

Wydanie II

50 algorytmów, które powinien znać każdy programista

Klasyczne i nowoczesne algorytmy
z dziedzin uczenia maszynowego,
projektowania oprogramowania,
systemów danych i kryptografii

Imran Ahmad



Helion 

<packt>

Tytuł oryginału: 50 Algorithms Every Programmer Should Know: Tackle computer science challenges with classic to modern algorithms in machine learning, software design, data systems, and cryptography, 2nd Edition

Tłumaczenie: Łukasz Piwko z wykorzystaniem fragmentów książki „40 algorytmów, które powinien znać każdy programista. Nauka implementacji algorytmów w Pythonie” w przekładzie Katarzyny Bogusławskiej

ISBN: 978-83-289-1107-9

Copyright © Packt Publishing 2023. First published in the English language under the title '50 Algorithms Every Programmer Should Know - Second Edition - (9781803247762)'.

Polish edition copyright © 2024 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.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/50alg2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<https://ftp.helion.pl/przyklady/50alg2.zip>

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

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

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści |

O autorze	15
O recenzentach	15
Przedmowa	17
Wprowadzenie	19

CZĘŚĆ 1. Wstęp i podstawowe algorytmy

ROZDZIAŁ 1

Wprowadzenie do algorytmów	25
Co to jest algorytm?	26
Fazy algorytmu	26
Środowisko programowania	28
Pakiety w Pythonie	28
Ekosystem SciPy	29
Jupyter Notebook	30
Techniki projektowania algorytmów	30
Wymiar danych	31
Wymiar obliczeniowy	33
Analiza efektywności	33
Analiza pamięciowej złożoności obliczeniowej	33
Czasowa złożoność obliczeniowa	35
Szacowanie efektywności	36
Notacja dużego O	36
Złożoność stała ($O(1)$)	38
Złożoność liniowa ($O(n)$)	39
Złożoność kwadratowa ($O(n^2)$)	39
Złożoność logarytmiczna	40
Wybór algorytmu	41
Walidacja algorytmu	41
Algorytmy dokładne, aproksymacyjne i randomizowane	41
Możliwość wyjaśnienia	43
Podsumowanie	43

ROZDZIAŁ 2**Struktury danych w algorytmach 44**

Struktury danych w Pythonie	44
Lista	45
Krotka	49
Słowniki i zbiory	50
Struktury danych Series i DataFrame	54
Macierz	57
Abstrakcyjne typy danych	58
Wektor	59
Stos	60
Kolejka	62
Drzewo	64
Podsumowanie	66

ROZDZIAŁ 3**Algorytmy sortowania i wyszukiwania 67**

Wprowadzenie do algorytmów sortowania	67
Zamiana wartości zmiennych w Pythonie	68
Sortowanie bąbelkowe	68
Sortowanie przez wstawianie	71
Sortowanie przez scalanie	73
Sortowanie Shella	75
Sortowanie przez wybieranie	77
Wybór właściwego algorytmu sortującego	78
Wprowadzenie do algorytmów wyszukiwania	79
Wyszukiwanie liniowe	80
Wyszukiwanie binarne	80
Wyszukiwanie interpolacyjne	81
Praktyczne przykłady	82
Podsumowanie	84

ROZDZIAŁ 4**Projektowanie algorytmów 85**

Wprowadzenie do projektowania algorytmów	85
Kwestia 1: Poprawność. Czy algorytm zwraca rezultat, jakiego oczekujemy?	86
Kwestia 2: Efektywność. Czy robi to w optymalny sposób?	87
Kwestia 3: Skalowalność. Jak efektywny będzie ten algorytm zastosowany do większych zbiorów danych?	91

Strategie algorytmiczne	92
Strategia „dziel i rządź”	92
Strategia programowania dynamicznego	94
Strategia algorytmu zachłannego	95
Praktyczny przykład — rozwiązanie problemu komiwojażera	97
Metoda siłowa	98
Zastosowanie algorytmu zachłannego	100
Porównanie trzech strategii	101
Algorytm PageRank	102
Definicja problemu	102
Implementacja algorytmu PageRank	102
Programowanie liniowe	105
Definicja problemu w programowaniu liniowym	105
Praktyczny przykład — planowanie przepustowości za pomocą programowania liniowego	106
Podsumowanie	108

ROZDZIAŁ 5

Algorytmy grafowe 109

Zwięźle wprowadzenie do grafów	109
Grafy jako szkielet nowoczesnych sieci danych	110
Podstawa grafów: węzły (lub wierzchołki)	111
Teoria grafów i analiza sieci	112
Reprezentacja grafów	112
Mechanika i typy grafów	112
Sieci egocentryczne	113
Wprowadzenie do teorii analizy sieciowej	115
Najkrótsza ścieżka	115
Wskaźnik centralności	117
Obliczanie wskaźników centralności w Pythonie	120
Analiza sieci społecznościowych	122
Przeglądanie grafu	123
Wyszukiwanie wszerek	123
Wyszukiwanie w głąb	127
Studium przypadku — wykrywanie oszustw za pomocą SNA	129
Wprowadzenie	129
Czym jest oszustwo w tym kontekście?	129
Prosta analiza pod kątem oszustwa	131
Podejście strażnicy	132
Podsumowanie	134

CZĘŚĆ 2. Algorytmy uczenia maszynowego

ROZDZIAŁ 6

Algorytmy nienadzorowanego uczenia maszynowego	137
Wprowadzenie do nienadzorowanego uczenia maszynowego	137
Uczenie nienadzorowane w cyklu życia eksploracji danych	138
Trendy badawcze w zakresie uczenia nienadzorowanego	141
Praktyczne przykłady	142
Algorytmy klasteryzacji	142
Algorytm k-średnich (algorytm centroidów)	146
Kroki grupowania hierarchicznego	151
Implementacja grupowania hierarchicznego	152
Algorytm DBSCAN	152
Tworzenie klastrów przy użyciu algorytmu DBSCAN w Pythonie	154
Ocena klastrów	155
Redukcja wymiarów	156
Analiza głównych składowych	157
Wyszukiwanie reguł asocjacyjnych	163
Rodzaje reguł	163
Wskaźniki reguł	164
Algorytmy analizy asocjacyjnej	166
Podsumowanie	171

ROZDZIAŁ 7

Tradycyjne algorytmy uczenia nadzorowanego	172
Nadzorowane uczenie maszynowe	173
Problemy nadzorowanego uczenia maszynowego	173
Warunki konieczne	176
Rozróżnienie między klasyfikatorami a regresorami	177
Algorytmy klasyfikujące	177
Wyzwanie dla klasyfikatorów	178
Tablica pomyłek	184
Kompromis między czułością i precyzją	187
Algorytm drzewa decyzyjnego	194
Algorytm klasyfikujący drzewa decyzyjnego	195
Wady i zalety klasyfikatorów opartych na drzewach decyzyjnych	197
Przypadki użycia	198
Metody zespolone	198
Implementacja wzmacniania gradientowego	199
Różnica między lasem losowym a wzmocnieniem gradientowym	202

Wykorzystanie algorytmu lasu losowego do wyzwania dla klasyfikatorów	202
Regresja logistyczna	204
Założenia	204
Określanie relacji	204
Funkcje straty i kosztu	205
Kiedy używać regresji logistycznej?	206
Wykorzystanie algorytmu regresji logistycznej do wyzwania dla klasyfikatorów	206
Maszyna wektorów nośnych	207
Wykorzystanie maszyny wektorów nośnych do wyzwania dla klasyfikatorów	208
Naiwny klasyfikator bayesowski	209
Twierdzenie Bayesa	209
Wyliczanie prawdopodobieństwa	210
Reguły mnożenia dla koniunkcji zdarzeń	210
Ogólne zasady mnożenia	211
Zasady dodawania dla alternatywy zdarzeń	211
Wykorzystanie naiwnego klasyfikatora bayesowskiego do wyzwania dla klasyfikatorów	211
Zwycięzcą wśród algorytmów klasyfikacji jest... ..	212
Algorytmy regresji	213
Wyzwanie dla regresji	213
Definicja problemu	214
Dane historyczne	214
Inżynieria cech w strumieniowym przetwarzaniu danych	214
Regresja liniowa	215
Prosta regresja liniowa	216
Ewaluacja regresorów	217
Regresja wielomianowa	218
Wykorzystanie algorytmu regresji liniowej do wyzwania dla regresji	219
Kiedy używa się regresji liniowej?	219
Wady regresji liniowej	220
Algorytm drzewa regresji	220
Wykorzystanie drzewa regresji do wyzwania dla regresji	220
Regresyjny algorytm wzmocnienia gradientowego	221
Wykorzystanie algorytmu wzmocnienia gradientowego do wyzwania dla regresji	221
Zwycięzcą wśród algorytmów regresji jest... ..	222
Praktyczny przykład: jak przewidywać pogodę	222
Podsumowanie	224

ROZDZIAŁ 8**Algorytmy sieci neuronowych 226**

Wprowadzenie do sieci neuronowych	227
Tło historyczne	227
Początki sztucznej inteligencji	228
Sieci neuronowe	229
Perceptrony	229
Intuicyjne rozumienie sieci neuronowych	230
Warstwowe architektury uczenia głębokiego	231
Trenowanie sieci neuronowej	234
Anatomia sieci neuronowej	234
Definicja gradientu prostego	235
Funkcje aktywacji	237
Funkcja kroku	238
Funkcja sigmoidalna	238
Jednostronnie obcięta funkcja liniowa (funkcja ReLU)	239
Tangens hiperboliczny (tanh)	241
Znormalizowana funkcja wykładnicza (funkcja softmax)	242
Narzędzia i modele	243
Keras	243
Wybór pomiędzy modelem sekwencyjnym a funkcjonalnym	248
TensorFlow	248
Podstawowe pojęcia TensorFlow	249
Matematyka tensorów	250
Rodzaje sieci neuronowych	251
Sieć konwolucyjna	251
Generatywne sieci przeciwstawne	252
Uczenie transferowe	253
Studium przypadku — użycie uczenia głębokiego do wykrywania oszustw	254
Metodyka	254
Podsumowanie	258

ROZDZIAŁ 9**Algorytmy przetwarzania języka naturalnego 259**

Wprowadzenie do przetwarzania języka naturalnego	259
Terminologia przetwarzania języka naturalnego	260
Wstępne przetwarzanie tekstu w NLP	261
Czyszczenie danych przy użyciu Pythona	266
Macierz słowo – dokument	267
Macierz TF-IDF	268
Podsumowanie i omówienie wyników	269

Wprowadzenie do wektorów słów	269
Implementacja osadzania słów za pomocą metody Word2Vec	270
Interpretowanie wartości podobieństwa	272
Zalety i wady Word2Vec	272
Studium przypadku — analiza sentymentu w recenzjach restauracji	273
Import potrzebnych bibliotek i załadowanie zbioru danych	273
Budowa czystego korpusu — wstępne przetwarzanie danych	274
Konwersja danych tekstowych na cechy numeryczne	274
Analiza wyników	275
Zastosowania NLP	275
Podsumowanie	276

ROZDZIAŁ 10

Modele sekwencyjne	277
Dane sekwencyjne	277
Typy modeli sekwencyjnych	279
Reprezentacja danych dla modeli sekwencyjnych	282
Wprowadzenie do rekurencyjnych sieci neuronowych	283
Architektura sieci RNN	283
Szkolenie sieci RNN w pierwszym kroku czasowym	285
Propagacja wsteczna w czasie	290
Ograniczenia podstawowych sieci RNN	291
Komórki GRU	293
Bramka aktualizacji	295
Implementacja bramki aktualizacji	295
Aktualizacja ukrytej komórki	296
Wprowadzenie do modeli LSTM	297
Bramka zapomnienia	298
Kandydująca komórka stanu	298
Komórka aktualizacji	298
Obliczanie stanu pamięci	299
Bramka wyjściowa	299
Składanie wszystkiego w całość	300
Kodowanie modeli sekwencyjnych	301
Podsumowanie	305

ROZDZIAŁ 11

Zaawansowane modelowanie sekwencyjne	307
Ewolucja zaawansowanych technik modelowania sekwencyjnego	308
Autokodery	308
Kodowanie automatycznego kodera	309
Przygotowanie środowiska	310
Model Seq2Seq	312
Koder	313
Wektor myśli	313
Dekoder	313
Tokeny specjalne w Seq2Seq	313
Dylemat informacyjny wąskiego gardła	314
Mechanizm uwagi	314
Czym jest uwaga w sieciach neuronowych	314
Trzy kluczowe aspekty mechanizmów uwagi	316
Mechanizmy uwagi — szczegóły	317
Ograniczenia mechanizmów uwagi	317
Samouwaga	318
Wagi uwagi	319
Koder — dwukierunkowe sieci RNN	319
Wektor myśli	320
Dekoder — zwykłe sieci RNN	320
Szkolenie a inferencja	320
Transformatory — kolejny po samouwadze etap ewolucji sieci neuronowych	321
Jakie są największe zalety transformatorów?	322
Analiza kodu w Pythonie	322
Interpretacja wyników	323
Duże modele językowe	323
Uwaga w modelach LLM	324
GPT i BERT — najbardziej znane modele NLP	324
Tworzenie zaawansowanych modeli LLM przy użyciu głębokich i szerokich modeli	326
Bottom of Form	327
Podsumowanie	328

CZĘŚĆ 3. Zagadnienia zaawansowane

ROZDZIAŁ 12

Systemy rekomendacji	331
Wprowadzenie do systemów rekomendacji	332
Typy systemów rekomendacji	332
Systemy rekomendacji oparte na treści	332
Systemy rekomendacji oparte na filtrowaniu kooperacyjnym	334
Hybrydowe systemy rekomendacji	336
Ograniczenia systemów rekomendacji	338
Zimny start	338
Wymagania dotyczące metadanych	339
Problem rzadkości danych	339
Obosieczny miecz wpływu społecznościowego w systemach rekomendacji	340
Obszary praktycznych zastosowań	340
Mistrzowskie wykorzystanie rekomendacji opartych na danych przez Netfliksa	341
Ewolucja systemu rekomendacji Amazona	341
Przykład praktyczny — tworzenie systemu rekomendacji	342
1. Przygotowanie środowiska	342
2. Ładowanie danych — recenzji i tytułów	342
3. Połączenie danych — tworzenie kompletnego widoku	343
4. Analiza opisowa — wyciąganie wniosków na podstawie ocen	344
5. Przygotowanie do generowania rekomendacji — tworzenie macierzy	344
6. Testowanie systemu — rekomendowanie filmów	345
Podsumowanie	347

ROZDZIAŁ 13

Algorytmiczne przetwarzanie danych	348
Wprowadzenie do algorytmów danych	348
Twierdzenie CAP w kontekście algorytmów danych	349
Przechowywanie danych w systemach rozproszonych	349
Twierdzenie CAP i kompresja danych	349
Twierdzenie CAP	350
Systemy CA	351
Systemy AP	351
Systemy CP	352

Algorytmy kompresji danych	352
Techniki kompresji bezstratnej	353
Praktyczny przykład — zarządzanie danymi w AWS, czyli koncentracja na twierdzeniu CAP i algorytmach kompresji	358
1. Zastosowanie twierdzenia CAP	358
2. Algorytmy kompresji	359
3. Ocena korzyści	359
Podsumowanie	360

ROZDZIAŁ 14

Kryptografia 361

Wprowadzenie do kryptografii	361
Waga najsłabszego ogniwa	362
Terminologia	362
Wymagania bezpieczeństwa	363
Podstawy projektowania szyfrów	365
Rodzaje technik kryptograficznych	368
Kryptograficzna funkcja skrótu	368
Szyfrowanie symetryczne	372
Szyfrowanie asymetryczne	374
Przykład — kwestie bezpieczeństwa we wdrażaniu modelu uczenia maszynowego	379
Atak man-in-the-middle	379
Obrona przed techniką masquerading	381
Szyfrowanie danych i modelu	381
Podsumowanie	383

ROZDZIAŁ 15

Algorytmy przetwarzania danych w dużej skali 384

Wprowadzenie do algorytmów wielkoskalowych	384
Charakterystyka wydajnej infrastruktury dla algorytmów wielkoskalowych	385
Elastyczność	386
Cechy dobrze zaprojektowanego algorytmu wielkoskalowego	386
Strategia przetwarzania przez wiele zasobów	389
Teoretyczne możliwości przetwarzania równoległego	390
Prawo Amdahla	390
Wyprowadzanie prawa Amdahla	390
CUDA — wykorzystanie architektury GPU do przetwarzania równoległego	393
Przetwarzanie klastrowe przy użyciu Apache Spark	397

Jak Apache Spark wspomaga wykonywanie algorytmów wielkoskalowych? ...	399
Przetwarzanie rozproszone	399
Przetwarzanie w pamięci	399
Algorytmy wielkoskalowe w przetwarzaniu chmurowym	399
Przykład	400
Podsumowanie	400

ROZDZIAŁ 16

Uwagi praktyczne	401
Problemy dotyczące rozwiązań algorytmicznych	401
Oczekiwać nieoczekiwanego	402
Porażka bota sztucznej inteligencji Twittera	403
Transparentność algorytmu	403
Algorytmy uczenia maszynowego i transparentność	404
Etyka i algorytmy	409
Problemy z algorytmami uczącymi się	410
Znaczenie kwestii etycznych	410
Czynniki wpływające na rozwiązania algorytmiczne	411
Ograniczanie stronniczości modeli	412
Kiedy używać algorytmów?	413
Zdarzenia według teorii czarnego łabędzia i ich wpływ na algorytmy	414
Podsumowanie	415

Algorytmy nienadzorowanego uczenia maszynowego

Rozdział

6

Ten rozdział poświęcony jest algorytmom nienadzorowanego uczenia maszynowego. Po jego lekturze będziesz umieć zastosować podstawowe algorytmy i metody uczenia nienadzorowanego do skutecznego rozwiązywania rzeczywistych problemów.

Zagadnienia poruszane w tym rozdziale:

- wprowadzenie do nienadzorowanego uczenia maszynowego,
- algorytmy klasteryzacji,
- redukcja wymiarów,
- reguły asocjacyjne.

Wprowadzenie do nienadzorowanego uczenia maszynowego

Jeśli dane nie są wygenerowane losowo, to można w nich wykryć pewne wzorce lub relacje łączące poszczególne elementy w wielowymiarowej przestrzeni. Nienadzorowane uczenie maszynowe to proces wykrywania i wykorzystywania takich wzorców ze zbiorów danych w celu lepszego ich zrozumienia i nadania im bardziej zrozumiałej struktury. Algorytmy nienadzorowanego uczenia maszynowego odkrywają te schematy i przy ich użyciu nadają strukturę zbiorowi danych. Identyfikacja tych wzorców pozwala lepiej zrozumieć i reprezentować dane. Ekstrakcja wzorców z nieprzetworzonych danych daje możliwość lepszego zrozumienia tych danych.

Koncepcję tę przedstawia rysunek 6.1.

W dalszej części tego rozdziału poznasz cykl życia CRISP-DM, popularny model procesu uczenia maszynowego, na którego podstawie dowiesz się, jakie miejsce zajmuje ten proces. Aby zrozumieć uczenie nienadzorowane, można sobie wyobrazić detektywa, który dzięki powiązaniu tropów formułuje wzorce lub grupy bez posiadania jakiegokolwiek wcześniejszej wiedzy na temat tego, jaki może być wynik. Tak jak spostrzeżenia detektywa mogą być kluczowe dla rozwiązania sprawy, uczenie nienadzorowane odgrywa kluczową rolę w cyklu uczenia maszynowego.

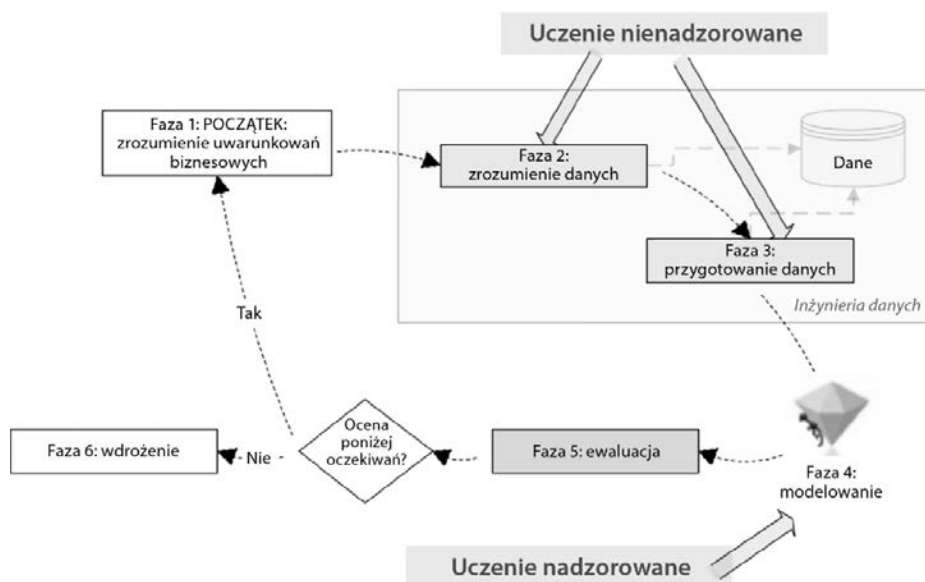


Rysunek 6.1. Ekstrakcja wzorców z nieoznakowanego zbioru nieprzetworzonych danych za pomocą nienadzorowanego uczenia maszynowego

Uczenie nienadzorowane w cyklu życia eksploracji danych

Najpierw przyjrzymy się trzem fazom typowego procesu uczenia maszynowego. Aby je zrozumieć, przeanalizujemy przykład zastosowania uczenia maszynowego w procesie eksploracji danych. Eksploracja danych to technika odkrywania istotnych korelacji, wzorców i trendów w określonym zbiorze danych. W tej książce poszczególne fazy eksploracji danych przy użyciu uczenia maszynowego opisują na podstawie metody **CRISP-DM** (ang. *Cross-Industry Standard Process for Data Mining*). Została ona stworzona przez zespół specjalistów ds. eksploracji danych zatrudnionych w różnych organizacjach, m.in. w Chryslerze i IBM). Więcej informacji na ten temat znajduje się na stronie <https://www.ibm.com/docs/en/spss-modeler/saas?topic=dm-crisp-help-overview>.

Cykl CRISP-DM składa się z sześciu faz, które są pokazane na rysunku 6.2.



Rysunek 6.2. Fazy cyklu CRISP-DM

Przyjrzymy się tym fazom po kolei.

Faza 1 — zrozumienie uwarunkowań biznesowych

Ta faza obejmuje zebranie wymagań i próbę zrozumienia całości problemu z biznesowego punktu widzenia. Zdefiniowanie zakresu problemu oraz właściwe przetłumaczenie go na kategorie uczenia maszynowego stanowi istotną część tej fazy. Na tym etapie identyfikowane są cele, definiowany jest zakres projektu oraz formułowane są oczekiwania zainteresowanych stron.

Uwaga

Trzeba pamiętać, że faza 1 w modelu CRISP-DM poświęcona jest zrozumieniu uwarunkowań biznesowych. Koncentruje się ona na tym, co trzeba zrobić, a nie jak to osiągnąć.

Faza 2 — rozumienie danych

Ten etap polega na zrozumieniu danych, z jakich możemy skorzystać w procesie eksploatacji. Dowiemy się, czy w zbiorach danych, którymi dysponujemy, dostępne są wszystkie informacje potrzebne do rozwiązania problemu zdefiniowanego w fazie 1. Aby zrozumieć wzorce występujące w danych, możemy używać takich narzędzi, jak wizualizacja danych, pulpity i raporty podsumowujące. Jak piszę w dalszej części tego rozdziału, algorytmy nienadzorowanego uczenia maszynowego mogą być także używane do odkrywania wzorców w danych oraz do szczegółowej analizy ich struktury, co pozwala na ich zrozumienie.

Faza 3 — przygotowanie danych

Chodzi tu o przygotowanie danych dla modelu uczenia maszynowego, który zostanie wytrenowany w fazie 4. W zależności od przypadku użycia i wymogów przygotowywanie danych może obejmować usunięcie wartości odstających, normalizację, usunięcie wartości pustych oraz redukcję liczby wymiarów danych. Bardziej szczegółowy opis tych spraw znajduje się w dalszej części tego rozdziału. Po przetworzeniu i przygotowaniu danych zazwyczaj dzieli się je według współczynnika 70 do 30. Większa część, zwana danymi szkoleniowymi, jest wykorzystywana do uczenia modelu różnych wzorów, a mniejsza — zwana danymi testowymi — zostaje zapisana w celu oceny skuteczności modelu w działaniu na nieznanach danych w fazie 5. Dodatkowo może zostać odłożony jeszcze jeden zbiór danych do weryfikacji i dostrajania modelu, aby zapobiec przeuczeniu.

Faza 4 — modelowanie

Na tym etapie formułujemy wzorce obecne w danych przez szkolenie modelu. Do szkolenia modelu używamy zbioru danych szkoleniowych przygotowanego w poprzedniej fazie. Szkolenie modelu polega na przekazaniu przygotowanych danych do algorytmu uczenia maszynowego. Ten identyfikuje wzorce obecne w danych i uczy się ich w iteracyjnym procesie nauki. Celem jest sformułowanie wzorców reprezentujących relacje i zależności między różnymi zmiennymi w zbiorze danych. W dalszych rozdziałach opisuję, jak złożoność i właściwości tych matematycznych formuł zależą od wybranego przez nas algorytmu. Na przykład: model regresji liniowej wygeneruje równanie liniowe, a model drzewa decyzyjnego utworzy model decyzji w postaci drzewa.

Oprócz szkolenia modelu w tej fazie cyklu CRISP-DM przeprowadzane jest także dostrajanie modelu. Proces ten obejmuje optymalizację parametrów algorytmu uczenia w celu zwiększenia jego wydajności, aby poprawić dokładność przewidywań. Używany jest opcjonalny zbiór weryfikacyjny, który pomaga w regulacji poziomu złożoności modelu i znalezieniu równowagi między nauką na podstawie danych a generalizacją na niewidziane dane. Zbiór weryfikacyjny, w kontekście uczenia maszynowego, to część zbioru służącego do dostrajania modelu predykcyjnego. Pomaga w dostosowywaniu poziomu złożoności modelu w celu znalezienia optymalnej równowagi między nauką na podstawie znanych danych i uogólnieniem tego na nieznane dane. Równowaga ta pomaga w zapobieganiu przeuczeniu, które objawia się tym, że model zbyt dobrze poznaje dane szkoleniowe, ale słabo sprawdza się na nowych danych, których jeszcze nie poznał. W związku z tym dostrajanie modelu nie tylko doskonali jego możliwości predykcyjne, ale również zwiększa jego niezawodność i pewność.

Faza 5 — ewaluacja

Na tym etapie oceniamy wyszkolony model przy użyciu danych testowych wydzielonych w fazie 3. Badamy jego skuteczność w odniesieniu do oczekiwań określonych w fazie 1. Oczekiwania podstawowe w uczeniu maszynowym stanowią punkt odniesienia, który można ustalić na różne sposoby, na przykład przez podstawowe systemy oparte na regułach, proste modele statystyczne, losowe zdarzenia, a nawet na podstawie skuteczności ludzi będących ekspertami. Celem tych podstawowych oczekiwań jest określenie minimalnego progu efektywności dla modelu uczenia maszynowego. Stanowią one podstawę porównawczą i punkt odniesienia do naszych oczekiwań. Jeśli wynik oceny modelu zgadza się z oczekiwaniami zdefiniowanymi w fazie 1, przechodzimy dalej. Jeśli nie, musimy wrócić i przejść jeszcze raz przez wszystkie poprzednie fazy, zaczynając od pierwszej.

Faza 6 — wdrożenie

Po zakończeniu fazy oceny sprawdzamy, czy skuteczność wyszkolonego modelu spełnia lub przekracza oczekiwania. Należy pamiętać, że pozytywny wynik oceny nie oznacza automatycznie gotowości do wdrożenia. Model sprawdził się na naszych danych testowych, ale to nie jedyne kryterium pozwalające ocenić, czy model jest gotowy do rozwiązywania rzeczywistych problemów zgodnie z definicją w fazie 1. Musimy sprawdzić, jak model spisze się na nowych danych, których nigdy wcześniej nie „widział”, jak będzie się integrował z istniejącymi systemami oraz jak będzie się sprawował w nieprzewidzianych przypadkach brzegowych. Dlatego dopiero po przeprowadzeniu dokładnych testów i uzyskaniu satysfakcjonujących wyników możemy przejść do wdrażania modelu w środowisku produkcyjnym, w którym będzie on dostarczał praktyczne rozwiązanie zdefiniowanego problemu.

Uwaga

Faza 2 (zrozumienie danych) i faza 3 (przygotowanie danych) modelu CRISP-DM sprowadzają się do zrozumienia i przygotowania danych na potrzeby trenowania modelu. Obejmują one przetwarzanie danych. W niektórych organizacjach do procesu inżynierii danych zatrudnia się specjalistów.

Oczywistym jest, że proces sugerowania rozwiązania problemu jest w pełni oparty na danych. Połączenie uczenia nadzorowanego i nienadzorowanego stosowane jest w celu sformułowania działającego rozwiązania. W tym rozdziale skupię się jednak na części obejmującej nienadzorowane uczenie maszynowe.

Uwaga

Na inżynierię danych składają się faza 2 i faza 3. Łącznie stanowią one najdłuższą część procesu uczenia maszynowego — mogą one pochłoniąć nawet 70% czasu i zasobów typowego projektu z zakresu uczenia maszynowego (*Data Management in Machine Learning: Challenges, Techniques, and Systems*, Cody et al, SIGMOD '17: *Proceedings of the 2017 ACM International Conference on Management of Data*, maj 2017 r.). Algorytmy uczenia nienadzorowanego mogą odegrać istotną rolę w inżynierii danych.

W kolejnych sekcjach przedstawiam więcej szczegółów na temat algorytmów nienadzorowanych.

Trendy badawcze w zakresie uczenia nienadzorowanego

Dziedzina badań w zakresie uczenia maszynowego przeszła wiele zmian. Początkowo skupiała się głównie na technikach uczenia nadzorowanego. Te metody można natychmiast wykorzystywać do zadań związanych z dedukcją, a do oczywistych korzyści z ich używania należą oszczędność czasu, redukcja kosztów oraz wyraźne zwiększenie dokładności przewidywań.

Jednak możliwości algorytmów nienadzorowanego uczenia maszynowego dopiero niedawno zwróciły uwagę badaczy. W odróżnieniu od algorytmów uczenia nadzorowanego działają one bez bezpośrednich instrukcji ani wstępnych założeń. Nadają się do eksplorowania szerszych „wymiarów” lub aspektów danych, umożliwiając w ten sposób bardziej wyczerpującą analizę zbioru danych.

Słowem wyjaśnienia: w terminologii uczenia maszynowego „cechy” są indywidualnymi, mierzalnymi właściwościami zjawisk poddawanych obserwacji. Na przykład w zbiorze danych dotyczących informacji o klientach cechami mogą być takie aspekty, jak wiek klienta, historia zakupów lub przeglądane strony. Natomiast „etykiety” reprezentują wyniki, jakie chcemy, aby nasz model przewidywał na podstawie tych cech.

Podczas gdy uczenie nadzorowane koncentruje się głównie na ustanawianiu relacji między cechami a określoną etykietą, uczenie nienadzorowane nie ogranicza się do określonej z góry etykiety. Ta technika może sięgać głębiej, odkrywając skomplikowane wzorce wśród różnych cech, które mogłyby zostać przeoczone przez metody nadzorowane. To sprawia, że uczenie nienadzorowane ma większy potencjał i szerszy zakres potencjalnych zastosowań.

Ta inherentna elastyczność uczenia nienadzorowanego nieodwołalnie pociąga za sobą pewną trudność. Większy obszar badań wymaga **większej mocy obliczeniowej**, co podnosi

koszty i wydłuża czas przetwarzania. Ponadto opanowanie skali lub zakresu zadań uczenia nienadzorowanego może być skomplikowane ze względu na badawczą naturę tych technik. Mimo to możliwość odkrywania ukrytych wzorców lub korelacji w danych czyni uczenie nienadzorowane potężnym narzędziem do analizy danych.

Obecnie trendy badawcze przesuwają się w kierunku integracji metod uczenia nadzorowanego i nienadzorowanego. Ta połączona strategia ma na celu wykorzystanie zalet obu technik.

Teraz przyjrzymy się paru praktycznym przykładom.

Praktyczne przykłady

Obecnie używa się uczenia nienadzorowanego do lepszego zrozumienia danych i nadania im struktury — m.in. w segmentacji na potrzeby marketingu, wykrywaniu oszustw czy analizie koszyka (o czym będzie mowa w dalszej części rozdziału). Przyjrzymy się przykładowi użycia uczenia nienadzorowanego do segmentacji rynku.

Segmentacja rynku przy użyciu uczenia nienadzorowanego

Uczenie nienadzorowane to potężne narzędzie do segmentacji rynku. Segmentacja rynku to proces dzielenia rynku docelowego na grupy na podstawie wspólnych cech, co pozwala firmom na dostosowanie strategii marketingowych i reklam, aby skutecznie docierać do określonych segmentów klientów i ich aktywować. Rynek docelowy można grupować według takich cech, jak demografia, zachowania lub bliskość geograficzna. Za pomocą algorytmów i technik statystycznych firmy wydobywają cenne informacje z danych klientów, odnajdują ukryte wzorce oraz grupują klientów w segmenty na podstawie podobieństw zachowań, preferencji lub cech. To oparte na danych podejście umożliwia marketerom tworzenie dopasowanych strategii, zwiększenie skuteczności targetowania oraz podniesienie ogólnej skuteczności kampanii marketingowych.

Algorytmy klasteryzacji

Jedna z najprostszych, a zarazem najczęściej stosowanych technik uczenia nienadzorowanego polega na grupowaniu podobnych wzorców za pomocą algorytmu klasteryzacji. Korzysta się z niej do zrozumienia pewnego aspektu danych związanych z problemem, jaki staramy się rozwiązać. Algorytmy klasteryzacji szukają naturalnych grup próbek danych. Ponieważ grupy te nie opierają się na żadnych wspólnych celach czy założeniach, technikę tę klasyfikuje się jako technikę uczenia nienadzorowanego.

Wyobraź sobie wielką bibliotekę pełną książek. Każda z nich reprezentuje punkt danych, który ma szereg atrybutów, takich jak gatunek, autor, data wydania itd. Teraz wyobraź sobie bibliotekarza (algorytm klastrowania), którego zadaniem jest uporządkowanie tych książek. Bez podanych z góry kategorii i instrukcji bibliotekarz zacznie sortować książki według ich atrybutów — kryminały razem, klasyki razem, książki jednego autora razem itd. Takie zbiory w elementach danych nazywamy „grupami naturalnymi” — elementy o podobnych cechach przynależą do tej samej grupy.

Grupy wskazane przez różne algorytmy klasteryzacji opierają się na znajdowaniu podobieństw pomiędzy różnymi próbkami danych w dziedzinie problemu. W kontekście uczenia maszynowego punkt danych jest zbiorem pomiarów lub spostrzeżeń, które istnieją w wielowymiarowej przestrzeni. Mówiąc prościej, jest to pojedyncza porcja informacji, która pomaga maszynie w nauce na temat zadania, które próbuje wykonać. Najlepszy sposób określenia podobieństwa między nimi będzie się różnił między kolejnymi problemami i będzie w dużej mierze zależał od ich natury. Przyjrzyjmy się jednak kilku metodom wyliczania podobieństwa między próbkami danych.

Określanie podobieństw

Działanie technik uczenia nienadzorowanego, takich jak algorytmy klastrujące, opiera się na znajdowaniu podobieństw między różnymi punktami danych w określonej przestrzeni problemu. Ich skuteczność w dużym stopniu zależy od naszej zdolności do poprawnego pomiaru tych podobieństw, które w terminologii uczenia maszynowego nazywają się „miarami odległości”. Czym dokładnie jest miara odległości?

Miara odległości to matematyczny wzór lub matematyczna metoda obliczania „odległości” lub podobieństwa między dwoma punktami danych. Należy mieć na uwadze, że w tym kontekście termin „odległość” nie oznacza dystansu fizycznego, tylko podobieństwo lub brak podobieństwa między punktami danych na podstawie ich cech.

W przypadku klastrowania możemy mówić o dwóch typach odległości: międzyklastrowej i wewnątrzklastrowej. Międzyklastrowa odległość to dystans między różnymi klastrami lub grupami punktów danych. Natomiast odległość wewnątrzklastrowa odnosi się do dystansu między punktami w obrębie jednego klastra lub jednej grupy. Celem dobrego algorytmu klastrującego jest maksymalizacja odległości międzyklastrowej (aby każdy klaster był oddzielony od pozostałych) przy jednoczesnej minimalizacji odległości wewnątrzklastrowej (aby punkty danych w jednym klastrze były jak najbardziej podobne do siebie). Poniżej przedstawiam trzy najpopularniejsze metody obliczania podobieństwa:

- odległość euklidesowa,
- metryka miejska,
- miara kosinusowa.

Zajmiemy się każdą z nich po kolei.

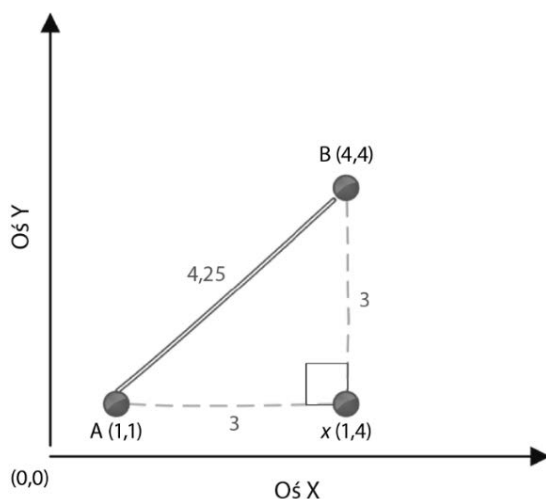
Odległość euklidesowa

Odległość między punktami może wyrażać podobieństwo między próbkami danych. Z tej metody korzysta się często w uczeniu nienadzorowanym, m.in. w klasteryzacji. Odległość euklidesowa to najpopularniejszy i najprostszy wskaźnik. W tym kontekście pojęcie odległości określa stopień podobieństwa lub różnicy między dwoma punktami w wielowymiarowej przestrzeni, co ma kluczowe znaczenie dla zrozumienia grupowania punktów danych. Jedną z najprostszych i najczęściej używanych miar tej odległości jest odległość euklidesowa.

Ten typ odległości można wyobrazić sobie jako dystans w prostej linii między dwoma punktami w trójwymiarowej przestrzeni — tak samo mierzymy odległość między punktami w rzeczywistym świecie. Weźmy na przykład dwa miasta zaznaczone na mapie.

Odległość euklidesowa między nimi byłaby równa długości łączącej je linii prostej, bez uwzględnienia rozmaitych przeszkód, takich jak góry czy rzeki.

Analogicznie w przestrzeni wielowymiarowej naszych danych odległość euklidesowa to najkrótsza „prosta linia”, jaką można poprowadzić między dwoma punktami danych. Pozwala ona dowiedzieć się, jaka odległość dzieli te dwa punkty na podstawie ich cech lub atrybutów. Przyjrzyjmy się dwóm punktom: A (1,1) i B (4,4) w dwuwymiarowej przestrzeni przedstawionej na wykresie widocznym na rysunku 6.3.



Rysunek 6.3. Obliczanie odległości euklidesowej między dwoma punktami

Aby obliczyć odległość między A i B, tj. $d(A,B)$, możemy skorzystać z poniższego twierdzenia Pitagorasa:

$$d(A,B) = \sqrt{(a_2 - b_2)^2 + (a_1 - b_1)^2} = \sqrt{(4 - 1)^2 + (4 - 1)^2} = \sqrt{9 + 9} = 4,25$$

Zwróć uwagę, że te obliczenia dotyczą dwuwymiarowej dziedziny problemu. W przypadku n -wymiarowej przestrzeni odległość między punktami A i B obliczamy następująco:

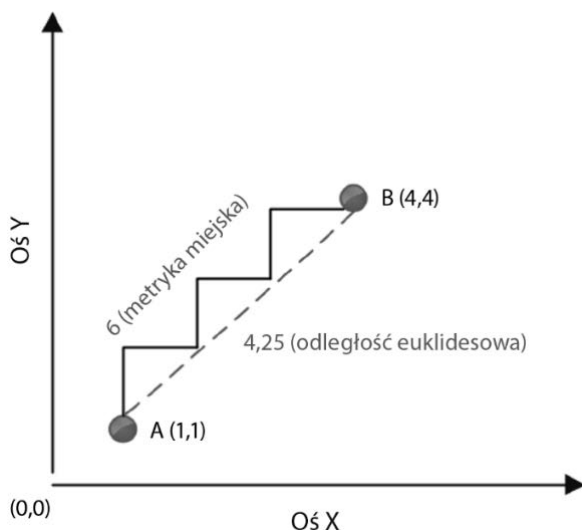
$$d(A,B) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

Metryka miejska

W wielu przypadkach zmierzenie najkrótszej linii między dwoma punktami za pomocą odległości euklidesowej nie oddaje rzeczywistego podobieństwa czy bliskości dwóch próbek danych. Przykładowo jeśli dwie próbki danych przedstawiają adresy na mapie, realna odległość punktu A od punktu B przemierzana środkami transportu takimi jak samochód czy taksówka będzie większa niż odległość euklidesowa. Wyobraź sobie gwarne miasto, w którym nie da się przechodzić przez budynki, aby dostać się z jednego punktu do innego (tak jak można w przypadku pomiaru odległości euklidesowej), tylko

trzeba poruszać się po sieci ulic. Metryka miejsca odzwierciedla taką właśnie nawigację w świecie rzeczywistym — pozwala obliczyć całkowitą odległość między punktami A i B w ramach sieci dróg.

W takich sytuacjach korzystamy z metryki miejskiej, która przedstawia odległość między dwoma punktami w sieci dróg jak w mieście. W odróżnieniu od odległości euklidesowej, która określa odległość w linii prostej, metryka miejsca lepiej odzwierciedla odległość między punktami w takich przypadkach. Porównanie odległości euklidesowej i metryki miejskiej widać na wykresie pokazanym na rysunku 6.4.



Rysunek 6.4. Obliczanie metryki miejskiej między dwoma punktami

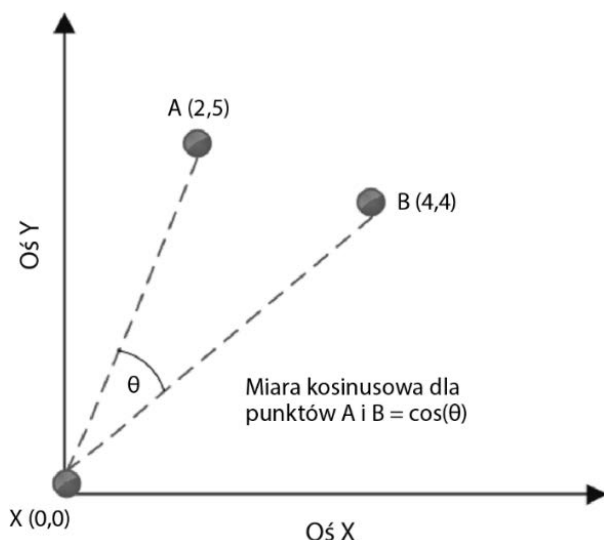
Metryka miejska na tym rysunku została przedstawiona w postaci zygzyka, który biegnie ściśle wzdłuż linii wykresu. Natomiast odległość euklidesowa ma postać prostej linii łączącej punkty A i B . Jest oczywiste, że wartość metryki miejskiej będzie zawsze równa lub wyższa niż odległość euklidesowa dla tego samego przypadku.

Miara kosinusowa

Choć odległość euklidesowa i metryka miejska świetnie sprawdzają się w prostych przestrzeniach o małej liczbie wymiarów, ich skuteczność maleje w bardziej skomplikowanych środowiskach o większej liczbie wymiarów. Przestrzeń wielowymiarowa to zbiór danych zawierający dużą liczbę cech lub zmiennych. Im większa liczba wymiarów (cech), tym mniej dokładne i bardziej czasochłonne stają się obliczenia odległości metodą euklidesową i miejską.

Aby uniknąć tej niedogodności, w kontekstach wielowymiarowych używamy **miary kosinusowej**. Wartość miary kosinusowej wylicza się na podstawie kosinusa kąta tworzonego pomiędzy dwoma punktami połączonymi z pewnym punktem początkowym. W tym przypadku nie chodzi o fizyczną odległość między punktami, tylko o kąt, który one tworzą.

Jeśli próbki danych są sobie bliskie w przestrzeni wielowymiarowej, to kąt między nimi jest mniejszy, niezależnie od liczby wymiarów, jaką cechują się dane. Z kolei jeśli są od siebie odległe, kąt będzie większy. Dlatego miara kosinusowa dostarcza precyzyjniejszą informację na temat podobieństwa w danych wielowymiarowych, co pozwala nam lepiej zrozumieć skomplikowane wzorce danych (rysunek 6.5).



Rysunek 6.5. Obliczanie miary kosinusowej

Uwaga

Dane tekstowe niemal zawsze można traktować jak dane wielowymiarowe. Wynika to z natury danych tekstowych, w których każde słowo można traktować jako osobny wymiar lub indywidualną cechę. Ponieważ miara kosinusowa wypada dobrze w wielowymiarowej przestrzeni, jest ona właściwym wyborem w pracy z tekstem.

Na powyższym rysunku 6.5 kosinus kąta między punktami A (2, 5) i B (4,4) to miara kosinusowa oznaczona symbolem θ . Punktem odniesienia jest początek układu, tj. X (0, 0). W rzeczywistości jednak dowolny punkt w dziedzinie problemu może stanowić punkt odniesienia, nie musi to być początek układu współrzędnych.

Teraz przyjrzymy się jednej z najpopularniejszych technik nienadzorowanego uczenia maszynowego, czyli algorytmowi k -średnich.

Algorytm k -średnich (algorytm centroidów)

Nazwa *algorytmu klasteryzacji k-średnich* bierze się stąd, że tworzy on k klastrów i przy użyciu średnich znajduje odległość między próbkami danych. Termin **średnie** oznacza metodę obliczania centroidu, czyli punktu centralnego, każdego klastra, który zasadniczo jest średnią wszystkich punktów danych w klastrze. Innymi słowy, algorytm ten oblicza

wartość średnią dla każdej cechy w klastrze, co powoduje powstanie nowego punktu danych — centroidu. Potem jest on wykorzystywany jako punkt odniesienia do wyznaczania odległości innych punktów danych.

Popularność algorytmu k -średnich wynika ze skalowalności i szybkości. Jest on bardzo wydajny obliczeniowo, ponieważ wykorzystuje prosty proces iteracyjny, w którym centroidy klastrów są stopniowo poprawiane, aż staną się reprezentatywne dla elementów klastra. Ta prostota sprawia, że algorytm jest wyjątkowo szybki i skalowalny, więc nadaje się do użycia z dużymi zbiorami danych.

Ważnym ograniczeniem algorytmu k -średnich jest to, że nie jest on w stanie określić optymalnej liczby klastrów k . Najlepiej, kiedy wartość ta zależy od naturalnych grup w konkretnym zbiorze danych. Rozumowanie, jakie stoi za tym ograniczeniem, dotyczy postawienia przede wszystkim na prostotę i szybkość, dlatego obliczanie wartości k pozostawiono mechanizmom zewnętrznym. W zależności od kontekstu problemu wartość k może być określona bezpośrednio. Na przykład: jeśli zadanie polega na segregacji klasy uczniów na zajęcia z przetwarzania danych na dwie grupy — jedną obejmującą osoby mające umiejętności w zakresie analizy danych i drugą z osobami potrafiącymi programować — wartość k naturalnie wyniesie dwa. Z kolei w innych przypadkach wartość k może nie być oczywista. Wtedy iteracyjna procedura oparta na próbach i błędach czy algorytm oparty na heurystykach będą tymi metodami, które pozwolą oszacować najwłaściwszą liczbę klastrów dla danego zbioru danych.

Logika klasteryzacji k -średnich

W tej części opisuję logikę algorytmu klasteryzacji k -średnich. Przyjrzymy się jego działaniu krok po kroku, aby umożliwić Ci jego dokładne zrozumienie. Ta część jest poświęcona logice algorytmu klastrowania k -średnich.

Inicjalizacja

W celu pogrupowania próbek danych algorytm k -średnich wykorzystuje miarę odległości, by ocenić podobieństwo czy bliskość punktów. Zanim użyjemy algorytmu k -średnich, musimy wybrać najlepszą miarę odległości między próbkami danych. Domyślnie zastosowana zostanie odległość euklidesowa. Jednak w zależności od natury i wymagań danych bardziej odpowiednia może być inna metoda, na przykład miara miejsca albo kosinusowa. Ponadto jeśli w zbiorze danych są liczne odchylenia, trzeba będzie zdecydować się na kryteria, które posłużą do wskazywania i usuwania obserwacji odstających od zbioru danych.

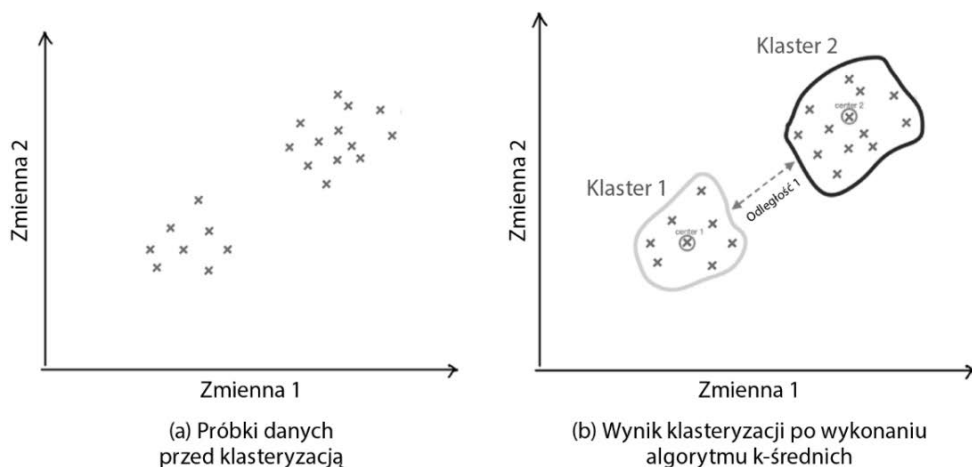
Elementy odstające można wykrywać różnymi metodami statystycznymi, takimi jak metoda Z -score lub IQR (ang. *Interquartile Range*). Teraz przyjrzymy się poszczególnym krokom wykonywania algorytmu k -średnich.

Kroki algorytmu k -średnich

Na algorytm k -średnich składają się następujące kroki:

-
- | | |
|--------|--|
| Krok 1 | Wybieramy liczbę klastrow — k . |
| Krok 2 | Spośród próbek danych losowo wybieramy k punktów jako środków klastrow. |
| Krok 3 | W oparciu o wybraną miarę odległości iteracyjnie obliczamy odległość każdego punktu w dziedzinie od każdego z ustalonych w powyższy sposób środków klastrow. W zależności od wielkości zbioru danych może to być mniej lub bardziej czasochłonny krok. Przykładowo jeśli w klastrze jest 10 000 punktów, a $k = 3$, oznacza to, że będzie trzeba obliczyć 30 000 odległości. |
| Krok 4 | Przypisujemy każdą próbkę danych w dziedzinie problemu do najbliższego środka klastra. |
| Krok 5 | Na tym etapie każda próbka danych jest już przypisana do klastra. Ale wykonywanie algorytmu się nie kończy, ponieważ wybór początkowych środków klastrow był losowy. Musimy teraz zweryfikować, że bieżące, losowo wybrane środki klastrow są rzeczywiście centrami każdego z klastrow. Przeliczamy środki klastrow poprzez ponowne wyliczenie średniej odległości każdego z punktów w klastrze od jego środka. Ten krok wyjaśnia, dlaczego algorytm ten nosi nazwę k -średnich. |
| Krok 6 | Jeśli centra klastrow zmieniły się w kroku 5, oznacza to, że musimy ponownie przeliczyć przypisania wszystkich próbek danych. W tym celu musimy ponownie wykonać zasobożerny krok 3. Jeśli centra klastrow się nie zmieniły lub osiągnęliśmy zadeklarowany wcześniej warunek wyjścia (np. maksymalną liczbę iteracji), wykonywanie algorytmu się kończy. |
-

Rysunek 6.6 przedstawia wynik działania algorytmu k -średnich w dwuwymiarowej dziedzinie problemu:



Rysunek 6.6. Wyniki klastrowania k -średnich (a) Punkty danych przed klastrowaniem; (b) Klaster utworzony przez algorytm klastrowania k -średnich

Zauważ, że dwa wynikowe klastry, powstałe w wyniku wykonania algorytmu k -średnich, są wyraźnie oddzielone. Teraz przyjrzymy się warunkowi wyjścia algorytmu k -średnich.

Warunek wyjścia

W algorytmach uczenia nienadzorowanego, takich jak k -średnich, warunek wyjścia odgrywa kluczową rolę w wybieraniu momentu zakończenia procesu iteracji przez algorytm. Domyślnym warunkiem wyjścia algorytmu k -średnich jest sytuacja, gdy nie ma już w kroku 5 środków do przesunięcia. Ale podobnie jak wiele innych algorytmów, również ten może potrzebować wiele czasu na wykonanie, zwłaszcza gdy przetwarzany jest duży zbiór danych o wielu wymiarach. Zamiast czekać, aż algorytm zakończy działanie, możemy sami jawnie zdefiniować warunek wyjścia w jeden z dwóch sposobów:

- Poprzez wskazanie maksymalnego czasu wykonania:
 - **Warunek wyjścia:** $t > t_{max}$, gdzie t to bieżący czas wykonania, a t_{max} to wyznaczony przez nas maksymalny czas wykonania algorytmu.
- Poprzez wskazanie maksymalnej liczby iteracji:
 - **Warunek wyjścia:** $m > m_{max}$, gdzie m to bieżący czas wykonania, a m_{max} to wyznaczony przez nas maksymalny czas wykonania algorytmu.

Implementacja algorytmu k -średnich

Wykonamy klastrowanie k -średnich na prostym dwuwymiarowym zbiorze danych z dwiema cechami — x i y . Wyobraź sobie rój światełek rozproszonych po całym ogrodzie w nocy. Twoim zadaniem jest ich pogrupowanie na podstawie dzielących ich odległości. To jest istota klastrowania k -średnich, popularnego algorytmu uczenia nienadzorowanego.

Mamy zbiór danych, jak nasz ogród, z punktami danych naniesionymi w dwuwymiarowej przestrzeni. Punkty te reprezentują współrzędne x i y :

```
import pandas as pd
dataset = pd.DataFrame({ 'x': [11, 21, 28, 17, 29, 33, 24, 45, 45, 52, 51, 52,
↪55, 53, 55, 61, 62, 70, 72, 10],
                        'y': [39, 36, 30, 52, 53, 46, 55,
↪59, 63, 70, 66, 63, 58, 23, 14, 8, 18, 7, 24, 10]})
```

Naszym zadaniem jest sklastrowanie tych punktów danych za pomocą algorytmu k -średnich.

Najpierw importujemy potrzebne biblioteki:

```
from sklearn import cluster
import matplotlib.pyplot as plt
```

Następnie inicjalizujemy klasę `KMeans` przez podanie liczby klastrów (k). W tym przykładzie przyjmujemy założenie, że chcemy podzielić zbiór danych na trzy klastry:

```
kmeans = cluster.KMeans(n_clusters=2)
```

Teraz przeszkolimy nasz model `KMeans` przy użyciu naszego zbioru danych. Dobrze jest wiedzieć, że ten model wymaga tylko macierzy cech (x), bez wektora docelowego (y), ponieważ jest to algorytm uczenia nienadzorowanego:

```
kmeans.fit(dataset)
```

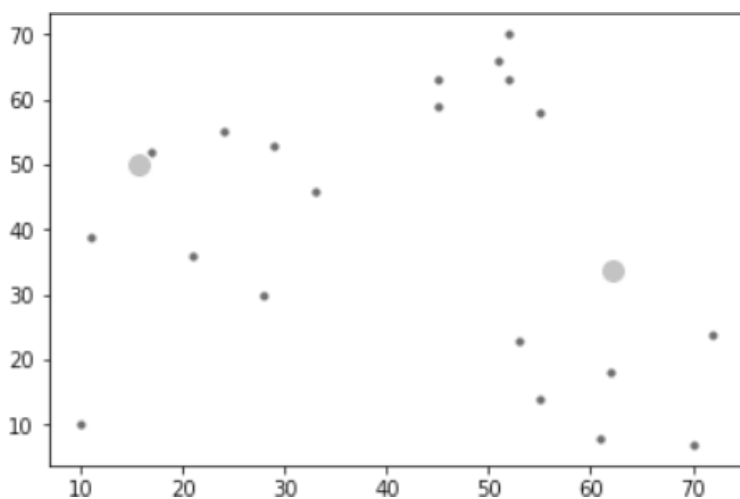
W następnej kolejności przyjrzymy się etykietom i środkom klastrów:

```
labels = labels = kmeans.labels_  
centers = kmeans.cluster_centers_  
print(labels)  
[0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0]  
print(centers)  
[[16.77777778 48.88888889]  
 [57.09090909 15.09090909]]
```

Na koniec, aby zwizualizować nasze klastry, narysujemy punkty danych, nadając im kolory zgodnie z klastrami, do których należą. Środki klastrów, zwane centroidami, także są narysowane:

```
plt.scatter(dataset['x'], dataset['y'], c=labels)  
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1],  
s=300, c='red')  
plt.show()
```

Niebieskie kropki na rysunku 6.7 oznaczają centroidy, a czerwone — punkty danych i ich klastry.



Rysunek 6.7. Wynik klastrowania *k*-średnich

Większe kropki na wykresie oznaczają środki (centroidy) klastrów wyznaczonych przez algorytm *k*-średnich.

Ograniczenia klasteryzacji *k*-średnich

Algorytm *k*-średnich został zaprojektowany z naciskiem na szybkość działania i prostotę. Z jego celowej prostoty przy projektowaniu wynikają następujące ograniczenia:

- Największe z nich jest takie, że początkowa liczba klastrów musi zostać z góry zdefiniowana i wskazana algorytmowi.
- Wyjściowe przypisanie środków klastrów jest losowe. Oznacza to, że przy każdym wykonaniu algorytmu klastry mogą się nieco różnić.

- Każda próbka danych jest przypisywana do jednego klastra.
- Klasteryzacja k -średnich zwraca zaburzone wyniki, gdy w zbiorze danych jest wiele wartości odstających.

Teraz przyjrzymy się technice nienadzorowanego uczenia maszynowego o nazwie grupowanie hierarchiczne.

Grupowanie hierarchiczne

Algorytm k -średnich przedstawia odgórne podejście do klasteryzacji, ponieważ rozpoczyna swoje działanie od najważniejszych próbek danych, tj. środków klastrów. Istnieje jednak także alternatywne podejście, zakładające, że algorytm rozpoczyna swoje działanie od dołu, który w tym kontekście oznacza pojedyncze próbki danych w dziedzinie problemu. Rozwiązanie to polega na grupowaniu podobnych próbek danych aż do dojścia do środków klastrów. Właśnie to oddolne podejście jest używane w algorytmach grupowania hierarchicznego, które przedstawię poniżej.

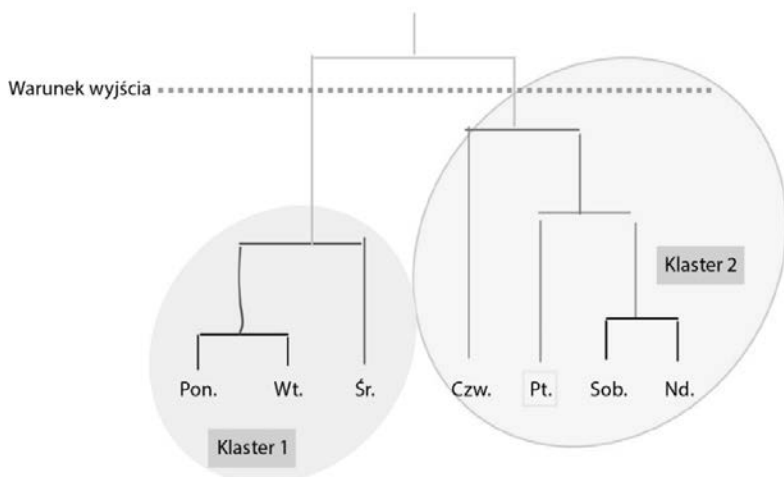
Kroki grupowania hierarchicznego

Grupowanie hierarchiczne obejmuje następujące kroki:

1. Tworzymy osobny klastery dla każdej próbki danych w dziedzinie problemu. Jeśli znajduje się w niej 100 punktów, algorytm rozpocznie pracę od 100 klastrów.
2. Grupujemy ze sobą tylko najbliższe sobie próbki.
3. Sprawdzamy warunek wyjścia, a jeśli nie jest on spełniony — powtarzamy krok 2.

Struktura klasteryzacji, jaka w ten sposób powstanie, nazywana jest **dendrogramem**.

W dendrogramie wysokość linii pionowych określa, jak bliskie sobie są próbki danych, co widać na rysunku 6.8.



Rysunek 6.8. Grupowanie hierarchiczne

Warunek wyjścia na rysunku 6.8 został zaznaczony przerywaną linią.

Implementacja grupowania hierarchicznego

Sprawdźmy teraz, jak będzie wyglądała implementacja grupowania hierarchicznego w Pythonie:

1. Zaczniemy od zaimportowania `AgglomerativeClustering` z biblioteki `sklearn.cluster` oraz pakietów `pandas` i `numpy`:

```
from sklearn.cluster import AgglomerativeClustering
import pandas as pd
import numpy as np
```
2. Następnie utworzymy 20 próbek danych w dwuwymiarowej przestrzeni:

```
dataset = pd.DataFrame({'x': [11, 11, 20, 12, 16, 33, 24, 14, 45, 52, 51, 52,
↪55, 53, 55, 61, 62, 70, 72, 10],
                        'y': [39, 36, 30, 52, 53, 46, 55,
↪59, 12, 15, 16, 18, 11, 23, 14, 8, 18, 7, 24, 70]})
```
3. Następnie stworzymy hierarchiczny klastrowanie poprzez podanie hiperparametrów. Hiperparametr to parametr konfiguracyjny modelu uczenia maszynowego, który zostaje ustawiony przed procesem szkolenia oraz ma wpływ na zachowanie i wydajność modelu. W celu uruchomienia algorytmu skorzystamy z funkcji `fit_predict`:

```
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean',
↪linkage='ward')
cluster.fit_predict(dataset)
```
4. Spójrzmy na przypisanie próbek danych do dwóch klastrow, jakie stworzyliśmy:

```
print(cluster.labels_)
[0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0]
```

Pewnie rzuci Ci się w oczy, że przypisanie wynikające z algorytmu k -średnich oraz to oparte na grupowaniu hierarchicznym są bardzo podobne.

Algorytm klastrowania hierarchicznego ma pewne zalety i wady w porównaniu z algorytmem klastrowania k -średnich. Jedną z najważniejszych zalet jest to, że klastrowanie hierarchiczne, w odróżnieniu od klastrowania k -średnich, nie wymaga określenia liczby klastrow z góry.

Cecha ta jest niezwykle przydatna, kiedy zbiór danych nie sugeruje wyraźnie optymalnej liczby klastrow. Ponadto algorytm klastrowania hierarchicznego generuje dendrogram, czyli przypominający drzewo schemat, który pozwala na wizualizację zagnieżdżonych grup danych i zrozumienie struktur hierarchicznych.

Klastrowanie hierarchiczne ma jednak również wady. Jest bardziej wymagające pod względem obliczeniowym niż algorytm k -średnich, przez co gorzej sprawdza się w przypadku dużych zbiorów danych.

Algorytm DBSCAN

DBSCAN (ang. *Density-based spatial clustering of applications with noise*) to nienadzorowana technika uczenia, w której klastrowanie jest wykonywane na podstawie gęstości punktów. Jej podstawowa idea opiera się na założeniu, że pogrupowanie punktów danych w wysoce zagęszczonej przestrzeni umożliwia osiągnięcie efektywnych klastrow.

To podejście do klastrowania pociąga za sobą dwie ważne implikacje:

- Algorytm prawdopodobnie umieści w grupach punkty, które mogą się różnić kształtem lub wzorem. Zatem metoda ta pomaga w tworzeniu klastrów dowolnego rodzaju kształtów. Pod słowem „kształt” w tym przypadku rozumiemy wzorec lub rozkład punktów danych w wielowymiarowej przestrzeni. Zaletą takiego postępowania jest to, że dane z rzeczywistego świata często są złożone i nieliniowe, a możliwość tworzenia klastrów obejmujących dowolne kształty umożliwia dokładniejsze reprezentowanie i zrozumienie takich danych.
- W odróżnieniu od algorytmu k -średnich w tym przypadku nie musimy określać liczby klastrów, ponieważ algorytm sam wykryje odpowiednią liczbę grup danych.

Kroki wykonywania algorytmu DBSCAN są następujące:

1. Algorytm określa sąsiedztwo każdego punktu danych. Słowo „sąsiedztwo” w tym kontekście oznacza obszar, w którym punkty danych są badane pod kątem odległości od interesującego nas punktu. Czynność ta jest zazwyczaj wykonywana przez policzenie punktów danych w pewnej odległości, która najczęściej jest reprezentowana przez zmienną ϵ . Określa ona maksymalną odległość, jaka może dzielić dwa punkty danych, aby zostały uznane za sąsiadujące. Standardowo dystans ten określa się za pomocą miary euklidesowej.
2. Następnie algorytm określa gęstość każdego punktu danych. Używa zmiennej o nazwie `min_samples`, która reprezentuje minimalną liczbę innych punktów danych, jaka powinna znajdować się w odległości ϵ od wybranego punktu, aby został uznany za „podstawowy egzemplarz”. Mówiąc prościej, podstawowy egzemplarz to punkt danych, który jest gęsto otoczony innymi punktami danych. Logiczne jest, że obszary o dużym zagęszczeniu punktów danych będą miały większą liczbę takich egzemplarzy.
3. Każde ze zidentyfikowanych sąsiedztw stanowi klastę. Należy pamiętać, że sąsiedztwo jednego egzemplarza podstawowego (punktu danych o minimalnej liczbie innych punktów danych w odległości ϵ) może obejmować inne egzemplarze podstawowe. To znaczy, że każdy egzemplarz podstawowy może być obecny w wielu klastrach, jeśli znajduje się wystarczająco blisko kilku punktów danych. W konsekwencji granice tych klastrów mogą się nakładać, co prowadzi do powstawania złożonych struktur wzajemnie połączonych klastrów.
4. Każdy punkt danych, który nie jest egzemplarzem podstawowym lub nie leży w sąsiedztwie egzemplarza podstawowego, jest uznawany za element odstający.

Teraz pokażę Ci, jak tworzyć klastry za pomocą algorytmu DBSCAN w Pythonie.

Tworzenie klastrow przy użyciu algorytmu DBSCAN w Pythonie

Najpierw zaimportujemy potrzebne funkcje z biblioteki sklearn:

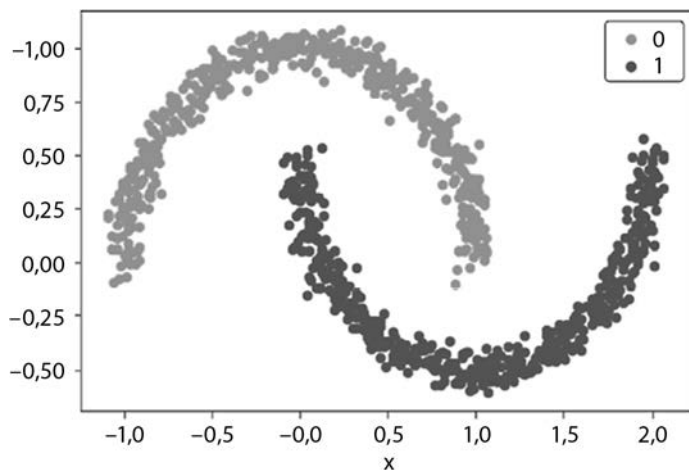
```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
```

Teraz użyjemy algorytmu DBSCAN do rozwiązania odrobinę bardziej skomplikowanego problemu klastrowania, który obejmuje struktury zwane „półksiężycami”. W tym kontekście „półksiężyc” to dwa zbiory punktów danych, które tworzą taki kształt i każdy z nich reprezentuje osobny klastrow. Takie zbiory danych stanowią wyzwanie, ponieważ klastrow tych nie da się rozdzielić liniowo, czyli nie da się w łatwy sposób wydzielić grup za pomocą prostej linii.

W tym momencie do gry wchodzi „nieliniowe granice klas”. W porównaniu z liniowymi granicami, które można reprezentować za pomocą prostej linii, granice nieliniowe są bardziej skomplikowane i często wymagają używania krzywych lub wielowymiarowych powierzchni, aby dokładnie posegregować różne klasy lub klastrow.

Aby wygenerować taki zbiór danych w kształcie półksiężyca, można użyć funkcji `make_moons()`, która tworzy wzór wiru przypominający dwa półksiężyce. Poziom „zaszumienia” tych zbiorów i liczbę próbek do wygenerowania można dowolnie dostosować.

Na rysunku 6.9 pokazane są wygenerowane zbiory danych, o których mowa.



Rysunek 6.9. Dane do użycia przez algorytm DBSCAN

Aby użyć algorytmu DBSCAN, musimy podać parametry `eps` i `min_samples`, o których była mowa wcześniej:

```
from matplotlib import pyplot
from pandas import DataFrame
# generuje dwuwymiarowy zbiór danych do klasyfikacji
X, y = make_moons (n_samples=1000, noise=0.05)
```



```
# wykres punktowy, kolory kropek oznaczają przynależność do klas
df = DataFrame (dict (x=X[,0], y=X[,1], label=y))
colors = {0: 'red', 1: 'blue'}
fig, ax = pyplot.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
pyplot.show()
```

Ocena klastrow

Celem poprawnej klasteryzacji jest taki podział próbek danych, by punkty należące do różnych klastrow można było od siebie odróżnić. Oznacza to, że:

- Próbkę danych należącą do tego samego klastra powinny być do siebie tak podobne, jak to tylko możliwe.
- Próbkę danych należącą do różnych klastrow powinny być od siebie tak odmienne, jak to tylko możliwe.

Możemy posłużyć się intuicją do oceny grupowań poprzez weryfikację wizualizacji klastrow, ale istnieją też metody matematyczne liczbowo ujmujące jakość grupowania. Nie tylko mierzą ścisłość każdego klastra (spójność) i stopień separacji między klastrami, ale dodatkowo zapewniają numeryczną (a więc obiektywną) metodę oceny jakości grupowania. Metoda sylwetki to jedna z technik porównywania ścisłości i rozdzielności klastrow stworzonych w wyniku działania algorytmu *k*-średnich. Jest to miara, która określa stopień spójności i rozdzielności w klastrach. Choć technika ta została opisana w kontekście *k*-średnich, to tak naprawdę można ją uogólnić i zastosować do oceny wyników dowolnego algorytmu klastrowania.

Analiza sylwetki każdemu punktowi danych przypisuje ocenę, zwaną współczynnikiem sylwetki, która mieści się w przedziale od 0 do 1. Określa ona odległość, w jakiej znajdują się punkty w jednym klastrze od punktów w klastrach sąsiednich.

Zastosowania klasteryzacji

Klasteryzację stosuje się wszędzie tam, gdzie w zbiorze danych chcemy odkryć mogące istnieć wzorce czy schematy.

Do zastosowań rządowych należą:

- **Analiza obszarów o podwyższonym wskaźniku przestępczości** — klastrowaniu poddaje się dane geolokalizacyjne, raporty powypadkowe itp. Technika ta pomaga w identyfikacji obszarów o wysokim wskaźniku przestępczości, co umożliwia organom państwa odpowiedzialnym za egzekwowanie prawa optymalizację tras patroli i skuteczniejsze wdrażanie zasobów.
- **Analiza społeczno-demograficzna** — dzięki klastrowaniu można analizować dane demograficzne, takie jak wiek, przychody, wykształcenie czy zawód. To pomaga w zrozumieniu sytuacji społeczno-ekonomicznej w różnych regionach, co można wykorzystać przy wprowadzaniu przepisów i realizacji usług w zakresie pomocy społecznej.

W badaniach rynkowych grupowania używa się do następujących celów:

- **Segmentacja rynku** — dzięki klasteryzacji danych klientów zawierających informacje o nawykach zakupowych, preferowanych produktach i wskaźników długości życia firmy mogą identyfikować segmenty rynku. Pozwala to na tworzenie produktów i strategii marketingowych dopasowanych do klienta.
- **Tworzenie celowanych reklam** — klastrowanie pomaga w analizowaniu zachowań online klientów, w tym ich zwyczajów przeglądania stron internetowych, współczynników kliknięć oraz historii zakupów. Umożliwia to tworzenie spersonalizowanych reklam dla każdej grupy klientów, wzmacnianie zaangażowania oraz zwiększanie wskaźników konwersji.
- **Klasyfikacja klientów** — za pomocą klastrowania firmy mogą kategoryzować klientów na podstawie ich interakcji z produktami lub usługami, ich komentarzy oraz ich lojalności. Ułatwia to zrozumienie ich zachowań, przewidywanie trendów oraz tworzenie strategii mających na celu zatrzymywanie klientów przy sobie.

W eksploracji danych i usuwaniu szumu z danych napływających w czasie rzeczywistym, takich jak notowania giełdowe, używana jest także **analiza głównych składowych** (ang. *principal component analysis* — PCA). W tym kontekście „szum” oznacza losowe lub nie-regularne fluktuacje, które mogą zasłaniać wzorce lub trendy obecne w danych. PCA pomaga w odfiltrowywaniu tych fluktuacji, umożliwiając analizę i interpolację czystszych danych.

Redukcja wymiarów

Każda cecha w naszych danych odpowiada wymiarowi w dziedzinie problemu. Ograniczanie liczby cech w dziedzinie problemu w celu jego uproszczenia nazywamy redukcją wymiarów. Można ją osiągnąć na dwa sposoby:

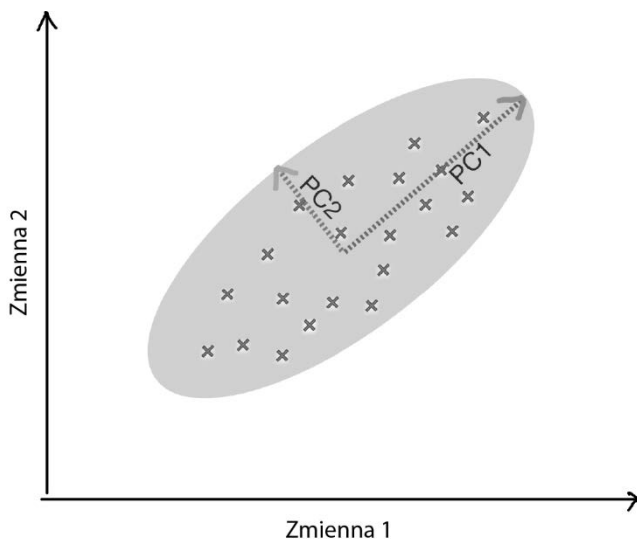
- **Wybór cech.** Wybranie zbioru cech, które są istotne w kontekście problemu, jaki staramy się rozwiązać.
- **Agregacja cech.** Łączenie dwóch lub większej liczby cech w celu zredukowania wymiarowości za pomocą algorytmów:
 - **analizy głównych składowych (PCA)** — liniowego algorytmu nienadzorowanego uczenia maszynowego,
 - **liniowej analizy dyskryminacyjnej (LDA)** — liniowego algorytmu nadzorowanego uczenia maszynowego,
 - **KPCA** — algorytmu nieliniowego.

Przyjrzyjmy się bliżej jednemu z popularnych algorytmów redukcji wymiarów, tj. analizie głównych składowych (PCA).

Analiza głównych składowych

PCA to metoda nienadzorowanego uczenia maszynowego, która jest zazwyczaj wykorzystywana do redukcji liczby wymiarów w zbiorach danych w procesie zwanym transformacją liniową. Mówiąc prościej, jest to sposób uproszczenia danych przez koncentrację na ich najważniejszych częściach, które identyfikuje się na podstawie ich wariancji.

Wyobraź sobie graficzną reprezentację zbioru danych, w której każdy punkt danych jest naniesiony w wielowymiarowej przestrzeni. PCA pomaga w identyfikacji podstawowych składników, które wskazują, gdzie dane są najbardziej zróżnicowane. Na rysunku 6.10 widać dwa takie składniki — PC1 i PC2. Ilustrują one ogólny „kształt” rozmieszczenia punktów danych.



Rysunek 6.10. Analiza składnika podstawowego

Każdy składnik podstawowy odpowiada nowemu, niższemu wymiarowi, który obejmuje jak najwięcej informacji. W praktycznym sensie te składniki główne można postrzegać jako wskaźniki podsumowujące oryginalnych danych, co ułatwia zapanowanie nad danymi i ich analizę. Na przykład w większym zbiorze danych dotyczących zachowań klientów PCA może pomóc w identyfikacji kluczowych czynników (składników podstawowych), które definiują większość zachowań klientów.

Wyznaczanie współczynników dla tych składników podstawowych obejmuje obliczanie wektorów i wartości własnych macierzy kowariancji danych, co jest tematem, którego dokładniejszy opis znajduje się nieco dalej. Współczynniki te służą jako wagi dla każdej oryginalnej cechy w nowej przestrzeni składników, określając wkład każdej cechy w składnik podstawowy.

Wyobraź sobie, że masz zbiór danych zawierający informacje o różnych aspektach gospodarki kraju, takich jak PKB, stopa zatrudnienia, inflacja itd. Dane te są bardzo obszerne i wielowymiarowe. W takim przypadku za pomocą PCA można by było zredukować te

wymiary do dwóch składników podstawowych — PC1 i PC2. Zawierałyby one najważniejsze informacje po odrzuceniu szumu i mniej istotnych szczegółów.

Powstały wykres, z osiami PC1 i PC2, stanowiłby łatwiejszą do interpretacji wizualną reprezentację danych, w której każdy punkt przedstawiałby stan gospodarki na podstawie kombinacji PKB, stopy zatrudnienia i innych czynników.

To czyni PCA cennym narzędziem do upraszczania i interpretacji danych o wielu wymiarach.

Spójrzmy na poniższy kod:

```
from sklearn.decomposition import PCA
import pandas as pd
url = "https://storage.googleapis.com/neurals/data/iris.csv"
iris = pd.read_csv(url)

iris
X = iris.drop('Species', axis=1)
pca = PCA(n_components=4)
pca.fit(X)
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
0 5.1 3.5 1.4 0.2 setosa
1 4.9 3.0 1.4 0.2 setosa
2 4.7 3.2 1.3 0.2 setosa
3 4.6 3.1 1.5 0.2 setosa
4 5.0 3.6 1.4 0.2 setosa
... ..
145 6.7 3.0 5.2 2.3 virginica
146 6.3 2.5 5.0 1.9 virginica
147 6.5 3.0 5.2 2.0 virginica
148 6.2 3.4 5.4 2.3 virginica
149 5.9 3.0 5.1 1.8 virginica
X = iris.drop('Species', axis=1)
pca = PCA(n_components=4)
pca.fit(X)
PCA(n_components=4)
```

Wyświetlmy teraz na ekranie wartości czynników w naszym modelu PCA (rysunek 6.11):

```
pca_df=(pd.DataFrame(pca.components_,columns=X.columns)) pca_df
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	
0	0.361387	-0.084523	0.856671	0.358289	← Czynniki dla składowej PC1
1	0.656589	0.730161	-0.173373	-0.075481	← Czynniki dla składowej PC2
2	-0.582030	0.597911	0.076236	0.545831	← Czynniki dla składowej PC3
3	-0.315487	0.319723	0.479839	-0.753657	← Czynniki dla składowej PC4

Rysunek 6.11. Diagram ukazujący współczynniki modelu PCA

Zauważ, że pierwotną ramkę danych opisywały cztery cechy: Sepal.Length, Sepal.Width, Petal.Length i Petal.Width. Widoczna powyżej ramka danych przedstawia czynniki dla wszystkich czterech głównych składowych: PC1, PC2, PC3, PC4 — przykładowo pierwszą wiersz wskazuje czynnik PC1, z którego można skorzystać, by zastąpić cztery oryginalne zmienne.

Należy zauważyć, że liczba składników podstawowych (w tym przypadku cztery: PC1, PC2, PC3 i PC4) nie musi wynosić dwa, jak w poprzednim przykładzie dotyczącym gośpodarki. Liczbę tę dobiera się na podstawie poziomu złożoności, na jakim chcemy pracować z naszymi danymi. Im więcej składników podstawowych wybierzemy, tym więcej oryginalnej różnorodności danych zachowamy kosztem zwiększenia złożoności.

W oparciu o te czynniki możemy wyliczyć główne składowe w wejściowych danych w ramce X:

```
X['PC1'] = X['Sepal.Length'] * pca_df['Sepal.Length'][0] + /
          X['Sepal.Width'] * pca_df['Sepal.Width'][0] + /
          X['Petal.Length'] * pca_df['Petal.Length'][0] + /
          X['Petal.Width'] * pca_df['Petal.Width'][0]

X['PC2'] = X['Sepal.Length'] * pca_df['Sepal.Length'][1] + /
          X['Sepal.Width'] * pca_df['Sepal.Width'][1] + /
          X['Petal.Length'] * pca_df['Petal.Length'][1] + /
          X['Petal.Width'] * pca_df['Petal.Width'][1]

X['PC3'] = X['Sepal.Length'] * pca_df['Sepal.Length'][2] + /
          X['Sepal.Width'] * pca_df['Sepal.Width'][2] + /
          X['Petal.Length'] * pca_df['Petal.Length'][2] + /
          X['Petal.Width'] * pca_df['Petal.Width'][2]

X['PC4'] = X['Sepal.Length'] * pca_df['Sepal.Length'][3] + /
          X['Sepal.Width'] * pca_df['Sepal.Width'][3] + /
          X['Petal.Length'] * pca_df['Petal.Length'][3] + /
          X['Petal.Width'] * pca_df['Petal.Width'][3]
```

Wyświetlmy teraz ramkę danych X po dokonaniu obliczeń głównych składowych (rysunek 6.12):

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	PC1	PC2	PC3	PC4
0	5.1	3.5	1.4	0.2	2.818240	5.646350	-0.659768	0.031089
1	4.9	3.0	1.4	0.2	2.788223	5.149951	-0.842317	-0.065675
2	4.7	3.2	1.3	0.2	2.613375	5.182003	-0.613952	0.013383
3	4.6	3.1	1.5	0.2	2.757022	5.008654	-0.600293	0.108928
4	5.0	3.6	1.4	0.2	2.773649	5.653707	-0.541773	0.094610
...	...	...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	7.446475	5.514485	-0.454028	-0.392844
146	6.3	2.5	5.0	1.9	7.029532	4.951636	-0.753751	-0.221016
147	6.5	3.0	5.2	2.0	7.266711	5.405811	-0.501371	-0.103650
148	6.2	3.4	5.4	2.3	7.403307	5.443581	0.091399	-0.011244
149	5.9	3.0	5.1	1.8	6.892554	5.044292	-0.268943	0.188390

Rysunek 6.12. Wydruk obliczeń składników podstawowych

A teraz spójrzmy na wartość wariancji i spróbujmy zrozumieć wnioski płynące z użycia analizy głównych składowych:

```
print(pca.explained_variance_ratio_)  
[0.92461872 0.05306648 0.01710261 0.00521218]
```

Taka macierz wariancji oznacza, że:

- Jeśli zdecydujemy się zastąpić cztery pierwotne cechy składową PC1, będziemy w stanie uchwycić ok. 92,3% wariancji oryginalnych zmiennych. Przez to, że nie uchwycimy 100% oryginalnych czterech cech, będziemy zdani na zastosowanie pewnych przybliżeń.
- Jeśli zdecydujemy się zastąpić cztery pierwotne cechy składowymi PC1 i PC2, uchwycimy dodatkowe 5,3% wariancji oryginalnych zmiennych.
- Jeśli zdecydujemy się zastąpić cztery pierwotne cechy składowymi PC1, PC2 i PC3, uchwycimy kolejne 0,017% wariancji oryginalnych zmiennych.
- Jeśli zdecydujemy się zastąpić cztery pierwotne cechy czterema głównymi składowymi, uchwycimy 100% wariancji oryginalnych zmiennych ($92,4 + 0,053 + 0,017 + 0,005$), ale zastępowanie czterech oryginalnych cech czterema głównymi składowymi jest bezcelowe, ponieważ w żaden sposób nie redukuje wymiarowości zbioru danych. Teraz przyjrzymy się ograniczeniom składników głównych.

Ograniczenia analizy głównych składowych

Mimo licznych zalet analiza głównych składowych ma też pewne ograniczenia opisane poniżej.

- Analiza głównych składowych jest najbardziej efektywna w odniesieniu do zmiennych ciągłych, ponieważ jej matematyczne podstawy są stworzone w celu pracy z danymi numerycznymi. Gorzej sprawdza się w przypadku zmiennych kategoriowych, które są powszechne w zbiorach danych zawierających takie atrybuty, jak płeć, narodowość czy typ produktu. Na przykład: gdybyśmy analizowali zbiór danych ankietowych zawierający odpowiedzi w postaci liczb (takich jak wiek lub dochód) i kategoriowe (na przykład wybrane preferencje lub opcje), analiza głównych składowych nie byłaby odpowiednim wyborem w przypadku danych kategoriowych.
- Ponadto działanie metody PCA bazuje na tworzeniu aproksymacji oryginalnych danych wielowymiarowych w przestrzeni o mniejszej liczbie wymiarów. Choć ta redukcja ułatwia pracę z danymi i ich przetwarzanie, odbywa się to kosztem utraty pewnej części informacji. Jest to kompromis, który należy dokładnie rozważyć w każdym przypadku. Na przykład: jeśli pracujesz na zbiorze danych biomedycznych, w którym każda cecha reprezentuje pewien marker genetyczny, zastosowanie metody PCA może wiązać się z ryzykiem utraty informacji, które mogą być potrzebne do zdiagnozowania lub wyleczenia jakiejś choroby.

Zatem, choć metoda PCA jest potężnym narzędziem do redukcji liczby wymiarów, zwłaszcza podczas pracy z dużymi zbiorami danych zawierającymi wiele wzajemnie powiązanych zmiennych liczbowych, należy uwzględnić jej ograniczenia, aby dokonać właściwego wyboru w danej sytuacji.

Reguły asocjacyjne

Wzorce w zbiorze danych to skarb, który należy odkryć, zrozumieć i wydobyć, by skorzystać z bogactwa informacji, jakie się w nim znajduje. Istnieje ważna grupa algorytmów, których celem jest analiza wzorców w konkretnym zbiorze danych. Jednym z bardziej popularnych przedstawicieli tej klasy jest algorytm **Apriori**, który umożliwia:

- zmierzenie częstotliwości występowania wzorca,
- ustalenie relacji *przyczynowo-skutkowej* pośród wzorców,
- wyliczenie przydatności wzorców poprzez porównanie ich dokładności z losowymi danymi.

Teraz przyjrzymy się paru przykładom zastosowania algorytmu Apriori.

Przykłady użycia

Z algorytmami reguł asocjacyjnych mamy często do czynienia wtedy, gdy staramy się określić związek przyczynowo-skutkowy pomiędzy różnymi zmiennymi w zbiorze danych. Pomagają one odpowiedzieć m.in. na poniższe pytania:

- Jakie wartości wilgotności powietrza, zachmurzenia i temperatury mogą prowadzić do opadów deszczu kolejnego dnia?
- Jak rodzaj szkody zgłoszony ubezpieczycielowi może wskazywać na oszustwo?
- Jakie połączenie leków może spowodować powikłania u pacjentów?

Jak wynika z tych przykładów, algorytm Apriori ma szeroki zakres zastosowań obejmujących wszystko: od analizy biznesowej po badania w dziedzinie służby zdrowia i ochrony środowiska. Algorytm ten jest potężnym narzędziem w zestawie analityka danych i umożliwia translację skomplikowanych wzorców na nadające się do wykorzystania informacje z różnych dziedzin.

Analiza koszykowa

Silniki poleceń, ważny temat, któremu w całości został poświęcony rozdział 12., są potężnym narzędziem do personalizacji wrażeń użytkowników. Istnieje jednak prostszy i efektywniejszy sposób generowania rekomendacji zwany analizą koszykową. W tego typu analizie dane dotyczą produktów, które są kupowane razem. W odróżnieniu od bardziej zaawansowanych silników poleceń ta metoda nie uwzględnia dodatkowych danych dotyczących użytkownika ani jego indywidualnych preferencji. Należy tu dokonać pewnego rozróżnienia. Silniki poleceń zazwyczaj tworzą spersonalizowane sugestie w oparciu o przeszłe zachowania użytkownika, jego preferencje i bogactwo innych informacji na jego temat. Natomiast analiza koszykowa skupia się wyłącznie na kombinacjach zakupionych produktów, bez względu na to, kto je kupił, oraz na to, jakie ta osoba ma preferencje.

Jedną z najważniejszych zalet analizy koszykowej jest względnie łatwe gromadzenie danych. Zbieranie pełnych danych na temat preferencji użytkownika może być skomplikowane i czasochłonne. Natomiast dane dotyczące produktów kupowanych razem często można w prosty sposób wydobyć z rekordów transakcyjnych, co czyni analizę koszykową wygodnym punktem startowym dla firmy do dalszych badań w dziedzinie rekomendacji.

Dla przykładu takie dane są generowane zawsze, gdy robimy zakupy w którymkolwiek hipermarkecie, a żeby je uzyskać, nie potrzeba żadnej specjalnej technologii.

Przez pojęcie specjalnych technologii rozumiemy dodatkowe kroki, takie jak przeprowadzanie ankiet wśród użytkowników, używanie śledzących plików cookie czy budowa złożonych potoków danych. W tym przypadku dane zbierają się same jako produkt uboczny procesu sprzedaży. Zgromadzone w ten sposób dane w pewnym okresie nazywają się **danymi transakcyjnymi**.

Gdy zastosujemy któryś z algorytmów reguł asocjacyjnych do danych transakcyjnych pochodzących z zakupowych koszyków w sklepach spożywczych, supermarketach czy sieciach fast food, mamy do czynienia z procesem analizy koszykowej. Mierzy on warunkowe prawdopodobieństwo kupienia pewnych produktów razem, co pomaga odpowiedzieć na pytania takie jak:

- Jakie jest optymalne rozłożenie produktów na sklepowych półkach?
- Jak należy prezentować produkty w katalogach sprzedażowych?
- Co należy polecać na podstawie zwyczajów zakupowych użytkownika?

Ponieważ analiza koszykowa pozwala oszacować, jak mają się względem siebie różne produkty, często korzysta się z niej w sprzedaży masowej, np. supermarketach, sklepach spożywczych, aptekach czy sieciach fast food. Jej wielką zaletą jest fakt, że wyniki takiej analizy tłumaczą się niemal same, więc nie sprawiają trudności użytkownikom biznesowym.

Przyjrzymy się typowemu supermarketowi. Wszystkie niepowtarzalne typy produktów, jakie są w nim dostępne, można przedstawić jako zbiór: $\pi = \{\text{produkt}_1, \text{produkt}_2, \dots, \text{produkt}_m\}$. Jeśli supermarket sprzedaje 500 różnych rodzajów produktów, zbiór π będzie miał wielkość 500.

Ludzie będą robić w tym sklepie zakupy. Za każdym razem, gdy ktoś wybierze produkt i za niego zapłaci przy kasie, produkt taki zostanie dodany do transakcji, tworząc zbiór towarów (*itemset*). Transakcje są grupowane w czasie, co przedstawia zbiór Δ , gdzie $\Delta = \{t_1, t_2, \dots, t_n\}$.

Spójrzmy na następujący prosty przykład danych transakcyjnych obejmujący zaledwie cztery transakcje. Podsumowałem je w poniższej tabeli:

t_1	Bramki, ochraniacze
t_2	Kij, bramka, ochraniacze, kask
t_3	Kask, piłka
t_4	Kij, ochraniacze, kask

Oto więcej szczegółów:

$\pi = \{\text{kij}, \text{bramka}, \text{ochraniacz}, \text{kask}, \text{piłka}\}$ to zbiór przedstawiający wszystkie rodzaje produktów dostępnych w sklepie.

Pochylmy się nad jedną z transakcji — t_3 — ze zbioru Δ . Produkty kupione w transakcji t_3 można przedstawić jako zbiór towarów $\text{itemset}(t_3) = \{\text{kask}, \text{piłka}\}$, co będzie świadczyło o tym, że klient nabył dwa produkty. Ten zbiór nazywa się zbiorem produktów,

ponieważ zawiera wszystkie produkty zakupione w jednej transakcji. Ponieważ w zbiorze towarów są dwa produkty, wielkość zbioru $\text{itemset}(t_3)$ będzie wynosiła dwa. Ta terminologia pozwala nam efektywniej klasyfikować i analizować wzorce.

Wyszukiwanie reguł asocjacyjnych

Reguła asocjacyjna opisuje matematycznie związek między elementami w jednej transakcji. Dzieje się to poprzez analizę związków pomiędzy dwoma zbiorami towarów, w formie $X \Rightarrow Y$, gdzie $X \subset \pi$, $Y \subset \pi$. Ponadto X i Y to zbiory niepokrywające się, co oznacza, że $X \cap Y = \emptyset$.

Regułę asocjacyjną można zapisać następująco:

$\{\text{kask, pi\laski}\} \Rightarrow \{\text{rower}\}$

W tym przykładzie $\{\text{kask, pi\laski}\}$ to X , a $\{\text{rower}\}$ to Y .

Przyjrzymy się różnym rodzajom reguł asocjacyjnych.

Rodzaje reguł

Wykonanie któregoś z algorytmów reguł asocjacyjnych zwykle wygeneruje wiele reguł dla danego zbioru danych transakcyjnych. Większość z nich będzie nic niewarta. Aby wybrać te, które dają użyteczną informację, klasyfikujemy reguły asocjacyjne jako:

- trywialne,
- niewytłumaczalne,
- użyteczne.

Przyjrzymy się każdemu z tych typów po kolei.

Reguły trywialne

Pośród ogromu wygenerowanych reguł wiele będzie bezużytecznych, ponieważ wyraża powszechną wiedzę. To właśnie takie reguły nazywamy trywialnymi. Nawet jeśli zaufanie do reguł trywialnych jest wysokie, nie skorzystamy na nich, ponieważ nie da się na ich podstawie uzyskać informacji wspierających proces decyzyjny. Pamiętaj, że „zaufanie” w tym przypadku odnosi się do miary używanej w związku z analizą, która określa prawdopodobieństwo wystąpienia określonego zdarzenia (powiedzmy B) pod warunkiem, że wystąpiło inne zdarzenie (A). Reguły trywialne można bezpiecznie i bez żalu pominąć.

Oto przykłady reguł trywialnych:

- Istnieje wysokie prawdopodobieństwo, że osoba skacząca z wieżowca zabije się.
- Więcej nauki prowadzi do lepszych rezultatów na egzaminach.
- Sprzedaż grzejników rośnie, gdy obniża się temperatura.
- Jazda z dużą prędkością na autostradzie zwiększa ryzyko spowodowania wypadku.

Reguły niewytłumaczalne

Spośród wszystkich wygenerowanych w wyniku działania algorytmu asocjacyjnego reguł te, które nie mają żadnego jasnego wytłumaczenia, są najtrudniejsze do użycia. Pamiętaj, że reguła będzie użyteczna tylko wtedy, gdy pozwala nam odkryć i zrozumieć nowy wzorzec, który może doprowadzić do pewnego działania. Jeśli tak nie jest i nie umiemy wyjaśnić, dlaczego zdarzenie X doprowadziło do Y , mamy do czynienia z regułą niewytłumaczalną, która stanowi jedynie matematyczny wzór opisujący bezcelową z naszego punktu widzenia relację dwóch zdarzeń, mogących w rzeczywistości być od siebie niezależnymi.

Oto przykłady reguł niewytłumaczalnych:

- Osoby w czerwonych koszulach uzyskują wyższe oceny na egzaminach.
- Zielone rowery są bardziej zagrożone kradzieżą.
- Ludzie, którzy kupują ogórki konserwowe, kupują także pieluchy.

Reguły użyteczne

Reguły użyteczne to skarb, którego szukamy. Osoby odpowiedzialne za biznes są w stanie je zrozumieć i na ich podstawie uzyskać wartościowe obserwacje. Pomagają one odkryć możliwe przyczyny pewnych zdarzeń, gdy korzysta z nich zespół osób posiadających wiedzę z danej dziedziny. Użyteczne reguły mogą na przykład wskazać najlepsze rozlokowanie produktów w sklepie na podstawie bieżących zachowań zakupowych klientów. Mogą one także zasugerować, które produkty warto umieścić razem, by zwiększyć ich szanse sprzedażowe, ponieważ klienci zwykle kupują je razem.

Oto przykłady reguł użytecznych i odpowiadających im działań:

- **Zasada 1.** Prezentowanie reklam użytkownikom mediów społecznościowych prowadzi do wyższego prawdopodobieństwa sprzedaży reklamowanych produktów.
Działanie: Wskazać alternatywne sposoby reklamowania produktu.
- **Zasada 2.** Stworzenie większej liczby punktów cenowych zwiększa szanse sprzedażowe.
Działanie: Jeden produkt można reklamować w promocyjnej cenie, podczas gdy cenę innego można podnieść.

Teraz przyjrzymy się, jak oceniać reguły.

Wskaźniki reguł

Jakość reguł asocjacyjnych ocenia się pod trzema względami:

- wsparcie,
- zaufanie,
- przyrost.

Warto wiedzieć, co się kryje za tymi hasłami.

Wsparcie

Wskaźnik wsparcia określa, jak często w zbiorze danych występuje wzorzec, którego szukamy. Oblicza się go, wyliczając liczbę wystąpień wzorca i dzieląc ją przez sumę wszystkich transakcji. W kontekście biznesowym te rzadkie przypadki mogłyby być wyjątkami lub elementami odstającymi, co może mieć poważne implikacje. Na przykład mogą one określać nietypowe zachowania klienta albo wyjątkowe trendy sprzedażowe, potencjalnie wskazując na okazje lub zagrożenia, które wymagają strategicznej uwagi.

Spojrzymy na następujący wzór dla konkretnego zbioru $itemset(a)$:

$liczbaItemset(a)$ — liczba transakcji obejmujących $itemset(a)$

$liczba_{suma}$ — łączna liczba transakcji

$$wsparcie(itemset(a)) = \frac{liczbaItemset(a)}{liczba_{suma}}$$

Wystarczy jedynie spojrzeć na wskaźnik wsparcia, by móc określić, jak rzadki jest dany wzorzec w zbiorze danych. Niska wartość wsparcia oznacza, że szukamy rzadko występującego zdarzenia. W kontekście biznesowym te rzadkie zdarzenia mogą być wyjątkowymi przypadkami lub elementami odstającymi, które mogą mieć istotne znaczenie. Na przykład mogą oznaczać nietypowe zachowania klientów albo wyjątkowe trendy sprzedażowe, wskazując na potencjalne okazje lub zagrożenia, które wymagają strategicznej uwagi.

Jeśli na przykład $itemset(a) = \{kask, piłka\}$ pojawia się w dwóch transakcjach z sześciu, wsparcie wynosi $(itemset(a)) = 2/6 = 0.33$.

Zaufanie

Zaufanie to wskaźnik wyrażający to, jak silnie możemy skorelować lewą stronę (X) z prawą (Y) za pomocą prawdopodobieństwa warunkowego. Obliczane jest prawdopodobieństwo, że zdarzenie X doprowadzi do zdarzenia Y , jeśli X nastąpi.

Matematycznie wzór wygląda następująco: $X \Rightarrow Y$.

Zaufanie do reguły przedstawia się jako $zaufanie(X \Rightarrow Y)$ i wylicza następująco:

$$zaufanie(X \Rightarrow Y) = \frac{wsparcie(X \cup Y)}{wsparcie(X)}$$

Przyjrzymy się przykładowi. Załóżmy, że zidentyfikowaliśmy następującą regułę:

$\{kask, piłka\} \Rightarrow \{bramki\}$

Zaufanie do tej zasady obliczymy za pomocą wzoru:

$$zaufanie(kask, piłka \Rightarrow bramki) = \frac{wsparcie(kask, piłka \cup bramki)}{wsparcie(kask, piłka)} = \frac{\frac{1}{6}}{\frac{2}{6}} = 0,5$$

Oznacza to, że jeśli ktoś ma w koszyku zbiór $\{kask, piłka\}$, to istnieje 0,5 czy też 50% prawdopodobieństwa, że znajdą się w nim także bramki.

Przyrost

Kolejnym sposobem oszacowania jakości reguły jest obliczenie jej przyrostu. Wskazuje on kierunek korelacji między poprzednikiem a następnikiem reguły asocjacyjnej. „Przyrost” oznacza stopień poprawy osiągnięty przez regułę w zakresie możliwości przewidywania wyniku w porównaniu z podejściem wzorcowym lub domyślnym. Reprezentuje zakres, w jakim reguła dostarcza dokładniejsze i bardziej przydatne przewidywania niż te, które można by było uzyskać przez uczynienie założeń wyłącznie na podstawie prawej strony równania. Gdyby zbiory towarów X i Y były od siebie niezależne, przyrost obliczyłobyśmy w ten sposób:

$$\text{przyrost}(X \Rightarrow Y) = \frac{\text{wsparcie}(X \cup Y)}{\text{wsparcie}(X) \cdot \text{wsparcie}(Y)}$$

Algorytmy analizy asocjacyjnej

Tę sekcję poświęcę dwóm algorytmom, które można stosować do analizy asocjacyjnej:

- **algorytmowi Apriori** przedstawionemu przez Agarwala i Srikanta w 1994 r.;
- **algorytmowi FP-Growth**, czyli ulepszeniu zaproponowanemu przez Hana i innych w 2001 r.

Każdemu z nich poświęcę trochę miejsca.

Algorytm Apriori

Algorytm Apriori to iteracyjny i wieloetapowy algorytm służący do generowania reguł asocjacyjnych. Opiera się on na podejściu generowania i testowania reguł.

Zanim uruchomimy ten algorytm, musimy zdefiniować dwie zmienne: $\text{support}_{\text{threshold}}$ i $\text{confidence}_{\text{threshold}}$.

Algorytm składa się z dwóch faz:

- **Etap generowania kandydatów.** Generowane są możliwe zbiory towarów, o wielkości odpowiadającej zadanemu parametrowi i wsparciu przekraczającemu wartość $\text{support}_{\text{threshold}}$.
- **Etap filtrowania.** Z rozwiązania usuwane są wszystkie reguły, których zaufanie nie przekracza wartości $\text{confidence}_{\text{threshold}}$.

Po tych dwóch krokach uzyskujemy końcowe rozwiązanie.

Ograniczenia algorytmu Apriori

Największą słabością algorytmu Apriori jest generowanie możliwych reguł na pierwszym etapie algorytmu. Przykładowo zbiór $\pi = \{\text{produkt}_1, \text{produkt}_2, \dots, \text{produkt}_m\}$ może wygenerować 2^m możliwych zbiorów danych. Ze względu na swoją wieloetapowość algorytm najpierw wygeneruje wszystkie te reguły, a następnie będzie szukał pośród nich zbiorów częstych. Ta wada algorytmu Apriori ma olbrzymi wpływ na jego wydajność i uniemożliwia jego stosowanie na większych zbiorach danych, ponieważ generuje on zbyt wiele zbiorów towarów, zanim znajdzie często powtarzające się produkty, co będzie miało wpływ na czas działania.

Teraz przyjrzymy się algorytmowi FP-Growth.

Algorytm FP-Growth

Algorytm FP-Growth (ang. *frequent pattern growth*) stanowi ulepszenie algorytmu Apriori. Rozpoczyna się on od kompresji zbioru danych do tzw. FP-drzewa, które jest drzewem uporządkowanym, a następnie drzewo to poddawane jest analizie w poszukiwaniu zbiorów częstych. Jak widać, algorytm ten obejmuje dwa kroki:

- wypełnienie FP-drzewa danymi,
- eksplorację drzewa w poszukiwaniu częstych zbiorów.

Każdy z tych kroków domaga się omówienia.

Wypełnienie FP-drzewa danymi

Spójrzmy na dane transakcyjne zaprezentowane w poniższej tabeli. Najpierw przedstawimy je w macierzy rzadkiej.

ID	Kij	Bramki	Ochraniacz	Kask	Piłka
1	0	1	1	0	0
2	1	1	1	1	0
3	0	0	0	1	1
4	1	0	1	1	0

Obliczmy teraz, jak często w transakcjach pojawia się każdy z produktów, a następnie posortujmy wyniki malejąco:

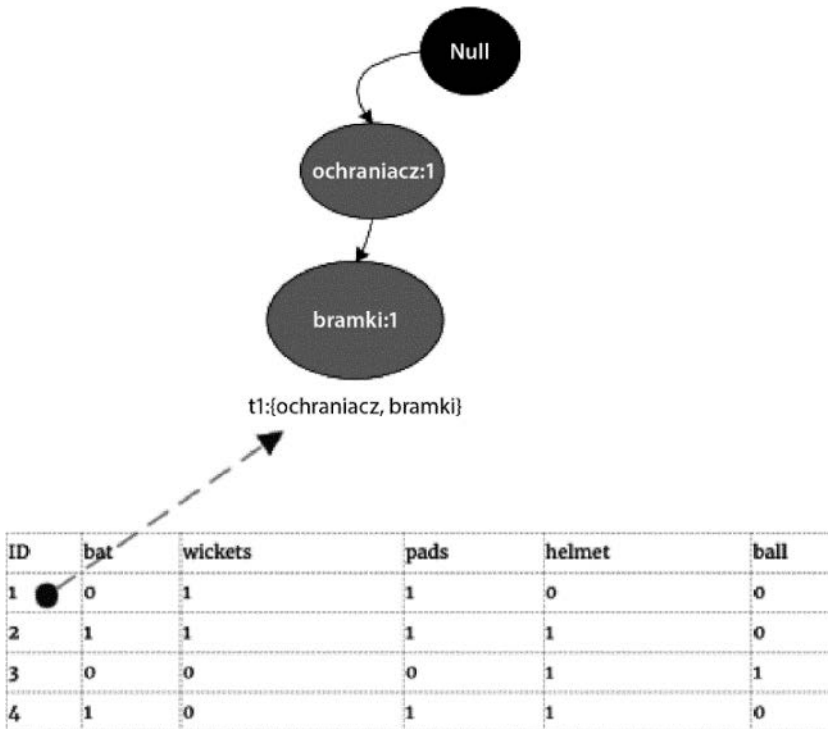
Produkt	Wsparcie
Ochraniacze	3
Kask	3
Kij	2
Bramki	2
Piłka	1

Przejdźmy teraz do uporządkowania danych w transakcjach zgodnie z powyższymi wartościami wsparcia:

ID	Wyjściowa kolejność produktów	Uporządkowana kolejność produktów
t1	Bramki, ochraniacz	Ochraniacz, bramki
t2	Kij, bramki, ochraniacz, kask	Kask, ochraniacz, bramki, kij
t3	Kask, piłka	Kask, piłka
t4	Kij, ochraniacz, kask	Kask, ochraniacz, kij

Aby zbudować FP-drzewo, zacznijmy od jego pierwszej gałęzi. Rozpoczyna się ono od wartości *Null* jako korzenia. Przy tworzeniu drzewa każdy produkt przedstawiamy jako węzeł, co widać na poniższym diagramie (reprezentacja transakcji *t1* w postaci drzewa).

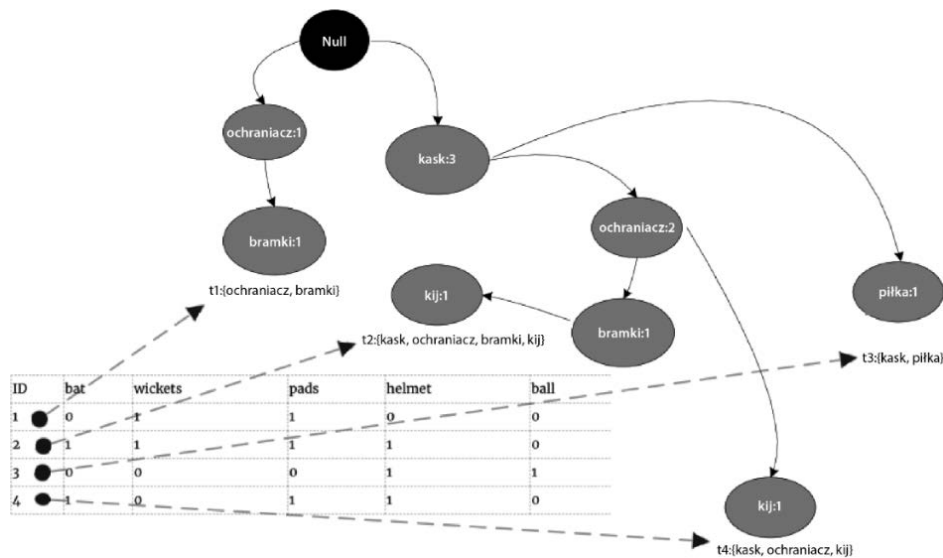
Zwróć uwagę, że etykietą każdego węzła jest nazwa produktu i wskaźnik wsparcia dodany po dwukropku — ochraniacze mają na tym etapie częstotliwość 1 (rysunek 6.13).



Rysunek 6.13. Reprezentacja pierwszej transakcji w postaci FP-drzewa

Korzystając z tego samego schematu, rozrysujmy wszystkie cztery transakcje, tworząc pełne FP-drzewo. Ma ono cztery liście, które kończą reprezentacje czterech zbiorów towarów w transakcjach. Pamiętaj, że musimy liczyć wsparcie każdego produktu i zwiększać jego wartość, gdy występuje on wielokrotnie — np. gdy dodamy do drzewa transakcję *t2*, wsparcie produktu „kask” urośnie do dwóch. Podobnie przy dodawaniu transakcji *t3* — wzrośnie do trzech. Końcowe, pełne drzewo widać na rysunku 6.14.

Zauważ, że widoczne na powyższym diagramie FP-drzewo jest drzewem uporządkowanym. To prowadzi nas do drugiej fazy drzewa algorytmu FP-Growth — eksploracji częstych wzorców.



Rysunek 6.14. FP-drzewo reprezentujące wszystkie transakcje

Eksploracja drzewa w poszukiwaniu częstych wzorców

Drugi etap algorytmu FP-Growth polega na wydobyciu częstych wzorców z FP-drzewa. Utworzenie uporządkowanego drzewa jest celowym ruchem mającym na celu zbudowanie struktury danych pozwalającej na łatwe nawigowanie w poszukiwaniu tych często występujących wzorców.

Zaczynamy od liści (czyli końcowych węzłów) i poruszamy się w górę — dla przykładu zaczniemy od węzła z produktem „kij”. Następnie musimy obliczyć warunkowy wzorec podstawowy dla tego produktu. Termin „warunkowy wzorec podstawowy” może brzmieć skomplikowanie, ale jest to jedynie kolekcja ścieżek, które prowadzą od określonego liścia do korzenia drzewa. Dla naszego produktu „kij” podstawowy wzorec warunkowy będzie obejmował wszystkie ścieżki od węzła „kij” do początku drzewa. W tym momencie krytycznego znaczenia nabiera znajomość różnicy między drzewem uporządkowanym i nieuporządkowanym. W drzewie uporządkowanym, takim jak FP-drzewo, elementy są rozmieszczone według ustalonego porządku, co upraszcza proces eksploracji w poszukiwaniu wzorców. Drzewo nieuporządkowane nie ma takiej ustrukturyzowanej konstrukcji, przez co wykrywanie częstych wzorców może być trudniejsze.

Podczas obliczania warunkowego wzorca bazowego dla „kijów” w istocie mapujemy wszystkie ścieżki z węzła „kije” do korzenia. Ścieżki te ujawniają elementy, które często występują razem z produktem „kij” w transakcjach. Zasadniczo podążamy „gałęzią” drzewa powiązaną z „kijem”, aby zrozumieć jej relacje z innymi produktami. Ilustracja graficzna wyjaśnia, skąd bierzemy te informacje oraz jak FP-drzewo pomaga w wykryciu częstych wzorców w danych transakcyjnych. Wzorec warunkowy dla kija będzie wyglądał następująco:

Bramki:1	Ochraniacz:1	Kask:1
Ochraniacz:1	Kask:1	

W przypadku kija zbiory częste będą wyglądały tak:

```
{bramki, ochraniacz, kask}: kij
{ochraniacz, kask}: kij
```

Implementacja algorytmu FP-Growth

Zobaczmy, jak możemy wygenerować reguły asocjacyjne za pomocą algorytmu FP-Growth zaimplementowanego w Pythonie. W tym celu skorzystamy z pakietu pyfpgrowth. Ponieważ wcześniej z niego nie korzystaliśmy, musimy go najpierw zainstalować:

```
!pip install pyfpgrowth
```

Następnie zaimportujemy pakiety, które będą potrzebne do implementacji algorytmu:

```
import pandas as pd
import numpy as np
import pyfpgrowth as fp
```

Teraz stworzymy dane wejściowe przechowywane w zmiennej transactionSet:

```
dict1 = {'id': [0,1,2,3],
        'items': [['bramki", "ochraniacze"],
                  ["kij", "bramki", "ochraniacze", "kask"],
                  ["kask", "ochraniacze"],
                  ["kij", "ochraniacze", "kask"]]}
transactionSet = pd.DataFrame(dict1)
id items
0 0 [bramki, ochraniacze]
1 1 [kij, bramki, ochraniacze, kask]
2 2 [kask, ochraniacze]
3 3 [kij, ochraniacze, kask]
```

Gdy mamy już dane wejściowe, wygenerujemy wzorce, które będą opierały się na parametrach, jakie przekazemy funkcji `find_frequent_patterns()`. Zauważ, że drugi parametr przekazywany tej funkcji określa minimalne wsparcie, które w tym przypadku wynosi 1:

```
patterns = fp.find_frequent_patterns(transactionSet['items'],1)
```

Na tym etapie wzorce zostały już wygenerowane. Wyświetlmy je na ekranie. Obejmują one połączenie produktu i jego wsparcia:

```
patterns
{('ochraniacz',): 1,
 ('kask', 'ochraniacz'): 1,
 ('bramki',): 2,
 ('ochraniacz', 'bramki'): 2,
 ('kij', 'bramki'): 1,
 ('kask', 'bramki'): 1,
 ('kij', 'ochraniacz', 'bramki'): 1,
 ('kask', 'ochraniacz', 'bramki'): 1,
 ('kij', 'kask', 'bramki'): 1,
 ('kij', 'kask', 'ochraniacz', 'bramki'): 1,
 ('kij',): 2,
 ('kij', 'kask'): 2,
 ('kij', 'ochraniacz'): 2,
 ('kij', 'kask', 'ochraniacz'): 2,
```



```
('ochraniacze',): 3,  
('kask',): 3,  
('kask', 'ochraniacze'): 2}
```

A teraz pora wygenerować reguły:

```
rules = fp.generate_association_rules(patterns,0.3)  
rules  
{('kask',): (('ochraniacze',), 0.6666666666666666),  
 ('pad',): (('kask',), 1.0),  
 ('ochraniacze',): (('kask',), 0.6666666666666666),  
 ('bramki',): (('kij', 'kask', 'ochraniacze'), 0.5),  
 ('kij',): (('kask', 'ochraniacze'), 1.0),  
 ('kij', 'ochraniacze'): (('kask',), 1.0),  
 ('kij', 'bramki'): (('kask', 'ochraniacze'), 1.0),  
 ('ochraniacze', 'bramki'): (('kij', 'kask'), 0.5),  
 ('kask', 'ochraniacze'): (('kij',), 1.0),  
 ('kask', 'bramki'): (('kij', 'ochraniacze'), 1.0),  
 ('kij', 'kask'): (('ochraniacze',), 1.0),  
 ('kij', 'kask', 'ochraniacze'): (('bramki',), 0.5),  
 ('kij', 'kask', 'bramki'): (('ochraniacze',), 1.0),  
 ('kij', 'ochraniacze', 'bramki'): (('kask',), 1.0),  
 ('kask', 'ochraniacze', 'bramki'): (('kij',), 1.0)}
```

Każda z nich ma stronę lewą i prawą, które oddziela dwukropek (:). Podane zostało też wsparcie każdej reguły w zbiorze danych wejściowych.

Podsumowanie

W tym rozdziale przyglądaliśmy się różnym technikom nienadzorowanego uczenia maszynowego. Wskazaliśmy okoliczności, w których rozsądnie jest próbować redukować wymiarowość problemu. Zaprezentowaliśmy służące do tego metody. Przedstawiliśmy też praktyczne przykłady, w których nienadzorowane uczenie maszynowe może być niezwykle pomocne, m.in. w analizie koszykowej czy wykrywaniu odchyleń.

W kolejnym rozdziale zajmę się technikami uczenia nadzorowanego. Zaczniemy od regresji liniowej i z czasem przejdziemy do bardziej zaawansowanych technik nadzorowanego uczenia maszynowego, takich jak algorytmy oparte na drzewie decyzyjnym, maszyna wektorów nośnych czy algorytm wzmacniania gradientowego. Zajmiemy się także naiwnym klasyfikatorem bayesowskim, który dobrze radzi sobie z nieustrukturyzowanymi danymi tekstowymi.

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

Algorytmy: klucz do sukcesu w programowaniu!

Wiedza o algorytmach jest niezbędna przy rozwiązywaniu problemów programistycznych i prowadzeniu złożonych obliczeń. Każdy programista powinien dobrze znać algorytmy, musi też umieć je zaprojektować, modyfikować i stosować. Niezależnie od tego, czy zajmujesz się uczeniem maszynowym, kwestiami bezpieczeństwa, czy inżynierią danych, rzetelne zrozumienie algorytmów jest Ci bardzo potrzebne.

Dzięki tej książce nauczysz się stosować algorytmy w praktycznych sytuacjach i zrozumiesz mechanizmy ich działania. Liczne przykłady pozwolą Ci się zapoznać z kilkoma sposobami ich projektowania i implementacji. Następnie poznasz algorytm określania pozycji stron w wynikach wyszukiwarek internetowych, związane z nimi grafy i algorytmy uczenia maszynowego, a także logikę. Zaznajomisz się ponadto z nowoczesnymi modelami sekwencyjnymi i ich wariantami, jak również algorytmami, metodami i architekturami implementacji dużych modeli językowych, takich jak ChatGPT. W ostatniej części tego przewodnika znajdziesz opis technik przetwarzania równoległego, przydatnego w zadaniach wymagających dużej mocy obliczeniowej.

W książce między innymi:

- projektowanie algorytmów przeznaczonych do złożonych zadań
- sieci neuronowe i techniki uczenia głębokiego
- struktury danych i algorytmy dostępne w bibliotekach Pythona
- algorytm grafowy służący do wykrywania oszustw za pomocą analizy sieciowej
- najnowocześniejsze algorytmy przetwarzania języka naturalnego
- tworzenie systemu rekomendacji filmów
- sekwencyjne modele uczenia maszynowego i nowoczesne modele LLM

Dr Imran Ahmad

jest ekspertem naukowym do spraw obróbki danych w Advanced Analytics Solution Center, jednostce Rządu Federalnego Kanady, gdzie zajmuje się algorytmami uczenia maszynowego w aplikacjach o krytycznym znaczeniu. Jest również wykładowcą w Carleton University w Ottawie, a od kilku lat naucza technologii Google Cloud i AWS.

Helion	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	ISBN 978-83-289-1107-9	
 HELION S.A. ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	 9 788328 911079	
Cena: 89,00 zł		

<packt>