Uczenie maszynowe z językiem JavaScript

Rozwiązywanie złożonych problemów

Packt

Burak Kanber

Tytuł oryginału: Hands-on Machine Learning with JavaScript:
Solve complex computational web problems using machine learning

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-5196-7

Copyright © Packt Publishing 2018. First published in the English language under the title 'Hands-on Machine Learning with JavaScript – (9781788998246)'

Polish edition copyright © 2019 by Helion SA

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 Helion SA 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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/umasjs
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

O autorze 7

O recenzencie 8

Wstęp 9

Rozdział 1. Poznawanie potencjału języka JavaScript 15

Język JavaScript 15
Uczenie maszynowe 18
Zalety i wyzwania związane ze stosowaniem języka JavaScript 20
Inicjatywa CommonJS 21
Node.js 22
Język TypeScript 24
Usprawnienia wprowadzone w ES6 26
Let i const 26
Klasy 27
Importowanie modułów 29
Funkcje strzałkowe 29
Literały obiektowe 31
Funkcja for…of 31
Obietnice 32
Funkcje async/wait 33
Przygotowywanie środowiska programistycznego 34
Instalowanie Node.js 34
Opcjonalne zainstalowanie Yarn 35
Tworzenie i inicjowanie przykładowego projektu 35
Tworzenie projektu „Witaj, świecie!” 36
Podsumowanie 38

Kup ksi k Pole ksi k

Rozdział 2. Badanie danych 39

Przetwarzanie danych 39
Identyfikacja cech 42
Przekleństwo wymiarowości 43
Wybór cech oraz wyodrębnianie cech 45
Przykład korelacji Pearsona 48
Czyszczenie i przygotowywanie danych 51
Obsługa brakujących danych 51
Obsługa szumów 53
Obsługa elementów odstających 58
Przekształcanie i normalizacja danych 61
Podsumowanie 68

Rozdział 3. Przegląd algorytmów uczenia maszynowego 69

Wprowadzenie do uczenia maszynowego 70
Typy uczenia 70
Uczenie nienadzorowane 72
Uczenie nadzorowane 75
Uczenie przez wzmacnianie 83
Kategorie algorytmów 84
Grupowanie 84
Klasyfikacja 84
Regresja 85
Redukcja wymiarowości 85
Optymalizacja 86
Przetwarzanie języka naturalnego 86
Przetwarzanie obrazów 87
Podsumowanie 87

Rozdział 4. Algorytmy grupowania na podstawie klastrów 89

Średnia i odległość 90
Pisanie algorytmu k-średnich 93
Przygotowanie środowiska 93
Inicjalizacja algorytmu 94
Testowanie losowo wygenerowanych centroidów 99
Przypisywanie punktów do centroidów 100
Aktualizowanie położenia centroidów 102
Pętla główna 106
Przykład 1. — k-średnich na prostych danych dwuwymiarowych 107
Przykład 2. — dane trójwymiarowe 114
Algorytm k-średnich, kiedy k nie jest znane 116
Podsumowanie 122

Kup ksi k

Pole ksi k

Rozdział 5. Algorytmy klasyfikacji 123

k najbliższych sąsiadów 124
Implementacja algorytmu KNN 125
Naiwny klasyfikator bayesowski 138
Tokenizacja 140
Implementacja algorytmu 141
Przykład 3. — ocenianie charakteru recenzji filmów 150
Maszyna wektorów nośnych 154
Lasy losowe 162
Podsumowanie 168

Rozdział 6. Algorytmy reguł asocjacyjnych 169

Z matematycznego punktu widzenia 171
Z punktu widzenia algorytmu 174
Zastosowania reguły asocjacji 176
Przykład — dane ze sprzedaży detalicznej 178
Podsumowanie 182

Rozdział 7. Przewidywanie z użyciem algorytmów regresji 183

Porównanie regresji i klasyfikacji 184
Podstawy regresji 185
Przykład 1. — regresja liniowa 189
Przykład 2. — regresja wykładnicza 193
Przykład 3. — regresja wielomianowa 198
Inne techniki analizy szeregów czasowych 200
Filtrowanie 201
Analiza sezonowości 203
Analiza fourierowska 204
Podsumowanie 206

Rozdział 8. Algorytmy sztucznych sieci neuronowych 209

Opis koncepcji sieci neuronowych 210
Uczenie metodą propagacji wstecznej 214
Przykład — XOR z użyciem TensorFlow.js 217
Podsumowanie 224

Rozdział 9. Głębokie sieci neuronowe 227

Konwolucyjne sieci neuronowe 228
Konwolucje oraz warstwy konwolucyjne 229
Przykład — zbiór MNIST ręcznie zapisanych cyfr 234
Rekurencyjne sieci neuronowe 241
SimpleRNN 242
Topologia GRU 246
Długa pamięć krótkoterminowa — LSTM 249
Podsumowanie 252

Kup ksi k

Pole ksi k

Rozdział 10. Przetwarzanie języka naturalnego w praktyce 253

Odległość edycyjna 255
Ważenie termów — odwrotna częstość w dokumentach 257
Tokenizacja 263
Stemming 270
Fonetyka 272
Oznaczanie części mowy 274
Techniki przekazywania słów do sieci neuronowych 276
Podsumowanie 279

Rozdział 11. Stosowanie uczenia maszynowego w aplikacjach czasu rzeczywistego 281

Serializacja modeli 282
Uczenie modeli na serwerze 283
Wątki robocze 286
Modele samodoskonalące oraz spersonalizowane 287
Potokowanie danych 290
Przeszukiwanie danych 291
Łączenie i agregacja danych 293
Przekształcenia i normalizacja 295
Przechowywanie i dostarczanie danych 298
Podsumowanie 300

Rozdział 12. Wybieranie najlepszego algorytmu dla aplikacji 303

Tryb uczenia 305
Zadanie do wykonania 308
Format, postać, wejście i wyjście 309
Dostępne zasoby 312
W razie problemów 313
Łączenie modeli 316
Podsumowanie 318

Skorowidz 321

Kup ksi k Pole ksi k

1

ROZDZIAŁ

Poznawanie potencjału języka JavaScript

W tym rozdziale zostaną opisane następujące zagadnienia:

- język JavaScript;
- uczenie maszynowe;
- zalety i wyzwania związane ze stosowaniem języka JavaScript;
- inicjatywa CommonJS;
- środowisko Node.js;
- język TypeScript;
- usprawnienia wprowadzone w ES6;
- przygotowywanie środowiska do programowania.

Język JavaScript

Zaczynałem pisać o *uczeniu maszynowym* (ang. *machine learning* — ML) w języku JavaScript w 2010 r. W tamtym czasie środowisko Node.js było czym całkowicie nowym, a JavaScript dopiero zaczynał udowadniać swoją przydatność jako język programowania. Przez długi okres historii internetu był postrzegany jako raczej mało poważny język, używany jedynie do tworzenia prostych, dynamicznych interakcji na stronach WWW.

Sposób postrzegania języka JavaScript zaczął się zmieniać w roku 2005, wraz z udostępnieniem frameworka *Prototype*, opracowanego w celu ułatwienia stosowania żądań AJAX i uproszczenia obsługi obiektu `XMLHttpRequest` w różnych przeglądarkach. Framework *Prototype* wprowadził znaną notację wykorzystującą znak `$`, w której np. wyrażenie `document.getEle↪mentById("myId")` można było zastąpić wyrażeniem `$("myId")`.

Kup ksi k

Pole ksi k

Rok później John Resig udostępnił swoją niezwykle popularną bibliotekę jQuery. Jak podaje serwis *https://w3techs.com*, biblioteka jQuery jest obecnie używana w 97% witryn WWW, których zestaw bibliotek jest znany serwisowi (to ok. 73% wszystkich istniejących witryn WWW). Biblioteka jQuery została opracowana po to, by najczęściej stosowane operacje w języku JavaScript można było wykonywać łatwo i tak samo we wszystkich przeglądarkach, udostępniając dla każdego programisty bardzo ważne narzędzia, takie jak obsługa żądań AJAX, przeglądanie i manipulowanie DOM (ang. *Document Object Model* — obiektowy model dokumentu) oraz animacje.

Później, w 2008 r., została udostępniona przeglądarka Chrome oraz stosowany w niej silnik języka JavaScript — V8. Zarówno Chrome, jak i V8 stanowiły znaczący postęp w wydajności działania JavaScriptu w porównaniu z innymi przeglądarkami: JavaScript stał się szybki, co w głównej mierze udało się osiągnąć dzięki zastosowaniu innowacyjnego kompilatora *just-in-time*, który generuje kod maszynowy bezpośrednio na podstawie kodu JavaScriptu.

Popularność języka JavaScript wzrosła jeszcze bardziej, kiedy jQuery i Chrome podbiły internet. Wcześniej programiści nie przepadali za JavaScriptem jako językiem programowania, jednak po pojawieniu się biblioteki jQuery i dzięki szybkim przeglądarkom stało się jasne, że JavaScript był narzędziem niedocenianym i dysponującym możliwościami, które wcześniej nie były w pełni wykorzystywane.

W 2009 r. społeczność programistów JavaScriptu postanowiła wyzwolić go ze środowiska przeglądarek WWW, w których działał do tej pory. Na początku tamtego roku została powołana inicjatywa CommonJS, a kilka miesięcy później — projekt Node.js. Celem modułów CommonJS było opracowanie standardowej biblioteki i poprawa ekosystemu języka JavaScript w taki sposób, by można go było używać poza środowiskiem przeglądarek WWW. Jednym z kierunków działań CommonJS było utworzenie standaryzowanego interfejsu wczytywania modułów, który pozwoliłby programistom na tworzenie bibliotek i udostępnianie ich innym.

Udostępnienie środowiska Node.js w połowie 2009 r. stało się nowym impulsem dla rozwoju całego świata języka JavaScript, udostępniając programistom używającym tego języka zupełnie nowy paradygmat: JavaScript jako język działający po stronie serwera. Zastosowanie w Node.js silnika V8 sprawiło, że środowisko to działało niezwykle szybko, choć nie tylko silnik V8 przyczynił się do jego wysokiej wydajności. W Node.js do przetwarzania żądań używana jest pętla obsługi zdarzeń, dzięki czemu, choć środowisko Node.js jest jednowątkowe, może obsługiwać duże ilości jednoczesnych żądań.

Nowości, jakimi były możliwość używania JavaScriptu po stronie serwera, jego zaskakująca wydajność oraz wczesne udostępnienie rejestru modułów npm, pozwalającego programistom na tworzenie modułów, ich publikowanie i poszukiwanie, stały się czynnikami, które przyciągnęły tysiące programistów. Standardowe biblioteki udostępniane wraz z Node.js ograniczały się głównie do niskopoziomowych bibliotek wejścia-wyjścia, więc programiści zaczęli ścigać się, kto stworzy pierwszą dobrą bibliotekę do obsługi żądań HTTP, pierwszy łatwy w obsłudze serwer HTTP, pierwszą bibliotekę wysokiego poziomu do przetwarzania obrazów itd. Błyskawiczny rozwój ekosystemu języka JavaScript przyczynił się z kolei do wzrostu zaufania wśród programistów, którzy wcześniej nie byli chętni do wykorzystywania

Kup ksi k Pole ksi k

tej technologii. Po raz pierwszy w swej historii JavaScript zaczął być traktowany jako prawdziwy język programowania, a nie jako coś, co tolerowano, bo było używane w przeglądarkach WWW.

W czasie gdy środowisko JavaScriptu powoli dojrzewało, społeczność użytkowników języka Python, zainspirowana sukcesami firmy Google, pracowała nad zagadnieniami uczenia maszynowego. W 2006 r. została udostępniona podstawowa i niezwykle popularna biblioteka obliczeniowa NumPy, choć w takiej czy innej formie istniała ona już od jakiegoś czasu. Biblioteka do uczenia maszynowego o nazwie *scikit-learn* została udostępniona w 2010 r. i właśnie w tym momencie zdecydowałem się edukować programistów JavaScriptu w kwestii zagadnień związanych z uczeniem maszynowym.

Popularność uczenia maszynowego w języku Python oraz łatwość tworzenia i uczenia modeli przy wykorzystaniu narzędzi takich jak *scikit-learn* zaskoczyła zarówno mnie, jak i wiele innych osób. Na moich oczach fala popularności uniosła balonik uczenia maszynowego — zauważyłem, że ze względu na wielką łatwość budowania i uruchamiania modeli wielu programistów przestało rozumieć mechanikę działania używanych algorytmów i technik. Uskarżali się oni na niewystarczająco działające modele, nie zdając sobie sprawy z tego, że to oni sami stanowią słabe ogniwo w łańcuchu.

W tamtym czasie uczenie maszynowe było postrzegane jako coś mistycznego, magicznego, akademickiego, dostępnego tylko dla garstki geniuszy i programistów Pythona. Ja miałem jednak zupełnie inną opinię na ten temat. Uczenie maszynowe to jedynie kategoria algorytmów i nie ma w niej nic magicznego. Większość z tych algorytmów jest w rzeczywistości całkiem prosta, i to nie bez przyczyny!

Zamiast pokazywać programistom, jak zaimportować klasyfikator bayesowski w Pythonie, wolałem wyjaśnić im, jak napisać ten algorytm od podstaw, i pomóc w zrobieniu ważnego kroku na drodze do wykształcenia intuicji. Pragnąłem także, by moi studenci w przeważającym stopniu zignorowali popularne biblioteki języka Python istniejące już w tamtym czasie — chciałem w ten sposób podkreślić, że algorytmy uczenia maszynowego można implementować w dowolnym języku programowania, a nie tylko w Pythonie.

Jako platformę do nauczania wybrałem język JavaScript. Szczerze mówiąc, wybrałem go, gdyż wiele osób w tamtym czasie uważało JavaScript za *zły* język programowania. Moje przesłanie było następujące: *Uczenie maszynowe jest proste, można je implementować nawet w JavaScripcie!* Na moje szczęście zarówno Node.js, jak i JavaScript błyskawicznie zyskiwały na popularności, a w kolejnych latach moje pierwsze artykuły na temat uczenia maszynowego w JavaScripcie zostały przeczytane przez miliony zainteresowanych programistów.

Po części wybrałem JavaScript także dlatego, że nie chciałem, by uczenie maszynowe było postrzegane jako narzędzie wykorzystywane jedynie przez wykładowców, naukowców zajmujących się informatyką czy też absolwentów uczelni. Wierzyłem, i wciąż wierzę, że przy odpowiedniej ilości praktyki i ćwiczeń algorytmy te mogą być dogłębnie zrozumiane przez każdego kompetentnego programistę. Wybrałem JavaScript, gdyż pozwalał mi on dotrzeć do nowych odbiorców — grup programistów zajmujących się tworzeniem interfejsów użytkownika oraz całościowym tworzeniem aplikacji — spośród których bardzo wiele osób to

Kup ksi k Pole ksi k

samoucy, którzy nigdy formalnie nie ukończyli studiów informatycznych. Skoro moim celem była demistyfikacja i popularyzacja zagadnień związanych z uczeniem maszynowym, czułem, że znacznie lepszym pomysłem będą próby dotarcia do społeczności programistów aplikacji internetowych niż do środowiska programistów używających Pythona, które w tamtym czasie znacznie lepiej znało się na uczeniu maszynowym.

Python od zawsze był (i wciąż jest) najpopularniejszym językiem do uczenia maszynowego, po części ze względu na swoją dojrzałość jako język programowania, po części ze względu na dojrzałość jego ekosystemu, a po części ze względu na sukcesy początkowych wysiłków związanych z implementacją uczenia maszynowego w tym języku. Jednak najnowszy rozwój JavaScriptu i jego ekosystemu sprawiają, że to właśnie JavaScript staje się bardziej atrakcyjny pod względem tworzenia rozwiązań dotyczących uczenia maszynowego. Uważam, że w ciągu kilku najbliższych lat zastosowanie JavaScriptu do uczenia maszynowego przeżyje renesans, zwłaszcza biorąc pod uwagę znaczący wzrost możliwości urządzeń mobilnych oraz popularności języka JavaScript.

Uczenie maszynowe

Kilka technik uczenia maszynowego istniało jeszcze, zanim pojawiły się komputery, jednak większość nowoczesnych, aktualnie używanych algorytmów uczenia maszynowego została opracowana w latach 70. i 80. ubiegłego wieku. W tamtym czasie były one interesujące, choć nie znalazły zastosowania praktycznego, dlatego też miały głównie charakter akademicki.

Co zatem przyczyniło się do tak poważnego wzrostu zainteresowania zagadnieniami uczenia maszynowego? Przede wszystkim komputery stały się w końcu na tyle potężne, by można na ich uruchamiać nietrywialne sieci neuronowe oraz modele ML. Nieco później wystąpiły jeszcze dwa kluczowe zdarzenia: stworzenie firmy Google oraz udostępnienie *Amazon Web Services* (AWS).

PageRank, algorytm ML obsługujący wyszukiwarkę Google, pokazał nam, jakie są możliwości stosowania uczenia maszynowego w biznesie. Sergey Brin i Larry Page, założyciele Google'a, udowodnili wszystkim, że za ogromnym sukcesem ich wyszukiwarki i współdziałającego z nią biznesu reklamowego stoi ten właśnie algorytm: stosunkowo proste równanie algebry liniowej, operujące na ogromnej macierzy.

> Warto zwrócić uwagę, że sieci neuronowe także są stosunkowo prostymi równaniami algebry liniowej operującymi na wielkiej macierzy.

To było uczenie maszynowe w całej swojej chwale: wielkie ilości danych zapewniające nieskrępowany wgląd w informacje i przekładające się na wielki sukces biznesowy. Dzięki nim świat zainteresował się uczeniem maszynowym pod kątem ekonomicznym.

Kup ksi k Pole ksi k

Po uruchomieniu EC2 oraz naliczania godzinowego platforma AWS zdemokratyzowała dostęp do zasobów obliczeniowych. Badacze i właściciele start-upów w początkowej fazie działalności zyskali możliwość szybkiego uruchamiania klastrów obliczeniowych, uczenia swoich modeli oraz ponownego skalowania klastrów w dół, unikając przy tym konieczności ponoszenia dużych nakładów finansowych na zakup i uruchamianie potężnych serwerów. Możliwości te przyczyniły się do wytworzenia nowego współzawodnictwa oraz początkowej generacji start-upów, produktów i inicjatyw bazujących na wykorzystaniu uczenia maszynowego.

Ostatnio uczenie maszynowe znowu zyskało na popularności, i to zarówno w środowiskach programistów, jak i biznesmenów. Pierwsza generacja start-upów i produktów bazujących na uczeniu maszynowym dojrzała już i wyraźnie pokazuje korzyści, jakie dla rynku może zapewniać uczenie maszynowe, a wiele z tych firm dogania lub już przegoniło konkurencję. Dążenie firm do zachowania konkurencyjności na rynku napędza zapotrzebowanie na rozwiązania korzystające z uczenia maszynowego.

Po koniec 2015 r. firma Google udostępniła bibliotekę **TensorFlow**, służącą do tworzenia i wykorzystywania sieci neuronowych, której wprowadzenie zdemokratyzowało wykorzystanie sieci neuronowych w podobnym stopniu jak wcześniej wprowadzenie EC2 zdemokratyzowało wykorzystanie mocy przetwarzania. Co więcej, także te start-upy pierwszej generacji, które koncentrowały się na programistach, dojrzały i obecnie przy użyciu prostego API możemy tworzyć żądania i przesyłać je na serwery AWS lub **Google Cloud Platform** (**GCP**), na których działają **konwolucyjne sieci neuronowe** (ang. *Convolutional Neural Network* — CNN) wyspecjalizowane w rozpoznawaniu obrazów, by dowiedzieć się, czy patrzymy na kota, kobietę, hamburgera, czy też na wszystkie te trzy rzeczy jednocześnie.

W miarę coraz to większego wykorzystania uczenia maszynowego jego wartość pod względem konkurencyjności będzie maleć — innymi słowy: firmy nie będą mogły już go używać po to, by zyskać znaczącą przewagę nad konkurentami, gdyż także oni będą stosować uczenie maszynowe. Obecnie wszyscy zajmujący się tą dziedziną korzystają z tych samych algorytmów, a konkurencja staje się wojną na dane. Jeśli wciąż chcemy konkurować w dziedzinie technologii, jeśli chcemy znaleźć rozwiązanie, które dziesięciokrotnie poprawi nasze obecne możliwości, to albo będziemy musieli na nie poczekać, albo, co byłoby nawet lepsze, musimy sami doprowadzić do kolejnego technologicznego przełomu.

Gdyby uczenie maszynowe nie przełożyło się na tak wielkie sukcesy rynkowe, byłby to zapewne koniec jego historii. Wszystkie ważne algorytmy byłyby powszechnie znane, a walka sprowadziłaby się do tego, kto będzie w stanie zebrać najlepsze dane, ogrodzić swój ogródek lub w najlepszy sposób wykorzystać ekosystem.

Jednak wszystko zmieniło się wraz z wprowadzeniem na rynek narzędzia TensorFlow. Teraz także dostęp do sieci neuronowych został znacznie ułatwiony. Obecnie zaskakująco łatwo można budować modele, uczyć je, uruchamiać na GPU i generować rzeczywiste wyniki. Mgiełka akademickiej tajemniczości okrywająca do tej pory sieci neuronowe rozwiała się, a tysiące programistów zaczęło bawić się udostępnionymi technikami, eksperymentować z nimi i ulepszać je. Doprowadzi to do drugiej wielkiej fali wzrostu popularności uczenia maszynowego, skupiającej się głównie wokół sieci neuronowych. Właśnie dziś rodzi się druga generacja start-upów korzystających z uczenia maszynowego i sieci neuronowych,

Kup ksi k Pole ksi k

które za kilka lat dojrzeją i okrzepną; można sądzić, że doczekamy się wtedy kolejnych przełomów oraz firm, którym udało się do nich doprowadzić.

Każdy rynkowy sukces, którego będziemy świadkami, doprowadzi do wzrostu zapotrzebowania na programistów zajmujących się zagadnieniami uczenia maszynowego. Poszerzanie się bazy utalentowanych programistów oraz powszechna dostępność technologii przyczyniają się do technologicznych przełomów. Każdy z takich przełomów wywiera wpływ na rynek i wiedzie do rynkowych sukcesów — ten cykl zamyka się i prowadzi do postępów następujących w coraz to szybszym tempie. Sądzę więc, że zdążamy w kierunku prawdziwego boomu **sztucznej inteligencji** (**AI**), i to z przyczyn ekonomicznych.

Zalety i wyzwania związane ze stosowaniem języka JavaScript

Bez względu na optymizm, z jakim zapatruję się na przyszłość wykorzystania technik uczenia maszynowego w języku JavaScript, większość programistów wciąż tworzy projekty w języku Python. Obecnie właściwie wszystkie duże produkcyjne systemy są tworzone właśnie w Pythonie lub w innych językach typowych dla uczenia maszynowego.

JavaScript, podobnie jak każde inne narzędzie, ma swoje zalety i wady. W przeszłości większość krytyki tego języka koncentrowała się na kilku zagadnieniach: dziwne zachowanie związane ze stosowaniem typów, model obiektowy bazujący na prototypach, problemy dotyczące organizacji dużej bazy kodu oraz zarządzanie głęboko zagnieżdżonymi wywołaniami asynchronicznymi (problem określany potocznie jako *piekło wywołań zwrotnych*, ang. *callback hell*). Na szczęście większość z tych historycznych problemów udało się rozwiązać dzięki wprowadzeniu **ES6**, czyli **ECSAScript 2015** — najnowszej aktualizacji składni języka JavaScript.

Niezależnie od tych ostatnich usprawnień większość programistów i tak wciąż opowiadałaby się przeciwko wykorzystaniu JavaScriptu w zastosowaniach związanych z uczeniem maszynowym z jednego podstawowego powodu: ekosystemu. Ekosystem związany z implementacją uczenia maszynowego w języku Python jest tak dojrzały i bogaty, że trudno usprawiedliwić wybór jakiegoś innego ekosystemu. Jednak taka logika zarówno sama się potwierdza, jak i sama sobie przeczy: jeśli chcemy, by ekosystem języka JavaScript w zakresie zagadnień uczenia maszynowego dojrzewał, to potrzeba nam odważnych osób, które będą go rozwijać i pracować nad rzeczywistymi problemami uczenia maszynowego. Na szczęście JavaScript już od kilku lat jest najpopularniejszym językiem programowania na GitHubie, a jego popularność wciąż rośnie, i to pod każdym względem.

Zastosowanie języka JavaScript w zagadnieniach uczenia maszynowego ma kilka zalet. Popularność JavaScriptu jest jedną z nich — choć implementacja uczenia maszynowego w JavaScripcie nie jest może obecnie zbyt popularna, to jednak sam język jest. Oczywiście wraz ze wzrostem zapotrzebowania na aplikacje korzystające z uczenia maszynowego, spadkiem ceny komponentów sprzętowych komputerów oraz wzrostem szybkości ich działania także popularność rozwiązań uczenia maszynowego w JavaScripcie stanie się znacznie większa.

Kup ksi k Pole ksi k

Istnieją tysiące zasobów poświęconych ogólnie nauce języka JavaScript, utrzymaniu serwerów działających na podstawie Node.js oraz wdrażaniu aplikacji napisanych w JavaScripcie. Ekosystem **Node Package Managera** (**npm**) jest ogromny i wciąż się rozwija, i choć nie znajdziemy w nim obecnie zbyt wielu dojrzałych pakietów związanych z uczeniem maszynowym, to jest tam sporo solidnie napisanych, użytecznych narzędzi, które zapewne niebawem pozytywnie przejdą próbę czasu.

Kolejną zaletą JavaScriptu jest jego uniwersalność. Nowoczesne przeglądarki WWW są w zasadzie przenośnymi platformami do wykonywania aplikacji, pozwalającymi na wykonywanie kodu na niemal dowolnych urządzeniach, i to w zasadzie bez konieczności wprowadzania w tym kodzie jakichkolwiek zmian. Narzędzia takie jak **Electron** (przez wielu uważane za zbytnio przeładowane) umożliwiają programistom szybkie tworzenie i wdrażanie aplikacji komputerowych, które można pobierać i uruchamiać w dowolnych systemach operacyjnych. Node.js pozwala na wykonywanie kodu w środowisku serwerowym. React Native pozwala przenosić kod JavaScriptu do rodzimego środowiska aplikacji mobilnych, a kiedyś może także sprawić, że będzie można tworzyć aplikacje dla komputerów biurkowych. JavaScript nie ogranicza się już do implementacji dynamicznych interakcji na stronach WWW, obecnie jest językiem programowania ogólnego przeznaczenia, który z powodzeniem można stosować na bardzo wielu platformach systemowych.

Wreszcie stosowanie JavaScriptu w rozwiązaniach uczenia maszynowego sprawia, że mogą się nimi zająć programiści tworzący interfejsy użytkownika aplikacji — grupa, która historycznie była pomijana w rozważaniach dotyczących uczenia maszynowego. Rozwiązania serwerowe są zazwyczaj preferowane jako narzędzia uczenia maszynowego, gdyż moc obliczeniowa jest dostępna właśnie na serwerach. Ten fakt sprawiał, że wcześniej zagadnienia uczenia maszynowego raczej nie były dostępne dla twórców aplikacji internetowych, jednak wraz z rozwojem sprzętu komputerowego nawet złożone modele ML mogą już być uruchamiane po stronie klienta niezależnie od tego, czy będzie to przeglądarka działająca na komputerze biurkowym, czy na urządzeniu mobilnym.

Jeśli obecnie zarówno programiści aplikacji internetowych, programiści zajmujący się interfejsami użytkownika, jak i programiści JavaScriptu zajmą się nauką zagadnień uczenia maszynowego, to cała ta społeczność zyska możliwość poprawiania narzędzi uczenia maszynowego, z których w przyszłości wszyscy będziemy mogli korzystać. Jeśli udostępnimy i rozpowszechnimy te technologie, wyjaśnimy pojęcia uczenia maszynowego możliwie jak największej liczbie osób, to w efekcie doprowadzimy do podniesienia poziomu wiedzy społeczności i wykształcimy kolejną generację badaczy zajmujących się uczeniem maszynowym.

Inicjatywa CommonJS

W 2009 r. Kevin Dangoor, inżynier fundacji Mozilla, doszedł do wniosku, że aby język JavaScript w zastosowaniach serwerowych odniósł sukces, potrzebna mu jest znacząca pomoc. Idea używania JavaScriptu po stronie serwera była już wówczas bardzo ekscytująca, jednak ze względu na liczne ograniczenia, zwłaszcza związane z ekosystemem języka, nie była jeszcze zbyt popularna.

Kup ksi k Pole ksi k

We wpisie opublikowanym na blogu w styczniu 2009 r. Dangoor przedstawił kilka przykładów tego, gdzie ta pomoc jest konieczna. Napisał, że ekosystem JavaScriptu będzie potrzebował standardowej biblioteki oraz standardowych interfejsów do wykonywania takich zadań jak operacje na plikach czy też dostęp do baz danych. Oprócz tego środowisko JavaScriptu potrzebowało sposobów na pakowanie, publikowanie i instalowanie bibliotek oraz rozwiązywanie zależności, jak również niezbędne było repozytorium pakietów, które zaspokajałoby wszystkie wymienione wcześniej potrzeby.

Efektem tych rozważań było powstanie inicjatywy **CommonJS**, której najważniejszym wkładem w ekosystem języka JavaScript jest format modułów CommonJS. Każdy, kto miał jakąkolwiek styczność z programowaniem w środowisku Node.js, najprawdopodobniej zna CommonJS: plik *package.json* jest zapisany właśnie w formacie specyfikacji pakietów opracowanym dla modułów CommonJS, a kod o przykładowej postaci `var app = require('./app.js')` w jednym pliku oraz `module.export = App` w pliku *app.js* korzysta ze specyfikacji modułów CommonJS.

Standaryzacja modułów i pakietów utorowała drogę do znacznego zwiększenia popularności języka JavaScript. Dzięki niej programiści mogli używać modułów do pisania złożonych aplikacji składających się z wielu plików, bez konieczności zaśmiecania globalnej przestrzeni nazw. Twórcy pakietów i bibliotek mogli pisać i publikować nowe biblioteki, operujące na wyższym poziomie abstrakcji niż standardowa biblioteka języka JavaScript. Node.js oraz npm szybko skorzystały z tego pomysłu i bazując na mechanizmie udostępniania pakietów, stworzyły wielki ekosystem.

Node.js

Udostępnienie Node.js, które nastąpiło w 2009 r., jest najprawdopodobniej najważniejszym momentem w historii języka JavaScript, choć nie mogłoby zaistnieć, gdyby nie udostępniona rok wcześniej przeglądarka Chrome oraz używany w niej silnik JavaScriptu — V8.

Ci spośród czytelników, którzy pamiętają moment pojawienia się przeglądarki Chrome, wiedzą zapewne, dlaczego zdominowała ona rynek przeglądarek WWW: była szybka, minimalistyczna, nowoczesna, pisanie aplikacji i rozszerzeń dla niej było łatwe, a kod JavaScriptu był w niej wykonywany szybciej niż w innych przeglądarkach WWW.

U podstaw Chrome leży otwarty projekt Chromium, w ramach którego został opracowany i stworzony silnik JavaScriptu V8. Innowacją, jaką silnik ten wprowadził do świata JavaScriptu, był nowy model wykonywania kodu: zamiast na bieżąco interpretować kod, silnik V8 zawiera kompilator JIT, który przekształca kod JavaScriptu bezpośrednio na rodzimy kod maszynowy. Takie rozwiązanie opłaciło się, a połączony efekt kosmicznej szybkości silnika V8 i status projektu otwartego sprawiły, że także inni twórcy zaczęli korzystać z silnika V8 do swoich celów.

Kup ksi k Pole ksi k

Twórcy Node.js zastosowali silnik V8, obudowali go architekturą sterowaną zdarzeniami i wyposażyli w niskopoziomową bibliotekę wejścia-wyjścia do obsługi dysków i plików. Krytyczną decyzją okazało się zastosowanie architektury sterowanej zdarzeniami. Inne języki i technologie serwerowe, takie jak PHP, zazwyczaj do obsługi równoczesnych żądań używają puli wątków, z których każdy jest blokowany na czas obsługi żądania. Node.js jest procesem jednowątkowym, jednak dzięki użyciu pętli zdarzeń pozwala uniknąć operacji powodujących blokowanie i zachęca do korzystania z logiki asynchronicznej, bazującej na wywołaniach zwrotnych. Choć wielu uważa, że ten jednowątkowy charakter Node.js jest jego wadą, środowisko to i tak jest w stanie obsługiwać wiele jednoczesnych żądań z dobrą wydajnością, co wystarczyło, by przyciągnąć do tej platformy licznych programistów.

Kilka miesięcy później został udostępniony projekt npm. Bazując na efektach pracy CommonJS, npm zapewnił twórcom pakietów możliwość publikowania swoich modułów w scentralizowanym rejestrze (**rejestrze npm**), a innym programistom zezwolił na instalowanie modułów i obsługę ich zależności przy użyciu narzędzia npm obsługiwanego z poziomu wiersza poleceń.

Środowisko Node.js zapewne nie przebiłoby się i nie zyskało powszechnej popularności, gdyby nie npm. Serwer Node.js udostępniał jedynie silnik języka JavaScript, pętlę obsługi zdarzeń oraz kilka niskopoziomowych API, jednak programiści pracujący nad większymi projektami wolą operować na abstrakcjach wyższego poziomu. Wykonując żądania HTTP czy też odczytując zawartość plików z dysku, programiści nie zawsze chcą operować na danych binarnych, odczytywać i zapisywać nagłówki czy też zawracać sobie głowę innymi zagadnieniami niskiego poziomu. npm oraz jego rejestr pozwalają społeczności programistów pisać i udostępniać — pod postacią modułów — własne abstrakcje wysokiego poziomu, które inni mogą w prosty sposób instalować i stosować w kodzie przy użyciu funkcji `require()`.

W odróżnieniu od innych języków programowania, które zazwyczaj dysponują wbudowanymi abstrakcjami wysokiego poziomu, twórcy Node.js mogli się skoncentrować na dostarczeniu podstawowych elementów konstrukcyjnych niskiego poziomu, a całą resztą zajęła się społeczność programistów. Społeczność ta wykonała wspaniałą robotę, tworząc takie fantastyczne abstrakcje jak `Express.js` — framework do tworzenia aplikacji internetowych, `Sequelize ORM` oraz setki tysięcy innych bibliotek gotowych do użycia po wykonaniu banalnego polecenia `npm install`.

Po pojawieniu się Node.js programiści JavaScriptu, którzy wcześniej nie znali żadnych języków serwerowych, zyskali możliwość tworzenia kompletnych aplikacji — obie ich części, kliencka i serwerowa, mogły być pisane w tym samym języku przez te same osoby.

Ambitni programiści zaczęli zatem tworzyć w języku JavaScript całe aplikacje, choć napotykali przy tym pewne problemy i wymyślali ich rozwiązania. Popularne stały się aplikacje jednostronicowe pisane wyłącznie w JavaScripcie, choć ich mankamentem były problemy z tworzeniem szablonów i organizacją bazy kodu. Społeczność odpowiedziała na te problemy, tworząc frameworki takie jak **Backbone.js** (będący pierwowzorem rozwiązań typu Angular czy React), **RequireJS** (mechanizm do wczytywania modułów CommonJS i AMD) oraz języki do tworzenia szablonów takie jak Mustache (poprzednik języka JSX).

Kup ksi k Pole ksi k

Kiedy programiści zaczęli mieć problemy ze stosowaniem na swoich jednostronicowych aplikacjach technik SEO, opracowali pomysł **aplikacji izomorficznych** (ang. *isomorphic applications*) oraz kodów, które można było wykonywać zarówno po stronie serwera (tak by mechanizmy wyszukiwarek internetowych mogły indeksować treści aplikacji), jak i po stronie klienta (tak by aplikacje były szybkie i mogły być pisane w JavaScripcie). To z kolei doprowadziło do powstania takich frameworków JavaScriptu jak **MeteorJS**.

W końcu programiści JavaScriptu tworzący aplikacje jednostronicowe uświadomili sobie, że ich potrzeby dotyczące rozwiązań serwerowych są często bardzo proste — ograniczają się do konieczności uwierzytelniania oraz zapisu i odczytu danych. To z kolei doprowadziło do powstania technologii eliminujących stosowanie serwera lub rozwiązań takich jak **Firebase** określanych mianem **database-as-a-service** (baza danych jako usługa — DBaaS), które z kolei wyznaczyły drogę i doprowadziły do wzrostu popularności mobilnych aplikacji pisanych w JavaScripcie. Mniej więcej w tym samym czasie pojawił się projekt Cordova/PhoneGap, który pozwalał programistom na umieszczanie kodu JavaScriptu w rodzimych komponentach WebView systemów iOS i Android oraz na publikowanie swoich aplikacji JavaScriptu w sklepach z aplikacjami mobilnymi.

W tej książce w bardzo dużym stopniu będziemy używać Node.js oraz npm. Większość zaprezentowanych w niej przykładów będzie korzystać z pakietów ML pobieranych przy użyciu npm.

Język TypeScript

Tworzenie i udostępnianie nowych pakietów za pośrednictwem npm nie było jedynym efektem wzrostu popularności języka JavaScript. Coraz częstsze stosowanie JavaScriptu jako głównego języka programowania sprawiło, że wielu programistów zaczęło się uskarżać na brak odpowiedniego zintegrowanego środowiska programistycznego (IDE) wraz z narzędziami wspomagającymi pracę w tym języku. Wcześniej zintegrowane środowiska programistyczne były bardziej popularne wśród programistów używających języków kompilowanych, o statycznym typowaniu, takich jak C oraz Java, gdyż w takich językach łatwiej jest przetwarzać i statycznie analizować kod. Naprawdę dobre środowiska programistyczne do tworzenia kodu w takich językach jak JavaScript czy PHP zaczęły się pojawiać stosunkowo niedawno, natomiast podobne narzędzia do programowania w Javie istniały już od wielu lat.

Firma Microsoft chciała dysponować lepszymi narzędziami oraz wsparciem dla swoich dużych projektów tworzonych w JavaScripcie, jednak na jej drodze stanęło kilka problemów związanych z językiem. W szczególności problemem okazało się dynamiczne typowanie stosowane w JavaScripcie (fakt, że zmiennej `var number` można było początkowo przypisać liczbę całkowitą o wartości `5`, a później jakiś dowolny obiekt), które uniemożliwiało stosowanie narzędzi do statycznej analizy kodu w celu zapewniania bezpieczeństwa typów oraz znacząco utrudniało odnajdywanie odpowiednich zmiennych lub obiektów, które mógłby sugerować mechanizm automatycznego uzupełniania. Co więcej, Microsoft chciał stosować paradygmat

Kup ksi k Pole ksi k

programowania obiektowego bazujący na klasach i udostępniający pojęcia interfejsów i kontraktów, a programowanie obiektowe w JavaScripcie bazuje na **prototypach**, a nie klasach.

Wszystkie te czynniki sprawiły, że firma Microsoft, w celu wsparcia dużych projektów informatycznych tworzonych w JavaScripcie, opracowała język TypeScript. TypeScript wprowadza klasy, interfejsy i statyczne typowanie. W odróżnieniu od stworzonego przez Google'a języka Dart, Microsoft zadbał, by TypeScript zawsze był ścisłym nadzbiorem JavaScriptu, co oznacza, że każdy prawidłowy kod JavaScriptu jest także prawidłowym kodem TypeScriptu. Kompilator TypeScriptu przeprowadza (w czasie kompilacji) statyczną kontrolę typów, pomagając programistom we wczesnym wykrywaniu błędów. Wsparcie dla statycznego typowania ułatwia także zintegrowanym środowiskom programistycznym dokładniejszą interpretację kodu, poprawiając tym samym doświadczenia związane z pisaniem kodu.

Kilka wczesnych usprawnień wprowadzonych przez TypeScript straciło na znaczeniu po udostępnieniu języka ECMAScript 2015, określanego także jako ES6. Na przykład mechanizm wczytywania modułów, składnia klas oraz składnia zapisu funkcji przy użyciu symbolu strzałki zostały zastosowane bezpośrednio w języku ES6, dlatego też obecnie TypeScript używa tych konstrukcji w wersjach z języka ES6; niemniej jednak TypeScript wciąż udostępnia statyczną kontrolę typów, której ES6 nie zapewnia.

Choć w przykładach zamieszczonych w tej książce nie będziemy używali języka TypeScript, wspominam tu o nim, gdyż kilka z bibliotek do uczenia maszynowego, które przedstawię, zostało w nim napisanych.

Jeden z przykładów dostępnych na stronie *deeplearn.js* zawiera następujący fragment kodu:

```
const graph = new Graph();
// Make a new input in the graph, called 'x', with shape [] (a Scalar).
const x: Tensor = graph.placeholder('x', []);
// Make new variables in the graph, 'a', 'b', 'c' with shape [] and random
// initial values.
const a: Tensor = graph.variable('a', Scalar.new(Math.random()));
const b: Tensor = graph.variable('b', Scalar.new(Math.random()));
const c: Tensor = graph.variable('c', Scalar.new(Math.random()));
```

Ta składnia wygląda jak kod JavaScriptu w wersji ES6, z wyjątkiem zapisu z dwukropkiem: `const x: Tensor = …` Ten kod informuje kompilator TypeScriptu, że `const x` musi być instancją klasy `Tensor`. Podczas kompilacji kodu TypeScript w pierwszej kolejności upewnia się, że wszędzie tam, gdzie występuje `x`, zostanie użyty obiekt klasy `Tensor` (w przeciwnym razie kompilator zgłosi błąd), a później, podczas generowania kodu JavaScriptu, po prostu pomija informacje o typach. Konwersja powyższego kodu TypeScriptu na kod JavaScriptu jest bardzo prosta — sprowadza się do usunięcia z definicji każdej zmiennej dwukropka i użytej za nim nazwy klasy `Tensor`.

Czytając tę książkę i analizując zawarte w niej przykłady, można oczywiście używać języka TypeScript, lecz trzeba to będzie odpowiednio uwzględnić w procesie budowy kodu przedstawionego w dalszej części rozdziału.

Kup ksi k

Pole ksi k

Usprawnienia wprowadzone w ES6

Komitet ECMAScript, definiujący specyfikację języka JavaScript, opublikował w czerwcu 2015 r. nową specyfikację — ECMAScript 6/ECMAScript 2015. Ten nowy standard języka, określany potocznie jako **ES6**, był kolejną główną wersją JavaScriptu i wprowadzał do niego wiele nowych paradygmatów, opracowanych w celu ułatwienia programowania w tym języku.

Choć ECMAScript definiuje specyfikację języka JavaScript, jego rzeczywiste implementacje zależą od twórców przeglądarek oraz różnych silników JavaScriptu. ES6 jest jedynie zbiorem wytycznych, a ponieważ twórcy przeglądarek implementują nowe możliwości języka zgodnie z własnymi terminarzami, dlatego jego specyfikacje i implementacje nieco się od siebie różnią. Możliwości zdefiniowane w ES6, takie jak klasy, nie były jeszcze zaimplementowane w najpopularniejszych przeglądarkach, a programiści już chcieli ich używać.

I tak pojawił się **Babel** — transpilator języka JavaScript. Babel potrafi analizować różne wersje języka JavaScript (ES6, ES7, ES8 oraz React JSX) i konwertować je, czy też kompilować, na kod zgodny z zaimplementowaną w przeglądarkach wersją języka ES5. Nawet dziś ES6 nie został jeszcze zaimplementowany w przeglądarkach w całości, dlatego dla programistów, którzy chcą pisać kod ES6, Babel pozostaje kluczowym narzędziem.

W przykładach przedstawionych w tej książce będziemy używali języka ES6. Jeśli ktoś nie zna stosowanej w nim nowej składni, to w dalszej części tego podrozdziału przedstawię kilka jego najważniejszych cech, które będziemy wykorzystywać w tej książce.

Let i const

W wersji ES5 języka JavaScript do definiowania zmiennych używane było słowo kluczowe `var`. W większości przypadków można je zastąpić słowem kluczowym `let`, przy czym główną różnicą między nimi jest zasięg zmiennych w odniesieniu do bloku. Tę subtelną różnicę pomiędzy `var` i `let` demonstruje poniższy przykład zaczerpnięty z witryny **MDN web docs** (noszącej wcześniej nazwę **Mozilla Developer Network**), ze strony *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let*:

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2; // ta sama zmienna!
    console.log(x); // 2
  }
  console.log(x); // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2; // inna zmienna
```


Kup ksi k Pole ksi k

```
    console.log(x); // 2
  }
  console.log(x); // 1
}
```

A zatem, choć w niektórych sytuacjach, takich jak ta przedstawiona na powyższym przykładzie, należy zachować dodatkową ostrożność, to jednak w większości przypadków można po prostu zastąpić `var` słowem kluczowym `let`.

W odróżnieniu od `let` słowo kluczowe `const` definiuje zmienną jako stałą — innymi słowy: po zainicjowaniu zmiennej zdefiniowanej jako `const` jej wartości nie można już zmieniać. Na przykład użycie kodu przedstawionego na kolejnym przykładzie spowoduje wyświetlenie błędu o treści zbliżonej do *invalid assignment to const a*[1]:

```
const a = 1;
a = 2;
```

Z drugiej strony, gdyby do zdefiniowania zmiennej `a` zostało użyte słowo kluczowe `let` lub `var`, to ten kod działałby prawidłowo.

> Warto zwrócić uwagę, że jeśli `a` jest obiektem, to wartości właściwości tego obiektu można modyfikować.

Kolejny przykładowy fragment kodu zostanie wykonany prawidłowo:

```
const obj = {};
obj.name = "Mój obiekt";
```

Z drugiej strony, próba przedefiniowania całego obiektu — `obj = {name: "inny obiekt"};` — spowodowałaby zgłoszenie błędu.

Uważam, że w większości kontekstów lepiej nadaje się `const` niż `let`, gdyż większość zmiennych nie wymaga przedefiniowywania. Dlatego też sugeruję, by jak najczęściej używać właśnie słowa kluczowego `const`, a na `let` decydować się tylko wtedy, gdy konieczne będzie późniejsze przedefiniowanie zmiennej.

Klasy

Jedną z wyczekiwanych zmian wprowadzonych w ES6 było dodanie klas oraz mechanizmu ich dziedziczenia. Wcześniej programowanie obiektowe w języku JavaScript wymagało stosowania mechanizmu dziedziczenia bazującego na prototypach, który dla wielu programistów nie był intuicyjny; przykład tego tradycyjnego mechanizmu dziedziczenia przedstawia poniższy fragment kodu:

[1] Z ang.: nieprawidłowe przypisanie do stałej a — *przyp. tłum.*

Kup ksi k   Pole ksi k

```
var Automobile = function(weight, speed) {
   this.weight = weight;
   this.speed = speed;
}
Automobile.prototype.accelerate = function(extraSpeed) {
  this.speed += extraSpeed;
}
var RaceCar = function (weight, speed, boost) {
   Automobile.call(this, weight, speed);
   this.boost = boost;
}
RaceCar.prototype = Object.create(Automobile.prototype);
RaceCar.prototype.constructor = RaceCar;
RaceCar.prototype.accelerate = function(extraSpeed) {
   this.speed += extraSpeed + this.boost;
}
```

W tym przypadku rozszerzenie obiektu wymaga umieszczenia w funkcji `constructor` klasy pochodnej, wywołania klasy bazowej, utworzenia kopii obiektu prototypu klasy bazowej i przesłonięcia konstruktora prototypu klasy bazowej konstruktorem prototypu klasy pochodnej. Większość programistów uważała, że czynności te są nieintuicyjne i uciążliwe.

W języku ES6 analogiczny kod wygląda następująco:

```
class Automobile {
   constructor(weight, speed) {
      this.weight = weight;
      this.speeed = speed;
   }
   accelerate(extraSpeed) {
      this.speed += extraSpeed;
   }
}
class RaceCar extends Automobile {
   constructor(weight, speed, boost) {
      super(weight, speed);
      this.boost = boost;
   }
   accelerate(extraSpeed) {
      this.speed += extraSpeed + this.boost;
   }
}
```

Powyższy kod jest już znacznie bardziej zbliżony do tego, czego moglibyśmy oczekiwać od kodu napisanego w obiektowym języku programowania, i jak widać, znacząco upraszcza dziedziczenie.

Koniecznie trzeba jednak pamiętać, że w niezauważalny dla programistów sposób ES6 wciąż korzysta z tego samego paradygmatu dziedziczenia bazującego na prototypach. Klasy są jedynie „syntaktycznym lukrem" przesłaniającym istniejący mechanizm, dlatego jedyną różnicą pomiędzy oboma przedstawionymi przykładami jest przejrzystość kodu.

Kup ksi k Pole ksi k

Importowanie modułów

W języku ES6 wprowadzono także interfejsy do importowania i eksportowania modułów. We wcześniejszym rozwiązaniu stosowanym przez CommonJS moduły były eksportowane przy użyciu konstrukcji `modules.export` oraz importowane przy użyciu funkcji `require(`*`nazwapliku`*`)`. W języku ES6 importowanie modułów wygląda nieco inaczej.

W jednym pliku umieszcza się definicję klasy oraz eksportuje ją w sposób przedstawiony na kolejnym przykładzie:

```
Class Automobile {
...
}
export default Automobile
```

Z kolei w innym pliku klasa ta jest importowana w następujący sposób:

```
import Automobile from './classes/automobile.js';
const myCar = new Automobile();
```

Obecnie Babel kompiluje moduły ES6 do formatu używanego przez moduły CommonJS, dlatego w razie stosowania tego narzędzia możemy używać zarówno modułów ES6, jak i CommonJS.

Funkcje strzałkowe

Jednym z dziwacznych, użytecznych, lecz równocześnie nieco denerwujących aspektów standardu ES5 JavaScriptu jest częste stosowanie wykonywanych asynchronicznie funkcji zwrotnych. Chyba każdy doskonale zna kod jQuery przypominający fragment przedstawiony poniżej:

```
$("#link").click(function() {
  var $self = $(this);
  doSomethingAsync(1000, function(resp) {
    $self.addClass("wasFaded");
    var processedItems = resp.map(function(item) {
      return processItem(item);
    });
    return shipItems(processedItems);
  });
});
```

Taki kod zmusza nas do utworzenia zmiennej `$self`, gdyż w wewnętrznej funkcji anonimowej początkowy kontekst jest tracony. Oprócz tego, ze względu na konieczność utworzenia trzech osobnych funkcji anonimowych, kod jest rozbudowany i utrudnia analizę.

Składnia funkcji strzałkowych (ang. *arrow funcions*) jest jednocześnie ułatwieniem składniowym pozwalającym na tworzenie funkcji anonimowych przy użyciu krótszego zapisu, jak też funkcjonalnym usprawnieniem, które zachowuje kontekst `this` w wywołaniu funkcji strzałkowej.

Kup ksi k Pole ksi k

Na przykład w języku ES6 powyższy fragment kodu można zapisać w następujący sposób:

```
$("#link").click(function() {
  dozsSomethingAsync(1000, resp => {
    $(this).addClass("wasFaded");
    const processedItems = resp.map(item => processItem(Item));
    return shipItems(processedItems);
  });
});
```

Jak widać, w tej wersji kodu nie musimy już tworzyć zmiennej `$self`, by zachować kontekst `this`, a wywołanie `.map` jest znacznie prostsze, gdyż nie wymaga stosowania słowa kluczowego `function`, nawiasów, nawiasów klamrowych ani instrukcji `return`.

Przeanalizujmy teraz przykłady odpowiadających sobie funkcji. Pierwsza z nich ma następującą postać:

```
const double = function(number) {
  return number * 2;
}
```

Tę funkcję można zapisać w następującej postaci:

```
const double = number => number * 2;
```

Lub w tożsamej postaci:

```
const double = (number) => { return number * 2; }
```

W tych przykładach definiowana funkcja anonimowa ma tylko jeden parametr — `number` — a zatem nie trzeba zapisywać go w nawiasach. Gdyby funkcja wymagała dwóch parametrów, konieczne byłoby zapisanie ich w nawiasach, jak pokazałem na kolejnym przykładzie. Co więcej, jeśli ciało funkcji składa się z jednej instrukcji, można pominąć otaczające ją nawiasy klamrowe oraz ewentualną instrukcję `return`.

Przyjrzyjmy się kolejnemu przykładowi dwóch odpowiadających sobie funkcji, które tym razem będą miały więcej niż jeden parametr:

```
const sorted = names.sort(function(a, b) {
  return a.localeCompare(b);
});
```

W języku ES6 powyższy kod można zapisać w następującej postaci:

```
const sorted = names.sort((a, b) => a.localeCompare(b));
```

Uważam, że funkcje strzałkowe najlepiej sprawdzają się w sytuacjach takich jak ta z ostatniego przykładu, w których wykonujemy jakieś przekształcenia danych, zwłaszcza z użyciem funkcji `Array.map`, `Array.filter`, `Array.reduce` oraz `Array.sort`, do których przekazujemy bardzo proste funkcje anonimowe. Funkcje strzałkowe są natomiast nieco mniej przydatne podczas stosowania jQuery, gdyż biblioteka ta zazwyczaj przekazuje dane przy użyciu kontekstu `this`, którym nie dysponujemy w przypadku stosowania anonimowych funkcji strzałkowych.

Kup ksi k Pole ksi k

Literały obiektowe

W języku ES6 wprowadzono także kilka usprawnień związanych z literałami obiektowymi. Tych usprawnień jest kilka, a najpopularniejszy z nich to niejawne określanie nazw właściwości obiektów. W języku ES5 wyglądałoby to następująco:

```
var name = 'Burak';
var title = 'Autor';
var object = {name: name, title: title};
```

W języku ES6, jeśli nazwa właściwości oraz nazwa zmiennej są takie same jak w powyższym przykładzie, to zapis literału obiektowego można uprościć do następującej postaci:

```
const name = 'Burak';
const title = 'Autor';
const object = {name, title};
```

Co więcej, w ES6 wprowadzony został operator rozproszenia obiektów (ang. *object spread operator*), ułatwiający płytkie scalanie obiektów. W ramach przykładu przeanalizujmy poniższy fragment kodu ES5:

```
function combinePreferences(userPreferences) {
  var defaultPreferences = {size: 'large', mode: 'view'};
  return Object.assign({}, defaultPreferences, userPreferences);
}
```

Powyższy kod utworzy nowy obiekt `defaultPreferences`, a następnie scali go z właściwościami przekazanego obiektu `userPreferences`. Przekazanie pustego obiektu jako pierwszego parametru metody `Object.assign()` zapewnia, że utworzony zostanie nowy obiekt, a obiekt `defaultParameters` pozostanie niezmieniony (co w powyższym przykładzie nie ma większego znaczenia, lecz może być ważne w przypadku zastosowania tego rozwiązania w rzeczywistym kodzie).

A teraz zobaczmy, jak ten sam kod można zapisać w języku ES6:

```
function combinePreferences(userPreferences) {
  var defaultPreferences = {size: 'large', mode: 'view'};
  return {...defaultPreferences, ...userPreferences};
}
```

Ten kod robi dokładnie to samo co poprzedni, lecz jest krótszy i według mnie łatwiejszy do zrozumienia, niż jego odpowiednik z języka ES5 korzystający z metody `Object.assign()`. Programiści używający frameworków React i Redux bardzo często używają operatora rozproszenia obiektów w operacjach związanych z zarządzaniem stanem reduktora.

Funkcja for…of

W języku ES5 pętle `for` operujące na tablicach często implementuje się przy użyciu składni pętli `for (indeks in tablica)`, jak pokazałem na poniższym przykładzie:

Kup ksi k	Pole ksi k

```
var items = [1, 2, 3];
for (var index in items) {
  var item = items[index];
  ...
}
```

Korzystając ze składni `for...of`, wprowadzonej w języku ES6, możemy skrócić to rozwiązanie o jeden wiersz kodu:

```
var items = [1, 2, 3];
for (const index of items) {
  ...
}
```

Obietnice

Obietnice (ang. *promises*), w takiej lub innej formie, są dostępne w języku JavaScript już od jakiegoś czasu. Ideę obietnic znają na pewno użytkownicy biblioteki jQuery. **Obietnica** to referencja do zmiennej generowanej asynchronicznie, która może być dostępna w przyszłości.

W przypadku języka ES5, jeśli nie korzystaliśmy z jakiejś osobnej biblioteki obietnic lub z obiektów odroczonych (ang. *deferred*) biblioteki jQuery, obietnice wymagały zastosowania funkcji zwrotnej do metody asynchronicznej i wykonania jej w przypadku prawidłowego przeprowadzenia tej asynchronicznej operacji. Oto przykład takiego rozwiązania:

```
function updateUser(user, settings, onComplete, onError) {
  makeAsyncApiRequest(user, settings, function(response) {
    if (response.isValid()) {
      onComplete(response.getBody());
    } else {
      onError(response.getError())
    }
  });
}
updateUser(user, settings, function(body) { ... }, function(error) { ...
});
```

W języku ES6 można zwrócić obiekt `Promise`, który hermetyzuje żądanie asynchroniczne i zostaje wyznaczony lub odrzucony, jak pokazałem na poniższym przykładzie:

```
function updateUser(user, settings) {
  return new Promise((resolve, reject) => {
    makeAsyncApiRequest(user, settings, function(response) {
      if (response.isValid()) {
        resolve(response.getBody());
      } else {
        reject(response.getError())
      }
    });
  });
}
updateUser(user, settings)
```


Kup ksi k Pole ksi k

```
.then(
  body => { ... },
  error => { ... }
);
```

Najważniejszą cechą obietnic jest to, że można je przekazywać jako obiekty, a funkcje, które je obsługują, można łączyć, tworząc łańcuchy.

Funkcje async/wait

Tak naprawdę słowa kluczowe `async` oraz `wait` nie należą do możliwości języka ES6, a raczej ES8. Choć obietnice znacznie ułatwiły obsługę wywołań asynchronicznych, w dużym stopniu wymuszają tworzenie łańcuchów wywołań, a w niektórych przypadkach wymagają stosowania paradygmatu programowania asynchronicznego, choć programista wolałby pisać funkcję działającą asynchronicznie, lecz wyglądającą tak, jakby jej kod był realizowany synchronicznie.

Przyjrzymy się przykładowi zaczerpniętemu ze strony MDN poświęconej funkcjom asynchronicznym (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function*):

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('wyznaczono');
    }, 2000);
  });
}
async function asyncCall() {
  console.log('wywołuję');
  var result = await resolveAfter2Seconds();
  console.log(result);
  // oczekiwane wyniki: "wyznaczono"
}
asyncCall();
```

Funkcja `resolveAfter2Seconds` to zwyczajna funkcja JavaScriptu, która zwraca obiekt obietnicy ES6. Cała magia tego rozwiązania zawiera się w funkcji `asyncCall` poprzedzonej słowem kluczowym `async`. Wewnątrz niej wywołujemy funkcję `resolveAfter2Seconds`, używając przy tym słowa kluczowego `await`, a nie konstrukcji `.then(result => console.log(result))`, bardziej znanej z kodu korzystającego z obietnic, stosowanego w języku ES6. Słowo kluczowe `await` sprawia, że nasza funkcja `async` poczeka na wyznaczenie obietnicy i dopiero potem przekaże sterowanie dalej, zwracając przy tym bezpośrednio wynik obiektu `Promise`. Dzięki temu słowa kluczowe `async` i `await` pozwalają przekształcać asynchroniczne funkcje używające obietnic na kod z pozoru wyglądający, jakby był wykonywany synchronicznie, poprawiając tym samym zachowanie przejrzystości wielokrotnie zagnieżdżanego kodu korzystającego z obietnic oraz funkcji asynchronicznych i ułatwiając ich analizę.

Słowa kluczowe `async` i `await` są elementami języka ES8, a nie ES6, dlatego kiedy niebawem zajmiemy się konfigurowaniem programu Babel, będziemy musieli pamiętać, by uwzględnić możliwości wszystkich nowych wersji ECMAScript, a nie tylko możliwości języka ES6.

Kup ksi k Pole ksi k

Przygotowywanie środowiska programistycznego

Przykłady przedstawione w tej książce będą używać zarówno środowiska przeglądarek WWW, jak i środowiska Node.js. Choć Node.js w wersji 8 oraz nowszych dysponuje obsługą standardów ES+, nie wszyscy producenci przeglądarek wyposażyli swoje produkty w kompletną obsługę wszystkich możliwości ES6+; z tego względu prezentowane w książce kody będziemy przetwarzać przy użyciu transpilatora Babel.

W tej książce będę się starał używać tej samej struktury projektu we wszystkich przykładach, niezależnie od tego czy będą one uruchamiane w środowisku Node.js z poziomu wiersza poleceń, czy też w przeglądarce WWW. Ponieważ struktura projektów, którą przygotujemy w tym rozdziale, będzie standardowa, nie wszystkie projekty przedstawione w tej książce będą korzystać ze wszystkich jej możliwości.

Oto narzędzia, które będą konieczne:

- Ulubiony edytor kodów, taki jak Vim, Emacs, Sublime Text czy też WebStorm.
- Nowoczesna i aktualna przeglądarka, taka jak Chrome lub Firefox.
- Node.js Version 8 LTS lub nowsza wersja; w tej książce używałem Node.js w wersji 9.4.0.
- Menedżer pakietów Yarn (opcjonalnie zamiast niego można używać programu npm).
- Różne narzędzia używane do budowania kodów, takie jak Babel czy Browserify.

Instalowanie Node.js

Dla użytkowników komputerów z systemem macOS najprostszym sposobem zainstalowania Node.js będzie skorzystanie z menedżera pakietów takiego jak **Homebrew** lub **MacPorts**. W celu zapewnienia jak najlepszej zgodności z przykładami przedstawionymi w tej książce należy zainstalować Node.js w wersji 9.4.0 lub nowszej.

Użytkownicy systemu Windows mogą zainstalować Node.js, używając menedżera pakietów **Chocolatey** bądź też postępując zgodnie z instrukcjami podanymi na stronie do pobierania Node.js: *https://nodejs.org/en/*.

Użytkownicy systemu Linux powinni uważać na instalowanie Node.js przy wykorzystaniu menedżera pakietów dostępnego w ich dystrybucji systemu, gdyż może on udostępniać znacznie starszą wersję Node.js niż najnowsza. Jeśli menedżer pakietów używa wersji starszej niż V8, to można dodać do menedżera odpowiednie repozytorium pakietów, zbudować Node.js z kodów źródłowych lub zainstalować jego wersję binarną odpowiednią dla używanego systemu.

Kup ksi k Pole ksi k

Po zainstalowaniu Node.js warto upewnić się, że środowisko działa, oraz sprawdzić, jaka jest jego wersja. W tym celu należy wykonać polecenie `node --version`; powinno ono wygenerować następujące wyniki:

```
C:\> node --version
 V9.4.0
```

Przy okazji warto sprawdzić, czy także program npm działa prawidłowo:

```
C:\> npm --version
 5.6.0
```

Opcjonalne zainstalowanie Yarn

Yarn to menedżer pakietów zgodny z npm, choć według mnie działa szybciej i jest łatwiejszy w obsłudze. Użytkownicy programu Homebrew w systemie macOS mogą zainstalować go, używając polecenia `brew install yarn`; wszyscy pozostali powinni postępować zgodnie z instrukcjami instalacji wyświetlonymi na stronie narzędzia (*https://yarnpkg.com/en/docs/install#windows-stable*).

Jeśli wolisz npm zamiast Yarn, nic nie stoi na przeszkodzie — oba narzędzia używają tego samego formatu zapisu plików *package.json*, choć różnią się nieco składnią poleceń takich jak `add`, `require` oraz `install`. Jeśli natomiast używasz programu npm, a nie Yarn, to będziesz musiał zastąpić polecenia podawane w dalszej części książki ich prawidłowymi odpowiednikami — nazwy pakietów pozostają takie same.

Tworzenie i inicjowanie przykładowego projektu

Użyj wiersza poleceń, ulubionego IDE lub menedżera plików, by utworzyć katalog o nazwie *UMwJS*, w nim katalog *Rozdzial01*, a w nim podkatalog *R01-Cw1*.

Następnie przejdź do katalogu *R01-Cw1* i wykonaj polecenie `yarn init`, które podobnie jak `npm init` spowoduje utworzenie pliku *package.json* i umożliwi podanie podstawowych informacji o projekcie. Odpowiedz na zadawane pytania. Pakiet, który tu tworzymy, nie będzie nigdzie publikowany, więc podawane odpowiedzi nie mają szczególnego znaczenia, jednak w odpowiedzi na pytanie o punkt wejścia do aplikacji (ang. *application entry point*) należy wpisać: `dist/index.js`.

Kolejnym krokiem będzie zainstalowanie kilku narzędzi do budowania kodów, których będziemy używać w przeważającej większości prezentowanych tu przykładowych projektów:

- `babel-core` — podstawowe pliki transpilatora Babel;
- `babel-preset-env` — ustawienia transpilatora Babel przeznaczone do parsowania kodów ES6, ES7 oraz ES8;
- `browserify` — narzędzie do łączenia kodów JavaScriptu, które pozwala scalać kilka plików w jeden;
- `babelify` — wtyczka Babel do narzędzia Browserify.

Kup ksi k Pole ksi k

Wszystkie te narzędzia można zainstalować jako niezbędne dla środowiska roboczego, używając w tym celu następującego polecenia:

```
yarn add -D babel-cli browserify babelify babel-preset-env
```

Tworzenie projektu „Witaj, świecie!”

Aby upewnić się, że kody można prawidłowo skompilować i uruchomić, stworzymy teraz bardzo prosty projekt typu „Witaj, świecie!” i dodamy do niego skrypt budujący.

W pierwszej kolejności w katalogu *R01-Cw1* utwórz dwa podkatalogi: *src* oraz *dist*. Tej konwencji będziemy używali we wszystkich projektach: katalog *src* będzie zawierał kody źródłowe napisane w JavaScripcie, natomiast katalog *dist* będzie zawierał zbudowane kody źródłowe oraz wszelkie dodatkowe zasoby (obrazy, pliki CSS, HTML itd.) konieczne do działania projektu.

Następnie w katalogu *src* utwórz plik *greeting.js* o następującej zawartości:

```
const greeting = name => 'Witaj, ' + name + '!';
export default greeting;
```

Oraz plik *index.js*, którego zawartość przedstawiłem poniżej:

```
import greeing from './greeting';
console.log(greeting(process.argv[2] || 'świecie'));
```

Ta bardzo prosta aplikacja pozwoli nam sprawdzić, czy można używać podstawowej składni języka ES6, mechanizmu wczytywania modułów oraz korzystać z argumentów wiersza poleceń przekazywanych do Node.js.

Następnie otwórz plik *package.json* w katalogu *R01-Cw1* i dodaj do niego poniższy fragment kodu:

```
"scripts": {
  "build-web": "browserify src/index.js -o dist/index.js -t [ babelify --presets
  ↪[ env ] ]",
  "build-cli": "browserify src/index.js --node -o dist/index.js -t [ babelify --
  ↪presets [ env ] ]",
  "start": "yarn build-cli && node dist/index.js"
},
```

Ten fragment definiuje trzy proste skrypty uruchamiane z poziomu wiersza poleceń:

- `build-web` — skrypt używa narzędzi Browserify i Babel do skompilowania wszystkiego, do czego odwołuje się plik *src/index.js*, do jednego pliku i zapisania go jako *dist/index.js*.
- `build-cli` — skrypt działa podobnie do `build-web`, z tym że używa dodatkowo flagi Browserify `node`, bez której nie bylibyśmy w stanie korzystać z argumentów wiersza poleceń przekazywanych do Node.js.
- `start` — skrypt jest przeznaczony wyłącznie dla projektów korzystających z Node.js i uruchamianych z poziomu wiersza poleceń, a jego wykonanie powoduje zarówno zbudowanie kodów źródłowych, jak i uruchomienie wynikowego kodu projektu.

Kup ksi k Pole ksi k

W efekcie plik *package.json* powinien wyglądać podobnie do przedstawionego poniżej:

```
{
  "name": "R01-Cw1-SrodProgram",
  "version": "0.0.1",
  "description": "Chapter one example",
  "main": "src/index.js",
  "author": "Burak Kanber",
  "license": "MIT",
  "scripts": {
    "build-web": "browserify src/index.js -o dist/index.js -t [ babelify --presets
    ↪[ env ] ]",
    "build-cli": "browserify src/index.js --node -o dist/index.js -t [ babelify --
    ↪presets [ env ] ]",
    "start": "yarn build-cli && node dist/index.js"
  },
  "dependencies": {
    "babel-core": "^6.26.0",
    "babel-preset-env": "^1.6.1",
    "babelify": "^8.0.0",
    "browserify": "^15.1.0"
  }
}
```

Teraz użyjemy tej prostej aplikacji do przeprowadzenia kilku testów. W pierwszej kolejności upewnimy się, że działa polecenie `yarn build-cli`. Jego wykonanie powinno wyglądać jak na kolejnym przykładzie:

```
$ yarn build-cli
yarn run v1.3.2
$ browserify src/index.js --node -o dist/index.js -t [ babelify --presets [ env ] ]
Done in 0.59s.
```

Po jego wykonaniu należy sprawdzić, czy został wygenerowany plik *dist/index.js*, oraz spróbować uruchomić ten plik przy użyciu kolejnego polecenia:

```
$ node dist/index.js
Witaj, świecie!
```

Warto także spróbować przekazać do skryptu jakiś argument, tak jak pokazałem poniżej:

```
$ node dist/index.js Adrian
Witaj, Adrian!
```

W ramach kolejnego testu, przedstawionego poniżej, wypróbujemy działanie polecenia `build-web`. Ponieważ w tym przypadku opcja `node` nie została zastosowana, spodziewamy się, że przekazywanie argumentu do skryptu nie będzie działać:

```
$ yarn build-web
yarn run v1.3.2
$ browserify src/index.js -o dist/index.js -t [ babelify --presets [ env ] ]
Done in 0.61s.
$ node dist/index.js Adrian
Witaj, świecie!
```


Kup ksi k   Pole ksi k

Bez zastosowania opcji node argumenty wiersza poleceń nie są przekazywane do skryptu, dlatego też, zgodnie z oczekiwaniami, wyświetli on domyślny komunikat: Witaj, świecie!

W ramach ostatniego testu pozostaje nam sprawdzić działanie polecenia yarn start i upewnić się, że powoduje ono zbudowanie aplikacji w wersji przeznaczonej do uruchamiania z poziomu wiersza poleceń oraz zapewnia prawidłowe przekazywanie do skryptu argumentów wiersza poleceń:

```
$ yarn start "good readers"
yarn run v1.3.2
$ yarn build-cli && node dist/index.js 'drogi czytelniku'
$ browserify src/index.js --node -o dist/index.js -t [ babelify --presets [env ] ]
Witaj, drogi czytelniku!
Done in 1.05s.
```

Jak widać na podstawie wyników, polecenie yarn start prawidłowo wygenerowało wersję aplikacji przeznaczoną do uruchamiania z poziomu wiersza poleceń oraz przekazało do niej argumenty podane w wierszu poleceń.

Będę się starał stosować tę samą strukturę we wszystkich przykładach prezentowanych w tej książce, niemniej jednak warto zwrócić uwagę na informacje podawane na początku każdego z rozdziałów, gdyż poszczególne przykłady mogą wymagać jakichś dodatkowych czynności konfiguracyjnych.

Podsumowanie

W tym rozdziale opisałem najważniejsze momenty w historii języka JavaScript, które miały znaczenie dla uczenia maszynowego, zaczynając od uruchomienia firmy Google (*http://www.google.com/*), a kończąc na stworzonej przez tę firmę i udostępnionej w 2017 r. bibliotece *deeplearn.js*.

Omówiłem tu także zalety, jakie zapewnia stosowanie języka JavaScript do implementacji rozwiązań uczenia maszynowego, jak również związane z nim wyzwania, w szczególności te dotyczące ekosystemu uczenia maszynowego.

W kolejnej części rozdziału przedstawiłem najważniejsze z ostatnich zmian wprowadzonych w języku JavaScript, także te zaimplementowane w standardzie ES6 — najnowszej stabilnej wersji JavaScriptu.

W końcu przygotowaliśmy przykładowe środowisko do programowania, składające się ze środowiska Node.js, menedżera pakietów Yarn oraz narzędzi Babel i Browserify, z którego będziemy korzystali w pozostałych przykładach przedstawionych w dalszej części książki.

W następnym rozdziale zajmiemy się badaniem i przetwarzaniem danych.

Kup ksi k Pole ksi k

Skorowidz

A

agregacja danych, 293
algorytm, 84
  Apriori, 178
  KNN, 123, 124
    implementacja, 125
  k-średnich, 89, 93
    centroidy, 99
    dane dwuwymiarowe, 107
    dane trójwymiarowe, 114
    implementacja, 107
    inicjalizacja, 94
    nieznane k, 116
    pętla główna, 106
    położenie centroidów, 102
  lasu losowego, 162
  maszyna wektorów nośnych, 154
  naiwnego klasyfikatora bayesowskiego, 141
algorytmy
  dostępne zasoby, 312
  grupowania, 84, 89
  klasyfikacji, 79, 85, 123
  łączenie modeli, 316
  optymalizacji, 86
  problemy, 313
  regresji, 79, 85, 183
  sztucznych sieci neuronowych, 209
  reguł asocjacyjnych, 169, 174
  tryb uczenia, 305
  uczenia maszynowego, 69
  uczenia nadzorowanego, 79
  wejście i wyjście, 309
alternatywa wykluczająca, 217
analiza, 72
  danych, 178
  fourierowska, 204
  głównych składowych, 47, 74
  nastawienia, 123
  nastrojów, 123
  regresji, 185
  sezonowości, 203
  szeregów czasowych, 200
ANN, Artificial Neural Networks, 209
aplikacje
  czasu rzeczywistego, 281
  izomorficzne, 24
asocjacje, 169

B

badanie danych, 39
biblioteka TensorFlow, 19, 217
bigram, 44
błąd średniokwadratowy, 214, 235
brakujące dane
  kategorialne, 52
  liczbowe, 52

C

cechy, 42
centroidy, 90, 99, 120
  aktualizowanie położenia, 102
  przypisywanie punktów, 100
CNN, Convolutional Neural Network, 227

CommonJS, 21
częstotliwość, 257
części mowy, 274
czyszczenie danych, 51

D

dane
- agregacja, 293
- czyszczenie, 51
- dostarczanie, 298
- graficzna prezentacja, 130
- kategorialne, 52
- liczbowe, 52
- łączenie, 293
- normalizacja, 61, 295
- potokowanie, 290
- przechowywanie, 298
- przekształcanie, 61, 295
- przeszukiwanie, 291
- przetwarzanie, 39
- przygotowywanie, 51
- sezonowe, 203
- uczące, 71

długa pamięć krótkoterminowa, LSTM, 249
DNN, Deep Neural Networks, 227

E

elementy odstające, 58
ES6, 26

F

filtrowanie
- słów pomijalnych, 140
- subtraktywne, 202

filtry
- FIR, 241
- środkowoprzepustowe, 201
- środkowozaporowe, 201

FIR, finite impulse response, 241
Firebase, 24
fonetyka, 272
format, 309
framework
- Backbone.js, 23
- MeteorJS, 24

funkcja for…of, 31
funkcje
- aktywacji, 211
- async/wait, 33
- logistyczne, 216
- strzałkowe, 29

G

głębokie sieci neuronowe, DNN, 227
Google Cloud Platform, 19
GRU, gated recurrent unit, 246
grupowanie, 84, 89

H

heurystyka, 70

I

identyfikacja cech, 42
implementacja
- algorytmu KNN, 125
- naiwnego klasyfikatora bayesowskiego, 141

importowanie modułów, 29
inicjalizacja algorytmu, 94
inicjatywa CommonJS, 21
instalowanie Node.js, 34

J

JavaScript, 15
jednostka LSTM, 250
język
- JavaScript, 15
- TypeScript, 24

K

k najbliższych sąsiadów, KNN, 124
kategorie algorytmów, 84
kernel trick, 156
klasa, 27
klastry, 89
klasyfikacja, 84, 184
klasyfikator
- k najbliższych sąsiadów, KNN, 123
- lasy losowe, 123, 162
- maszyna wektorów nośnych, 123, 154
- naiwny klasyfikator bayesowski, 123, 138

KNN, k-Nearest Neighbors, 123
konwolucje, 229
konwolucyjne sieci neuronowe, CNN, 19, 40, 227
korelacja Pearsona, 48
krzywa logistyczna, 216

L

lasy losowe, 124, 162
lematyzacja, 272
literały obiektowe, 31
LSTM, Long Short-Term Memory, 249

Ł

łączenie
  danych, 293
  modeli, 316

M

macierz
  konwolucji, 230, 231
  rzadka, 277
mapa odległości, 127
maszyna wektorów nośnych, 123, 154
menedżer pakietów Yarn, 35
metody osadzone, 47
miara Giniego, 163
modele
  samodoskonalące, 287
  spersonalizowane, 287
moduł, 29

N

nadmierne dopasowanie, 81
naiwny klasyfikator bayesowski, 123, 138
neuron, 210
  GRU, 247
  rekurencyjny, 243
NLP, Natural Language Processing, 209, 253
Node Package Managera, 21
Node.js, 22
  instalowanie, 34
normalizacja, 61, 295

O

obciążenie, 81
obietnice, 32
obrazy, 87
  rozmycie, 231
  wyostrzenie, 230
obsługa
  brakujących danych, 51
  elementów odstających, 58
  szumów, 53
odbarwianie zdjęcia, 133
odległość, 90
  Dmerou-Levenshteina, 255
  edycyjna, 255
odwrotna częstość, 257
ogólna sztuczna inteligencja, AGI, 209
operacja XOR, 217
oprogramowanie jako usługa, SaaS, 284
optymalizacja, 86

P

pierwiastek błędu średniokwadratowego, 189
pierwszy projekt, 36
pomiary dokładności, 76
postać, 309
potokowanie danych, 290
prawdopodobieństwo warunkowe, 139
problem XOR, 217
problemy optymalizacyjne, 307
projekt
  tworzenie, 35
propagacja wsteczna, 214
przekazywanie słów, 276
przekleństwo wymiarowości, 43
przekształcanie danych, 61
przeszukiwanie danych, 291
przetwarzanie
  języka naturalnego, NLP, 44, 86, 209, 253
  obrazów, 87
  sygnałów cyfrowych, 201
  wstępne, 72
przeuczenie, 81
przewidywanie, 183
przygotowywanie danych, 51

Kup ksi k

Pole ksi k

R

redukcja wymiarowości, 85
regresja, 85, 184, 185
  liniowa, 188, 189
  wielomianowa, 198
  wykładnicza, 193
reguły asocjacji, 171, 176
rekurencyjne sieci neuronowe, RNN, 227, 241
RNN, Recurrent Neural Network, 227, 241
rozpoznawanie mowy, 272
rzeczywistość rozszerzona, 284

S

SaaS, Software as a Service, 284
schemat blokowy, 163
serializacja modeli, 282
sezonowość, 203
sieci neuronowe, 219
  głębokie, DNN, 227
  konwolucyjne, SNN, 40, 227
  LSTM, 249
  rekurencyjne, RNN, 227, 241
  splotowe, 40
  sztuczne, ANN, 209
SimpleRNN, 242
słowa, 276
słowo kluczowe
  const, 26
  let, 26
splotowe sieci neuronowe, 40
stemming, 270
szeregi czasowe, 200
sztuczna inteligencja, AI, 20
sztuczne sieci neuronowe, ANN, 209
szumy, 53
szybka transformacja Fouriera, 205

Ś

średnia, 90
środowisko programistyczne, 34, 93

T

TensorFlow.js, 217
testowanie centroidów, 99
tokenizacja, 140, 263
topologia GRU, 246
trendy, 203
trigram, 44
tryb uczenia, 305
tworzenie projektu, 35
TypeScript, 24
typy uczenia, 70

U

uczenie
  maszynowe, 15, 18, 69, 70
  metodą propagacji wstecznej, 214
  modeli na serwerze, 283
  nadzorowane, 70, 75
  nienadzorowane, 70, 72
  przez wzmacnianie, 71, 83
  sieci typu LSTM, 250
  z wykorzystaniem reguł asocjacyjnych, 171
unigram, 44
usprawnienia, 26

W

wariancja, 81
warstwy
  gęste, 221
  GRU, 248
  konwolucyjne, 229
  SimpleRNN, 242
ważenie termów, 257
wątki
  robocze, 286
  usługowe, 286
wejście i wyjście, 309
wektor
  gęsty, 277
  rzadki, 277
wsparcie, 172
wybieranie
  cech, 45
  najlepszego algorytmu, 303
wykres
  danych, 55
  straty i dokładności, 244
wykrywanie krawędzi, 233
wymiarowość, 43
wynik trafności, 258
wyodrębnianie cech, 45
wyznaczenie częstotliwości, 257

Kup ksi k   Pole ksi k

X

XOR, 217

Y

Yarn, 35

Z

zalety języka, 20
zastosowanie reguły asocjacji, 176
zbiór
  elementów, 171
  MNIST, 234
zestaw uczący, 71
znikający gradient, 249

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

Uczenie maszynowe — coś dla wyjadaczy JavaScriptu!

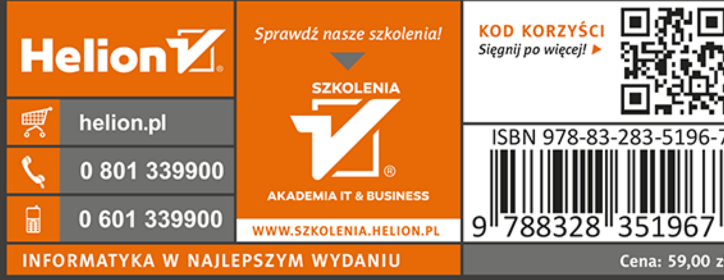
Uczenie maszynowe jeszcze kilka lat temu stanowiło niemal wiedzę tajemną. Nieliczni eksperci w tej dziedzinie publikowali materiały w naukowym, matematycznym języku, który wymagał biegłości w algebrze liniowej czy rachunku wektorowym. Korzystano najczęściej z Pythona i jego bibliotek. Obecnie, wraz ze wzrostem popularności uczenia maszynowego, zwiększają się możliwości jego praktycznej implementacji. Rzeczywista biegłość w tej dziedzinie wymaga jednak dogłębnego zrozumienia mechaniki działania algorytmów stosowanych w uczeniu maszynowym. Implementacja tych algorytmów w JavaScripcie jest znakomitym wyborem: język ten stał się dojrzałym, potężnym i wszechstronnym narzędziem do rozwiązywania złożonych problemów.

Chcesz nauczyć się implementacji algorytmów uczenia maszynowego bez zbytniego zagłębiania się w niuanse matematyczne? Jeśli dodatkowo znasz język JavaScript, ta książka jest dla Ciebie idealnym wyborem. Wyjaśniono w niej, w jaki sposób tworzyć własne implementacje, podano też przykłady przydatnych bibliotek. Sporo miejsca poświęcono sieciom neuronowym, ich architekturze i przykładom zastosowania. Przedstawiono takie zagadnienia jak wykrywanie twarzy, filtrowanie spamu, tworzenie systemów rekomendacji, rozpoznawanie znaków oraz przetwarzanie języka naturalnego. Znalazły się tu również wskazówki dotyczące dobierania odpowiednich bibliotek JavaScriptu, takich jak NaturalNode, brain, harthur oraz klasyfikatory, co umożliwia projektowanie bardziej inteligentnych aplikacji.

Najciekawsze zagadnienia przedstawione w książce:

- potencjał JavaScriptu w uczeniu maszynowym
- algorytmy grupowania, klasyfikacji, reguły kojarzenia
- algorytmy regresji, przewidywanie wzorców i predykcja
- sieci neuronowe i głębokie sieci neuronowe
- uczenie maszynowe w aplikacjach czasu rzeczywistego

Burak Kanber — inżynier, przedsiębiorca. Od ponad 20 lat zajmuje się tworzeniem oprogramowania oraz doradztwem, jest również współtwórcą kilku startupów technologicznych. Specjalizuje się w technologiach sieciowych (języki Python i JavaScript należą do jego ulubionych), inżynierii (fascynują go zwłaszcza systemy kontroli i pojazdy hybrydowe) oraz zagadnieniach zwinnego wytwarzania oprogramowania. Napisał bardzo popularną serię artykułów *Machine Learning in JavaScript*.



ISBN 978-83-283-5196-7
9 788328 351967
Cena: 59,00 zł