

Wydanie III

Python

Uczenie maszynowe w przykładach
TensorFlow 2, PyTorch i scikit-learn

Yuxi (Hayden) Liu



Helion 

Packt >

Tytuł oryginału: Python Machine Learning By Example: Build intelligent systems using Python, TensorFlow 2, PyTorch, and scikit-learn, 3rd Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-8870-3

Copyright © Packt Publishing 2020. First published in the English language under the title 'Python Machine Learning By Example - Third Edition - (9781800209718)'

Polish edition copyright © 2022 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/pytuc3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

O autorze	11
O korektorach merytorycznych	13
Przedmowa	15
Rozdział 1. Pierwsze kroki z uczeniem maszynowym w Pythonie	19
Wprowadzenie do uczenia maszynowego	20
Dlaczego uczenie maszynowe jest potrzebne?	20
Różnice między uczeniem maszynowym a automatyką	22
Zastosowania uczenia maszynowego	23
Wstępne wymagania	24
Trzy rodzaje uczenia maszynowego	25
Krótka historia rozwoju algorytmów uczenia maszynowego	27
Istota uczenia maszynowego	28
Uogólnianie danych	29
Nadmierne i niedostateczne dopasowanie modelu	
oraz kompromis między obciążeniem a wariancją	30
Zapobieganie nadmiernemu dopasowaniu poprzez weryfikację krzyżową	34
Zapobieganie nadmiernemu dopasowaniu za pomocą regularyzacji	37
Zapobieganie nadmiernemu dopasowaniu poprzez selekcję cech i redukcję	
wymiarowości	39
Wstępne przetwarzanie danych i inżynieria cech	40
Wstępne przetwarzanie i eksploracja danych	41
Inżynieria cech	44
Łączenie modeli	45
Głosowanie i uśrednianie	46
Agregacja bootstrap	46
Wzmacnianie	46
Składowanie	49

Instalacja i konfiguracja oprogramowania	49
Przygotowanie Pythona i środowiska pracy	50
Instalacja najważniejszych pakietów Pythona	51
Wprowadzenie do pakietu TensorFlow 2	53
Podsumowanie	54
Ćwiczenia	54

Rozdział 2. Tworzenie systemu rekomendacji filmów na bazie naiwnego klasyfikatora Bayesa **55**

Pierwsze kroki z klasyfikacją	56
Klasyfikacja binarna	56
Klasyfikacja wieloklasowa	57
Klasyfikacja wieloetykietowa	59
Naiwny klasyfikator Bayesa	59
Twierdzenie Bayesa w przykładach	60
Mechanizm naiwnego klasyfikatora Bayesa	62
Implementacja naiwnego klasyfikatora Bayesa	65
Implementacja od podstaw	66
Implementacja z wykorzystaniem pakietu scikit-learn	69
Budowanie systemu rekomendacyjnego na bazie klasyfikatora Bayesa	69
Ocena jakości klasyfikacji	74
Strojenie modeli poprzez weryfikację krzyżową	78
Podsumowanie	80
Ćwiczenia	81
Bibliografia	81

Rozdział 3. Rozpoznawanie twarzy przy użyciu maszyny wektorów nośnych **82**

Określanie granic klas za pomocą maszyny wektorów nośnych	83
Scenariusz 1. Określenie hiperpłaszczyzny rozdzielającej	84
Scenariusz 2. Określenie optymalnej hiperpłaszczyzny rozdzielającej	84
Scenariusz 3. Przetwarzanie punktów odstających	88
Implementacja maszyny wektorów nośnych	88
Scenariusz 4. Więcej niż dwie klasy	90
Scenariusz 5. Rozwiązywanie nierozdzielnego liniowo problemu za pomocą jądra	94
Wybór między jądrem liniowym a radialną funkcją bazową	98
Klasyfikowanie zdjęć twarzy za pomocą maszyny wektorów nośnych	101
Badanie zbioru zdjęć twarzy	101
Tworzenie klasyfikatora obrazów opartego na maszynie wektorów nośnych	103
Zwiększanie skuteczności klasyfikatora obrazów za pomocą analizy głównych składowych	105
Klasyfikacja stanu płodu w kardiologii	106
Podsumowanie	108
Ćwiczenia	108

Rozdział 4. Prognozowanie kliknięć reklam internetowych przy użyciu algorytmów drzewiastych	109
Wprowadzenie do prognozowania kliknięć reklam	110
Wprowadzenie do dwóch typów danych: liczbowych i kategoryalnych	111
Badanie drzewa decyzyjnego od korzeni do liści	112
Budowanie drzewa decyzyjnego	113
Wskaźniki jakości podziału zbioru	116
Implementacja drzewa decyzyjnego od podstaw	122
Implementacja drzewa decyzyjnego za pomocą biblioteki scikit-learn	129
Prognozowanie kliknięć reklam za pomocą drzewa decyzyjnego	129
Gromadzenie drzew decyzyjnych: las losowy	135
Gromadzenie drzew decyzyjnych: drzewa ze wzmocnieniem gradientowym	137
Podsumowanie	139
Ćwiczenia	140
Rozdział 5. Prognozowanie kliknięć reklam internetowych przy użyciu regresji logistycznej	141
Przekształcanie cech kategoryalnych w liczbowe: kodowanie porządkowe i „1 z n”	142
Klasyfikowanie danych z wykorzystaniem regresji logistycznej	144
Wprowadzenie do funkcji logistycznej	145
Przejście od funkcji logistycznej do regresji logistycznej	146
Trening modelu opartego na regresji logistycznej	149
Trening modelu opartego na regresji logistycznej z gradientem prostym	150
Prognozowanie kliknięć reklam z wykorzystaniem regresji logistycznej z gradientem prostym	154
Trening modelu opartego na regresji logistycznej ze stochastycznym gradientem prostym	156
Trening modelu opartego na regresji logistycznej z regularyzacją	158
Selekcja cech w regularyzacji L1	159
Trening modelu na dużym zbiorze danych z uczeniem online	160
Klasyfikacja wieloklasowa	163
Implementacja regresji logistycznej za pomocą pakietu TensorFlow	165
Selekcja cech z wykorzystaniem lasu losowego	167
Podsumowanie	168
Ćwiczenia	168
Rozdział 6. Skalowanie modelu prognozującego do terabajtowych dzienników kliknięć	169
Podstawy Apache Spark	170
Komponenty	170
Instalacja	172
Uruchamianie i wdrażanie programów	173
Programowanie z wykorzystywaniem modułu PySpark	174
Trenowanie modelu na bardzo dużych zbiorach danych za pomocą narzędzia Apache Spark	177
Załadowanie danych o kliknięciach reklam	177
Podzielenie danych i umieszczenie ich w pamięci	180

Zakodowanie „1 z n” cech kategoryalnych	181
Trening i testy modelu regresji logistycznej	184
Inżynieria cech i wartości kategoryalnych przy użyciu narzędzia Apache Spark	186
Mieszanie cech kategoryalnych	186
Interakcja cech, czyli łączenie zmiennych	189
Podsumowanie	192
Ćwiczenia	192
Rozdział 7. Prognozowanie cen akcji za pomocą algorytmów regresji	194
Krótkie wprowadzenie do giełdy i cen akcji	195
Co to jest regresja?	196
Pozyskiwanie cen akcji	197
Pierwsze kroki z inżynierią cech	199
Pozyskiwanie danych i generowanie cech	202
Szacowanie za pomocą regresji liniowej	205
Jak działa regresja liniowa?	206
Implementacja regresji liniowej od podstaw	207
Implementacja regresji liniowej z wykorzystaniem pakietu scikit-learn	210
Implementacja regresji liniowej z wykorzystaniem pakietu TensorFlow	211
Prognozowanie za pomocą regresyjnego drzewa decyzyjnego	212
Przejsięcie od drzewa klasyfikacyjnego do regresyjnego	212
Implementacja regresyjnego drzewa decyzyjnego	214
Implementacja lasu regresyjnego	218
Prognozowanie za pomocą regresji wektorów nośnych	218
Implementacja regresji wektorów nośnych	219
Ocena jakości regresji	220
Prognozowanie cen akcji za pomocą trzech algorytmów regresji	222
Podsumowanie	225
Ćwiczenia	226
Rozdział 8. Prognozowanie cen akcji za pomocą sieci neuronowych	227
Demistyfikacja sieci neuronowych	228
Pierwsze kroki z jednowarstwową siecią neuronową	228
Funkcje aktywacji	229
Propagacja wstecz	231
Wprowadzanie kolejnych warstw do sieci neuronowej i uczenie głębokie	232
Tworzenie sieci neuronowej	234
Implementacja sieci neuronowej od podstaw	234
Implementacja sieci neuronowej przy użyciu pakietu scikit-learn	237
Implementacja sieci neuronowej przy użyciu pakietu TensorFlow	237
Dobór właściwej funkcji aktywacji	239
Zapobieganie nadmiernemu dopasowaniu sieci	240
Dropout	240
Wczesne zakończenie treningu	241
Prognozowanie cen akcji za pomocą sieci neuronowej	242
Trening prostej sieci neuronowej	242
Dostrojenie parametrów sieci neuronowej	243
Podsumowanie	248
Ćwiczenie	249

Rozdział 9. Badanie 20 grup dyskusyjnych przy użyciu technik analizy tekstu	250
Jak komputery rozumieją ludzi, czyli przetwarzanie języka naturalnego	251
Czym jest przetwarzanie języka naturalnego?	251
Historia przetwarzania języka naturalnego	252
Zastosowania przetwarzania języka naturalnego	253
Przegląd bibliotek Pythona i podstawy przetwarzania języka naturalnego	254
Instalacja najważniejszych bibliotek	254
Korpusy	255
Tokenizacja	257
Oznaczanie części mowy	258
Rozpoznawanie jednostek nazwanych	259
Stemming i lematyzacja	260
Modelowanie semantyczne i tematyczne	261
Pozyskiwanie danych z grup dyskusyjnych	261
Badanie danych z grup dyskusyjnych	264
Przetwarzanie cech danych tekstowych	267
Zliczanie wystąpień wszystkich tokenów	267
Wstępne przetwarzanie tekstu	270
Usuwanie stop-słów	270
Upraszczenie odmian	271
Wizualizacja danych tekstowych z wykorzystaniem techniki t-SNE	272
Co to jest redukcja wymiarowości?	272
Redukcja wymiarowości przy użyciu techniki t-SNE	273
Podsumowanie	276
Ćwiczenia	276
Rozdział 10. Wyszukiwanie ukrytych tematów w grupach dyskusyjnych poprzez ich klastrowanie i modelowanie tematyczne	277
Nauka bez wskazówek, czyli uczenie nienadzorowane	278
Klastrowanie grup dyskusyjnych metodą k-średnich	280
Jak działa klastrowanie metodą k-średnich?	280
Implementacja klastrowania metodą k-średnich od podstaw	281
Implementacja klastrowania metodą k-średnich z wykorzystaniem pakietu scikit-learn	289
Dobór wartości k	291
Klastrowanie danych z grup dyskusyjnych metodą k-średnich	293
Odkrywanie ukrytych tematów grup dyskusyjnych	296
Modelowanie tematyczne z wykorzystaniem nieujemnej faktoryzacji macierzy	297
Modelowanie tematyczne z wykorzystaniem ukrytej alokacji Dirichleta	300
Podsumowanie	303
Ćwiczenia	304
Rozdział 11. Dobre praktyki uczenia maszynowego	305
Proces rozwiązywania problemów uczenia maszynowego	306
Dobre praktyki przygotowywania danych	307
Dobra praktyka nr 1. Dokładne poznanie celu projektu	307
Dobra praktyka nr 2. Zbieranie wszystkich istotnych pól	307

Dobra praktyka nr 3. Ujednolicenie danych	308
Dobra praktyka nr 4. Opracowanie niekompletnych danych	308
Dobra praktyka nr 5. Przechowywanie dużych ilości danych	311
Dobre praktyki tworzenia zbioru treningowego	312
Dobra praktyka nr 6. Oznaczanie cech kategoryalnych liczbami	312
Dobra praktyka nr 7. Rozważenie kodowania cech kategoryalnych	313
Dobra praktyka nr 8. Rozważenie selekcji cech i wybór odpowiedniej metody	313
Dobra praktyka nr 9. Rozważenie redukcji wymiarowości i wybór odpowiedniej metody	314
Dobra praktyka nr 10. Rozważenie normalizacji cech	315
Dobra praktyka nr 11. Inżynieria cech na bazie wiedzy eksperckiej	316
Dobra praktyka nr 12. Inżynieria cech bez wiedzy eksperckiej	316
Dobra praktyka nr 13. Dokumentowanie procesu tworzenia cech	318
Dobra praktyka nr 14. Wyodrębnianie cech z danych tekstowych	318
Dobre praktyki trenowania, oceniania i wybierania modelu	322
Dobra praktyka nr 15. Wybór odpowiedniego algorytmu początkowego	323
Dobra praktyka nr 16. Zapobieganie nadmiernemu dopasowaniu	325
Dobra praktyka nr 17. Diagnostowanie nadmiernego i niedostatecznego dopasowania	325
Dobra praktyka nr 18. Modelowanie dużych zbiorów danych	327
Dobre praktyki wdrażania i monitorowania modelu	328
Dobra praktyka nr 19. Zapisywanie, ładowanie i wielokrotne stosowanie modelu	328
Dobra praktyka nr 20. Monitorowanie skuteczności modelu	330
Dobra praktyka nr 21. Regularne aktualizowanie modelu	331
Podsumowanie	331
Ćwiczenia	331
Rozdział 12. Kategoryzacja zdjęć odzieży przy użyciu konwolucyjnej sieci neuronowej	332
Bloki konstrukcyjne konwolucyjnej sieci neuronowej	333
Warstwa konwolucyjna	333
Warstwa nieliniowa	335
Warstwa redukująca	335
Budowanie konwolucyjnej sieci neuronowej na potrzeby klasyfikacji	337
Badanie zbioru zdjęć odzieży	338
Klasyfikowanie zdjęć odzieży za pomocą konwolucyjnej sieci neuronowej	342
Tworzenie sieci	342
Trening sieci	345
Wizualizacja filtrów konwolucyjnych	347
Wzmacnianie konwolucyjnej sieci neuronowej poprzez uzupełnianie danych	349
Odwracanie obrazów w poziomie i pionie	349
Obracanie obrazów	351
Przesuwanie obrazów	352
Usprawnianie klasyfikatora obrazów poprzez uzupełnianie danych	354
Podsumowanie	356
Ćwiczenia	356

Rozdział 13. Prognozowanie sekwencji danych przy użyciu rekurencyjnej sieci neuronowej	357
Wprowadzenie do uczenia sekwencyjnego	358
Architektura rekurencyjnej sieci neuronowej na przykładzie	358
Mechanizm rekurencyjny	359
Sieć typu „wiele do jednego”	361
Sieć typu „jedno do wielu”	362
Sieć synchroniczna typu „wiele do wielu”	362
Sieć niesynchroniczna typu „wiele do wielu”	363
Trening rekurencyjnej sieci neuronowej	364
Długoterminowe zależności i sieć LSTM	365
Analiza recenzji filmowych za pomocą sieci neuronowej	367
Analiza i wstępne przetworzenie recenzji	368
Zbudowanie prostej sieci LSTM	370
Poprawa skuteczności poprzez wprowadzenie dodatkowych warstw	372
Pisanie nowej powieści „Wojna i pokój” za pomocą rekurencyjnej sieci neuronowej	374
Pozyskanie i analiza danych treningowych	374
Utworzenie zbioru treningowego dla generatora tekstu	376
Utworzenie generatora tekstu	378
Trening generatora tekstu	380
Zaawansowana analiza języka przy użyciu modelu Transformer	383
Architektura modelu	383
Samouwaga	383
Podsumowanie	386
Ćwiczenia	386
Rozdział 14. Podejmowanie decyzji w skomplikowanych warunkach z wykorzystaniem uczenia przez wzmacnianie	387
Przygotowanie środowiska do uczenia przez wzmacnianie	388
Instalacja biblioteki PyTorch	388
Instalacja narzędzi OpenAI Gym	390
Wprowadzenie do uczenia przez wzmacnianie z przykładami	391
Komponenty uczenia przez wzmacnianie	391
Sumaryczna nagroda	392
Algorytmy uczenia przez wzmacnianie	393
Problem FrozenLake i programowanie dynamiczne	394
Utworzenie środowiska FrozenLake	394
Rozwiązanie problemu przy użyciu algorytmu iteracji wartości	397
Rozwiązanie problemu przy użyciu algorytmu iteracji polityki	400
Metoda Monte Carlo uczenia przez wzmacnianie	403
Utworzenie środowiska Blackjack	403
Ocenianie polityki w metodzie Monte Carlo	405
Sterowanie Monte Carlo z polityką	407
Problem taksówkarza i algorytm Q-uczenia	411
Utworzenie środowiska Taxi	411
Implementacja algorytmu Q-uczenia	414
Podsumowanie	418
Ćwiczenia	418
Skorowidz	419

Prognozowanie kliknięć reklam internetowych przy użyciu algorytmów drzewiastych

W poprzednim rozdziale zbudowałeś klasyfikator obrazów. W tym i następnym rozdziale rozwiążesz jeden z najbardziej typowych, opartych na danych, problemów z internetowej branży reklamowej: prognozowanie kliknięć reklam. Na podstawie informacji o użytkownikach i odwiedzanych przez nich stron model będzie określał prawdopodobieństwo kliknięcia danej reklamy. Skupimy się na algorytmach drzewiastych (drzewie decyzyjnym, lesie losowym i drzewach ze wzmocnieniem gradientowym) i wykorzystamy je do rozwiązania problemu wagi miliardów dolarów. Będziemy badać drzewa decyzyjne od korzeni do liści, a także wersję zagregowaną algorytmu, czyli las drzew. Nie jest to rozdział wyłącznie teoretyczny. Zawiera opisy wielu obliczeń i implementacji modeli od podstaw. Wykorzystasz popularne, drzewiaste pakiety Pythona: *XGBoost* i *scikit-learn*.

W tym rozdziale są opisane następujące zagadnienia:

- dwa rodzaje cech: liczbowe i kategoryjne,
- mechanizm działania drzewa decyzyjnego,
- implementacja drzewa decyzyjnego,

- prognozowanie kliknięć,
- gromadzenie drzew i agregacja bootstrap,
- mechanizm działania lasu losowego,
- prognozowanie kliknięć za pomocą lasu losowego,
- drzewo decyzyjne ze wzmocnieniem gradientowym,
- implementacja drzewa decyzyjnego ze wzmocnieniem gradientowym za pomocą pakietu XGBoost.

Wprowadzenie do prognozowania kliknięć reklam

Reklama internetowa to branża warta miliardy dolarów. Reklamy są publikowane w różnych formatach, m.in. w postaci banerów składających się z tekstu, obrazów, obiektów flash oraz treści multimedialnych, takich jak audio i wideo. Reklamodawcy i agencje, aby dotrzeć do potencjalnych klientów i dostarczyć im treści, umieszczają reklamy w różnych witrynach internetowych, a nawet w aplikacjach mobilnych w całym internecie.

Reklama internetowa jest jednym z najszerszych obszarów zastosowań uczenia maszynowego. Oczywiście reklamodawcy i konsumenci są żywotnie zainteresowani dobrze ukierunkowanymi reklamami. Od 20 lat modele uczenia maszynowego są wykorzystywane na szeroką skalę do prognozowania skuteczności ukierunkowanych reklam, tj. wyliczania prawdopodobieństwa, że odbiorca z określonej grupy wiekowej będzie zainteresowany danym produktem, że klient o określonych dochodach kupi produkt po obejrzeniu reklamy, że osoba często odwiedzająca witrynę sportową poświęci więcej czasu na obejrzenie reklamy itd. Najczęściej stosowaną miarą skuteczności reklamy jest **współczynnik klikalności** (ang. *click-through rate*, CTR), czyli stosunek kliknięć do łącznej liczby wyświetleń danej reklamy. Im wyższa jego wartość, tym lepiej ukierunkowana jest reklama i tym bardziej skuteczna kampania internetowa.

Prognozowaniu kliknięć reklam towarzyszą zarówno potęga, jak i wyzwania urządzenia maszynowego. Prognozowanie polega głównie na binarnym klasyfikowaniu, czy określona reklama na danej stronie (lub w danej aplikacji) zostanie kliknięta przez danego użytkownika. Wykorzystuje się przy tym trzy następujące aspekty predykcyjne:

- treść reklamy i zawartą w niej informację (kategoria, położenie, tekst, format itp.),
- treść strony i informacje o jej właścicielu (kategoria, kontekst, domena itp.),
- informacje o użytkowniku (wiek, płeć, zawód, dochody, zainteresowania, historia przeglądania stron, wykorzystywane urządzenie itp.).

Załóżmy, że pracujesz w agencji obsługującej kilku klientów i Twoim zadaniem jest prezentowanie właściwych reklam odpowiednim odbiorcom. Przyjmijmy też, że masz zbiór danych wyodrębniony z milionów rekordów z kampanii przeprowadzonych miesiąc temu. Tabela 4.1

przedstawia ich przykładowy mały fragment. W rzeczywistości cech predykcyjnych mogą być tysiące. Musisz opracować model klasyfikacyjny, który nauczy się tych reklam i będzie prognozował ich skuteczność w przyszłości.

Tabela 4.1. Treningowe i testowe próbki danych reklamowych

Kategoria reklamy	Kategoria witryny	Domena witryny	Wiek użytkownika	Płeć	Zawód	Zaint. sportowe	Zaint. techniczne	Klik
Motoryzacja	Wiadomości	cnn.com	25 – 34	M	Specjalista	Tak	Tak	1
Moda	Wiadomości	bbc.com	35 – 54	K	Specjalista	Nie	Nie	0
Motoryzacja	Edukacja	onlinestudy.com	17 – 24	K	Student	Tak	Tak	0
Jedzenie	Rozrywka	movie.com	25 – 34	M	Urzędnik	Tak	Tak	1
Moda	Sport	football.com	55+	M	Emeryt	Tak	Nie	0
...
Jedzenie	Wiadomości	abc.com	17 – 24	M	Student	Tak	Tak	?
Motoryzacja	Rozrywka	movie.com	35 – 54	K	Specjalista	Tak	Nie	?

Jak widać, większość cech jest kategoryalna. W praktyce dane są liczbowe lub kategoryalne. Przyjrzyjmy im się dokładniej w następnym podrozdziale.

Wprowadzenie do dwóch typów danych: liczbowych i kategoryalnych

Dane w zbiorze przedstawionym w poprzednim podrozdziale są **kategoryalne**, na przykład mężczyzna lub kobieta, jedna z czterech grup wiekowych, jedna z kategorii witryn, sportowe zainteresowania użytkownika. Są to inne cechy niż liczbowe, z którymi miałeś do czynienia do tej pory.

Cechy kategoryalne, zwane również **jakościowymi**, reprezentują charakterystyczne, odrębne grupy danych i są policzalne. Mogą, ale nie muszą, tworzyć logiczne szeregi. Na przykład dochód gospodarstwa domowego (niski, średni, wysoki) jest cechą **porządkową**, podczas gdy kategoria reklamy taką cechą nie jest.

Natomiast cechy liczbowe, zwane również **ilościowymi**, są miarami w znaczeniu matematycznym i są oczywiście porządkowe. Na przykład częstość występowania słowa w tekście i wariant TF-IDF to, odpowiednio, liczbowe cechy dyskretne i ciągłe. Zbiór danych kardiogramów (<https://archive.ics.uci.edu/ml/datasets/Cardiotocography>) z poprzedniego rozdziału

zawiera dane liczbowe zarówno o cechach dyskretnych (na przykład liczba przyspieszeń na sekundę lub liczba ruchów płodu na sekundę), jak i ciągłych (na przykład średnia zmienność długoterminowa).

Cechy kategoryjne mogą również być wartościami liczbowymi. Na przykład liczby od 1 do 12 mogą reprezentować miesiące w roku, a 1 i 0 mogą oznaczać mężczyznę i kobietę. Nie mają one jednak implikacji matematycznych.

Opisany w poprzednim rozdziale naiwny klasyfikator Bayesa działa zarówno z cechami liczbowymi, jak i kategoryjnymi. Szansa, prawdopodobieństwo $P(x | y)$ lub $P(\text{cecha} | \text{klasa})$ są wyliczane w ten sam sposób. Natomiast maszyna wektorów nośnych do wyliczenia i zmaksymalizowania marginesu wymaga cech liczbowych.

Teraz zastanówmy się nad możliwością prognozowania kliknięć za pomocą naiwnego klasyfikatora Bayesa i wyjaśnieniem reklamodawcom zasady działania modelu. Mogą oni jednak mieć trudności ze zrozumieniem prawdopodobieństwa *a priori*, prawdopodobieństwa występowania poszczególnych atrybutów i ich iloczynów. Czy istnieje klasyfikator, który jest łatwy w interpretacji, potrafi bezpośrednio przetwarzać dane kategoryjne i który można wyjaśnić klientom? Odpowiedź brzmi: drzewo decyzyjne!

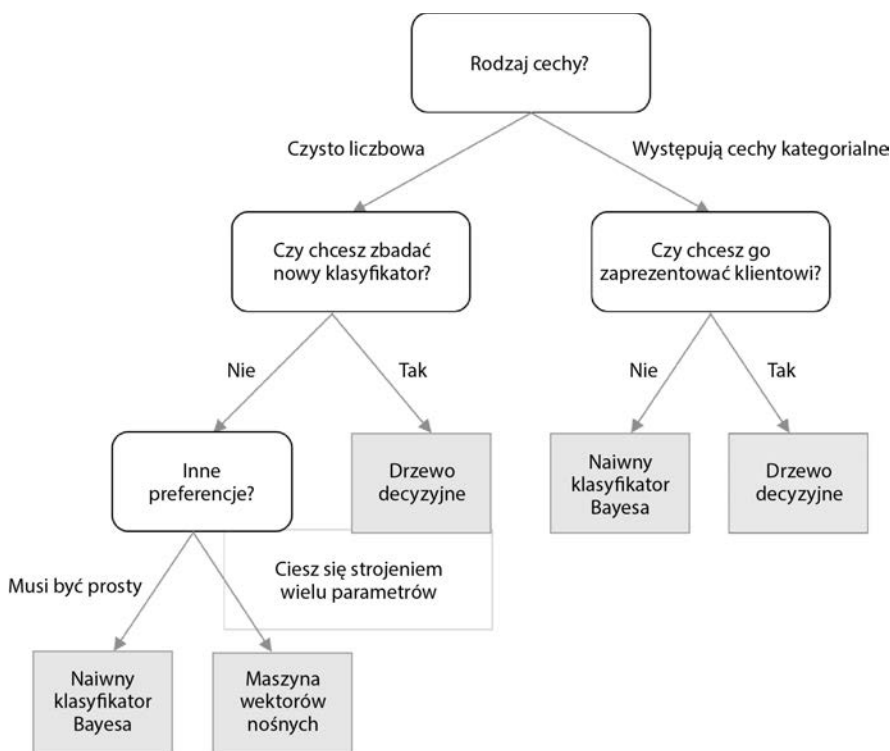
Badanie drzewa decyzyjnego od korzeni do liści

Drzewo decyzyjne jest grafem, czyli sekwencyjnym diagramem, ilustrującym wszystkie możliwe wybory oraz ich wyniki. Każdy **węzeł** wewnętrzny, począwszy od **korzenia**, reprezentuje bazę do podejmowania decyzji, a każda **gałąź** reprezentuje przejście do innego węzła. Ponadto każdy **końcowy węzeł**, czyli **liść**, reprezentuje uzyskany wynik.

Załóżmy, że podjęliśmy kilka decyzji, które doprowadziły nas do rozwiązania naszego problemu reklamowego przy użyciu drzewa decyzyjnego. Ilustruje to rysunek 4.1.

Pierwszy wybór, czyli korzeń (ang. *root*), dotyczy rodzaju cechy: liczbowej lub kategoryjnej. Dane o kliknięciach reklam są głównie kategoryjne, więc trafiają do właściwej gałęzi. W kolejnym węźle Twoja praca musi być zrozumiała dla klientów nietechnicznych. Przechodzimy więc po właściwej gałęzi do liścia, w którym wybieramy drzewo decyzyjne.

Przyjrzyj się różnym ścieżkom i sprawdź, jakie problemy możesz rozwiązywać. Drzewo decyzyjne w serii testów (reprezentowanych jako gałęzie) wiąże próbki z klasami (reprezentowanymi jako węzły). W każdym węźle pojawia się pytanie dotyczące wartości i rodzaju cechy. W zależności od odpowiedzi próbki są dzielone na podzbiory. Kolejne testy są przeprowadzane do chwili podjęcia decyzji dotyczącej wyboru ostatecznej etykiety próbki. Ścieżka od korzenia do końcowego liścia reprezentuje proces podejmowania decyzji i zasady klasyfikacji.



Rysunek 4.1. Wyszukiwanie odpowiedniego algorytmu przy użyciu drzewa decyzyjnego

W prostszym scenariuszu, pokazanym na rysunku 4.2, w którym jest prognozowane kliknięcie lub brak kliknięcia reklamy autonomicznego samochodu, można ręcznie skonstruować klasyfikator drzewa decyzyjnego, operujący na dostępnym zbiorze danych. Na przykład, jeżeli użytkownik interesuje się techniką i posiada samochód, z dużym prawdopodobieństwem kliknie reklamę. Natomiast osoba spoza tej podgrupy, na przykład kobieta o wysokich dochodach, raczej tego nie zrobi. Przetrenowany klasyfikator można wykorzystać do prognozowania na podstawie nowych danych wejściowych, czy użytkownik kliknie reklamę, czy nie.

Jak się już przekonałeś, po skonstruowaniu drzewa decyzyjnego klasyfikacja nowej próbki jest prosta: zaczynając od korzenia, sprawdzamy warunki i podążamy zgodnie z odpowiednimi gałęziami, aż do osiągnięcia końcowego liścia i przypisania etykiety do nowej próbki.

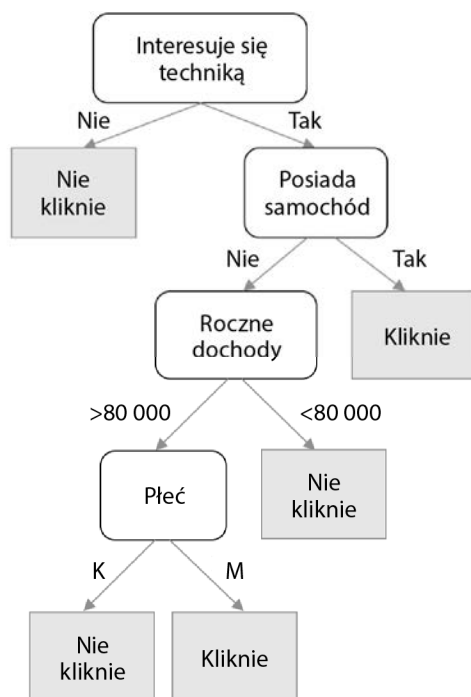
Jak zatem buduje się prawidłowe drzewo decyzyjne?

Budowanie drzewa decyzyjnego

Drzewo decyzyjne buduje się, sukcesywnie dzieląc próbki treningowe na kolejne podzbiory. Proces dzielenia jest rekurencyjnie powtarzany w każdym podzbiorze. Za każdym razem w węźle są sprawdzane cechy podzbioru zgodnie z zadanymi warunkami. Jeżeli wszystkie próbki

Płeć	Roczne dochody	Posiada samochód	Interesuje się techniką	Kliknie
M	200 000	Tak	Tak	1
K	5000	Nie	Nie	0
K	100 000	Tak	Tak	1
M	10 000	Tak	Nie	0
M	80 000	Nie	Nie	0
...
...

M	120 000	Tak	Tak	?
K	70 000	Nie	Tak	?



Rysunek 4.2. Prognozowanie kliknięcia lub jego braku za pomocą przetrenowanego klasyfikatora drzewa decyzyjnego

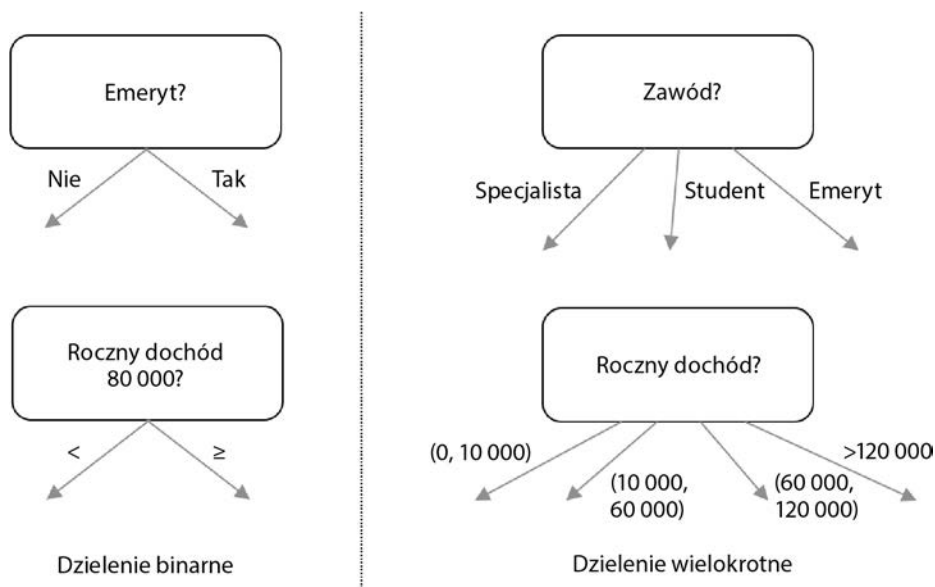
w podzbiorze mają tę samą etykietę lub gdy dalsze dzielenie nie poprawi czystości klasy, proces rekurencyjnego dzielenia w tym węźle się kończy.

Teoretycznie zbiór o n różnych cechach (liczbowych lub kategoryalnych) można podzielić binarnie na n sposobów (na przykład według odpowiedzi *Tak / Nie*), nie mówiąc już o innych metodach dzielenia, na przykład potrójnych lub poczwórnych. Metody podziału ilustruje rysunek 4.3.

Dla m -wymiarowego zbioru danych, dzielonego bez uwzględniania kolejności cech, istnieje n^m możliwych drzew.

Zostało opracowanych wiele algorytmów efektywnego i dokładnego konstruowania drzewa decyzyjnego, z których najpopularniejsze to:

- **Iteracyjny dychotomizer 3** (ang. *Iterative Dichotomiser 3*, ID3). Algorytm ten „zachłannie” przeszukuje drzewo od góry do dołu pod kątem najlepszego atrybutu podziału zbioru danych w każdej iteracji bez cofania się.
- **C4.5**. Udoskonalona wersja algorytmu ID3, umożliwiająca cofanie się. Algorytm przechodzi przez zbudowane drzewo i zastępuje gałęzie liśćmi, jeśli w ten sposób poprawi się czystość podzbioru.



Rysunek 4.3. Przykłady binarnego i wielokrotnego dzielenia zbioru próbek

- **Drzewo klasyfikacyjne i regresyjne** (ang. *Classification and Regression Tree*, CART). Algorytm buduje drzewo, wykorzystując dzielenie binarne, które wkrótce zostanie opisane bardziej szczegółowo.
- **Automatyczny, chi-kwadratowy detektor interakcji** (ang. *Chi-squared Automatic Interaction Detector*, CHAID). Algorytm często stosowany w marketingu bezpośrednim, wykorzystujący zaawansowane metody statystyczne. W skrócie, określa on optymalny sposób łączenia zmiennych predykcyjnych w celu jak najlepszego opisanie wyniku.

Podstawową ideą powyższych algorytmów jest zachłanne rozwijanie drzewa z zastosowaniem serii lokalnych optymalizacji przy wybieraniu najistotniejszej cechy, według której ma być dzielony zbiór danych. Wybór istotnej cechy i jej optymalnej wartości niezbędnej do podziału zbioru zostanie opisany w następnym podrozdziale.

Najpierw szczegółowo zbadamy algorytm CART, a następnie zaimplementujemy go, ponieważ jest najbardziej godny uwagi. Drzewo jest budowane poprzez binarne dzielenie węzła na lewy i prawy. Po każdym podziale algorytm zachłannie wyszukuje najbardziej istotną kombinację cechy i jej wartości. Za pomocą funkcji oceniającej jakość podziału sprawdzane są wszystkie możliwe kombinacje. Algorytm po wybraniu cechy i wartości dzieli według nich zbiór danych w następujący sposób:

- Próbki, których cechy kategoryjne mają wybraną wartość lub których cechy liczbowe mają wartość większą od wybranej, są umieszczane w prawym nowym węźle.
- Pozostałe próbki są umieszczane w lewym nowym węźle.

Proces dzielenia jest powtarzany rekurencyjnie i próbki są dzielone na dwie podgrupy. Gdy zbiór zostanie uporządkowany, proces dzielenia kończy się na podgrupie spełniającej jedno z dwóch poniższych kryteriów:

- **Minimalna liczba próbek w nowym węźle:** jeżeli liczba próbek nie jest większa od minimalnej wymaganej do dalszego podziału, dzielenie się kończy, aby zapobiec nadmiernemu dostosowaniu się drzewa do zbioru treningowego i w rezultacie nadmiernemu dopasowaniu modelu.
- **Maksymalna głębokość drzewa:** węzeł nie jest dzielony, jeżeli głębokość drzewa, zdefiniowana jako liczba podziałów dokonanych od góry do dołu, począwszy od węzła głównego, a skończywszy na węźle końcowym, jest nie mniejsza od zadanej maksymalnej wartości. Głębsze drzewo byłoby bardziej dostosowane do zbioru treningowego i doprowadziłoby do nadmiernego dopasowania modelu.

Węzeł bez gałęzi staje się liściem, a dominująca w nim klasa próbek prognozowanym wynikiem. Po dokonaniu wszystkich podziałów drzewo jest gotowe. Węzłom końcowym są przypisane etykiety, a węzły pośrednie są umieszczone we wszystkich punktach podziału (cecha + wartość).

Teraz zajmiemy się wyborem optymalnej cechy i wartości do dzielenia zbioru, a następnie, zgodnie z obietnicą, zaimplementujemy algorytm CART od podstaw.

Wskaźniki jakości podziału zbioru

Wybierając najlepszą kombinację cechy i wartości jako kryterium podziału zbioru, do oceny jakości podziału stosuje się dwa wskaźniki: **zanieczyszczenie Giniego** (ang. *Gini impurity*) i **przyrost informacji** (ang. *information gain*).

Zanieczyszczenie Giniego

Zanieczyszczenie Giniego, jak sugeruje nazwa, określa niedoskonałość rozkładu klas, czyli stopień ich zmieszania. Załóżmy, że mamy zbiór danych o K klasach, a próbki należące do klasy k ($1 \leq k \leq K$) stanowią część f_k całego zbioru ($0 \leq f_k \leq 1$). Zanieczyszczenie Giniego takiego zbioru określa następująca formuła:

$$\text{Zanieczyszczenie Giniego} = 1 - \sum_{k=1}^K f_k^2$$

Im mniejsze jest zanieczyszczenie Giniego, tym „czystszy” jest zbiór danych. Załóżmy dla przykładu, że mamy zbiór składający się z jednej klasy. Jej udział w zbiorze jest równy 1, a pozostałych klas jest równy 0. Zatem zanieczyszczenie takiego zbioru jest równe $1 - (1^2 + 0^2) = 0$. Innym przykładem jest duży zbiór wyników rzutu monetą. Połowę próbek stanowią orły, a drugą połowę reszki. Zanieczyszczenie Giniego tego zbioru jest równe $1 - (0,5^2 + 0,5^2) = 0,5$.

Zanieczyszczenie Giniego zbioru binarnego o różnych udziałach próbek klasy dodatniej można wizualizować za pomocą następującego kodu:

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

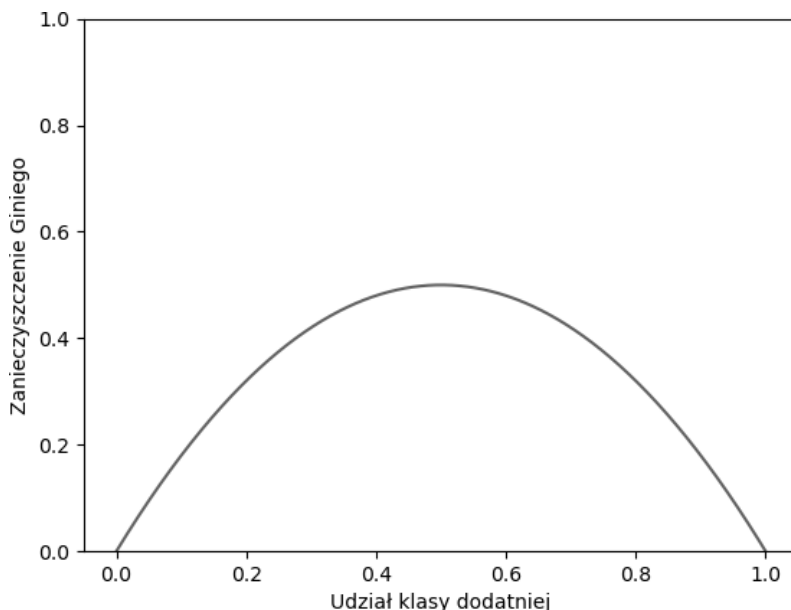
Udział klasy dodatniej w zbiorze zawiera się w przedziale od 0 do 1:

```
>>> pos_fraction = np.linspace(0.00, 1.00, 1000)
```

Poniższy kod wylicza wartości zanieczyszczenia Giniego, a następnie tworzy wykres zanieczyszczenia w funkcji udziału klasy dodatniej. Wyrażenie `1-pos_fraction` opisuje udział klasy ujemnej:

```
>>> gini = 1 - pos_fraction**2 - (1-pos_fraction)**2
>>> plt.plot(pos_fraction, gini)
>>> plt.ylim(0, 1)
>>> plt.xlabel('Udział klasy dodatniej')
>>> plt.ylabel('Zanieczyszczenie Giniego')
>>> plt.show()
```

Rysunek 4.4 przedstawia wynik działania kodu.



Rysunek 4.4. Wykres zanieczyszczenia Giniego w funkcji udziału klasy dodatniej

Jak widać, w przypadku binarnym zanieczyszczenie osiąga maksymalną wartość 0,5, jeżeli udział klasy dodatniej wynosi 50%, a dla udziału 100% lub 0% jest równe 0.

Mając etykiety zbioru danych, możesz zaimplementować następującą funkcję wyliczającą zanieczyszczenie Giniego:

```
>>> def gini_impurity(labels):
...     # Zbiór pusty jest również czysty
```

```
...     if not labels:
...         return 0
...     # Zliczenie wystąpień każdej etykiety
...     counts = np.unique(labels, return_counts=True)[1]
...     fractions = counts / float(len(labels))
...     return 1 - np.sum(fractions ** 2)
```

Przetestuj funkcję na kilku przykładach:

```
>>> print(f'gini_impurity([1, 1, 0, 1, 0]):.4f')
0.4800
>>> print(f'gini_impurity([1, 1, 0, 1, 0, 0]):.4f')
0.5000
>>> print(f'gini_impurity([1, 1, 1, 1]):.4f')
0.0000
```

Aby ocenić jakość podziału, po prostu sumuje się wartości zanieczyszczenia Giniego dla wszystkich otrzymanych podgrup, stosując proporcje każdej podgrupy jako wagi. Jak poprzednio, im mniejsza jest ważona suma wskaźników, tym lepszy jest podział.

Przeanalizujmy przykład reklamy samochodu samosterującego, pokazany w tabeli 4.2. Dane zostały podzielone na grupy raz według płci i raz według zainteresowania techniką.

Tabela 4.2. Podział danych według płci i zainteresowania techniką

Płeć	Zainteresowanie techniką	Klik	Podział wg płci	Płeć	Zainteresowanie techniką	Klik	Podział wg zainteresowań
M	Tak	1	Grupa 1	M	Tak	1	Grupa 1
K	Nie	0	Grupa 2	K	Nie	0	Grupa 2
K	Tak	1	Grupa 2	K	Tak	1	Grupa 1
M	Nie	0	Grupa 1	M	Nie	0	Grupa 2
M	Nie	1	Grupa 1	M	Nie	1	Grupa 2
Podział wg płci				Podział wg zainteresowań			

W pierwszym przypadku ważne zanieczyszczenie Giniego jest równe:

$$\begin{aligned} \text{Zanieczyszczenie Giniego 1} &= \frac{3}{5} \left\{ 1 - \left[\left(\frac{2}{3} \right)^2 + \left(\frac{1}{3} \right)^2 \right] \right\} + \frac{2}{5} \left\{ 1 - \left[\left(\frac{1}{2} \right)^2 + \left(\frac{1}{2} \right)^2 \right] \right\} \\ &= 0,467 \end{aligned}$$

W drugim przypadku zanieczyszczenie wynosi:

$$\begin{aligned} \text{Zanieczyszczenie Giniego 2} &= \frac{2}{5} \{ 1 - [1^2 + 0^2] \} + \frac{3}{5} \left\{ 1 - \left[\left(\frac{1}{3} \right)^2 + \left(\frac{2}{3} \right)^2 \right] \right\} \\ &= 0,267 \end{aligned}$$

Zatem lepszą strategią jest dzielenie zbioru według zainteresowania techniką niż według płci.

Przyrost informacji

Inny wskaźnik, przyrost informacji, opisuje poprawę czystości zbioru po podziale, czyli, innymi słowy, zmniejszenie niepewności wyniku podziału. Im większa wartość tego wskaźnika, tym lepszy podział. Wylicza się go jako różnicę entropii przed podziałem i po nim.

Entropia jest probabilistyczną miarą niepewności. Dla zbioru danych o K klasach, w którym próbki należące do klasy k ($1 \leq k \leq K$) stanowią część f_k całego zbioru ($0 \leq f_k \leq 1$), entropia jest zdefiniowana w następujący sposób:

$$\text{Entropia} = - \sum_{k=1}^K f_k \cdot \log_2 f_k$$

Niska entropia implikuje czystszy zbiór o mniejszej niepewności. W idealnym przypadku, gdy wszystkie próbki zbioru należą tylko do jednej klasy, entropia jest równa $-(1 \cdot \log_2 1 + 0) = 0$. W przykładzie z rzutami monetą entropia wynosi $-(0,5 \cdot \log_2 0,5 + 0,5 \cdot \log_2 0,5) = 1$.

Jak poprzednio, możesz zwizualizować wartość entropii w funkcji udziału dodatniej klasy w zbiorze binarnym za pomocą poniższego kodu:

```
>>> pos_fraction = np.linspace(0.00, 1.00, 1000)
>>> ent = - (pos_fraction * np.log2(pos_fraction) +
...         (1 - pos_fraction) * np.log2(1 - pos_fraction))
>>> plt.plot(pos_fraction, ent)
>>> plt.xlabel('Udział klasy dodatniej')
>>> plt.ylabel('Entropia')
>>> plt.ylim(0, 1)
>>> plt.show()
```

Rysunek 4.5 przedstawia uzyskany wynik.

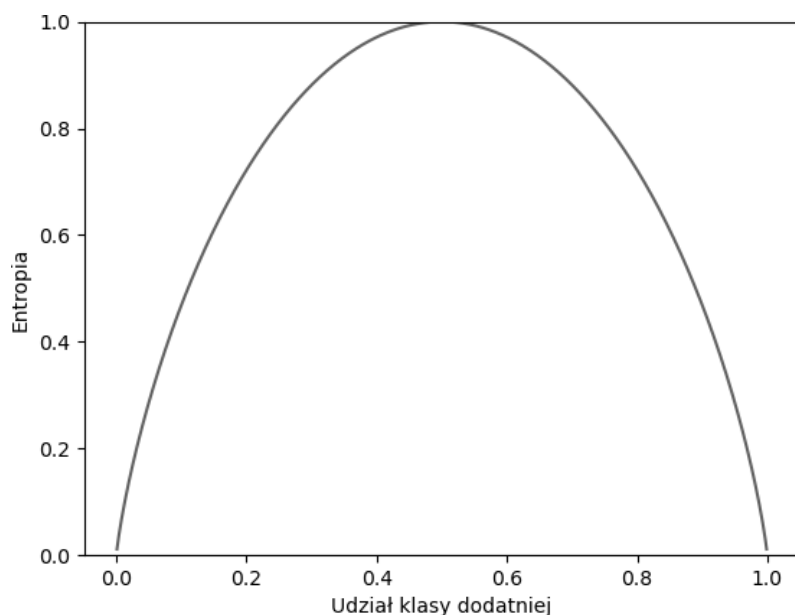
Jak widać, w przypadku binarnym entropia osiąga maksymalną wartość równą 1, jeżeli udział klasy dodatniej wynosi 50%, a dla udziału 100% lub 0% jest równa 0.

Mając etykiety zbioru danych, można zaimplementować następującą funkcję wyliczającą entropię:

```
>>> def entropy(labels):
...     if not labels:
...         return 0
...     counts = np.unique(labels, return_counts=True)[1]
...     fractions = counts / float(len(labels))
...     return - np.sum(fractions * np.log2(fractions))
```

Przetestuj funkcję na kilku przykładach:

```
>>> print(f'{entropy([1, 1, 0, 1, 0]):.4f}')
0.9710
>>> print(f'{entropy([1, 1, 0, 1, 0, 0]):.4f}')
1.0000
>>> print(f'{entropy([1, 1, 1, 1]):.4f}')
-0.0000
```



Rysunek 4.5. Entropia w funkcji udziału klasy dodatniej

Teraz, gdy już w pełni poznałeś entropię, możesz sprawdzić, jak przyrost informacji odzwierciedla spadek entropii po podziale zbioru. Wskaźnik ten definiuje się jako różnicę entropii przed podziałem (rodzic) i po nim (dzieci):

$$\text{Przyrost informacji} = \text{Entropia}(\text{przed}) - \text{Entropia}(\text{po}) = \text{Entropia}(\text{rodzic}) - \text{Entropia}(\text{dzieci})$$

Entropię po podziale wylicza się, podobnie jak zanieczyszczenie Giniego, jako ważoną sumę entropii wszystkich dzieci.

W procesie tworzenia węzła drzewa decyzyjnego celem jest znalezienie takiego punktu podziału, dla którego przyrost informacji jest największy. Ponieważ entropia węzła rodzicielskiego nie zmienia się, wystarczy jedynie wyliczyć entropie węzłów potomnych powstałych po podziale. Najlepszy podział jest taki, w którym entropia węzłów potomnych jest najmniejsza.

Aby lepiej zrozumieć to zagadnienie, ponownie przeanalizujemy przykład reklamy samochodu samotestującego.

Po pierwszym podziale wynikowa entropia jest równa:

$$\begin{aligned} \text{Entropia } 1 &= \frac{3}{5} \left[-\left(\frac{2}{3} \cdot \log_2 \frac{2}{3} + \frac{1}{3} \cdot \log_2 \frac{1}{3} \right) \right] + \frac{2}{5} \left[-\left(\frac{1}{2} \cdot \log_2 \frac{1}{2} + \frac{1}{2} \cdot \log_2 \frac{1}{2} \right) \right] \\ &= 0,951 \end{aligned}$$

Natomiast w drugim:

$$\text{Entropia 2} = \frac{2}{5} [-(1 \cdot \log_2 1 + 0)] + \frac{3}{5} \left[-\left(\frac{1}{3} \cdot \log_2 \frac{1}{3} + \frac{2}{3} \cdot \log_2 \frac{2}{3}\right) \right] = 0,551$$

Do celów badawczych wyliczmy również przyrost informacji:

$$\text{Entropia przed} = -\left(\frac{3}{4} \cdot \log_2 \frac{3}{5} + \frac{2}{5} \cdot \log_2 \frac{2}{5}\right) = 0,971$$

$$\text{Przyrost informacji 1} = 0,971 - 0,951 = 0,020$$

$$\text{Przyrost informacji 2} = 0,971 - 0,551 = 0,420$$

Zgodnie z powyższymi wyliczeniami preferowany jest drugi podział, co potwierdza wniosek uzyskany po wyliczeniu zanieczyszczenia Giniego.

Zazwyczaj wybór pomiędzy zanieczyszczeniem Giniego a przyrostem informacji ma niewielki wpływ na skuteczność przetrenowanego modelu drzewa decyzyjnego. Oba wskaźniki mierzą ważne zanieczyszczenie zbioru po podziale. Można je połączyć w następującej funkcji wyliczającej ważne zanieczyszczenie:

```
>>> criterion_function = {'gini': gini_impurity,
...                        'entropy': entropy}
>>> def weighted_impurity(groups, criterion='gini'):
...     """
...     Funkcja wyliczająca ważne zanieczyszczenie zbioru po podziale
...     @param groups: lista węzłów potomnych, z których każdy
...                     zawiera listę etykiet klas
...     @param criterion: wskaźnik jakości podziału
...                     'gini' oznacza zanieczyszczenie Giniego,
...                     a 'entropy' przyrost informacji
...     @return: float, weighted impurity
...     """
...     total = sum(len(group) for group in groups)
...     weighted_sum = 0.0
...     for group in groups:
...         weighted_sum += len(group) / float(total) *
...             criterion_function[criterion](group)
...     return weighted_sum
```

Przetestuj funkcję na przykładach wcześniej wyliczonych ręcznie:

```
>>> children_1 = [[1, 0, 1], [0, 1]]
>>> children_2 = [[1, 1], [0, 0, 1]]
>>> print(f"Entropia 1: {weighted_impurity(children_1,
...     'entropy'):.4f}")
Entropia 1: 0.9510
>>> print(f"Entropia 2: {weighted_impurity(children_2,
...     'entropy'):.4f}")
Entropia 1: 0.5510
```

Teraz, mając solidną wiedzę na temat wskaźników jakości podziału zbioru, zaimplementuj w następnym podrozdziale od podstaw algorytm CART.

Implementacja drzewa decyzyjnego od podstaw

Algorytm CART zaimplementujesz, wykorzystując prosty zbiór danych przedstawiony w tabeli 4.3.

Tabela 4.3. Przykład danych reklamowych

Zainteresowania	Zawód	Klik
Technika	Specjalista	1
Moda	Student	0
Moda	Specjalista	0
Sport	Student	0
Technika	Student	1
Technika	Emeryt	0
Sport	Specjalista	1

Na początek zadecyduj o pierwszym punkcie podziału, korzeniu, poprzez wypróbowanie wszystkich kombinacji wartości obu cech. W tym celu wylicz zanieczyszczenie Giniego za pomocą zdefiniowanej wcześniej funkcji `weighted_impurity()`.

Jeżeli podzielimy zbiór według zainteresowania użytkownika techniką, w pierwszej grupie znajdą się próbki nr 1, 5 i 6, a w drugiej grupie pozostałe próbki. Zatem w pierwszej grupie będą klasy `[1, 1, 0]`, a w drugiej `[0, 0, 0, 1]`. Zanieczyszczenie Giniego będzie wtedy równe:

$$\text{Gini}(\text{zainteresowania, technika}) = \text{weighted_impurity}([1, 1, 0], [0, 0, 0, 1]) = 0,405$$

Jeżeli podzielimy zbiór według zainteresowania użytkownika modą, w pierwszej grupie znajdą się próbki nr 2 i 3, a w drugiej grupie pozostałe próbki. Zatem w pierwszej grupie będą klasy `[0, 0]`, a w drugiej `[1, 0, 1, 0, 1]`. Zanieczyszczenie Giniego będzie wtedy równe:

$$\text{Gini}(\text{zainteresowania, moda}) = \text{weighted_impurity}([0, 0], [1, 0, 1, 0, 1]) = 0,343$$

Analogicznie wylicz współczynniki zanieczyszczenia dla innych podziałów:

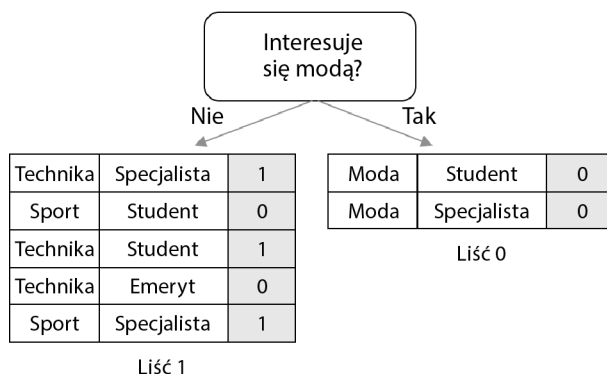
$$\text{Gini}(\text{zainteresowania, sport}) = \text{weighted_impurity}([0, 1], [1, 0, 0, 1, 0]) = 0,486$$

$$\text{Gini}(\text{zawód, specjalista}) = \text{weighted_impurity}([0, 0, 1, 0], [1, 0, 1]) = 0,405$$

$$\text{Gini}(\text{zawód, student}) = \text{weighted_impurity}([0, 0, 1, 0], [1, 0, 1]) = 0,405$$

$$\text{Gini}(\text{zawód, emeryt}) = \text{weighted_impurity}([1, 0, 0, 0, 1, 1], [1]) = 0,429$$

Korzeń będzie dzielony według zainteresowania użytkownika modą, ponieważ ta kombinacja daje najmniejsze ważone zanieczyszczenie i największy przyrost informacji. Pierwszy poziom drzewa będzie więc wyglądał jak na rysunku 4.6.



Rysunek 4.6. Podział danych według zainteresowania użytkownika modą

Jeśli jednopoziomowe drzewo jest wystarczające, można na tym zakończyć proces dzielenia i przypisać prawej gałęzi etykietę 0, a lewej, większej gałęzi etykietę 1. Można też iść dalej tą drogą, budując drugi poziom lewej gałęzi (prawej nie da się bardziej podzielić). Niezbędne wyliczenia są następujące:

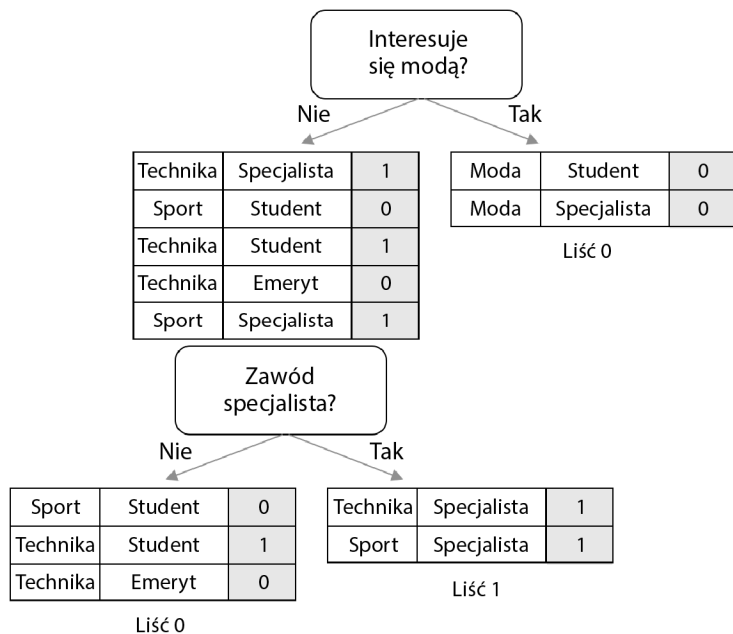
$\text{Gini}(\text{zainteresowania}, \text{technika}) = \text{weighted_impurity}([0, 1], [1, 1, 0]) = 0,467$
 $\text{Gini}(\text{zainteresowania}, \text{sport}) = \text{weighted_impurity}([1, 1, 0], [0, 1]) = 0,467$
 $\text{Gini}(\text{zawód}, \text{specjalista}) = \text{weighted_impurity}([0, 1, 0], [1, 1]) = 0,267$
 $\text{Gini}(\text{zawód}, \text{student}) = \text{weighted_impurity}([1, 0, 1], [0, 1]) = 0,467$
 $\text{Gini}(\text{zawód}, \text{emeryt}) = \text{weighted_impurity}([1, 0, 1, 1], [0]) = 0,300$

Najmniejsze zanieczyszczenie wprowadza podział według zawodu specjalisty. Drzewo przyjmuje wtedy kształt pokazany na rysunku 4.7.

Proces podziału można kontynuować, dopóki drzewo nie osiągnęło maksymalnej głębokości, a węzły zawierają wystarczającą liczbę próbek.

Teraz, po wyjaśnieniu procesu budowania drzewa, nadszedł czas na kodowanie. Zaczniemy od najlepszego kryterium podziału. Ważone zanieczyszczanie dwóch potencjalnych węzłów potomnych wylicza się tak samo jak wcześniej, natomiast wyliczenia dwóch metryk są nieco inne. W celu zwiększenia wydajności danymi wejściowymi będą teraz tablice NumPy. Funkcja wyliczająca zanieczyszczenie Giniego wygląda tak:

```
>>> def gini_impurity_np(labels):
...     # Zbiór pusty jest również czysty
...     if labels.size == 0:
...         return 0
...     # Zliczenie wystąpień każdej etykiety
...     counts = np.unique(labels, return_counts=True)[1]
...     fractions = counts / float(len(labels))
...     return 1 - np.sum(fractions ** 2)
```



Rysunek 4.7. Kolejny podział danych według zawodu specjalisty

Do wyliczenia entropii posłuży poniższa funkcja:

```
>>> def entropy_np(labels):
...     # Zbiór pusty jest również czysty
...     if labels.size == 0:
...         return 0
...     counts = np.unique(labels, return_counts=True)[1]
...     fractions = counts / float(len(labels))
...     return - np.sum(fractions * np.log2(fractions))
```

Funkcję `weighted_impurity()` również zmodyfikuj jak niżej:

```
>>> def weighted_impurity(groups, criterion='gini'):
...     """
...     Funkcja wyliczająca ważone zanieczyszczenie zbioru po podziale
...     @param groups: lista węzłów potomnych, z których każdy
...                     zawiera listę etykiet klas
...     @param criterion: wskaźnik jakości podziału
...                     'gini' oznacza zanieczyszczenie Giniego,
...                     a 'entropy' przyrost informacji
...     @return: float, weighted impurity
...     """
...     total = sum(len(group) for group in groups)
...     weighted_sum = 0.0
...     for group in groups:
...         weighted_sum += len(group) / float(total) *
...             criterion_function_np[criterion](group)
...     return weighted_sum
```

Następnie zdefiniuj funkcję pomocniczą, dzielącą dany węzeł na węzły potomne lewy i prawy na podstawie cechy i wartości. Funkcja sprawdza, czy cecha jest liczbowa, czy kategorialna, a następnie odpowiednio dzieli dane.

```
>>> def split_node(X, y, index, value):
    """
    ...     Funkcja dzieląca zbiór X na podstawie cechy i wartości
    ...     @param X: numpy.ndarray, cechy zbioru
    ...     @param y: numpy.ndarray, docelowy zbiór
    ...     @param index: int, indeks cechy wykorzystywanej do dzielenia
    ...     @param value: wartość cechy wykorzystywanej do dzielenia
    ...     @return: dwie listy w formacie [X, y] reprezentujące
    ...             węzły potomne lewy i prawy
    ...     """
    ...     x_index = X[:, index]
    ...     # Jeżeli cecha jest liczbowa
    ...     if X[0, index].dtype.kind in ['i', 'f']:
    ...         mask = x_index >= value
    ...         # Jeżeli cecha jest kategorialna
    ...     else:
    ...         mask = x_index == value
    ...     # Podział na węzły potomne lewy i prawy
    ...     left = [X[~mask, :], y[~mask]]
    ...     right = [X[mask, :], y[mask]]
    ...     return left, right
```

Po zaimplementowaniu funkcji dzielącej zbiór i oceniającej jakość podziału zdefiniuj funkcję zachłannie przeszukującą wszystkie możliwości podziału i zwracającą najlepszą opcję, zgodną z zadaniem kryterium, wraz z wynikowymi węzłami potomnymi:

```
>>> def get_best_split(X, y, criterion):
    """
    ...     Funkcja wyszukująca najlepszy punkt podziału
    ...     zbioru X, y i zwracająca węzły potomne
    ...     @param X: numpy.ndarray, cechy zbioru
    ...     @param y: numpy.ndarray, docelowy zbiór
    ...     @param criterion: kryterium gini lub entropy
    ...     @return: dict {index: indeks cechy, value: wartość cechy,
    ...                 children: węzły potomne lewy i prawy}
    ...     """
    ...     best_index, best_value, best_score, children = None, None, 1, None
    ...     for index in range(len(X[0])):
    ...         for value in np.sort(np.unique(X[:, index])):
    ...             groups = split_node(X, y, index, value)
    ...             impurity = weighted_impurity(
    ...                 [groups[0][1], groups[1][1]], criterion)
    ...             if impurity < best_score:
    ...                 best_index, best_value, best_score, children =
    ...                     index, value, impurity, groups
    ...     return {'index': best_index, 'value': best_value, 'children': children}
```

Proces dzielenia jest powtarzany rekurencyjnie dla każdego węzła potomnego. Gdy zostaną spełnione kryteria zakończenia, proces jest przerywany, a liściowi jest przypisywana główna etykieta:

```
>>> def get_leaf(labels):
...     # Zwrócenie liścia z główną etykietą
...     return np.bincount(labels).argmax()
```

Na koniec zdefiniuj funkcję rekurencyjną, która łączy wszystko razem:

- Przypisuje etykietę węzłowi liścia, jeśli jeden z dwóch węzłów potomnych jest pusty.
- Przypisuje etykietę węzłowi, jeśli głębokość aktualnej gałęzi przekracza maksymalną dozwoloną.
- Przypisuje etykietę węzłowi, jeśli nie zawiera on wystarczającej liczby próbek wymaganych do dalszego podziału.
- W przeciwnym razie dokonuje podziału w optymalnym punkcie.

Funkcja ta ma następującą postać:

```
>>> def split(node, max_depth, min_size, depth, criterion):
...     """
...     Funkcja dzieląca węzeł lub przypisująca mu końcową etykietę
...     @param node: słownik z informacjami o węźle
...     @param max_depth: int, maksymalna głębokość drzewa
...     @param min_size: int, minimalna liczba próbek wymagana do podziału węzła
...     @param depth: int, głębokość aktualnego węzła
...     @param criterion: kryterium gini lub entropii
...     """
...     left, right = node['children']
...     del (node['children'])
...     if left[1].size == 0:
...         node['right'] = get_leaf(right[1])
...         return
...     if right[1].size == 0:
...         node['left'] = get_leaf(left[1])
...         return
...     # Sprawdzenie, czy aktualna głębokość nie przekracza maksymalnej
...     if depth >= max_depth:
...         node['left'], node['right'] =
...             get_leaf(left[1]), get_leaf(right[1])
...         return
...     # Sprawdzenie, czy lewy węzeł potomny zawiera wystarczającą liczbę próbek
...     if left[1].size <= min_size:
...         node['left'] = get_leaf(left[1])
...     else:
...         # Jeżeli tak, dzielimy go dalej
...         result = get_best_split(left[0], left[1], criterion)
...         result_left, result_right = result['children']
...         if result_left[1].size == 0:
...             node['left'] = get_leaf(result_right[1])
...         elif result_right[1].size == 0:
...             node['left'] = get_leaf(result_left[1])
...         else:
```

```

...         node['left'] = result
...         split(node['left'], max_depth, min_size, depth + 1, criterion)
...         # Sprawdzenie, czy prawy węzeł potomny zawiera wystarczającą liczbę próbek
...         if right[1].size <= min_size:
...             node['right'] = get_leaf(right[1])
...         else:
...             # Jeżeli tak, dzielimy go dalej
...             result = get_best_split(right[0], right[1], criterion)
...             result_left, result_right = result['children']
...             if result_left[1].size == 0:
...                 node['right'] = get_leaf(result_right[1])
...             elif result_right[1].size == 0:
...                 node['right'] = get_leaf(result_left[1])
...             else:
...                 node['right'] = result
...                 split(node['right'], max_depth, min_size, depth + 1, criterion)

```

Na koniec zdefiniuj funkcję inicjującą budowanie drzewa:

```

>>> def train_tree(X_train, y_train, max_depth, min_size, criterion='gini'):
...     """
...     Funkcja inicjująca budowanie drzewa
...     @param X_train: lista próbek treningowych (cechy)
...     @param y_train: lista próbek treningowych (ceł)
...     @param max_depth: int, maksymalna głębokość drzewa
...     @param min_size: int, minimalna liczba próbek wymagana do podziału węzła
...     @param criterion: kryterium gini lub entropii
...     """
...     X = np.array(X_train)
...     y = np.array(y_train)
...     root = get_best_split(X, y, criterion)
...     split(root, max_depth, min_size, 1, criterion)
...     return root

```

Przetestuj funkcję na przykładach, które wcześniej wyliczyłeś ręcznie:

```

>>> X_train = [['technika', 'specjalista'],
...             ['moda', 'student'],
...             ['moda', 'specjalista'],
...             ['sport', 'student'],
...             ['technika', 'student'],
...             ['technika', 'emeryt'],
...             ['sport', 'specjalista']]
>>> y_train = [1, 0, 0, 0, 1, 0, 1]
>>> tree = train_tree(X_train, y_train, 2, 2)

```

Napisz funkcję wyświetlającą drzewo, aby sprawdzić, czy wygenerowane przez model jest identyczne ze zbudowanym ręcznie:

```

>>> CONDITION = {'numerical': {'yes': '>=', 'no': '<'},
...               'categorical': {'yes': 'to', 'no': 'to nie'}}
>>> def visualize_tree(node, depth=0):
...     if isinstance(node, dict):

```

```

...         if node['value'].dtype.kind in ['i', 'f']:
...             condition = CONDITION['numerical']
...         else:
...             condition = CONDITION['categorical']
...         print('{}|- X{} {} {}'.format(depth * ' ',
...             node['index'] + 1, condition['no'], node['value']))
...         if 'left' in node:
...             visualize_tree(node['left'], depth + 1)
...         print('{}|- X{} {} {}'.format(depth * ' ',
...             node['index'] + 1, condition['yes'], node['value']))
...         if 'right' in node:
...             visualize_tree(node['right'], depth + 1)
...     else:
...         print(f"{depth * ' '}[{node}]")
>>> visualize_tree(tree)
|- X1 to nie moda
  |- X2 to nie specjalista
    [0]
  |- X2 to specjalista
    [1]
|- X1 to moda
  [0]

```

Funkcję możesz przetestować na przykładzie liczbowym, jak niżej:

```

>>> X_train_n = [[6, 7],
...               [2, 4],
...               [7, 2],
...               [3, 6],
...               [4, 7],
...               [5, 2],
...               [1, 6],
...               [2, 0],
...               [6, 3],
...               [4, 1]]
>>> y_train_n = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
>>> tree = train_tree(X_train_n, y_train_n, 2, 2)
>>> visualize_tree(tree)
|- X2 < 4
  |- X1 < 7
    [1]
  |- X1 >= 7
    [0]
|- X2 >= 4
  |- X1 < 2
    [1]
  |- X1 >= 2
    [0]

```

Drzewa decyzyjne wygenerowane przez model są takie same jak utworzone ręcznie.

Teraz, gdy po zaimplementowaniu drzewa decyzyjnego od podstaw masz o nim solidną wiedzę, możesz zająć się tworzeniem go za pomocą biblioteki *scikit-learn*.

Implementacja drzewa decyzyjnego za pomocą biblioteki scikit-learn

W tym podrozdziale wykorzystasz dobrze opracowany i zoptymalizowany moduł `DecisionTree` ↪ `Classifier`, przeznaczony do tworzenia drzew decyzyjnych (<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>):

```
>>> from sklearn.tree import DecisionTreeClassifier
>>> tree_sk = DecisionTreeClassifier(criterion='gini',
...     max_depth=2, min_samples_split=2)
>>> tree_sk.fit(X_train_n, y_train_n)
```

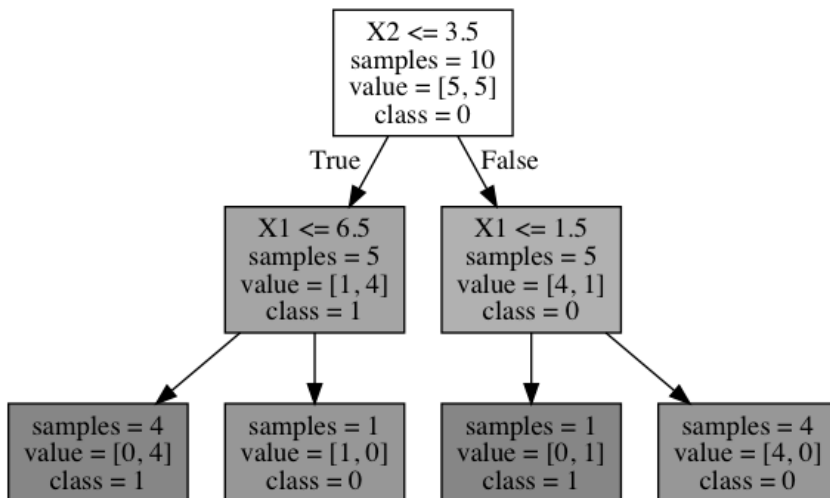
Do zwizualizowania utworzonego właśnie drzewa użyj funkcji `export_graphviz()`:

```
>>> export_graphviz(tree_sk, out_file='tree.dot',
...     feature_names=['X1', 'X2'], impurity=False,
...     filled=True, class_names=['0', '1'])
```

Po uruchomieniu kodu powstanie plik `tree.dot`, który możesz przekształcić w obraz PNG za pomocą programu Graphviz (opis i instrukcję instalacyjną znajdziesz na stronie <http://www.graphviz.org>). W tym celu użyj następującego polecenia:

```
dot -Tpng tree.dot -o tree.png
```

Rysunek 4.8 przedstawia uzyskany wynik.



Rysunek 4.8. Wizualizacja drzewa

Wygenerowane drzewo jest takie samo jak utworzone wcześniej ręcznie.

Wiem, że nie możesz się doczekać zastosowania drzewa decyzyjnego do prognozowania kliknięć reklam. Zajmiemy się tym w następnym podrozdziale.

Prognozowanie kliknięć reklam za pomocą drzewa decyzyjnego

Po przeanalizowaniu kilku przykładów nadszedł czas, aby zająć się prognozowaniem kliknięć reklam za pomocą algorytmu drzewa decyzyjnego, który właśnie dokładnie poznałeś i przećwicyłeś. Wykorzystasz zbiór danych z konkursu uczenia maszynowego Kaggle Click-Through Rate Prediction (prognozowanie współczynnika klikalności, <https://www.kaggle.com/c/avazu-ctr-prediction>), dostępnego na stronie <https://www.kaggle.com/c/avazu-ctr-prediction/data>.

Wystarczy pobrać i rozpakować plik *train.gz* (co trwa dłuższą chwilę), jedyny zawierający próbki z etykietami. W tym podrozdziale skupimy się tylko na pierwszych 300 000 próbek.

Plik zawiera pola przedstawione w tabeli 4.4.

Tabela 4.4. Opis zbioru danych i przykładowe wartości

Pole	Opis	Przykładowe wartości
Id	Identyfikator reklamy	1000009418151094273, 10000169349117863715
Click	0 – brak kliknięcia, 1 – kliknięcie	0, 1
Hour	Czas w formacie RMMDDGG	14102100
C1	Zanonimizowana zmienna kategoryczna	1005, 1002
banner_pos	Położenie banera	1, 0
site_id	Identyfikator strony	1fbe01fe, fe8cc448, d6137915
site_domain	Zakodowana domena strony	bb1ef334, f3845767
site_category	Zakodowana kategoria strony	28905ebd, 28905ebd
app_id	Identyfikator aplikacji mobilnej	ecad2386
app_domain	Domena aplikacji mobilnej	7801e8d9
app_category	Kategoria reklamy	07d7df22

Tabela 4.4. Opis zbioru danych i przykładowe wartości (ciąg dalszy)

Pole	Opis	Przykładowe wartości
device_id	Identyfikator urządzenia mobilnego	a99f214a
device_ip	Adres IP	ddd2926e
device_model	Zakodowany model urządzenia, na przykład iPhone 6, Samsung	44956a24
device_type	Zakodowany typ urządzenia, na przykład tablet, smartfon	1
device_conn_type	Zakodowany typ połączenia, na przykład Wi-Fi, 3G	0, 2
C14 – C21	Zanonimizowane zmienne kategoryjne	

Przejrzyj początkową zawartość pliku za pomocą poniższego polecenia, dzięki któremu wynik jest bardziej czytelny, ponieważ wartości są wyświetlane w wyrównanych kolumnach:

```
head train | sed 's/,/, /g;s/,/, /g' | column -s, -t
```

Rysunek 4.9 przedstawia początkową zawartość pliku.

id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id		
app_domain	app_category	device_id	device_ip	device_model	device_type	device_conn_type	C14	C15		
C16	C17	C18	C19	C20	C21					
1000009418151094273	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386		
7801e8d9	07d7df22	a99f214a	ddd2926e	44956a24	1	2		15706	320	
50	1722	0	35	-1	79					
10000169349117863715	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386		
7801e8d9	07d7df22	a99f214a	96809ac8	711ee120	1	0		15704	320	
50	1722	0	35	100084	79					
10000371904215119486	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386		
7801e8d9	07d7df22	a99f214a	b3cf8def	8a4875bd	1	0		15704	320	
50	1722	0	35	100084	79					
10000640724480838376	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386		
7801e8d9	07d7df22	a99f214a	e8275b8f	6332421a	1	0		15706	320	
50	1722	0	35	100084	79					
10000679056417042096	0	14102100	1005	1	fe8cc448	9166c161	0569f928	ecad2386		
7801e8d9	07d7df22	a99f214a	9644d0bf	779d90c2	1	0		18993	320	
50	2161	0	35	-1	157					
10000720757801103869	0	14102100	1005	0	d6137915	bb1ef334	f028772b	ecad2386		
7801e8d9	07d7df22	a99f214a	05241af0	8a4875bd	1	0		16920	320	
50	1899	0	431	100077	117					
10000724729988544911	0	14102100	1005	0	8fda644b	25d4cfcd	f028772b	ecad2386		
7801e8d9	07d7df22	a99f214a	b264c159	be6db1d7	1	0		20362	320	
50	2333	0	39	-1	157					
10000918755742328737	0	14102100	1005	1	e151e245	7e091613	f028772b	ecad2386		
7801e8d9	07d7df22	a99f214a	e6f67278	be74e6fe	1	0		20632	320	
50	2374	3	39	-1	23					
10000949271186029916	1	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386		
7801e8d9	07d7df22	a99f214a	37e8da74	5db079b5	1	2		15707	320	
50	1722	0	35	-1	79					

Rysunek 4.9. Kilka pierwszych wierszy pliku danych

Nie przejmuj się zanonimizowanymi, zakodowanymi wartościami. Są to cechy kategoryjne. Każda zakodowana wartość odpowiada pewnej rzeczywistej i sensownej wartości. W pliku jest zapisana w takiej formie ze względu na politykę prywatności. Na przykład C1 oznacza płeć użytkownika, a 1005 i 1002, odpowiednio, mężczyznę i kobietę.

Teraz odczytaj zbiór danych za pomocą modułu *pandas*, niezwykle przydatnego w przetwarzaniu danych tabelarycznych:

```
>>> import pandas as pd
>>> n_rows = 300000
>>> df = pd.read_csv("train.csv", nrows=n_rows)
```

Powyższy kod odczytuje z pliku pierwsze 300 000 wierszy i umieszcza je w obiekcie DataFrame. Przyjrzyjmy się pobieżnie kilku pierwszym wierszom:

```
>>> print(df.head(5))
```

	id	click	hour	C1	banner_pos	site_id	...	C16	C17	C18	C19	C20	C21
0	1.000009e+18	0	14102100	1005	0	1fbe01fe	...	50	1722	0	35	-1	79
1	1.000017e+19	0	14102100	1005	0	1fbe01fe	...	50	1722	0	35	100084	79
2	1.000037e+19	0	14102100	1005	0	1fbe01fe	...	50	1722	0	35	100084	79
3	1.000064e+19	0	14102100	1005	0	1fbe01fe	...	50	1722	0	35	100084	79
4	1.000068e+19	0	14102100	1005	1	fe8cc448	...	50	2161	0	35	-1	157

Zmienna docelowa znajduje się w kolumnie *click*:

```
>>> Y = df['click'].values
```

Oprócz tego istnieje kilka kolumn (cech), które należy usunąć, ponieważ nie zawierają przydatnych informacji (*id*, *hour*, *device_id* i *device_ip*):

```
>>> X = df.drop(['click', 'id', 'hour', 'device_id', 'device_ip'],
axis=1).values
>>> print(X.shape)
(300000, 19)
```

Każda próbka posiada 19 cech predykcyjnych.

Teraz podziel dane na zasoby treningowy i testowy. Zazwyczaj w tym celu próbki wybiera się losowo. Jednak w tym przypadku są one uszeregowane chronologicznie, według kolumny *hour*. Oczywiście nie można prognozować wyników z przeszłości, wykorzystując próbki z przyszłości. Dlatego do zbioru treningowego wybierz 90% pierwszych próbek, a pozostałe umieść w zbiorze testowym:

```
>>> n_train = int(n_rows * 0.9)
>>> X_train = X[:n_train]
>>> Y_train = Y[:n_train]
>>> X_test = X[n_train:]
>>> Y_test = Y[n_train:]
```

Jak wspomniałem wcześniej, modele drzew decyzyjnych mogą przetwarzać cechy kategoryjne. Jednak ponieważ algorytmy zaimplementowane w pakiecie *scikit-learn* (w wersji 1.0.1 z października 2021 r.) pozwalają tylko na wprowadzanie liczb, przekształć cechy kategoryczne w liczbowe.

Pamiętaj jednak, że zazwyczaj nie trzeba tego robić. Na przykład klasyfikator drzewa decyzyjnego, który opracowaliśmy wcześniej od podstaw, może bezpośrednio przyjmować cechy kategoryjne.

Teraz, za pomocą modułu `OneHotEncoder`, zawartego w pakiecie *scikit-learn*, przekształć tekstowe cechy kategoryjne w wektory, wykorzystując kodowanie „1 z n ”. Kodowanie „1 z n ” zostało krótko opisane w rozdziale 1., „Pierwsze kroki z uczeniem maszynowym w Pythonie”. Dla przypomnienia, polega ono na przekształcaniu k możliwych wartości cech kategoryjnych w k cech binarnych. Na przykład cechę `site_category`, przyjmującą trzy wartości, `news`, `education` i `sports`, można przekształcić w trzy elementy binarne, `is_news`, `is_education` i `is_sports`, z których każdy przyjmuje wartość 1 lub 0.

Zainicjuj obiekt `OneHotEncoder` w następujący sposób:

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> enc = OneHotEncoder(handle_unknown='ignore')
```

Następnie przetrenuj go:

```
>>> X_train_enc = enc.fit_transform(X_train)
>>> X_train_enc[0]
<1x8385 sparse matrix of type '<class 'numpy.float64'>'
with 19 stored elements in Compressed Sparse Row format>
>>> print(X_train_enc[0])
(0, 2) 1.0
(0, 6) 1.0
(0, 30) 1.0
(0, 1471) 1.0
(0, 2743) 1.0
(0, 3878) 1.0
(0, 4000) 1.0
(0, 4048) 1.0
(0, 6663) 1.0
(0, 7491) 1.0
(0, 7494) 1.0
(0, 7861) 1.0
(0, 8004) 1.
(0, 8008) 1.0
(0, 8085) 1.0
(0, 8158) 1.0
(0, 8163) 1.0
(0, 8202) 1.0
(0, 8383) 1.0
```

Każda przekształcona próbka jest rozrzedzonym wektorem.

Do przekształcenia zbioru testowego użyj przetrenowanego dekodera „1 z n ”:

```
>>> X_test_enc = enc.transform(X_test)
```

Pamiętaj, że obiekt dekodera „1 z n ” zainicjowałeś argumentem `handle_unknown='ignore'`, aby zapobiec błędom wynikającym z nieznanych wartości cechy kategoryjnej. Jeżeli na przykład pojawi się próbka, której cecha `site_category` będzie miała wartość `movie` (film), wszystkie

trzy przekształcone cechy binarne (`is_news`, `is_education` i `is_sports`) uzyskają wartość 0. Gdyby nie został użyty powyższy argument, pojawiłby się błąd.

Teraz przetrenuj model drzewa decyzyjnego, wykorzystując przeszukiwanie siatki, opisane w rozdziale 3., „Rozpoznawanie twarzy przy użyciu maszyny wektorów nośnych”. W celach demonstracyjnych dostosuj tylko hiperparametr `max_depth`, choć tak naprawdę należałoby uwzględnić również inne hiperparametry, na przykład `min_samples_split` i `class_weight`. Wskaźnikiem klasyfikacyjnym niech będzie pole pod krzywą lub krzywa ROC, ponieważ zbiór jest nie zrównoważony. Tylko 51 211 spośród 300 000 próbek zawiera informacje o kliknięciach, czyli współczynnik klikalności jest równy 17% (zachęcam Cię do samodzielnego sprawdzenia rozkładu klas). Określ trzy maksymalne wartości głębokości drzewa: 3, 10 i nieograniczoną:

```
>>> from sklearn.tree import DecisionTreeClassifier
>>> parameters = {'max_depth': [3, 10, None]}
```

Zainicjuj model, wykorzystując jako kryterium zanieczyszczenie Giniego oraz minimalną liczbę próbek wymaganych do podziału węzła równą 30:

```
>>> decision_tree = DecisionTreeClassifier(criterion='gini',
min_samples_split=30)
>>> from sklearn.model_selection import GridSearchCV
```

Do przeszukiwania siatki zastosuj potrójną weryfikację krzyżową (ponieważ jest wystarczająca liczba próbek regularnych), a najlepszy hiperparametr wybierz na podstawie pola pod krzywą. Zwróć uwagę na argument `n_jobs=-1`, oznaczający wykorzystanie wszystkich dostępnych procesorów:

```
>>> grid_search = GridSearchCV(decision_tree, parameters,
...                             n_jobs=-1, cv=3, scoring='roc_auc')
>>> grid_search.fit(X_train, y_train)
>>> print(grid_search.best_params_)
{'max_depth': 10}
```

Użyj modelu z optymalnym hiperparametrem do prognozowania przyszłych próbek:

```
>>> decision_tree_best = grid_search.best_estimator_
>>> pos_prob = decision_tree_best.predict_proba(X_test)[: , 1]
>>> from sklearn.metrics import roc_auc_score
>>> print(f'Pole pod krzywą ROC dla zbioru testowego: {roc_auc_score(Y_test,
...                             pos_prob):.3f}')
Pole pod krzywą ROC dla zbioru testowego: 0.719
```

Pole pod krzywą dla optymalnego modelu drzewa decyzyjnego jest równe 0,72. To nie jest dużo, ale klikalność reklamy zależy od wielu skomplikowanych czynników ludzkich, dlatego jej prognozowanie nie jest łatwym zadaniem. Możesz dalej optymalizować hiperparametr, ale uzyskany wynik jest całkiem dobry. Gdyby losowo prognozować wyniki, 17% kliknięć dałoby pole pod krzywą równe 0,496:

```
>>> pos_prob = np.zeros(len(Y_test))
>>> click_index = np.random.choice(len(Y_test),
```

```

...             int(len(Y_test) * 51211.0/300000),
...             replace=False)
>>> pos_prob[click_index] = 1
>>> print(f'Pole pod krzywą ROC dla zbioru testowego: {roc_auc_score(Y_test,
...             pos_prob):.3f}')
Pole pod krzywą ROC dla zbioru testowego: 0.496

```

Jak widać, algorytm drzewa decyzyjnego to sekwencja zachłannych poszukiwań najlepszego punktu podziału na każdym kroku na podstawie zbioru treningowego. Skutkuje to jednak tendencją modelu do nadmiernego dopasowania, ponieważ optymalne punkty podziału dotyczą głównie próbek treningowych. Na szczęście istnieje technika korygująca ten efekt: **gromadzenie** (ang. *ensembling*), a las losowy jest modelem gromady drzew, zazwyczaj skuteczniejszym od zwykłego drzewa decyzyjnego.

Gromadzenie drzew decyzyjnych: las losowy

Technika gromadzenia, czyli **agregacja bootstrap** (ang. *bootstrap aggregating* lub *bagging*), o której krótko wspomniałem w rozdziale 1., „Pierwsze kroki z uczeniem maszynowym w Pythonie”, skutecznie zapobiega nadmiernemu dopasowaniu modelu. Dla przypomnienia, polega na wielokrotnym, losowym wybieraniu próbek ze zbioru treningowego ze zwracaniem. Każdy otrzymany w ten sposób podzbiór jest wykorzystywany do trenowania modelu. Wyniki uzyskane za pomocą osobno przetrenowanych modeli są zbierane i w procesie głosowania podejmowana jest ostateczna decyzja.

Agregacja bootstrap, jak wspomniałem w poprzednim akapicie, zmniejsza wariancję modelu drzewa decyzyjnego. Dzięki niej uzyskane wyniki są zazwyczaj lepsze niż wygenerowane przez pojedyncze drzewo. Jednak w niektórych przypadkach, gdy jedna lub kilka cech jest silnych, poszczególne drzewa są konstruowane w dużej mierze w oparciu o te cechy i w rezultacie są silnie skorelowane. Agregacja wielu skorelowanych drzew nie robi większej różnicy. Aby drzewa nie były skorelowane, w algorytmie lasu losowego podczas poszukiwania najlepszego punktu podziału każdego węzła uwzględniane są wyłącznie losowe podzbiory cech. Poszczególne drzewa są następnie trenowane na różnych, sekwencyjnych zbiorach, co zapewnia większą różnorodność i skuteczność modelu. Las losowy jest odmianą modelu agregacji bootstrap z dodatkowym agregowaniem cech.

Aby zastosować las losowy w projekcie prognozowania kliknięć reklam, użyj pakietu *scikit-learn*. Podobnie jak podczas implementowania drzewa decyzyjnego w poprzednim podrozdziale, modyfikuj tylko hiperparametr `max_depth`:

```

>>> from sklearn.ensemble import RandomForestClassifier
>>> random_forest = RandomForestClassifier(n_estimators=100,
...             criterion='gini', min_samples_split=30,
...             n_jobs=-1)

```

Oprócz hiperparametrów `max_depth`, `min_samples_split` i `class_weight`, które są ważne w pojedynczym drzewie decyzyjnym, bardzo zalecane jest modyfikowanie hiperparametrów związanych z losowym lasem (zbiorem drzew), na przykład `n_estimators`. Hiperparametr `max_depth` możesz dostroić w następujący sposób:

```
>>> grid_search = GridSearchCV(random_forest, parameters,
...                             n_jobs=-1, cv=3, scoring='roc_auc')
>>> grid_search.fit(X_train, y_train)
>>> print(grid_search.best_params_)
{'max_depth': None}
```

Do prognozowania przyszłych, nieznanych przypadków użyj modelu z hiperparametrem `max_depth` równym `None`, oznaczającym, że węzły będą dzielone do chwili, aż zostanie spełnione inne kryterium zakończenia procesu.

```
>>> random_forest_best = grid_search.best_estimator_
>>> pos_prob = random_forest_best.predict_proba(X_test)[: , 1]
>>> print('Pole pod krzywą ROC dla zbioru testowego:
...       {0:.3f}'.format(roc_auc_score(y_test, pos_prob)))
Pole pod krzywą ROC dla zbioru testowego: 0.759
```

Jak widać, skuteczność lasu losowego jest zdecydowanie większa.

Podsumujmy najważniejsze hiperparametry, które należy dostrajać:

- `max_depth`: maksymalna głębokość pojedynczego drzewa. Jeżeli jest zbyt duży, model ma tendencję do nadmiernego dopasowania, a w przeciwnym razie do niedopasowania.
- `min_samples_split`: minimalna liczba próbek wymaganych do dalszego podziału węzła. Zbyt mała wartość skutkuje tendencją modelu do nadmiernego dopasowania, a zbyt duża do niedopasowania. Dobrze jest zacząć od wartości 10, 30 lub 50.

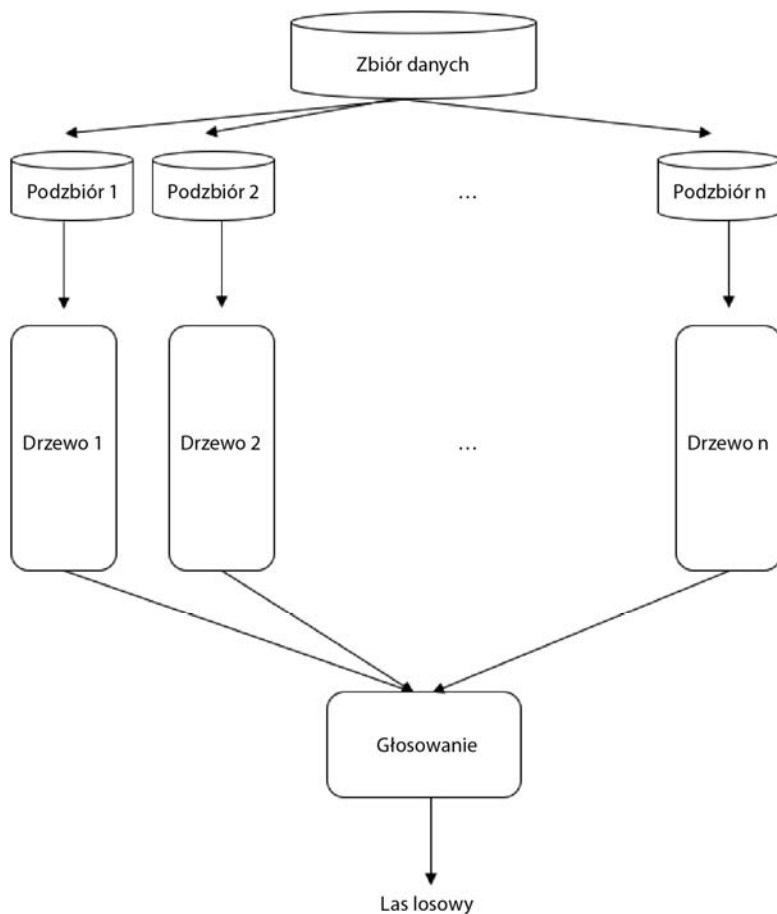
Opisane dwa hiperparametry są ogólnie związane z tworzeniem pojedynczych drzew decyzyjnych, natomiast poniższe dotyczą lasu losowego, czyli zbioru drzew:

- `max_features`: liczba cech uwzględnianych podczas wyszukiwania najlepszego punktu podziału. Zazwyczaj dla m -wymiarowego zbioru danych zalecaną wartością tego hiperparametru jest \sqrt{m} (po zaokrągleniu). W pakiecie *scikit-learn* osiąga się to za pomocą hiperparametru `max_features="sqrt"`. Inne wartości to `log2`, 20% lub 50% oryginalnej liczby cech.
- `n_estimators`: liczba drzew uwzględnianych podczas głosowania. Ogólnie, im więcej drzew, tym większa skuteczność modelu, ale dłuższy czas obliczeń. Zazwyczaj stosuje się wartości 100, 200 i 500 tego hiperparametru.

Teraz zajmijmy się drzewami decyzyjnymi ze wzmocnieniem gradientowym.

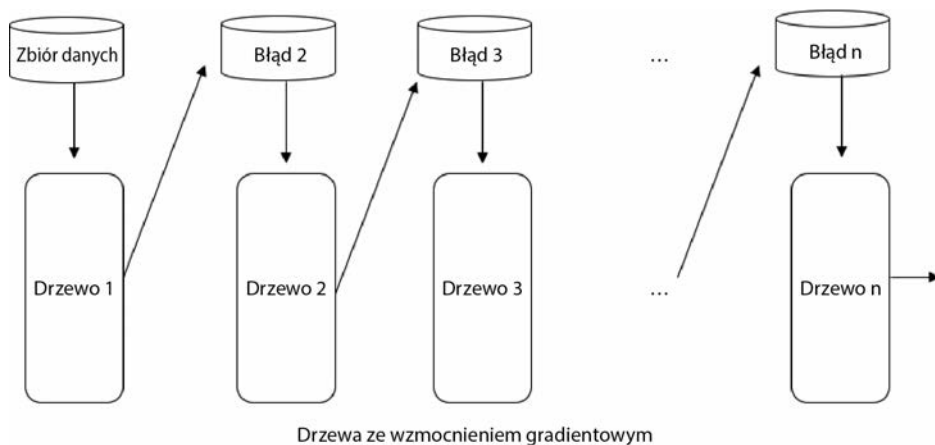
Gromadzenie drzew decyzyjnych: drzewa ze wzmocnieniem gradientowym

Inna technika zespołowa, **wzmacnianie** (ang. *boosting*), polega na iteracyjnym, a nie równoległym, łączeniu wielu modeli. Poszczególne drzewa nie są trenowane osobno. W szczególności w modelu **drzew ze wzmocnieniem gradientowym** (ang. *gradient boosted tree*, GBT), zwanym również modelem **maszyn ze wzmocnieniem gradientowym**, poszczególne drzewa są trenowane sukcesywnie, a zadaniem kolejnego jest poprawianie błędów poprzedniego. Różnice pomiędzy lasem losowym a drzewami ze wzmocnieniem gradientowym są pokazane na rysunkach 4.10 i 4.11. W lesie losowym każde drzewo jest budowane niezależnie od innych, przy użyciu innego podzbioru danych, po czym wyniki są uśredniane lub wybierany jest jeden z nich w drodze głosowania (rysunek 4.10).



Rysunek 4.10. Działanie lasu losowego

W modelu drzew ze wzmocnieniem gradientowym za każdym razem jest budowane jedno drzewo, a wyniki są łączone na bieżąco (rysunek 4.11).



Rysunek 4.11. Działanie modelu drzew ze wzmocnieniem gradientowym

Do zaimplementowania drzew ze wzmocnieniem gradientowym użyj pakietu XGBoost (<https://xgboost.readthedocs.io/en/latest>). Najpierw zainstaluj go za pomocą następującego polecenia:

```
pip install xgboost
```

Jeżeli pojawi się problem, zainstaluj lub zaktualizuj pakiet CMake:

```
pip install CMake
```

Aby prognozować kliknięcia reklam za pomocą drzewa ze wzmocnieniem gradientowym, wykonaj poniższe kroki:

1. Najpierw przekształć etykietę w zmienną dwuwymiarową, tj. wartość 0 w wektor [1; 0], a 1 w [0; 1]:

```
>>> from sklearn.preprocessing import LabelEncoder
>>> le = LabelEncoder()
>>> Y_train_enc = le.fit_transform(Y_train)
```

2. Zimportuj moduł XGBoost i zainicjuj model. Szybkość uczenia określającą, jak sprawnie model (każde drzewo) będzie się uczyć w każdym kroku, ustaw na 0,1. Hiperparametr ten będzie dokładniej omówiony w rozdziale 5., „Prognozowanie kliknięć reklam internetowych przy użyciu regresji logistycznej”. Hiperparametrowi `max_depth` dla każdego drzewa przypisz wartość 10. Dodatkowo niech będzie trenowana sekwencja 1000 drzew:

```
>>> import xgboost as xgb
>>> model = xgb.XGBClassifier(learning_rate=0.1, max_depth=10,
                             n_estimators=1000)
```


3. Przetrenuj model na przygotowanym wcześniej zbiorze treningowym:

```
>>> model.fit(X_train_enc, Y_train)
```

4. Użyj przetrenowanego modelu do prognozowania kliknięć. Dodatkowo wylicz pole pod krzywą ROC:

```
>>> pos_prob = model.predict_proba(X_test_enc)[: , 1]
>>> print(f'Pole pod krzywą ROC dla zbioru testowego: {roc_auc_score(Y_test,
pos_prob):.3f}')
Pole pod krzywą ROC dla zbioru testowego: 0.771
```

Pole pod krzywą dla modelu drzew ze wzmocnieniem gradientowym jest równe 0,77.

W tym podrozdziale poznałeś inny model gromady drzew, tj. ze wzmocnieniem gradientowym, i zastosowałeś go do prognozowania kliknięć reklam.

Podsumowanie

Ten rozdział rozpoczął się od wprowadzenia do typowego problemu uczenia maszynowego, tj. prognozowania kliknięć reklam internetowych i związanych z nim wyzwań, m.in. cech kategoryalnych. Następnie zostały opisane algorytmy drzew decyzyjnych, operujących zarówno na cechach liczbowych, jak i kategoryalnych.

W dalszej części zostały szczegółowo omówione różne algorytmy, ich zasady działania, sposoby konstruowania drzew oraz wskaźniki określające jakość podziału węzłów (zanieczyszczenia Giniego i entropia). Po ręcznym zbudowaniu drzewa zaimplementowałeś algorytm CART od podstaw.

Dowiedziałeś się, jak zbudować model drzewa decyzyjnego za pomocą pakietu *scikit-learn* i zastosowałeś go do prognozowania kliknięć reklam. Zwiększyłeś jego skuteczność, implementując algorytm lasu losowego opartego na cechach. Na koniec poznałeś różne sposoby strojenia modelu lasu losowego. W dodatkowym podrozdziale została opisana implementacja modelu drzew ze wzmocnieniem gradientowym za pomocą pakietu XGBoost. Agregacja bootstrap i wzmocnienie gradientowe to dwie techniki gromadzenia drzew umożliwiające osiągnięcie większej skuteczności modelu.

Najlepszym sposobem udoskonalania swoich umiejętności jest praktyka. Zachęcam Cię do wykonania opisanych niżej ćwiczeń, zanim przejdziesz do następnego rozdziału, w którym rozwiążemy problem prognozowania kliknięć za pomocą innego algorytmu: regresji logistycznej.

Ćwiczenia

1. Czy w modelu drzewa decyzyjnego, prognozującym kliknięcia reklam, można również dostrajać inne hiperparametry, na przykład `min_samples_split` i `class_weight`? Jak można wtedy uzyskać największe pole pod krzywą ROC?
2. Czy w modelu lasu losowego, prognozującym kliknięcia reklam, można również dostrajać inne hiperparametry, na przykład `min_samples_split`, `max_features` i `n_estimators`? Jak można wtedy uzyskać największe pole pod krzywą ROC?
3. Jakie hiperparametry można dostrajać w modelu drzew ze wzmocnieniem gradientowym? Jak można wtedy uzyskać największe pole pod krzywą ROC? Możesz się tego dowiedzieć na stronie https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn.

Skorowidz

A

- agent, 391
- agregacja bootstrap, 46, 135
- akcja, 391
- algorytmy
 - iteracji
 - polityki, 400
 - wartości, 397
- Q-uczenia, 411
 - implementacja, 414
- regresji, 194, 222
- uczenia
 - maszynowego, 27
 - przez wzmacnianie, 393
- alokacja Dirichleta, 300
- analiza
 - głównych składowych, 82, 105
 - języka, 383
 - tekstu, 250
- Apache
 - Hadoop YARN, 174
 - Mesos, 174
 - Spark, 170
 - instalacja, 172
 - komponenty, 170
 - trenowanie modelu, 177
- architektura modelu Transformer, 384
- automatyka, 22

B

- badanie danych, 264
 - zbiór zdjęć odzieży, 338
 - zbiór zdjęć twarzy, 101
- biblioteka
 - Gensim, 255, 261
 - NLTK, 254, 255
 - pandas, 107
 - PyTorch, 388
 - spaCy, 254
 - TextBlob, 255
- binaryzacja, 316
- Blackjack, 403
- błąd
 - bezwzględny, 220
 - średniokwadratowy, 220, 231

C

- cechy, 56
 - binarne, 181
 - ilościowe, 111
 - kategorialne, 111, 142, 186
 - liczbowe, 142
 - porządkowe, 111
- centroid, 285, 290
- charakterystyka operacyjna odbiornika, 76
- ciągłe wartości docelowe, 196
- części mowy, 258

D

dane
 kategorialne, 111
 treningowe, 38, 56
 detektor interakcji, 115
 dobór wartości k , 291
 dobre praktyki
 przygotowywania danych, 307
 trenowania, 322
 wdrażania, 328
 dopasowanie modelu
 nadmierne, 30, 37, 39
 niedostateczne, 32
 R^2 , 220
 dropout, 240
 drzewa decyzyjne, 112, 324
 budowanie, 113
 gromadzenie, 135, 137
 implementacja, 122, 129, 214
 prognozowanie kliknięć, 130
 regresyjne, 115, 212, 214
 drzewo klasyfikacyjne, 115, 212
 dyskretyzacja, 317

E

eksploracja danych, 41
 entropia, 119
 etykieta, 56

F

filtr konwolucyjny, 347
 formuła oczekiwania Bellmana, 400
 FrozenLake, 394
 funkcja
 jądra, 96
 kosztu, 148
 logistyczna, 145
 funkcje aktywacji, 229, 239
 liniowa, 239
 ReLU, 231, 239
 sigmoidalna, 239
 softmax, 239
 tangens hiperboliczny, 239

G

generator tekstu, 376, 378
 trenowanie, 380
 generowanie cech, 202
 giełda, 195
 analiza
 fundamentalna, 195
 techniczna, 195
 indeksy giełdowe, 198
 pozyskiwanie
 cen akcji, 197
 danych, 202
 prognozowanie cen akcji, 222
 głębokość drzewa, 116
 głosowanie, 46
 gradient, 53
 prosty, 150, 154
 stochastyczny, 156
 grafy obliczeniowe, 53
 granice decyzyjne, 99
 GraphX, 171
 gromadzenie, 325
 GRU, Gated Recurrent Units, 365
 grupowanie danych, 45
 grupy dyskusyjne, 261, 264

H

HDFS, Hadoop Distributed File System, 177
 hiperpłaszczyzna, 83
 decyzyjna, 219
 dodatnia, 85
 liniowa, 95
 nieoptymalna, 85
 nierozdzielająca, 84
 optymalna, 85
 rozdzielająca, 84
 ujemna, 85

I

idealna krzywa uczenia, 326
 implementacja
 algorytmu Q-uczenia, 414
 drzewa decyzyjnego, 122, 129
 klastrowania, 281, 289
 lasu regresyjnego, 218
 maszyny wektorów nośnych, 88
 naiwnego klasyfikatora Bayesa, 65

- regresji
 - liniowej, 207–211
 - logistycznej, 165
 - wektorów nośnych, 219
- regresyjnego drzewa decyzyjnego, 214
- sieci neuronowej, 234, 237
- imputowanie, 41
- instalacja
 - biblioteki PyTorch, 388
 - narzędzi OpenAI Gym, 390
- interakcja, 44, 186, 317
 - cech, 189
- inżynieria cech, 40, 44, 186, 199
- iteracja
 - polityki, 400
 - wartości, 397

J

- jakość
 - klasyfikacji, 74
 - regresji, 220
- jądro, 94
 - gaussowskie, 96
 - liniowe, 98, 101
 - sigmoidalne, 98
 - wielomianowe, 98
- język naturalny, 251

K

- klasa, 56, 90
 - CountVectorizer, 268
 - LatentDirichletAllocation, 301
- klaster, 174
- klastrowanie, 277, 279
 - danych, 293
 - grup dyskusyjnych, 280
 - implementacja, 281, 289
 - metodą k-średnich, 284–288
- klasyfikacja, 56
 - binarna, 56
 - stanu płodu, 106
 - wieloetykietowa, 59
 - wieloklasowa, 57, 163
- klasyfikator
 - Bayesa, 69
 - obrazów, 103, 105, 354
- klasyfikowanie zdjęć odzieży, 342

- kodowanie
 - „1 z n”, 42, 186
 - etykiet, 42
 - porządkowe, 142
- kojarzenie, 279
- kolizja mieszania, 187
- konwolucja, 334
- korpusy, 255
- korzeń, 112
- krzywa
 - ROC, 77
 - uczenia nadmierne dopasowanego modelu, 326
 - uczenia niedopasowanego modelu, 327
- Kubernetes, 174

L

- las
 - losowy, 135, 167, 324
 - regresyjny
 - implementacja, 218
- lematyzacja, 260
- liść, 112
- LSTM, Long Short-Term Memory, 365
 - budowanie sieci, 370
 - poprawa skuteczności, 372
 - struktura komórki, 366

Ł

- łączenie
 - modeli, 45
 - zmiennych, 189

M

- macierz
 - faktoryzacji, 297
 - pomylek, 74
 - rozrzedzona, 43
- mapa cech, 334
- margines, 85
- maszyna wektorów nośnych, 82, 99, 101, 103, 324
 - implementacja, 88
- mechanizm rekurencyjny, 359

metoda
 k-średnich, 280, 285–289, 293
 Monte Carlo, 403, 405
 z polityką, 407
 wstrzymywania, 36
 mieszanie cech, 186
 kategoryalnych, 186
 MLLib, 171
 MNIST, 57
 model
 tematyczny, 296
 Transformer, 383, 384
 modelowanie
 sekwencyjne, 358
 semantyczne, 261
 tematyczne, 261, 277, 297, 300
 moduł
 BernoulliNB, 69
 PySpark, 174
 monitorowanie modelu, 328

N

nadmierne dopasowanie, 240
 nagroda, 391
 naiwny klasyfikator Bayesa, 55, 59, 323
 implementacja, 65
 narzędzia OpenAI Gym, 390
 niedostateczne dopasowania, 325
 nieujemna faktoryzacja macierzy, 297

O

obciążenie, 31, 33, 146
 obserwacje, 196
 ocena f1, 75
 odsetek prawdziwie pozytywny, 75
 OpenAI Gym, 390
 osadzanie słów, 261, 319

P

pakiet
 NumPy, 51
 Pandas, 52
 scikit-learn, 52, 69, 210, 237, 289
 SciPy, 52
 TensorFlow, 2, 52, 165, 211, 237
 panel TensorBoard, 246
 pierwiastek błędu średniokwadratowego, 220

plik Excela, 107
 polecenie pip install xldr, 107
 polityka
 deterministyczna, 393
 stochastyczna, 393
 prawdopodobieństwo
 a posteriori, 63
 a priori, 62
 precyzja, 74
 problem
 FrozenLake, 394
 taksówkarza, 411
 prognozowanie, 212, 218
 cen akcji, 194, 222, 227, 242
 kliknięć reklam, 109, 130, 141
 sekwencji danych, 357
 programowanie, 23
 dynamiczne, 394
 propagacja wstecz, 231
 w czasie, 365
 przebieg
 agregacji bootstrap, 47
 wzmacniania, 48
 przechwycenie, 146
 przekształcanie cech kategoryalnych, 142
 przetwarzanie
 języka naturalnego, 233, 251–254
 tekstu, 270
 przygotowywanie danych
 dobre praktyki, 307
 przyrost informacji, 119
 punkty odstające, 88
 Python
 instalacja, 49
 konfiguracja, 49

R

radialna funkcja bazowa, 96, 98, 101
 RDD, Resilient Distributed Datasets, 174
 redukcja wymiarowości, 39, 272, 273
 regresja, 196
 liniowa, 197, 206
 implementacja, 207, 210, 211
 logistyczna, 144, 146, 323
 implementacja, 165
 softmax, 163
 testy modelu, 184
 trening modelu, 149
 z gradientem prostym, 150, 154

- z regularyzacją, 158
- ze stochastycznym gradientem prostym, 156
- ocena jakości, 220
- softmax, 239
- wektorów nośnych, 218
 - implementacja, 219
- regularyzacja, 37, 325
 - L1, 159
 - L2, 158
- rekurencyjne sieci neuronowe, 357
- ReLU, Rectified Linear Unit, 229, 239
- rozpoznawanie twarzy, 82
- rozproszone przetwarzanie danych, 261
- rozrzut, 75
- rzutowanie, 279

S

- samouwaga, 383
- sekwencja, 358
- selekcja cech, 39, 167
- sieć neuronowa, 227, 325
 - analiza recenzji, 367
 - dostrojenie parametrów, 243
 - dropout, 240
 - funkcje aktywacji, 239
 - głęboka, 233
 - implementacja, 234, 237
 - konwolucyjna, 332, 335
 - bloki konstrukcyjne, 333
 - budowanie, 337
 - klasyfikowanie zdjęć, 342
 - obracanie obrazów, 351
 - przesuwanie obrazów, 352
 - trening sieci, 345
 - tworzenie, 342
 - uzupełnianie danych, 354
 - wzmocnianie, 349
 - nadmierne dopasowanie, 240
 - plytka, 228
 - prognozowanie cen akcji, 242
 - rekurencyjna, 357
 - architektura, 359
 - GRU, 365
 - jedno do wielu, 362
 - LSTM, 365
 - pisanie tekstu, 374
 - trening, 364, 374
 - wiele do jednego, 361
 - wiele do wielu, 362, 363

- sztuczna, 228
- trening, 242
- tworzenie, 234
- warstwy, 228, 232
- zakończenie treningu, 241
- sigmoida, 145, 229, 239
- skalowanie, 43
 - modelu, 169
 - w pionie, 311
 - w poziomie, 312
- składowanie, 49
- Spark
 - Core, 170
 - SQL, 170
 - Streaming, 171
- stan, 391
- Standalone, 174
- stemming, 260
- stochastyczny gradient prosty, 156
- strojenie modeli, 78
- strumieniowanie, 105
- sumaryczna nagroda, 392
- system rekomendacyjny, 69
- szybkość uczenia, 150

Ś

- średni błąd bezwzględny, 220
- środowisko, 391
 - Blackjack, 403
 - FrozenLake, 394

T

- tangens hiperboliczny, 229, 239
- technika
 - dropout, 240
 - t-SNE, 272, 273
- testy modelu, 184
- TF-IDF, 318
- token, 267
- tokenizacja, 257
- transformacja
 - Boxa-Coxa, 44
 - wielomianowa, 44, 317
 - wykładnicza, 44
- trenowanie
 - dobrze praktyki, 322
 - generatora tekstu, 380
 - modelu, 177, 184
 - sieci neuronowej, 242
 - rekurencyjnej, 364, 376

t-SNE, 272, 273
 twierdzenie Bayesa, 60
 tworzenie
 drzewa decyzyjnego, 113
 generatora tekstu, 378
 klasyfikatora obrazów, 103
 mapy cech, 334
 sieci
 konwolucyjnej, 337
 LSTM, 370
 neuronowej, 234
 środowiska
 Blackjack, 403
 FrozenLake, 394
 zbioru treningowego, 312

U

uczenie
 głębokie, 232
 maszynowe, 20, 28
 problemy, 306
 przygotowywanie danych, 307
 nadzorowane, 26, 56
 nienadzorowane, 26, 278
 online, 160
 przez wzmacnianie, 26, 387
 agent, 391
 akcja, 391
 algorytmy, 393
 metoda Monte Carlo, 403
 nagroda, 391, 392
 stan, 391
 środowisko, 391
 sekwencyjne, 358
 ukryta alokacja Dirichleta, 300
 uogólnianie danych, 29
 uproszczenie, 325
 usuwanie stop-słów, 270
 uśrednianie, 46

W

wariancja, 31, 33
 wariant TF-IDF, 111
 warstwa
 konwolucyjna, 333
 nieliniowa, 335
 redukująca, 335
 samouwagi, 383

wczesne
 zakończenie treningu, 241
 zatrzymanie, 38
 wdrażanie
 dobre praktyki, 328
 wektor
 nośny, 83, 85
 wag, 146
 wektoryzacja, 261
 weryfikacja
 krzyżowa, 34, 78, 325
 k-krotna, 78
 wewnętrzna, 36
 zewnętrzna, 36
 LOOCV, 35
 węzeł, 112, 116
 wielkość kroku, 150
 wielomian wyższego rzędu, 37
 wielomianowa regresja logistyczna, 163
 wizualizacja danych tekstowych, 272
 worek słów, 267
 wskaźnik
 jakości, 116
 skorygowany R^2 , 220
 współczynnik
 jądra, 96
 klikalności, 110
 wstępne przetwarzanie danych, 40
 wygładzanie Laplace'a, 64
 wykres zanieczyszczenia Giniego, 117
 wyszukiwanie według podobieństwa, 261
 wzmacnianie, 46, 137
 wzmocnienie gradientowe, 137

Z

zanieczyszczenie Giniego, 116
 zastosowania uczenia maszynowego, 23
 zbiór
 MNIST, 58
 testowy, 154, 180
 treningowy, 154, 180, 312
 zliczanie tokenów, 267
 zmienne predykcyjne, 56

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion

Wypróbuj najlepsze praktyki uczenia maszynowego z Pythonem!

Systemy oparte na uczeniu maszynowym są coraz bardziej wyrafinowane. Spośród wielu narzędzi służących do implementacji algorytmów uczenia maszynowego najpopularniejszy okazał się Python wraz z jego bibliotekami. Znajomość tych narzędzi umożliwia sprawne tworzenie systemów uczących się, jednak uzyskanie spektakularnych wyników wymaga doświadczenia i wprawy. Konieczne są więc ćwiczenia i praktyka w samodzielnym rozwiązywaniu problemów.

To trzecie wydanie popularnego podręcznika, który ułatwi Ci zdobycie praktycznej wiedzy o uczeniu maszynowym w Pythonie. Zapoznasz się z różnymi technikami implementacji algorytmów uczenia maszynowego. Przeanalizujesz rzeczywiste przykłady techniki eksploracyjnej analizy danych, inżynierii cech, klasyfikacji danych, regresji, klastrowania i przetwarzania języka naturalnego. To wydanie uzupełniono o najnowsze zagadnienia ważne dla biznesu, takie jak tworzenie systemu rekomendacji, rozpoznawanie twarzy, prognozowanie cen akcji, klasyfikowanie zdjęć, prognozowanie sekwencji danych i zastosowanie uczenia przez wzmacnianie w podejmowaniu decyzji. Dzięki książce poznasz omawiane zagadnienia od strony praktycznej i zdobędziesz wiedzę potrzebną do skutecznego rozwiązywania problemów z systemami uczącymi się.

W książce między innymi:

- gruntowne podstawy uczenia maszynowego i nauki o danych
- techniki eksploracji i analizy danych za pomocą kodu Pythona
- trenowanie modeli za pomocą Apache Spark
- przetwarzanie języka naturalnego przy użyciu bibliotek Pythona
- praktyczne wdrażanie modeli i algorytmów uczenia maszynowego
- korzystanie z bibliotek Pythona: TensorFlow 2, PyTorch i scikit-learn

Yuxi (Hayden) Liu rozwija modele uczenia maszynowego w Google. Wcześniej pracował naukowo nad zastosowaniami uczenia maszynowego w takich dziedzinach jak reklama internetowa i cyberbezpieczeństwo. Jest entuzjastą edukacji i autorem wielu książek o uczeniu maszynowym. Pierwsze wydanie tego podręcznika zajmowało czołową pozycję w rankingu Amazona w latach 2017 i 2018.

Helion 	Sprawdź nasze szkolenia!	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	SZKOLENIA 	ISBN 978-83-283-8870-3	
 0 801 339900	AKADEMIA IT & BUSINESS		
 0 601 339900	WWW.SZKOLENIA.HELION.PL	9 788328 388703	
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 99,00 zł	

Packt