

O'REILLY®



Data science od podstaw

ANALIZA DANYCH W PYTHONIE

Tytuł oryginału: Data Science from Scratch: First Principles with Python

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-4602-4

© 2018 Helion S.A.

Authorized Polish translation of the English edition of *Data Science from Scratch*

ISBN 9781491901427 © 2015 Joel Grus

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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 HELION SA 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.

HELION SA

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/dascpo.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

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

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Przedmowa	9
Rozdział 1. Wprowadzenie	13
Znaczenie danych	13
Czym jest analiza danych?	13
Hipotetyczna motywacja	14
Rozdział 2. Błyskawiczny kurs Pythona	25
Podstawy	25
Bardziej skomplikowane zagadnienia	36
Dalsza eksploracja	43
Rozdział 3. Wizualizacja danych	45
Pakiet matplotlib	45
Wykres słupkowy	47
Wykresy liniowe	50
Wykresy punktowe	51
Dalsza eksploracja	52
Rozdział 4. Algebra liniowa	55
Wektory	55
Macierze	59
Dalsza eksploracja	61
Rozdział 5. Statystyka	63
Opis pojedynczego zbioru danych	63
Korelacja	67
Paradoks Simpsona	70
Inne pułapki związane z korelacją	71
Korelacja i przyczynowość	71
Dalsza eksploracja	72

Rozdział 6. Prawdopodobieństwo	73
Zależność i niezależność	73
Prawdopodobieństwo warunkowe	74
Twierdzenie Bayesa	75
Zmienne losowe	77
Ciągły rozkład prawdopodobieństwa	77
Rozkład normalny	79
Centralne twierdzenie graniczne	81
Dalsza eksploracja	83
Rozdział 7. Hipotezy i wnioski	85
Sprawdzanie hipotez	85
Przykład: rzut monetą	85
Przedziały ufności	89
Hakowanie wartości p	90
Przykład: przeprowadzanie testu A-B	91
Wnioskowanie bayesowskie	92
Dalsza eksploracja	95
Rozdział 8. Metoda gradientu prostego	97
Podstawy metody gradientu prostego	97
Szacowanie gradientu	98
Korzystanie z gradientu	101
Dobór właściwego rozmiaru kroku	101
Łączenie wszystkich elementów	102
Stochastyczna metoda gradientu prostego	103
Dalsza eksploracja	104
Rozdział 9. Uzyskiwanie danych	105
Strumienie stdin i stdout	105
Wczytywanie plików	107
Pobieranie danych ze stron internetowych	109
Korzystanie z interfejsów programistycznych	115
Przykład: korzystanie z interfejsów programistycznych serwisu Twitter	117
Dalsza eksploracja	120
Rozdział 10. Praca z danymi	121
Eksploracja danych	121
Oczyszczanie i wstępne przetwarzanie danych	126
Przetwarzanie danych	127
Przeskalowanie	130
Redukcja liczby wymiarów	132
Dalsza eksploracja	137

Rozdział 11. Uczenie maszynowe	139
Modelowanie	139
Czym jest uczenie maszynowe?	140
Nadmierne i zbyt małe dopasowanie	140
Poprawność	143
Kompromis pomiędzy wartością progową a wariancją	145
Ekstrakcja i selekcja cech	146
Dalsza eksploracja	147
Rozdział 12. Algorytm k najbliższych sąsiadów	149
Model	149
Przykład: ulubione języki	151
Przekleństwo wymiarowości	155
Dalsza eksploracja	159
Rozdział 13. Naiwny klasyfikator bayesowski	161
Bardzo prosty filtr antyspamowy	161
Bardziej zaawansowany filtr antyspamowy	162
Implementacja	163
Testowanie modelu	165
Dalsza eksploracja	167
Rozdział 14. Prosta regresja liniowa	169
Model	169
Korzystanie z algorytmu spadku gradientowego	172
Szacowanie maksymalnego prawdopodobieństwa	172
Dalsza eksploracja	173
Rozdział 15. Regresja wieloraka	175
Model	175
Dalsze założenia dotyczące modelu najmniejszych kwadratów	176
Dopasowywanie modelu	177
Interpretacja modelu	178
Poprawność dopasowania	178
Dygresja: ładowanie wstępne	179
Błędy standardowe współczynników regresji	180
Regularyzacja	181
Dalsza eksploracja	183

Rozdział 16. Regresja logistyczna	185
Problem	185
Funkcja logistyczna	187
Stosowanie modelu	189
Poprawność dopasowania	190
Maszyny wektorów nośnych	190
Dalsza eksploracja	194
Rozdział 17. Drzewa decyzyjne	195
Czym jest drzewo decyzyjne?	195
Entropia	197
Entropia podziału	198
Tworzenie drzewa decyzyjnego	199
Łączenie wszystkiego w całość	202
Lasy losowe	204
Dalsza eksploracja	205
Rozdział 18. Sztuczne sieci neuronowe	207
Perceptrony	207
Jednokierunkowe sieci neuronowe	209
Propagacja wsteczna	211
Przykład: pokonywanie zabezpieczenia CAPTCHA	213
Dalsza eksploracja	216
Rozdział 19. Grupowanie	217
Idea	217
Model	218
Przykład: spotkania	219
Wybór wartości parametru k	221
Przykład: grupowanie kolorów	222
Grupowanie hierarchiczne z podejściem aglomeracyjnym	224
Dalsza eksploracja	228
Rozdział 20. Przetwarzanie języka naturalnego	229
Chmury wyrazowe	229
Modele n -gram	231
Gramatyka	234
Na marginesie: próbkowanie Gibbsa	236
Modelowanie tematu	237
Dalsza eksploracja	241

Rozdział 21. Analiza sieci społecznościowych	243
Pośrednictwo	243
Centralność wektorów własnych	248
Grafy skierowane i metoda PageRank	251
Dalsza eksploracja	253
Rozdział 22. Systemy rekomendujące	255
Ręczne rozwiązywanie problemu	255
Rekomendowanie tego, co jest popularne	256
Filtrowanie kolaboratywne oparte na użytkownikach	257
Filtrowanie kolaboratywne oparte na zainteresowaniach	260
Dalsza eksploracja	261
Rozdział 23. Bazy danych i SQL	263
Polecenia CREATE TABLE i INSERT	263
Polecenie UPDATE	265
Polecenie DELETE	265
Polecenie SELECT	266
Polecenie GROUP BY	267
Polecenie ORDER BY	269
Polecenie JOIN	270
Zapytania składowe	272
Indeksy	272
Optymalizacja zapytań	273
Bazy danych NoSQL	274
Dalsza eksploracja	274
Rozdział 24. Algorytm MapReduce	275
Przykład: liczenie słów	275
Dlaczego warto korzystać z algorytmu MapReduce?	277
Algorytm MapReduce w ujęciu bardziej ogólnym	277
Przykład: analiza treści statusów	278
Przykład: mnożenie macierzy	280
Dodatkowe informacje: zespaland	281
Dalsza eksploracja	281
Rozdział 25. Po prostu zabierz się za praktykę	283
IPython	283
Matematyka	283
Korzystanie z gotowych rozwiązań	284
Szukanie danych	286
Zabierz się za analizę	287
Skorowidz	289

Algorytm k najbliższych sąsiadów

Jeżeli chcesz zdenerwować swoich sąsiadów, to powiedz o nich prawdę.

— Pietro Aretino

Założmy, że chcesz przewidzieć, na kogo oddam swój głos w kolejnych wyborach prezydenckich. Jeżeli nic o mnie nie wiesz, ale dysponujesz jakimiś danymi na mój temat, to możesz przyjrzeć się planom wyborczym moich *sąsiadów*. Mieszkam na przedmieściach Seattle, a moi sąsiedzi niezmiennie głosują na kandydata Partii Demokratycznej, zatem obstawienie tego, że również i ja będę głosował na kandydata tej partii, wydaje się dość rozsądne.

Założmy, że posiadasz nie tylko dane dotyczące miejsca, w którym mieszkam. Dysponujesz danymi dotyczącymi mojego wieku, dochodu, liczby dzieci itd. Są to rzeczy, które do pewnego stopnia wpływają na moje zachowanie. Przyglądanie się preferencjom wyborczym moich sąsiadów, którzy są mi bliscy również w płaszczyźnie tych dodatkowych wymiarów, wydaje się być jeszcze lepszym pomysłem. W taki sposób działa **metoda klasyfikacji najbliższych sąsiadów**.

Model

Metoda najbliższych sąsiadów to najprostszy istniejący model predyktywny. Nie wymaga ona tworzenia zadanych założeń matematycznych ani stosowania ciężkiej artylerii. Wymaga ona jedynie:

- miary odległości;
- założenia, że punkty znajdujące się bliżej siebie są do siebie bardziej podobne.

Większość technik, z których będziemy korzystać w tej książce, analizuje cały zbiór danych w celu znalezienia w nim pewnych prawidłowości. Algorytm najbliższych sąsiadów dość często ignoruje wiele informacji, ponieważ przewidywania parametrów nowego elementu zbioru danych zależą tylko od parametrów punktów, które są położone najbliżej.

Ponadto metoda najbliższych sąsiadów nie pomoże Ci zrozumieć czynników wpływających na analizowany fenomen. Przewidywanie moich preferencji wyborczych na podstawie preferencji moich sąsiadów nie mówi zbyt wiele o przyczynach moich preferencji. Przyczyny mógłby określić alternatywny model przewidujący moje preferencje wyborcze na podstawie np. mojego dochodu i stanu cywilnego.

Ogólnie rzecz biorąc, dysponujemy elementami zbioru danych i odpowiadającymi im etykietami. Etykiety te mogą mieć postać np. wartości True lub False, które określają, czy dana wartość wejściowa spełnia warunek taki jak np. „jest spamem”, „jest trucizną” lub „jest filmem wartym obejrzenia”. Etykiety mogą być również kategoriami — takimi jak np. kategorie wiekowe filmów (G, PG, PG-13, R i NC-17) — nazwiskami kandydatów startujących w wyborach prezydenckich lub nazwami ulubionego języka programowania.

W naszym przypadku elementy zbioru danych będą wektorami, a więc możemy korzystać z funkcji `distance` utworzonej w rozdziale 4.

Załóżmy, że wybraliśmy wartość k równą np. 3 lub 5 i chcemy dokonać klasyfikacji nowych obserwacji — znaleźć k najbliższych elementów zbioru danych, oznaczonych etykietami, i na ich podstawie ustalić etykietę nowej obserwacji.

W tym celu potrzebujemy funkcji liczącej głosy. Oto przykładowa implementacja mechanizmu liczenia głosów:

```
def raw_majority_vote(labels):
    votes = Counter(labels)
    winner, _ = votes.most_common(1)[0]
    return winner
```

Niestety mechanizm taki nie robi niczego inteligentnego z węzłami. Załóżmy, że określamy kategorię wiekową filmów i pięciu najbliższym filmom nadano kategorie G, G, PG, PG i R. W związku z tym mamy dwa głosy na kategorię G i dwa głosy na kategorię PG. W takim przypadku możemy skorzystać z kilku rozwiązań:

- Wybrać zwycięzcę w sposób losowy.
- Określić wagi głosów na podstawie odległości i wybrać zwycięzcę na podstawie wagi.
- Zmniejszać wartość parametru k aż do jednoznacznego określenia zwycięzcy.

Zaimplementujemy trzecie rozwiązanie:

```
def majority_vote(labels):
    """Funkcja zakłada, że etykiety są ustawione w kolejności
    od najbliższej do najdalszej."""
    vote_counts = Counter(labels)
    winner, winner_count = vote_counts.most_common(1)[0]
    num_winners = len([count
                       for count in vote_counts.values()
                       if count == winner_count])

    if num_winners == 1:
        return winner                                     # Ustalono jednoznacznie zwycięzcę. Zwróć go.
    else:
        return majority_vote(labels[:-1]) # Odrzuć najdalszą obserwację i spróbuj ponownie.
```

Koniec końców metoda ta zwróci zwycięzcę. W najgorszym wypadku zostaniemy z jedną etykietą, która zostanie uznana za zwycięzcę.

Dysponując tą funkcją, możemy z łatwością utworzyć klasyfikator:

```
def knn_classify(k, labeled_points, new_point):
    """Każdemu elementowi zbioru danych powinna być przypisana etykieta."""

    # Ustaw punkty oznaczone etykietami w kolejności od najbliższego do najdalszego.
    by_distance = sorted(labeled_points,
                          key=lambda (point, _): distance(point, new_point))

    # Ustal etykiety k najbliższych punktów.
    k_nearest_labels = [label for _, label in by_distance[:k]]

    # Wybierz zwycięzcę na podstawie tych etykiet.
    return majority_vote(k_nearest_labels)
```

Przyjrzyjmy się działaniu tego mechanizmu w praktyce.

Przykład: ulubione języki

Właśnie udało Ci się uzyskać dostęp do pierwszej ankiety przeprowadzonej wśród użytkowników serwisu DataSciencecenter, która miała na celu ustalenie preferowanych przez nich języków programowania (użytkownicy biorący udział w ankiecie mieszkają tylko w większych miastach):

```
# Każdy element zbioru ma formę: ([długość geogr., szerokość geogr.], ulubiony_język).

cities = [([(-122.3, 47.53], "Python"), # Seattle
          ([ -96.85, 32.85], "Java"),   # Austin
          ([ -89.33, 43.13], "R"),      # Madison
          # I tak dalej...
```

Osoba zarządzająca działem aktywizacji społeczności chce dowiedzieć się, czy wyniki ankiety mogą zostać użyte w celu przewidzenia preferowanego języka programowania w miejscach, które nie były objęte ankietą.

Jak zwykle warto zacząć pracę od przedstawienia danych na wykresie (zobacz rysunek 12.1):

```
# Klucz definiuje język, a wartość jest parą (długość geogr., szerokość geogr.).
plots = { "Java" : ([], []), "Python" : ([], []), "R" : ([], []) }

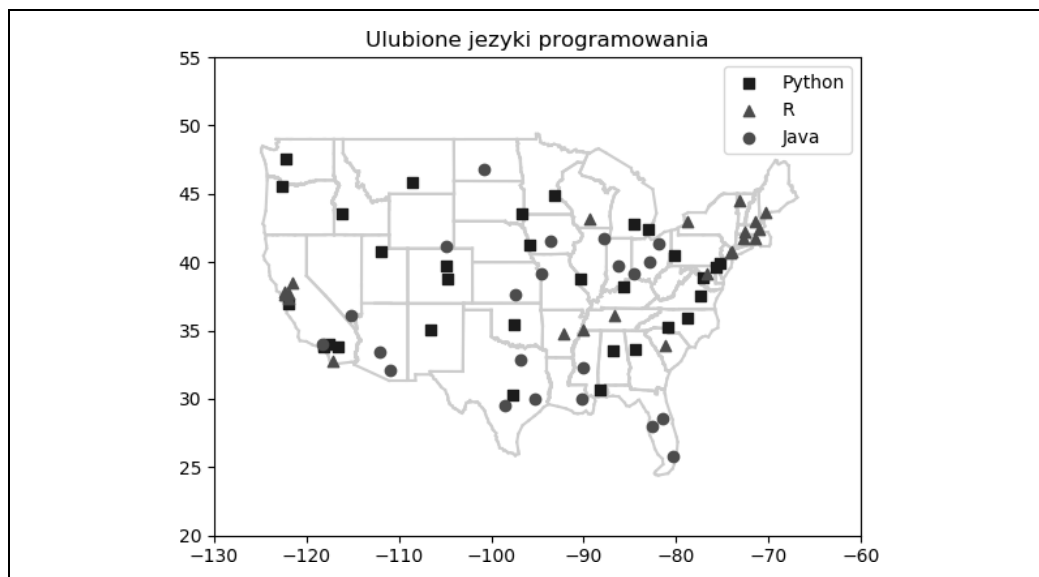
# Chcemy nadać każdemu językowi inny znak i kolor.
markers = { "Java" : "o", "Python" : "s", "R" : "^" }
colors = { "Java" : "r", "Python" : "b", "R" : "g" }

for (longitude, latitude), language in cities:
    plots[language][0].append(longitude)
    plots[language][1].append(latitude)

# Utwórz wykres punktowy danych każdego języka.
for language, (x, y) in plots.iteritems():
    plt.scatter(x, y, color=colors[language], marker=markers[language],
               label=language, zorder=10)

plot_state_borders(plt)    # Załóżmy, że mamy funkcję generującą kontury stanów.

plt.legend(loc=0)          # Pozwól pakietowi matplotlib wybrać najlepsze miejsce na legendę.
plt.axis([-130,-60,20,55]) # Definicja osi.
plt.title("Ulubione języki programowania")
plt.show()
```



Rysunek 12.1. Ulubione języki programowania



W kodzie znajduje się odwołanie do niezdefiniowanej funkcji `plot_state_borders()`. Implementację tej funkcji znajdziesz w pliku `plot_state_borders.py` wchodzącym w skład archiwum z kodem znajdującym się na stronie <ftp://ftp.helion.pl/przyklady/dascpo.zip>, ale warto spróbować zaimplementować ją samodzielnie:

1. Poszukaj w internecie współrzędnych geograficznych granic pomiędzy poszczególnymi stanami.
2. Zamień dane, które uda Ci się znaleźć, na listę segmentów `[(dlug1, szer1), (dlug2, szer2)]`.
3. Nanieś segmenty na wykres za pomocą funkcji `plt.plot()`.

Wydaje się, że te same języki programowania są lubiane w zbliżonych do siebie miejscach, a więc model generujący przewidywania możemy oprzeć na algorytmie k najbliższych sąsiadów.

Na początek zobaczmy, co się stanie, jeżeli spróbujemy przewidzieć język preferowany w każdym mieście na podstawie etykiet sąsiadów, a nie etykiety wybranego miasta:

```
# Wypróbuj kilka wartości k.
for k in [1, 3, 5, 7]:
    num_correct = 0

    for city in cities:
        location, actual_language = city
        other_cities = [other_city
                        for other_city in cities
                        if other_city != city]

        predicted_language = knn_classify(k, other_cities, location)

        if predicted_language == actual_language:
            num_correct += 1

    print k, "sąsiad(ów):", num_correct, "poprawnie na", len(cities)
```

Wydaje się, że algorytm 3 najbliższych sąsiadów działa najlepiej — zwraca poprawne wyniki w około 59% przypadków:

```
1 sąsiad(ów): 40 poprawnie na 75
3 sąsiad(ów): 44 poprawnie na 75
5 sąsiad(ów): 41 poprawnie na 75
7 sąsiad(ów): 35 poprawnie na 75
```

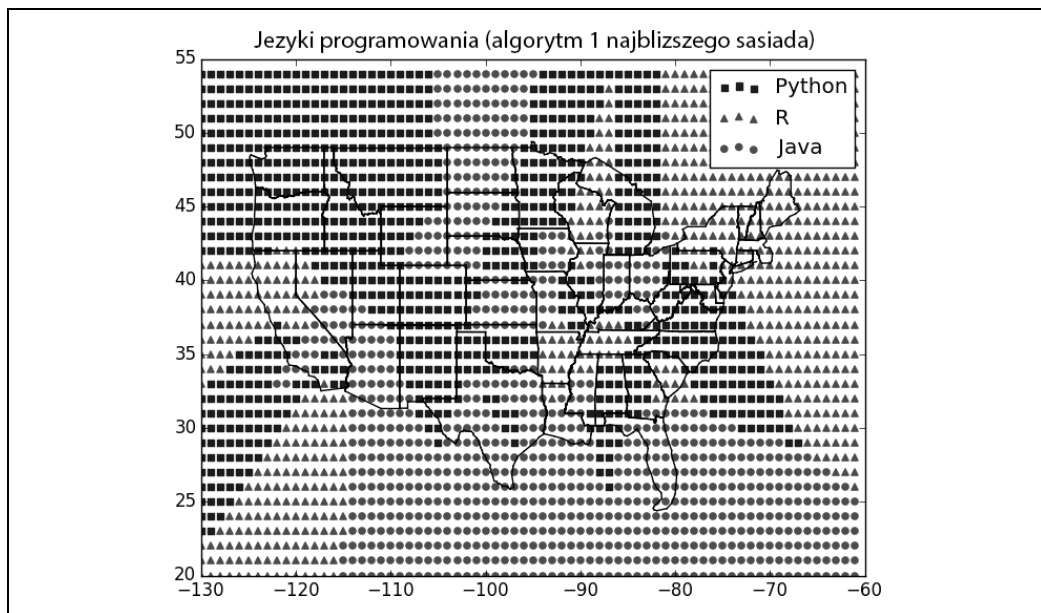
Teraz możemy sprawdzić sposób, w jaki zostaną sklasyfikowane obszary w wyniku użycia każdej wartości parametru *k*. Zrobimy to, klasyfikując całą siatkę punktów, a następnie przedstawiając je na bazie wygenerowanego wcześniej wykresu miast:

```
plots = { "Java" : ([], []), "Python" : ([], []), "R" : ([], []) }

k = 1 # Lub k = 3, lub k = 5, lub...

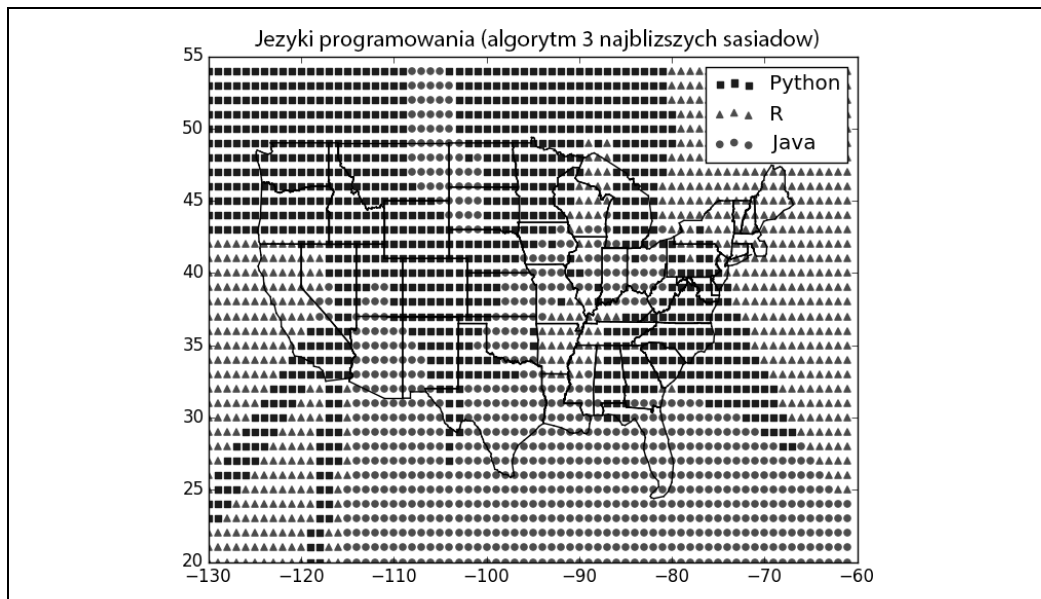
for longitude in range(-130, -60):
    for latitude in range(20, 55):
        predicted_language = knn_classify(k, cities, [longitude, latitude])
        plots[predicted_language][0].append(longitude)
        plots[predicted_language][1].append(latitude)
```

Na rysunku 12.2 pokazano graficzną interpretację tego, co stanie się po przyjęciu parametru *k* równego 1 (przyglądamy się tylko jednemu najbliższemu sąsiadowi).



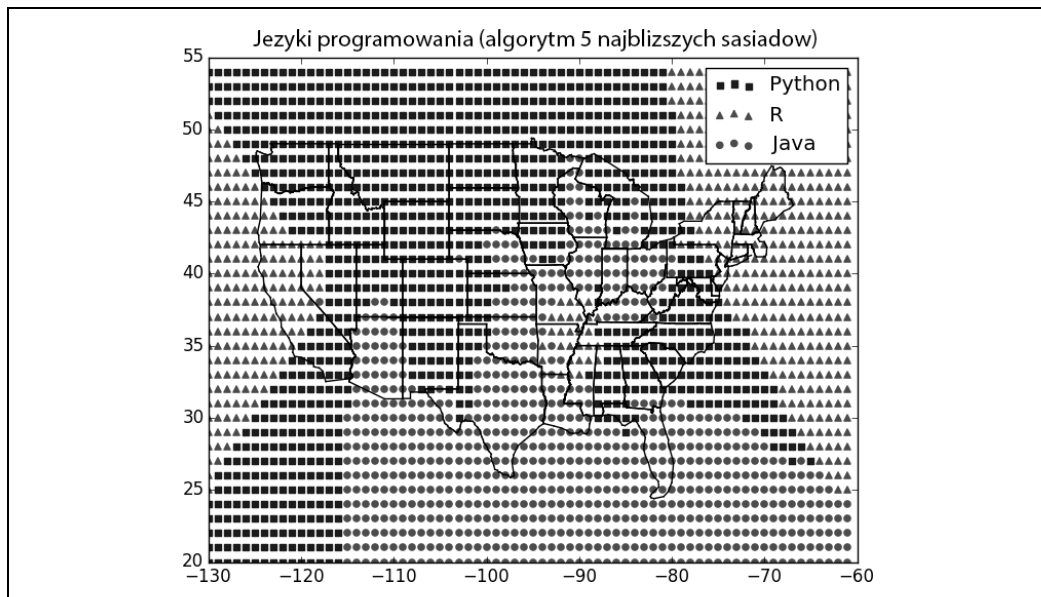
Rysunek 12.2. Języki programowania (algorytm 1 najbliższego sąsiada)

Widocznych jest wiele gwałtownych zmian języka — granice są ostre. Wzrost liczby analizowanych sąsiadów powoduje wygładzenie granic pomiędzy poszczególnymi językami (zobacz rysunek 12.3).



Rysunek 12.3. Języki programowania (algorytm 3 najbliższych sąsiadów)

Po zwiększeniu liczby analizowanych sąsiadów do pięciu uzyskujemy jeszcze bardziej gładkie granice (zobacz rysunek 12.4).



Rysunek 12.4. Języki programowania (algorytm 5 najbliższych sąsiadów)

W tym przypadku dane poszczególnych wymiarów można ze sobą porównywać, ale gdyby nie dało się ich porównywać, to należałoby wówczas skorzystać z technik skalowania opisanych w rozdziale 10., w podrozdziale „Przeskalowanie”.

Przekleństwo wymiarowości

Tzw. przekleństwo wymiarowości sprawia, że algorytm k najbliższych sąsiadów ma problemy z przetwarzaniem danych o większej liczbie wymiarów. Wynika to z *ogromu* przestrzeni wielowymiarowych. Punkty w przestrzeniach wielowymiarowych są zwykle od siebie bardzo oddalone. Można to sprawdzić poprzez generowanie losowych par punktów znajdujących się w d-wymiarowej przestrzeni, a następnie pomiar dzielących je odległości.

Generowanie wartości losowych nie powinno już sprawiać Ci problemów:

```
def random_point(dim):
    return [random.random() for _ in range(dim)]
```

To samo dotyczy tworzenia funkcji obliczającej odległość:

```
def random_distances(dim, num_pairs):
    return [distance(random_point(dim), random_point(dim))
            for _ in range(num_pairs)]
```

Dla każdej liczby wymiarów (od 1 do 100) będziemy obliczać 10 000 wartości odległości, które zostaną użyte w celu określenia średniej odległości pomiędzy punktami, a także minimalnej odległości pomiędzy punktami przy danej liczbie wymiarów (zobacz rysunek 12.5):

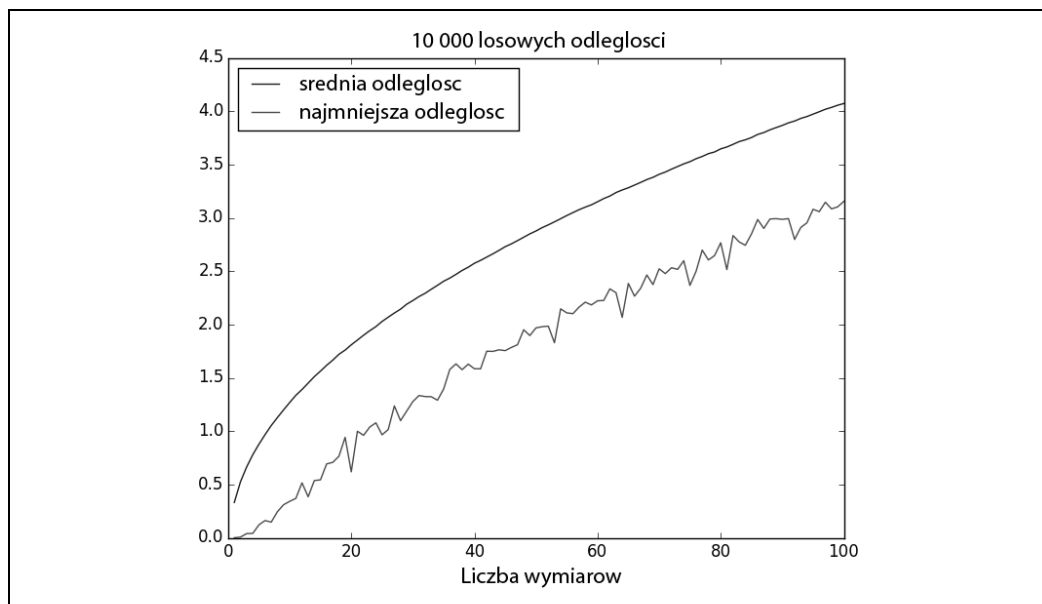
```
dimensions = range(1, 101)

avg_distances = []
min_distances = []

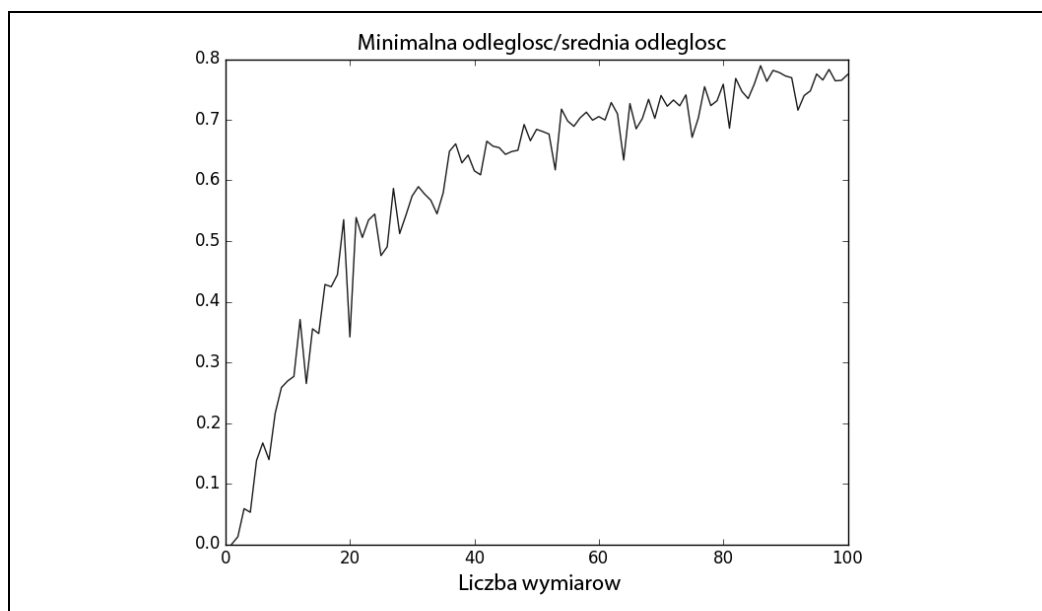
random.seed(0)
for dim in dimensions:
    distances = random_distances(dim, 10000) # 10,000 losowych par.
    avg_distances.append(mean(distances))    # Określ wartość średnią.
    min_distances.append(min(distances))     # Określ wartość najmniejszą.
```

Wzrost liczby wymiarów sprawia, że średnia odległość pomiędzy punktami wzrasta, ale największym problemem jest zmiana stosunku pomiędzy najmniejszą odległością i średnią odległością (zobacz rysunek 12.6):

```
min_avg_ratio = [min_dist / avg_dist
                  for min_dist, avg_dist in zip(min_distances, avg_distances)]
```



Rysunek 12.5. Przebieg średniej odległości



Rysunek 12.6. Kolejne skutki przekleństwa wymiarowości

W zbiorach danych o małej liczbie wymiarów najbliższe punkty są zwykle do siebie bardziej zbliżone. Dwa punkty znajdują się blisko siebie, jeżeli są bliskie na płaszczyźnie każdego wymiaru. Dodanie do zbioru każdego kolejnego wymiaru (nawet takiego, który zawiera tylko szum) może doprowadzić do zwiększenia odległości pomiędzy punktami. W przypadku dużej liczby wymiarów punkty reprezentujące elementy zbioru danych charakteryzują się zwykle większą średnią odległością od siebie, a więc bliskość dwóch punktów nie jest tak wyraźna jak w przypadku zbioru o mniejszej liczbie wymiarów, o ile dane nie zawierają wielu struktur, które sprawiają, że zbiór można analizować tak, jakby składał się z mniejszej liczby wymiarów.

Problem ten można również powiązać z gęstością przestrzeni o większej liczbie wymiarów.

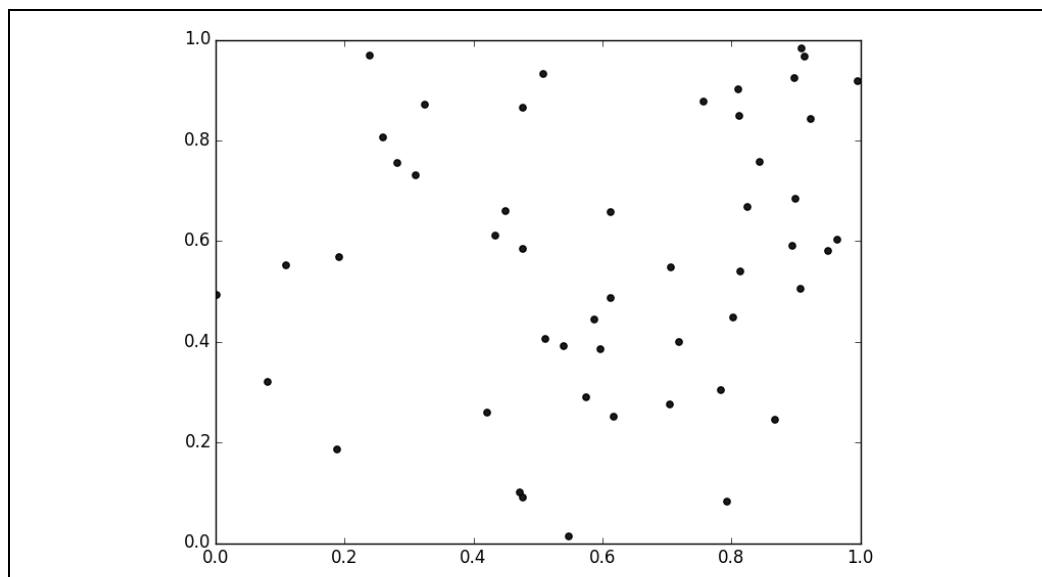
Jeżeli wybierzemy 50 losowych liczb z zakresu od 0 do 1, to prawdopodobnie uzyskamy dość dobrą próbkę interwału jednostkowego (zobacz rysunek 12.7).



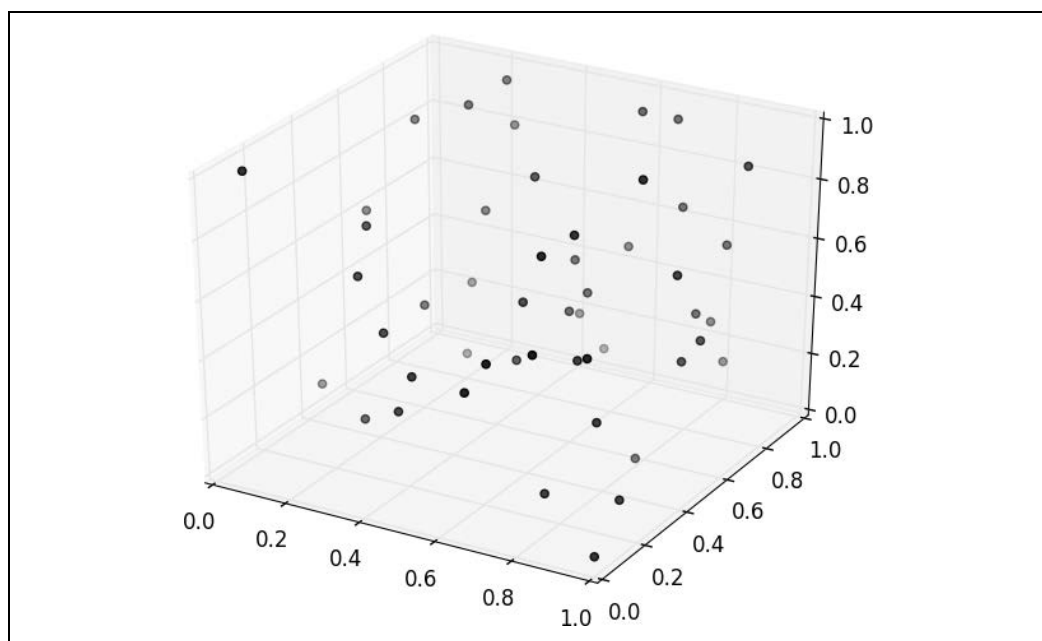
Rysunek 12.7. Pięćdziesiąt losowych wartości w przestrzeni jednowymiarowej

Jeżeli w przestrzeni dwuwymiarowej wybierzemy 50 losowo rozmieszczonych punktów, to będą one zajmowały mniejszy obszar przestrzeni (zobacz rysunek 12.8).

Pięćdziesiąt punktów umieszczonych w losowych miejscach przestrzeni trójwymiarowej będzie od siebie oddalonych jeszcze bardziej (zobacz rysunek 12.9).



Rysunek 12.8. Pięćdziesiąt punktów rozmieszczonych losowo w przestrzeni dwuwymiarowej



Rysunek 12.9. Pięćdziesiąt punktów rozmieszczonych losowo w przestrzeni trójwymiarowej

Pakiet `matplotlib` nie tworzy zbyt dobrych wizualizacji przestrzeni czterowymiarowych, a więc nie będę generował bardziej rozbudowanych wykresów, ale z trzech przedstawionych dotychczas wykresów wynika, że wzrost liczby wymiarów zwiększa ilość pustej przestrzeni pomiędzy wartościami. Wzrost liczby wymiarów bez wykładniczego przyrostu liczby punktów doprowadza do wzrostu pustych przestrzeni pomiędzy punktami, które mają być użyte do generowania przewidywań.

Jeżeli chcesz korzystać z algorytmu k najbliższych sąsiadów w przestrzeniach wielowymiarowych, to lepiej jest zacząć pracę z takimi danymi od zastosowania jednej z technik redukcji wymiarów.

Dalsza eksploracja

- Pakiet `scikit-learn` zawiera implementacje wielu modeli algorytmów najbliższych sąsiadów (<http://bit.ly/1ycP5rj>).

A

agregacja bootstrapowa, 204
algebra liniowa, 55
algorytm
 ID3, 200
 k najbliższych sąsiadów, 149
 MapReduce, 275, 277
 spadku gradientowego, 172
analiza
 danych, 13
 LDA, 237
 sieci społecznościowych, 243
 treści statusów, 278
API, 115
argumenty nazwane i nienazwane, 42

B

baza danych, 263
 NoSQL, 274
białe znaki, 26
biblioteka
 D3.js, 52
 gensim, 241
 mrjob, 282
 Natural Language Toolkit, 241
 NetworkX, 253
 NumPy, 284
 pandas, 120
 PyBrain, 216
 Pylearn2, 216
 scikit-learn, 23
 Scrapy, 120
 Twython, 119
błąd typu 1, 87
błędy standardowe współczynników regresji, 180

C

CAPTCHA, 213
cechy, 146
centralne twierdzenie graniczne, 81
centralność, 250
 wektorów własnych, 247
chmura wyrazowa, 230
ciągły rozkład prawdopodobieństwa, 77
currying, 40

D

dane uwierzytelniające, 118
data science, 9
defaultdict, 32
dobór rozmiaru kroku, 101
dopasowanie modelu, 140, 177
 nadmierne, 140
 zbyt słabe, 141
drzewa
 decyzyjne, 195
 klasyfikacyjne, 196
 regresyjne, 196
dykretny rozkład prawdopodobieństwa, 77
dyspersja, 66
dystrybuanta, 78

E

eksploracja danych, 121, 140
 jednowymiarowych, 121
ekstrakcja cech, 146
entropia, 197
 podziału, 198
enumerate, 41

F

filtr antyspamowy, 161, 162
filtrowanie kolaboratywne, 257, 260
format JSON, 115
formatowanie kodu, 26
funkcja, 28
 Counter, 33
 csv.DictReader, 109
 csv.writer, 109
 gęstości prawdopodobieństwa, 77
 gradient, 97
 logistyczna, 187
 make_hist, 82
 partial, 40
 plt.plot, 50
 sorted, 36
 sys.stdin, 105
 zip, 41
funkcje składowe, 39

G

generator, 37
Gephi, 253
gęstość centralności, 247
gradient, 97, 101
grafy skierowane, 251
gramatyka, 234
Graphlab, 261
grupowanie, 217
 hierarchiczne, 224
 kolorów, 222

H

Hadoop, 281
hakowanie wartości p, 90
hiperpłaszczyzna, 191, 192
hipoteza zerowa, 85, 91
histogram, 48
 rozkładu normal, 123
HTML, 110

I

iloczyn skalarny, 57, 58
indeksy, 272
interfejs programistyczny, 115, 117

interpreter, 25
IPython, 283
istotność, 87
iteracja, 41
iterator, 37

J

jednokierunkowe sieci neuronowe, 209
język R, 286

K

klasa, 39
klasyfikator, 163
kod źródłowy stron, 112
korekta ciągłości, 88
korelacja, 67, 68, 71
kowariancja, 68
krok, 101
krotka, tuple, 31

L

lasy losowe, 204
LDA, Latent Dirichlet Analysis, 237
liczba wymiarów, 132
liczenie słów, 275
licznik, 33
lista, list, 29
logarytm prawdopodobieństwa, 188
losowość, 38

Ł

ładowanie wstępne, 179
łamanie zabezpieczenia CAPTCHA, 213
łańcuch, 29

M

macierz, 59
 korelacji, 124
 kwadratowa, 249
 mnożenie, 248, 280
 wykresów punktowych, 124
maszyny wektorów nośnych, 190
mechanizm łączący, 281
mediana, 65

metoda

gradientu prostego, 97, 103

k średnich, 218

klasyfikacji najbliższych sąsiadów, 149

najmniejszych kwadratów, 170

PageRank, 251

sort, 36

mnożenie macierzy, 248, 280

moc testu, 87

model

biznesowy, 139

nadmiernie dopasowany, 142

najmniejszych kwadratów, 176

n-gram, 231

szumu, 141

modele parametryczne, 140

modelowanie, 139

predyktywne, 140

tematu, 237

moduł, 27

random, 38

scipy.stats, 83

Statsmodels, 183

wektora, 58

N

nadmierne dopasowanie, 140

naiwny klasyfikator bayesowski, 161

narzędzia funkcyjne, 40

narzędzie

Gephi, 253

Graphlab, 261

nauka o danych, 9

Netflix Prize, 261

niedomiar zmiennoprzecinkowy, 162

NLP, natural language procession, 229

NoSQL, 274

O

obiekt

JSON, 115

pliku, 107

oczyszczanie danych, 126

odchylenie standardowe, 67

operacje arytmetyczne, 28

optymalizacja zapytań, 273

P

pakiet

ggplot, 53

matplotlib, 45, 285

NumPy, 61

pandas, 72, 285

scikit-learn, 104, 137, 159, 183

SciPy, 72, 228

seaborn, 52

skicit-learn, 167, 285

StatsModels, 72

paradoks Simpsona, 70

parametr k , 221

parsowanie, 110

zdań, 235

perceptron, 207

pliki

binarne, 108

tekstowe, 107

pobieranie danych ze stron, 109

pochodne cząstkowe, 100

podobieństwo kosinusowe, 257

polecenie

CREATE TABLE, 263

DELETE, 265

except, 29

GROUP BY, 267

INSERT, 263

JOIN, 270

open, 107

ORDER BY, 269

SELECT, 266

try, 29

UPDATE, 265

polskie znaki diakrytyczne, 28

poprawność, 143

dopasowania, 178, 190

pośrednictwo, 243

prawdopodobieństwo, 73, 172

warunkowe, 74

programowanie obiektowe, 39

projekt Bokeh, 53

propagacja wsteczna, 211

próbkiowanie Gibbsa, 236

przedział ufności, 89

przekleństwo wymiarowości, 155

przepływ sterowania, 34

przeskalowanie, 130

przetwarzanie
 danych, 126, 127
 języka naturalnego, NLP, 229
przyczynowość, 71

R

redukcja liczby wymiarów, 132
regresja
 grzbietowa, 182
 lasso, 183
 liniowa, 169
 logistyczna, 185
 wieloraka, 175
regularyzacja, 181
rekomendacja, 255
relacyjna baza danych, 263
rozkład
 a posteriori, 92, 94
 a priori, 92
 beta, 92
 jednostajny dyskretny, 77
 normalny, 79
 prawdopodobieństwa, 77
 t-Studenta, 91, 181
rozpakowywanie argumentów, 41

S

selekcja cech, 146
serializacja, 115
serwis DataSciencester, 14
sieci społecznościowe, 243
sieć neuronowa, 207
 jednokierunkowa, 209
skalar, 55
składanie list, 36
słownik, dictionary, 15, 31
słowo kluczowe
 def, 28
 key, 36
sortowanie, 36
sprawdzanie hipotez, 85
SQL, 263
statystyka, 63
stochastyczna metoda gradientu prostego, 103
stopień
 centralności, 244
 pośrednictwa, 244

struktura danych dict, 15
strumień
 stdin, 105
 stdout, 105
systemy rekomendujące, 255
szacowanie gradientu, 98
szacowanie maksymalnego
 prawdopodobieństwa, 172
sztuczka z funkcją jądra, 192

Ś

ścieżki decyzyjne, 195
średnia, 64
 harmoniczna, 144

T

tendencje centralne, 64
test
 A-B, 91
 jednostronny, 87
testowanie modelu, 165
transpozycja macierzy, 260
twierdzenie Bayesa, 75
tworzenie
 drzewa decyzyjnego, 199
 kodu, 26
 systemów rekomendacji, 261

U

uczenie
 częściowo nadzorowane, 140
 maszynowe, 139
 nadzorowane, 140
 nienadzorowane, 140
uwierzytelnianie, 116

W

wariancja, 145
wartości
 binarne, 143
 logiczne, 35
wartość progowa, 145
wczytywanie plików, 107
wektor, 55
 własny macierzy, 249

Z

- węzły
 - decyzyjne, 200
 - końcowe, 200
- wizualizacja danych, 45, 285
- wnioskowanie bayesowskie, 92
- współczynnik
 - centralności, 17
 - determinacji, 171
 - przewidywania, 144
 - R kwadrat, 171
- wstępne
 - ładowanie, 179
 - przetwarzanie danych, 126
- wyjątki, 29
- wykres
 - liniowy, 50
 - punktowy, 51
 - słupkowy, 47
- wyrażenia regularne, 38

- zapytania składowe, 272
- zasady tworzenia kodu, 26
- zbiór, set, 33, 63
- zdarzenia, 73
 - niezależne, 73
 - zależne, 73
- zespawanie, 281
- zmienna
 - fikcyjna, 175
 - losowa, 77
 - składowa, 70

PROGRAM PARTNERSKI

— GRUPY HELION —



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

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA
Helion 

Python. Wyciśniesz z danych każdą kroplę wiedzy!

Współczesne ogromne zbiory danych zawierają odpowiedzi na prawie każde pytanie. Równocześnie nauka o danych jest dziedziną, która cokolwiek onieśmiela. Znajduje się gdzieś pomiędzy subtelными umiejętnościami hakerskimi, twardą wiedzą z matematyki i ze statystyki a merytoryczną znajomością zagadnień z danej branży. Co więcej, dziedzina ta niezwykle dynamicznie się rozwija. Jednak trud włożony w naukę o danych niewątpliwie się opłaca: biegły analityk danych może liczyć na dobrze płatną, inspirującą i bardzo atrakcyjną pracę.

Dzięki tej książce opanujesz najważniejsze zagadnienia związane z matematyką i ze statystyką, będziesz także rozwijać umiejętności hakerskie. W ten sposób zyskasz podstawy pozwalające na rozpoczęcie przygody z analizą danych. Gruntownie zapoznasz się z niezbędnymi narzędziami i algorytmami. Pozwoli Ci to lepiej zrozumieć ich działanie. Poszczególne przykłady, którymi zilustrowano omawiane zagadnienia, są przejrzyste, dobrze opisane i zrozumiałe. Podczas lektury książki poznasz biblioteki, które umożliwią zaimplementowanie omówionych technik podczas analizy dużych zbiorów danych. Szybko się przekonasz, że aby zostać analitykiem danych, wystarczy odrobina ciekawości, sporo chęci, mnóstwo ciężkiej pracy i... ta książka.

Joel Grus — jest inżynierem oprogramowania, analitykiem danych i autorem świetnie sprzedających się książek. Obecnie zajmuje się pracą badawczą w Allen Institute for Artificial Intelligence w Seattle. Wcześniej był zatrudniony w firmie Google i kilku startupach. Mieszka w Seattle, gdzie regularnie uczestniczy w spotkaniach lokalnej społeczności analityków danych.

Najważniejsze zagadnienia:

- Praktyczne wprowadzenie do Pythona
- Podstawy algebry liniowej, statystyki i rachunku prawdopodobieństwa w analizie danych
- Podstawy uczenia maszynowego
- Implementacje algorytmów modeli, w tym naiwny klasyfikator bayesowski, regresja liniowa, regresja logistyczna, drzewa decyzyjne, sieci neuronowe i grupowanie, MapReduce
- Systemy rekomendacji i mechanizmy przetwarzania języka naturalnego

Helion 

 helion.pl

 **HELION SA**
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

Sprawdź nasze szkolenia!

SZKOLENIA



AKADEMIA IT & BUSINESS

WWW.SZKOLENIA.HELION.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-283-4602-4



9 788328 346024