Politechnika Krakowska
im. Tadeusza Kościuszki

Odkrywanie wad serca z wykorzystaniem algorytmu ID3 (SPECT Heart)

Damian Sygut

Numer albumu:**143654**

Komputerowe Wspomaganie Decyzji

Projekt 2023/2024

Wydział Inżynierii Elektrycznej i Komputerowej

Politechnika Krakowska

Cel Projektu

Opracowanie modelu na bazie drzew decyzyjnych, wspierającego specjalistów w efektywnej diagnozie wad serca.

Wstęp

Nowoczesna medycyna dynamicznie wykorzystuje postęp technologiczny, w tym uczenie maszynowe, do efektywniejszej diagnozy schorzeń, w tym wad serca. Projekt ten skupia się na zastosowaniu algorytmu ID3 w analizie danych medycznych z zestawu SPECT Heart, z celem identyfikacji wad serca. Dane zostały uzyskane z UCI Machine Learning Repository.

Teoretyczne Podstawy Uczenia Maszynowego

Uczenie maszynowe jest kluczowym składnikiem współczesnej sztucznej inteligencji, które skupia się na opracowaniu algorytmów zdolnych do uczenia się i dokonywania prognoz lub decyzji na podstawie danych. Istnieją różne rodzaje uczenia maszynowego, każdy z nich ma swoje zastosowania i najlepsze praktyki:

1. **Uczenie Nadzorowane**: Uczenie nadzorowane odnosi się do procesu, w którym model uczy się na podstawie etykietowanych danych. Dane wejściowe (cechy) są sparowane z odpowiednimi wynikami (etykietami), a model uczy się przewidywać etykiety dla nowych, niewidzianych danych.
2. **Uczenie Nienadzorowane**: W uczeniu nienadzorowanym model analizuje i grupuje nieetykietowane dane na podstawie ich cech, pomagając w odkrywaniu ukrytych wzorców i struktur w danych.
3. **Uczenie przez Wzmacnianie**: Uczenie przez wzmacnianie polega na trenowaniu modelu poprzez nagradzanie pożądanych zachowań i karanie niepożądanych, zachęcając model do samodzielnego odkrywania optymalnych strategii działania w danym środowisku.

`

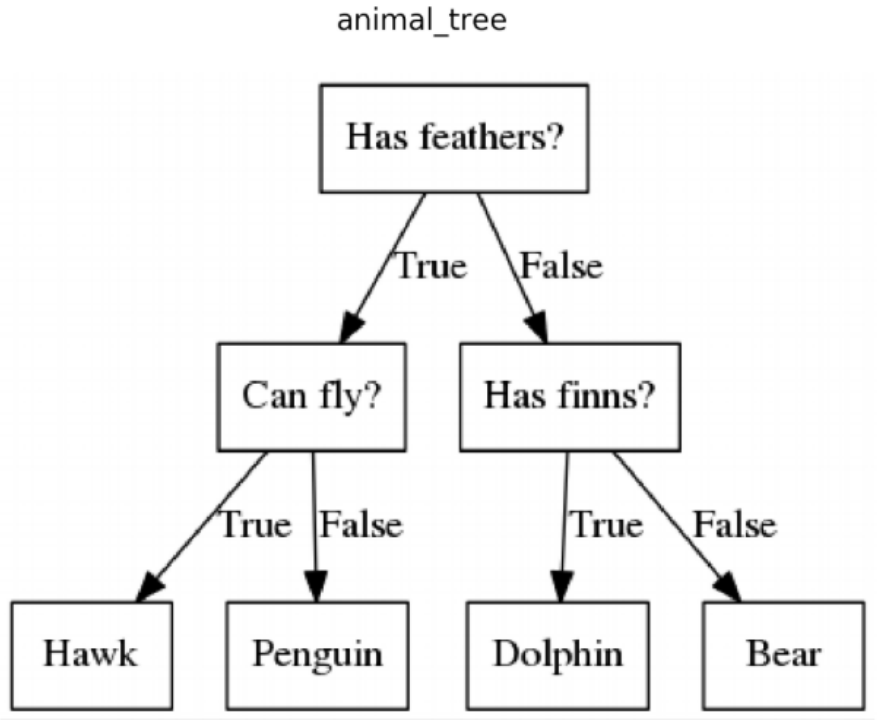
Proces Tworzenia Modeli Uczenia Maszynowego

Wyróżniamy cztery etapy: przetwarzanie wstępne, uczenie, ewaluacja i prognozowanie.

1. **Przygotowanie i Przetwarzanie Wstępne Danych (Preprocessing)**: Ten etap jest fundamentem sukcesu każdego modelu uczenia maszynowego. Polega na dokładnym czyszczeniu danych, ich normalizacji oraz transformacji, aby zapewnić, że model będzie pracował na czystych, spójnych i odpowiednio sformatowanych danych. Usuwane są tu anomalie, brakujące wartości, a także dokonuje się normalizacji różnych skal cech.
2. **Faza Uczenia (Learning)**: W tej fazie następuje rzeczywiste "uczenie" modelu. Model analizuje dostarczone dane treningowe, ucząc się rozpoznawać wzorce i zależności. To, jakie techniki i algorytmy zostaną zastosowane, zależy od specyfiki problemu - może to być na przykład regresja liniowa w przypadku prognozowania wartości numerycznych lub sieci neuronowe w zadaniach klasyfikacji.
3. **Ewaluacja Modelu (Evaluation)**: Po etapie uczenia, model jest testowany, aby ocenić jego efektywność i zdolność do generalizacji na nowych danych. Używa się do tego celu zestawu danych testowych, które nie były wykorzystane podczas treningu. Jest to kluczowy moment, w którym można ocenić, czy model jest gotowy do użycia w rzeczywistych scenariuszach, czy też wymaga dalszego dostosowania.
4. **Stosowanie Modelu do Prognozowania (Prediction)**: Po pomyślnej ewaluacji model jest gotowy do stosowania na rzeczywistych, nieznanych wcześniej danych. W tej fazie model wykorzystuje swoją nauczoną wiedzę do dokonywania przewidywań, które mogą być użyte do podejmowania decyzji lub dalszych analiz.

Drzewa Decyzyjne

Są to narzędzia graficzne wspierające decyzje, złożone z korzeni, węzłów i liści.
Reprezentują procesy decyzyjne, klasyfikując obserwacje od korzenia do liścia.



Na podstawie załączonego obrazka drzewa decyzyjnego dotyczącego klasyfikacji zwierząt, otrzymujemy następujące reguły, które odpowiadają każdej ścieżce od korzenia do liścia:
IF (Has feathers? = True) AND (Can fly? = True) THEN (Animal = Hawk)
IF (Has feathers? = True) AND (Can fly? = False) THEN (Animal = Penguin)
IF (Has feathers? = False) AND (Has finns? = True) THEN (Animal = Dolphin)
IF (Has feathers? = False) AND (Has finns? = False) THEN (Animal = Bear)

Stosując logikę uproszczenia, można zauważyć, że niezależnie od tego, czy zwierzę ma pióra, czy nie, jeśli umie latać, to spośród naszych zwierząt wskazuje to na orłą. Zatem reguła dotycząca identyfikacji orła może być zredukowana do prostszej formy, pomijając atrybut dotyczący piór:
IF (Can fly? = True) THEN (Animal = Hawk)

Inicjacja Drzewa Decyzyjnego
Początkowo sprawdzane jest, czy dane wejściowe składają się wyłącznie z przykładów jednej klasy. Jeśli tak, tworzony jest liść drzewa, który oznacza konkretną klasę. Jest to sytuacja idealna, w której wszystkie obserwacje można jednoznacznie zaklasyfikować bez dalszego podziału.

Selekcja Atrybutów

Następnie algorytm wybiera atrybut, który najlepiej dzieli dane na podzbiory według określonej cechy. Wybór ten jest dokonywany na podstawie miary informacyjnej, jaką jest entropia – wybierany jest atrybut, który maksymalizuje przyrost informacji, tj. najbardziej zmniejsza niepewność klasyfikacji.

Rekurencyjne Budowanie Drzewa

Proces dzielenia jest powtarzany rekurencyjnie dla każdego wytworzonego podzbioru. Każdy kolejny węzeł drzewa reprezentuje kolejny atrybut, który najlepiej segreguje dane na tym etapie. Proces kontynuowany jest aż do osiągnięcia jednorodności klas w każdym podzbiorze lub do spełnienia innych warunków zakończenia procesu, jak np. osiągnięcie maksymalnej głębokości drzewa.

Znaczenie Entropii w Procesie Podziału

Entropia to kluczowa koncepcja w teorii informacji, która w kontekście ID3 służy jako metryka dla oceny, jak dobrze dany atrybut radzi sobie z podziałem danych. Niższa wartość entropii w podzbiorze oznacza większą pewność co do klasyfikacji:

Obliczanie Entropii

Entropia jest obliczana na podstawie prawdopodobieństwa występowania każdej z klas w danym zbiorze. Im bardziej jednorodny jest zbiór (wszystkie przykłady należą do jednej klasy), tym entropia jest niższa.

Ograniczenia ID3 i Rozwój Algorytmu C4.5

ID3 jest algorytmem potężnym, ale posiada kilka ograniczeń, które C4.5 stara się przezwyciężyć:

Problemy z Atrybutami Ciągłymi

ID3 wymaga dyskretnych atrybutów, co może być ograniczeniem w przypadku danych ciągłych. C4.5 wprowadza metody radzenia sobie z takimi atrybutami poprzez ich dyskretyzację.

Brakujące Dane

ID3 nie obsługuje sytuacji, gdzie niektóre dane są niekompletne. C4.5 oferuje mechanizmy, takie jak uzupełnianie brakujących wartości, co pozwala na bardziej elastyczne radzenie sobie z niekompletnymi zbiorami danych.

Przycinanie Drzewa: W celu uniknięcia nadmiernego dopasowania, C4.5 wprowadza przycinanie drzewa, które usuwa gałęzie, które nie przyczyniają się znacząco do poprawy klasyfikacji.

Walidacja Krzyżowa i Macierz Pomyłek

Walidacja krzyżowa (cross-validation) oraz macierz pomyłek są technikami ewaluacyjnymi, które pozwalają na dokładną ocenę wydajności modelu klasyfikacyjnego:

Walidacja Krzyżowa: Dzieli dane na wielokrotne zestawy treningowe i testowe, aby zapewnić, że ocena modelu jest niezależna od przypadkowego podziału danych. Pozwala to na bardziej wiarygodne oszacowanie, jak model będzie działał na nowych danych.

Macierz Pomyłek

Jest narzędziem oceny jakości klasyfikacji, która przedstawia szczegółowo, jak dobrze model radzi sobie z przewidywaniem klas. Zawiera ona informacje o prawdziwie pozytywnych, fałszywie pozytywnych, prawdziwie negatywnych oraz fałszywie negatywnych przewidywaniach, umożliwiając obliczenie takich metryk jak precyzja, pełność oraz dokładność modelu.

Szczegółowa Analiza Zestawu Danych SPECT Heart

Zestaw danych SPECT Heart skupia się na wykorzystaniu obrazowania medycznego, a konkretnie technologii Single Proton Emission Computed Tomography, do diagnozowania chorób serca. Składający się z 267 przypadków, każdy opisany jest za pomocą 23 atrybutów, które obejmują 22 binarne atrybuty częściowej diagnostyki oraz jeden atrybut klasy docelowej, który wskazuje na obecność lub brak wady serca. Wszystkie te dane są starannie podzielone na dwie kluczowe grupy: zestaw treningowy i testowy, pozwalając na skuteczne trenowanie i ocenę modelu predykcyjnego.

Rozbudowana Analiza Eksploracyjna Danych

Baza danych opisuje diagnozowanie obrazów tomografii emisji pojedynczych protonów (Single Proton Emission Computed Tomography - SPECT) serca.

Każdy pacjent jest sklasyfikowany jako normalny lub nietypowy.

Zestaw danych obejmuje 267 zestawów obrazów SPECT (pacjentów), które zostały przetworzone w celu wydobycia cech podsumowujących oryginalne obrazy SPECT.

W rezultacie dla każdego pacjenta stworzonych zostało 44 cechy ciągłe.

Ten wzorzec został następnie przetworzony, aby uzyskać 22 wzorce cech binarnych. Algorytm CLIP3 został użyty do generowania reguł klasyfikacyjnych na podstawie tych wzorców. Reguły te osiągnęły dokładność na poziomie 84,0% w porównaniu z diagnozami kardiologów.

SPECT jest doskonałym zbiorem danych do testowania algorytmów uczenia maszynowego; zawiera 267 przypadków, opisanych 23 atrybutami binarnymi.

Zmienne:

OVERALL_DIAGNOSIS: 0,1 (atrybut klasowy, binarny)

F1-F22: 0,1 (cząstkowa diagnoza 1-22, binarnie)

Zbiór danych jest podzielony na:

Dane treningowe ("SPECT.train" - 80 przypadków)

Dane testowe ("SPECT.test" - 187 przypadków)

Diagnoza w bazie danych skupia się na analizie obrazów SPECT w celu określenia, czy pacjent jest zdrowy czy ma nieprawidłowości. Każdy przypadek jest opisany za pomocą 23 atrybutów binarnych, które reprezentują różne aspekty diagnozy serca. Algorytm CLIP3 był używany do generowania reguł klasyfikacyjnych, które osiągnęły wysoką dokładność w porównaniu z diagnozami ekspertów.

Działanie programu:

Tworzenie i analiza modelu

Nadanie nazw kolumnom + wczytanie danych z pliku

```
import pandas as pd
import numpy as np
import seaborn as sbn
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree , export_text
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
columns = ['target','F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11', 'F12', 'F13', 'F14',
           'F15', 'F16', 'F17', 'F18', 'F19', 'F20', 'F21', 'F22']

# Wczytanie danych z pliku
test_data = pd.read_csv("SPECT.test",header=None,names=columns)
train_data = pd.read_csv("SPECT.train",header=None,names=columns)
```

Wyświetlenie kilku pierwszych rekordów z danych treningowych

```
train_data.head()
```

	target	F1	F2	F3	F4	F5	F6	F7	F8	F9	...	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22
0	1	0	0	0	1	0	0	0	1	1	...	1	1	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	1	1	...	1	1	0	0	0	0	0	0	0	1
2	1	1	0	1	0	1	0	0	1	0	...	1	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	1	1
4	1	0	0	0	0	0	0	0	1	0	...	1	0	1	1	0	0	0	0	0	0

5 rows × 23 columns

Wyświetlenie statystyk opisowe dla wszystkich kolumn danych treningowych

```
print(train_data.describe())
```

```
          target         F1         F2         F3         F4         F5  \
count  80.000000  80.000000  80.000000  80.000000  80.000000  80.000000
mean    0.500000   0.362500   0.162500   0.262500   0.212500   0.300000
std     0.503155   0.483755   0.371236   0.442769   0.411658   0.461149
min     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
25%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
50%     0.500000   0.000000   0.000000   0.000000   0.000000   0.000000
75%     1.000000   1.000000   0.000000   1.000000   0.000000   1.000000
max     1.000000   1.000000   1.000000   1.000000   1.000000   1.000000

              F6         F7         F8         F9  ...        F13        F14  \
count  80.000000  80.000000  80.000000  80.000000  ...  80.000000  80.000000
mean    0.125000   0.262500   0.275000   0.187500  ...   0.350000   0.200000
std     0.332805   0.442769   0.449331   0.392775  ...   0.479979   0.402524
min     0.000000   0.000000   0.000000   0.000000  ...   0.000000   0.000000
25%     0.000000   0.000000   0.000000   0.000000  ...   0.000000   0.000000
50%     0.000000   0.000000   0.000000   0.000000  ...   0.000000   0.000000
75%     0.000000   1.000000   1.000000   0.000000  ...   1.000000   0.000000
max     1.000000   1.000000   1.000000   1.000000  ...   1.000000   1.000000

              F15        F16        F17        F18        F19        F20  \
count  80.000000  80.000000  80.000000  80.000000  80.000000  80.000000
mean    0.075000   0.175000   0.100000   0.075000   0.187500   0.225000
std     0.265053   0.382364   0.301893   0.265053   0.392775   0.420217
min     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
25%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
50%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
75%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
max     1.000000   1.000000   1.000000   1.000000   1.000000   1.000000

              F21       F22
count  80.000000  80.00000
mean    0.237500   0.32500
std     0.428236   0.47133
min     0.000000   0.00000
25%     0.000000   0.00000
50%     0.000000   0.00000
75%     0.000000   1.00000
max     1.000000   1.00000

[8 rows x 23 columns]
```

Jak możemy zauważyć w każdej kolumnie danych treningowych jest 80 niepustych wartości, co oznacza, że nie ma brakujących danych dla tych kolumn. To jest pozytywne, ponieważ kompletność danych jest ważna dla poprawnego funkcjonowania modelu.

Średnia z kolumny "target"=Overall Diagnosis równa 50% oznacza ze połowa przypadków ma pozytywną diagnozę (która jest reprezentowana przez 1), a druga połowa ma negatywną diagnozę (reprezentowaną przez 0).

Wyświetlenie kilku pierwszych rekordów z danych testowych

```
test_data.head()
```

	target	F1	F2	F3	F4	F5	F6	F7	F8	F9	...	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22
0	1	1	0	0	1	1	0	0	0	1	...	0	1	1	1	0	0	1	1	0	0
1	1	1	0	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	1	0	1	0	0	1	...	0	1	1	0	0	0	0	0	0	1
3	1	0	1	1	1	0	0	1	0	1	...	1	1	0	1	0	0	0	0	1	0
4	1	0	0	1	0	0	0	0	1	0	...	1	1	0	1	0	0	0	0	0	1

5 rows × 23 columns

Statystyki opisowe dla wszystkich kolumn danych testowych

```
print(test_data.describe())
```

```
           target          F1          F2          F3          F4          F5  \
count  187.000000  187.000000  187.000000  187.000000  187.000000  187.000000
mean     0.919786    0.481283    0.283422    0.449198    0.315508    0.449198
std      0.272353    0.500991    0.451870    0.498748    0.465965    0.498748
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      1.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

               F6          F7          F8          F9  ...         F13  \
count  187.000000  187.000000  187.000000  187.000000  ...  187.000000
mean     0.283422    0.294118    0.491979    0.363636  ...    0.556150
std      0.451870    0.456868    0.501278    0.482337  ...    0.498171
min      0.000000    0.000000    0.000000    0.000000  ...    0.000000
25%      0.000000    0.000000    0.000000    0.000000  ...    0.000000
50%      0.000000    0.000000    0.000000    0.000000  ...    1.000000
75%      1.000000    1.000000    1.000000    1.000000  ...    1.000000
max      1.000000    1.000000    1.000000    1.000000  ...    1.000000

              F14         F15         F16         F17         F18         F19  \
count  187.000000  187.000000  187.000000  187.000000  187.000000  187.000000
mean     0.347594    0.219251    0.368984    0.160428    0.155080    0.272727
std      0.477485    0.414850    0.483825    0.367988    0.362953    0.446557
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      1.000000    0.000000    1.000000    0.000000    0.000000    1.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

              F20         F21         F22
count  187.000000  187.000000  187.000000
mean     0.363636    0.417112    0.449198
std      0.482337    0.494405    0.498748
min      0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000
50%      0.000000    0.000000    0.000000
75%      1.000000    1.000000    1.000000
max      1.000000    1.000000    1.000000

[8 rows x 23 columns]
```

W zbiorze testowym wszystkie kolumny mają 187 rekordów, co oznacza, że analizowany zbiór danych składa się z 187 przypadków.

Kolumna "target" ma średnią równą 0.92. Oznacza to, że średnio 92% przypadków ma pozytywną diagnozę (wartość 1), co sugeruje, że większość przypadków diagnozy w zbiorze danych jest pozytywna(Pacjent posiada wadę serca).

Wyświetlenie procentowej ilości osób z chorobami serca w podanych zbiorach

```
label_column = 'target'

# Wyświetlenie procentowej ilości osób z chorobami serca w zbiorze testowym
percent_with_heart_disease_test = test_data[label_column].value_counts(normalize=True) * 100
print("Procentowa ilość osób z chorobami serca w zbiorze testowym:")
print(percent_with_heart_disease_test)

# Wyświetlenie procentowej ilości osób z chorobami serca w zbiorze treningowym
percent_with_heart_disease_train = train_data[label_column].value_counts(normalize=True) * 100
print("\nProcentowa ilość osób z chorobami serca w zbiorze treningowym:")
print(percent_with_heart_disease_train)
```

```
Procentowa ilość osób z chorobami serca w zbiorze testowym:
target
1    91.97861
0     8.02139
Name: proportion, dtype: float64

Procentowa ilość osób z chorobami serca w zbiorze treningowym:
target
1    50.0
0    50.0
Name: proportion, dtype: float64
```

Przygotowanie danych do uczenia modelu.

```
#przygotowanie danych do uczenia i analizy modelu

X_train = train_data.drop("target", axis=1)
y_train = train_data["target"]

X_test = test_data.drop("target", axis=1)
y_test = test_data["target"]
```

Uczenie modelu

```
# Uczenie modelu
model = DecisionTreeClassifier(random_state=17)
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=17)
```

Porównanie dokładności modelu na danych treningowych i danych testowych

```python
# Przewidywanie na danych treningowych
y_train_pred = model.predict(X_train)

# Przewidywanie na danych testowych
y_test_pred = model.predict(X_test)

# Przewidywanie na danych testowych
accuracy_test = accuracy_score(y_test, y_test_pred)

# Przewidywanie na danych treningowych
accuracy_train = accuracy_score(y_train, y_train_pred)

# Dokładność na danych testowych
print(f'Dokładność na danych treningowych: {accuracy_train}')
print(f'Dokładność na danych testowych: {accuracy_test}')
```

```
Dokładność na danych treningowych: 0.9375
Dokładność na danych testowych: 0.679144385026738
```

Dokładność na danych treningowych

Model osiągnął wysoką dokładność na danych treningowych (93.75%). Oznacza to, że dobrze dopasował się do danych, ale istnieje ryzyko, że istnieje mozliwośc overfittingu na tym konkretnym zestawie

Dokładność na danych testowych:

Dokładność na danych testowych jest niższa (67.91%). To może sugerować, że model może mieć trudności z generalizacją na nowe dane, co jest klasycznym objawem overfittingu.

Naszym celem jest uzyskanie modelu, który dobrze generalizuje na nowe dane, a nie tylko na danych treningowych dlatego spróbujemy znaleźć najlepsze dopasowanie dla modelu.

Overfitting

Overfitting w kontekście uczenia maszynowego, zwłaszcza dla drzew decyzyjnych, oznacza, że model dopasowuje się zbyt dokładnie do danych treningowych, uwzględniając nawet ich szum czy przypadkowe fluktuacje. W efekcie taki model może bardzo dobrze radzić sobie na danych treningowych, ale słabo generalizować swoje umiejętności do nowych, nieznanych danych.

Dla drzew decyzyjnych overfitting może wystąpić, gdy:

Głębokość drzewa jest zbyt duża:

Drzewo decyzyjne zbyt głębokie może idealnie dopasować się do każdej obserwacji treningowej, nawet do przypadkowego szumu, co prowadzi do nadmiernej złożoności

modelu. To zjawisko jest szczególnie widoczne, gdy drzewo ma bardzo wiele "pytań" (węzłów) i "odpowiedzi" (liści).

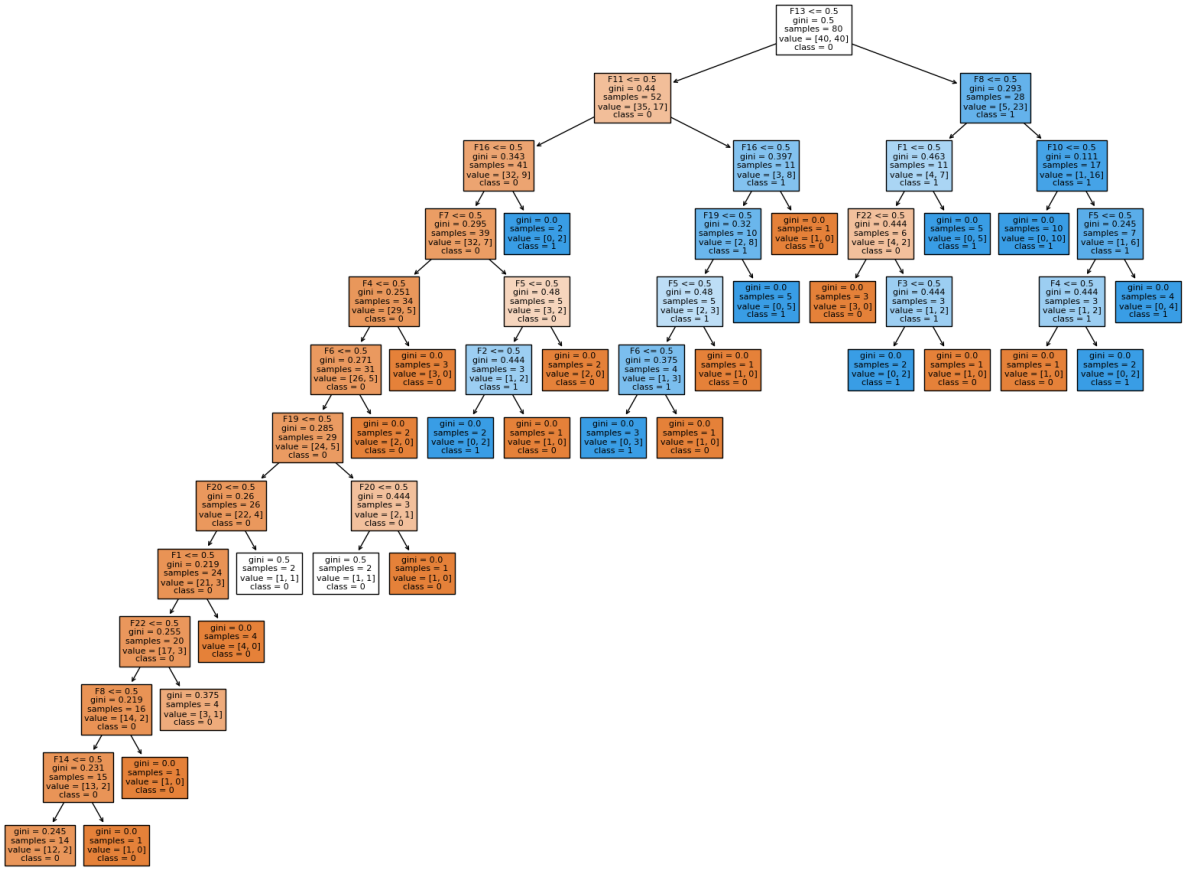
Gdy w zbiorze treningowym jest zbyt mała liczba próbek

Overfitting może być bardziej prawdopodobne, gdy mamy niewielką ilość danych treningowych, co sprawia, że model jest bardziej podatny na przypadkowe fluktuacje. Skutki overfittingu to ogólnie słaba zdolność generalizacji modelu do nowych danych, co prowadzi do niższej skuteczności na zbiorze testowym.

Aby przeciwdziałać overfittingowi, można zastosować techniki takie jak ograniczanie głębokości drzewa, stosowanie minimalnej liczby próbek w liściu, czy korzystanie z technik regularyzacji. Ważne jest także korzystanie z danych walidacyjnych w procesie treningu, aby monitorować wydajność modelu i dostosowywać jego parametry w trakcie uczenia.

Graficzna reprezentacja naszego drzewa decyzyjnego

```
plt.figure(figsize=(20, 15))
plot_tree(model, filled=True, feature_names=list(X_train.columns), class_names=list([str(c) for c in model.classes_]))
plt.show()
```



Wizualizacja ważności poszczególnych feature’ów

```
# Wizualizacja ważności cech:
feature_importances = model.feature_importances_
feature_names = X_train.columns
feature_importance_dict = dict(zip(feature_names, feature_importances))

sorted_features = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)
for feature, importance in sorted_features:
    print(f"{feature}: {importance}")
```

```
F13: 0.2691476989533146
F11: 0.1352284653659568
F16: 0.11264216652195373
F5: 0.07552195824334057
F1: 0.07330322665095886
F6: 0.048719839898903786
F4: 0.04462003571023612
F22: 0.0433405327573794
F3: 0.04146868250539957
F2: 0.04031677465802737
F8: 0.037525554105404216
F19: 0.029430176089891544
F7: 0.016865546683867163
F20: 0.015700282438943346
F10: 0.015161140049972476
F14: 0.0010079193664506673
F9: 0.0
F12: 0.0
F15: 0.0
F17: 0.0
F18: 0.0
F21: 0.0
```

Cechy, które mają większą ważność, są kluczowe dla decyzji modelu. W tym przypadku, F13, F11, i F16 wydają się być najbardziej istotne.

Wysoka ważność pewnych cech może skłaniać model do zbyt silnego uwzględniania szczegółów treningowych, co może pogorszyć jego zdolność do generalizacji na nowe dane. Wybór optymalnej głębokości drzewa i innych parametrów może być kluczowy dla uzyskania równowagi między precyzją a zdolnością do generalizacji.

Szukanie najlepszych parametrów dla naszego modelu

```
depths = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16 ]
criterions = ['gini', 'entropy']

for criterion in criterions:

    for depth in depths:
        model = DecisionTreeClassifier(criterion=criterion, max_depth=depth, random_state=17)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = np.mean(cross_val_score(model, X_train, y_train, cv=10, scoring='accuracy'))

        # Wyświetlenie wyników kroswalidacji
        print(f"Kryterium: {criterion}, Głębokość: {depth}")
        print(f"Średnia dokładność: {accuracy:.4f}")
        print("Macierz Konfuzji:")
        print(confusion_matrix(y_test, y_pred))
        print("\n")
```

Ten kod przeprowadza iteracyjną analizę różnych kombinacji kryteriów podziału (gini lub entropy) i głębokości drzewa decyzyjnego na danych testowych. Dla każdej kombinacji obliczana jest średnia dokładność modelu za pomocą kroswalidacji (cross-validation) z użyciem 10 podziałów „cv=10”. Ponadto, dla każdej kombinacji, wyświetlana jest macierz konfuzji na danych testowych.

Kod ten ma na celu znalezienie najlepszych kombinacji parametrów, które sprawią, że model drzewa decyzyjnego będzie skuteczny zarówno na danych treningowych, jaki danych testowych, co przyczynia się do jego ogólnej zdolności do generalizacji.

Wyniki:

Dla różnych głębokości z wykorzystaniem kryterium podziału ‘gini’:

```
Kryterium: gini, Głębokość: 1
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 13   2]
 [ 70 102]]

Kryterium: gini, Głębokość: 2
Średnia dokładność: 0.7375
Macierz Konfuzji:
[[ 12   3]
 [ 51 121]]

Kryterium: gini, Głębokość: 3
Średnia dokładność: 0.7250
Macierz Konfuzji:
[[ 12   3]
 [ 53 119]]

Kryterium: gini, Głębokość: 4
Średnia dokładność: 0.7375
Macierz Konfuzji:
[[ 11   4]
 [ 43 129]]

Kryterium: gini, Głębokość: 5
Średnia dokładność: 0.7125
Macierz Konfuzji:
[[ 12   3]
 [ 53 119]]

Kryterium: gini, Głębokość: 6
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 12   3]
 [ 54 118]]

Kryterium: gini, Głębokość: 7
Średnia dokładność: 0.6500
Macierz Konfuzji:
[[ 12   3]
 [ 54 118]]

Kryterium: gini, Głębokość: 8
Średnia dokładność: 0.6625
Macierz Konfuzji:
[[ 12   3]
 [ 57 115]]

Kryterium: gini, Głębokość: 9
Średnia dokładność: 0.6750
Macierz Konfuzji:
[[ 12   3]
 [ 54 118]]

Kryterium: gini, Głębokość: 10
Średnia dokładność: 0.6375
Macierz Konfuzji:
[[ 12   3]
 [ 56 116]]

Kryterium: gini, Głębokość: 11
Średnia dokładność: 0.6750
Macierz Konfuzji:
[[ 12   3]
 [ 59 113]]

Kryterium: gini, Głębokość: 12
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 12   3]
 [ 57 115]]

Kryterium: gini, Głębokość: 14
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 12   3]
 [ 57 115]]

Kryterium: gini, Głębokość: 16
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 12   3]
 [ 57 115]]
```

Dla różnych glebokosci z wykorzystaniem kryterium podzialu ‘entropy’:

```
Kryterium: entropy, Głębokość: 1
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 13   2]
 [ 70 102]]
```

```
Kryterium: entropy, Głębokość: 2
Średnia dokładność: 0.7375
Macierz Konfuzji:
[[ 12   3]
 [ 51 121]]
```

```
Kryterium: entropy, Głębokość: 3
Średnia dokładność: 0.7125
Macierz Konfuzji:
[[ 12   3]
 [ 53 119]]
```

```
Kryterium: entropy, Głębokość: 4
Średnia dokładność: 0.7375
Macierz Konfuzji:
[[ 11   4]
 [ 43 129]]
```

```
Kryterium: entropy, Głębokość: 5
Średnia dokładność: 0.7000
Macierz Konfuzji:
[[ 12   3]
 [ 49 123]]
```

```
Kryterium: entropy, Głębokość: 6
Średnia dokładność: 0.6750
Macierz Konfuzji:
[[ 12   3]
 [ 49 123]]
```

```
Kryterium: entropy, Głębokość: 7
Średnia dokładność: 0.6625
Macierz Konfuzji:
[[ 12   3]
 [ 53 119]]
```

```
Kryterium: entropy, Głębokość: 8
Średnia dokładność: 0.6875
Macierz Konfuzji:
[[ 13   2]
 [ 61 111]]
```

```
Kryterium: entropy, Głębokość: 9
Średnia dokładność: 0.6625
Macierz Konfuzji:
[[ 12   3]
 [ 49 123]]
```

```
Kryterium: entropy, Głębokość: 10
Średnia dokładność: 0.6625
Macierz Konfuzji:
[[ 12   3]
 [ 54 118]]
```

```
Kryterium: entropy, Głębokość: 11
Średnia dokładność: 0.6500
Macierz Konfuzji:
[[ 12   3]
 [ 52 120]]
```

```
Kryterium: entropy, Głębokość: 12
Średnia dokładność: 0.6250
Macierz Konfuzji:
[[ 12   3]
 [ 52 120]]
```

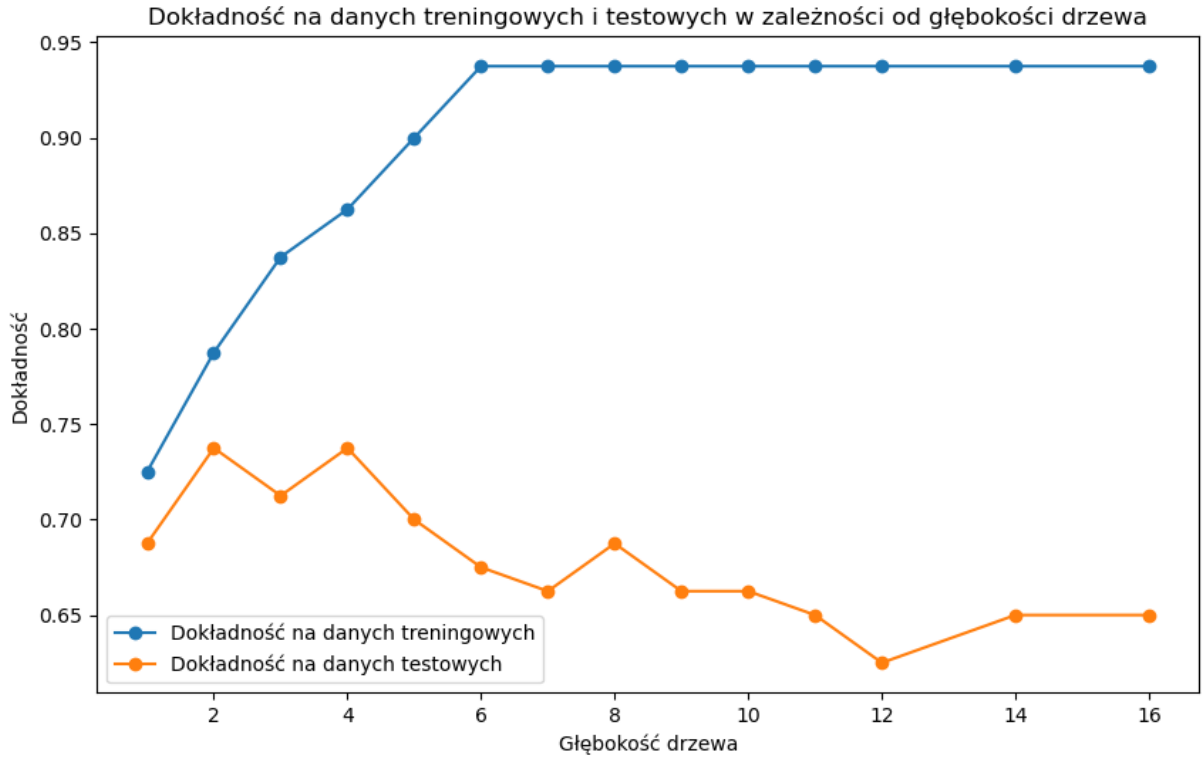
```
Kryterium: entropy, Głębokość: 14
Średnia dokładność: 0.6500
Macierz Konfuzji:
[[ 12   3]
 [ 54 118]]
```

```
Kryterium: entropy, Głębokość: 16
Średnia dokładność: 0.6500
Macierz Konfuzji:
[[ 12   3]
 [ 54 118]]
```

Przestawienie wyników dokładności modelu na wykresie i porównanie ich do dokładności na danych treningowych.

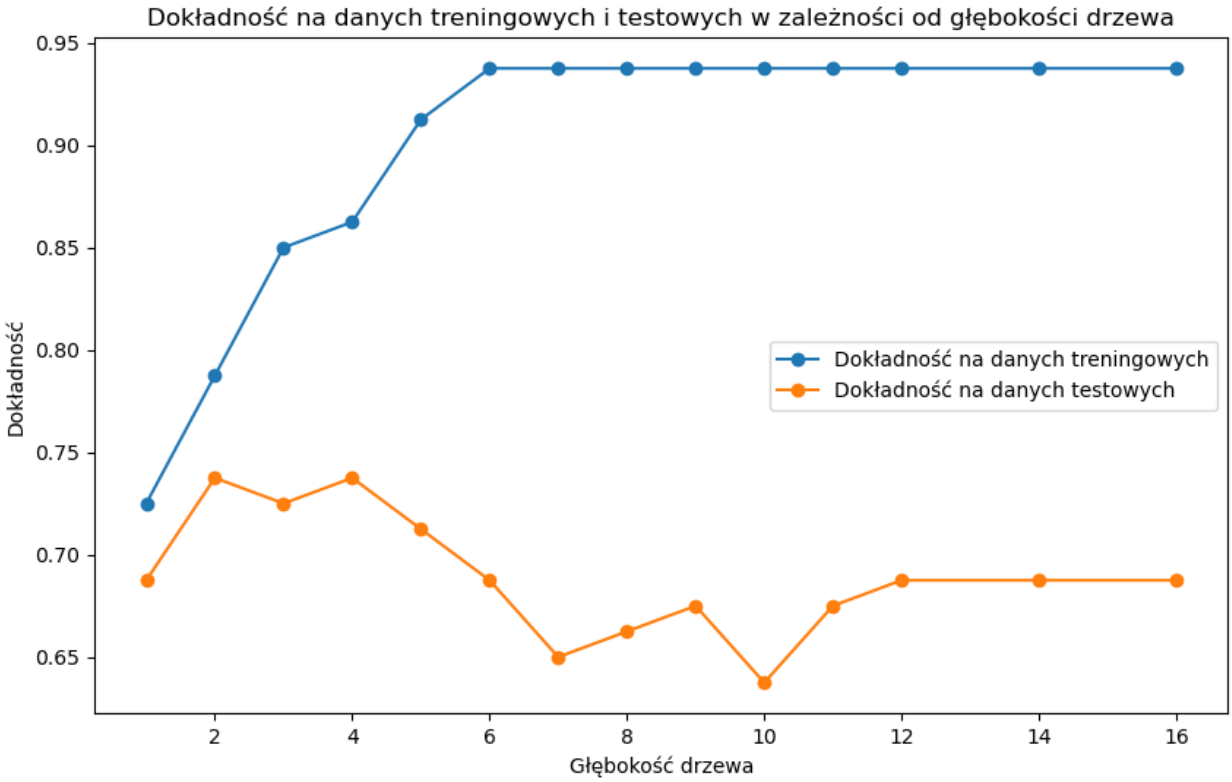
*Dla kryterium podzialu **entropy***

```
import matplotlib.pyplot as plt

depths = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16]
criterions = ['entropy']

train_accuracies = []
test_accuracies = []

for criterion in criterions:
    for depth in depths:
        model = DecisionTreeClassifier(criterion=criterion, max_depth=depth, random_state=4)
        model.fit(X_train, y_train)

        # Dokładność na danych treningowych
        train_pred = model.predict(X_train)
        train_accuracy = accuracy_score(y_train, train_pred)
        train_accuracies.append(train_accuracy)

        # Dokładność na danych testowych
        test_pred = model.predict(X_test)
        test_accuracy = accuracy_score(y_test, test_pred)
        test_accuracies.append(test_accuracy)

# Stworzenie wykresu
plt.figure(figsize=(10, 6))
plt.plot(depths * len(criterions), train_accuracies, label='Dokładność na danych treningowych', marker='o')
plt.plot(depths * len(criterions), test_accuracies, label='Dokładność na danych testowych', marker='o')
plt.xlabel('Głębokość drzewa')
plt.ylabel('Dokładność')
plt.title('Dokładność na danych treningowych i testowych w zależności od głębokości drzewa')
plt.legend()
```



*Dla kryterium podzialu **gini***

```
import matplotlib.pyplot as plt

depths = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16]
criterions = ['gini']

train_accuracies = []
test_accuracies = []

for criterion in criterions:
    for depth in depths:
        model = DecisionTreeClassifier(criterion=criterion, max_depth=depth, random_state=4)
        model.fit(X_train, y_train)

        # Dokładność na danych treningowych
        train_pred = model.predict(X_train)
        train_accuracy = accuracy_score(y_train, train_pred)
        train_accuracies.append(train_accuracy)

        # Dokładność na danych testowych
        test_pred = model.predict(X_test)
        test_accuracy = accuracy_score(y_test, test_pred)
        test_accuracies.append(test_accuracy)
# Stworzenie wykresu
plt.figure(figsize=(10, 6))
plt.plot(depths * len(criterions), train_accuracies, label='Dokładność na danych treningowych', marker='o')
plt.plot(depths * len(criterions), test_accuracies, label='Dokładność na danych testowych', marker='o')
plt.xlabel('Głębokość drzewa')
plt.ylabel('Dokładność')
plt.title('Dokładność na danych treningowych i testowych w zależności od głębokości drzewa')
plt.legend()
```



Wnioski:

Ogólna dokładność modelu drzewa decyzyjnego waha się między 0.62 a 0.74 %.

Obydwa kryteria, gini i entropy, wydają się prowadzić do podobnych wyników w kontekście średniej dokładności.

Nie ma znacznej różnicy w wynikach dla obu kryteriów, co sugeruje, że wybór kryterium nie jest kluczowym czynnikiem wpływającym na dokładność modelu.

W miarę zwiększania głębokości drzewa, średnia dokładność modelu zmienia się.

Najlepsza dokładność wydaje się być osiągana dla głębokości drzewa równego 4, niezależnie od wybranego kryterium.

Macierz konfuzji dla głębokości 4:

[[11 4]

[43 129]]

11 prawidłowych przewidywań klasy 0 - model poprawnie sklasyfikował 11 przypadków, gdzie Overall_diagnosis=0.

4 błędy klasyfikacji klasy 0, model błędnie sklasyfikował 4 przypadki wskazał Overall_diagnosis=0 , podczas gdy w rzeczywistości były one klasy 1.

43 błędy klasyfikacji klasy 1 - model błędnie sklasyfikował 43 przypadki Overall_diagnosis jako klasa 1, podczas gdy w rzeczywistości były one klasy 0.

129 prawidłowych przewidywań klasy 1 - Model poprawnie sklasyfikował 129 przypadków, gdzie rzeczywiście Overall_diagnosis=1.

Dla głębokości większych niż 4, obserwujemy tendencję do spadku dokładności, co sugeruje możliwość wystąpienia nadmiernego dopasowania (overfittingu). Dlatego przy uczeniu modelu sugerujemy wybór głębokości = 4 ponieważ wtedy otrzymujemy najlepsze dopasowanie dla danych testowych oraz zapobiegamy overfittingowi .

Podsumowując

Model drzewa decyzyjnego, w tym przypadku z algorytmem ID3, okazał się dosyć skutecznym narzędziem w diagnozowaniu wad serca. Optymalizacja parametrów oraz analiza jakości modelu były kluczowe dla uzyskania satysfakcjonujących wyników klasyfikacji.
Dokładność osiągnięta przez nasz model wynosi około 74%. Pomimo że wypada nieco gorzej w porównaniu z wynikami na zbiorze treningowym około (94%), nadal sugeruje, że metoda ta może być użyteczna w dziedzinie medycyny .
Aby zminimalizować szanse overfittingu, podjęliśmy kilka kluczowych kroków. Wykorzystaliśmy technikę kroswalidacji, co pozwoliło na obiektywną ocenę skuteczności modelu na różnych podzbiorach danych. Ponadto, przeprowadziliśmy testowanie różnych głębokości drzewa decyzyjnego, co pozwoliło nam dostosować model do danych, minimalizując ryzyko zbytniego dopasowania do zbioru treningowego.

W rezultacie uzyskany model wydaje się być stabilny i zdolny do generalizacji na nowe dane, co sprawia, że jest obiecującym narzędziem w kontekście diagnozowania wad serca.

BIBLIOGRAFIA

https://www.cs.put.poznan.pl/pboinski/files/ED/ED_slajdy_drzewa.pdf

https://towardsai.net/p/programming/decision-trees-explained-with-a-practical-example-fe47872d3b53

https://deepnote.com/@yura-ueno/Decision-Tree-and-Random-Forest-d6026e7d-09ab-44b3-8960-16878db21a3f

https://www.datacamp.com/tutorial/decision-tree-classification-python