

**Mateusz Pazdan**

Realizacja algorytmu drzewa decyzyjnego oraz badanie wpływu metaparametrów na zmiany poprawności kwalifikacji.

**Praca projektowa z modułu Sztuczna Inteligencja**

Rzeszów, 2024

Spis treści

[1. Wstęp 2](#_Toc168329492)

[1.1. Cel projektu 2](#_Toc168329493)

[1.2. Specyfikacja danych 2](#_Toc168329494)

[1.3. Przygotowanie danych 3](#_Toc168329495)

[2. Zagadnienia teoretyczne 5](#_Toc168329496)

[2.1. Drzewo decyzyjne 5](#_Toc168329497)

[2.2. Kryterium wyboru testu 7](#_Toc168329498)

[2.3. Poprawność klasyfikacji 7](#_Toc168329499)

[2.4. Walidacja krzyzowa 7](#_Toc168329500)

[3. Implementacja drzewa decyzyjnego 9](#_Toc168329501)

[3.1. Przygotowanie danych 9](#_Toc168329502)

[3.2. Eksperymenty z drzewem decyzyjnym 11](#_Toc168329503)

[3.2.1. Eksperyment 1 11](#_Toc168329504)

[3.2.2. Ekpseryment 2 14](#_Toc168329505)

[3.2.3. Eksperyment 3 17](#_Toc168329506)

[3.2.4. Eksperyment 4 20](#_Toc168329507)

[3.2.5. Eksperyment 5 23](#_Toc168329508)

[4. Wnioski 26](#_Toc168329509)

[5. Bibliografia 27](#_Toc168329510)

# Wstęp

## Cel projektu

Celem tego projektu jest ocena efektywności drzewa decyzyjnego w klasyfikacji danych medycznych związanych z przetrwaniem pacjentów po operacjach raka piersi, poprzez badanie wpływu różnych metaparametrów na poprawność klasyfikacji. W ramach projektu zbadano wpływ niżej wymienonych metaparametrów.

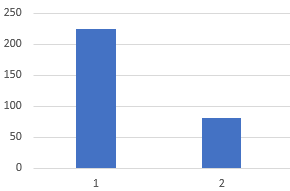
* Kryterium podziału (‘Criterion’)
* Maksymalna głębokość drzewa (‘max\_depth’)
* Metaparamter Splitter – strategia wyboru podziału węzła (‘Splitter’)
* Minimalna liczba próbek wymagana do podziału węzła (‘min\_samples\_split’)
* Minimalna liczba próbek wymagana do utworzenia liścia (‘min\_samples\_leaf’)
* Minimalna ważona frakcja sumy wag próbek wymagana do utworzenia liścia (‘min\_weight\_fraction\_leaf’)
* Maksymalna liczba cech branych pod uwagę (‘max\_features’)
* Maksymalna liczba liści (max\_leaf\_nodes’)
* Minimalne zmniejszenie nieczynności wymagane do podziału węzła (‘min\_impurity\_decrease’)
* Minimalna złożoność kosztów cięcia (‘ccp\_alpha’)

## Specyfikacja danych

W Projekcie wykorzystamy zbiór danych *Haberman's Survival* zawierający informacje dotyczące pacjentów, którzy przeszli operację raka piersi w szpitalu Uniwersytetu Świętej Franciszki w Chicago w latach 1958-1970. Podanych zbiór składa się z 306 rekordów opisancyh 4 cechami:

* *age* - Wiek pacjenta w momencie operacji.
* *Operation\_year*- Rok operacji w przedziale 1958 do 1970 jako 2 ostatnie cyfry.
* *Number of positive axillary nodes detected* - Liczba zarejestrowanych węzłów chłonnych.
* *Surtival status* - Status przeżycia po operacji, gdzie 1 (pacjent żył ponad 5 lat po operacji), a 2 (pacjent zmarł w ciągu 5 lat po operacji).

Układ wierszowy, co oznacza, że każda próbka jest reprezentowana przez wiersz, a każda cecha jest reprezentowana przez kolumne. Jako parametry przujmuje się kolumny 1-3. Kolumna 4 zawiera prawdziwą wartość klasyfikacji.



Rysunek 1 Ilość poszczególnych klasyfikatorów w zbiorze Haberman

## Przygotowanie danych

W zbiorze Haberman podczas przygotowywania danych nie napotkano na błędne rekordy lub brakuące dane. Ujednolicono dane przeprowadzając normalizację, według wzoru:

- Znormalizowana wartość cechy

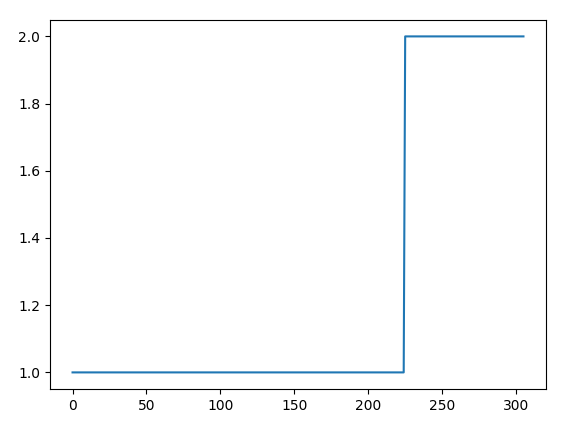
- maksymalna wartość znormalizowanej cechy (1)

- minimalna wartość znormalizowanej cechy (-1)

- maksymalna wartość cechy nieznormalizowana

- minimalna wartość cechy nieznormalizowana

Przed normalizacją wartości cechy 1 w przedziale [30, 83], cechy 2 w przedziale [58, 69], dla 3 cechy w przedziale [0, 52]. Dzięki normalizacji otrzymaliśmy znormalizowane wartości cech w przedziale od -1 do 1. Na koniec dane poddano sortowaniu, zostały one ułożone względem identyfikatorów statusu przeżycia, najpierw te z wartością „1”, a pózniej z wartością „2”, co przedstawia poniższy wykres.

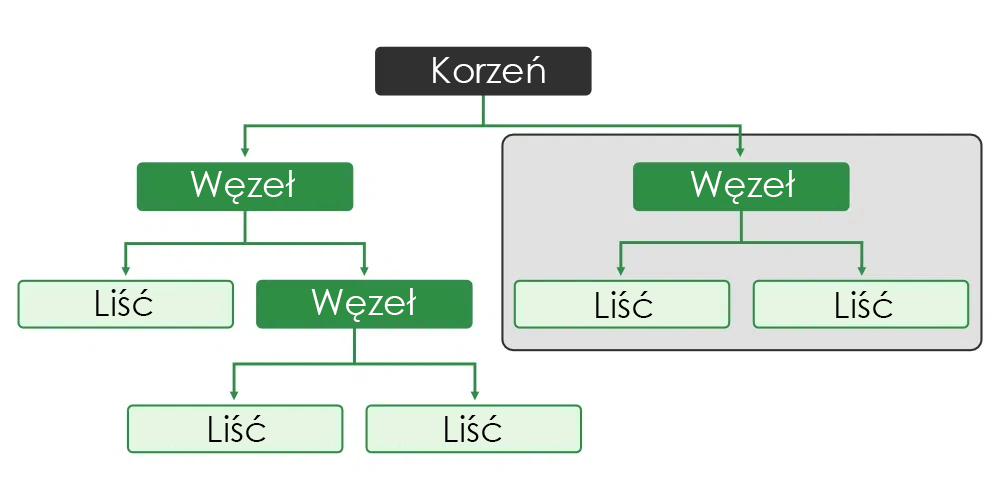


Rysunek 2 Wykres rozkladu cechy wyjściowej po posortowaniu

# Zagadnienia teoretyczne

## Drzewo decyzyjne

Drzewo decyzyjne jest popularną strukturą używaną w uczeniu maszynowym. Opisuje w sposób graficzny klasyfikator, czyli funkcję, która zbiorowi cech obiektu przypisuje etykietę klasy. Reprezentuja ona możliwe rozwiązania w oparciu o określone warunki.



Rysunek 3 Reprezentacja drzewa decyzyjnego

Drzewo decyzyjne złożone jest z:

* **Korzeń –** najwyższy węzeł w drzewie, reprezentuje cały zbiór danych przed jakimikolwiek podziałami.
* **Węzłów** – reprezentują one testy przeprowadzane na wartościach atrybutów próbek zbioru danych. Tutaj dokonuje się podziału danych na mniejsze grupy w zależności od wyniku testu.
* **Gałęzi** - wychodzące z węzłów reprezentują różne wyniki przeprowadzanych testów. Każda gałąź prowadzi do kolejnego węzła lub liścia na podstawie rezultatu testu na nadrzędnym węźle.
* **Liści** - to węzły końcowe, które utożsamiane są z etykietami klasowymi. Liście są miejscem, gdzie podejmowana jest ostateczna decyzja dla danej ścieżki.

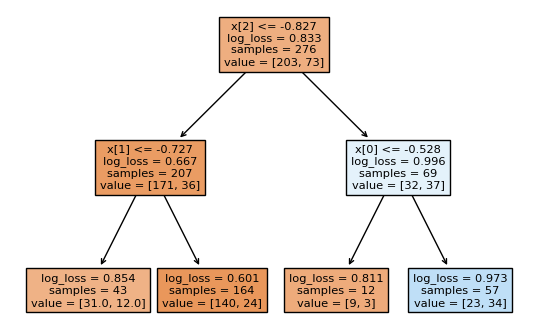
Kontrukcja drzewa decyzyjnego polega na kolejnym przechodzeniu przez poziomy drzewa podejmując decyzje o utworzeniu węzła lub liścia, rozpoczynając od korzenia, aby zapewnić możliwe wysoką sprawność klasyfikatora. W zależności od tego, czy kryterium stopu jest spełnione. Jeśli tak, tworzony jest liść i dobierana jest do niego etykieta, w przeciwnym przypadku tworzony jest węzeł i wybierany jest dla niego test. **Kryterium stopu** jest spełnione, jeśli:

* Aktualny zbiór przypadków zawiera wyłącznie przykłady jednej klasy,
* Wyraźna większość przypadków ma te samą klasę,
* Wyczerpał się zbiór testów.

Uzyskane drzewo powinno charakteryzować się możliwie niewielkim rozmiarem, uczenie polega zatem na wyborze takich testów, aby możliwie szybko i z jak najmniejszym błędem osiągnąć liście wskazujące etykiety klas.

Ważną rolę w tworzeniu drzewa decyzyjnego odgrywa wybranie odpowiedniego **testu**, ponieważ to on w największym stopniu decyduje o własnościach uzyskanego rozwiązania. Można wyróznić testy oparte o wiele bądź jeden atrybut. Częściej stosowane są te oparte o jeden atrybut, prowadzą one do dwóch możliwych rozwiązań. Ważnym problemem jest również wybór atrybutu. Najlepszy będzie taki, który zapewnia maksymalną rozróżnialność elementów różnych klas, a z drugiej jego użycie pozwala generować drzewo o prostej strukturze.

Poniżej na rysunku 4 przedstawiono drzewo decyzyjne z eksperymentu 1 dla głębokości równej 2 i kryterium log\_loss.



Rysunek 4 Drzewo decyzyjne dla eksperymentu 1

## Kryterium wyboru testu

Kryterium wyboru testu odnosie się do metaparemtru „Criterion”. Mówi nam o tym, jakie kryterium zostanie zastosowane przy dokonywaniu optymalnego podziału. W funkcji DecisionTreeClassifier domyślnie jest to gini.

**Entropia** – w kontekście drzewa decyzyjnego mierzy nieczystość klas w danym węźle drzewa, im niższa wartość entropii, tym lepszy podział węzła.

k – liczba klas

– prawdopodobieństwo wystąpienia klasy i w węźle j

**Gini** – jest miarą niepewności węzła w kontekście klasyfikacji. Mierzy prawdopodobieństwo, że losowo wybrane próbki w danym węźle zostaną źle sklafyfikwane. Im niższa wartość Giniego, tym lepszy podział węzła.

k – liczba klas

– prawdopodobieństwo wystąpienia klasy i w węźle j

## Poprawność klasyfikacji

Termin odnoszący się do miary skuteczności klasyfikatora, która określa procent przypadków, w których klasyfikacja modelu jest zgodna z rzeczywistymi etykietami klas danych testowych. Ogólna miara jakości klasyfikacji, którą można interpretować jako stosunek poprawnie sklasyfikowanych przypadków do ogólnej liczby przypadków.

## Walidacja krzyzowa

Walidacja krzyżowa (cross-validation) to technika oceny modeli uczenia maszynowego, która pozwala na oszacowanie ich wydajności na nowych, nieznanych danych. Jest szczególnie użyteczna, gdy dostępny zbiór danych jest ograniczony. Przebieg walidacji krzyżowej składa się z kilku kroków:

* Podział danych na k równych podzbiorów (foldów).
* Rekursyjne trenowanie i testowanie k-krotnie. Jeden z foldów to zbiór testowy, pozostałe k-1 foldów są używane jako zbiór treningowy.
* Po zakończeniu iteracji wyniki są uśredniane, aby uzyskać końcową ocenę wydajności.

W przypadku drzew decyzyjnych walidacja krzyżowa jest użyteczna z kilku powodów:

* Lepsze oszacowanie wydajności, ponieważ model jest testowany na różnych podzbiorach danych, a nie tylko na jednym zbiorze treningowym.
* Lepsza ocena stabilności, ze względu wyeliminowanie problemu wrażliwości drzew decyzyjnych na zmienność danych.
* Minimalizacja ryzyka przeuczenia, ponieważ drzewo nie dopasowuje się tylko do jednych danych treningowych.

# Implementacja drzewa decyzyjnego

Implementacja drzewa decyzyjnego jest kluczowym elementem naszego projektu, w którym badamy wpływ różnych metaparametrów na poprawność klasyfikacji. Algorytm drzewa decyzyjnego oraz przygotowanie danych do eksperymentów na potrzeby tego projektu zostały zrealizowany przy użyciu języka Python i bibliotek *numpy*, *sklearn*, *hickle*, oraz *matplotlib*.

## Przygotowanie danych

Przygotowanie danych korzystając ze zbioru danych *Haberman* w tym projekcie obejmuje wczytanie danych z pliku *haberman.txt,* następnie przekształcenie danych do odpwiedniej formy, znormalizowanie cech do przedziału <-1, 1>, posortowanie ze względu na cechę wyjściową, wizualizację wyników za pomocą biblioteki *matplotlib* izapisanie do pliku *haberman.hkl.* Taka struktura przygotowania danych zapewnia, że są one w odpowiedniej formie do dalszej analizy oraz budowy modeli decyzyjnych.

import hickle as hkl

import numpy as np

import matplotlib.pyplot as plt

filename = './przygotowanie hkl\haberman.txt'

data = np.loadtxt(filename, delimiter=',', dtype=str)

*# Wyciagamy wszystkie cechy z 1,2,3 kolumny i transonujemy aby uzyskac orientacje kolumnową*

x = data[:,0:-1].astype(float).T

*# wyciagamy wszystkie cechy wynikowe z ostatniej kolumny*

y\_t = data[:,-1].astype(float)

*# przeksztalcamy aby uzyskac ostnia kolumne w jednym wierszu*

y\_t = y\_t.reshape(1,y\_t.shape[0])                                                                   *# dla kodowanie klas naturalno-liczbowego*

*# min i max dla każdej z cech przed normalizacją*

*# print(np.transpose([np.array(range(x.shape[0])), x.min(axis=1), x.max(axis=1)]))*

*# normalizacja do przedziału <-1,1> wg zależnoci*

*# x\_norm = (x\_norm\_max-x\_norm\_min)\*(x-x\_min)/(x\_max-x\_min) + x\_norm\_min*

*# w której x\_norm\_max oraz x\_norm\_min są docelowymi wartociami rozpiętoci cechy*

*# wyciagamy najmniejsza wartosc kazdej cechy [30. 58. 0.]*

x\_min = x.min(axis=1)

print(x\_min)

*# wyciagamy najwieksza wartosc kazdej cechy [83. 69. 52.]*

x\_max = x.max(axis=1)

*# ustalenie min, max po znormalizowaniu*

x\_norm\_max = 1

x\_norm\_min = -1

*# stworzenie tablicy zerowej o takim samym kształcie jak x do przechowywania znormalizowanych danych.*

x\_norm = np.zeros(x.shape)

*# sprawdzenie rozpiętoci cech przed normalizacją*

s = np.transpose([np.array(range(x.shape[0])), x.min(axis=1), x.max(axis=1)])

*# print("\n---------------------\n", "Przed normalizacja\n ---------------------\n", s)*

*# przejscie po kazdym wierszu i znormalizowanie danych do przedzialu <-1, 1>*

for i in range(x.shape[0]):

    x\_norm[i,:] = (x\_norm\_max-x\_norm\_min)/(x\_max[i]-x\_min[i])\* \

        (x[i,:]-x\_min[i]) + x\_norm\_min

*# sprawdzenie rozpiętoci cech po normalizacji*

s = np.transpose([np.array(range(x.shape[0])), x\_norm.min(axis=1), x\_norm.max(axis=1)])

*# print("\n---------------------\n", "Po normalizacji\n ---------------------\n", s)*

*# sortowanie y\_t*

y\_t\_s\_ind = np.argsort(y\_t)

x\_n\_s = np.zeros(x.shape)

y\_t\_s = np.zeros(y\_t.shape)

for i in range(x.shape[1]):

    y\_t\_s[0,i] = y\_t[0,y\_t\_s\_ind[0,i]]

    x\_n\_s[:,i] = x\_norm[:,y\_t\_s\_ind[0,i]]

*# Do pliku dane są zapisane w orientacji kolumnowej kolumna - proba*

plt.plot(y\_t\_s[0])

plt.show()

hkl.dump([x,y\_t,x\_norm,x\_n\_s,y\_t\_s],'haberman.hkl')

## Eksperymenty z drzewem decyzyjnym

Poniżej zostaną przedstawione eksperymenty z drzewem decyzyjnym przy użyciu różnych metaparametrów, aby ocenić wydajność modelu za pomocą walidacji krzyżowej. Wyniki są wizualizowane na trójwymiarowych wykresach. W eksperymentach badany jest wpływ dwóch parametrów jednocześnie, ustalając najlepsze wartości dla każdego z parametru. Następnie kolejne dwa parametry są badane z użyciem wcześniej ustalonych najlepszych wartości. To pozwoli nam zrozumieć wpływ metaparametrów na poprawność klasyfikacji drzewa decyzyjnego.

Implementacja w języku Python zawiera pobranie danych z pliku *haberman.hkl,* zmianę wartości cechy wyjściowej, zmianę cech do orientacji wierszowe, zadeklarowanie dwóch kolejnych metaparametrów, ustalenie parametrów walidacji krzyżowej, w naszym wypadku dane są podzielone na 10 części, badanie dla różnych kombinacji metaparametrów poprawności klasyfikacji oraz wizualizację wyników na wykresie trójwymiarowym. W każdym kolejnym eksperymencie wykorzystywane są najlepsze wartości wszystkich poprzednich metaparametrów z poprzednich eksperymentów.

### Eksperyment 1

W ramach pierwszego eksperymentu analizie poddajemy wpływ dwóch metaparametrów klasyfikatora drzewa decyzyjnego *max\_depth*, czyli maksymalna głebokość drzewa decyzyjnego, która domyślnie ustawiona jest na None, czyli bez ograniczeń. *Criterion,*  mówi nam o tym, jakie kryterium zostanie zastosowane przy dokonywaniu optymalnego podziału. Badamy, jak te metaparametry wpływają na dokładność klasyfikacji (PK) przy użyciu walidacji krzyżowej z 10 podziałami. Dla *max\_depth* badamy wartości od 1 do 20, a dla *criterion* badamy 3 opcje: entropy, gini, log\_loss.

import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot\_tree

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('./badanie\_metaparamterow\haberman.hkl')

*# dane w orientacji kolumnowej [[],[],[]]*

*# Zmiana wartości cechy wyjściowej z 1,2 na 0,1*

y\_t\_s -= 1

*# Przetransponowanie cech wejściowych na orientacje wierszową*

x=x\_n\_s.T

*# Przetransponowanie cechy wyjściowej na orientacje wierszową*

y\_t = np.squeeze(y\_t\_s)

*# dla pierwszego zestawu 2 metaparametrów*

criterion\_vec = ["gini", "entropy", "log\_loss"]

criterion\_num\_vec = np.array(range(len(criterion\_vec))) *# [0 1 2] wektor kryteriów*

*# Wypisanie wszystkich kryteriów*

for ind\_criterion in range(len(criterion\_vec)):

    print(criterion\_vec[ind\_criterion])

max\_depth\_vec = np.array(range(1,21,1)) *# [ 1 2 3 ... 20] wektor badanych głebokości*

data = x

target = y\_t

*# Ustalenie ilości części (foldów) na którą zostaną podzielone dane*

CVN = 10

skfold = StratifiedKFold(n\_splits=CVN)

*# Stworzenie pustego wektora na dane wyjsciowe dla kazdej kombinacji metaparemtrow*

PK\_cr\_md\_vec = np.zeros([len(criterion\_vec),len(max\_depth\_vec)])

print(PK\_cr\_md\_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]

*# Inicjalizacja klasyfikatora drzewa decyzyjnego z odpowiednimi parametrami.*

            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)

*# if max\_depth\_vec[max\_depth\_ind] == 1:*

*#    plot\_tree(decision\_tree)*

*# Liczba próbek w zbiorze testowym*

            n\_test\_samples = test.size

*# Obliczenie dokładności predykcji dla tego podziału.*

            PK\_vec[i] = np.sum(result == y\_test) / n\_test\_samples

*# Obliczenie średniej dokładności dla danej kombinacji kryterium i głębokości drzewa.*

        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='viridis')

ax.set\_xlabel('criterion')

ax.set\_xticks(np.array(range(len(criterion\_vec))))

ax.set\_xticklabels(criterion\_vec)

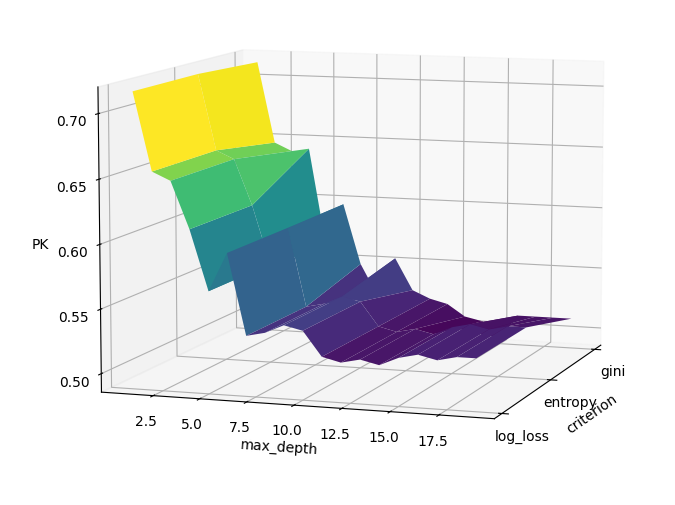
ax.set\_ylabel('max\_depth')

ax.set\_zlabel('PK')

ax.view\_init(30, 200)

plt.show()

Analizując wyniki dla kryterium *log\_loss* oraz *entropy* okazały się być identyczne i właśnie dla nich zostały uzyskane najlepsze wartości PK, które osiągneły 71,56%, dla kryterium *gini* były one gorsze, wynosiły 71,22%. Dla drugiego parametru, czyli głębokości drzewa, najlepsza wartość została uzyskana przy głębokości równej 1. W miarę zwiększania głębokości drzewa, dokładność stopniowo spada. Ostatecznie najlepszym wyborem będzie głebokość drzewa 1 oraz kryterium *entropy*, ze względu na popularność i łatwość interpretacji.



Rysunek 5 Wykres zależności PK od max\_depth oraz criterion algorytmu drzewa decyzyjnego dla zbioru Haberman

### Ekpseryment 2

W ramach drugiego eksperymentu analizie poddajemy parę dwóch kolejnych metaparametrów, *splitter* oraz *min\_samples\_split* oraz ich wpływ na poprawnośc klasyfikacji drzewa decyzyjnego. *Splitter* jest parametrem określającym strategię podziału węzłów, mamy do wyboru ‘best’, który wybiera najlepszy podział w węźle, ‘random’, który wybiera podział losowo spośród możliwych podziałów. *Min\_samples\_split* mówi nam o minimalnej liczbie obserwacji w liściu potrzebnej do dokonania podziału, domyślnie ustawiona jest wartość 2. Oznacza to, że aby dany węzeł został podzielony, musi zawierać co najmniej tyle próbek, ile wynosi wartość *min\_samples\_split*.

import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot\_tree

import hickle as hkl

from sklearn.model\_selection import StratifiedKFold

import matplotlib.pyplot as plt

*# Wczytanie danych z pliku hickle*

x, y\_t, x\_norm, x\_n\_s, y\_t\_s = hkl.load('./badanie\_metaparamterow/haberman.hkl')

*# Zmiana wartości cechy wyjściowej z 1,2 na 0,1*

y\_t\_s -= 1

*# Przetransponowanie cech wejściowych na orientacje wierszową*

x = x\_n\_s.T

*# Przetransponowanie cechy wyjściowej na orientacje wierszową*

y\_t = np.squeeze(y\_t\_s)

*# Definicja metaparametrów do badania*

splitter\_vec = ["best", "random"]

min\_samples\_split\_vec = np.array(range(2, 21, 1))

data = x

target = y\_t

*# Ustalenie ilości części (foldów) na którą zostaną podzielone dane*

CVN = 10

skfold = StratifiedKFold(n\_splits=CVN)

*# Stworzenie pustego wektora na dane wyjsciowe dla kazdej kombinacji metaparemtrow*

PK\_cr\_md\_vec = np.zeros([len(splitter\_vec), len(min\_samples\_split\_vec)])

for splitter\_ind in range(len(splitter\_vec)):

    for min\_samples\_split\_ind in range(len(min\_samples\_split\_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]

*# Inicjalizacja klasyfikatora drzewa decyzyjnego z odpowiednimi parametrami.*

            decision\_tree = DecisionTreeClassifier(

                random\_state=0,

                max\_depth=1,

                criterion='entropy',

                splitter=splitter\_vec[splitter\_ind],

                min\_samples\_split=min\_samples\_split\_vec[min\_samples\_split\_ind]

            )

            decision\_tree = decision\_tree.fit(x\_train, y\_train)

            result = decision\_tree.predict(x\_test)

*# Liczba próbek w zbiorze testowym*

            n\_test\_samples = test.size

*# Obliczenie dokładności predykcji dla tego podziału.*

            PK\_vec[i] = np.sum(result == y\_test) / n\_test\_samples

*# Obliczenie średniej dokładności dla danej kombinacji kryterium i głębokości drzewa.*

        PK\_cr\_md\_vec[splitter\_ind, min\_samples\_split\_ind] = np.mean(PK\_vec)

        print("splitter: {} | min\_samples\_split: {} | PK: {}".format(

            splitter\_vec[splitter\_ind],

            min\_samples\_split\_vec[min\_samples\_split\_ind],

            PK\_cr\_md\_vec[splitter\_ind, min\_samples\_split\_ind]

        ))

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(111, projection='3d')

X, Y = np.meshgrid(np.array(range(len(splitter\_vec))), min\_samples\_split\_vec)

surf = ax.plot\_surface(X, Y, PK\_cr\_md\_vec.T, cmap='viridis')

ax.set\_xlabel('splitter')

ax.set\_xticks(np.array(range(len(splitter\_vec))))

ax.set\_xticklabels(splitter\_vec)

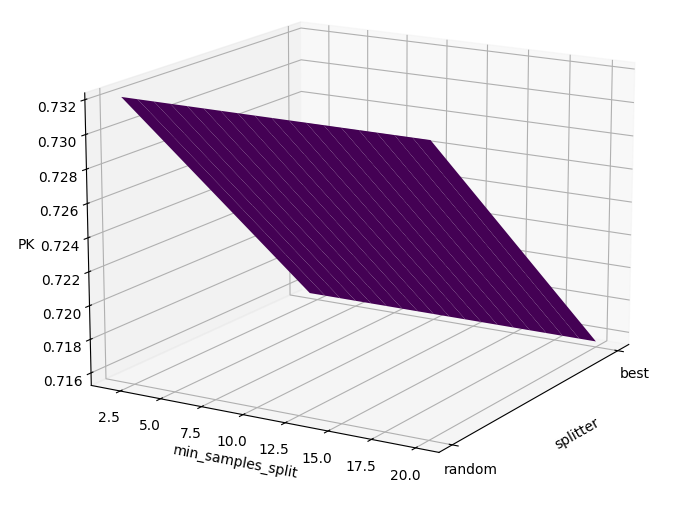
ax.set\_ylabel('min\_samples\_split')

ax.set\_zlabel('PK')

ax.view\_init(30, 200)

plt.show()

Analizując wyniki eksperymentu najlepsza uzyskana poprawność klasyfikacji wynosi 73,2% dla metaparametru *splitter,* gdy wybierzemy opcję ‘random’ uzyskujemy lepsze wyniki niż dla opcji ‘best’ przy każdej wartości *min\_samples\_split,*  to oznacza, że lepszą opcją będzie wybór losowego podziału. Parametr *min\_samples\_split* dla żadnej z badanych wartości nie powoduje zmiany PK, z tego powodu dla uproszczenia wybieramy wartość 2, czyli wartość domyślną.



Rysunek 6 Wykres zależności PK od min\_samples\_split oraz splitter algorytmu drzewa decyzyjnego dla zbioru Haberman

### Eksperyment 3

W ramach trzeciego eksperymentu analizie poddajemy parę dwóch kolejnych metaparametrów *min\_samples\_leaf* oraz *min\_weight\_fraction* oraz ich wpływ na poprawnośc klasyfikacji drzewa decyzyjnego. *Min\_samples\_leaf* to parametr, który określa minimalną liczbę próbek, które muszą znajdować się w liściu drzewa decyzyjnego po podziale. *Min\_weight\_fraction*  to minimalna wymagana ułamkowa suma wag, która musi znaleźć się w liściu drzewa.

import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot\_tree

import hickle as hkl

from sklearn.model\_selection import StratifiedKFold

import matplotlib.pyplot as plt

*# Wczytanie danych z pliku hickle*

x, y\_t, x\_norm, x\_n\_s, y\_t\_s = hkl.load('./badanie\_metaparamterow/haberman.hkl')

*# Zmiana wartości cechy wyjściowej z 1,2 na 0,1*

y\_t\_s -= 1

*# Przetransponowanie cech wejściowych na orientacje wierszową*

x = x\_n\_s.T

*# Przetransponowanie cechy wyjściowej na orientacje wierszową*

y\_t = np.squeeze(y\_t\_s)

*# Definicja metaparametrów do badania*

min\_samples\_leaf\_vec = np.array(range(1, 10, 1))

min\_weight\_fraction\_leaf\_vec = np.linspace(0, 0.3, 10)

data = x

target = y\_t

*# Ustalenie ilości części (foldów) na którą zostaną podzielone dane*

CVN = 10

skfold = StratifiedKFold(n\_splits=CVN)

*# Stworzenie pustego wektora na dane wyjsciowe dla kazdej kombinacji metaparametrow*

PK\_cr\_md\_vec = np.zeros([len(min\_samples\_leaf\_vec), len(min\_weight\_fraction\_leaf\_vec)])

for min\_samples\_leaf\_ind in range(len(min\_samples\_leaf\_vec)):

    for min\_weight\_fraction\_leaf\_ind in range(len(min\_weight\_fraction\_leaf\_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]

*# Inicjalizacja klasyfikatora drzewa decyzyjnego z odpowiednimi parametrami.*

            decision\_tree = DecisionTreeClassifier(

                random\_state=0,

                max\_depth=1,

                criterion='entropy',

                splitter='random',

                min\_samples\_split=2,

                min\_samples\_leaf=min\_samples\_leaf\_vec[min\_samples\_leaf\_ind],

                min\_weight\_fraction\_leaf=min\_weight\_fraction\_leaf\_vec[min\_weight\_fraction\_leaf\_ind]

            )

            decision\_tree = decision\_tree.fit(x\_train, y\_train)

            result = decision\_tree.predict(x\_test)

*# Liczba próbek w zbiorze testowym*

            n\_test\_samples = test.size

*# Obliczenie dokładności predykcji dla tego podziału.*

            PK\_vec[i] = np.sum(result == y\_test) / n\_test\_samples

*# Obliczenie średniej dokładności dla danej kombinacji kryterium i głębokości drzewa.*

        PK\_cr\_md\_vec[min\_samples\_leaf\_ind, min\_weight\_fraction\_leaf\_ind] = np.mean(PK\_vec)

        print("min\_samples\_leaf: {} | min\_weight\_fraction\_leaf: {} | PK: {}".format(

            min\_samples\_leaf\_vec[min\_samples\_leaf\_ind],

            min\_weight\_fraction\_leaf\_vec[min\_weight\_fraction\_leaf\_ind],

            PK\_cr\_md\_vec[min\_samples\_leaf\_ind, min\_weight\_fraction\_leaf\_ind]

        ))

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(111, projection='3d')

X, Y = np.meshgrid(min\_samples\_leaf\_vec, min\_weight\_fraction\_leaf\_vec)

surf = ax.plot\_surface(X, Y, PK\_cr\_md\_vec.T, cmap='viridis')

ax.set\_xlabel('min\_samples\_leaf')

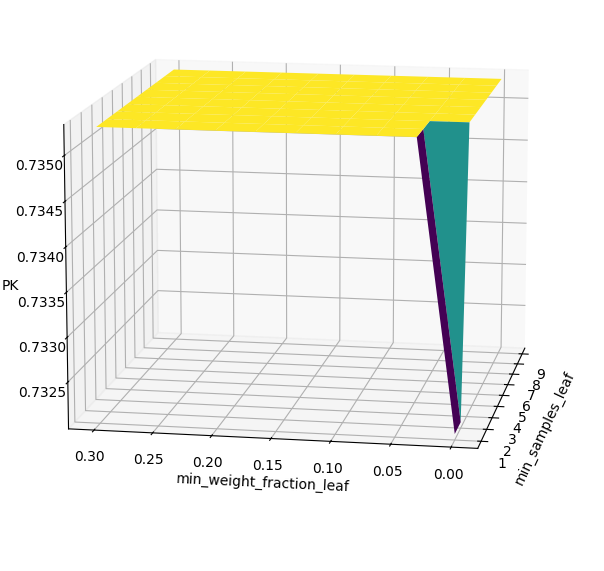
ax.set\_ylabel('min\_weight\_fraction\_leaf')

ax.set\_zlabel('PK')

ax.view\_init(30, 200)

plt.show()

Analizując wyniki dla wszystkich kombinacji wartości min\_samples\_leaf (od 1 do 9) i min\_weight\_fraction\_leaf (od 0.0 do 0.3), dokładność klasyfikacji jest bardzo zbliżona i wynosi 73,52%. Ze względu na brak znaczącej róznicy dla zmian metaparametrówwybierzemy takie wartości, aby jak najmniej ograniczyć elastyczność konstrukcji drzewa, czyli *min\_samples\_leaf* 1, a dla *min\_weight\_fraction\_leaf*  ustalimy wartość 0.03. Dla tych parametrów uzyskana była najwyższa wartość PK.



Rysunek 7 Wykres zależności PK od min\_weight\_fraction\_leaf oraz min\_samples\_leaf algorytmu drzewa decyzyjnego dla zbioru Haberman

### Eksperyment 4

W ramach czwartego eksperymentu analizie poddajemy parę dwóch kolejnych metaparametrów *max\_features*, *max\_leaf\_ nodes* oraz ich wpływ na poprawnośc klasyfikacji drzewa decyzyjnego. *Max\_features* stosowanydo ograniczenia liczby cech rozważanych przy każdym podziale węzła. *Max\_leaf\_nodes* to parametr który ogranicza maksymalną liczbę liści w drzewie decyzyjnym.

import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot\_tree

import hickle as hkl

from sklearn.model\_selection import StratifiedKFold

import matplotlib.pyplot as plt

*# Wczytanie danych z pliku hickle*

x, y\_t, x\_norm, x\_n\_s, y\_t\_s = hkl.load('./badanie\_metaparamterow/haberman.hkl')

*# Zmiana wartości cechy wyjściowej z 1,2 na 0,1*

y\_t\_s -= 1

*# Przetransponowanie cech wejściowych na orientacje wierszową*

x = x\_n\_s.T

*# Przetransponowanie cechy wyjściowej na orientacje wierszową*

y\_t = np.squeeze(y\_t\_s)

*# Definicja metaparametrów do badania*

max\_features\_vec = np.arange(1, 4, 1)  *# Liczba cech od 1 do liczby cech w danych*

max\_leaf\_nodes\_vec = np.arange(2, 21, 2)  *# Liczba liści od 2 do 20, co 2*

data = x

target = y\_t

*# Ustalenie ilości części (foldów) na którą zostaną podzielone dane*

CVN = 10

skfold = StratifiedKFold(n\_splits=CVN)

*# Stworzenie pustego wektora na dane wyjsciowe dla kazdej kombinacji metaparametrow*

PK\_cr\_md\_vec = np.zeros([len(max\_features\_vec), len(max\_leaf\_nodes\_vec)])

for max\_features\_ind in range(len(max\_features\_vec)):

    for max\_leaf\_nodes\_ind in range(len(max\_leaf\_nodes\_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]

*# Inicjalizacja klasyfikatora drzewa decyzyjnego z odpowiednimi parametrami.*

            decision\_tree = DecisionTreeClassifier(

                random\_state=0,

                max\_depth=1,

                criterion='entropy',

                splitter='random',

                min\_samples\_split=2,

                min\_samples\_leaf=1,

                min\_weight\_fraction\_leaf=0.03,

                max\_features=max\_features\_vec[max\_features\_ind],

                max\_leaf\_nodes=max\_leaf\_nodes\_vec[max\_leaf\_nodes\_ind]

            )

            decision\_tree = decision\_tree.fit(x\_train, y\_train)

            result = decision\_tree.predict(x\_test)

*# Liczba próbek w zbiorze testowym*

            n\_test\_samples = test.size

*# Obliczenie dokładności predykcji dla tego podziału.*

            PK\_vec[i] = np.sum(result == y\_test) / n\_test\_samples

*# Obliczenie średniej dokładności dla danej kombinacji kryterium i głębokości drzewa.*

        PK\_cr\_md\_vec[max\_features\_ind, max\_leaf\_nodes\_ind] = np.mean(PK\_vec)

        print("max\_features: {} | max\_leaf\_nodes: {} | PK: {}".format(

            max\_features\_vec[max\_features\_ind],

            max\_leaf\_nodes\_vec[max\_leaf\_nodes\_ind],

            PK\_cr\_md\_vec[max\_features\_ind, max\_leaf\_nodes\_ind]

        ))

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(111, projection='3d')

X, Y = np.meshgrid(max\_features\_vec, max\_leaf\_nodes\_vec)

surf = ax.plot\_surface(X, Y, PK\_cr\_md\_vec.T, cmap='viridis')

ax.set\_xlabel('max\_features')

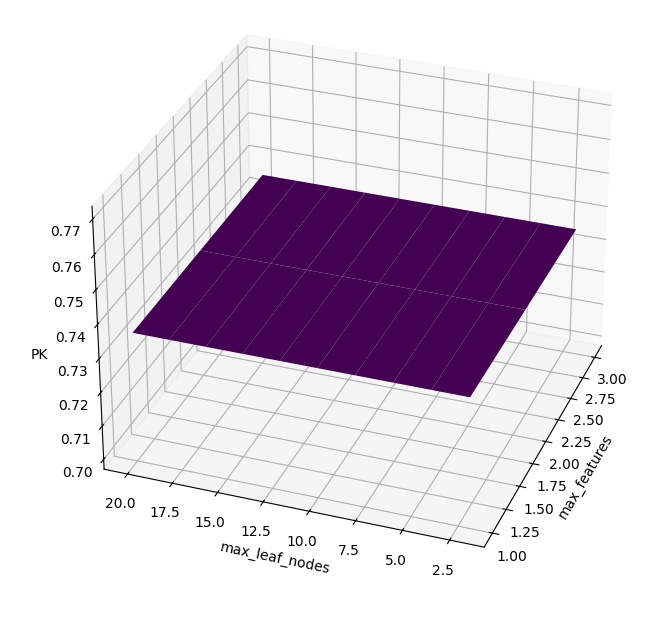
ax.set\_ylabel('max\_leaf\_nodes')

ax.set\_zlabel('PK')

ax.view\_init(30, 200)

plt.show()

Analizując wyniki eksperymentu można zauważyć, że zmiana wartości badanych metaparametrów nie zmienia wartości poprawności klasyfikacji, które utrzymują się na poziomie 73,52%. Pomimo braku różnic w wynikach, warto wybrać wartości metaparametrów, które mogą być optymalne przy dalszych eksperymentach. Ostatecznie przyjmiemy wartości dla *max\_features* 3 oraz dla *max\_leaf\_nodes 10*, przyjmujemy średnią wartość z badanego zakresu, aby uniknąć przeuczenia.



Rysunek 8 Wykres zależności PK od max\_leaf\_nodes oraz max\_features algorytmu drzewa decyzyjnego dla zbioru Haberman

### Eksperyment 5

W ramach piątego eksperymentu analizie poddajemy parę dwóch ostatnich metaparametrów *min\_impurity\_decrease*, *ccp\_alpha* oraz ich wpływ na poprawnośc klasyfikacji drzewa decyzyjnego. *Min\_impurity\_decrease* to minimalna redukcja zanieczyszczenia wymagana do podziału węzła. Im wyższa wartość *min\_impurity\_decrease*, tym bardziej restrykcyjne będą podziały w drzewie, co może prowadzić do mniejszej skomplikowania modelu i unikania przeuczenia. *Ccp\_alpha* to parametr przycinania kosztów złożoności.

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 hickle*

x, y\_t, x\_norm, x\_n\_s, y\_t\_s = hkl.load('./badanie\_metaparamterow/haberman.hkl')

*# Zmiana wartości cechy wyjściowej z 1,2 na 0,1*

y\_t\_s -= 1

*# Przetransponowanie cech wejściowych na orientacje wierszową*

x = x\_n\_s.T

*# Przetransponowanie cechy wyjściowej na orientacje wierszową*

y\_t = np.squeeze(y\_t\_s)

*# Definicja metaparametrów do badania*

min\_impurity\_decrease\_vec = np.linspace(0, 0.2, 10)

ccp\_alpha\_vec = np.linspace(0, 0.2, 10)

data = x

target = y\_t

*# Ustalenie ilości części (foldów) na którą zostaną podzielone dane*

CVN = 10

skfold = StratifiedKFold(n\_splits=CVN)

*# Stworzenie pustego wektora na dane wyjściowe dla każdej kombinacji metaparametrów*

PK\_cr\_md\_vec = np.zeros([len(min\_impurity\_decrease\_vec), len(ccp\_alpha\_vec)])

for min\_impurity\_decrease\_ind in range(len(min\_impurity\_decrease\_vec)):

    for ccp\_alpha\_ind in range(len(ccp\_alpha\_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]

*# Inicjalizacja klasyfikatora drzewa decyzyjnego z odpowiednimi parametrami.*

            decision\_tree = DecisionTreeClassifier(

                random\_state=0,

                max\_depth=1,

                criterion='entropy',

                splitter='random',

                min\_samples\_split=2,

                min\_samples\_leaf=1,

                min\_weight\_fraction\_leaf=0.03,

                max\_features=3,

                max\_leaf\_nodes=10,

                min\_impurity\_decrease=min\_impurity\_decrease\_vec[min\_impurity\_decrease\_ind],

                ccp\_alpha=ccp\_alpha\_vec[ccp\_alpha\_ind]

            )

            decision\_tree = decision\_tree.fit(x\_train, y\_train)

            result = decision\_tree.predict(x\_test)

*# Liczba próbek w zbiorze testowym*

            n\_test\_samples = test.size

*# Obliczenie dokładności predykcji dla tego podziału.*

            PK\_vec[i] = np.sum(result == y\_test) / n\_test\_samples

*# Obliczenie średniej dokładności dla danej kombinacji kryterium i głębokości drzewa.*

        PK\_cr\_md\_vec[min\_impurity\_decrease\_ind, ccp\_alpha\_ind] = np.mean(PK\_vec)

        print("min\_impurity\_decrease: {} | ccp\_alpha: {} | PK: {}".format(

            min\_impurity\_decrease\_vec[min\_impurity\_decrease\_ind],

            ccp\_alpha\_vec[ccp\_alpha\_ind],

            PK\_cr\_md\_vec[min\_impurity\_decrease\_ind, ccp\_alpha\_ind]

        ))

*# Wizualizacja wyników*

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(111, projection='3d')

X, Y = np.meshgrid(min\_impurity\_decrease\_vec, ccp\_alpha\_vec)

surf = ax.plot\_surface(X, Y, PK\_cr\_md\_vec.T, cmap='viridis')

ax.set\_xlabel('min\_impurity\_decrease')

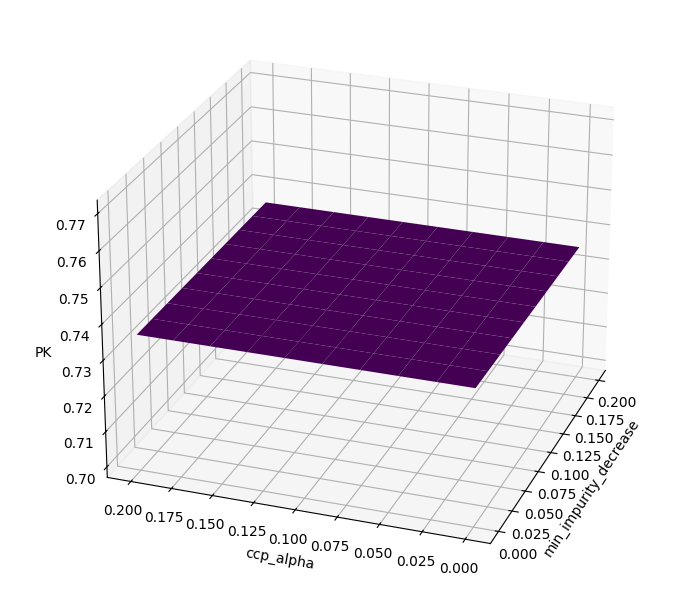
ax.set\_ylabel('ccp\_alpha')

ax.set\_zlabel('PK')

ax.view\_init(30, 200)

plt.show()

Analizując wyniki podobnie jak w poprzednim eksperymencie zmiana wartości podanych w eksperymencie parametrów nie wpływa na zmiane poprawności klasyfikacji drzewa i utrzymuje się na poziomie 73,52%. Brak zmian, podobnie jak w poprzednim eksperymencie może być spowodowany ograniczoną głebokością drzewa.



Rysunek 9 Wykres zależności PK od ccp\_alpha oraz min\_impurity\_decrease algorytmu drzewa decyzyjnego dla zbioru Haberman

# Wnioski

Celem projektu było zapoznanie się z drzewem decyzyjnym, czyli stosunkowo prostym algorytmem w uczeniu maszynowym oraz zbadaniem wpływu metaparametrów na poprawność klasyfikacji. Eksperymenty zostały przeprowadzone na zbiorze danych *Haberman*, który jest prostym zbiorem, składa się z niewielkiej liczby cech oraz stosunkowo niewielkiej liczby rekordów, sam zbiór jest dobrze przygotowany do analizy, ponieważ wszystkie cechy domyślnie są numeryczne, co znacznie ułatwia pracę. Cechą wyjściową zbioru jest to czy pacjent przeżyje do 5 lat po operacji. Pierwszym krokiem było zapoznanie się ze zbiorem danych, na którym będą przeprowadzane eksperymenty. Następnym zadaniem było zapoznanie się z informacjami dotyczącymi drzewa oraz wszystkimi pojęciami, które będą wykorzystywane w eksperymencie. Główna koncepcja, która stoi za drzewem to zadawanie pytań, na które można odpowiedzieć tak lub nie. Wybór tych pytań jest zależny od kryteriów.

W eksperymentach naszym celem było dobranie takich metaparametrów, aby uzyskać najlepszą możliwą poprawność klasyfikacji dla danego zbioru, wykorzystana do tego została walidacja krzyżowa z podziałem na 10 foldów. W każdym kolejnym eksperymencie określanie były uprzednio wybrane najbardziej optymalne metaparametry z poprzednich testów. W pierwszym eksperymencie parametry *criterion*  oraz *max\_depth* odgrywały znaczącą rolę dla poprawności klasyfikacji. Zbiór danych, na którym przeprowadzane były eksperymenty posiada małą ilość, ze względu na to wyniki dla różnych kryteriów nie różniły się znacząco, a dla log\_loss oraz entropy były identyczne. Powodem tego może być fakt, że oba te kryteria dążą do maksymalizacji czystości węzła, co w prostych zbiorach danych może prowadzić do identycznych podziałów. Dla *max\_depth,* który blokuje maksymalną głębokość, aby ograniczyć rozrost drzewa i bardziej sprecyzować model oraz doprowadzić do uogólnienia rozwiązania problemu, wraz ze wzrostem wartości maksymalnej wyniki spadały, co wskazywało na przeuczenie (overfitting). Od drugiego eksperymentu wyniki jakie były otrzymywane nie zmieniały się znacząco, a często nie zmieniały się w ogóle, na co mogła wpływać ustalona maksymalna głębokość na 1, przez co maksymalny możliwy podział był tylko jeden. Dla parametru *min\_samples\_split* brak jakichkolwiek zmian, ze względu właśnie na ograniczenie w postaci jednego możliwego podziału korzenia, który zawiera wszystkie możliwe próbki. Kolejnym metaparamtrem jest *splitter,* który określa sposób podziału węzła. Dla niego lepsza wartość została uzyskana przy opcji random, może to wynikać z tego, że opcja best koncentruje się na znajdowaniu najlepszego możliwego podziału w danym momencie, co może prowadzić do znalezienia lokalnie optymalnych, ale globalnie suboptymalnych rozwiązań. Może prowadzić również do overfittingu, ponieważ zawsze wybiera najlepszy możliwy podział dla danych treningowych, co może nie oddziaływać dobrze na dane testowe. Reszta badanych metaparametrów w żadnym stopniu nie wpływała na wyniki poprawności klasyfikacji. Ostatecznie przy ostatnim eksperymencie najlepszy uzyskany wynik poprawności wynosił 73,52%, a najbardziej znaczącymi paramatrami był *criterion* oraz *max\_depth.*

# Bibliografia

[1] <https://archive.ics.uci.edu/dataset/43/haberman+s+survival>

[2] <http://materialy.prz-rzeszow.pl/pracownik/pliki/34/Przyk%C5%82adowe%20skrypty%203.pdf>

[3] <http://materialy.prz-rzeszow.pl/pracownik/pliki/34/SI_P2__Procedura%20przygotowania%20danych.pdf>

[4] <https://gdudek.el.pcz.pl/images/Dydaktyka/Wyklad3_UM_DDkl.pdf>

[5] <https://jakbadacdane.pl/drzewo-decyzyjne-a-hiperparametry/>

[6] <https://scikit-learn.org/stable/modules/tree.html>

[7] <https://miroslawmamczur.pl/czym-jest-i-jak-dziala-drzewo-decyzyjne/>

[8] <https://gdudek.el.pcz.pl/images/Dydaktyka/Wyklad3_UM_DDkl.pdf>