9 | PK: 0.96
criterion: gini | max_depth: 10 | PK: 0.96
criterion: gini | max_depth: 11 | PK: 0.96
criterion: entropy | max_depth: 1 | PK: 0.6666666666666667
criterion: entropy | max_depth: 2 | PK: 0.9466666666666667
criterion: entropy | max_depth: 3 | PK: 0.96
criterion: entropy | max_depth: 4 | PK: 0.9533333333333334
criterion: entropy | max_depth: 5 | PK: 0.96
criterion: entropy | max_depth: 6 | PK: 0.96
criterion: entropy | max_depth: 7 | PK: 0.96
criterion: entropy | max_depth: 8 | PK: 0.96
criterion: entropy | max_depth: 9 | PK: 0.96
criterion: entropy | max_depth: 10 | PK: 0.96
criterion: entropy | max_depth: 11 | PK: 0.96
```

Tab. 8.5.1 Wynik przestawiający poprawność klasyfikacji dla eksperymentu 5.1.1.

```
splitter: best | min_samples_split: 2 | PK: 0.96
splitter: best | min_samples_split: 5 | PK: 0.96
splitter: best | min_samples_split: 10 | PK: 0.96
splitter: best | min_samples_split: 20 | PK: 0.96
splitter: random | min_samples_split: 2 | PK: 0.9666666666666668
splitter: random | min_samples_split: 5 | PK: 0.9666666666666668
splitter: random | min_samples_split: 10 | PK: 0.9666666666666668
splitter: random | min_samples_split: 20 | PK: 0.9666666666666668
```

Tab. 8.5.2 Wynik przestawiający poprawność klasyfikacji dla eksperymentu 5.1.2.

```
min_samples_leaf: 1 | min_weight_fraction_leaf: 0.0 | PK: 0.9666666666666668
min_samples_leaf: 1 | min_weight_fraction_leaf: 0.1 | PK: 0.9333333333333333
min_samples_leaf: 1 | min_weight_fraction_leaf: 0.2 | PK: 0.9333333333333333
min_samples_leaf: 1 | min_weight_fraction_leaf: 0.3 | PK: 0.7
min_samples_leaf: 1 | min_weight_fraction_leaf: 0.4 | PK: 0.6599999999999999
min_samples_leaf: 1 | min_weight_fraction_leaf: 0.5 | PK: 0.33333333333333337
min_samples_leaf: 2 | min_weight_fraction_leaf: 0.0 | PK: 0.9666666666666668
min_samples_leaf: 2 | min_weight_fraction_leaf: 0.1 | PK: 0.9333333333333333
min_samples_leaf: 2 | min_weight_fraction_leaf: 0.2 | PK: 0.9333333333333333
min_samples_leaf: 2 | min_weight_fraction_leaf: 0.3 | PK: 0.7
min_samples_leaf: 2 | min_weight_fraction_leaf: 0.4 | PK: 0.6599999999999999
min_samples_leaf: 2 | min_weight_fraction_leaf: 0.5 | PK: 0.33333333333333337
min_samples_leaf: 3 | min_weight_fraction_leaf: 0.0 | PK: 0.9666666666666668
min_samples_leaf: 3 | min_weight_fraction_leaf: 0.1 | PK: 0.9333333333333333
min_samples_leaf: 3 | min_weight_fraction_leaf: 0.2 | PK: 0.9333333333333333
min_samples_leaf: 3 | min_weight_fraction_leaf: 0.3 | PK: 0.7
min_samples_leaf: 3 | min_weight_fraction_leaf: 0.4 | PK: 0.6599999999999999
min_samples_leaf: 3 | min_weight_fraction_leaf: 0.5 | PK: 0.33333333333333337
min_samples_leaf: 4 | min_weight_fraction_leaf: 0.0 | PK: 0.9666666666666668
min_samples_leaf: 4 | min_weight_fraction_leaf: 0.1 | PK: 0.9333333333333333
min_samples_leaf: 4 | min_weight_fraction_leaf: 0.2 | PK: 0.9333333333333333
min_samples_leaf: 4 | min_weight_fraction_leaf: 0.3 | PK: 0.7
min_samples_leaf: 4 | min_weight_fraction_leaf: 0.4 | PK: 0.6599999999999999
min_samples_leaf: 4 | min_weight_fraction_leaf: 0.5 | PK: 0.33333333333333337
min_samples_leaf: 5 | min_weight_fraction_leaf: 0.0 | PK: 0.9666666666666668
min_samples_leaf: 5 | min_weight_fraction_leaf: 0.1 | PK: 0.9333333333333333
min_samples_leaf: 5 | min_weight_fraction_leaf: 0.2 | PK: 0.9333333333333333
min_samples_leaf: 5 | min_weight_fraction_leaf: 0.3 | PK: 0.7
min_samples_leaf: 5 | min_weight_fraction_leaf: 0.4 | PK: 0.6599999999999999
min_samples_leaf: 5 | min_weight_fraction_leaf: 0.5 | PK: 0.33333333333333337
min_samples_leaf: 6 | min_weight_fraction_leaf: 0.0 | PK: 0.9600000000000002
min_samples_leaf: 6 | min_weight_fraction_leaf: 0.1 | PK: 0.9333333333333333
min_samples_leaf: 6 | min_weight_fraction_leaf: 0.2 | PK: 0.9333333333333333
min_samples_leaf: 6 | min_weight_fraction_leaf: 0.3 | PK: 0.7
min_samples_leaf: 6 | min_weight_fraction_leaf: 0.4 | PK: 0.6599999999999999
min_samples_leaf: 6 | min_weight_fraction_leaf: 0.5 | PK: 0.33333333333333337
```

Tab. 8.5.3 Wynik przestawiający poprawność klasyfikacji dla eksperymentu 5.1.3.

```
max_features: sqrt | max_leaf_nodes: 2 | PK: 0.6
max_features: sqrt | max_leaf_nodes: 3 | PK: 0.7666666666666667
max_features: sqrt | max_leaf_nodes: 4 | PK: 0.8266666666666668
max_features: sqrt | max_leaf_nodes: 5 | PK: 0.8333333333333334
max_features: sqrt | max_leaf_nodes: 6 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 7 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 8 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 9 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 10 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 11 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 12 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 13 | PK: 0.8466666666666667
max_features: sqrt | max_leaf_nodes: 14 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 2 | PK: 0.6
max_features: log2 | max_leaf_nodes: 3 | PK: 0.7666666666666667
max_features: log2 | max_leaf_nodes: 4 | PK: 0.8266666666666668
max_features: log2 | max_leaf_nodes: 5 | PK: 0.8333333333333334
max_features: log2 | max_leaf_nodes: 6 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 7 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 8 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 9 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 10 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 11 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 12 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 13 | PK: 0.8466666666666667
max_features: log2 | max_leaf_nodes: 14 | PK: 0.8466666666666667
```

Tab. 8.5.4 Wynik przestawiający poprawność klasyfikacji dla eksperymentu 5.1.4.

```
min_impurity_decrease: 0.0 | ccp_alpha: 0.0 | PK: 0.8333333333333334
min_impurity_decrease: 0.0 | ccp_alpha: 0.1 | PK: 0.8333333333333334
min_impurity_decrease: 0.0 | ccp_alpha: 0.2 | PK: 0.8200000000000001
min_impurity_decrease: 0.0 | ccp_alpha: 0.3 | PK: 0.6066666666666667
min_impurity_decrease: 0.0 | ccp_alpha: 0.4 | PK: 0.6
min_impurity_decrease: 0.0 | ccp_alpha: 0.5 | PK: 0.38
min_impurity_decrease: 0.0 | ccp_alpha: 0.6 | PK: 0.33333333333333337
min_impurity_decrease: 0.1 | ccp_alpha: 0.0 | PK: 0.8333333333333334
min_impurity_decrease: 0.1 | ccp_alpha: 0.1 | PK: 0.8333333333333334
min_impurity_decrease: 0.1 | ccp_alpha: 0.2 | PK: 0.8200000000000001
min_impurity_decrease: 0.1 | ccp_alpha: 0.3 | PK: 0.6066666666666667
min_impurity_decrease: 0.1 | ccp_alpha: 0.4 | PK: 0.6
min_impurity_decrease: 0.1 | ccp_alpha: 0.5 | PK: 0.38
min_impurity_decrease: 0.1 | ccp_alpha: 0.6 | PK: 0.33333333333333337
min_impurity_decrease: 0.2 | ccp_alpha: 0.0 | PK: 0.8200000000000001
min_impurity_decrease: 0.2 | ccp_alpha: 0.1 | PK: 0.8200000000000001
min_impurity_decrease: 0.2 | ccp_alpha: 0.2 | PK: 0.8200000000000001
min_impurity_decrease: 0.2 | ccp_alpha: 0.3 | PK: 0.6066666666666667
min_impurity_decrease: 0.2 | ccp_alpha: 0.4 | PK: 0.6
min_impurity_decrease: 0.2 | ccp_alpha: 0.5 | PK: 0.38
min_impurity_decrease: 0.2 | ccp_alpha: 0.6 | PK: 0.33333333333333337
min_impurity_decrease: 0.3 | ccp_alpha: 0.0 | PK: 0.6066666666666667
min_impurity_decrease: 0.3 | ccp_alpha: 0.1 | PK: 0.6066666666666667
min_impurity_decrease: 0.3 | ccp_alpha: 0.2 | PK: 0.6066666666666667
min_impurity_decrease: 0.3 | ccp_alpha: 0.3 | PK: 0.6066666666666667
min_impurity_decrease: 0.3 | ccp_alpha: 0.4 | PK: 0.6
min_impurity_decrease: 0.3 | ccp_alpha: 0.5 | PK: 0.38
min_impurity_decrease: 0.3 | ccp_alpha: 0.6 | PK: 0.33333333333333337
min_impurity_decrease: 0.4 | ccp_alpha: 0.0 | PK: 0.6
min_impurity_decrease: 0.4 | ccp_alpha: 0.1 | PK: 0.6
min_impurity_decrease: 0.4 | ccp_alpha: 0.2 | PK: 0.6
min_impurity_decrease: 0.4 | ccp_alpha: 0.3 | PK: 0.6
min_impurity_decrease: 0.4 | ccp_alpha: 0.4 | PK: 0.6
min_impurity_decrease: 0.4 | ccp_alpha: 0.5 | PK: 0.38
min_impurity_decrease: 0.4 | ccp_alpha: 0.6 | PK: 0.33333333333333337
min_impurity_decrease: 0.5 | ccp_alpha: 0.0 | PK: 0.38
min_impurity_decrease: 0.5 | ccp_alpha: 0.1 | PK: 0.38
min_impurity_decrease: 0.5 | ccp_alpha: 0.2 | PK: 0.38
min_impurity_decrease: 0.5 | ccp_alpha: 0.3 | PK: 0.38
min_impurity_decrease: 0.5 | ccp_alpha: 0.4 | PK: 0.38
min_impurity_decrease: 0.5 | ccp_alpha: 0.5 | PK: 0.38
```

Tab. 8.5.5 Wynik przestawiający poprawność klasyfikacji dla eksperymentu 5.1.5.