Porównanie metod klasyfikacji danych

Karolina Maruszak

Marzec 2021

Spis treści

Wstęp

Klasyfikacja jest podejściem uczenia nadzorowanego. Jest to proces kategoryzowania danego zestawu danych na klasy. Może być przeprowadzany zarówno na danych ustrukturyzowanych, jak i nieustrukturyzowanych. Innymi słowy jest to proces porządkowania i analizowania danych według odpowiednich kategorii. Co więcej obejmuje tagowanie danych w celu ułatwienia ich wyszukiwania i śledzenia, jak również eliminuje wielokrotne duplikacje danych.

Niewątpliwie istnieje wiele metod klasyfikacji danych, jednak w dokumencie tym uwaga zostanie poświęcona głównie drzewom decyzyjnym, a dokładniej porównaniu metod w oparciu o pojedyncze drzewo z algorytmami budującymi lasy losowe.

Rozdział 1

Drzewa decyzyjne

1.1 Definicja

Drzewa decyzyjne to nieparametryczna metoda uczenia nadzorowanego, stosowana do klasyfikacji i regresji. Celem jest stworzenie modelu, który przewiduje wartość zmiennej docelowej poprzez poznanie prostych reguł decyzyjnych wywnioskowanych na podstawie cech danych. Im głębsze drzewo, tym bardziej złożone reguły decyzyjne i dopasowany model.

Algorytm drzewa decyzyjnego buduje model klasyfikacyjny w postaci struktury drzewiastej. Drzewa te są zbudowane z dwóch rodzajów elementów: węzłów i gałęzi. Węzły środkowe (wewnętrzne) mają dwie lub więcej gałęzi. Następuje w nich wartościowanie wybranej zmiennej. Z kolei liście, są to końcowe węzły drzewa, w których dochodzi do predykcji kategorii. Natomiast korzeń drzewa (węzeł główny) ewaluuje zmienną, która najlepiej dzieli dane. Warto również wspomnieć o tym, że drzewa decyzyjne są konstruowane poprzez rekurencyjne ewaluowanie różnych cech i używanie w każdym węźle cechy, która najlepiej dzieli dane.

Popularność drzew decyzyjnych wynika z kilku czynników. Przede wszystkim są one proste do zrozumienia i zinterpretowania, ponieważ drzewa można wizualizować. Co więcej wymagają niewielkiego przygotowania danych (niektóre techniki często wymuszają normalizację danych). Drzewa decyzyjne mogą obsługiwać zarówno dane jakościowe, jak i liczbowe. Plusem jest również możliwość walidacji modelu za pomocą testów statystycznych. To umożliwia uwzględnienie niezawodności modelu. Drzewa decyzyjne mogą także używać zapytań SQL do uzyskiwania dostępu do baz danych.

Drzewa decyzyjne mają też swoje wady. Między innymi może dojść do tworzenia zbyt złożonych drzew, które nie generalizują dobrze danych. Nazywa się to nadmiernym dopasowaniem. Aby uniknąć tego problemu, konieczne są mechanizmy takie jak przycinanie, ustalanie minimalnej liczby próbek wymaganych w węźle liścia lub

ustawianie maksymalnej głębokości drzewa.

1.2 Założenia podczas tworzenia drzewa decyzyjnego

- Na początku cały zbiór uczący jest traktowany jako root.
- Atrybuty są kategorialne. Jeśli wartości są ciągłe, są dyskretyzowane przed zbudowaniem modelu.
- Przykłady są partycjonowane rekurencyjnie na podstawie wartości atrybutów.
- Atrybuty testu są wybierane na podstawie miary heurystycznej lub statystycznej.

Decyzja o dokonaniu strategicznych podziałów ma duży wpływ na dokładność drzewa. Kryteria decyzyjne są różne dla drzew klasyfikacyjnych i regresyjnych. Drzewa decyzyjne używają wielu algorytmów do podejmowania decyzji o podzieleniu węzła na dwa lub więcej węzłów podrzędnych. Drzewo decyzyjne dzieli węzły na wszystkie dostępne zmienne, a następnie wybiera podział, który daje najbardziej jednorodne podwęzły.

1.3 Kryteria podziału

Entropia:

Entropia jest miarą losowości przetwarzanych informacji. Im wyższa entropia, tym trudniej wyciągnąć wnioski z tej informacji. Dla zbioru punktów danych S, wyliczana jest z następującego wzoru:

$$E(S) = -\sum_{i=1}^{k} p_i \log_2 p_i \tag{1.1}$$

gdzie:
$\{p_1 \dots p_k\}$ - rozkład klas dla k-klas ze zbioru S

Ogólna entropia podziału r dla cechy f wyliczana jest następująco[1]:

$$E_f(S \implies S_1, \dots, S_r) = \frac{\Sigma_{i=1}^{r} |S_i| E(S_i)}{|S|} \tag{1.2}$$

W przypadku tego kryterium, wybierany jest podział o najniższej entropii (algorytm ID3, C4.5).

Zysk informacji:

Zysk informacji (z ang. information gain) służy do dzielenia węzłów, gdy zmienna docelowa jest kategoryczna. Mierzy on jak dobrze dany atrybut oddziela przykłady treningowe, zgodnie z ich klasyfikacją docelową. Działa na koncepcji entropii. A dokładniej suma entropii wyjściowych ważona rozmiarem podzbiorów, jest odejmowana od entropii zbioru danych przed podziałem. Ta różnica mierzy właśnie zysk informacji.

$$IG_f(S) = E(S) - E_f(S \implies S_1, \ldots, S_r) \tag{1.3}$$

W przypadku tego kryterium, atrybut o maksymalnym zysku informacji wybierany jest jako atrybut podziału.

Względny zysk informacji:

Zmniejsza wpływ atrybutów o dużej liczbie wartości wprowadzając koncepcję z ang. Split Info. Definiuje się to jako sumę wag pomnożoną przez logarytm wag, gdzie wagi są stosunkiem liczby punktów danych w bieżącym podzbiorze do liczby punktów danych w nadrzędnym zbiorze danych. Widać to na poniższym wzorze:

$$I_f(S \implies S_1, \ldots, S_r) = -\sum_{i=1}^{r} \frac{|S_i|}{|S|} \log_2(\frac{|S_i|}{|S|}) \tag{1.4}$$

Zatem wzór na względny zysk informacji ma postać:

$$IGR_f = \frac{IG_f(S)}{I_f(S \implies S_1, \ldots, S_r)} \tag{1.5}$$

W przypadku tego kryterium, atrybut o maksymalnym względnym zysku informacji wybierany jest jako atrybut podziału.

Gini index:

Indeks Giniego oblicza się, odejmując sumę kwadratów prawdopodobieństw każdej klasy od jedności. Załóżmy, że:
$\{p_1 \ldots p_k\}$ - rozkład klas dla k-klas ze zbioru S
To wtedy Gini index dla zbioru punktów S obliczany jest następująco:

$$G(S) = 1 - \sum_{i=1}^{r} p_i^2 \tag{1.6}$$

Wybierany jest podział z najniższym indeksem Giniego (algorytm CART)

1.4 ID3

Pierwszym algorytmem któremu się przyjrzymy, będzie ID3. Algorytm ten buduje drzewa decyzyjne przy użyciu podejścia zachłannego. Algorytm zachłanny, jak sama nazwa wskazuje, zawsze dokonuje wyboru, który w danym momencie wydaje się najlepszy. Co więcej przy wyborze atrybutów algorytm ID3 opiera się na wyliczaniu entropii (1.1) i zysku informacji (1.3).

Kolejne kroki algorytmu:

- Algorytm zaczyna się od oryginalnego zestawu jako węzła głównego (root).
- Następnie dokonuje obliczenia entropii i zysku informacji dla każdego atrybutu.
- Wybiera atrybut, który ma najmniejszy przyrost entropii i największy zysk informacji.
- Cały zbiór jest następnie dzielony według wybranego atrybutu w celu utworzenia podzbioru danych.
- Algorytm powtarza się w każdym podzbiorze, biorąc pod uwagę tylko atrybuty nigdy wcześniej nie wybrane.

Niestety główną wadą algorytmu ID3 jest nadmierne rozrastanie się drzewa. Co więcej algorytm ten jest trudniejszy w użyciu w przypadku danych ciągłych i brakujących danych.

1.5 C4.5

C4.5 jest rozszerzeniem wcześniej omawianaego algorytmu ID3. Używa on względnego zysku informacji jako kryterium podziłu (1.5). John Ross Quinlan wprowadzając ten algorytm, wyszedł naprzeciw ograniczeniom, które miały miejsce przy ID3.

Po pierwsze, algorytm usunął ograniczenie, że cechy muszą być kategorialne poprzez dynamiczne definiowanie atrybutu dyskretnego, który dzieli ciągłą wartość atrybutu na dyskretny zbiór przedziałów. Zatem obsługuje zarówno atrybuty ciągłe, jak i dyskretne.

Co więcej algorytm ten radzi sobie z klasyfikacją danych treningowych z brakującymi wartościami.

Kolejna modyfikacja dotyczy zmniejszania wpływu atrybutów o dużej liczbie wartości, poprzez stosowanie względnego zysku informacji (co zostało wspomniane przy definiowaniu wspomnianego kryterium podziału).

Bardzo ważną możliwością, którą zapewnia C4.5 jest przycinanie drzewa. Metoda ta stosowana jest w celu zapobiegania nadmiernemu dopasowaniu drzewa. Polega na oszacowaniu wskaźnika błędów każdego poddrzewa i zastąpieniu poddrzewa węzłem liścia, jeśli szacowany błąd liścia jest niższy. [6] Idea wygląda następująco: załóżmy, że można oszacować współczynnik błędów dowolnego węzła w drzewie decyzyjnym, w tym węzłów liści. Zaczynając od dołu drzewa, jeśli szacunki wskazują, że drzewo będzie dokładniejsze, gdy dzieci węzła n zostaną usunięte, a n zostanie węzłem liścia, to wtedy C4.5 dokona usunięcia dzieci n. Gdyby oszacowania były doskonałe, zawsze prowadziłoby to do lepszego drzewa decyzyjnego. W praktyce, mimo że te szacunki są bardzo zgrubne, metoda często działa całkiem dobrze.

1.6 C5.0

Jednym z najbardziej znanych algorytmów do implementacji drzew decyzyjnych jest algorytm C5.0 (następca C4.5). W porównaniu z bardziej zaawansowanymi i wyrafinowanymi modelami uczenia maszynowego, drzewa decyzyjne algorytmu C5.0 generalnie działają prawie równie dobrze, ale są znacznie łatwiejsze do zrozumienia i wdrożenia. Algorytm C5.0 używa entropii jako kryterium podziału.

Porównajmy oba algorytmy C4.5 oraz C5.0. Warto wspomnieć o tym, że w C4.5 wszystkie błędy są traktowane jako równe, ale w praktycznych zastosowaniach niektóre błędy klasyfikacyjne są poważniejsze niż inne. C5.0 umożliwia zdefiniowanie oddzielnego kosztu dla każdej przewidywanej / rzeczywistej pary klas.

C5.0 ma kilka nowych typów danych oprócz tych dostępnych w C4.5, w tym daty, godziny, sygnatury czasowe, uporządkowane atrybuty dyskretne i etykiety przypadków. Oprócz brakujących wartości, C5.0 umożliwia oznaczenie wartości jako nie mających zastosowania. Ponadto C5.0 udostępnia narzędzia do definiowania nowych atrybutów jako funkcji innych atrybutów.

1.7 CART

CART(Classification and Regression Trees) to algorytm wprowadzony przez Leo Breimana. Jak sama nazwa wskazuje, może być używany zarówno do klasyfikacji i regresji. Algorytm CART stanowi podstawę dla ważnych algorytmów, takich jak bagging dla drzew decyzyjnych, lasów losowych czy boostingu. Co więcej drzewo CART to binarne drzewo decyzyjne, które jest tworzone przez wielokrotne dzielenie węzła na dwa węzły potomne, zaczynając od węzła głównego, który zawiera całą próbkę treningową. Algorytm ten wykorzystuje kryterium podziału o nazwie gini index (1.6) do

tworzenia punktów decyzyjnych dla zadań klasyfikacyjnych.

1.8 CHAID

CHAID (Chi-square Automatic Interaction Detector) wykorzystuje testy chi-kwadrat, aby znaleźć najbardziej dominującą cechę. Jest on najstarszym algorytmem do budowy drzew decyzyjnych w historii. Dla każdej wybranej pary wartości CHAID sprawdza, czy uzyskana wartość p jest większa niż określony próg scalania. Jeśli odpowiedź jest pozytywna, łączy wartości i szuka dodatkowej potencjalnej pary do połączenia. Proces powtarza się, dopóki nie zostaną znalezione żadne znaczące pary. Następnie wybierany jest najlepszy atrybut wejściowy, który ma być użyty do podziału bieżącego węzła, tak że każdy węzeł potomny składa się z grupy jednorodnych wartości wybranego atrybutu.

Co więcej CHAID obsługuje brakujące wartości, traktując je wszystkie jako jedną prawidłową kategorię.Algorytm ten nie wykonuje przycinania.

1.9 MSI-Minimum Surfeit and Inaccuracy

Drzewa MSI to zestaw modeli uczenia maszynowego opartych na algorytmie drzewa decyzyjnego. Główna różnica w stosunku do innych metod polega na tym, że nie ma hiperparametrów do optymalizacji dla podstawowego klasyfikatora. Drzewo jest regulowane wewnętrznie, aby uniknąć nadmiernego dopasowania. Aby to osiągnąć, algorytm musi automatycznie zrozumieć, kiedy rosnące drzewo decyzyjne zwiększa niepotrzebną złożoność i musi dokonywać pomiarów takiej złożoności w sposób współmierny do niektórych aspektów jakości predykcji, np. niedokładności. W tym celu używana jest koncepcja złożoności Kołmogorowa.

1.10 Rain Forest

Podstawowe algorytmy drzew decyzyjnych działają dobrze, gdy zestaw danych mieści się w pamięci głównej. Jednak z powodu tego, że zbiory danych zwykle zwiększają się szybciej niż pamięć komputera, to drzewa decyzyjne często muszą być konstruowane na podstawie zestawów danych znajdujących się na dysku. Istnieją pewne wyzwania związane ze skalowaniem algorytmu konstrukcji drzewa decyzyjnego do obsługi dużych zbiorów danych. W 1998 roku Gehrke zaproponował framework o nazwie RainForest do skalowania konstrukcji drzewa decyzyjnego. Można go używać z

dowolnymi kryteriami podziału. Kiedy mówimy o skalowaniu, mamy oczywiście na myśli możliwość efektywnego konstruowania modelu przy dużej ilości danych.

[5] Rain Forest skaluje budowę drzewa decyzyjnego do większych zbiorów danych, jednocześnie efektywnie wykorzystując dostępną pamięć główną. Odbywa się to poprzez wyodrębnienie zestawu AVC (Attribute-Value, Classlabel) dla danego atrybutu i przetwarzanego węzła. Zestaw AVC dla atrybutu po prostu rejestruje liczbę wystąpień każdej etykiety klasy dla każdej odrębnej wartości, jaką może przyjąć atrybut. Rozmiar zbioru AVC dla danego węzła i atrybutu jest proporcjonalny do iloczynu liczby odrębnych wartości atrybutu z liczbą różnych etykiet klas. Zestaw AVC można skonstruować przez wykonanie jednego przejścia przez rekordy uczące skojarzone z węzłem.

Każdy węzeł posiada grupę AVC, która z kolei jest zbiorem zestawów AVC dla wszystkich atrybutów. Kluczową obserwacją jest to, że chociaż grupa AVC nie zawiera wystarczających informacji, aby zrekonstruować zbiór danych uczących, to zawiera wszystkie informacje niezbędne do wyboru kryterium podziału węzła. Rain Forest obejmuje szereg algorytmów wykorzystywnych do podziału węzłów drzew decyzyjnych na niższych poziomach (przy założeniu, że cała grupa AVC węzła głównego mieści się w pamięci głównej oraz grupy AVC każdego kolejnego węzła również mieszczą się w pamięci głównej):

- RF-read (algorytm odczytu) - w algorytmie tym zbiór danych nigdy nie jest podzielony. Algorytm postępuje poziom po poziomie. W pierwszym kroku budowana jest grupa AVC dla węzła głównego i wybierane są kryteria podziału. Na każdym z niższych poziomów wszystkie węzły na tym poziomie są przetwarzane w jednym przejściu, jeśli grupa AVC dla wszystkich węzłów mieści się w pamięci głównej. Jeśli nie, wykonywane są wielokrotne przejścia przez zbiór danych wejściowych, aby podzielić węzły na tym samym poziomie drzewa. Ponieważ zbiór danych uczących nie jest podzielony na partycje, może to oznaczać wielokrotne odczytywanie każdego rekordu dla jednego poziomu drzewa.

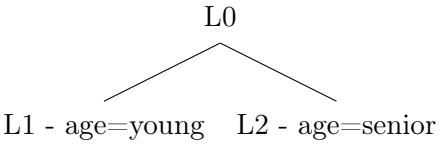
- RF-write (algorytm zapisu) - algorytm ten dzieli i przepisuje zbiór danych po każdym przejściu. Co więcej zestaw treningowy jest czytany dwukrotnie i zapisywany raz [3].

- RF-hybrid - algorytm ten łączy dwa poprzednie algorytmy. Co więcej algorytm RF-read oraz RF-hybrid są w stanie wykorzystać dostępną pamięć główną do przyspieszenia obliczeń, ale bez wymagania, aby zestaw danych znajdował się w pamięci głównej.

Przykład:

Dane treningowe:

ID	AGE	INCOME	CLASS
1	young	65	G
2	young	15	B
3	young	75	G
4	senior	40	B
5	senior	100	G
6	senior	60	G



Zbiór AVC age dla L0:

VALUE	CLASS	COUNT
young	B	1
young	G	2
senior	B	1
senior	G	2

Zbiór AVC income dla L0:

VALUE	CLASS	COUNT
15	B	1
40	B	1
60	G	1
65	G	1
75	G	1
100	G	1

Zbiór AVC age dla L1:

VALUE	CLASS	COUNT
young	B	1
young	G	2

Zbiór AVC income dla L1:

VALUE	CLASS	COUNT
15	B	1
65	G	1
75	G	1

1.11 BOAT

W tej sekcji zaprezentowany zostanie algorytm konstrukcji drzewa decyzyjnego, BOAT(Bootstrapped Optimistic Algorithm for Tree Construction), który przyjmuje zupełnie inne podejście do skalowalności - nie jest oparty na wykorzystaniu żadnych specjalnych struktur danych. Zamiast tego wykorzystuje technikę statystyczną znaną z ang. jako bootstrapping w celu utworzenia kilku mniejszych próbek (lub podzbiorów) danych uczących, z których każda mieści się w pamięci. Każdy podzbiór jest

używany do konstruowania drzewa, w wyniku czego powstaje kilka drzew. Są one badane i używane do konstruowania nowego drzewa (drzewo to jest bardzo zbliżone do drzewa, które zostałoby wygenerowane przy użyciu całego zestawu danych).

Podobnie jak omówiony wcześniej Rain Forest, BOAT ma zastosowanie do szerokiej gamy metod podziału. [4] Jest to pierwszy algorytm konstrukcji drzewa decyzyjnego, który dokonuje dwóch skanów całego zbioru danych. Inne algorytmy dokonują pojedynczego skanowania dla każdego poziomu z osobna.

Okazało się, że BOAT jest dwa do trzech razy szybszy niż RainForest, podczas konstruowania dokładnie tego samego drzewa. Dodatkową zaletą BOAT jest to, że można go używać do aktualizacji przyrostowych. Oznacza to, że BOAT może pobierać nowe wstawienia i usunięcia danych uczących i aktualizować drzewo decyzyjne, aby odzwierciedlić te zmiany, bez konieczności rekonstruowania drzewa od podstaw.

1.12 Stosowane biblioteki języka Python do implementacji drzew decyzyjnych

Popularną biblioteką do implementacji klasyfikatorów w Pythonie jest biblioteka Scikit-Learn. Przyjrzyjmy się dostępnym klasyfikatorom i metodom. które zapewnia nam wspomniana biblioteka.

DecisionTreeClassifier jest to klasyfikator drzewa decyzyjnego. Innymi słowy jest to klasa zdolna do wykonywania klasyfikacji wieloklasowej na zestawie danych.
Podobnie jak w przypadku innych klasyfikatorów, DecisionTreeClassifier przyjmuje jako dane wejściowe dwie tablice: tablicę X, o kształcie (ilość próbek, ilość cech) zawierającą próbki uczące oraz tablicę Y zawierającą wartości całkowite, kształt (ilość próek,), zawierającą etykiety klas dla danych treningowych.
Co więcej, DecisionTreeClassifier posiada parametr criterion, w którym należy określić kryterium podziału. Obsługiwane są gini oraz entropia. Defaultowym kryteirum jest gini.

Jednym z problemów jakie można spotkać przy klasyfikacji jest fakt, że część algorytmów nie jest przystosowana do pracy na danych kategorycznych. Co zrobić w takiej sytuacji? Z pomocą przychodzi kodowanie One-Hot-Encoding, które stosowane jest właśnie do danych kategorycznych przekształcając je na dane numeryczne poprzez ich binarne kodowanie. Dzięki temu algorytmy, które normalnie działają na danych numerycznych bez problemu poradzą sobie z przetwarzaniem takich danych.
OneHotEncoder dostępny jest w bibliotecte sklearn.preprocessing.

W przypadku klasyfikacji, należy podzielić dane na treningowe i testowe. Jest do tego dostępna metoda: sklearn.model_selection.train_test_split.

Niestety nie ma jednej biblioteki w Pythonie, która obsługuje powyższe omawiane algorytmy. Szczególnie we wspomnianej bibliotece scikit-learn nie znajdziemy gotowych funckji na implementację poszczególnych algorytmów drzew decyzyjnych. Dostępny jest jednak framework o nazwie Chefboost, który obsługuje algorytmy: ID3, C4.5, CART, CHAID, lasy losowe, boosting. Dla drzew MSI dostępny jest pakiet msitrees, który implementuje MSIDecisionTreeClassifier().

Rozdział 2

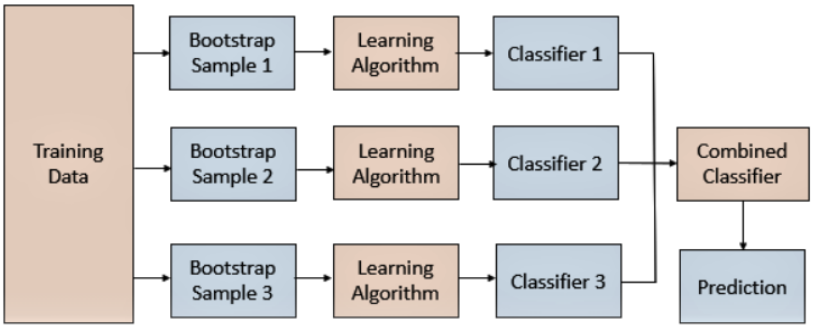
Boosting i Bagging

Boosting i Bagging, to techniki służące do zwiększania dokładności klasyfikacji. Dokładniej, łączą kilka drzew decyzyjnych w celu uzyskania lepszej wydajności predykcyjnej niż wykorzystanie pojedynczego drzewa decyzyjnego. Z tego powodu należą one do metod zespołowych (z ang. ensemble methods) czyli takich, które tworzą wiele modeli, a następnie łączą je w celu uzyskania lepszych wyników. Metody zespołowe zwykle dają dokładniejsze rozwiązania niż pojedynczy model.

2.1 Bagging

Metodę zaproponował Leo Breiman w 1994 roku, aby zapobiec nadmiernemu dopasowaniu ("przetrenowaniu"). Jest ona oparta na bootstrappingu i agregacji, tak więc uwzględnia zalety obu podejść. Stąd też, technika ta inaczej nazywana jest z ang. Bootstrap Aggregation. Metoda baggingu jest skuteczna w przypadku mniejszych zestawów danych. Jest stosowana w drzewach decyzyjnych, gdzie znacząco podnosi stabilność modeli w redukcji wariancji i poprawie dokładności, co (tak jak wyżej zostało wspomniane) eliminuje problem overfittingu.

Idea metody jest zatem prosta: chcemy dopasować kilka niezależnych modeli i otrzymać model o niższej wariancji. Jednak w praktyce nie możemy dopasować w pełni niezależnych modeli, ponieważ wymagałoby to zbyt dużej ilości danych. Dlatego polegamy na dobrych „przybliżonych właściwościach” próbek typu bootstrap, aby dopasować modele, które są prawie niezależne. Najpierw tworzymy wiele próbek bootstrap, aby każda nowa próbka bootstrap działała, jako kolejny (prawie) niezależny zestaw danych. Następnie możemy dopasować słaby klasyfikator do każdej z tych próbek. Ostateczny wynik opiera się o głosowanie większościowe klasyfikatorów. Poniższy diagram pokazuje działanie omawianej metody[7]:



Rozszerzeniem techniki baggingu jest algorytm lasów losowych, któremu dokładniej przyjrzymy się w kolejnym rozdziale.

2.2 Boosting

Technika boostingu działa podobnie do metody baggingu: tworzymy rodzinę modeli, które są agregowane, aby uzyskać silny klasyfikator, który osiąga lepsze wyniki. Jednak w przeciwieństwie do baggingu, które ma na celu głównie zmniejszenie wariancji, boosting jest iteracyjną techniką polegającą na dopasowywaniu sekwencyjnie wielu słabych klasyfikatorów w bardzo adaptacyjny sposób: każdy model w sekwencji jest dopasowany, nadając większą wagę obserwacjom w zbiorze danych, które były niepoprawnie przewidywane przez poprzednie klasyfikatory (czyli za każdym razem źle przewidzianym obserwcjom zwiększana jest waga). Tak więc każdy kolejny klasyfikator będzie koncentrował swoje wysiłki na trudniejszych do przewidywania obserwacjach, tak aby na końcu procesu otrzymać silny klasyfikator.

Jednym z podstawowych algorytmów boostingu jest AdaBoost, który oczywiście pozwala na łączenie wielu słabych klasyfikatorów w jeden silny klasyfikator. Słabymi klasyfikatormai w omawianym algorytmie, są drzewa decyzyjne z jednym poziomem. Ponieważ drzewa te są tak krótkie i zawierają tylko jedną decyzję do klasyfikacji, często nazywane są pniakami decyzyjnymi (decision stumps). AdaBoost może być zarówno stosowany do klasyfikacji jak i regresji.

Rozdział 3

Random Forest

3.1 Definicja

Random Forest, opracowany przez Leo Breimana [2], to algorytm nadzorowanego uczenia maszynowego oparty o strukturę drzewa, który wykorzystuje moc wielu drzew decyzyjnych do podejmowania decyzji. Dokładniej mówiąc łączy dane wyjściowe wielu (losowo utworzonych) drzew decyzyjnych w celu wygenerowania ostatecznego wyniku. Oznacza to, że zamiast polegać na jednym drzewie decyzyjnym, las losowy bierze predykcję z każdego drzewa i na podstawie większości głosów przewiduje ostateczny wynik. Co więcej większa liczba drzew w lesie prowadzi do większej dokładności i zapobiega problemowi przeuczenia. Warto dodać również, że algorytm lasu losowego jest rozszerzeniem metody baggingu, ponieważ wykorzystuje zarówno bagging, jak i losowość cech do tworzenia nieskorelowanego lasu drzew decyzyjnych.

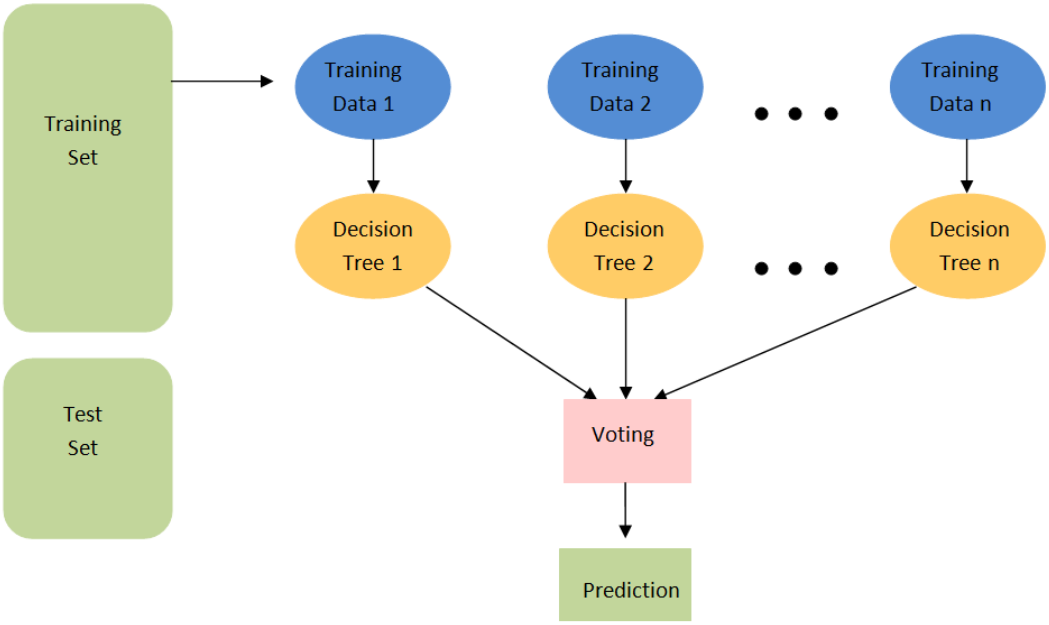
Lasy losowy definiowany jest formalnie w następujący sposób [2]: jest to klasyfikator, który składa się ze zbioru klasyfikatorów o strukturze drzewiastej $\{h_k(x, T_k)\}, k = 1, 2, \ldots, L$, gdzie T_k to niezależne próbki losowe o identycznym rozkładzie, a każde drzewo oddaje głos jednostkowy na najpopularniejszą klasę na wejściu x.

Działanie algorytmu prezentuje się następująco:

1. Zaczynamy od wyboru losowych próbek z danego zbioru danych.
2. Następnie algorytm konstruuje drzewo decyzyjne dla każdej próbki z osobna. Wtedy otrzyma wynik predykcji z każdego drzewa decyzyjnego.
3. Na tym etapie głosowanie zostanie przeprowadzone dla każdego przewidywanego wyniku.

4. Na koniec należy wybrać jako ostateczny, wynik predykcji z największą liczbą głosów.

Można zauważyć to również na poniższym diagramie:



3.2 Zalety i wady

Niewątpliwie algorytm Random Forest ma swoje zalety i wady. Zacznijmy od tych pierwszych. Przede wszystkim to o czym mowa była już wcześniej: algorytm lasu losowego zmniejsza nadmierne dopasowanie w drzewach decyzyjnych i pomaga poprawić dokładność. Co więcej, działa zarówno dla problemów klasyfikacyjnych jak i regresyjnych. Nie można oczywiście zapomnieć o tym, że algorytm ten działa dobrze zarówno z wartościami kategorycznymi, jak i ciągłymi. To nie koniec zalet jakie niesie za sobą Random Forest. Jest on w stanie nie tylko obsługiwać duże zbiory danych o dużej wymiarowości, ale także automatyzuje brakujące wartości, ktore są obecne w danych.

Niestety pomimo tych wszystkich pozytywnych właściwości, algorytm lasu losowego niesie za sobą również wady. Przede wszystkim wymaga on dużej mocy obliczeniowej, a także zasobów, ponieważ jak wiemy tworzy wiele drzew, aby następnie połączyć ich wyniki. Z tego powodu również trenowanie danych zajmuje dużo czasu. Co więcej, chociaż algorytm może być używany zarówno do zadań klasyfikacji, jak

i regresji, to zdecydowanie lepiej radzi sobie z problemami klasyfikacyjnymi aniżeli regresyjnymi.

3.3 Dostępne biblioteki w języku Python

W Scikit-Learn zoptymalizowany zespół losowych drzew decyzyjnych jest zaimplementowany w estymatorze RandomForestClassifier, który automatycznie zajmuje się całą randomizacją. Wystarczy wybrać liczbę estymatorów, a bardzo szybko dopasuje się do zespołu drzew.

Rozdział 4

Porównanie algorytmów na wybranych zbiorach danych

Porównanie zostało dokonane na czterech zbiorach danych.
W przypadku algorytmów: ID3, C4.5, CART oraz CHAID dokładność zbadana została przy użyciu frameworku chefboost.
Dla algorytmu MSI, wykorzystany został pakiet msitrees (https://pypi.org/project/msitrees/).
Do porównania wykorzystane zostały również klasyfikatory: DecisionTreeClassifier (biblioteka sklearn.tree), AdaBoostClassifier (biblioteka sklearn.ensemble) oraz RandomForestClassifier (biblioteka sklearn.ensemble). Pod uwagę wzięte zostały przede wszystkim dokładność algorytmu oraz czas budowy drzewa decyzyjnego.

4.1 Breast Cancer Wisconsin

Jest to zbiór danych dostępny w bibliotece sklearn.datasets, dotyczący raka piersi. Dane zostały uzyskane ze szpitali Uniwersytetu Wisconsin w Madison od dr. Williama H. Wolberga. W zbiorze tym pojawiają się dwie możliwe klasy: Benign(nowotwór łagodny) oraz Malignant(nowotwór złośliwy). Rozmiar zbioru to 569 wierszy oraz 31 kolumn. Poniżej można zobaczyć jak prezentuje się zbiór.

mean oncavity	mean concave points	mean symmetry	mean fractal dimension	radius error	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0.30010	0.14710	0.2419	0.07871	1.0950	...	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890	Benign
0.08690	0.07017	0.1812	0.05667	0.5435	...	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902	Benign
0.19740	0.12790	0.2069	0.05999	0.7456	...	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758	Benign
0.24140	0.10520	0.2597	0.09744	0.4956	...	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300	Benign
0.19800	0.10430	0.1809	0.05883	0.7572	...	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678	Benign
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.24390	0.13890	0.1726	0.05623	1.1760	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	Benign
0.14400	0.09791	0.1752	0.05533	0.7655	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637	Benign
0.09251	0.05302	0.1590	0.05648	0.4564	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820	Benign
0.35140	0.15200	0.2397	0.07016	0.7260	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400	Benign
0.00000	0.00000	0.1587	0.05884	0.3857	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039	Malignant

ID3:

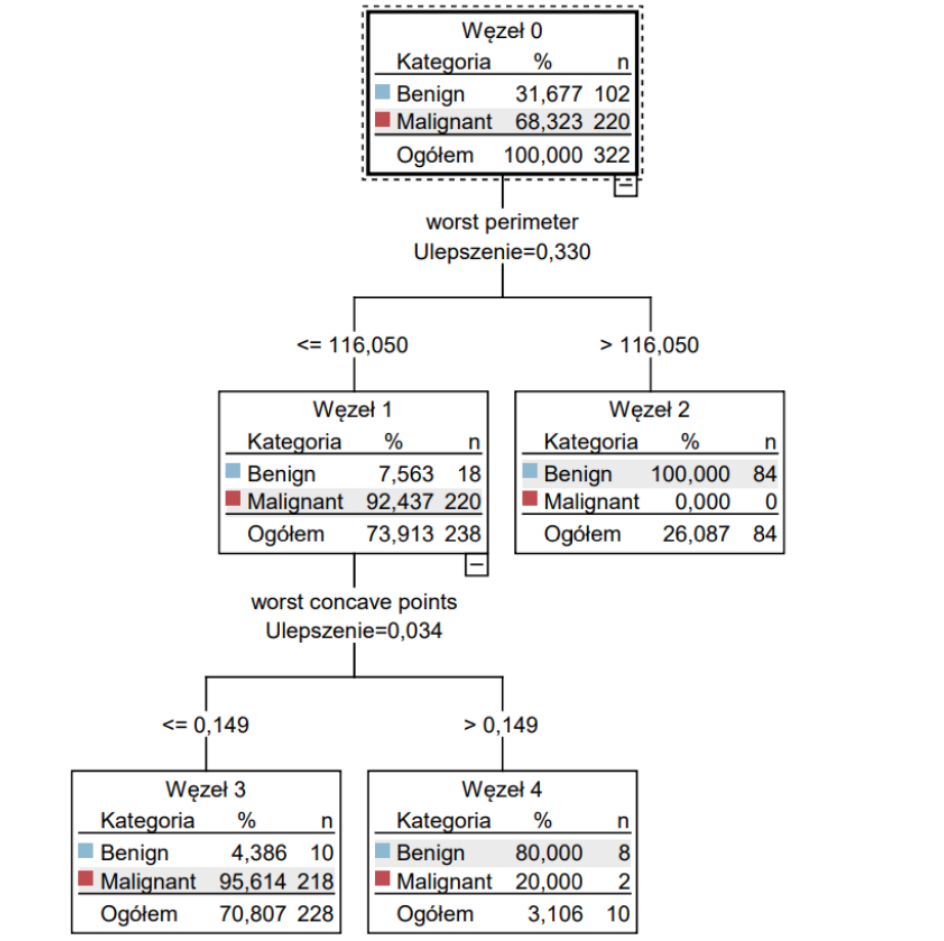
Drzewo decyzyjne zostało zbudowane algorytmem ID3 wykorzystując framework chefboost. Czas budowy drzewa wyniósł 18 sekund. Dokładność dla algorytmu ID3 na podanym zbiorze danych osiągnęła 89%.

C4.5:

Dla algorytmu C4.5 czas budowy drzewa wyniósł 21 sekund. Dokładność dla algorytmu C4.5 na podanym zbiorze danych osiągnęła 85%.

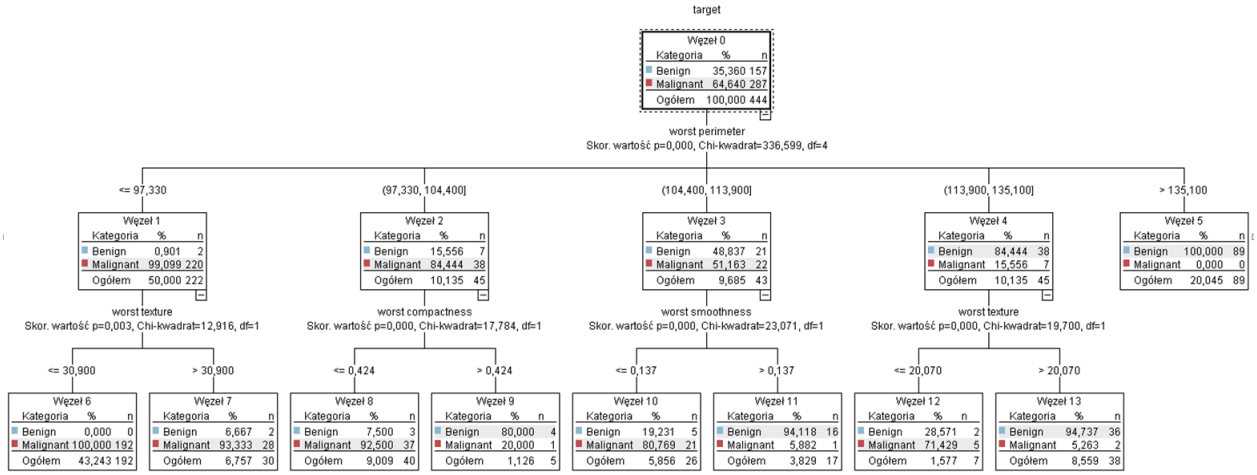
CART:

Czas budowy drzewa wyniósł 11.76 sekund. Dokładność dla algorytmu CART na podanym zbiorze danych osiągnęła 23%. Poniżej można dostrzec drzewo zbudowane algorytmem CART, wykonane za pomocą Clementine. Najistotniejszym predyktorem okazał się worst perimeter.



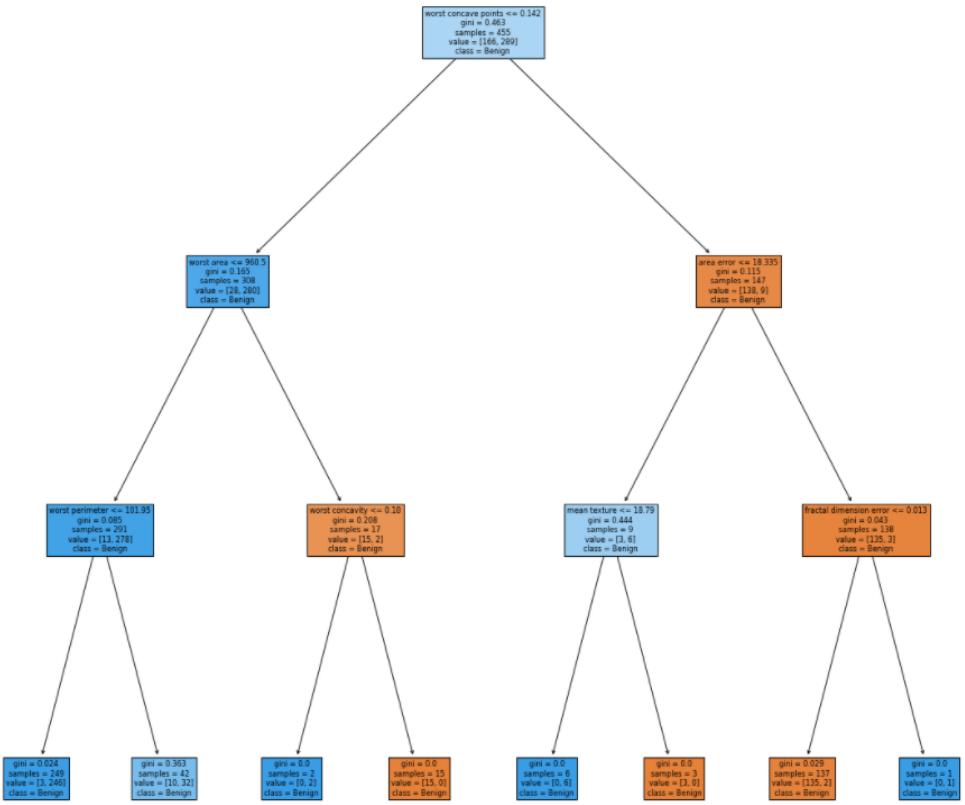
CHAID:

Algorytmem CHAID, drzewo zostało zbudowane w 22.6 sekund. Dokładność dla algorytmu CHAID na podanym zbiorze danych osiągnęła 93%. Poniżej znajduje się drzewo zbudowane tym algorytmem, Jak w przypadku CART wykonane zostało ono przy pomocy Clementine. Najistotniejszym predyktorem również w tym wypadku okazał się worst perimeter.

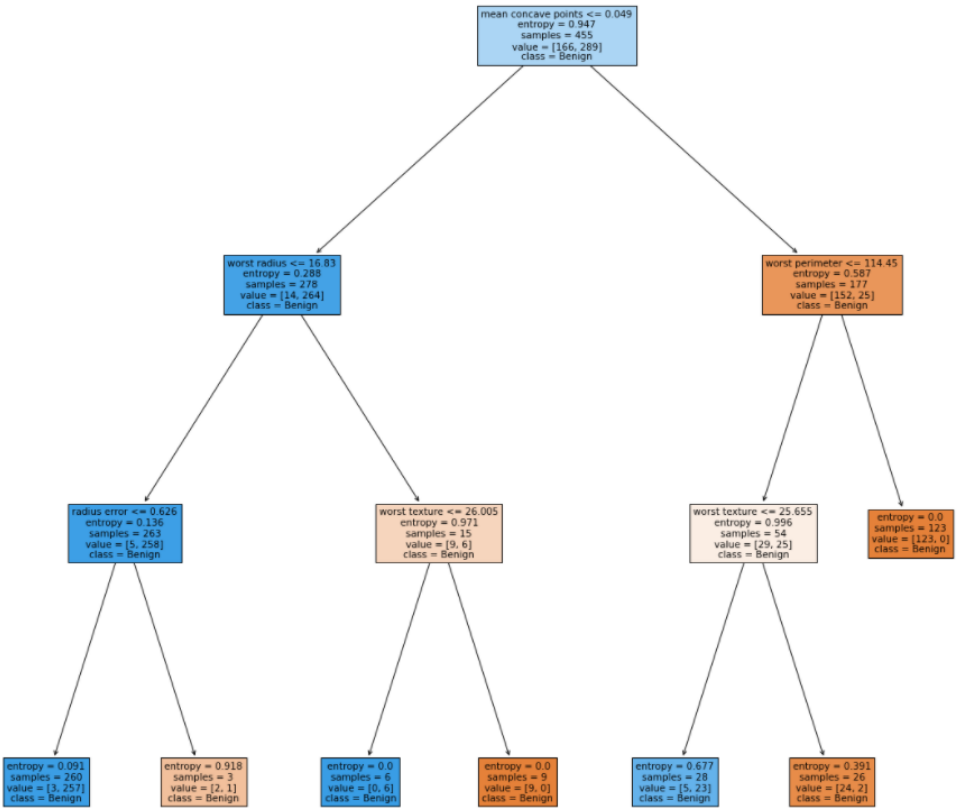


DecisionTreeClassifier:

Klasyfikator dostępny w bibliotece sklearn.tree. Dokładność została zbadana dla dwóch kryteriów: gini oraz entropy. W przypadku klasyfikatora z kryterium podziału określonym jako gini, dokładność drzewa decyzyjnego wyniosła 97%. Z kolei w drugim przypadku dokładność osiągnęła 94%. Poniżej przedstawione zostały drzewa, zbudowane przy użyciu wspomnianego klasyfikatora, zarówno dla kryterium podziału gini(Rys. 4.1) jak i entropy(Rys. 4.2). W przypadku drzewa z kryterium podziału gini, najistotniejszymi predykatorami są: worst concave points oraz worst are. Przy kryterium podziału entropy, najistotniejszymi predyktorami są: worst perimeter, mean concave points oraz mean concavity.



Rysunek 4.1: Drzewo decyzyjne z kryterium podziału gini



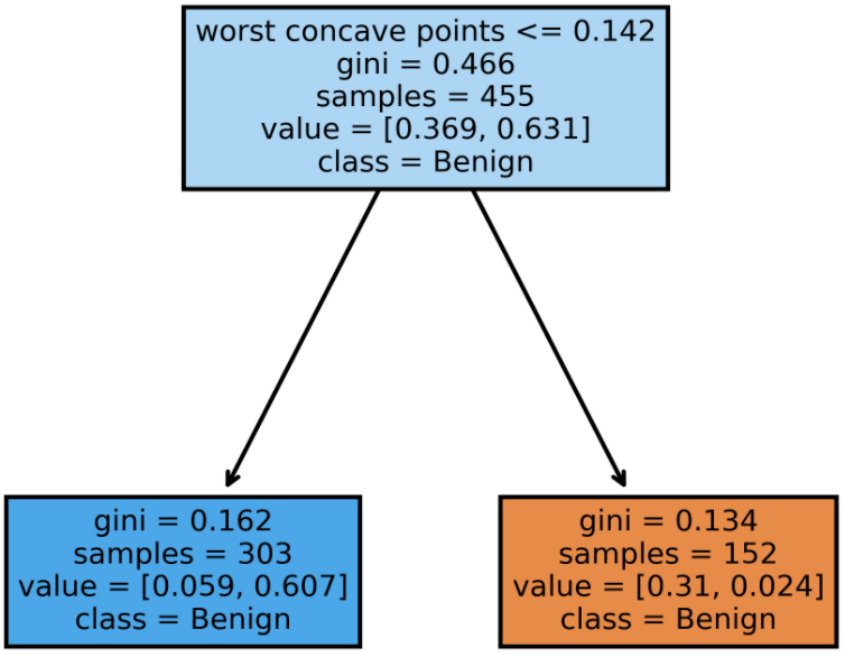
Rysunek 4.2: Drzewo decyzyjne z kryterium podziału entropy

MSI:

Dla algorytmu MSI, do budowy drzewa wykorzystany został MSIDecisionTreeClassifier dostępny w bibliotece msitrees.tree. Dokładność w tym wypadku osiągnęła 94%.

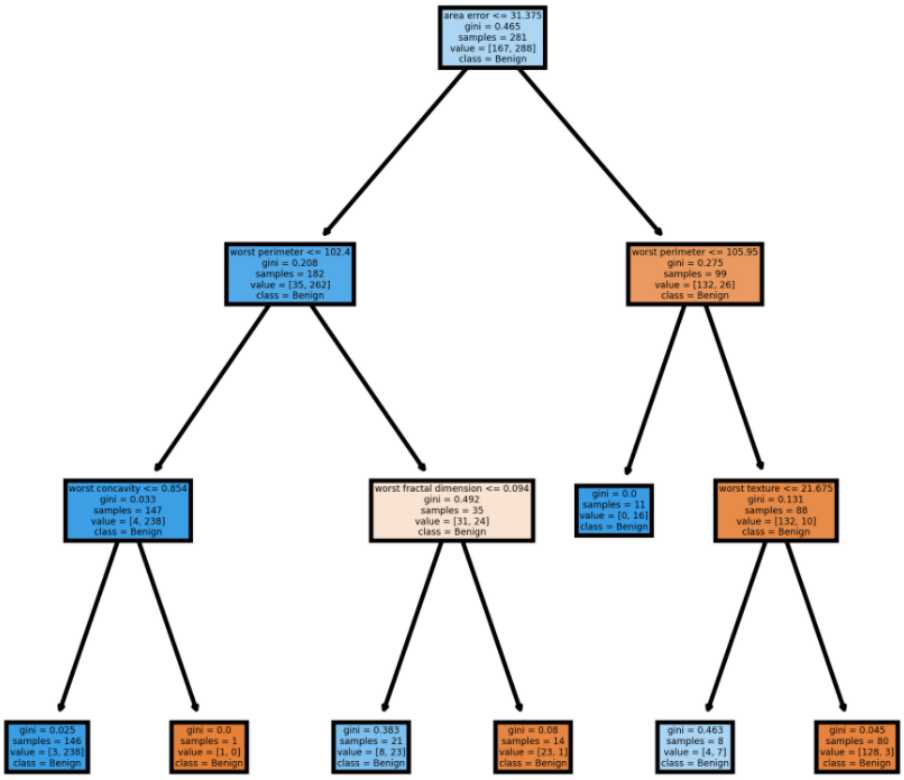
AdaBoostClassifier:

Dla omawianego zbioru danych dokładność przy pomocy tego klasyfikatora osiągnęła wartość 99%. Poniżej znajduje się drzewo zbudowane tym klasyfikatorem. Najistotniejszym predyktorem okazuje się worst concave points.



RandomForestClassifier:

Jest to klasyfikator dostępny w bibliotece sklearn.ensemble. Dla zbioru danych dotyczącego raka piersi, dokładność przy pomocy tego klasyfikatora osiąga wartość 98%. Poniżej znajduje się drzewo zbudowane tym klasyfikatorem. Najistotniejsze predyktory to: area error, worst perimeter.



Porównanie accuracy	
ID3	89%
C4.5	85%
CART	23%
CHAID	93%
DecisionTreeCalssifier (gini)	97%
DecisionTreeCalssifier (entropy)	94%
MSI	93%
AdaBoostClassifier	99%
RandomForestClassifier	98%

Dla zbioru danych Breast Cancer Wisconsin najmniejszą dokładność daje algorytm CART, natomiast najwyższą AdaBoostClassifier.

Porównajmy jeszcze czas budowy drzewa dla algorytmów.

Porównanie czasu budowy drzewa	
ID3	18s
C4.5	21s
CART	11.76s
CHAID	22.6s
DecisionTreeCalssifier (gini)	0.012s
DecisionTreeCalssifier (entropy)	0.011s
MSI	4s
AdaBoostClassifier	0.16s
RandomForestClassifier	0.19s

Najdłużej budowane było drzewo algorytmem CHAID, natomiast najkrócej poradził sobie DecisionTreeCalssifier.

4.2 Titanic

Zbiór ten dostarcza dane na temat ofiar katastrofy Titanica. Posiada informacje o pasażerach statku wraz z binarną etykietą określającą czy przeżyli katastrofę, czy utonęli. Każdy rekord opisany jest cechami takimi jak: Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked. Rozmiar zbioru to 891 wierszy oraz 11 kolumn. Zbiór danych prezentuje się poniżej.

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Decision
0	1	3	0	22.0	1	0	0	7.2500	0	0	Died
1	2	1	1	38.0	1	0	1	71.2833	1	1	Survived
2	3	3	1	26.0	0	0	2	7.9250	0	0	Survived
3	4	1	1	35.0	1	0	3	53.1000	2	0	Survived
4	5	3	0	35.0	0	0	4	8.0500	0	0	Died
...	...	...	...	...	...	...	...	...	...	...	...
886	887	2	0	27.0	0	0	677	13.0000	0	0	Died
887	888	1	1	19.0	0	0	678	30.0000	146	0	Survived
888	889	3	1	0.0	1	2	614	23.4500	0	0	Died
889	890	1	0	26.0	0	0	679	30.0000	147	1	Survived
890	891	3	0	32.0	0	0	680	7.7500	0	2	Died

891 rows × 11 columns

ID3:

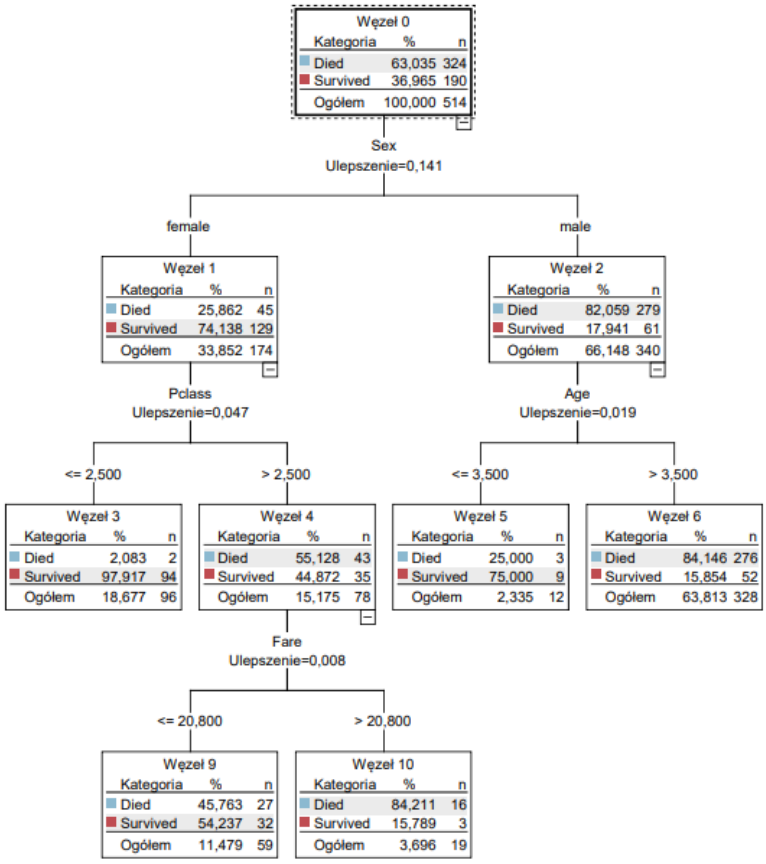
Czas budowy drzewa wyniósł 6.95 sekund. Dokładność dla algorytmu ID3 na podanym zbiorze danych osiągnęła 54%.

C4.5:

Czas budowy drzewa wyniósł 6.58 sekund. Dokładność dla algorytmu C4.5 na podanym zbiorze danych osiągnęła 77%.

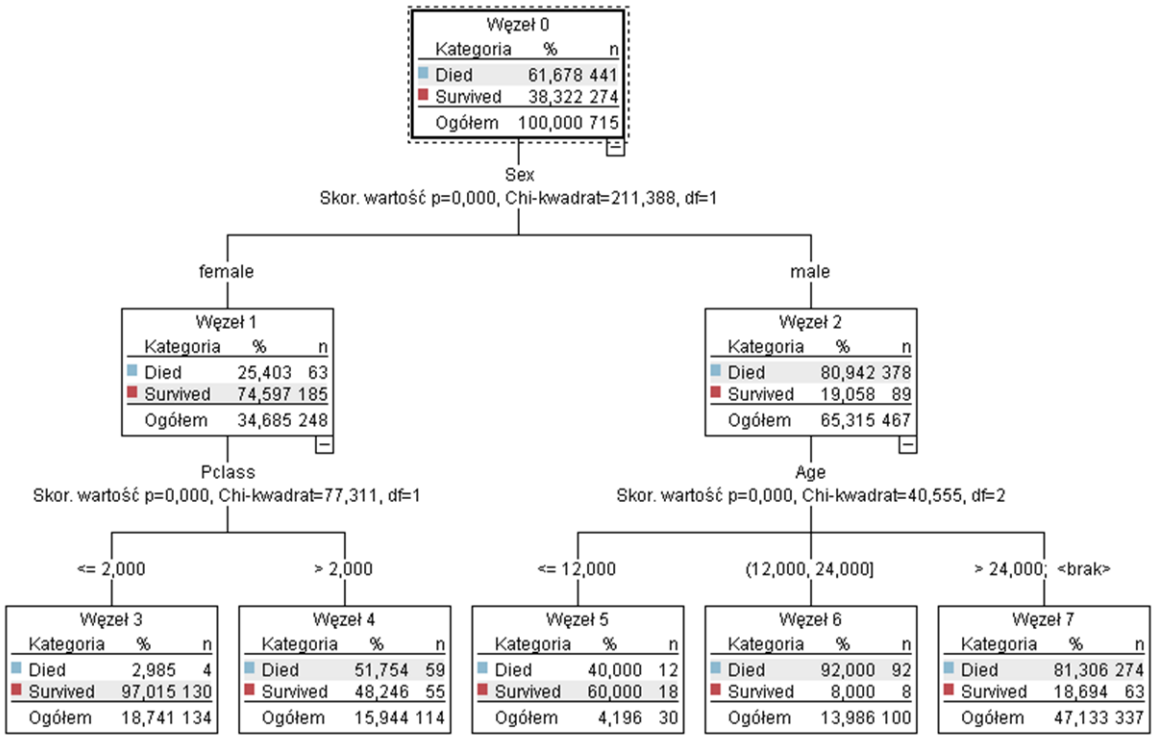
CART:

Dla algorytmu CART czas budowy drzewa wyniósł 6.87 sekund. Dokładność dla algorytmu CART na podanym zbiorze danych osiągnęła 54%. Drzewo zostało zbudowane przy pomocy Clementine. Najważniejszym predyktorem jest płeć pasażera.



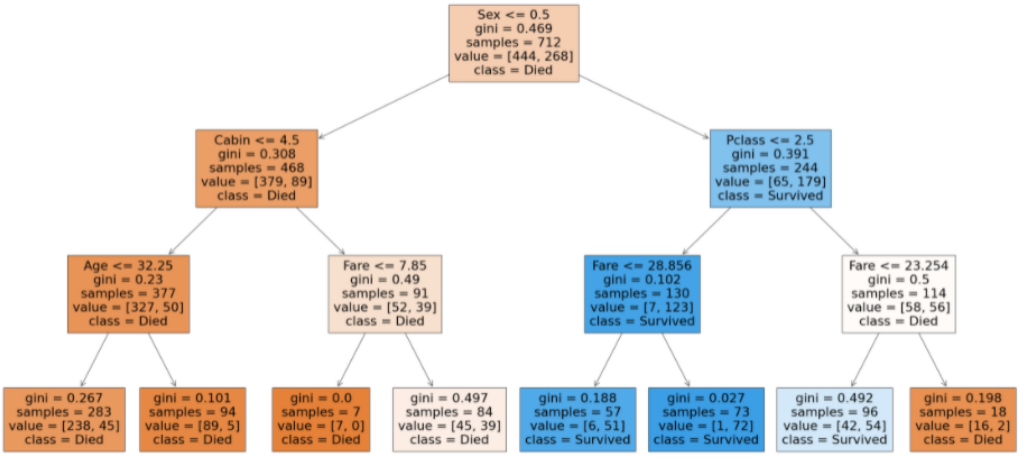
CHAID:

Drzewo zostało zbudowane w 6.58 sekund. Dokładność dla algorytmu CHAID na podanym zbiorze danych osiągnęła 54%. Drzewo widoczne jest poniżej. W tym wypadku również najistotniejszym predyktorem okazała się płeć.

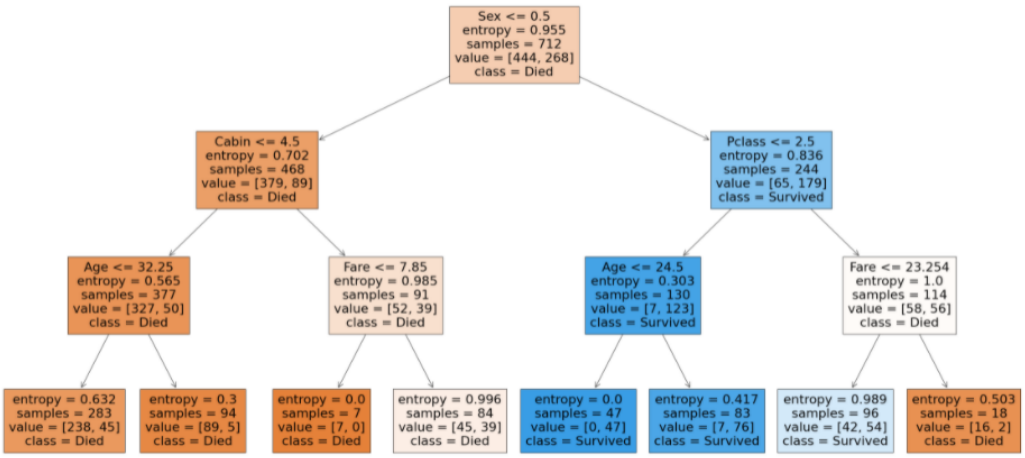


DecisionTreeClassifier:

Tak jak w przypadku poprzedniego zbioru danych, tak i tutaj dokładność została zbadana dla dwóch kryteriów: gini oraz entropy. W przypadku klasyfikatora z kryterium podziału określonym jako gini, dokładność drzewa decyzyjnego wyniosła : 83%, natomiast w drugim przypadku 82% Poniżej również przedstawione zostały drzewa, zbudowane przy użyciu wspomnianego klasyfikatora, zarówno dla kryterium podziału gini(Rys. 4.3) jak i entropy(Rys. 4.4). Najistotniejszymi predyktorami w obu przypadkach są: Sex, Cabin, Pclass.



Rysunek 4.3: Drzewo decyzyjne z kryterium podziału gini



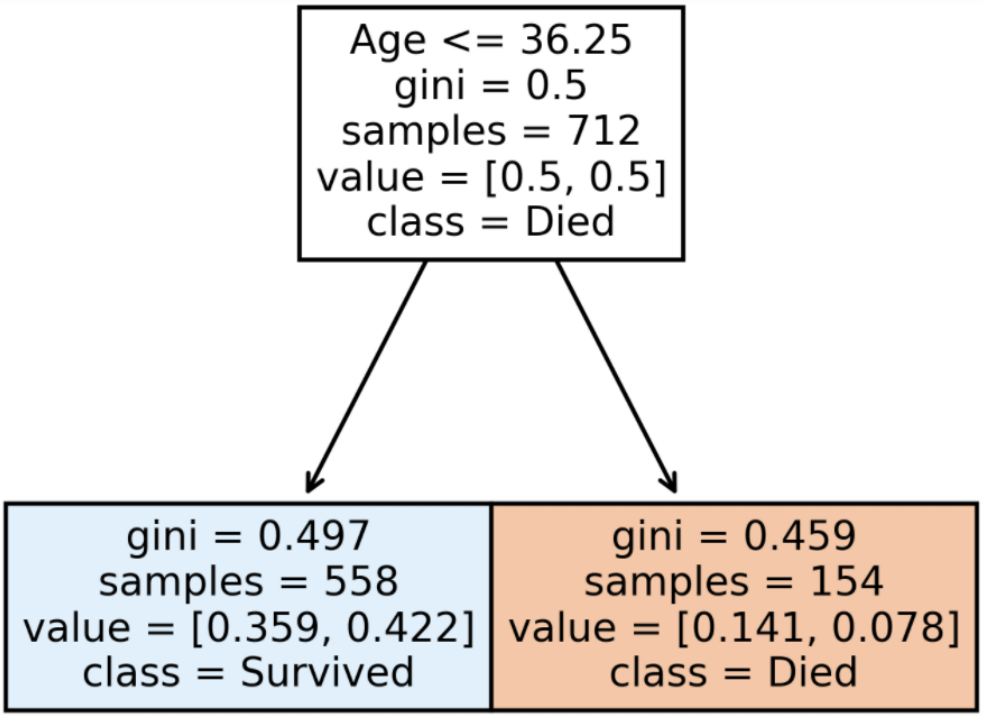
Rysunek 4.4: Drzewo decyzyjne z kryterium podziału entropy

MSI:

Dla algorytmu MSI, do budowy drzewa wykorzystany został MSIDecisionTreeClassifier dostępny w bibliotece msitrees.tree. Dokładność w tym wypadku osiągnęła 80%.

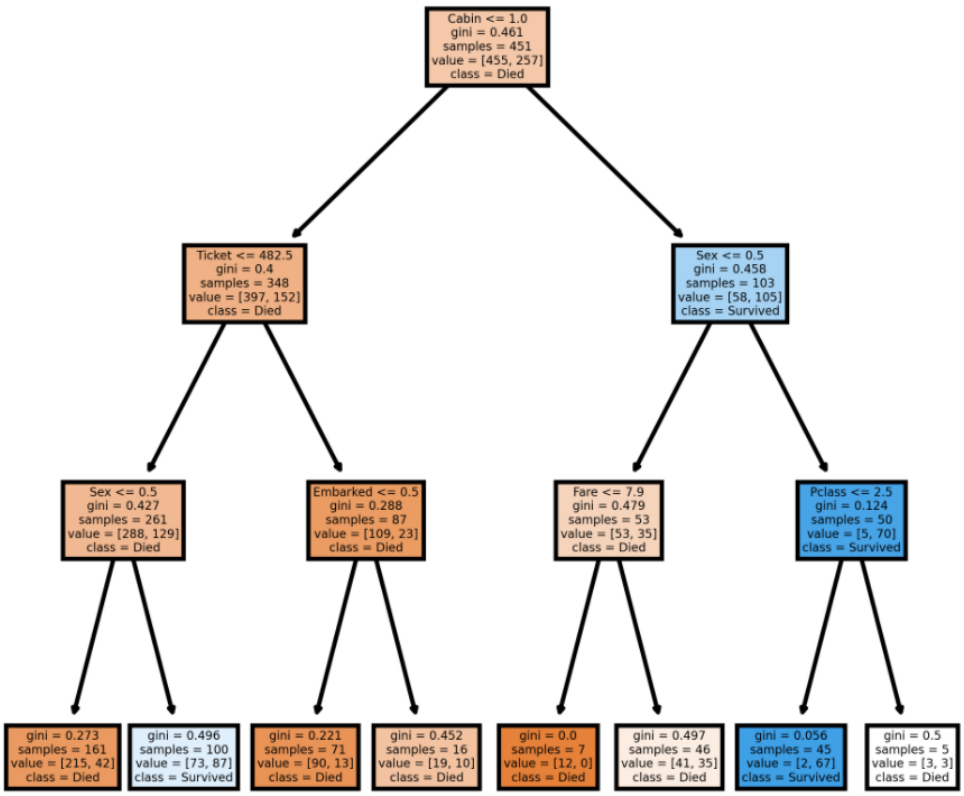
AdaBoostClassifier:

Dla omawianego zbioru danych dokładność przy pomocy tego klasyfikatora osiągnęła wartość 81%. Poniżej znajduje się drzewo zbudowane tym klasyfikatorem. Najistotniejszym predyktorem jest Age.



RandomForestClassifier:

Dla zbioru danych Titanic, dokładność przy pomocy tego klasyfikatora osiąga wartość 80%. Poniżej znajduzje się drzewo zbudowane przy pomocy tego klasyfikatora. Najistotniejszymi predyktorami są: Cabin, Ticket, Sex.



Porównanie accuracy	
ID3	54%
C4.5	77%
CART	54%
CHAID	54%
DecisionTreeCalssifier (gini)	83%
DecisionTreeCalssifier (entropy)	82%
MSI	80%
AdaBoostClassifier	81%
RandomForestClassifier	80%

Dla zbioru danych Titanic najmniejszą dokładność dają algorytmy: ID3, CART, CHAID natomiast najwyższą: DecisionTreeCalssifier dla kryterium podziału gini.

Porównajmy jeszcze czas budowy drzewa dla algorytmów

Porównanie czasu budowy drzewa	
ID3	6.95s
C4.5	6.58s
CART	6.87s
CHAID	6.58s
DecisionTreeCalssifier (gini)	0.01s
DecisionTreeCalssifier (entropy)	0.012s
MSI	0.20s
AdaBoostClassifier	0.32s
RandomForestClassifier	0.37s

Najdłużej budowane zostało drzewo algorytmem ID3. Najkrócej zajęło to Decision-TreeCalssifier.

4.3 Wine Quality

Jest to zbiór danych dotyczący jakości wina czerwonego(portugalskiego wina „Vinho Verde"). Rekordy opisane są następującymi cechami: fixed acidity, volatile acidity, citric acid residual sugar chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol. W zbiorze pojawiają sie możliwe klasy: (5, 7, 6, 4, 8, 3) które są oceną jakości wina. Rozmiar zbioru to 1599 wierszy oraz 12 kolumn. Zbiór danych możemy zaobserwować poniżej.

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates	alcohol	Decision
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	five
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	five
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	five
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	six
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	five
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	five
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	six
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	six
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	five
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	six

1599 rows × 12 columns

ID3:

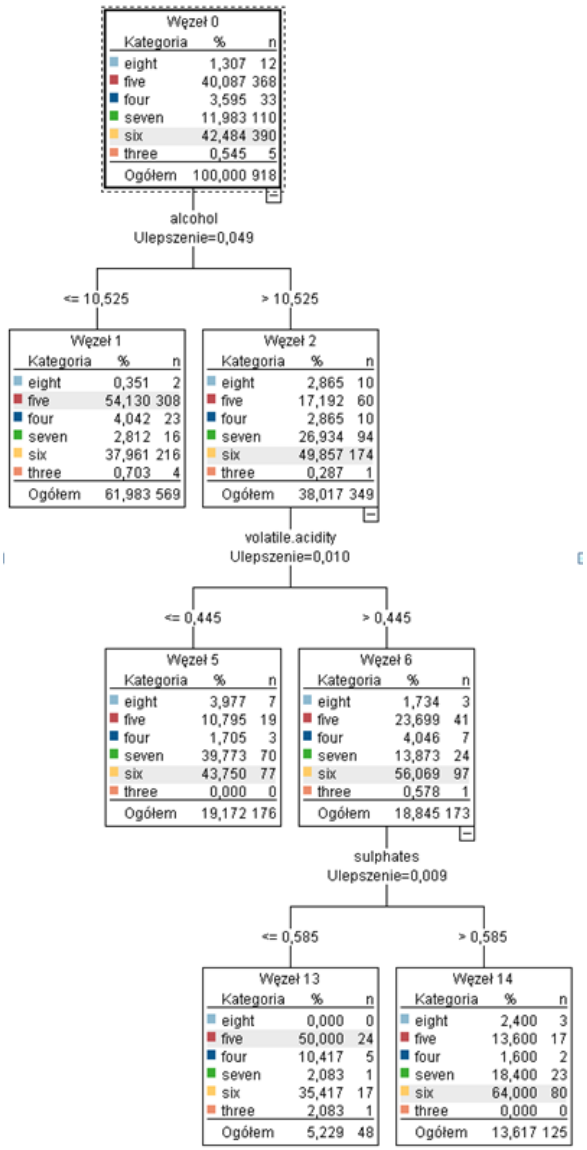
Czas budowy drzewa wyniósł 59.4 sekund. Dokładność dla algorytmu ID3 na podanym zbiorze danych osiągnęła 54%.

C4.5:

Czas budowy drzewa wyniósł 7.71 sekund. Dokładność dla algorytmu C4.5 na podanym zbiorze danych osiągnęła 40%.

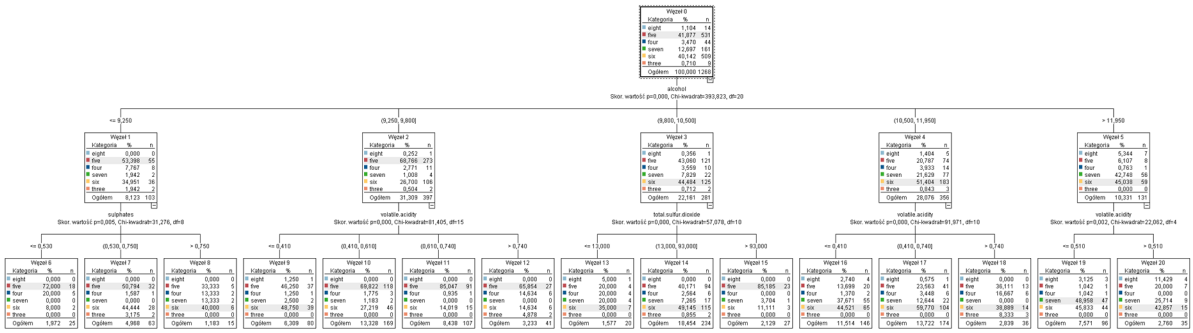
CART:

Drzewo zostało zbudowane w 34.7 sekund. Dokładność dla algorytmu CART na podanym zbiorze danych osiągnęła 40%. Najważniejszym predyktorem na podstawie poniższego drzewa okazł się atrybut alcohol.



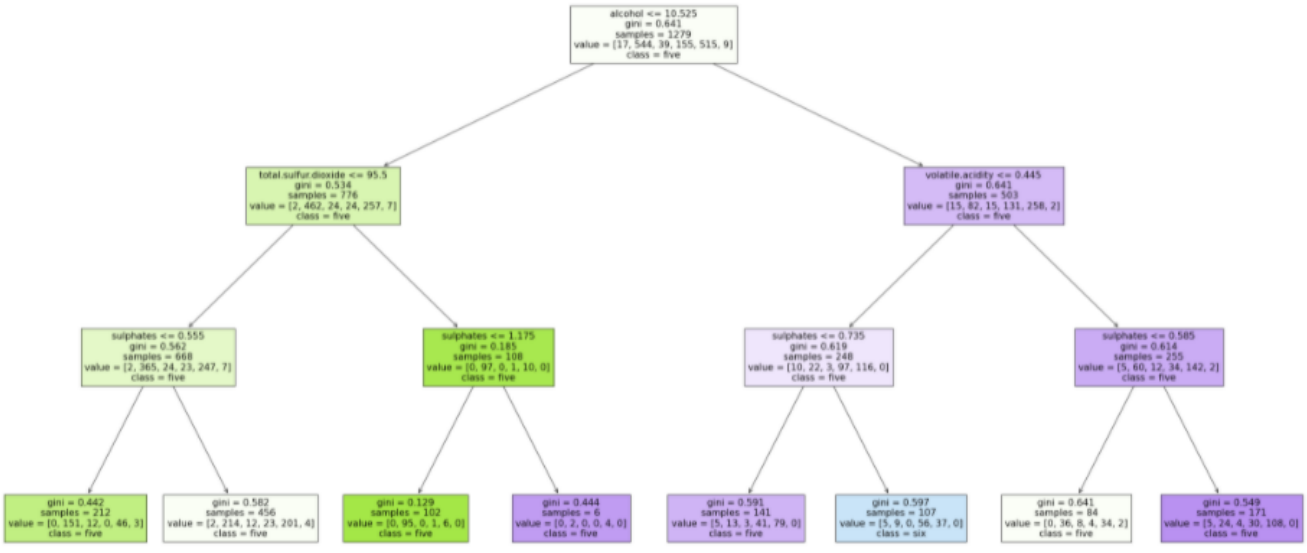
CHAID:

Czas budowy drzewa wyniósł 64.5 sekund. Dokładność dla algorytmu CHAID na podanym zbiorze danych osiągnęła 39%. Dla CHAID na omawianym zbiorze danych najistotniejszym predyktorem jest również atrybut alcohol.

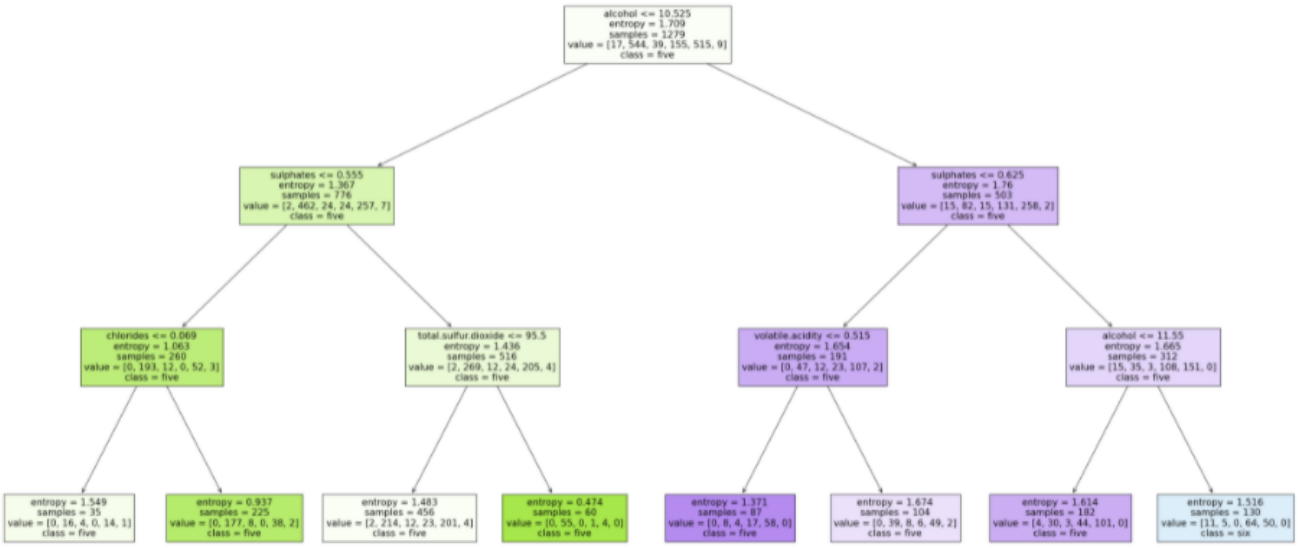


DecisionTreeClassifier:

Klasyfikator dostępny w bibliotece sklearn.tree. Dokładność została zbadana dla dwóch kryteriów: gini oraz entropy. W przypadku klasyfikatora z kryterium podziału określonym jako gini, dokładność drzewa decyzyjnego wyniosła 53%. Z kolei w drugim przypadku dokładność osiągnęła 56%. Poniżej przedstawione zostały drzewa, zbudowane przy użyciu wspomnianego klasyfikatora, zarówno dla kryterium podziału gini(Rys. 4.5) jak i entropy(Rys. 4.6). Dla obu kryteriów podziału pojawiają się te same naijstotniejsze predyktory: alcohol, sulphates.



Rysunek 4.5: Drzewo decyzyjne z kryterium podziału gini



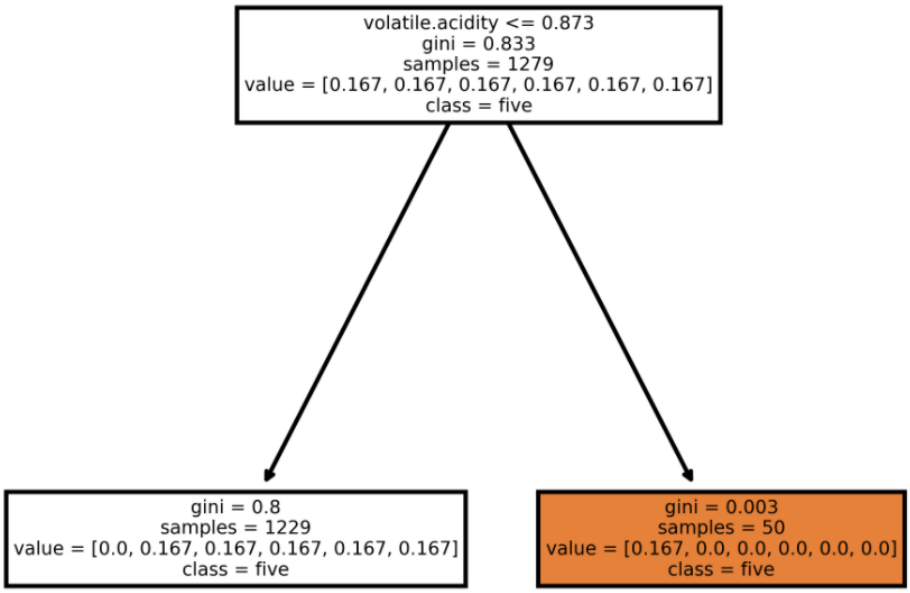
Rysunek 4.6: Drzewo decyzyjne z kryterium podziału entropy

MSI:

Dokładność w tym wypadku osiągnęła 53%.

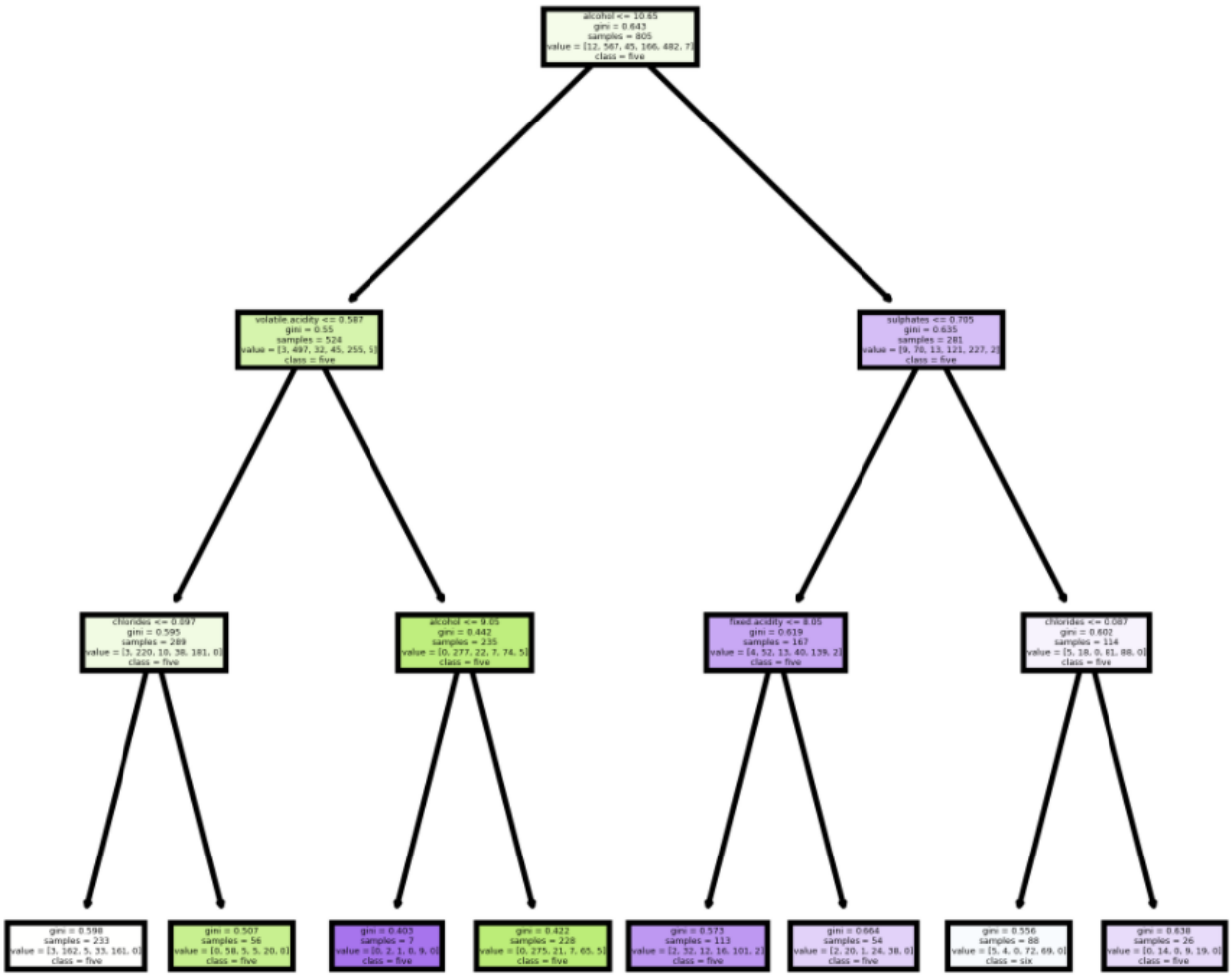
AdaBoostClassifier:

Jest to klasyfikator dostępny w bibliotece sklearn.ensemble. Dla omawianego zbioru danych dokładność przy pomocy tego klasyfikatora osiągnęła wartość 53%. Poniżej znajduje się drzewo zbudowane tym klasyfikatorem. Najistotniejszym predyktorem jest volatile.acidity.



RandomForestClaassifier:

Dla omawianego zbioru danych Wine Quality dokładność przy pomocy tego klasyfikatora osiągnęła wartość 56%. Drzewo zbudowane tym klasyfikatorem można dostrzec poniżej. Zauważamy, że najistotniejszymi predykatorami są m.in. alcohol oraz sulphates.



Porównanie accuracy	
ID3	54%
C4.5	40%
CART	40%
CHAID	39%
DecisionTreeCalssifier (gini)	53%
DecisionTreeCalssifier (entropy)	56%
MSI	53%
AdaBoostClassifier	53%
RandomForestClassifier	56%

Dla zbioru danych Wine Quality najmniejszą dokładność daje algorytm CHAID na-

tomiast najwyższą dokładność daje DecisionTreeClassifier dla kryterium podziału entropy, oraz RandomForestClassifier.

Porównajmy jeszcze czas budowy drzewa dla algorytmów.

Porównanie czasu budowy drzewa	
ID3	59.4s
C4.5	7.71s
CART	34.7s
CHAID	64.5s
DecisionTreeCalssifier (gini)	0.001s
DecisionTreeCalssifier (entropy)	0.016s
MSI	0.66s
AdaBoostClassifier	0.17s
RandomForestClassifier	0.23s

Najdłużej budowane jest drzewo algorytmem CHAID.

4.4 Car evaluation

Zbiór ten dotyczy oceny samochodów. Odbywa się ona na podstawie atrybutów takich jak: buying (cena zakupu), maintenance (cena utrzymania), no doors (liczba drzwi), no persons (pojemność w przeliczeniu na osoby do przewozu), luggage boot (rozmiar bagażnika), safety (szacunkowe bezpieczeństwo samochodu). Możliwe decyzje (klasy) to: unacc, acc, good, vgood. Rozmiar zbioru to 1727 wierszy oraz 7 kolumn. Zbiór danych prezentuje sie następująco:

	buying	maintenance	no_doors	no_persons	luggage_boot	safety	Decision
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc
...	...	...	...	...	...	...	...
1722	low	low	5more	more	med	med	good
1723	low	low	5more	more	med	high	vgood
1724	low	low	5more	more	big	low	unacc
1725	low	low	5more	more	big	med	good
1726	low	low	5more	more	big	high	vgood

1727 rows × 7 columns

ID3:

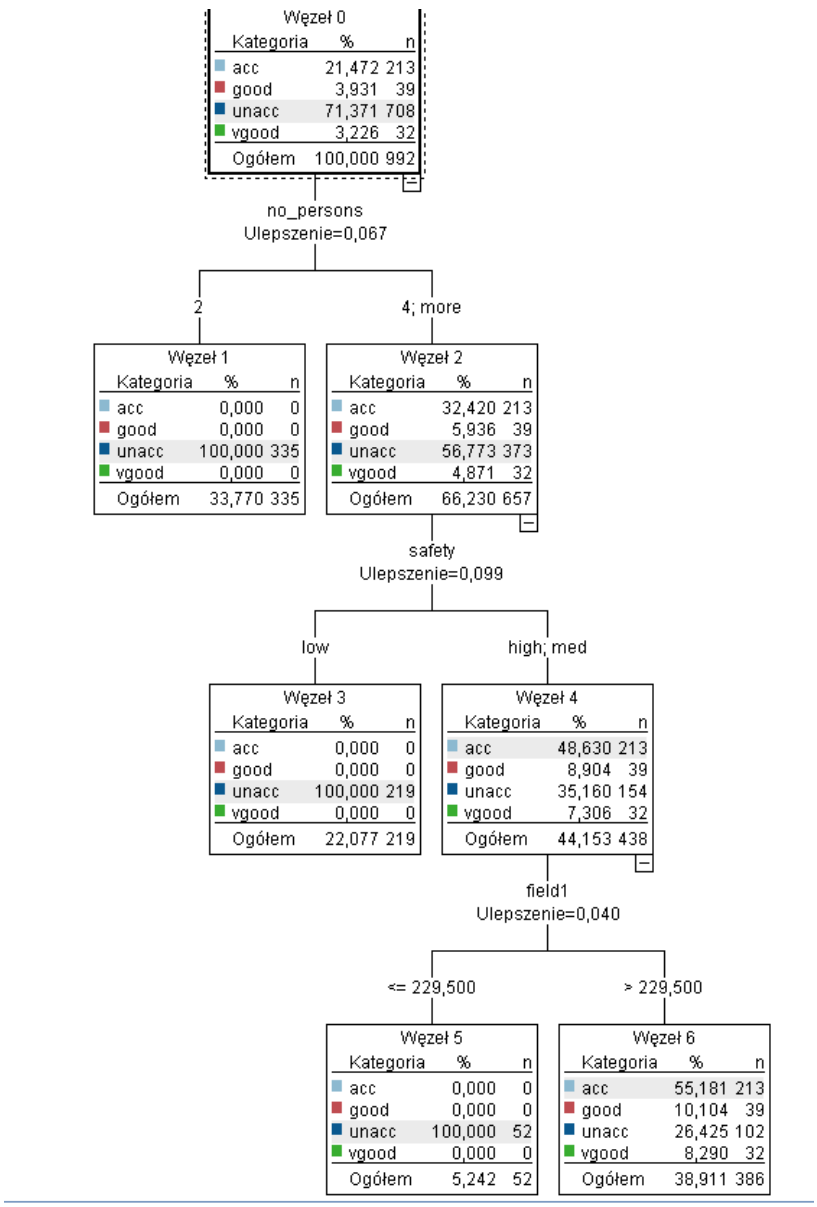
Czas budowy drzewa wyniósł 18 sekund. Dokładność dla algorytmu ID3 na podanym zbiorze danych osiągnęła 95%.

C4.5:

Drzewo zostało zbudowane w 21.5 sekund. Dokładność dla algorytmu C4.5 na podanym zbiorze danych osiągnęła 95%.

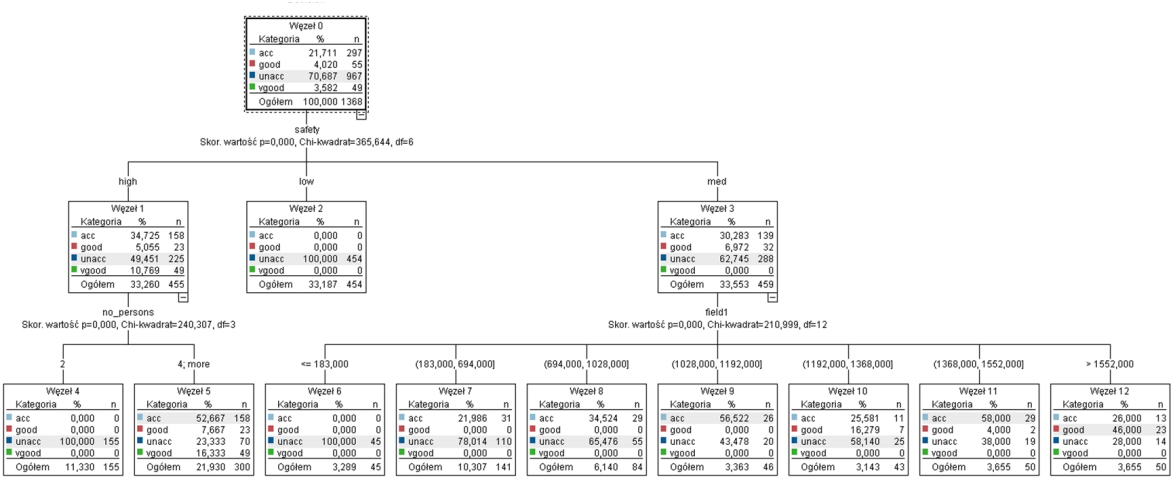
CART:

Czas budowy drzewa wyniósł 18 sekund. Dokładność dla algorytmu CART na podanym zbiorze danych osiągnęła 93%. Najistotniejszym predyktorem w drzewie okazał się atrybut no persons oraz safety.



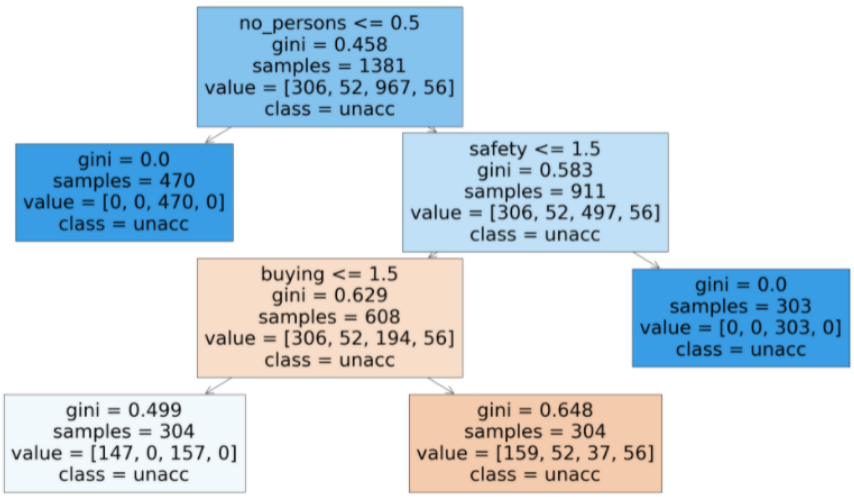
CHAID:

Czas budowy drzewa wyniósł 24.63 sekund. Dokładność dla algorytmu CHAID na podanym zbiorze danych osiągnęła 91%. Najważniejszym preyktorem dla tego drzewa jest atrybut safety.

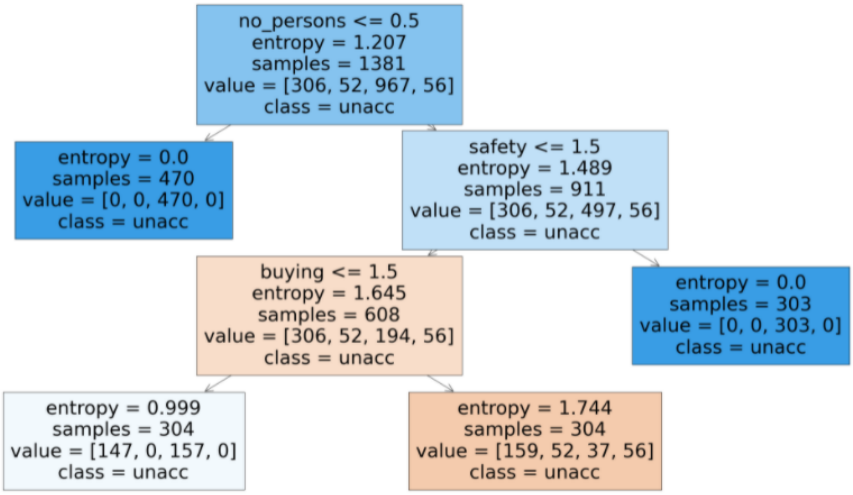


DecisionTreeClassifier:

Dokładność została zbadana dla dwóch kryteriów: gini oraz entropy. Dla kryterium podziału drzewa: gini dokładność wyniosła 83% natomiast dla entropy: 82%. Poniżej przedstawione zostały drzewa, zbudowane przy użyciu wspomnianego klasyfikatora, zarówno dla kryterium podziału gini(Rys. 4.7) jak i entropy(Rys. 4.8). Najistotniejsze predyktory: no persons, safety.



Rysunek 4.7: Drzewo decyzyjne z kryterium podziału gini



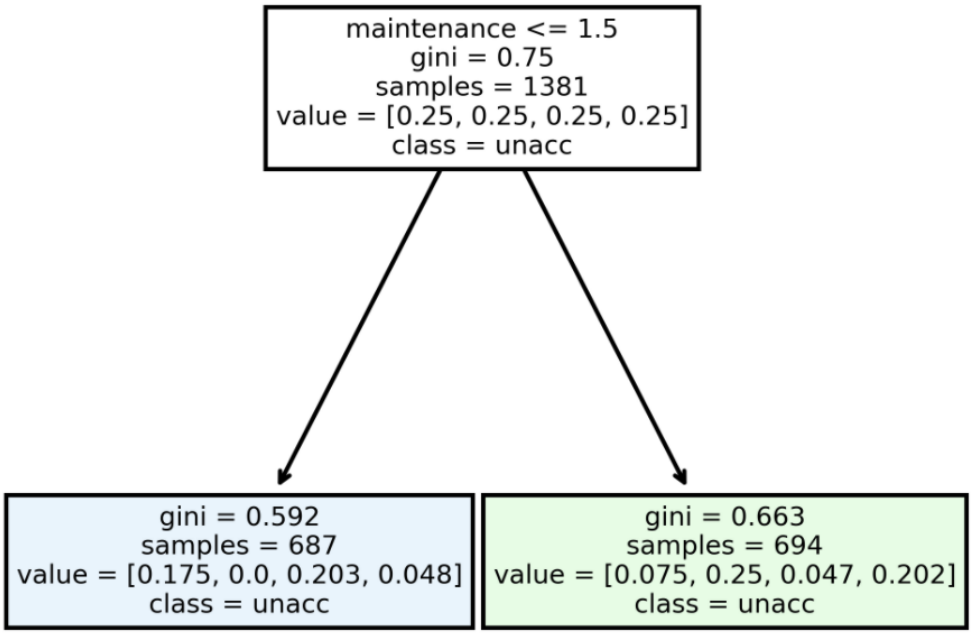
Rysunek 4.8: Drzewo decyzyjne z kryterium podziału entropy

MSI:

Dokładność w tym wypadku osiągnęła 77%.

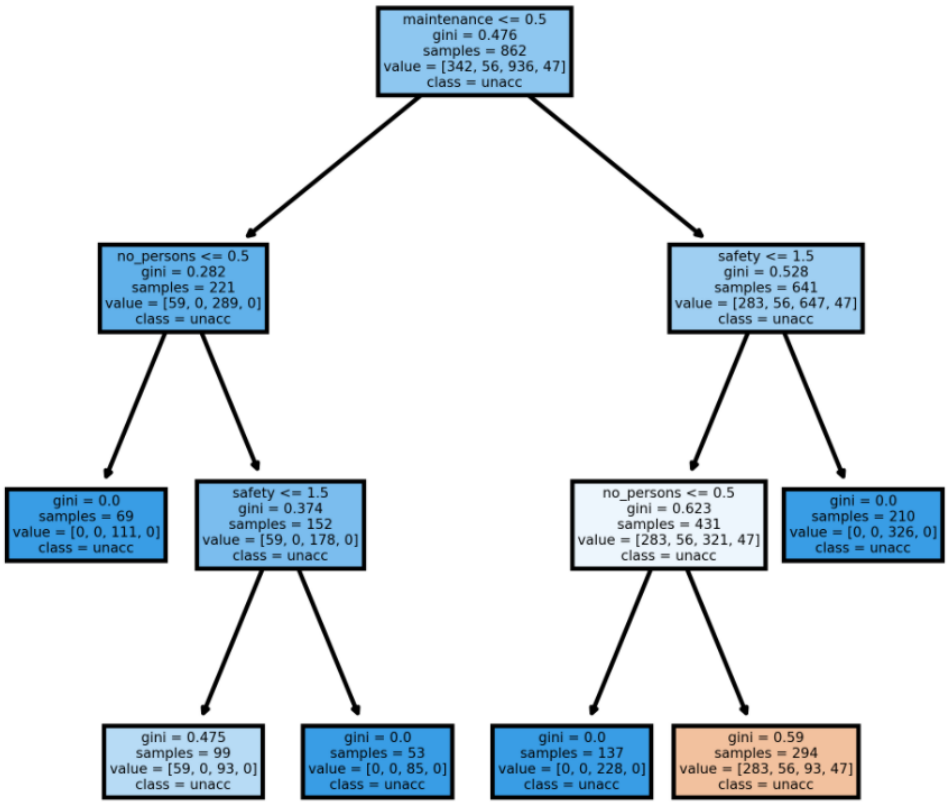
AdaBoostClassifier:

Dla omawianego zbioru danych dokładność przy pomocy tego klasyfikatora osiągnęła wartość 88%. Poniżej znajduje się drzewo zbudowane tym klasyfikatorem. Najistotniejszym predyktorem jest maintenance.



RandomForestClassifier:

Dla zbioru danych Car evaluation dokładność przy pomocy tego klasyfikatora osiągnęła wartość 84%. Drzewo zbudowane tym klasyfikatorem dostrzegamy poniżej. Można zauważyć, że najistotniejszymi predyktorami okazały się: maintenance, safety oraz no persons.



Porównanie accuracy	
ID3	95%
C4.5	95%
CART	93%
CHAID	91%
DecisionTreeCalssifier (gini)	83%
DecisionTreeCalssifier (entropy)	82%
MSI	77%
AdaBoostClassifier	88%
RandomForestClassifier	84%

Dla zbioru danych Car evaluation najmniejszą dokładność daje algorytm MSI natomiast najwyższą dokładność C4.5 oraz ID3.

Porównajmy jeszcze czas budowy drzewa dla algorytmów: ID3, C4.5, CART oraz CHAID.

Porównanie czasu budowy drzewa	
ID3	18s
C4.5	21.5s
CART	18s
CHAID	24.63s
DecisionTreeCalssifier (gini)	0.007s
DecisionTreeCalssifier (entropy)	0.008s
MSI	0.12s
AdaBoostClassifier	0.14s
RandomForestClassifier	0.22s

Widać, że najdłużej budowane było drzewo algorytmem CHAID.

Rozdział 5

Podsumowanie

Celem pracy było porównanie metod klasyfikacji danych w oparciu o pojedyncze drzewo z algorytmami budującymi lasy losowe. Algorytmy jakie zostały wzięte pod uwagę to: ID3, C4.5, CART, CHAID, MSI. Do porównania wykorzystane zostały także klasyfikatory: DesicionTreeClassifier, AdaBoostClassifier oraz RandomForestClassifier. Jeżeli chodzi o implementację, to dokonana została ona przy pomocy języka Python oraz przy wizualizacji niektórych drzew pomocnym narzędziem okazało się Clementine. Mowa tu o CART, CHAID, DecisionTreeCalssifier (gini), DecisionTreeCalssifier (entropy), AdaBoostClassifier, RandomForestClassifier. Do budowy drzew algorytmami ID3, C4.5, CART oraz CHAID skorzystano z frameworku chefboost, dla MSI pomocny był pakiet msitrees, natomiast klasyfikatory dostępne są w bibliotece sklearn.

Algorytmy porównane zostały na czterech zbiorach danych. Pierszym z nich jest Breast Cancer Wisconsin dostępny w bibliotece sklearn.datasets, który dotyczy raka piersi. Są tu dwie możliwe klasy: nowotwór łagodny oraz złośliwy. Największą dokładność zapewnił klasyfikator AdaBoostClassifier (99%), natomiast spośród algorytmów budujących pojedyncze drzewa najlepszy okazał się CHAID z dokładnością 93%. Najniższa dokładność pojawiła się w przypadku algorytmu CART (23%). Jeżeli chodzi o czas budowy drzew to oczywiście zaskakujące nie jest, że najkrótszy okazał się on dla klasyfikatorów, natomiast biorąc pod uwagę pojedyncze drzewa to najdłużej budowane było drzewo algorytmem CHAID (22.6s). Z kolei najkrócej budowane było drzewo CART (11.76s). Dla CART oraz CHAID najistotniejszym predyktorem okazał się atrybut worst perimeter. W przypadku DecisionTreeCalssifier z kryterium podziału gini, najistotniejszymi predykatorami są: worst concave points oraz worst are. Przy kryterium podziału entropy, są to: worst perimeter,mean concave points oraz mean concavity. Dla AdaBoostClassifier najistotniejszym predyktorem jest worst concave points.

Kolejnym zbiorem na którym dokonano analiz jest Titanic, który dostarcza dane na temat ofiar katastrofy Titanica. Pojawiają się tu dwie klasy: Died oraz Survived mówiące o tym czy pasażer przeżył katastrofę czy zmarł. W przypadku algorytmów pojedynczych drzew decyzyjnych największą dokładność osiągnął algorytm MSI(80%) natomiast najmniejszą algorytmy ID3, CART i CHAID z identyczną dokładnością: 54%. W przypadku klasyfikatorów najlepszy okazał się DecisionTreeClassifier z kryterium podziału gini z dokładnością 83%. Jeżeli chodzi o czas budowy drzewa to dla algorytmów dotyczących pojedynczych drzew, czas budowy był praktycznie identyczny w granicach 7s. W pzypadku predyktorów to najistotniejszymi okazały się: Sex, Age oraz Cabin, a więc to wiek czy płeć miały istotny wpływ na to czy pasażer mógł przeżyć.

Trzeci zbiór danych to Wine Quality dotyczący jakości wina czerwonego. Klasy pojawiające się w tym zbiorze to skala jakości wina od 3-8. Okazało się, że jest to najtrudniejszy zbiór. Dokładności nie osiągają wartości nawet 60%. ID3 osiąga 54% dokładności. Najmniejszą daje algorytm CHAID w wielkości 39%. Czas budowy drzewa też jest dłuższy niż w poprzednich zbiorach, ponieważ algorytmem CHAID drzewo budowane jest aż 65s. Najistotniejszymi predyktorami są: alcohol oraz volatile.acidity.

Ostatni zbiór danych to Car evaluation dotyczący oceny samochodów na podstawie m.in. ceny zakupu, ceny utrzymania, liczby drzwi, bezpieczeństwa czy rozmiaru bagażnika. Jest to zbiór dla którego dokładności osiągnęły największe wartości. Mieszczą się one w granicach 80-90%. Ciekawe jest również, że algorytmy budujące pojedyncze drzewa osiągają większą dokładność niż klasyfikatory. Najlepszy okazał się ID3 oraz C4.5. Czasy budowy drzew są podobne w granicach 18-24 sekund. Najkrócej budowały się drzewa z największą dokładnością czyli ID3 oraz C4.5. Najistotniejszymi predyktorami okazały się: no persons oraz maintenance.

Biorąc pod uwagę powyższą analizę, spośród algorytmów budujących pojedyncze drzewo bardzo dobrze radzi sobie algorytm C4.5, który praktycznie na każdym zbiorze danych, w porównaniu do innych algorytmów, osiąga zadawalające wyniki w dokładności i czasie budowy drzewa. Również ID3 daje dobre wyniki.

Literatura

[1] Charu C Aggarwal. *Data mining: the textbook*. Springer, 2015.

[2] Leo Breiman. "Random forests". W: *Machine learning* 45.1 (2001), s. 5–32.

[3] Johannes Gehrke, Raghu Ramakrishnan i Venkatesh Ganti. "Rainforest-a framework for fast decision tree construction of large datasets". W: *VLDB*. T. 98. Citeseer. 1998, s. 416–427.

[4] Johannes Gehrke i in. "BOAT—optimistic decision tree construction". W: *Proceedings of the 1999 ACM SIGMOD international conference on Management of Data*. 1999, s. 169–180.

[5] V. Lee, Lin Liu i R. Jin. "Decision Trees: Theory and Algorithms". W: *Data Classification: Algorithms and Applications*. 2014.

[6] Steven L Salzberg. *C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993*. 1994.

[7] Dipayan Sarkar i Vijayalakshmi Natarajan. *Ensemble Machine Learning Cookbook: Over 35 practical recipes to explore ensemble machine learning techniques using Python*. Packt Publishing Ltd, 2019.