

Alberto Boschetti, Luca Massaron

Python

Podstawy nauki o danych

Wydanie II

Helion 

Packt 

Tytuł oryginału: Python Data Science Essentials, Second Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-3423-6

Copyright © Packt Publishing 2016

First published in the English language under the title
'Python Data Science Essentials – Second Edition – (9781786462138)'

Polish edition copyright © 2017 by Helion SA. All rights reserved.

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 jakiejkolwiek 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 Wydawnictwo HELION 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 Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/pypod2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorach	9
O recenzencie	10
Wprowadzenie	11
Rozdział 1. Pierwsze kroki	15
Wprowadzenie do nauki o danych i Pythona	16
Instalowanie Pythona	17
Python 2 czy Python 3?	18
Instalacja krok po kroku	19
Instalowanie pakietów	20
Aktualizowanie pakietów	22
Dystrybucje naukowe	22
Środowiska wirtualne	25
Krótki przegląd podstawowych pakietów	28
Wprowadzenie do środowiska Jupyter	37
Szybka instalacja i pierwsze testowe zastosowanie	41
Magiczne polecenia w Jupyterze	42
W jaki sposób notatniki Jupytera mogą być pomocne dla badaczy danych?	44
Zastępniki Jupytera	49
Zbiory danych i kod używane w książce	50
Proste przykładowe zbiory danych z pakietu scikit-learn	50
Podsumowanie	59

Rozdział 2. Przekształcanie danych	61
Proces pracy w nauce o danych	62
Wczytywanie i wstępne przetwarzanie danych za pomocą biblioteki pandas	64
Szybkie i łatwe wczytywanie danych	64
Radzenie sobie z problematycznymi danymi	67
Radzenie sobie z dużymi zbiorami danych	70
Dostęp do danych w innych formatach	73
Wstępne przetwarzanie danych	75
Wybieranie danych	78
Praca z danymi kategorialnymi i tekstowymi	81
Specjalny rodzaj danych — tekst	83
Scraping stron internetowych za pomocą pakietu Beautiful Soup	89
Przetwarzanie danych za pomocą pakietu NumPy	92
N-wymiarowe tablice z pakietu NumPy	92
Podstawowe informacje o obiektach ndarray z pakietu NumPy	93
Tworzenie tablic z pakietu NumPy	95
Przekształcanie list w jednowymiarowe tablice	95
Kontrolowanie ilości zajmowanej pamięci	96
Listy niejednorodne	98
Od list do tablic wielowymiarowych	99
Zmiana wielkości tablic	100
Tablice generowane przez funkcje z pakietu NumPy	101
Pobieranie tablicy bezpośrednio z pliku	102
Pobieranie danych ze struktur z biblioteki pandas	103
Szybkie operacje i obliczenia z użyciem pakietu NumPy	104
Operacje na macierzach	106
Tworzenie wycinków i indeksowanie tablic z pakietu NumPy	108
Dodawanie „warstw” tablic z pakietu NumPy	110
Podsumowanie	112
Rozdział 3. Potok danych	113
Wprowadzenie do eksploracji danych	113
Tworzenie nowych cech	117
Redukcja liczby wymiarów	120
Macierz kowariancji	120
Analiza głównych składowych	121
Analiza głównych składowych dla big data — typ RandomizedPCA	125
Analiza czynników ukrytych	126
Liniowa analiza dyskryminacyjna	127
Analiza ukrytych grup semantycznych	128
Analiza składowych niezależnych	129
Analiza głównych składowych oparta na funkcji jądra	129
Algorytm t-SNE	131
Ograniczone maszyny Boltzmanna	132

Wykrywanie i traktowanie wartości odstających	133
Wykrywanie obserwacji odstających za pomocą technik jednoczynnikowych	134
Klasa EllipticEnvelope	136
Klasa OneClassSVM	140
Miary używane do walidacji	144
Klasyfikacja wieloklasowa	144
Klasyfikacja binarna	147
Regresja	148
Testy i walidacja	148
Walidacja krzyżowa	153
Iteratory walidacji krzyżowej	155
Próbkowanie i bootstrapping	157
Optymalizacja hiperparametrów	159
Tworzenie niestandardowych funkcji oceny	162
Skracanie czasu przeszukiwania siatki parametrów	164
Wybór cech	166
Wybór na podstawie wariancji cech	167
Wybór za pomocą modelu jednoczynnikowego	168
Rekurencyjna eliminacja	169
Wybór na podstawie stabilności i regularyzacji L1	171
Opakowywanie wszystkich operacji w potok	173
Łączenie cech i tworzenie łańcuchów transformacji	174
Tworzenie niestandardowych funkcji transformacji	176
Podsumowanie	177
Rozdział 4. Uczenie maszynowe	179
Przygotowywanie narzędzi i zbiorów danych	179
Regresja liniowa i logistyczna	181
Naiwny klasyfikator bayesowski	184
Algorytm kNN	187
Algorytmy nieliniowe	188
Stosowanie algorytmu SVM do klasyfikowania	190
Stosowanie algorytmów SVM do regresji	192
Dostrajanie algorytmu SVM	193
Strategie oparte na zestawach algorytmów	195
Pasting z użyciem losowych próbek	196
Bagging z użyciem słabych klasyfikatorów	196
Podprzestrzenie losowe i obszary losowe	197
Algorytmy Random Forests i Extra-Trees	198
Szacowanie prawdopodobieństwa na podstawie zestawów	200
Sekwencje modeli — AdaBoost	202
Metoda GTB	202
XGBoost	203

Przetwarzanie big data	206
Tworzenie przykładowych dużych zbiorów danych	207
Skalowalność ze względu na ilość danych	208
Radzenie sobie z szybkością napływu danych	210
Radzenie sobie z różnorodnością	211
Przegląd algorytmów z rodziny SGD	213
Wprowadzenie do uczenia głębokiego	214
Krótkie omówienie przetwarzania języka naturalnego	221
Podział na tokeny	221
Stemming	222
Oznaczanie części mowy	223
Rozpoznawanie nazw własnych	224
Stop-słowa	225
Kompletny przykład z obszaru nauki o danych — klasyfikowanie tekstu	225
Przegląd technik uczenia nienadzorowanego	227
Podsumowanie	237
Rozdział 5. Analizy sieci społecznościowych	239
Wprowadzenie do teorii grafów	239
Algorytmy dla grafów	244
Wczytywanie grafów, zapisywanie ich w pliku i pobieranie z nich podpróbek	252
Podsumowanie	255
Rozdział 6. Wizualizacje, wnioski i wyniki	257
Wprowadzenie do pakietu Matplotlib	257
Rysowanie krzywych	259
Stosowanie paneli	260
Wykresy punktowe określające relacje w danych	262
Histogramy	263
Wykresy słupkowe	264
Wyświetlanie rysunków	265
Wybrane przykłady graficzne z użyciem pakietu pandas	268
Wykresy punktowe	271
Metoda współrzędnych równoległych	273
Opakowywanie poleceń z pakietu Matplotlib	274
Wprowadzenie do biblioteki seaborn	274
Wzbogacanie możliwości z zakresu eksploracji danych	279
Interaktywne wizualizacje z użyciem pakietu Bokeh	284
Zaawansowane reprezentacje dotyczące uczenia się na podstawie danych	288
Krzywe uczenia	288
Krzywe walidacji	290
Znaczenie cech w algorytmie Random Forests	292
Wykresy częściowej zależności oparte na drzewach GBT	293
Budowanie serwera predykcji w modelu ML-AAS	294
Podsumowanie	299

Dodatek A. Utrwalanie podstaw Pythona	301
Lista zagadnień do nauki	302
Listy	302
Słowniki	304
Definiowanie funkcji	305
Klasy, obiekty i programowanie obiektowe	307
Wyjątki	308
Iteratory i generatory	309
Instrukcje warunkowe	310
Wyrażenia listowe i słownikowe	311
Nauka przez obserwację, lekturę i praktykę	311
Masowe otwarte kursy online	311
PyCon i PyData	312
Interaktywne sesje w Jupyterze	312
Nie wstydź się — podejmij wyzwanie	312
Skorowidz	315

Zespół wydania oryginalnego

Autorzy

Alberto Boschetti
Luca Massaron

Recenzent

Zacharias Voulgaris

Redaktor zlecający

Veena Pagare

Redaktor zamawiający

Namrata Patil

Redaktor ds. treści

Mayur Pawanikar

Redaktor techniczny

Vivek Arora

Redaktor

Vikrant Phadke

Koordynator projektu

Nidhi Joshi

Korektor

Safis Editing

Indeksowanie

Aishwarya Gangawane

Grafik

Disha Haria

Koordynator prac produkcyjnych

Arvindkumar Gupta

Przekształcanie danych

Zabieramy się do pracy z danymi! W tym rozdziale dowiesz się, jak przekształcać (ang. *munge*) dane. Na czym polega ten proces?

Angielskie słowo *munge* to techniczne pojęcie wymyślone mniej więcej pół wieku temu przez studentów uczelni MIT (ang. *Massachusetts Institute of Technology*). Przekształcanie polega na modyfikowaniu w serii dobrze określonych i odwracalnych kroków fragmentu pierwotnych danych na inne (i, miejmy nadzieję, przydatniejsze) dane. Źródłem tego pojęcia jest kultura hakerska; w kontekście nauki o danych często stosowane są inne (prawie synonimiczne) pojęcia, takie jak **data wrangling** lub przygotowywanie danych. Jest to bardzo ważna część potoku operacji w inżynierii danych.

Od tego rozdziału będziemy posługiwać się żargonowymi i technicznymi określeniami z obszaru rachunku prawdopodobieństwa i statystyki (takimi jak rozkład prawdopodobieństwa, statystyki opisowe lub testowanie hipotez). Niestety nie możemy szczegółowo objaśnić wszystkich tych pojęć, ponieważ naszym głównym celem jest zaprezentowanie podstawowych aspektów Pythona związanych z projektami z dziedziny nauki o danych. Dlatego zakładamy, że część tych określeń już znasz. Jeśli chcesz przypomnieć sobie lub nawet zapoznać się z wprowadzeniem do omawianych zagadnień, zachęcamy do zaznajomienia się z prowadzonym przez Ramesha Sridharana kursem skierowanym do początkujących statystyków i badaczy z obszaru nauk społecznych. Materiały do kursu znajdziesz na stronie <http://www.mit.edu/~6.s085/>.

Po wyjaśnieniu założeń pora przedstawić tematy omawiane w tym rozdziale:

- proces pracy w nauce o danych (dzięki niemu będziesz wiedział, co się dzieje i co czeka Cię w następnych krokach),
- wczytywanie danych z pliku,
- wybieranie potrzebnych danych,
- radzenie sobie z brakującymi lub błędnymi danymi,

- dodawanie, wstawianie i usuwanie danych,
- grupowanie i modyfikowanie danych w celu uzyskania nowych i znaczących informacji,
- tworzenie macierzy lub tablicy ze zbiorem danych przekazywanej do części potoku odpowiedzialnej za generowanie modelu.

Proces pracy w nauce o danych

Choć każdy projekt w obszarze nauki o danych jest inny, na potrzeby tego omówienia możemy podzielić idealny projekt na serię uproszczonych etapów.

Cały proces rozpoczyna się od uzyskania danych (jest to etap pobierania lub pozyskiwania danych). Dostępne są tu różne możliwości: od prostego pobrania danych, przez scalanie ich za pomocą systemów RDBMS lub repozytoriów NoSQL, po generowanie syntetycznych danych lub **scraping** danych z sieciowych interfejsów API lub stron HTML.

Zwłaszcza gdy zajmujesz się nowymi wyzwaniami, wczytywanie danych może okazać się krytycznym aspektem pracy badacza danych. Dane mogą pochodzić z wielu źródeł: baz danych, plików CSV, plików Excela, kodu w HTML-u, rysunków, nagrań dźwiękowych, interfejsów API (https://pl.wikipedia.org/wiki/Application_Programming_Interface) zwracających pliki w formacie JSON itd. Ponieważ możliwości jest tak wiele, tylko pokrótce omawiamy ten aspekt pracy i przedstawiamy podstawowe narzędzia do pobierania danych (nawet jeśli jest ich za dużo) do pamięci komputera za pomocą plików tekstowych dostępnych na twardym dysku, w sieci WWW lub w tabelach systemu RDBMS.

Po udanym wczytaniu danych następuje etap ich przekształcania. Choć dane są już dostępne w pamięci, ich postać z pewnością jest niedostosowana do analiz i eksperymentów. Dane w rzeczywistym świecie są złożone, nieuporządkowane, a często nawet błędne. Nieraz w danych występują braki. Jednak dzięki zestawowi podstawowych struktur danych i instrukcji z Pythona poradzisz sobie z wszystkimi problematycznymi danymi i będziesz mógł przekazać je do następnych etapów projektu. Przekształcone dane mają postać typowego zbioru danych z obserwacjami w wierszach i zmiennymi w kolumnach. Uzyskanie zbioru danych to podstawowy wymóg we wszelkich analizach statystycznych i z obszaru uczenia maszynowego. Takie zbiory danych są nazywane plikami płaskimi (gdy powstały w wyniku scalenia wielu relacyjnych tabel z bazy danych) lub macierzami danych (gdy kolumny i wiersze nie są opisane, a przechowywane wartości to same liczby).

Etap przekształcania danych daje wprawdzie mniej satysfakcji niż inne, bardziej intelektualnie stymulujące fazy (takie jak stosowanie algorytmów lub uczenie maszynowe), zapewnia jednak podstawy dla wszelkich złożonych i zaawansowanych analiz pozwalających uzyskać wartość dodaną. Powodzenie projektu w dużym stopniu zależy od tego właśnie etapu.

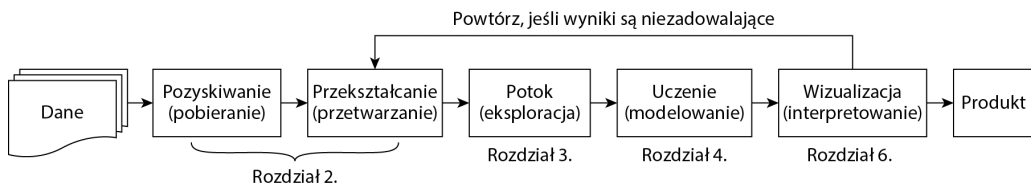
Po kompletnym zdefiniowaniu zbioru danych rozpoczyna się nowy etap. Zaczynasz badać dane, a następnie w pętli opracowywać i testować hipotezy. Załóżmy, że przyjrzałeś się graficznej postaci danych. Na podstawie statystyk opisowych tworzysz nowe zmienne, wykorzystując wiedzę z dziedziny. Ustalasz, co zrobić z nadmiarowymi i nieoczekiwanymi informacjami (przede wszystkim z obserwacjami odstającymi), i wybierasz najbardziej znaczące zmienne oraz skuteczne parametry do przetestowania za pomocą określonych algorytmów uczenia maszynowego. Warto jednak zauważyć, że czasem tradycyjne techniki uczenia maszynowego nie są dostosowane do rozwiązywanego problemu. Trzeba wtedy uciec się do analizy grafów lub innych metod z obszaru nauki o danych.

Ten etap ma postać potoku, w którym dane są przetwarzane w ramach serii kroków. Następnie tworzony jest model, przy czym możesz uznać, że trzeba powtórzyć cały proces i zacząć od przekształcania danych lub innego miejsca potoku, wprowadzić poprawki lub spróbować innych eksperymentów. Te kroki są powtarzane do czasu uzyskania sensownych wyników.

Z naszego doświadczenia wynika, że niezależnie od tego, jak obiecujące masz plany w momencie rozpoczynania analizy danych, ostatecznie uzyskasz rozwiązanie znacznie odbiegające od pierwotnych wizji. Uzyskane wyniki eksperymentów wpływają na przekształcanie danych, optymalizowanie rozwiązania, modele i łączną liczbę iteracji potrzebnych do udanego zakończenia projektu. Dlatego jeśli chcesz odnosić sukcesy jako badacz danych, nie wystarczy wymyślać teoretycznie dobrych rozwiązań. Konieczne jest szybkie budowanie prototypów dużej liczby rozwiązań w celu upewnienia się, która ścieżka będzie najlepsza. Naszym celem jest pomóc Ci maksymalnie przyspieszyć wykonywanie zadań za pomocą prezentowanych tu fragmentów kodu, które możesz wykorzystać w procesie pracy w nauce o danych.

Wyniki z projektu są przedstawiane za pomocą miary błędu lub optymalizacji (którą należy starannie dobrać pod kątem celów biznesowych). Oprócz określenia miary błędu dokonania można też zaprezentować za pomocą możliwych do zinterpretowania wniosków. Należy je słownie lub wizualnie przedstawić sponsorom projektu lub innym badaczom danych. W tym zakresie bardzo ważna jest umiejętność odpowiedniego wizualizowania wyników i wniosków za pomocą tabel, wykresów i diagramów.

Ten proces można opisać za pomocą akronimu *OSEMN* — *obtain* (czyli pobieranie), *scrub* (przetwarzanie), *explore*, *model* i *iNterpret* — przedstawionego przez Hilary Mason i Chrisa Wigginsa w słynnym, opisującym taksonomię nauki o danych wpisie z blogu *dataists* (<http://www.dataists.com/2010/09/a-taxonomy-of-data-science/>). Akronim *OSEMN* można łatwo zapamiętać, ponieważ rymuje się z angielskimi słowami *possum* i *awesome*.



Taksonomia OSEMN oczywiście nie jest szczegółowym opisem wszystkich aspektów procesu pracy w nauce o danych. Jest jednak prostym sposobem wyróżnienia najważniejszych punktów zwrotnych w tym procesie. Na przykład na etapie eksploracji występuje ważny etap „wykrywania danych”, na którym pojawiają się wszystkie nowe lub zmodyfikowane cechy. Bardzo istotny jest też poprzedzający etap „określenia reprezentacji danych”. Faza uczenia (opisana w rozdziale 4. „Uczenie maszynowe”) obejmuje nie tylko tworzenie modelu, ale też jego walidację.

Będziemy niestrudzenie przypominać, że wszystko rozpoczyna się od przekształcania danych. Zadanie to może wymagać aż 80% pracy nad całym projektem. Ponieważ nawet najdłuższa podróż rozpoczyna się od pierwszego kroku, od razu wkraczamy w ten rozdział i pokazujemy, jakich cegiełek należy użyć w udanym etapie przekształcania danych.

Wczytywanie i wstępne przetwarzanie danych za pomocą biblioteki pandas

W poprzednim rozdziale wyjaśniliśmy, gdzie znaleźć przydatne zbiory danych, oraz omówiliśmy podstawowe instrukcje importu pakietów Pythona. Skoro masz już gotowy zestaw narzędzi, w tym podrozdziale zobaczysz, jak za pomocą biblioteki pandas i pakietu NumPy wczytywać dane, operować nimi, przetwarzać je i dopracowywać.

Szybkie i łatwe wczytywanie danych

Zacznijmy od pliku CSV i biblioteki pandas. Ta biblioteka udostępnia najbardziej kompletną funkcję do wczytywania danych tabelarycznych z pliku (lub na podstawie adresu URL). Domyślnie funkcja ta zapisuje dane w specjalnej strukturze danych z biblioteki pandas, indeksuje wiersze, rozdziela zmienne za pomocą niestandardowych ograniczników, ustala typ danych odpowiedni dla każdej kolumny, przeprowadza konwersję danych (jeśli to konieczne), a także parsuje daty i obsługuje brakujące lub błędne wartości.

```
Węjście: import pandas as pd
iris_filename = 'datasets-uci-iris.csv'
iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None,
names= ['sepal_length', 'sepal_width', 'petal_length',
'petal_width',
'target'])
```

Możesz podać nazwę pliku, znak używany jako separator (sep), znak oddzielający część ułamkową liczb (decimal), określić, czy dane obejmują nagłówki (header), a także przekazać nazwy zmiennych (za pomocą parametru names i listy). W tym przykładzie parametry sep=', ' i decimal='.' przyjmują wartości domyślne, dlatego są opcjonalne. Jednak dla plików CSV z obszaru Europy należy podać oba te parametry, ponieważ w wielu krajach europejskich (a także w niektórych państwach azjatyckich) separator i znak oddzielający część ułamkową są inne od ustawień domyślnych.

Jeśli zbiór danych nie jest zapisany w komputerze, możesz wykonać opisane poniżej kroki, aby pobrać go z internetu:

```
import urllib
url = "http://aima.cs.berkeley.edu/data/iris.csv"
set1 = urllib.request.Request(url)
iris_p = urllib.request.urlopen(set1)
iris_other = pd.read_csv(iris_p, sep=',', decimal='.',
header=None, names= ['sepal_length', 'sepal_width',
'petal_length', 'petal_width', 'target'])
iris_other.head()
```

Wynikowy obiekt `iris` jest typu `DataFrame` z biblioteki `pandas`. Ten typ to coś więcej niż prosta lista lub prosty słownik Pythona. W dalszych punktach omawiamy niektóre właściwości tego typu danych. Aby poznać zawartość zbioru danych, możesz wyświetlić pierwsze lub ostatnie wiersze za pomocą następujących instrukcji:

Wejście: `iris.head()`

Wyjście:

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Wejście: `iris.tail()`
`[...]`

Te funkcje wywołane bez argumentów wyświetlają po pięć wierszy. Jeśli chcesz otrzymać inną liczbę wierszy, wywołaj funkcję, podając tę liczbę jako argument:

Wejście: `iris.head(2)`

Ta instrukcja wyświetla tylko dwa pierwsze wiersze. Aby poznać nazwy kolumn, możesz wywołać następującą metodę:

Wejście: `iris.columns`
Wyjście: `Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'target'], dtype='object')`

Wynikowy obiekt jest bardzo interesujący. Wygląda jak lista, ale w rzeczywistości jest indeksem biblioteki `pandas`. Zgodnie z nazwą obiektu określa indeksy nazw kolumn. Na przykład aby pobrać kolumnę `target`, wykonaj takie oto kroki:

```

Wejście: Y = iris['target']
Y
Wyjście:
0      Iris-setosa
1      Iris-setosa
2      Iris-setosa
3      Iris-setosa
...
149    Iris-virginica
Name: target, dtype: object

```

Obiekt Y jest typu Series z biblioteki pandas. Na razie możesz traktować ten typ jak jednowymiarową tablicę z etykietami osi. Później przyjrzymy się mu bliżej. Zapamiętaj, że klasa Index z biblioteki pandas działa jak słownik z indeksem kolumn tabeli. Zauważ, że listę kolumn można uzyskać także za pomocą ich nazw:

```

Wejście: X = iris[['sepal_length', 'sepal_width']]
X
Wyjście:

```

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

[150 rows x 2 columns]

Tu wynik jest obiektem typu DataFrame z biblioteki pandas. Dlaczego wywołania tej samej funkcji zwracają tak różne wyniki? W pierwszej instrukcji zażądaliśmy kolumny. Dlatego dane wyjściowe to wektor jednowymiarowy (typu Series z biblioteki pandas). W drugiej instrukcji zażądaliśmy wielu kolumn i otrzymaliśmy wynik podobny do macierzy (a wiemy, że macierze są reprezentowane za pomocą typu DataFrame z biblioteki pandas). Dla początkujących łatwym sposobem na dostrzeżenie różnicy jest przyjrzenie się nagłówkowi danych wyjściowych. Jeśli kolumnom przypisane są nazwy, używany jest typ DataFrame. Gdy wynikiem jest wektor bez nagłówka, używany jest typ Series.

Przedstawiliśmy już standardowe kroki procesu pracy w nauce o danych. Po wczytaniu zbioru danych zwykle należy oddzielić cechy od docelowych wyników. W problemach wymagających klasyfikowania docelowe wyniki to zwykle wartości dyskretne lub łańcuchy znaków oznaczające klasę powiązaną z poszczególnymi zestawami cech.

Dalsze kroki wymagają ustalenia, jak rozbudowany jest problem. Dlatego trzeba poznać wielkość zbioru danych. Zwykle w celu ustalenia liczby obserwacji zliczane są wiersze, a w celu określenia liczby cech należy zliczyć kolumny.

Aby uzyskać liczbę wymiarów zbioru danych, zastosuj atrybut `shape` obiektu typu `DataFrame` lub `Series`:

```

Wejście: print (X.shape)
Wyjście: (150, 2)
Wejście: print (Y.shape)
Wyjście: (150,)

```

Wynikowym obiektem jest krotka zawierająca wielkość macierzy (lub tablicy) w każdym wymiarze. Zauważ, że dla typu `Series` używany jest ten sam format (jednoelementowa krotka).

Radzenie sobie z problematycznymi danymi

Na tym etapie zapewne lepiej rozumiesz już podstawy procesu pracy i jesteś gotowy zmierzyć się z problematycznymi zbiorami danych. W praktyce kłopoty z danymi zdarzają się bardzo często. Zobacz, co się stanie, jeśli plik CSV zawiera nagłówki, za to brakuje w nim niektórych wartości i dat. Aby przykład był realistyczny, załóżmy, że dane pochodzą z biura podróży. Obejmują one temperaturę w trzech popularnych lokalizacjach i miejscowość wybraną przez klienta:

```

Date,Temperature_city_1,Temperature_city_2,Temperature_city_3,
Which_destination
20140910,80,32,40,1
20140911,100,50,36,2
20140912,102,55,46,1
20140912,60,20,35,3
20140914,60,,32,3
20140914,,57,42,2

```

Używane są tu tylko liczby całkowite, a plik zawiera nagłówki. Pierwszą próbę wczytania zbioru danych ilustruje następujące polecenie:

```

Wejście: import pandas as pd
Wejście: fake_dataset = pd.read_csv('a_loading_example_1.csv',
sep=',') fake_dataset
Wyjście:

```

	Date	Temperature_city_1	Temperature_city_2	Temperature_city_3	Which_destination
0	20140910	80.0	32.0	40	1
1	20140911	100.0	50.0	36	2
2	20140912	102.0	55.0	46	1
3	20140913	60.0	20.0	35	3
4	20140914	60.0	NaN	32	3
5	20140915	NaN	57.0	42	2

Biblioteka pandas automatycznie przypisuje kolumnom nazwy pobrane z pierwszego wiersza. W danych widoczny jest pierwszy problem: wszystkie dane (nawet daty) zostały przetworzone jako liczby całkowite (mogą być też przetwarzane jako łańcuchy znaków). Jeśli format dat jest typowy, można spróbować zastosować procedurę automatycznego ich przetwarzania, wskazując kolumnę z datami. W poniższym przykładzie technika ta sprawdza się bardzo dobrze z następującymi argumentami:

Wejście: `fake_dataset = pd.read_csv('a_loading_example_1.csv', parse_dates=[0])`
fake_dataset
Wyjście:

	Date	Temperature_city_1	Temperature_city_2	Temperature_city_3	Which_destination
0	2014-09-10	80.0	32.0	40	1
1	2014-09-11	100.0	50.0	36	2
2	2014-09-12	102.0	55.0	46	1
3	2014-09-13	60.0	20.0	35	3
4	2014-09-14	60.0	NaN	32	3
5	2014-09-15	NaN	57.0	42	2

Aby wyeliminować luki w danych (opisane jako NaN — ang. *not a number*, czyli „nie liczba”), można zastąpić je sensownymi wartościami (tu może być to 50 stopni Fahrenheita). W niektórych sytuacjach jest to uzasadnione (dalej w rozdziale opisujemy więcej problemów i strategii radzenia sobie z lukami w danych). Oto potrzebna instrukcja:

Wejście: `fake_dataset.fillna(50)`
Wyjście:

	Date	Temperature_city_1	Temperature_city_2	Temperature_city_3	Which_destination
0	2014-09-10	80.0	32.0	40	1
1	2014-09-11	100.0	50.0	36	2
2	2014-09-12	102.0	55.0	46	1
3	2014-09-13	60.0	20.0	35	3
4	2014-09-14	60.0	50.0	32	3
5	2014-09-15	50.0	57.0	42	2

Po tej operacji luki w danych zniknęły zastąpione stałą wartością 50.0. Radzenie sobie z lukami w danych też wymaga różnych strategii. Zamiast stosować pokazaną wcześniej instrukcję, możesz zastąpić luki ujemną stałą wartością, aby podkreślić fakt, że różni się ona od pozostałych danych (a interpretację zostawiamy wtedy algorytmowi odpowiedzialnemu za uczenie):

Wejście: `fake_dataset.fillna(-1)`

Zauważ, że ta technika uzupełnia luki tylko w widoku danych (nie modyfikuje pierwotnego obiektu typu DataFrame). Aby zmienić same dane, zastosuj argument `inplace=True`.

Wartości NaN można zastąpić także średnią lub medianą dla kolumny. Pozwala to zminimalizować błąd związany ze zgadywaniem brakujących wartości.

Wejście: `fake_dataset.fillna(fake_dataset.mean(axis=0))`

Metoda `.mean` oblicza średnią arytmetyczną wartości z określonej osi.

Zauważ, że ustawienie `axis=0` oznacza obliczanie średniej z uwzględnieniem wielu wierszy. Średnie są wtedy obliczane dla kolumn. Opcja `axis=1` powoduje uwzględnienie wielu kolumn i uzyskanie wyników dla wierszy. W ten sam sposób działają inne metody przyjmujące parametr `axis` (zarówno z biblioteki pandas, jak i z pakietu NumPy).

Metoda `.median` działa analogicznie do metody `.mean`, ale oblicza medianę. Jest to przydatne, gdy średnia jest mało reprezentatywna i daje zniekształcone dane (na przykład gdy cecha przyjmuje wiele skrajnych wartości).

Inny problem z obsługą rzeczywistych zbiorów danych związany jest z błędami i nieprawidłowymi wierszami. Po ich napotkaniu metoda `load_csv` kończy pracę i zgłasza wyjątek. Możliwym rozwiązaniem, akceptowalnym, gdy błędne obserwacje nie stanowią większości danych, jest odrzucanie wierszy powodujących wyjątki. W wielu sytuacjach prowadzi to do treningu algorytmu uczenia maszynowego bez błędnych informacji. Załóżmy, że używasz błędnie sformatowanego zbioru danych i chcesz wczytać tylko poprawne wiersze, ignorując te nieprawidłowo sformatowane.

Poniżej pokazujemy, jak zrobić to z użyciem opcji `error_bad_lines`:

```

Va11,Va12,Va13
0,0,0
1,1,1
2,2,2,2
3,3,3
Wejście: bad_dataset = pd.read_csv('a_loading_example_2.csv',
error_bad_lines=False)
bad_dataset
Wyjście:
Skipping line 4: expected 3 fields, saw 4

```

	Val1	Val2	Val3
0	0	0	0
1	1	1	1
2	3	3	3

Radzenie sobie z dużymi zbiorami danych

Jeśli zbiór danych, który chcesz wczytać, jest za duży, aby zmieścić go w pamięci, możesz zastosować wsadowy algorytm uczenia maszynowego, działający w danym momencie tylko na porcjach danych. Podejście wsadowe ma sens także wtedy, gdy potrzebujesz jedynie próbki danych (na przykład chcesz tylko podejrzeć ich zawartość). W Pythonie możliwe jest wczytywanie danych w porcjach. Ta operacja to strumieniowanie danych, ponieważ zbiór danych przepływa do obiektu typu `DataFrame` lub innej struktury danych w postaci ciągłego strumienia. Natomiast w sytuacjach opisywanych wcześniej zbiór danych jest w pełni wczytywany do pamięci w jednym kroku.

Za pomocą biblioteki `pandas` można podzielić i wczytać plik na dwa sposoby. Pierwszy polega na wczytywaniu zbioru danych w porcjach o równej wielkości. Każda porcja to fragment zbioru danych zawierający wszystkie kolumny i ograniczoną liczbę wierszy (nie więcej niż określono w parametrze `chunksiz` w wywołaniu funkcji). Zauważ, że wtedy dane wyjściowe funkcji `read_csv` nie są obiektem typu `DataFrame`, ale obiektem przypominającym iterator. W celu wczytania wyników do pamięci trzeba zastosować iterację do tego obiektu:

Wejście:

```
import pandas as pd
iris_chunks = pd.read_csv(iris_filename, header=None,
names=['C1', 'C2', 'C3', 'C4', 'C5'], chunksize=10)
for chunk in iris_chunks:
    print ('Wymiary:', chunk.shape)
    print (chunk, '\n')
```

Wyjście: Wymiary: (10, 5) C1 C2 C3 C4 C50 5.1 3.5 1.4 0.2

Iris-setosa1 4.9 3.0 1.4 0.2 Iris-setosa2 4.7 3.2 1.3 0.2 Iris-setosa3 4.6 3.1 1.5 0.2

Iris-setosa4 5.0 3.6 1.4 0.2 Iris-setosa5 5.4 3.9 1.7 0.4 Iris-setosa6 4.6 3.4 1.4 0.3

Iris-setosa7 5.0 3.4 1.5 0.2 Iris-setosa8 4.4 2.9 1.4 0.2 Iris-setosa9 4.9 3.1

1.5 0.1 Iris-setosa...

Zwracanych jest też 14 innych fragmentów tego rodzaju, każdy o wymiarach (10×5). Inna metoda wczytywania dużych zbiorów danych polega na zażądaniu iteratora. Wtedy można dynamicznie określać długość (liczbę wierszy) każdego fragmentu obiektu typu `DataFrame`:

```
Wejście: iris_iterator = pd.read_csv(iris_filename, header=None,
names=['C1', 'C2', 'C3', 'C4', 'C5'], iterator=True)
```

```
Wejście: print (iris_iterator.get_chunk(10).shape)
```

Wyjście: (10, 5)

```

Wejście: print (iris_iterator.get_chunk(20).shape)
Wyjście: (20, 5)
Wejście: piece = iris_iterator.get_chunk(2) piece
Wyjście:

```

	C1	C2	C3	C4	C5
0	4.8	3.1	1.6	0.2	Iris-setosa
1	5.4	3.4	1.5	0.4	Iris-setosa

W tym przykładzie najpierw definiowany jest iterator. Następnie kod pobiera porcję danych obejmującą 10 wierszy. Później pobieranych jest 20 następnych wierszy, a w ostatnim kroku — dwa wiersze, które są wyświetlane.

Oprócz biblioteki pandas możesz zastosować pakiet CSV. Udostępnia on dwie funkcje pozwalające poruszać się po niewielkich porcjach danych z plików. Są to funkcje `reader` i `DictReader`. Aby przyjrzeć się tym funkcjom, najpierw należy zaimportować pakiet CSV:

```

Wejście: import csv

```

Funkcja `reader` przenosi dane z dysków na listy Pythona. Funkcja `DictReader` przekształca dane na słownik. Obie funkcje iteracyjnie przetwarzają wiersze wczytywanego pliku. Funkcja `reader` zwraca dokładnie to, co wczyta, bez znaku powrotu karetki i po rozbiciu danych na listę na podstawie separatora (domyślnie używany jest przecinek, można to jednak zmienić). Funkcja `DictReader` odwzorowuje dane z listy na słownik, którego klucze są definiowane na podstawie pierwszego wiersza (jeśli dostępny jest nagłówek) lub parametru `fieldnames` (przy użyciu listy łańcuchów znaków z nazwami kolumn).

Wczytywanie list za pomocą natywnych technik nie jest ograniczeniem. Łatwiej jest wtedy przyspieszyć pracę kodu za pomocą szybkich implementacji Pythona (takich jak PyPy). Ponadto listy zawsze można przekształcić w tablice `ndarrays` z pakietu `NumPy` (jest to struktura danych, którą wkrótce przedstawimy). Po wczytaniu danych do słowników o formacie podobnym do JSON-a można stosunkowo łatwo uzyskać obiekt typu `DataFrame`. Ta technika wczytywania jest bardzo skuteczna, gdy dane są rzadkie (wiersze nie obejmują wszystkich cech). Wtedy słownik zawiera tylko niezerowe wartości (różne od `null`), co pozwala zaoszczędzić dużo miejsca. Późniejsze przeniesienie danych ze słownika do obiektu typu `DataFrame` jest bardzo łatwe.

Oto prosty przykład, w którym używane są możliwości pakietu CSV.

Przyjmijmy, że plik *datasets-uci-iris.csv* pobrany ze strony <http://mldata.org/> jest bardzo duży i nie można go w całości wczytać do pamięci (to tylko fikcyjne założenie, ponieważ na początku rozdziału pokazaliśmy, że plik ten obejmuje tylko 150 przykładów i nie zawiera wiersza z nagłówkiem).

W takiej sytuacji jedyną możliwością jest wczytywanie pliku w porcjach. Zacznijmy od eksperymentu:

```

Wejście:
with open(iris_filename, 'rt') as data_stream:
    # Tryb 'rt'
    for n, row in enumerate(csv.DictReader(data_stream,
        fieldnames = ['sepal_length', 'sepal_width',
        'petal_length', 'petal_width', 'target'],
        dialect='excel')):
        if n== 0:
            print (n,row)
        else:
            break

Wyjście:
0 {'petal_width': '0.2', 'target': 'Iris-setosa', 'sepal_width': '3.5',
'sepal_length': '5.1', 'petal_length': '1.4'}

```

Jak działa przedstawiony kod? Otwiera połączenie z plikiem w trybie odczytu danych binarnych i tworzy dla pliku alias `data_stream`. Użycie polecenia `with` sprawia, że plik zostanie zamknięty po kompletnym wykonaniu instrukcji umieszczonych w następnym wcięciu.

Dalej kod uruchamia iterację (`for ... in ...`) i w wywołaniu `enumerate` uruchamia funkcję `csv.DictReader`, wczytującą dane ze strumienia `data_stream`. Ponieważ plik nie zawiera nagłówka, nazwy pól są podane w parametrze `fieldnames`. Parametr `dialect` określa, że używany jest standardowy plik CSV (dalej przedstawiamy wskazówki dotyczące wartości tego parametru).

W iteracji kod działa tak, że jeśli wczytał pierwszy wiersz, to go wyświetla. W przeciwnym razie zatrzymuje pętlę za pomocą instrukcji `break`. Polecenie `print` wyświetla tu wiersz o numerze 0 i zawartość słownika. Dlatego można ustalić wartość każdego elementu z wiersza, podając klucze w postaci nazw zmiennych.

Można napisać podobny kod z instrukcją `csv.reader`:

```

Wejście: with open(iris_filename, 'rt') as data_stream:
    for n, row in enumerate(csv.reader(data_stream,
        dialect='excel')):
        if n==0:
            print (row)
        else:
            break

Wyjście: ['5.1', '3.5', '1.4', '0.2', 'Iris-setosa']

```

Ta wersja jest jeszcze prostsza. Wyświetla też prostsze dane wyjściowe — listę z sekwencją wartości z wiersza.

Teraz na podstawie drugiej wersji można napisać generator wywoływany w pętli `for`. Pobiera on „w locie” dane z pliku w blokach o wielkości zdefiniowanej w parametrze `batch` funkcji:

```

Wejście:
def batch_read(filename, batch=5):
    # Otwieranie strumienia danych
    with open(filename, 'rt') as data_stream:
        # Zerowanie porcji danych
        batch_output = list()
        # Iteracyjne przetwarzanie pliku
        for n, row in enumerate(csv.reader(data_stream,
            dialect='excel')):
            # Jeśli porcja jest odpowiedniej wielkości...
            if n > 0 and n % batch == 0:
                # ...zostaje zwrócona jako tablica ndarray.
                yield(np.array(batch_output))
                # Zerowanie porcji i rozpoczynanie od nowa
                batch_output = list()
            # W przeciwnym razie do porcji należy dodać wiersz.
            batch_output.append(row)
        # Po zakończeniu pętli wyświetlane są pozostałe wartości.
        leftyield(np.array(batch_output))

```

Podobnie jak w poprzednim przykładzie dane są pobierane za pomocą funkcji `csv.reader` opakowanej w funkcję `enumerate`. Ta ostatnia łączy pobieraną listę danych z numerami obserwacji (numeracja rozpoczyna się od zera). Na podstawie numeru obserwacji kod albo dodaje dane do porcji, albo zwraca sterowanie do głównego programu za pomocą generatywnej funkcji `yield`. Ten proces jest powtarzany do czasu wczytania i zwrócenia całego pliku w porcjach.

```

Wejście:
import numpy as np
for batch_input in batch_read(iris_filename, batch=3):
    print (batch_input)
    break
Wyjście:
[['5.1' '3.5' '1.4' '0.2' 'Iris-setosa']
 ['4.9' '3.0' '1.4' '0.2' 'Iris-setosa']
 ['4.7' '3.2' '1.3' '0.2' 'Iris-setosa']]

```

Taką funkcję można wykorzystać w procesie uczenia za pomocą metody **SGD** (ang. *stochastic gradient descent*), co pokazujemy w rozdziale 4. „Uczenie maszynowe”, gdzie wracamy do tego fragmentu kodu i rozbudowujemy go do bardziej zaawansowanej postaci.

Dostęp do danych w innych formatach

Do tej pory używaliśmy tylko plików CSV. Biblioteka `pandas` udostępnia podobne mechanizmy i funkcje do wczytywania zbiorów danych w formatach Excel, HDFS, SQL, JSON, HTML i Stata. Ponieważ te formaty nie są używane we wszystkich projektach z obszaru nauki o danych, opanowanie wczytywania i obsługi plików w tych formatach pozostawiamy czytelnikom.

Możesz zajrzeć do szczegółowej dokumentacji dostępnej w witrynie biblioteki pandas. W przykładowym kodzie zamieszczonym na stronie poświęconej książce pokazany jest prosty przykład wczytywania tabel SQL-owych.

Obiekty typu DataFrame można też tworzyć w wyniku scalania sekwencji i innych danych przypominających listy. Zauważ, że dane skalarne są przekształcane w listy.

```

Wejście: import pandas as pd
my_own_dataset = pd.DataFrame({'Col1': range(5), 'Col2':
[1.0]*5, 'Col3': 1.0, 'Col4': 'Witaj, świecie!'})
my_own_dataset
Wyjście:

```

	Col1	Col2	Col3	Col4
0	0	1.0	1.0	Witaj, świecie!
1	1	1.0	1.0	Witaj, świecie!
2	2	1.0	1.0	Witaj, świecie!
3	3	1.0	1.0	Witaj, świecie!
4	4	1.0	1.0	Witaj, świecie!

Dla każdej tworzonej kolumny należy podać nazwy (klucze ze słownika) i wartości (wartości w słowniku odpowiadające danym kluczom). W tym przykładzie kolumny Col2 i Col3 są tworzone w inny sposób, ale zawierają te same wynikowe wartości. Pokazana technika pozwala tworzyć za pomocą bardzo prostej funkcji obiekty typu DataFrame obejmujące wartości różnych typów danych.

W tym procesie należy zadbać o to, by nie używać list o różnej wielkości. W przeciwnym razie zgłoszony zostanie wyjątek:

```

Wejście: my_wrong_own_dataset = pd.DataFrame({'Col1': range(5),
'Col2': 'string', 'Col3': range(2)})
...
ValueError: arrays must all be same length

```

Aby sprawdzić typy danych poszczególnych kolumn, zastosuj atrybut dtypes:

```

In: my_own_dataset.dtypes
Col1      int64
Col2      float64
Col3      float64
Col4      object
dtype: object

```

Ta technika jest bardzo przydatna, gdy chcesz zbadać typ danych (kategorialny, całkowitoliczbowy, zmiennoprzecinkowy) i precyzję. Czasem można przyspieszyć pracę kodu, zaokrąglając liczby zmiennoprzecinkowe do liczb całkowitych, rzutując liczby zmiennoprzecinkowe

o podwójnej precyzji na ich odpowiedniki o pojedynczej precyzji lub używając tylko jednego typu danych. W następnym fragmencie pokazujemy, jak rzutować dane. Ten kod możesz potraktować jako ogólny przykład ilustrujący zmienianie typu kolumn z danymi:

```

Wejście: my_own_dataset['Col1'] = my_own_dataset['Col1'].astype(float)
my_own_dataset.dtypes
Wyjście:
Col1    float64
Col2    float64
Col3    float64
Col4     object
dtype: object

```

Wstępne przetwarzanie danych

Potrąfisz już zaimportować zbiór danych, nawet jeśli jest duży i problematyczny. Teraz należy poznać podstawowe procedury wstępnego przetwarzania danych, pozwalające dostosować te dane do następnego kroku procesu pracy w nauce o danych.

Jeśli chcesz zastosować funkcję tylko do wybranych wierszy, powinieneś najpierw utworzyć **maskę**. Maską to sekwencja wartości logicznych (True lub False), informujących, czy dany wiersz ma być uwzględniany przez daną funkcję.

Załóżmy, że chcesz uwzględnić wszystkie wiersze ze zbioru Iris, w których wartość cechy `sepal_length` jest większa niż 6. Można zrobić to tak:

```

Wejście: mask_feature = iris['sepal_length'] > 6.0
Wyjście: mask_feature
0      False
1      False
...
146    True
147    True
148    True
149    False

```

W tym prostym przykładzie od razu widać, dla których obserwacji warunek jest spełniony (True), a dla których nie jest (False). Określa to, które wiersze zostaną wybrane.

Teraz zobacz, jak wykorzystać maskę wyboru w innym kodzie. Załóżmy, że chcemy zastąpić docelową wartość Iris-virginica wartością Nowa wartość. Można to zrobić za pomocą dwóch poniższych wierszy kodu:

```

Wejście: mask_target = iris['target'] == 'Iris-virginica'
Wyjście: iris.loc[mask_target, 'target'] = 'Nowa wartość'

```

Zobaczysz, że wszystkie wystąpienia nazwy *Iris-virginica* zostaną zastąpione określeniem Nowa wartość. Metodę `.loc()` omawiamy dalej. Na razie zapamiętaj, że umożliwia ona dostęp do danych z macierzy za pomocą indeksów wierszy i kolumn.

Aby zobaczyć nową listę danych z kolumny wartości docelowych, możesz zastosować metodę `unique()`. Jest ona bardzo przydatna do wstępnego badania zbioru danych:

```

Wejście: iris['target'].unique()
Wyjście: array(['Iris-setosa', 'Iris-versicolor', 'Nowa wartość'],
dtype=object)

```

Jeśli chcesz zapoznać się ze statystykami dotyczącymi poszczególnych cech, możesz odpowiednio pogrupować kolumny lub zastosować maskę. Metoda `groupby` z biblioteki `pandas` generuje podobne wyniki co klauzula `GROUP BY` z instrukcji SQL-owych. Następną stosowaną metodą powinna być metoda agregująca wywoływana dla jednej kolumny lub ich zbioru. Na przykład metoda agregująca `mean()` z biblioteki `pandas` to odpowiednik SQL-owej funkcji `AVG()` (zwraca średnią wartości z grupy), metoda agregująca `var()` oblicza wariancję, metoda `sum()` sumuje wartości, metoda `count()` zlicza wiersze w grupie itd. Zauważ, że wynikiem jest obiekt typu `DataFrame`, dlatego sekwencję operacji można połączyć w łańcuch. Dalej pokazujemy kilka przykładów zastosowania metody `groupby`. Pogrupowanie obserwacji według wartości docelowej (etykiety) pozwala ustalić różnice między średnimi z grup i wariancje w poszczególnych grupach.

```

Wejście: grouped_targets_mean = iris.groupby(['target']).mean()
grouped_targets_mean
Wyjście:

```

	sepal_length	sepal_width	petal_length	petal_width
target				
Iris-setosa	5.006	3.418	1.464	0.244
Iris-versicolor	5.936	2.770	4.260	1.326
New label	6.588	2.974	5.552	2.026

```

Wejście: grouped_targets_var = iris.groupby(['target']).var()
grouped_targets_var
Wyjście:

```

	sepal_length	sepal_width	petal_length	petal_width
target				
Iris-setosa	0.124249	0.145180	0.030106	0.011494
Iris-versicolor	0.266433	0.098469	0.220816	0.039106
New label	0.404343	0.104004	0.304588	0.075433

Jeśli później zechcesz posortować obserwacje za pomocą funkcji, możesz zastosować metodę `.sort_index()`:

Wejście: `iris.sort_index(by='sepal_length').head()`
Wyjście:

	sepal_length	sepal_width	petal_length	petal_width	target
13	4.3	3.0	1.1	0.1	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa

Jeżeli zbiór danych zawiera szeregi czasowe (na przykład z liczbowymi wartościami docelowymi) i chcesz zastosować do nich operację `rolling` (ponieważ w danych występuje szum), możesz zrobić to tak:

Wejście: `smooth_time_series = pd.rolling_mean(time_series, 5)`

W ten sposób obliczana jest średnia krocząca wartości. Możesz też zastosować następujące polecenie:

Wejście: `median_time_series = pd.rolling_median(time_series, 5)`

W ten sposób obliczana jest mediana krocząca. W obu sytuacjach uwzględniane okno obejmuje pięć obserwacji.

Ogólniejsza metoda `apply()` z biblioteki `pandas` pozwala wykonywać dowolne operacje na wierszach lub kolumnach. Należy ją wywoływać bezpośrednio dla obiektów typu `DataFrame`. Pierwszym argumentem tej metody jest funkcja, która ma być wywoływana dla wierszy lub kolumn. Drugi argument określa oś, do której stosowana jest ta funkcja. Zauważ, że można tu stosować funkcje wbudowane, biblioteczne, lambdy i dowolne funkcje zdefiniowane przez użytkownika.

W ramach przykładu ilustrującego tę dającą duże możliwości technikę spróbujmy zliczyć niezerowe elementy z każdego wiersza. Za pomocą metody `apply` jest to proste:

Wejście: `iris.apply(np.count_nonzero, axis=1).head()`
Wyjście: 0 5
 1 5
 2 5
 3 5
 4 5
dtype: int64

Aby ustalić liczbę niezerowych elementów dla cech (czyli dla kolumn), wystarczy zmienić wartość drugiego argumentu na 0:

```

Wejście: iris.apply(np.count_nonzero, axis=0)
Wyjście: sepal_length    150
          sepal_width     150
          petal_length    150
          petal_width     150
          target          150
          dtype: int64

```

Aby zastosować funkcję do elementów, należy wywołać metodę `applymap()` dla obiektu typu `DataFrame`. Wtedy podawany jest tylko jeden argument — stosowana funkcja.

Załóżmy, że chcesz ustalić długość łańcuchów znaków z poszczególnych komórek. Aby uzyskać tę wartość, najpierw trzeba rzutować zawartość każdej komórki na łańcuch znaków, a następnie ustalić długość łańcucha. Za pomocą metody `applymap` można to zrobić w bardzo prosty sposób:

```

Wejście: iris.applymap(lambda e1:len(str(e1))).head()
Wyjście:

```

	sepal_length	sepal_width	petal_length	petal_width	target
0	3	3	3	3	11
1	3	3	3	3	11
2	3	3	3	3	11
3	3	3	3	3	11
4	3	3	3	3	11

Wybieranie danych

Ostatnie zagadnienie związane z biblioteką `pandas` dotyczy wybierania danych. Zaczniemy od przykładu. Może zetknąłeś się już ze zbiorem danych obejmującym kolumnę z indeksem. W jaki sposób poprawnie zaimportować ją za pomocą biblioteki `pandas`? I jak wykorzystać taką kolumnę do uproszczenia sobie pracy?

Posłużymy się tu bardzo prostym zbiorem danych zawierającym kolumnę z indeksem (z wartościami licznika, a nie z wartościami cechy). Aby przykład był uniwersalny, indeks rozpoczyna się od wartości 100. Tak więc indeks wiersza 0 to 100:

```

n,va11,va12,va13
100,10,10,C
101,10,20,C
102,10,30,B
103,10,40,B
104,10,50,A

```

Gdy spróbujesz wczytać ten plik w standardowy sposób, pole `n` zostanie potraktowane jak cecha (kolumna). Samo w sobie nie jest to błędem, jednak należy uważać, by nie zastosować omyłkowo indeksu jako cechy. Dlatego lepiej zapisać go oddzielnie. Jeśli indeks przypadkowo zostanie użyty na etapie uczenia (generowania modelu), może spowodować „wyciek informacji”, co jest istotnym źródłem błędów w uczeniu maszynowym.

Jeśli indeks zawiera liczby losowe, nie wpłynie na skuteczność modelu. Jeżeli jednak obejmuje liczby porządkowe, czas lub nawet elementy z informacjami (gdy określone przedziały wartości reprezentują pozytywne skutki, a inne przedziały oznaczają skutki negatywne), może powodować włączenie w model danych uzyskanych w wyniku „wycieku informacji”. Takie informacje nie będą dostępne w trakcie stosowania modelu do nowych danych (ponieważ indeks nie będzie istniał).

Wejście: `import pandas as pd`

Wejście: `dataset = pd.read_csv('a_selection_example_1.csv')` `dataset`

Wyjście:

	n	val1	val2	val3
0	100	10	10	C
1	101	10	20	C
2	102	10	30	B
3	103	10	40	B
4	104	10	50	A

Dlatego gdy wczytujesz taki zbiór danych, możesz określić, że `n` to kolumna z indeksem. Ponieważ jest to pierwsza kolumna, można zastosować następujące polecenie:

Wejście: `dataset = pd.read_csv('a_selection_example_1.csv', index_col=0)` `dataset`

Wyjście:

	val1	val2	val3
n			
100	10	10	C
101	10	20	C
102	10	30	B
103	10	40	B
104	10	50	A

Tu po wczytaniu zbioru danych indeks jest poprawny. Dostęp do wartości komórek można teraz uzyskać na kilka sposobów. Omawiamy je jeden po drugim.

Pierwsza technika polega na wskazaniu kolumny i potrzebnego wiersza (za pomocą jego indeksu).

Aby pobrać wartość kolumny `va13` z piątego wiersza (o indeksie `n=104`), można wywołać następującą instrukcję:

```
Wejście: dataset['va13'][104]
Wyjście: 'A'
```

Zachowaj staranność, ponieważ nie jest to macierz, a nasuwać się może podanie najpierw wiersza, a dopiero potem kolumny. Pamiętaj, że używany jest obiekt typu `DataFrame`, gdzie operator `[]` najpierw określa kolumnę, a dopiero potem element w uzyskanym obiekcie typu `Series`.

Podobnie wygląda dostęp do danych za pomocą metody `.loc()`:

```
Wejście: dataset.loc[104, 'va13']
Wyjście: 'A'
```

Tu najpierw należy podać indeks, a następnie potrzebne kolumny. Ta technika działa tak jak metoda `.ix()`, która jednak funkcjonuje dla indeksów dowolnego rodzaju (etykiet i pozycji) oraz zapewnia więcej możliwości.

Zauważ, że metoda `ix()` musi odgadnąć, jakiego rodzaju indeks podajesz. Dlatego jeśli nie chcesz mieszać etykiet z indeksami pozycyjnymi, stosuj metody `loc` i `iloc`, zapewniające bardziej ustrukturyzowane podejście.

```
Wejście: dataset.ix[104, 'va13']
Wyjście: 'A'
Wejście: dataset.ix[104, 2]
Wyjście: 'A'
```

Istnieje też w pełni zoptymalizowana funkcja `iloc()`, która pozwala określić pozycję w sposób typowy dla macierzy. Aby wskazać komórkę, należy najpierw podać numer wiersza, a następnie numer kolumny:

```
Wejście: dataset.iloc[4, 2]
Wyjście: 'A'
```

Pobieranie podmacierzy to bardzo intuicyjna operacja. Wystarczy określić listę indeksów zamiast wartości skalarnych:

```
Wejście: dataset[['va13', 'va12']][0:2]
```

Jest to odpowiednik poniższej instrukcji:

```
Wejście: dataset.loc[range(100, 102), ['va13', 'va12']]
```

Następna instrukcja działa tak samo:

```
Wejście: dataset.ix[range(100, 102), ['va13', 'va12']]
```

To polecenie też daje ten sam wynik:

Wejście: `dataset.ix[range(100, 102), [2, 1]]`

Podobnie jak to:

Wejście: `dataset.iloc[range(2), [2,1]]`

Wszystkie te instrukcje zwracają następujący obiekt typu DataFrame:

Wyjście:

	val3	val2
n		
100	C	10
101	C	20

Praca z danymi kategorialnymi i tekstowymi

Zwykle będziesz pracował z dwoma podstawowymi rodzajami danych: kategorialnymi i liczbowymi. Dane liczbowe, określające na przykład temperaturę, kwotę pieniędzy, dni użytkowania lub numer domu, mogą mieć postać liczb zmiennoprzecinkowych (1,0, -2,3, 99,99 itd.) lub całkowitych (-3, 9, 0, 1 itd.). Każda wartość, jaką takie dane mogą przyjąć, pozostaje w bezpośredniej relacji z innymi wartościami. Jest tak, ponieważ takie dane są porównywalne. Możesz więc stwierdzić, że cecha o wartości 2,0 jest większa (dwukrotnie) od cechy o wartości 1,0. Dane tego rodzaju są bardzo dobrze zdefiniowane i łatwe do zrozumienia. Dostępne są dla nich operatory binarne, na przykład „równa się”, „większy niż” i „mniejszy niż”.

Bardzo ważnym aspektem danych liczbowych jest to, że sensowne są dla nich podstawowe statystyki (na przykład średnie). Dla pozostałych kategorii nie jest to prawda, dlatego stanowi istotną cechę danych tego rodzaju.

Innym rodzajem danych, z jakimi możesz się zetknąć w pracy, są dane kategorialne (nazywane też nominalnymi). Dane kategorialne reprezentują atrybut, którego nie można zmierzyć, i przyjmują wartości (nazywane czasem poziomami) ze skończonego lub nieskończonego zbioru. Na przykład pogoda to cecha kategorialna, ponieważ przyjmuje wartości z nieciągłego zbioru (sunny, cloudy, snow, rainy, foggy). Innymi przykładami są cechy reprezentujące adresy URL, adresy IP, produkty z koszyka w sklepie internetowym, identyfikatory urządzeń itd. Dla tych danych nie da się zdefiniować operatorów „równa się”, „większy niż” i „mniejszy niż”, dlatego też nie można ich uporządkować.

Dodatkiem powiązaniem zarówno z danymi kategorialnymi, jak i liczbowymi są wartości logiczne. Można je traktować i jako dane kategorialne (obecność lub brak cechy), i jako prawdopodobieństwo wystąpienia danej cechy (coś zostało wyświetlone lub nie). Ponieważ wiele algorytmów uczenia maszynowego nie przyjmuje kategorialnych danych wejściowych, wartości logiczne często służą do kodowania cech kategorialnych jako wartości liczbowych.

Wróćmy do przykładu dotyczącego pogody. Jeśli chcesz odwzorować określającą pogodę cechę przyjmującą wartości ze zbioru [sunny, cloudy, snowy, rainy, foggy] i zakodować te wartości jako cechy binarne, utwórz pięć cech o wartościach True i False (po jednej cesze dla każdego poziomu cechy kategorialnej). Teraz odwzorowanie będzie proste:

```
Categorical_feature = sunny    binary_features = [1, 0, 0, 0, 0]
Categorical_feature = cloudy   binary_features = [0, 1, 0, 0, 0]
Categorical_feature = snowy    binary_features = [0, 0, 1, 0, 0]
Categorical_feature = rainy     binary_features = [0, 0, 0, 1, 0]
Categorical_feature = foggy     binary_features = [0, 0, 0, 0, 1]
```

Tylko jedna binarna cecha informuje o występowaniu określonej cechy kategorialnej. Pozostałe wartości są równe 0. Ten prosty krok pozwala przejść od świata kategorialnego do liczbowego. Odbywa się to kosztem większego zapotrzebowania na pamięć i dodatkowych obliczeń. Zamiast jednej cechy jest ich teraz pięć. Ogólnie zamiast jednej cechy kategorialnej o N poziomach trzeba utworzyć N cech przyjmujących dwie wartości liczbowe (1 lub 0). Ta operacja to kodowanie zero-jedynkowe lub, bardziej technicznie, binaryzacja cech nominalnych.

W wykonywaniu tej operacji pomaga biblioteka pandas, pozwalająca łatwo przeprowadzić odwzorowanie przy użyciu jednej instrukcji:

```
Wejście: import pandas as pd
categorical_feature = pd.Series(['sunny', 'cloudy', 'snowy',
                                'rainy', 'foggy'])
mapping = pd.get_dummies(categorical_feature)
mapping
Wyjście:
```

	cloudy	foggy	rainy	snowy	sunny
0	0.0	0.0	0.0	0.0	1.0
1	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	1.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0

Dane wyjściowe to obiekt typu DataFrame obejmujący poziomy kategorialne jako etykiety i odpowiednie cechy binarne w kolumnach. Aby odwzorować wartość kategorialną na listę wartości liczbowych, wykorzystaj możliwości biblioteki pandas:

```

Wejście: mapping['sunny']
Wyjście:
0  1.0
1  0.0
2  0.0
3  0.0
4  0.0
Name: sunny, dtype: float64
Wejście: mapping['cloudy']
Wyjście:
0  0.0
1  1.0
2  0.0
3  0.0
4  0.0
Name: cloudy, dtype: float64

```

W tym przykładzie wartość `sunny` jest odwzorowywana na wartości logiczne (1, 0, 0, 0, 0), wartość `cloudy` — na wartości logiczne (0, 1, 0, 0, 0) itd.

Tę samą operację można wykonać także za pomocą innego pakietu narzędzi — `scikit-learn`. Tu rozwiązanie jest bardziej złożone, ponieważ wymaga wcześniejszego przekształcenia tekstu na indeksy kategoryjne. Efekt jest jednak taki sam. Wróćmy do omawianego przykładu:

```

Wejście:
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
ohe = OneHotEncoder()
levels = ['sunny', 'cloudy', 'snowy', 'rainy', 'foggy']
fit_levs = le.fit_transform(levels)
ohe.fit([[fit_levs[0]], [fit_levs[1]], [fit_levs[2]], [fit_levs[3]],
[fit_levs[4]]])
print (ohe.transform([le.transform(['sunny'])]).toarray())
print (ohe.transform([le.transform(['cloudy'])]).toarray())
Wyjście:
[[ 0.  0.  0.  0.  1.]]
[[ 1.  0.  0.  0.  0.]]

```

Obiekt typu `LabelEncoder` odwzorowuje tekst na od 0 do N liczb całkowitych (zauważ, że otrzymane wartości działają jak zmienne kategoryjne, ponieważ określanie ich kolejności nie ma sensu). Tu pięć wartości jest odwzorowywanych na pięć zmiennych binarnych.

Specjalny rodzaj danych — tekst

Poznaj teraz inny rodzaj danych. Tekstowe dane wyjściowe często są używane w algorytmach uczenia maszynowego, ponieważ stanowią naturalną reprezentację danych w naszym języku. Te dane są tak bogate, że zawierają też odpowiedź na stawiane pytania. W trakcie przetwarzania tekstu najczęściej używane są zbiory słów z powtórzeniami. W tym podejściu każde

słowo jest traktowane jak cecha, a tekst jest wektorem gromadzącym niezerowe elementy odpowiadające wszystkim zawartym w nim cechom (czyli słowom). Jaka jest liczba cech w tekstowym zbiorze danych? To proste — wystarczy określić unikatowe słowa i ustalić ich liczbę. W bardzo rozbudowanym tekście zawierającym wszystkie angielskie słowa ta liczba wynosi około 600 000. Jeśli nie chcesz dodatkowo przetwarzać tekstu (usuwać określeń w trzeciej osobie, skrótów, form skróconych i akronimów), to ta liczba może być jeszcze wyższa, ale zdarza się to rzadko. W prostym podejściu, jakie stosujemy w tej książce, wystarczy pozwolić Pythonowi wykonywać jego zadania.

W tym punkcie używamy tekstowego zbioru danych. Jest to znany zbiór danych *20 Newsgroups* (więcej informacji na jego temat znajdziesz na stronie <http://qwone.com/~jason/20Newsgroups/>). Jest to kolekcja około 20 000 dokumentów obejmujących 20 tematów z grup dyskusyjnych. Jest to jeden z najpopularniejszych (jeśli nie najpopularniejszy) zbiorów danych używanych w kontekście klasyfikowania i klastrowania tekstu. W instrukcji importu pobieramy tylko podzbiór tego zbioru, obejmujący tematy naukowe (medycynę i przestrzeń kosmiczną):

```
Wejście: from sklearn.datasets import fetch_20newsgroups
categories = ['sci.med', 'sci.space'] twenty_sci_news =
fetch_20newsgroups(categories=categories)
```

Gdy pierwszy raz uruchomisz ten fragment, kod automatycznie pobierze zbiór danych i umieści go w katalogu domyślnym `$HOME/scikit_learn_data/20news_home/`. Z obiektu ze zbiorem danych można pobrać lokalizację plików, ich zawartość i etykietę (temat dyskusji, z której pochodzi dany dokument). Te informacje są dostępne w atrybutach `.filenames`, `.data` i `.target` obiektu:

```
Wejście: print(twenty_sci_news.data[0])
Wyjście: From: flb@flb.optiplan.fi ("F.Baube[tm]") Subject:
Vandalizing the sky
X-Added: Forwarded by Space Digest
Organization: [via International Space University]
Original-Sender: isu@VACATION.VENARI.CS.CMU.EDU
Distribution: sci
Lines: 12
From: "Phil G. Fraering" <pgf@sr103.cacs.usl.edu> [...]
Wejście: twenty_sci_news.filenames
Wyjście: array([
'/Users/datascientist/scikit_learn_data/20news_home/20news-
bydatetrain/sci.space/61116',
'/Users/datascientist/scikit_learn_data/20news_home/20news-
bydatetrain/sci.med/58122',
'/Users/datascientist/scikit_learn_data/20news_home/20news-
bydatetrain/sci.med/58903', ...,
'/Users/datascientist/scikit_learn_data/20news_home/20news-
bydatetrain/sci.space/60774', [...])
Wejście: print (twenty_sci_news.target[0])
print (twenty_sci_news.target_names[twenty_sci_news.target[0]])
Wyjście:
1
sci.space
```


Wartości docelowe są kategoryjne, ale są reprezentowane za pomocą liczb całkowitych (0 oznacza `sci.med`, a 1 to `sci.space`). Jeśli chcesz się z nimi zapoznać, podaj odpowiedni indeks tablicy `twenty_sci_news.target`.

Najłatwiejszy sposób przetwarzania takiego tekstu polega na przekształceniu zawartości zbioru danych na serię słów. Oznacza to, że dla każdego dokumentu należy zliczyć wystąpienia poszczególnych słów.

W ramach przykładu przygotujmy niewielki, łatwy w przetwarzaniu zbiór danych:

- `Document_1`: *We love data science,*
- `Document_2`: *Data science is great.*

W całym zbiorze danych, obejmującym dokumenty `Document_1` i `Document_2`, występuje tylko sześć różnych słów: `we`, `love`, `data science`, `is` i `great`. Na podstawie tej tablicy można utworzyć dla każdego dokumentu wektor cech:

```
Feature_Document_1 = [1 1 1 1 0 0]
Feature_Document_2 = [0 0 1 1 1 1]
```

Zauważ, że pomijamy tu pozycje słów i zachowujemy tylko liczbę wystąpień słów w poszczególnych dokumentach. To wszystko.

Dla zbioru danych *20 Newsgroups* można za pomocą Pythona wykonać podobną operację w prosty sposób:

```

Wejście:
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
word_count = count_vect.fit_transform(twenty_sci_news.data)
word_count.shape
Wyjście: (1187, 25638)

```

Najpierw tworzony jest obiekt typu `CountVectorizer`. Następnie kod wywołuje metodę `fit_transform` zliczającą słowa z poszczególnych dokumentów i generującą dla każdego dokumentu wektor cech. Następnie sprawdzany jest rozmiar macierzy. Zauważ, że wynikowa macierz jest rzadka, ponieważ często w każdym dokumencie występuje tylko ograniczony zestaw słów (liczba niezerowych elementów w każdym wierszu jest niewielka, a przechowywanie nadmiarowych zer nie ma sensu). Wymiary danych wyjściowych to (1187, 25638). Pierwsza wartość oznacza liczbę obserwacji w zbiorze danych (liczbę dokumentów), a druga jest liczbą cech (liczbą unikatowych słów w zbiorze danych).

Po przekształceniach z użyciem obiektu typu `CountVectorizer` każdy dokument zostaje powiązany z wektorem cech. Przyjrzyj się najpierw pierwszemu dokumentowi:

```

Wejście: print (word_count[0])
Wyjście:
(0, 10827) 2
(0, 10501) 2

```

```
(0, 17170) 1
(0, 10341) 1
(0, 4762) 2
(0, 23381) 2
(0, 22345) 1
(0, 24461) 1
(0, 23137) 7
[...]
```

Zauważ, że dane wyjściowe to wektor rzadki, w którym zapisywane są tylko niezerowe elementy. Aby bezpośrednio sprawdzić wystąpienia słów, wywołaj następujący kod:

```
Wejście: word_list = count_vect.get_feature_names()
for n in word_count[0].indices:
    print ('Słowo: "%s". Liczba wystąpień: %i' % (word_list[n],
word_count[0, n]))
Wyjście:
Słowo: from. Liczba wystąpień: 2
Słowo: flb. Liczba wystąpień: 2
Słowo: optiplan. Liczba wystąpień: 1
Słowo: fi. Liczba wystąpień: 1
Słowo: baube. Liczba wystąpień: 2
Słowo: tm. Liczba wystąpień: 2
Słowo: subject. Liczba wystąpień: 1
Słowo: vandalizing. Liczba wystąpień: 1
Słowo: the. Liczba wystąpień: 7
[...]
```

Do tej pory wszystko wygląda prosto, prawda? Przejdźmy więc do następnego zadania, bardziej złożonego i przydatnego. Zliczanie słów jest wartościowe, ale można zrobić coś więcej. Obliczmy częstotliwość ich występowania. Ta miara pozwala porównywać częstość występowania słów w zbiorach danych o różnej wielkości. Pozwala określić, czy dany wyraz jest słowem pomijanym (ang. *stop word*; są to bardzo popularne słowa takie jak „a”, „an”, „the” lub „is”), czy jest rzadki i unikatowy. Zwykle unikatowe słowa są najważniejsze, ponieważ pozwalają ustalić charakter tekstu i wysoce różnicujące cechy w procesie uczenia. Aby ustalić częstotliwość wystąpień poszczególnych słów we wszystkich dokumentach, uruchom następujący kod:

```
Wejście:
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vect = TfidfVectorizer(use_idf=False, norm='l1')
word_freq = tf_vect.fit_transform(twenty_sci_news.data)
word_list = tf_vect.get_feature_names()
for n in word_freq[0].indices:
    print ('Słowo: "%s". Częstotliwość: %0.3f' % (word_list[n],
word_freq[0, n]))
Wyjście:
Słowo: "from". Częstotliwość: 0.022
Słowo: "flb". Częstotliwość: 0.022
Słowo: "optiplan". Częstotliwość: 0.011
Słowo: "fi". Częstotliwość: 0.011
Słowo: "baube". Częstotliwość: 0.022
```

```
Słowo: "tm". Częstotliwość: 0.022
Słowo: "subject". Częstotliwość: 0.011
Słowo: "vandalizing". Częstotliwość: 0.011
Słowo: "the". Częstotliwość: 0.077
[...]
```

Suma częstotliwości wynosi tu 1 (lub, z powodu przybliżeń, jest bliska tej wartości). Dzieje się tak, ponieważ wybraliśmy normę 11. W tej sytuacji częstotliwość słów jest określana za pomocą funkcji dystrybucji prawdopodobieństwa. Czasem przydatne jest zwiększenie różnic między rzadkimi i popularnymi słowami. Wtedy w celu znormalizowania wektora cech można zastosować normę 12.

Jeszcze skuteczniejszy sposób wektoryzacji danych tekstowych polega na zastosowaniu narzędzia `Tfidf`. W skrócie można wyjaśnić, że mnoży ono częstotliwość występowania słów w danym dokumencie przez odwrotność częstotliwości występowania słowa w zbiorze dokumentów (a dokładniej logarytm tej częstotliwości)¹. Jest to bardzo przydatne do ustalania słów, które skutecznie opisują każdy dokument i dobrze różnicują dokumenty ze zbioru danych. Narzędzie `Tfidf` zyskało znaczną popularność od czasu rozpoczęcia przetwarzania danych tekstowych za pomocą komputerów. Jest ono używane w większości wyszukiwarek i programów do wyszukiwania informacji, ponieważ pozwala skutecznie mierzyć podobieństwo (i odległość) między zdaniami. Dlatego jest optymalnym narzędziem do pobierania dokumentów na podstawie zapytania wprowadzonego przez użytkownika.

```
Wejście:
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer() # Domyślne ustawienie: use_idf=True
word_tfidf = tfidf_vect.fit_transform(twenty_sci_news.data)
word_list = tfidf_vect.get_feature_names()
for n in word_tfidf[0].indices:
    print ('Słowo: "%s". Wartość tf-idf: %0.3f' % (word_list[n],
        word_tfidf[0, n]))

Out:
Słowo: "fred". Wartość tf-idf: 0.089
Słowo: "twilight". Wartość tf-idf: tf-idf 0.139
Słowo: "evening". Wartość tf-idf: tf-idf 0.113
Słowo: "in". Wartość tf-idf: tf-idf 0.024
Słowo: "presence". Wartość tf-idf: tf-idf 0.119
Słowo: "its". Wartość tf-idf: tf-idf 0.061
Słowo: "blare". Wartość tf-idf: tf-idf 0.150
Słowo: "freely". Wartość tf-idf: tf-idf 0.119
Słowo: "may". Wartość tf-idf: tf-idf 0.054
Słowo: "god". Wartość tf-idf: tf-idf 0.119
Słowo: "blessed". Wartość tf-idf: tf-idf 0.150
Słowo: "is". Wartość tf-idf: tf-idf 0.026
Słowo: "profiting". Wartość tf-idf: tf-idf 0.150
[...]
```

¹ Dokładne wyjaśnienie wzoru na TFIDF znajdziesz na stronie <https://pl.wikipedia.org/wiki/TFIDF> — *przyp. tłum.*

W tym przykładzie cztery najważniejsze informacyjnie słowa z pierwszego dokumentu to: *caste*, *baube*, *flb* i *tm* (to dla nich uzyskaliśmy najwyższy wynik *tf-idf*). To oznacza, że częstotliwość tych słów w danym dokumencie jest wysoka, natomiast w innych dokumentach pojawiają się one stosunkowo rzadko. W kontekście teorii informacji oznacza to, że entropia tych słów w dokumencie jest wysoka, a w zbiorze wszystkich dokumentów — niska.

Do tej pory generowaliśmy cechy dla każdego słowa. A może połączyć grupy słów ze sobą? Efekt ten można uzyskać, stosując bigramy zamiast unigramów. Gdy używane są bigramy (i, ogólnie, *n*-gramy), ważne są obecność lub nieobecność słowa oraz jego sąsiedztwo (czyli wyrazy znajdujące się w pobliżu i ich pozycje). Możesz oczywiście łączyć unigramy z *n*-gramami w celu uzyskania rozbudowanego wektora cech dla każdego dokumentu. Przetestujmy działanie *n*-gramów w następującym prostym przykładzie:

```

Wejście:
text_1 = 'we love data science'
text_2 = 'data science is hard'
documents = [text_1, text_2]
documents
Wyjście: ['we love data science', 'data science is hard']
Wejście: # To domyślne rozwiązanie stosowane już wcześniej.
count_vect_1_grams = CountVectorizer(ngram_range=(1, 1),
stop_words=[], min_df=1)
word_count = count_vect_1_grams.fit_transform(documents)
word_list = count_vect_1_grams.get_feature_names()
print ("Lista słów = ", word_list)
print ("Tekst text_1 można opisać za pomocą określeń", [word_list[n] + "(" +
str(word_count[0, n]) + ")" for n in word_count[0].indices])
Wyjście:
Lista słów = ['data', 'hard', 'is', 'love', 'science', 'we']
Tekst text_1 można opisać za pomocą określeń ['we(1)', 'love(1)', 'data(1)',
'science(1)']
Wejście: # Teraz generowany jest wektor wystąpień bigramów.
count_vect_1_grams = CountVectorizer(ngram_range=(2, 2))
word_count = count_vect_1_grams.fit_transform(documents)
word_list = count_vect_1_grams.get_feature_names()
print ("Lista słów = ", word_list)
print ("Tekst text_1 można opisać za pomocą określeń", [word_list[n] + "(" +
str(word_count[0, n]) + ")" for n in word_count[0].indices])
Wyjście:
Lista słów = ['data science', 'is hard', 'love data',
'science is', 'we love']
Tekst text_1 można opisać za pomocą określeń ['we love(1)', 'love data(1)',
'data science(1)']
Wejście: # Teraz generowany jest wektor z uni- i bigramami.
count_vect_1_grams = CountVectorizer(ngram_range=(1, 2))
word_count = count_vect_1_grams.fit_transform(documents)
word_list = count_vect_1_grams.get_feature_names()
print ("Lista słów = ", word_list)
print ("Tekst text_1 można opisać za pomocą określeń", [word_list[n] + "(" +
str(word_count[0, n]) + ")" for n in word_count[0].indices])

```

Wyjście:

```
Lista słów = ['data', 'data science', 'hard', 'is', 'is hard',
'love', 'love data', 'science', 'science is', 'we', 'we love']
Tekst text 1 można opisać za pomocą określeń ['we(1)', 'love(1)', 'data(1)',
'science(1)', 'we love(1)', 'love data(1)', 'data science(1)']
```

W tym przykładzie połączyliśmy w intuicyjny sposób pierwsze podejście z drugim. Tu używany jest typ `CountVectorizer`, jednak w ramach tej techniki bardzo często stosowany jest też typ `TfidfVectorizer`. Zauważ, że stosowanie *n*-gramów skutkuje wykładniczym wzrostem liczby cech.

Jeśli liczba cech jest za duża (słownictwo jest zbyt bogate, występuje za dużo *n*-gramów lub komputer ma ograniczoną wydajność), możesz zastosować sztuczkę zmniejszającą złożoność problemu (najpierw jednak oceń, czy ograniczenie złożoności jest warte wzrostu wydajności). Często stosowana jest sztuczka z haszowaniem, polegająca na generowaniu skrótów dla słów (lub *n*-gramów). Dla niektórych słów skróty są takie same, dlatego powstają kubelki ze słowami. Kubelki to zbiory semantycznie niepowiązanych słów o takich samych skrótach. Za pomocą obiektu typu `HashingVectorizer` możesz (tak jak w poniższym przykładzie) ustalić liczbę kubelków. Wynikowa macierz oczywiście odzwierciedla to ustawienie:

```
Wejście: from sklearn.feature_extraction.text import HashingVectorizer
hash_vect = HashingVectorizer(n_features=1000)
word_hashed = hash_vect.fit_transform(twenty_sci_news.data)
word_hashed.shape
Wyjście: (1187, 1000)
```

Zauważ, że nie można odwrócić procesu haszowania (jest to nieodwracalny proces generowania skrótu danych). Dlatego po przekształceniach trzeba pracować z cechami w postaci skrótów. Haszowanie ma sporo zalet: umożliwia szybkie przekształcanie zbiorów słów w wektory cech (cechami są wtedy kubelki skrótów), łatwe uwzględnianie w cechach niespotykanych wcześniej słów i unikanie nadmiernego dopasowywania modelu do danych (ponieważ do kubelków trafiają niepowiązane słowa).

Scraping stron internetowych za pomocą pakietu Beautiful Soup

W poprzednim punkcie pokazaliśmy, jak operować danymi tekstowymi, gdy zbiór danych jest już dostępny. A co zrobić, gdy konieczne są scraping strony internetowej i ręczne pobranie danych?

Taka sytuacja zdarza się częściej, niż mógłbyś podejrzewać. Scraping jest bardzo popularnym zagadnieniem w obszarze nauki o danych. Oto przykłady:

- Instytucje finansowe stosują scraping stron, by poznać nowe szczegóły i uzyskać informacje na temat firm, w które inwestują. Doskonałymi źródłami takich analiz są serwisy informacyjne, sieci społecznościowe, blogi, fora i witryny korporacyjne.

- Agencje reklamowe i domy mediowe analizują nastroje konsumentów i popularność różnych materiałów w internecie, aby zrozumieć reakcje odbiorców.
- Firmy specjalizujące się w analizach i rekomendacjach stosują scraping stron w celu zrozumienia wzorców zachowań użytkowników i ich modelowania.
- W porównywarkach internet jest używany do porównywania cen, produktów i usług. Użytkownicy otrzymują dzięki temu aktualizowany zbiorczy obraz bieżącej sytuacji.

Niestety bardzo trudno jest automatycznie przetwarzać witryny, ponieważ są one budowane i konserwowane przez inne osoby oraz różnią się infrastrukturą, lokalizacją, językami i strukturą. Jedynym wspólnym aspektem jest reprezentacja opracowana w standardowym języku, którym zwykle jest HTML.

To dlatego zdecydowana większość dostępnych obecnie narzędzi do scrapingu stron internetowych potrafi tylko na ogólnym poziomie „zrozumieć” strony HTML i poruszać się po nich. Jeden z najczęściej stosowanych parserów stron internetowych to BeautifulSoup. Jest to napisane w Pythonie bardzo stabilne i proste w użyciu narzędzie. Potrafi wykrywać błędy i fragmenty nieprawidłowo zbudowanego kodu na stronach HTML-owych (pamiętaj, że strony internetowe są często pisane przez ludzi, dlatego mogą zawierać błędy).

Kompletny opis narzędzia BeautifulSoup wymagałby całej książki. W tym miejscu przedstawiamy tylko wybrane aspekty. Przede wszystkim BeautifulSoup nie jest robotem internetowym (ang. *crawler*). Dlatego do wczytywania stron należy zastosować na przykład bibliotekę `urllib`.

Pobierzmy teraz z Wikipedii kod strony poświęconej Williamowi Szekspirowi:

```
Wejście: import urllib.request
url = 'https://en.wikipedia.org/wiki/William_Shakespeare'
request = urllib.request.Request(url)
response = urllib.request.urlopen(request)
```

Teraz należy nakazać narzędziu BeautifulSoup wczytanie zasobu i parsowanie danych za pomocą parsera HTML-a:

```
Wejście: from bs4 import BeautifulSoup
soup = BeautifulSoup(response, 'html.parser')
```

Gdy obiekt `soup` jest już gotowy, można kierować do niego zapytania. W celu pobrania tytułu zażądaj atrybutu `title`:

```
Wejście: soup.title
Wyjście: <title>William Shakespeare - Wikipedia, the free encyclopedia</title>
```

Widać tu, że zwracany jest cały znacznik `title`. Umożliwia to dokładną analizę zagnieżdżonej struktury strony w HTML-u. Załóżmy, że chcesz poznać kategorie powiązane ze stroną Wikipedii poświęconą Williamowi Szekspirowi. Bardzo przydatne jest wtedy utworzenie grafu dla artykułu i rekurencyjne pobieranie oraz parsowanie powiązanych stron. Najpierw należy

ręcznie przeanalizować stronę w HTML-u, by ustalić odpowiedni znacznik zawierający szukane informacje. Pamiętaj, że w nauce o danych „nie ma nic za darmo”. Nie istnieją funkcje automatycznego wykrywania danych, a poza tym ustalone informacje mogą się zmienić, gdy w Wikipedii zastosowany zostanie nowy format stron.

W wyniku ręcznych analiz okazuje się, że kategorie są przechowywane w znaczniku `div` o nazwie `mw-normal-catlinks`. Oprócz pierwszego odsyłacza wszystkie pozostałe są odpowiednie. Pora przystąpić do pisania kodu z uwzględnieniem poczynionych obserwacji. Kod wyświetla tu każdą kategorię, tytuł strony, do której prowadzi odsyłacz, i względny odnośnik:

```

Wejście:
section = soup.find_all(id='mw-normal-catlinks')[0]
for catlink in section.find_all("a")[1:]:
    print(catlink.get("title"), "->", catlink.get("href"))
Wyjście:
Category:William Shakespeare -> /wiki/Category:William_Shakespeare
Category:1564 births -> /wiki/Category:1564_births
Category:1616 deaths -> /wiki/Category:1616_deaths
Category:16th-century English male actors ->
/wiki/Category:16thcentury_English_male_actors
Category:English male stage actors ->
/wiki/Category:English_male_stage_actors
Category:16th-century English writers -> /wiki/Category:16thcentury_English_writers

```

Dwukrotnie używamy tu metody `find_all`, aby znaleźć wszystkie znaczniki HTML-a z tekstem podanym w argumentach. W pierwszym przypadku interesuje nas identyfikator. W drugim szukamy wszystkich znaczników `a`.

Na podstawie danych wyjściowych i z zastosowaniem tego samego kodu do nowych adresów URL można rekurencyjnie pobrać odpowiednie strony kategorii z Wikipedii i dotrzeć do kategorii nadrzędnych.

Ostatnia uwaga na temat scrapingu — pamiętaj, że ta technika nie zawsze jest dozwolona. Nawet gdy jej stosowanie jest dopuszczalne, zadbaj o ograniczenie szybkości pobierania danych. Jeśli szybkość pobierania będzie zbyt wysoka, serwer witryny może uznać to za prowadzony na niewielką skalę atak DoS i dodać Twój adres IP do czarnej listy oraz go zablokować. Więcej informacji znajdziesz w warunkach użytkowania witryny (możesz też skontaktować się z administratorami). Pobieranie danych z witryn chronionych prawem autorskim może prowadzić do problemów. To dlatego większość firm stosujących scraping korzysta w tym celu z usług zewnętrznych dostawców lub podpisuje specjalne umowy z właścicielami witryn.

Przetwarzanie danych za pomocą pakietu NumPy

Przedstawiliśmy już najważniejsze polecenia z biblioteki pandas służące do wczytywania danych i wstępnego przetwarzania ich w pamięci — w całości, w mniejszych porcjach lub nawet jako pojedynczych wierszy danych. W następnym etapie potoku należy zająć się tymi danymi w celu przygotowania macierzy z danymi dostosowanej do procedur uczenia (nadzorowanego lub nienadzorowanego).

Jako najlepszą praktykę zalecamy rozbić tego zadania na etap, w którym dane wciąż są niejednorodne (obejmują wartości liczbowe i symboliczne), i kolejny etap, gdy dane mają już postać tabeli liczb. Tabela (macierz) danych obejmuje wiersze reprezentujące obserwacje oraz kolumny z zaobserwowanymi wartościami (pełniającymi funkcję zmiennych).

Zgodnie z naszymi poradami powinieneś przełączać się między dwoma najważniejszymi pakietami Pythona służącymi do analiz naukowych (pandas i NumPy) oraz dwiema podstawowymi strukturami danych z tych pakietów (DataFrame i ndarray). Używany przez ciebie potok pracy będzie wtedy wydajniejszy i szybszy.

Ponieważ struktura danych, która ma trafić do następnej fazy (uczenia maszynowego), jest macierzą reprezentowaną za pomocą typu ndarray z pakietu NumPy, zacznijmy od efektu, jaki chcemy uzyskać. Zobacz, jak wygenerować obiekt tego typu.

N-wymiarowe tablice z pakietu NumPy

Python udostępnia natywne struktury danych, na przykład listy i słowniki, z których w miarę możliwości powinieneś korzystać. Listy pozwalają przechowywać sekwencje niejednorodnych obiektów (na tej samej liście możesz przechowywać na przykład liczby, tekst, grafikę i dźwięki). Natomiast słowniki są oparte na tablicy wyszukiwania (tablicy z haszowaniem) i pozwalają uzyskać dostęp do danych. Tymi danymi może być dowolny obiekt Pythona. Często jest nim lista lub inny słownik. Tak więc za pomocą słowników można budować złożone, wielowymiarowe struktury danych.

Listy i słowniki mają jednak określone ograniczenia. Przede wszystkim powodują problemy z pamięcią i wydajnością. Nie są zoptymalizowane pod kątem zajmowania prawie przyległych bloków pamięci, co może prowadzić do problemów, gdy próbujesz zastosować wysoce zoptymalizowane algorytmy lub obliczenia z użyciem wielu procesorów (pamięć stanowi wtedy wąskie gardło). Dlatego te struktury doskonale nadają się do przechowywania danych, ale nie do wykonywania na nich operacji. Jeśli więc chcesz operować na danych, musisz najpierw zdefiniować niestandardowe funkcje i iteracyjnie pobierać elementy listy lub słownika. Gdy przetwarzane są duże zbiory danych, iteracja często okazuje się nieoptymalna.

Pakiet NumPy udostępnia klasę `ndarray` (reprezentującą n -wymiarowe tablice) o następujących cechach:

- Jest optymalna ze względu na pamięć (i, oprócz innych aspektów, skonfigurowana tak, by przekazywać dane do procedur w języku C lub Fortran z blokami pamięci uporządkowanymi w sposób zapewniający najwyższą wydajność).
- Umożliwia szybkie obliczenia z obszaru algebry liniowej (wektoryzacja) i operacje na elementach (broadcasting²) bez konieczności stosowania iteracji w postaci pętli `for`, które w Pythonie są zwykle kosztowne obliczeniowo.
- Funkcje z ważnych bibliotek (takich jak SciPy i scikit-learn) wymagają tablic jako danych wyjściowych.

Te cechy związane są jednak z pewnymi ograniczeniami. Oto wady obiektów typu `ndarray`:

- Zwykle przechowują one tylko elementy jednego konkretnego typu danych, który można wcześniej zdefiniować (istnieje wprawdzie możliwość zdefiniowania złożonych i niejednorodnych typów danych, jednak korzystanie z nich w kontekście analiz byłoby bardzo trudne).
- Po zainicjowaniu takich obiektów ich wielkość jest stała. Jeśli zechcesz zmienić wielkość obiektu, musisz utworzyć go od nowa.

Podstawowe informacje o obiektach `ndarray` z pakietu NumPy

W Pythonie tablica to blok przyległych w pamięci danych określonego typu z nagłówkiem wskazującym schemat indeksowania i deskryptorem typu danych.

Dzięki schematowi indeksowania za pomocą tablic można reprezentować wielowymiarowe struktury danych, w których indeks każdego elementu jest krotką obejmującą n liczb całkowitych, gdzie n to liczba wymiarów. Gdy tablica jest jednowymiarowa (czyli jest wektorem sekwencyjnych danych), indeks rozpoczyna się od zera (tak jak w listach Pythona).

Gdy tablica jest dwuwymiarowa, trzeba zastosować indeks w postaci dwóch liczb całkowitych (krotki ze współrzędnymi x, y), dla trzech wymiarów potrzebne są trzy liczby całkowite (krotka x, y, z) itd.

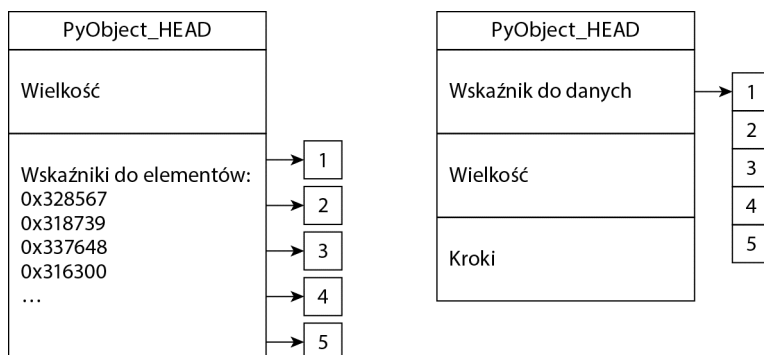
W każdej lokalizacji wskazywanej przez indeks tablica zawiera dane określonego typu. Tablica może przechowywać różne liczbowe typy danych, a także łańcuchy znaków i inne obiekty Pythona. Można też tworzyć niestandardowe typy danych i dodawać w ten sposób sekwencje różnych typów, choć odradzamy to rozwiązanie i zalecamy stosowanie w takich sytuacjach typu `DataFrame` z biblioteki `pandas`. Struktury danych z tej biblioteki zapewniają więcej możliwości,

² Czyli dostosowywanie kształtu mniejszej tablicy do większej; zobacz <https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html> — przyp. tłum.

gdy intensywnie używane są niejednorodne typy danych (jeśli takie typy są potrzebne badaczowi danych). W tej książce stosujemy tablice z pakietu NumPy tylko dla konkretnych, zdefiniowanych typów, a do obsługi niejednorodnych danych posługujemy się biblioteką pandas.

Ponieważ typ elementów tablicy i liczba zajmowanych przez nie bajtów są zdefiniowane od początku, w procesie tworzenia tablicy rezerwowana jest przestrzeń dokładnie mieszcząca wszystkie dane. Dlatego dostęp do elementów tablicy, ich modyfikowanie i obliczenia na nich odbywają się dość szybko, choć dzieje się to kosztem tego, że tablica jest stała i nie można zmienić jej struktury.

Listy Pythona są bardzo nieporęczne i powolne. Są kolekcjami wskaźników łączących listę z rozproszonymi w pamięci lokalizacjami zawierającymi same dane. Na następnym rysunku widać, że typ ndarray obejmuje tylko wskaźnik prowadzący do jednej lokalizacji w pamięci, gdzie przechowywane są uporządkowane sekwencyjnie dane. Gdy chcesz uzyskać dostęp do danych z tablicy ndarray, wymaga to mniej operacji i korzystania z mniejszej liczby lokalizacji w pamięci niż przy stosowaniu list. Z tego wynika wydajność i szybkość takich tablic w pracy z dużymi zbiorami danych. Wadą jest to, że zawartości tych tablic nie można zmieniać. Wstawianie do nich danych lub usuwanie z nich danych wymaga utworzenia nowej tablicy.



Niezależnie od liczby wymiarów tablicy ndarray dane zawsze są porządkowane jako ciągła sekwencja wartości (w ciągłym bloku pamięci). Znajomość wielkości tablicy i kroków (określających, ile bajtów w pamięci trzeba przeskoczyć, aby przejść do następnej pozycji na danej osi) pozwala na łatwe poprawne reprezentowanie tablicy i operowanie na niej.

Gdy trzeba reprezentować wiele wymiarów za pomocą list, konieczne jest utworzenie list zagnieźdzonych, co dodatkowo zwiększa koszty dostępu do danych i fragmentację danych.

Może się wydawać, że to tylko niepotrzebne gadanie informatyków. W końcu badacze danych chcą jedynie szybko wykonywać za pomocą Pythona przydatne operacje. To oczywiście słuszne podejście, jednak zwięzła („szybka”) składnia nie zawsze przekłada się na szybkie wykonywanie kodu. Jeśli opanujesz wewnętrzne mechanizmy pakietu NumPy i biblioteki pandas, będziesz mógł przyspieszyć pracę kodu i wykonywać więcej operacji w krótszym czasie. Zetknęliśmy się z sytuacją, w której czas pracy składniowo poprawnego kodu przekształcającego dane z użyciem pakietu NumPy i biblioteki pandas udało się w wyniku odpowiedniej refaktoryzacji skrócić o 95%!

Na potrzeby tej książki bardzo ważne jest też, by zrozumieć, że w ramach dostępu lub przekształcania tablicy czasem kod tylko tworzy widok jej zawartości, a czasem ją kopiuje. W trakcie **tworzenia widoku** tablicy wywoływana jest procedura, która umożliwia konwersję danych z tablicy na inne dane, przy czym źródłowa tablica się nie zmienia. W poprzednim przykładzie tworzenie widoku powoduje jedynie zmianę atrybutu opisującego wielkość tablicy ndarray. Same dane pozostają nietknięte. Dlatego transformacje danych związane z tworzeniem widoku tablicy są nietrwałe — chyba że utrwalisz je w nowej tablicy.

Natomiast w trakcie **kopiowania** powstaje nowa tablica o nowej strukturze (zajmująca świeżą pamięć). Nie polega to tylko na zmianie parametru określającego wielkość tablicy. Zamiast tego rezerwowany jest nowy przyległy blok pamięci, do którego dane zostają skopiowane.

Obiekty typu DataFrame z biblioteki pandas składają się z jednowymiarowych tablic ndarray z pakietu NumPy. Dlatego też cechują się szybkością i wydajnym wykorzystaniem pamięci typowymi dla tablic ndarray, gdy operacje dotyczą kolumn (ponieważ każda kolumna to tablica ndarray). Gdy operacje dotyczą wierszy, obiekty typu DataFrame są mniej wydajne, ponieważ sekwencyjnie używane są różne kolumny (czyli inne tablice ndarray). Ponadto wydajniejsze jest wskazywanie lokalizacji w obiektach typu DataFrame za pomocą indeksu pozycyjnego, a nie przy użyciu indeksu z biblioteki pandas. Jest tak, ponieważ w tablicach ndarray do wskazywania pozycji służą liczby całkowite. Używanie indeksów z biblioteki pandas (które mogą być tekstowe) wymaga przekształcenia indeksu na odpowiednią pozycję, aby móc poprawnie operować na danych.

Tworzenie tablic z pakietu NumPy

Tablice z pakietu NumPy można tworzyć na kilka sposobów. Oto niektóre z nich:

- przekształcenie istniejącej struktury danych w tablicę,
- utworzenie tablicy od podstaw i zapełnienie jej domyślnymi lub obliczonymi wartościami,
- wczytanie danych z dysku do tablicy.

Jeśli chcesz przekształcić istniejącą strukturę danych, najwygodniej użyć listy ustrukturyzowanej lub obiektu typu DataFrame.

Przekształcanie list w jednowymiarowe tablice

Jednym z najczęściej wykonywanych zadań związanych z pracą z danymi jest przekształcanie listy w tablicę.

W trakcie przeprowadzania takiej operacji ważne jest, by uwzględnić zawartość obiektów z listy, ponieważ wpływa to na liczbę wymiarów i typ dtype wynikowej tablicy.

Zacznijmy od listy zawierającej same liczby całkowite:

```

Wejście: import numpy as np
Wejście: # Transformacja listy w jednowymiarową tablicę
list_of_ints = [1,2,3]
Array_1 = np.array(list_of_ints)
Wejście: Array_1
Wyjście: array([1, 2, 3])

```

Pamiętaj, że dostęp do jednowymiarowej tablicy można uzyskać w ten sam sposób co do standardowych list Pythona (za pomocą rozpoczynających się od zera indeksów):

```

Wejście: Array_1[1] # Wyświetlanie drugiej wartości
Wyjście: 2

```

Można pobrać dodatkowe informacje na temat typu obiektu i typu jego elementów (wynikowy typ zależy od tego, czy system jest 32-bitowy, czy 64-bitowy):

```

Wejście: type(Array_1)
Wyjście: numpy.ndarray
Wejście: Array_1.dtype
Wyjście: dtype('int64')

```

Domyślny typ dtype zależy od używanego systemu operacyjnego.

Prosta lista liczb całkowitych zostanie przekształcona w jednowymiarową tablicę, czyli w wektor 32-bitowych liczb całkowitych (od -2^{31} do $2^{31}-1$; są to domyślne wartości liczb całkowitych w systemie używanym przez nas do wykonywania przykładów).

Kontrolowanie ilości zajmowanej pamięci

Możesz uznać, że stosowanie typu `int64` dla ograniczonego zakresu wartości to marnotrawstwo pamięci.

W sytuacjach, gdy używanych jest dużo danych, możesz obliczyć ilość pamięci zajmowaną przez obiekt `Array_1`:

```

Wejście: import numpy as np
Wejście: Array_1.nbytes
Wyjście: 24

```

Zauważ, że w systemach 32-bitowych (lub przy stosowaniu 32-bitowej wersji Pythona w systemie 64-bitowym) wynikiem jest 12.

Aby zmniejszyć ilość zajmowanej pamięci, możesz określić typ najlepiej dopasowany do tablicy:

```

Wejście: Array_1 = np.array(list_of_ints, dtype= 'int8')

```

Teraz utworzona prosta tablica zajmuje jedną czwartą pamięci potrzebnej w poprzedniej wersji. Ten przykład może wydawać się oczywisty i nadmiernie uproszczony, jednak gdy przetwarzasz miliony wierszy i kolumn, zdefiniowanie typu danych optymalnie dostosowanego do analiz pozwala uratować sytuację i zmieścić wszystkie dane w pamięci.

Dla informacji czytelników przedstawiamy tabelę z typami najczęściej używanymi w aplikacjach z obszaru nauki o danych. W tabeli tej znajdziesz też informacje o pamięci zajmowanej przez pojedyncze elementy poszczególnych typów.

Typ	Wielkość w bajtach	Opis
bool	1	Wartości logiczne (True lub False) zapisywane jako bajt
int_	4	Domyślny typ całkowitoliczbowy (zwykle int32 lub int64)
int8	1	Bajt (od -128 do 127)
int16	2	Liczba całkowita (od -32 768 do 32 767)
int32	4	Liczba całkowita (od -2^{31} do $2^{31}-1$)
int64	8	Liczba całkowita (od -2^{63} do $2^{63}-1$)
uint8	1	Liczba całkowita bez znaku (od 0 do 255)
uint16	2	Liczba całkowita bez znaku (od 0 do 65 535)
uint32	3	Liczba całkowita bez znaku (od 0 do $2^{32}-1$)
uint64	4	Liczba całkowita bez znaku (od 0 do $2^{64}-1$)
float_	8	Skrótowy zapis typu float64
float16	2	Liczba zmiennoprzecinkowa o połowicznej precyzji (wykładnik 5 bitów, mantysa 10 bitów)
float32	4	Liczba zmiennoprzecinkowa o pojedynczej precyzji (wykładnik 8 bitów, mantysa 23 bity)
float64	8	Liczba zmiennoprzecinkowa o podwójnej precyzji (wykładnik 11 bitów, mantysa 52 bity)

Istnieją też inne typy liczbowe (na przykład dla liczb zespolonych), które są stosowane rzadziej, ale mogą okazać się potrzebne w danej aplikacji (przykładowo do generowania spektrogramu). Kompletny zestaw typów znajdziesz w podręczniku użytkownika pakietu NumPy na stronie <https://docs.scipy.org/doc/numpy/user/basics.types.html>.

Jeśli chcesz zmienić typ tablicy, możesz łatwo utworzyć nową tablicę, rzutując pierwotną na określony nowy typ:

```

Wejście: Array_1b = Array_1.astype('float32')
Array_1b
Wyjście: array([ 1., 2., 3.], dtype=float32)

```

Jeśli tablica zajmuje dużo pamięci, metoda `.astype` zawsze kopiuje dane i tworzy nową tablicę.

Listy niejednorodne

Co się stanie, gdy utworzysz listę składającą się z niejednorodnych elementów, na przykład liczb całkowitych, liczb zmiennoprzecinkowych i łańcuchów znaków? Sytuacja się skomplikuje. Oto krótki przykład ilustrujący taki scenariusz:

```

Wejście: import numpy as np
complex_list = [1,2,3] + [1.,2.,3.] + ['a','b','c']
Array_2 = np.array(complex_list[:3]) # Początkowo lista wejściowa
# zawiera same liczby całkowite.
print ('complex_list[:3]', Array_2.dtype)
Array_2 = np.array(complex_list[:6]) # Następnie zawiera liczby całkowite i zmiennoprzecinkowe.
print ('complex_list[:6]', Array_2.dtype)
Array_2 = np.array(complex_list) # W ostatnim kroku dodajemy łańcuchy znaków.
print ('complex_list[:] ',Array_2.dtype)
Wyjście:
complex_list[:3] int64
complex_list[:6] float64
complex_list[:] <U32

```

Z danych wyjściowych wynika, że wartości typu `int` są zastępowane wartościami typu `float`, a z kolei te są ostatecznie przekształcane w łańcuchy znaków (sekwencja `<U32` oznacza łańcuchy znaków Unicode o wielkości do 32 bitów).

W trakcie tworzenia tablic na podstawie list możesz łączyć elementy różnych typów. Najbardziej charakterystycznym dla Pythona sposobem sprawdzenia wynikowego typu jest zbadanie atrybutu `dtype` uzyskanej tablicy.

Zauważ, że jeśli nie jesteś pewien, jakie dane zawiera tablica, musisz to sprawdzić. W przeciwnym razie może się okazać, że operowanie wynikową tablicą będzie niemożliwe, co doprowadzi do błędu (z powodu nieobsługiwanego typu operandu):

```

Wejście: # Sprawdzanie, czy tablica z pakietu NumPy zawiera elementy odpowiedniego typu liczbowego.
print (isinstance(Array_2[0],np.number))
Wyjście: False

```

W procesie przekształcania danych przypadkowe natrafienie na wyjściową tablicę łańcuchów znaków oznacza, że programista w poprzednim kroku zapomniał przekształcić zmienne na liczby (na przykład w trakcie zapisywania wszystkich danych w obiekcie typu `DataFrame`). W podrozdziale „Praca z danymi kategorialnymi i tekstowymi” omawiamy proste sposoby postępowania w takich sytuacjach.

Najpierw jednak warto dokończyć omawianie tworzenia tablic na podstawie list. Wcześniej wspomnieliśmy, że typ obiektów z listy wpływa na liczbę wymiarów tablicy.

Od list do tablic wielowymiarowych

Lista zawierająca obiekty tekstowe lub liczby jest przekształcana w tablicę jednowymiarową (reprezentującą na przykład wektor współczynników), natomiast lista list daje tablicę dwuwymiarową, a lista list list skutkuje utworzeniem tablicy trójwymiarowej:

```

Wejście: import numpy as np
# Przekształcanie listy w tablicę dwuwymiarową
a_list_of_lists = [[1,2,3],[4,5,6],[7,8,9]]
Array_2D = np.array(a_list_of_lists )
Array_2D
Wyjście:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

```

Wcześniej wspomnieliśmy, że za pomocą indeksów możesz wskazywać pojedyncze wartości. Tu używane są dwa indeksy — jeden określający wiersz (oś 0) i drugi określający kolumnę (oś 1):

```

Wejście: Array_2D[1,1]
Wyjście: 5

```

Do rozwiązywania problemów z obszaru nauki o danych standardowo używane są tablice dwuwymiarowe. Możesz też zetknąć się z tablicami trójwymiarowymi z dodatkowym wymiarem reprezentującym na przykład czas:

```

Wejście: # Przekształcanie listy w tablicę wielowymiarową
a_list_of_lists_of_lists = [[[1,2],[3,4],[5,6]],
                             [[7,8],[9,10],[11,12]]]
Array_3D = np.array(a_list_of_lists_of_lists)
Array_3D
Wyjście:
array([[[ 1, 2],
        [ 3, 4],
        [ 5, 6]],
       [[ 7, 8],
        [ 9, 10],
        [11, 12]]])

```

Aby uzyskać dostęp do pojedynczych elementów tablicy trójwymiarowej, należy podać trzy indeksy:

```

Wejście: Array_3D[0,2,0] # Dostęp do piątego elementu
Wyjście: 5

```

Tablice można tworzyć za pomocą krotek, podobnie jak buduje się listy. Słowniki można przekształcać w listy dwuwymiarowe za pomocą metody `.items()`, zwracającej kopię listy par klucz-wartość ze słownika:

```

Wejście: np.array({1:2,3:4,5:6}.items())
Wyjście: array([[1, 2], [3, 4], [5, 6]])

```

Zmiana wielkości tablic

Wcześniej wspomnieliśmy, że możliwa jest zmiana typu elementów tablicy. Teraz pokazujemy najczęściej stosowane instrukcje zmieniające kształt istniejącej tablicy.

Zacznijmy od przykładu ilustrującego metodę `.reshape`. Jako parametr przyjmuje ona krotkę n -elementową z wielkością nowych wymiarów:

```
Wejście: import numpy as np
# Zmiana kształtu tablicy z pakietu NumPy
original_array = np.array([1, 2, 3, 4, 5, 6, 7, 8])
Array_a = original_array.reshape(4,2)
Array_b = original_array.reshape(4,2).copy()
Array_c = original_array.reshape(2,2,2)
# Uwaga — metoda reshape tworzy widoki danych, a nie ich kopie.
original_array[0] = -1
```

Pierwotna tablica to jednowymiarowy wektor liczb całkowitych od 1 do 8.

- Do zmiennej `Array_a` przypisywana jest tablica `original_array` o wymiarach zmienionych na 4×2 .
- Podobna operacja wykonywana jest na zmiennej `Array_b`, przy czym tu dodatkowo wywoływana jest metoda `.copy()`, powodująca skopiowanie tablicy do nowej zmiennej.
- Do zmiennej `Array_c` przypisywana jest tablica trójwymiarowa o wymiarach $2 \times 2 \times 2$.
- Po wykonaniu tych operacji wartość pierwszego elementu tablicy `original_array` jest zmieniana z 1 na -1.

Jeśli teraz sprawdzisz zawartość tablic, zauważysz, że wymiary tablic `Array_a` i `Array_c` są zgodne z oczekiwaniami, jednak wartość pierwszego elementu wynosi -1. Dzieje się tak, ponieważ te tablice dynamicznie odzwierciedlają pierwotną tablicę, której widok przedstawiają:

```
Wejście: Array_a
Wyjście:
array([[ -1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8]])
Wejście: Array_c
Wyjście:
array([[[ -1,  2],
        [ 3,  4]],
       [[ 5,  6],
        [ 7,  8]])])
```

Tylko w tablicy `Array_b`, o wartościach skopiowanych przed zmodyfikowaniem pierwotnej tablicy, pierwszy element jest równy 1:


```

Wejście: Array_b
Wyjście:
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])

```

Jeśli chcesz zmienić wymiary pierwotnej tablicy, powinieneś zastosować metodę `resize`:

```

Wejście: original_array.resize(4,2)   original_array
Wyjście: array([[ -1, 2],
               [ 3, 4],
               [ 5, 6],
               [ 7, 8]])

```

Te same efekty można uzyskać, przypisując do atrybutu `.shape` krotkę z wartościami reprezentującymi wielkość określonych wymiarów:

```

Wejście: original_array.shape = (4,2)

```

Jeśli tablica jest dwuwymiarowa i chcesz przestawić wiersze z kolumnami (czyli przeprowadzić transpozycję), zastosuj metodę `.T` lub `transpose()`. Wynikiem takiej transformacji jest (podobnie jak przy stosowaniu metody `.reshape`) widok.

```

Wejście: original_array
Wyjście:
array([[ -1, 2],
       [ 3, 4],
       [ 5, 6],
       [ 7, 8]])

```

Tablice generowane przez funkcje z pakietu NumPy

Jeśli potrzebujesz wektora lub macierzy o określonych cechach (na przykład z zerami, jedynkami, sekwencją liczb porządkowych lub liczbami losowymi z wybranego rozkładu statystycznego), funkcje z pakietu NumPy zapewnią Ci bogate możliwości.

Zacznijmy od tego, że za pomocą funkcji `arrange` można łatwo utworzyć tablicę z pakietu NumPy zawierającą liczby porządkowe (całkowite). Funkcja ta generuje wartości całkowitoliczbowe z danego przedziału (zwykle zaczynającego się od zera), po czym można określić wymiary uzyskanych danych:

```

Wejście: import numpy as np
Wejście: ordinal_values = np.arange(9).reshape(3,3)
Wyjście:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

```

Jeśli chcesz odwrócić kolejność wartości w tablicy, zastosuj następujące polecenie:

```
Wejście: np.arange(9)[::-1]
Wyjście: array([8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Jeżeli chcesz uzyskać losowe liczby całkowite (bez określonej kolejności i z możliwymi powtórzeniami), wywołaj poniższą instrukcję:

```
Wejście: np.random.randint(low=1, high=10, size=(3,3)).reshape(3,3)
```

Przydatne mogą być też tablice zawierające same zera i jedynki. Możesz też tworzyć macierze jednostkowe:

```
Wejście: np.zeros((3,3))
Wejście: np.ones((3,3))
Wejście: np.eye(3)
```

Jeśli tablica będzie używana do przeszukiwania siatki parametrów (ang. *grid-search*) w celu ustalenia optymalnych parametrów, najbardziej przydatne będą wartości ułamkowe rosnące ze stałym krokiem lub w postępie logarytmicznym:

```
Wejście: fractions = np.linspace(start=0, stop=1, num=10)
Wejście: growth = np.logspace(start=0, stop=1, num=10, base=10.0)
```

Do inicjowania wektora lub macierzy współczynników przydatne mogą być rozkłady statystyczne (na przykład normalny lub jednostajny).

Ta instrukcja tworzy macierz o wymiarach 3×3 z wartościami standaryzowanymi zgodnie z rozkładem normalnym (ze średnią równą 0 i odchyleniem standardowym równym 1):

```
Wejście: std_gaussian = np.random.normal(size=(3,3))
```

Jeśli potrzebna jest inna średnia lub inne odchylenie standardowe, zastosuj następujące polecenie:

```
Wejście: gaussian = np.random.normal(loc=1.0, scale= 3.0, size=(3,3))
```

Parametr *loc* określa średnią, a *scale* wyznacza odchylenie standardowe.

Innym rozkładem statystycznym często wybieranym do inicjowania wektorów jest rozkład jednostajny:

```
Wejście: rand = np.random.uniform(low=0.0, high=1.0, size=(3,3))
```

Pobieranie tablicy bezpośrednio z pliku

Tablice z pakietu NumPy można też tworzyć bezpośrednio na podstawie danych z pliku.

Wykorzystajmy przykład z poprzedniego rozdziału:

```
Wejście: import numpy as np
housing = np.loadtxt('regression-datasets-housing.csv', delimiter=',', dtype=float)
```

Funkcja `loadtxt` z pakietu NumPy przyjmuje parametry `filename`, `delimiter` i `dtype`, po czym wczytuje dane do tablicy, chyba że parametr `dtype` jest nieprawidłowy (na przykład zmienna jest typu `string`, a wymagany typ elementów tablicy to `float`), tak jak w następującej instrukcji:

Wejście: `np.loadtxt('datasets-uci-iris.csv', delimiter=',', dtype=float)`
Wyjście: `ValueError: could not convert string to float: Iris-setosa`

Pobieranie danych ze struktur z biblioteki pandas

Interakcja z biblioteką pandas jest prosta. Ponieważ biblioteka ta jest oparta na pakiecie NumPy, można łatwo pobierać tablice z obiektów typu `DataFrame` oraz przekształcać tablice na takie obiekty.

Zacznijmy od wczytania danych do obiektu typu `DataFrame`. Odpowiedni będzie tu zbiór danych `BostonHouse` wczytywany w poprzednim rozdziale z repozytorium ML:

Wejście: `import pandas as pd`
`import numpy as np`
`housing_filename = 'regression-datasets-housing.csv'`
`housing = pd.read_csv(housing_filename, header=None)`

W tym przykładzie naturalnie zakładamy, że wykonujesz instrukcje w katalogu, w którym znajduje się plik z danymi. Jeśli pracujesz w innym katalogu, powinieneś podać prowadzącą do niego ścieżkę w zmiennej `housing_filename`. Aby podać poprawną kompletną ścieżkę, w systemie Windows zastosuj ukośniki odwrotne (`\`), natomiast w systemach uniksowych i Mac OS posłuż się zwykłymi ukośnikami (`/`). Funkcja `os.path.join()` pozwala pisać kod zgodny z różnymi systemami i ułatwia programistom pracę (ponieważ zwraca ścieżkę dostosowaną do systemu, w którym wywołano tę funkcję). Jeśli plik znajduje się na przykład w katalogu *mydir*, zmień przedstawiony fragment kodu w następujący sposób:

Wejście: `import pandas as pd`
`import os`
`import numpy as np`
`housing_filename = os.path.join('mydir',`
 `'regression-datasets-housing.csv')`
`housing = pd.read_csv(housing_filename, header=None)`

Zgodnie z wyjaśnieniami z punktu „Listy niejednorodnej” na tym etapie metoda `.values` zwróci tablicę elementów typu zgodnego z wszystkimi typami z używanego obiektu typu `DataFrame`:

Wejście: `housing_array = housing.values`
`housing_array.dtype`
Wyjście: `dtype('float64')`

Tu używany jest typ `float64`, ponieważ jest on ogólniejszy od typu `int`:

Wejście: `housing.dtypes`
Wyjście: `0 float64`
`1 int64`
`2 float64`

```

3    int64
4    float64
5    float64
6    float64
7    float64
8    int64
9    int64
10   int64
11   float64
12   float64
13   float64
dtype: object

```

Użycie metody `.dtypes` do sprawdzenia typów używanych w obiekcie typu `DataFrame` przed pobraniem danych do tablicy z pakietu NumPy pozwala ustalić atrybut `dtype` wynikowej tablicy. Dzięki temu można zdecydować, czy przed przejściem do dalszych zadań przekształcić, czy zmienić typ zmiennych z obiektu typu `DataFrame` (co opisaliśmy w podrozdziale „Praca z danymi kategorialnymi i tekstowymi” w tym rozdziale).

Szybkie operacje i obliczenia z użyciem pakietu NumPy

Gdy kod ma wykonywać matematyczne operacje na tablicach, wystarczy zastosować określoną operację do tablicy, posługując się stałą liczbową (wartością skalarną) lub tablicą tej samej wielkości:

```

Wejście: import numpy as np
Wejście: a = np.arange(5).reshape(1,5)
Wejście: a += 1
Wejście: a*a
Wyjście: array([[ 1,  4,  9, 16, 25]])

```

W efekcie operacja jest wykonywana na poszczególnych elementach. Na każdym elemencie wykonywana jest operacja z użyciem wartości skalarnej lub odpowiedniego elementu z innej tablicy.

Gdy zadanie dotyczy tablic o różnych wymiarach, można wykonywać operacje na elementach bez konieczności zmiany struktury danych, przy czym jeden z przetwarzanych wymiarów musi mieć wielkość 1. W takiej sytuacji wymiar o wielkości 1 jest „rozciągany” w celu dopasowania go do wymiaru drugiej tablicy. Ten proces to **broadcasting**. W ten sposób pakiet NumPy wykonuje operacje matematyczne na tablicach o różnych wymiarach. Pomaga to pisać bardziej elegancki i wydajniejszy kod.

Oto przykład:

```

Wejście: a = np.arange(5).reshape(1,5) + 1 b = np.arange(5).reshape(5,1)
+ 1 a * b
Wyjście: array([[ 1,  2,  3,  4,  5], [ 2,  4,  6,  8, 10],
               [ 3,  6,  9, 12, 15], [ 4,  8, 12, 16, 20],
               [ 5, 10, 15, 20, 25]])

```

Pokazany kod jest odpowiednikiem następujących instrukcji:

```

Wejście: a2 = np.array([1,2,3,4,5] * 5).reshape(5,5)
b2 = a2.T
a2 * b2

```

Nie wymaga jednak zajmowania dodatkowej pamięci przez pierwotne tablice w celu wykonania mnożenia par elementów.

Pakiet NumPy obejmuje też liczne inne funkcje działające na elementach tablic: `abs()`, `sign()`, `round()`, `floor()`, `sqrt()`, `log()` i `exp()`.

Inne funkcje pakietu NumPy wykonujące standardowe operacje to `sum()` i `prod()`. Zwracają one sumę i iloczyn wierszy lub kolumn tablicy na podstawie podanej osi:

```

Wejście: print (a2)
Wyjście:
[[1 2 3 4 5]
 [1 2 3 4 5]
 [1 2 3 4 5]
 [1 2 3 4 5]
 [1 2 3 4 5]]
Wejście: np.sum(a2, axis=0)
Wyjście: array([ 5, 10, 15, 20, 25])
Wejście: np.sum(a2, axis=1)
Wyjście: array([15, 15, 15, 15, 15])

```

Gdy wykonujesz operacje na danych, pamiętaj, że dostępne w pakiecie NumPy funkcje dla tablic działają bardzo szybko w porównaniu z metodami dla prostych list Pythona. Przeprowadźmy kilka eksperymentów. Spróbujmy najpierw porównać wyrażenie listowe z tablicą w kontekście dodawania stałej do elementów:

```

Wejście: %timeit -n 1 -r 3 [i+1.0 for i in range(10**6)]
%timeit -n 1 -r 3 np.arange(10**6)+1.0
Wyjście: 1 loops, best of 3: 158 ms per loop
1 loops, best of 3: 6.64 ms per loop

```

Instrukcja `%timeit` w narzędziu IPython umożliwia łatwe porównywanie czasu wykonywania operacji. Parametr `-n 1` powoduje, że fragment kodu jest wykonywany tylko w jednej pętli. Parametr `-r 3` powoduje trzykrotne ponowienie wykonania pętli (tu pętla jest tylko jedna) i zwrócenie najlepszego wyniku uzyskanego w tych powtórzeniach.

Wyniki na Twoim komputerze będą zależały od używanego systemu i jego konfiguracji. Jednak różnica między standardowymi operacjami Pythona a funkcjami z pakietu NumPy powinna pozostać duża. Gdy używane są małe zbiory danych, jest ona niezauważalna, jednak może być wyraźnie odczuwalna, gdy pracujesz z większymi zbiorami danych lub powtarzasz w pętli ten sam potok analiz dla różnych parametrów czy zmiennych.

Różnica będzie też widoczna, gdy wykonywane są skomplikowane operacje takie jak obliczanie pierwiastka kwadratowego:

```

Wejście: import math
%timeit -n 1 -r 3 [math.sqrt(i) for i in range(10**6)]
Wyjście:
1 loops, best of 3: 222 ms per loop
Wejście: %timeit -n 1 -r 3 np.sqrt(np.arange(10**6))
Wyjście: 1 loops, best of 3: 6.9 ms per loop

```

Operacje na macierzach

Oprócz wykonywania obliczeń na elementach można też, z użyciem funkcji `np.dot()`, mnożyć dwuwymiarowe tablice w sposób typowy dla macierzy (na przykład mnożyć wektor przez macierz lub macierz przez macierz):

```

Wejście: import numpy as np
M = np.arange(5*5, dtype=float).reshape(5,5)
M
Wyjście:
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.,  9.],
       [10., 11., 12., 13., 14.],
       [15., 16., 17., 18., 19.],
       [20., 21., 22., 23., 24.]])

```

W ramach przykładu stosujemy dwuwymiarową tablicę liczb całkowitych od 0 do 24. Wymiary tej tablicy wynoszą 5×5 .

Teraz zdefiniujemy wektor współczynników i tablicę obejmującą jako kolumny ten wektor w pierwotnej postaci i po jego odwróceniu:

```

Wejście: coefs = np.array([1., 0.5, 0.5, 0.5, 0.5])
coefs_matrix = np.column_stack((coefs, coefs[::-1]))
print (coefs_matrix)
Wyjście:
[[ 1.  0.5]
 [ 0.5 0.5]
 [ 0.5 0.5]
 [ 0.5 0.5]
 [ 0.5 1. ]]

```

Teraz można pomnożyć tablicę przez wektor za pomocą funkcji np.dot:

```
Wejście: np.dot(M,coefs)
Wyjście: array([ 5., 20., 35., 50., 65.]
```

Można też pomnożyć wektor przez tablicę:

```
Wejście: np.dot(coefs,M)
Wyjście: array([ 25., 28., 31., 34., 37.]
```

Jeszcze inna możliwość to pomnożenie tablicy przez ułożone w warstwy wektory współczynników (czyli macierz o wymiarach 5×2):

```
Wejście: np.dot(M,coefs_matrix)
Wyjście: array([[ 5., 7.],
                 [ 20., 22.],
                 [ 35., 37.],
                 [ 50., 52.],
                 [ 65., 67.]])
```

Pakiet NumPy udostępnia też klasę `matrix` (jest to klasa pochodna od `ndarray`, dziedzicząca wszystkie jej atrybuty i metody). Macierze z pakietu NumPy domyślnie są wyłącznie dwuwymiarowe (tablice są wielowymiarowe). Mnożenie macierzy (podobnie jak potęgowanie) nie odbywa się na elementach, tylko na całych macierzach. Dostępne są też przeznaczone dla macierzy specjalne metody `.H` (wyznacza transpozycję macierzy zespolonej) i `.I` (wyznacza macierz odwrotną). Oprócz wygody wynikającej z działania w sposób podobny jak w MATLAB-ie funkcje te nie zapewniają konkretnych korzyści. Stosowanie macierzy może utrudniać zrozumienie skryptów, ponieważ trzeba uwzględniać różny zapis mnożenia dla macierzy i obiektów.

Od Pythona 3.5 dostępny jest nowy operator `@` przeznaczony do mnożenia macierzy. Dotyczy to wszystkich pakietów Pythona (nie tylko pakietu NumPy). Wprowadzenie tego nowego operatora daje kilka korzyści.

Przede wszystkim chroni to przed przypadkami zastosowania operatora `*` do mnożenia macierzy. Operator `*` będzie stosowany tylko do tych operacji na elementach, które polegają na tym, że gdy używane są dwie macierze (lub dwa wektory) o tych samych wymiarach, operacja jest wykonywana na elementach z tych samych pozycji z obu struktur.

Ponadto kod wzorów będzie bardziej czytelny, co znacznie ułatwi jego interpretację. Nie będziesz musiał jednocześnie analizować operatorów `(+, -, / i *)` i metod `(.dot)`, ponieważ używane będą tylko operatory `(+, -, /, * i @)`.

Więcej informacji na ten temat (tu przedstawiamy tylko formalne wprowadzenie; Tobie pozostaje sprawdzić, jak działa metoda `.dot` z operatorem `@`) i przykładowe aplikacje znajdziesz w dokumencie PEP 465 (ang. *Python Enhancement Proposal*) w witrynie Python Foundation: <https://www.python.org/dev/peps/pep-0465/>.

Tworzenie wycinków i indeksowanie tablic z pakietu NumPy

Indeksowanie umożliwia generowanie widoków tablic ndarray w wyniku wskazania danych udostępnianych w widoku. Można pobrać wycinek kolumn lub wierszy albo konkretny indeks.

Zdefiniujmy roboczą tablicę:

```
Wejście: import numpy as np
M = np.arange(100, dtype=int).reshape(10,10)
```

Ten kod tworzy dwuwymiarową tablicę o wymiarach 10×10 . Zaczniemy od pobrania wycinka obejmującego jeden wymiar. Notacja pobierania danych z jednego wymiaru jest taka sama jak dla list Pythona:

```
[indeks_początkowy_włącznie:indeks_końcowy_wykluczany:kroki]
```

Załóżmy, że chcemy pobrać parzyste wiersze z przedziału od 2 do 8:

```
Wejście: M[2:9:2,:]  
Wyjście: array([[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],  
               [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],  
               [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],  
               [80, 81, 82, 83, 84, 85, 86, 87, 88, 89]])
```

Po pobraniu wycinka wierszy można dodatkowo pobrać wycinek kolumn. Wybierzmy kolumny od indeksu 5:

```
Wejście: M[2:9:2,5:]  
Wyjście:  
array([[25, 26, 27, 28, 29],  
       [45, 46, 47, 48, 49],  
       [65, 66, 67, 68, 69],  
       [85, 86, 87, 88, 89]])
```

Podobnie jak na listach można stosować ujemne wartości indeksów, aby rozpocząć odliczanie od końca. Ponadto ujemne wartości parametrów (na przykład kroku) powodują odwrócenie kolejności elementów w wyjściowej tablicy — tak jak w następnym przykładzie, gdzie odliczanie rozpoczyna się od kolumny o indeksie 5, ale odbywa się w odwrotnej kolejności i kończy na indeksie 0:

```
Wejście: M[2:9:2,5::-1]  
Wyjście:  
array([[25, 24, 23, 22, 21, 20],  
       [45, 44, 43, 42, 41, 40],  
       [65, 64, 63, 62, 61, 60],  
       [85, 84, 83, 82, 81, 80]])
```


Można też tworzyć indeksy logiczne określające, które wiersze i kolumny kod ma pobrać. Poprzedni przykład można więc zreplikować z użyciem zmiennych `row_index` i `col_index`:

```

Wejście: row_index = (M[:,0]>=20) & (M[:,0]<=80)
col_index = M[0,:]>=5
M[row_index,:][:,col_index]
Wyjście:
array([[25, 26, 27, 28, 29],
       [35, 36, 37, 38, 39],
       [45, 46, 47, 48, 49],
       [55, 56, 57, 58, 59],
       [65, 66, 67, 68, 69],
       [75, 76, 77, 78, 79],
       [85, 86, 87, 88, 89]])

```

Nie można w tym samym nawiasie kwadratowym kontekstowo stosować indeksów logicznych określających kolumny i wiersze (choć można stosować obok nich zwykle indeksy całkowitoliczbowe dla innych wymiarów). Dlatego najpierw trzeba za pomocą indeksów logicznych określić wybierane wiersze, a następnie, w nowym nawiasie kwadratowym (dotyczącym wybranych wierszy), wskazać potrzebne kolumny.

Jeśli potrzebujesz na poziomie globalnym wybrać elementy tablicy, możesz też zastosować maskę opartą na wartościach logicznych:

```

Wejście: mask = (M>=20) & (M<=90) & ((M / 10.) % 1 >= 0.5)
M[mask]
Wyjście:
array([25, 26, 27, 28, 29, 35, 36, 37, 38, 39, 45, 46, 47, 48,
       49, 55, 56, 57, 58, 59, 65, 66, 67, 68, 69, 75, 76, 77, 78, 79,
       85, 86, 87, 88, 89])

```

To podejście jest przydatne zwłaszcza wtedy, gdy trzeba wykonać operacje na fragmencie tablicy określonym za pomocą maski (na przykład `M[mask]=0`).

Inny sposób wskazywania potrzebnych elementów z tablicy polega na podaniu całkowitoliczbowych indeksów wierszy lub kolumn. Takie indeksy można zdefiniować albo za pomocą funkcji np. `where()`, która przekształca dotyczący tablicy warunek logiczny w indeksy, albo w wyniku podania sekwencji indeksów całkowitoliczbowych (mogą one występować w dowolnej kolejności, a nawet się powtarzać). Ta technika to **specjalne indeksowanie** (ang. *fancy indexing*):

```

Wejście: row_index = [1,1,2,7]
col_index = [0,2,4,8]

```

Po zdefiniowaniu indeksów wierszy i kolumn należy zastosować je kontekstowo, aby pobrać elementy, których współrzędne są określone za pomocą krotki z wartościami obu indeksów (wierszy i kolumn):

```

Wejście: M[row_index,col_index]
Wyjście: array([10, 12, 24, 78])

```

W tym podejściu wybierane są punkty o następujących współrzędnych: (1,0), (1,2), (2,4) i (7,8). Inną możliwością to, jak pokazano wcześniej, wybranie najpierw wierszy, a potem kolumn w odrębnych nawiasach kwadratowych:

```

Wejście: M[row_index,:][:,col_index]
Wyjście:
array([[10, 12, 14, 18],
       [10, 12, 14, 18],
       [20, 22, 24, 28],
       [70, 72, 74, 78]])

```

Pamiętaj, że pobieranie wycinków i stosowanie indeksów tworzy tylko widoki danych. Jeśli na podstawie widoku chcesz utworzyć nowe dane, musisz zastosować metodę `.copy()` do wycinka i przypisać wynik do innej zmiennej. W przeciwnym razie wszelkie zmiany z pierwotnej tablicy będą odzwierciedlane w wycinku i vice versa, co jest zwykle niepożądane. Metoda `copy` jest stosowana tak:

```

Wejście: N = M[2:9:2,5:].copy()

```

Dodawanie „warstw” tablic z pakietu NumPy

Gdy operujesz na tablicach dwuwymiarowych, dostępne są często używane operacje (na przykład dodawanie nowych danych i zmiennych), które funkcje z pakietu NumPy pozwalają wykonywać łatwo i szybko.

Najczęściej stosowaną taką operacją jest dodawanie nowych wierszy do tablicy.

Utwórzmy początkową tablicę:

```

Wejście: import numpy as np
dataset = np.arange(50).reshape(10,5)

```

Teraz utwórzmy jeden wiersz, a następnie zestaw wierszy, które później kolejno dodamy do tablicy:

```

In: single_line = np.arange(1*5).reshape(1,5)
a_few_lines = np.arange(3*5).reshape(3,5)

```

Spróbujmy dodać jeden wiersz:

```

Wejście: np.vstack((dataset,single_line))

```

Wystarczy podać krotkę zawierającą początkową pionową tablicę oraz tablicę, którą należy dodać. To samo polecenie zadziała także wtedy, gdy zechcesz dodać więcej wierszy:

```

Wejście: np.vstack((dataset,a_few_lines))

```

Jeśli chcesz dodać ten sam wiersz kilkakrotnie, krotka może reprezentować strukturę nowej, scalonej tablicy:

Wejście: `np.vstack((dataset, single_line, single_line))`

Inne często wykonywane zadanie to dodawanie nowej zmiennej do istniejącej tablicy. Wtedy należy zastosować funkcję `hstack` (h to skrót od *horizontal*, czyli „w poziomie”) zamiast używanego wcześniej polecenia `vstack` (v to skrót od *vertical*, czyli „w pionie”).

Załóżmy, że chcesz dodać do pierwotnej tablicy tablicę bias z wartościami jednostkowymi:

Wejście: `bias = np.ones(10).reshape(10,1) np.hstack((dataset, bias))`

Jeśli nie zmienisz wymiarów tablicy bias (może być ona wtedy dowolną sekwencją danych o tej samej długości co wiersze pierwotnej tablicy), będziesz mógł dodać ją jako sekwencję za pomocą funkcji `column_stack()`. Efekt będzie taki sam, przy czym nie trzeba będzie zmieniać wymiarów tablicy:

In: `bias = np.ones(10)
np.column_stack((dataset, bias))`

Dodawanie wierszy i kolumn do tablic dwuwymiarowych to wszystko, czego potrzebujesz do skutecznego przetwarzania danych w projektach z obszaru nauki o danych. Poznaj teraz kilka innych funkcji związanych z nieco odmiennymi problemami z danymi.

Choć tablice dwuwymiarowe są standardowo używaną strukturą, możesz też operować na strukturach trójwymiarowych. Dlatego przydatna może okazać się funkcja `dstack()`, która działa podobnie jak funkcje `hstack()` i `vstack()`, ale operuje na trzeciej osi (prowadzącej w głąb):

Wejście: `np.dstack((dataset*1, dataset*2, dataset*3))`

W tym przykładzie trzeci wymiar określa dla pierwotnej dwuwymiarowej tablicy mnożnik powodujący stopniowy wzrost wartości wzdłuż tego wymiaru (może on reprezentować czas lub poziom zmiany).

Następnym problematycznym zadaniem jest wstawianie wiersza lub (częściej) kolumny w określonym miejscu tablicy. Może pamiętasz, że tablice to przyległe bloki pamięci. Wstawianie danych wymaga podzielenia pierwotnej tablicy i utworzenia nowej. Polecenie `insert` z pakietu NumPy pomaga uzyskać ten efekt w szybki i prosty sposób:

Wejście: `np.insert(dataset, 3, bias, axis=1)`

Musisz podać tablicę, w której chcesz wstawić dane (`dataset`), pozycję (indeks 3), wstawianą sekwencję (tu jest to tablica `bias`) i oś używaną w trakcie wstawiania (oś 1 oznacza oś pionową).

Można oczywiście wstawiać całe tablice (nie tylko wektory), takie jak `bias`, przy czym ich wymiary muszą być dostosowane do osi, na której odbywa się wstawianie. W następnej instrukcji w celu wstawienia tej samej tablicy do niej samej trzeba przeprowadzić jej transpozycję:

Wejście: `np.insert(dataset, 3, dataset.T, axis=1)`

Dane można też wstawiać z użyciem innych osi. W następnej instrukcji używana jest oś 0 (pozioma), ale można uwzględnić dowolny wymiar pierwotnej tablicy:

Wejście: `np.insert(dataset, 3, np.ones(5), axis=0)`

Pierwotna tablica jest dzielona w podanej pozycji na wybranej osi. Następnie podzielone dane są łączone ze wstawianymi.

Podsumowanie

W tym rozdziale wyjaśniliśmy, że biblioteka `pandas` i pakiet `NumPy` zapewniają wszystkie narzędzia potrzebne do wczytywania i skutecznego przekształcania danych.

Zaczęliśmy od biblioteki `pandas` i używanych w niej struktur danych, `DataFrame` i `Series`, po czym przeszliśmy do wyjściowych dwuwymiarowych tablic z pakietu `NumPy`, które są strukturą danych dostosowaną do późniejszych eksperymentów i uczenia maszynowego. Przy okazji omówiliśmy takie zagadnienia jak operowanie wektorami i macierzami, kodowanie danych kategoryalnych, przetwarzanie danych tekstowych, uzupełnianie luk w danych, poprawianie błędów, przetwarzanie danych, scalanie ich i wstawianie.

Biblioteka `pandas` i pakiet `NumPy` udostępniają oczywiście znacznie więcej funkcji niż zaprezentowane tu podstawowe cegiełki w postaci poleceń i procedur. Teraz możesz wziąć dowolne nieprzetworzone dane i odpowiednio uporządkować je oraz przekształcić zgodnie z potrzebami określonego projektu z obszaru nauki o danych.

W następnym rozdziale przejdziemy do bardziej zaawansowanych operacji na danych. W tym rozdziale omówiliśmy podstawowe operacje dotyczące przekształcania danych potrzebne w ramach uczenia maszynowego. W następnym rozdziale opisujemy operacje, które pozwalają poprawić wyniki (czasem nawet w znacznym stopniu).

Skorowidz

A

- addytywny biały szum gaussowski, AWGN, 120
- aktualizowanie pakietów, 22
- algorytm
 - AdaBoost, 202
 - DBSCAN, 230
 - Extra-Trees, 198
 - kNN, 187
 - LDA, 232
 - Random Forests, 48, 198, 292
 - SGDClassifier, 213
 - SGDRegressor, 213
 - SVM, 190, 192
 - t-SNE, 131
 - XGBoost, 203
- algorytmy
 - boostingu, 195
 - dla grafów, 245
 - nieliniowe, 188
 - uśredniania, 195
- Anaconda, 23
- analiza
 - czynników ukrytych, LFA, 126
 - dyskryminacyjna, 127
 - głównych składowych, PCA, 121, 125, 129
 - macierzy błędów, 206
 - sieci społecznościowych, SNA, 239
 - składowych niezależnych, ICA, 129
 - ukrytych grup semantycznych, LSA, 128
- AWG, Arbitrary Waveform Generator, 126
- AWGN, Additive White Gaussian Noise, 120

B

- bagging, 196, 197
- Beautiful Soup, 32, 89
- bezwzględny błąd prognozy, MAE, 118
- biblioteka
 - LIBSVM, 180
 - pandas, 64, 68, 73, 103
 - seaborn, 274
- big data, 206
- bliskość, 247
- błąd, 308
 - średniokwadratowy, 148
- Bokeh, 284
- bootstrapping, 157
- BOW, Bag of Words, 233
- broadcasting, 104
- budowanie serwera predykcji, 294

C

- cechy sieci, 215
- centralność oparta
 - na bliskości, 247
 - na mierze harmonicznej, 248
 - na pośrednictwie, 246
 - na stopniu, 247
 - na wektorach własnych, 248
- czas przeszukiwania, 164
- czułość, 146, 184

D

dane

- duże zbiory, 70
- eksploracja, 113
- kategorialne, 81
- pobieranie, 103
- problematiczne, 67
- przetwarzanie, 92
- tekstowe, 81
- wczytywanie, 64
- wstępne przetwarzanie, 64, 75
- wybieranie, 78

data wrangling, 61

definiowanie funkcji, 305

dostęp do danych, 73

dostrajanie algorytmu SVM, 193

drzewo GBT, 293

duże zbiory danych, 70, 207

dystribucje naukowe, 22

E

EDA, Exploratory Data Analysis, 113, 268

eksploracja danych, 268, 279

eksploracyjna analiza danych, EDA, 113, 268

Enthought Canopy, 24

F

format GML, 252

funkcja

- Bootstrap, 159
- GenericUnivariateSelect, 168
- GridSearchCV, 160, 164
- StandardScaler, 134

funkcje, 305

- liniowe, 129
- oceny, 162
- sigmoidalne, 129
- transformacji, 176
- wielomianowe, 129

G

generatory, 309

- danych, 58
- tablic, 101

Gensim, 33

GML, Graph Modeling Language, 252

grafy, 240

- nieskierowane, 240
- skierowane, 240
- wczytywanie, 252

GTB, Gradient Tree Boosting, 202

H

haszowanie, 211

hiperparametry, 159

hipotezy

- o wysokiej wariancji, 289
- obciążone wysokim błędem systematycznym, 288

histogramy, 263, 269

I

ICA, Independent Component Analysis, 129

informacje o obiektach, 93

instalacja, 17, 19

pakietów, 20

instrukcje warunkowe, 310

interfejs API, 180

IQR, 119

iterator, 309

- LeavaPOut, 157
- LeaveOneLabelOut, 157
- LeaveOneOut, 156
- LeavePLabelOut, 157
- StratifiedKFold, 156

iteratory walidacji krzyżowej, 155

J

jądro, 141

jednostka ReLU, 216

Jupyter, 30, 37, 312

- instalacja, 41
- notatniki, 44
- polecenia, 42
- zastępniki, 49

K

Keras, 36

klasa, 307

- covariance.EllipticEnvelope, 135
- EllipticEnvelope, 136

- learning_curve, 290
- plot_partial_dependence, 294
- SGDClassifier, 209
- SGDRegressor, 213
- sklearn.svm.LinearSVC, 190
- sklearn.svm.NuSVC, 190
- sklearn.svm.NuSVR, 192
- sklearn.svm.OneClassSVM, 135, 190
- sklearn.svm.SVC, 190
- sklearn.svm.SVR, 192

- klasyfikacja
 - binarna, 147
 - wieloklasowa, 144
- klasyfikator LogisticRegression, 184
- klasyfikowanie, 190
 - tekstu, 225
- kopiowanie, 95
- krzywa
 - ROC, 147
 - uczenia, 288
 - walidacji, 290

L

- LDA, Latent Dirichlet Allocation, 232
- LDA, Linear Discriminant Analysis, 127
- LFA, Latent Factor Analysis, 126
- liniowa analiza dyskryminacyjna, LDA, 127
- listy, 95, 302
 - niejednorodne, 98
- LSA, Latent Semantical Analysis, 128

Ł

- łańcuch transformacji, 174
- łączenie cech, 174

M

- macierz, 106
 - błędów, 145
 - kowariancji, 120
 - odwrotna, 183
- MAE, Mean Absolute Error, 118, 148
- maksymalna głębokość pobierania, 254
- maska, 75
- Matplotlib, 31, 257

- mediana, 119
- metoda
 - GTB, 202
 - wektorów nośnych, SVM, 188
 - współrzędnych równoległych, 273
- miara harmoniczna, 248
- model
 - jednoczynnikowy, 168
 - ML-AAS, 294

N

- naivny klasyfikator bayesowski, 184
- narzędzie
 - conda, 23, 27
 - easy_install, 20
 - REPL, 41
 - TfIdf, 87
- nauka o danych, 16
- NER, Named Entity Recognition, 224
- NetworkX, 32
- niestandardowe funkcje transformacji, 176
- NLTK, Natural Language Toolkit, 32, 221
- NumPy, 29, 92–95

O

- obiekt, 307
- obserwacje
 - oczekiwane, 133
 - odstające, 133
 - typowe, 133
- obszary losowe, 197
- ograniczone maszyny Boltzmanna, RBM, 132
- opakowywanie
 - operacji, 173
 - poleceń, 274
- operacje
 - na macierzach, 106
 - na tablicach, 104
- operator @, 107
- opowiadanie, 41
- optymalizacja hiperparametrów, 159
- OSEMN, 63
- oznaczanie części mowy, 223

P

pakiet

- Anaconda, 23
- Beautiful Soup, 32, 89
- Bokeh, 284
- Enthought Canopy, 24
- Gensim, 33
- Jupyter, 30
- Keras, 36
- Matplotlib, 31, 257
- NetworkX, 32, 245
- NLTK, 32, 221
- NumPy, 29, 92, 93, 95
- Pandas, 30, 268
- PyPy, 33
- Python(x, y), 25
- Scikit-learn, 30, 50, 135, 189, 296
- SciPy, 29
- seaborn, 284
- Statsmodels, 31
- Theano, 35
- WinPython, 25
- XGBoost, 33

pakiety

- aktualizowanie, 22
- instalowanie, 20, 23
- podstawowe, 28

pamięć, 96

- Pandas, 30, 268

panele, 260

parametr

- alpha, 214
- C, 194
- degree, 194
- epsilon, 194, 214
- gamma, 141, 194
- kernel, 194
- ll_ratio, 214
- learning_rate, 214
- n_iter, 213
- nu, 141, 194
- penalty, 214
- shuffle, 214

- pasting, 196, 197

- PCA, Principal Component Analysis, 121

pliki

- CSV, 55, 73
- tekstowe, 55

pobieranie

- danych, 103
- tablic, 102

podpróba, 254

- podprzestrzenie losowe, 197

- podział na tokeny, 221

- polecenie pyplot.plot, 260

- pośrednictwo, 246

potok, 173

- danych, 113

- prawdopodobieństwo, 200

- precyzja, 146, 184

- problematiczne dane, 67

- programowanie obiektowe, 307

- próbkiowanie, 157

przekształcanie

- danych, 61
- list, 95

- przeszukiwanie siatki parametrów, 164

przetwarzanie

- big data, 206

- danych, 64, 75, 92

- języka naturalnego, 221

- PyCon, Python Conference, 312

- PyData, 312

- PyPI, Python Package Index, 28

- PyPy, 33

- Python(x, y), 25

R

- radialna funkcja bazowa, RBF, 129, 190

- RBF, Radial Basis Function, 190

- RBM, Restricted Boltzmann Machine, 132

- redukcja liczby wymiarów, 120

- regresja, 148, 192

- liniowa i logistyczna, 181

- regularyzacja, 150

- L1, 171

- rekurencyjna eliminacja, 169

- relacje w danych, 262

repozytorium

- MLdata.org, 54

- zbiorów danych, 180

- ROC, receiver operating characteristics curve, 147

- rozkład według wartości osobliwych, SVD, 122, 125

- rozpoznawanie nazw własnych, NER, 224

rozstęp międzykwartylowy, 264
 różnorodność, 211
 rysowanie krzywych, 259
 rysunki, 265

S

Scikit-learn, 30, 50
 SciPy, 29
 scraping, 62
 stron internetowych, 89
 seaborn, 284
 sekwencje modeli, 202
 serwer predykcji, 294
 SGD, stochastic gradient descent, 208, 213
 siatka parametrów, 164
 sieci
 głębokie, 215
 neuronowe ze sprzężeniem do przodu, 216
 społecznościowe, 239
 skalowalność, 208
 skracanie czasu przeszukiwania, 164
 słabe klasyfikatory, 196
 słowniki, 304
 SNA, Social Network Analysis, 239
 spadek wzdłuż gradientu, 202
 Statsmodels, 31
 stemming, 222
 stopień, 141, 247
 stop-słowa, 225
 stosowanie paneli, 260
 SVD, Singular Value Decomposition, 122, 125
 SVM, Support Vector Machine, 188
 szacowanie prawdopodobieństwa, 200
 sztuczna inteligencja, 214
 szum zgodny z generatorem sygnałowym, AWG,
 126
 szybkie operacje, 104
 szybkość napływu danych, 210

Ś

średni bezwzględny błąd prognozy, MAE, 148
 środowisko
 wirtualne, 25
 Jupyter, 37

T

tablice
 dodawanie wierszy, 110
 generowane przez funkcje, 101
 jednowymiarowe, 95
 pobieranie, 102
 n-wymiarowe, 92
 wielowymiarowe, 99
 zmiana wielkości, 100
 techniki jednoczynnikowe, 134
 tekst, 83
 teoria grafów, 240
 testy, 148
 Theano, 35
 token, 221
 trafność, 146
 twierdzenie Bayesa, 186
 tworzenie
 łańcuchów transformacji, 174
 niestandardowych funkcji transformacji, 176
 nowych cech, 117
 tablic, 95
 widoku, 95
 wycinków, 108
 typ danych, 97
 typy liczbowe, 97

U

uczenie
 głębokie, 214
 maszynowe, 179
 nienadzorowane, 227
 przyrostowe, 208, 210
 się na podstawie danych, 288

W

walidacja, 144, 148, 290
 krzyżowa, 153
 progresywna, 209
 wariancja cech, 167
 wczytywanie
 danych, 64
 grafów, 252
 wektory własne, 248

WinPython, 25
wizualizacje, 284
wskaźnik F1, 146
współczynnik R2, 148
wstępne przetwarzanie danych, 64, 75
wybieranie danych, 78
wybór cech, 166
 model jednoczynnikowy, 168
 na podstawie stabilności, 171
 na podstawie wariancji, 167
 regularyzacja L1, 171
wycinki, 108
wyjątki, 308
wykresy
 częściowej zależności, 293
 pudełkowe, 269
 punktowe, 262, 271
 słupkowe, 264

wykrywanie
 obserwacji odstających, 134
 wartości odstających, 133
wyrażenia
 listowe, 311
 słownikowe, 311
wyświetlanie rysunków, 265

X

XGBoost, 33, 203

Z

zarządzanie środowiskami, 27
zastępniki Jupytera, 49
zbiory danych, 50, 180
zmiana wielkości tablic, 100

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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 WYDAWNICZA



Helion SA

Nauka o danych — fascynujące algorytmy i potężne grafy!

Nauka o danych jest nową, interdyscyplinarną dziedziną, funkcjonującą na pograniczu algebry liniowej, modelowania statystycznego, lingwistyki komputerowej, uczenia maszynowego oraz metod akumulacji danych. Jest przydatna między innymi dla analityków biznesowych, statystyków, architektów oprogramowania i osób zajmujących się sztuczną inteligencją. Szczególnie praktycznym narzędziem dla tych specjalistów jest język Python, który zapewnia doskonałe środowisko do analizy danych, uczenia maszynowego i algorytmicznego rozwiązywania problemów.

Niniejsza książka jest doskonałym wprowadzeniem do nauki o danych. Jej autorzy wskażą Ci prostą i szybką drogę do rozwiązywania różnych problemów z tego obszaru za pomocą Pythona oraz powiązanych z nim pakietów do analizy danych i uczenia maszynowego. Dzięki lekturze przejdziesz przez kolejne etapy modyfikowania i wstępnego przetwarzania danych, poznając przy tym podstawowe operacje związane z wczytywaniem danych, przekształcaniem ich, poprawianiem na potrzeby analiz, eksplorowaniem i przetwarzaniem. Poza podstawami opanujesz też zagadnienia uczenia maszynowego, w tym uczenia głębokiego, techniki analizy grafów oraz wizualizacji danych.

Najważniejsze zagadnienia przedstawione w książce:

- konfiguracja środowiska Jupyter Notebook
- najważniejsze operacje stosowane w nauce o danych
- potoki danych i uczenie maszynowe
- wprowadzenie do grafów i wizualizacje
- biblioteki i pakiety Pythona służące do badań danych

Alberto Boschetti specjalizuje się w przetwarzaniu sygnałów i statystyce. Jest doktorem inżynierii telekomunikacyjnej. Zajmuje się przetwarzaniem języków naturalnych, analityką behawioralną, uczeniem maszynowym i przetwarzaniem rozproszonym.

Luca Massaron specjalizuje się w statystycznych analizach wieloczynnikowych, uczeniu maszynowym, statystyce, eksploracji danych i algorytmice. Fascynuje go potencjał, jaki drzemie w nauce o danych.

Helion

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/novosci>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-3423-6



9 788328 334236

cena: 59,00 zł

Packt