Helion

Uczenie maszynowe w C#

Szybkie, sprytne i solidne aplikacje

Packt

Matt R. Cole

Tytuł oryginału: Hands-On Machine Learning with C#

Tłumaczenie: Wojciech Moch

ISBN: 978-83-283-5233-9

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/uczmas
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

1

ROZDZIAŁ

Podstawy uczenia maszynowego

Witam wszystkich na kartach książki *Uczenie maszynowe w C#. Szybkie, sprytne i solidne aplikacje*. Celem, jaki postawiłem sobie w tej książce, jest przekazanie doświadczonym programistom języka C# informacji o wielu dostępnych, otwartych frameworkach do uczenia maszynowego i zaprezentowanie przykładów wykorzystania tych pakietów. Przy okazji porozmawiamy sobie na temat protokołowania, rozpoznawania twarzy i ruchu, drzew decyzyjnych, rozpoznawania obrazów, intuicyjnego uczenia głębokiego, obliczeń kwantowych i licznych innych zagadnień. W wielu przypadkach już po kilku minutach będziesz w stanie zacząć pracę. Mam wielką nadzieję, że w tej książce każdy znajdzie dla siebie coś ciekawego. Powodem, dla którego ją napisałem, było przede wszystkim moje 30-letnie doświadczenie w pracy z programistami.

Od zawsze pracowałem z technologiami firmy Microsoft i zawsze miałem okazję obserwować, jak programiści próbują wyszukać zasoby niezbędne do rozwiązywania codziennych problemów. Musimy pogodzić się z tym, że nikt z nas nie ma czasu na to, by pracować w taki sposób, w jaki by chciał, a jedynie wybrańcy mają szczęście pracować w prawdziwych działach badawczo-rozwojowych. Mimo to przez wszystkie te lata dokonaliśmy ogromnego postępu. Niektórzy, z racji wieku, pamiętają papierową kopię podręcznika do języka C oraz 50 innych książek leżących na biurku. Dzisiaj wystarczy wpisać zapytanie do wyszukiwarki Google, aby otrzymać dokładnie (no, przynajmniej czasami) to, czego nam w danym momencie potrzeba. Teraz jednak, gdy wkraczamy w erę sztucznej inteligencji, nasze środowisko znowu zaczyna się zmieniać. Zapytania do Google, formułowane przez programistę języka C#, nie zawsze zwracają satysfakcjonujące dla niego odpowiedzi, ponieważ niemal wszystkie dotyczą one Pythona, R, MATLAB-a lub Octave. Trzeba też pamiętać, że choć uczenie maszynowe istnieje już od wielu lat, to dopiero niedawno amerykańskie korporacje zainteresowały się tym tematem, a co za tym idzie, zajmuje się nim teraz coraz więcej osób. Dzisiaj mamy pod dostatkiem mocy obliczeniowej, a świat akademicki również uczynił wielkie postępy w próbach upowszechnienia stosowanych tutaj technik. Nasz świat jest jednak dość przerażającym miejscem.

Gdzie ma się zatem zwrócić programista C# i .NET? Odpowiedź na to pytanie rozpocznę od przedstawienia w następnym podrozdziale krótkiej opowieści, która jest równie prawdziwa jak to, że niebo jest niebieskie. Przynajmniej tutaj, na Florydzie.

W tym rozdziale zajmiemy się następującymi zagadnieniami:

- wydobywanie danych,
- **sztuczna inteligencja** (SI) oraz bio-SI,
- uczenie głębokie,
- probabilistyka i statystyka,
- uczenie nadzorowane,
- uczenie nienadzorowane,
- uczenie ze wzmocnieniem,
- lepiej jest kupić, zbudować czy skorzystać z otwartych źródeł.

Wprowadzenie do uczenia maszynowego

Miałem kiedyś przełożonego, któremu powiedziałem, że korzystam z uczenia maszynowego, aby uzyskać więcej informacji z naszych danych. Jego reakcją było pytanie „Co takiego masz nadzieję z tego uzyskać? Będzie to coś, czego jeszcze sam nie wiem ". Jeżeli w trakcie własnej kariery nie zdarzyło Ci się mieć takiego szefa, to gratuluję. Przy okazji, nie szukacie może ludzi do pracy? Najprawdopodobniej jednak każdy już natknął się lub natknie na kogoś takiego. Oto jak rozwinęła się ta rozmowa. I, uprzedzając pytania, nie, nie zwolniłem się z pracy.

Ja: „Celem jest uzyskanie informacji i szczegółów na temat dostępnych funduszy oraz ich związków z rzeczywistymi potrzebami użytkownika".

On: „Ależ ja już to wszystko wiem. Uczenie maszynowe to tylko modne hasło, a ostatecznie chodzi jedynie o dane, którymi sami zarządzamy. Wszystko inne to tylko krzykliwe hasła. Po co mielibyśmy się tym zajmować? W jaki sposób miałoby nam to w czymkolwiek pomóc?"

Ja: „Pozwól, że zadam Ci pytanie. Jak sądzisz, co się dzieje, gdy wpiszesz jakieś zapytanie na stronie Google?"

On (*oczy jak pięciozłotówki i pierwsze oznaki gniewu*)

On: „O co ci chodzi? To oczywiste, że Google porównuje moje zapytanie z innymi zapytaniami, które zgodnie z historią dotyczyły tego samego".

Ja: „OK, a jak to wszystko jest robione?"

On (*gniew zaczyna wzbierać w nim coraz bardziej*)

On: „Z pewnością ich komputery przeszukują sieć i porównują moje kryteria wyszukiwania z innymi".

Ja: „Ale czy kiedykolwiek zastanawiałeś się, w jaki sposób takie zapytanie jest porównywane z miliardami innych i jak wszystkie te dane na temat zapytań są aktualizowane? Z całą pewnością nie mogą tego robić ludzie, bo to nie ta skala".

On: „Oczywiście algorytmy muszą być doskonale dopasowane, tak żeby zwracały to, czego nam potrzeba, albo przynajmniej dawały jakiekolwiek podpowiedzi".

Ja: „Zgadza się, a za tym wszystkim stoi właśnie uczenie maszynowe". (Nie zawsze, ale to już nie jest istotne!)

On: „I tak nie bardzo wiem, czego nowego można się dowiedzieć z tych danych".

Bądźmy szczerzy, czasami nawet najlepsza logika nie przemoże oporów przed jakimikolwiek zmianami. Historia ta ma jednak jeszcze jedno, ważniejsze znaczenie, zupełnie niezwiązane z szefem zaprzeczającym wszystkiemu, czego się do tej pory nauczyliśmy. W świecie uczenia maszynowego znacznie trudniej jest udowodnić lub wykazać, co się tak naprawdę dzieje; czy mechanizmy działają jak należy, czy też nie; w jaki sposób właściwie działają; dlaczego działają (albo nie) tak, a nie inaczej. Jest to szczególnie trudne dla osób, które nie zmagają się na co dzień z problemami programistycznymi. Ale nawet mimo rozmowy z innym programistą możemy mieć problemy ze zrozumieniem zasady działania algorytmu.

Oto kilka pytań, które warto sobie zadać, aby sprawdzić, czy uczenie maszynowe będzie dla nas właściwym rozwiązaniem:

- Czy interesują Cię jedynie *modne hasła* (a może sama firma chce się nimi posługiwać), czy też naprawdę chcesz skorzystać z tego rodzaju rozwiązania?
- Czy masz wszystkie potrzebne Ci dane?
- Czy dane są dostatecznie oczyszczone (więcej na ten temat później)?
- Czy wiesz, jak uzyskać dane, których może Ci brakować? A co ważniejsze, jak zorientować się, że tych danych brakuje?
- Czy masz dużo danych, czy tylko niewielką ich ilość?
- Czy istnieje już znane i sprawdzone rozwiązanie, którego można by użyć?
- Czy wiesz, co chcesz osiągnąć?
- Czy wiesz, w jaki sposób osiągniesz wyznaczony cel?
- Jak to wszystko wyjaśnisz innym?
- Jak wyjaśnisz zainteresowanym działanie tych mechanizmów?

To tylko kilka pytań, na które będziemy musieli sobie odpowiedzieć na początku przygody z uczeniem maszynowym. Chodzi tu przede wszystkim o wyrobienie w sobie czegoś, co nazywam **nastawianiem się na uczenie maszynowe** (ang. *machine learning mindset*).

Ostatnio zdarza się, że osoba, która przygotuje zapytanie SQL zwracające więcej niż tylko jeden wiersz danych, od razy zaczyna się nazywać **naukowcem od danych**. Jeżeli takie określenie pojawia się w CV, to nie ma w tym nic złego. Każdy potrzebuje od czasu do czasu takiego łechtania ego. Ale czy tak naprawdę możemy się nazywać naukowcami od danych? I co właściwie

oznacza to określenie? Mam nadzieję, że przed końcem tej książki znajdziemy odpowiedzi na te pytania, a przynajmniej przygotujemy środowisko ułatwiające każdemu znalezienie własnych odpowiedzi.

Nie wszyscy z nas mają szczęście pracować w środowisku badawczym lub akademickim. Większość z nas musi codziennie gasić małe pożary, a wprowadzane rozwiązania często są działaniami taktycznymi, które trzeba zaimplementować w ciągu dwóch godzin. To właśnie tym zajmujemy się jako programiści języka C#. Przez cały dzień siedzimy przy swoich biurkach, przy odrobinie szczęścia ze słuchawkami na uszach, i tworzymy kod. Ale czy kiedykolwiek mamy dość czasu, żeby przygotować i rozwijać projekt w odpowiedni sposób? Jeżeli by tak było, to w swoich projektach nie mielibyśmy aż tak wielkiego technicznego długu (bo chyba każdy spisuje szczegóły tego długu w swoich projektach).

Musimy działać sprytnie, żeby zdążyć przed wyznaczonym czasem. Niekiedy udaje się to dzięki poświęceniu większej ilości czasu na myślenie, a mniejszej na tworzenie kodu. Podejście akademickie okazuje się czasem zbawienne, bo niczym nie da się zastąpić wiedzy. Niestety większość produkcyjnego kodu tworzonego przez korporacje nie powstaje w takich językach akademickich jak Python, R, Matlab czy Octave. Mimo dostępności tych wszystkich rozwiązań akademickich nie możemy z nich skorzystać, bo nie są dostosowane do charakterystyki naszej pracy.

Zatrzymajmy się na chwilę, żeby w tym miejscu podziękować wszystkim pracującym w społeczności otwartych źródeł. To dzięki ich staraniom mamy wiele doskonałych rozwiązań otwartoźródłowych, które pomagają nam w naszej codziennej pracy. Musimy złożyć wielkie podziękowania za to, że społeczność otwartych źródeł pozwala nam wykorzystywać owoce swojej pracy. Jednym z celów tej książki jest zaprezentowanie wybranych narzędzi otwartoźródłowych i opisanie sposobów ich wykorzystywania. Przy okazji chcę też przedstawić choć trochę wiedzy na temat używanych tu mechanizmów, tak żeby prezentowane narzędzia nie były całkowitą czarną skrzynką.

Właściwie wszędzie widzimy różne modne hasła. Przez pewien czas dojazdy do i z pracy zajmowały mi od dwóch do czterech godzin dziennie, w czasie których często widywałem plakaty reklamowe zawierające pojęcia **„uczenie maszynowe”** albo „sztuczna inteligencja”. Takie hasła znajdziemy właściwie wszędzie, ale co tak naprawdę one oznaczają? SI, uczenie maszynowe, nauka danych, **NLP** (ang. *Natural Language Processing* — przetwarzanie języka naturalnego), wydobywanie danych, neurony… o rety! Może się wydawać, że gdy tylko korporacje zajęły się tym tematem, to świetnie opisana i zdefiniowana dziedzina stała się bałaganiarskim, dostępnym dla wszystkich mikrozarządzanym projektem o całkowicie nierealnych oczekiwaniach. Jeden z moich klientów powiedział nawet: „Nie wiem, co to wszystko znaczy, ale nie chcę pozostać w tyle”.

W pierwszej kolejności musimy nauczyć się prawidłowego podejścia do tworzenia projektu z uczeniem maszynowym. Zacznijmy od podania kilku definicji:

Tom Mitchell przedstawił taką definicję uczenia maszynowego:

> *Mówimy, że program komputerowy uczy się z doświadczenia E, podczas wykonywania klasy zadań T, przy zastosowaniu miary wydajności P, jeżeli jego wydajność realizacji zadań T, mierzona przez P, będzie wzrastać wraz ze zwiększaniem doświadczenia E.*

Przyjmiemy tutaj nieco zmienioną definicję. Mam nadzieję, że pozwoli nam ona na lepszą obronę w przypadku pojawienia się pytań, dlaczego wybieramy tę ścieżkę:

> *Uczenie maszynowe jest zbiorem technik, których można użyć do przetwarzania dużych ilości danych w najefektywniejszy sposób i które pozwolą nam uzyskać z tych danych ważne wyniki i przeprowadzić analizę tych danych.*

A co z takimi szczegółami jak różne używane **techniki**? Nie miejmy złudzeń. Takie techniki jak probabilistyka lub statystyka są cały czas w użyciu, choć na pierwszy rzut oka ich nie widać. Narzędzia, których będziemy używać w naszych przykładach, będą ukrywać szczegóły, tak jak robią to języki Python, R i inne! To powiedziawszy, muszę też zaznaczyć, że wielką niesprawiedliwością z mojej strony byłoby pominięcie pewnych podstawowych elementów. Dlatego będę o nich mówił już za chwilę. Nie chcę tutaj umniejszać ich znaczenia, ponieważ są one niezwykle istotne w naszej pracy, ale celem tej książki jest jak najszybsze umożliwienie programistom języka C# pracy z uczeniem maszynowym. Przedstawię zatem wystarczającą ilość informacji, aby każdy mógł poznać znaczenie poszczególnych chwytliwych haseł, dzięki czemu praca nie będzie polegała wyłącznie na wywoływaniu funkcji API z czarnej skrzynki. Zachęcam każdego do zgłębiania dostępnej wiedzy akademickiej z tej dziedziny. Uczenie maszynowe i sztuczna inteligencja to dziedziny rozwijające się dynamicznie, dlatego warto starać się być na bieżąco. Im większą wiedzę zdobędziesz, tym większa będzie szansa na zaakceptowanie proponowanego projektu!

Skoro wspomniałem już o **chwytliwych hasłach**, to na samym początku postaram się wyjaśnić kilka pojęć. Wydobywanie danych, uczenie maszynowe, sztuczna inteligencja… ta lista jest bardzo długa. W tym miejscu opiszę tylko kilka z nich, ale istnieje prosty sposób na wyobrażenie sobie ich wszystkich.

Jesteś na wycieczce ze swoją rodziną. Przyjmijmy, że masz już dzieci, i odłóżmy na razie rozmowy o tym, *czy jeszcze daleko*! Jedziecie sobie autostradą i nagle jedno z dzieci (jeszcze całkiem malutkie) woła: „CĘŻARÓWKA", wskazując widoczną za oknem ciężarówkę. Dziecko jest jeszcze bardzo małe, skąd zatem wiedziało, że ten akurat rodzaj pojazdu nazywa się tak, a nie inaczej? Wiedziało o tym, ponieważ wcześniej za każdym razem, gdy o to pytało, Twoja odpowiedź brzmiała *Tak* lub *Nie*. Dokładnie tak samo działa uczenie maszynowe. Co więcej, każda odpowiedź *Tak* lub *Nie* oznacza uczenie ze wzmocnieniem, a to zbliża nas już do tematyki uczenia głębokiego. Jak widać, każdy z nas uczył swoje dzieci maszynowo, nawet o tym nie wiedząc.

Mam nadzieję, że to pomogło choć trochę.

Wydobywanie danych

Wydobywanie danych (ang. *data mining*) polega na wyszukiwaniu w ogromnych zbiorach danych pewnych określonych informacji. Przeglądamy dostępne nam dane w poszukiwaniu czegoś specyficznego. Na przykład firmy obsługujące karty kredytowe mogą korzystać z wydobywania danych, aby poznać nawyki kupujących poprzez analizowanie dokonywanych przez nich zakupów oraz miejsc, w jakich te zakupy robią. Takie informacje stają się później bardzo użyteczne, na przykład w tworzeniu spersonalizowanych reklam.

Z drugiej strony uczenie maszynowe koncentruje się na realizacji zadania polegającego na wyszukiwaniu zadanych informacji z wykorzystaniem odpowiednich algorytmów. Czy to jasne?

Tyle powinno na razie wystarczyć. Przedstawię jeszcze odnośnik do świetnego artykułu opisującego wydobywanie danych: *https://blog.udacity.com/2014/12/24-data-science-resources-keep-finger-pulse.html*.

Sztuczna inteligencja

Sztuczna inteligencja to wyższy poziom uczenia maszynowego. Niektórzy definiują ją jako uczenie maszynowe, które działa tak sprawnie albo nawet sprawniej niż sami ludzie. Osobiście nie umiem jeszcze wydać tu osądu. Im więcej czytam codziennych informacji, tym bardziej zastanawiam się, która to inteligencja jest niby sztuczna, ale też czym właściwie jest inteligencja. Istnieje już bardzo wiele najróżniejszych definicji, ale generalnie za sztuczną inteligencję uważana jest maszyna robiąca to, co człowiek mógłby zrobić lub powinien był zrobić, i wykonująca to w taki sposób, że osoba postronna nie jest w stanie odróżnić odpowiedzi maszyny od ludzkich. W każdym razie sztuczna inteligencja to bardzo rozległy temat, który ma tyle znaczeń, ilu jest zajmujących się nim ludzi.

Bio-SI

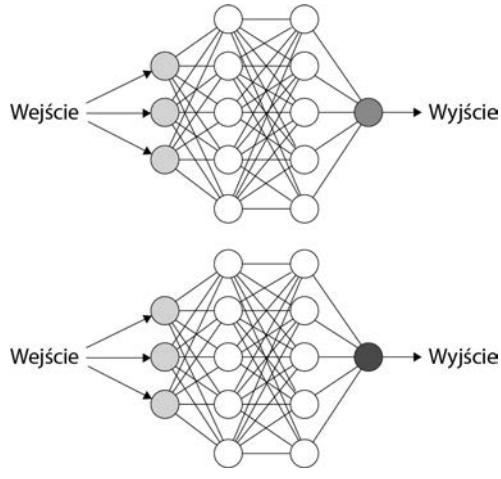
Termin Bio-SI oznacza techniki umieszczające element biologiczny obok elementu obliczeniowego. Genotypy, fenotypy, neurony, neurony lustrzane, neurony kanoniczne, synapsy… te wszystkie pojęcia pojawiają się w ramach ogólnej kategorii **sztucznych sieci neuronowych** (ang. *Artificial Neural Networks* — ANN)! Bio-SI są najczęściej stosowane w medycynie. Na razie nie musimy się tym więcej zajmować, ale dobrze jest wiedzieć, że takie pojęcie istnieje i że to biologia jest bazą dla jego zastosowania.

Uczenie głębokie

Przez wiele lat uważano, że sieci neuronowe (wykorzystujące koncepcję ukrytych warstw) potrzebują zaledwie jednej ukrytej warstwy do rozwiązania dowolnego problemu. Dzięki przyrostowi mocy obliczeniowej, redukcji cen sprzętu oraz rozwojowi algorytmów sieci neuronowych dzisiaj powszechnie stosuje się setki, a nawet tysiące **ukrytych warstw** w tworzonych sieciach. Zwiększenie liczby ukrytych warstw to między innymi jedna z cech uczenia głębokiego. Oto obrazowe porównanie, które powinno wyjaśnić kilka rzeczy:

Brak ukrytych warstw

Jak widać na poniższym rysunku, w prezentowanej tutaj sieci wykorzystano kilka ukrytych warstw:



Wiele ukrytych warstw (białe kółka)

Probabilistyka i statystyka

Można w to wierzyć lub nie, ale to właśnie tym zajmujemy się przede wszystkim. Działania te zostały tylko dobrze ukryte przed naszym postrzeganiem. Pozwólcie jednak, że przedstawię tutaj mocno uproszczoną i skróconą lekcję. Tak na rozruszanie.

Widzisz, że po śniegu idzie niedźwiedź polarny. Zaczynasz się zastanawiać, jakie odciski łap zostawia na śniegu. To właśnie jest probabilistyka! Później widzisz na śniegu odciski jakichś

łap i zastanawiasz się, czy zostawił je niedźwiedź polarny. A to z kolei jest statystyka! I to wszystko, lekcja zakończona. Teraz zapewne zastanawiasz się, czy autor ma jeszcze wszystkie klepki, dlatego spieszę z dodatkowym wyjaśnieniem.

- Probabilistyka zajmuje się przewidywaniem prawdopodobieństwa przyszłych zdarzeń.
- Statystyka zajmuje się analizowaniem częstości występowania przeszłych zdarzeń.

Rozpoczynanie projektu uczenia maszynowego

Pomówmy teraz o tym, jak będziemy rozpoczynać nasz projekt uczenia maszynowego — przy okazji będziemy pracować nad rozwijaniem własnego nastawienia do tej techniki. Zaczniemy od zdefiniowania podstawowych kroków, które trzeba wykonać w ramach przygotowań do każdego nowego projektu. Takie kroki możemy podzielić na przedstawione dalej kategorie.

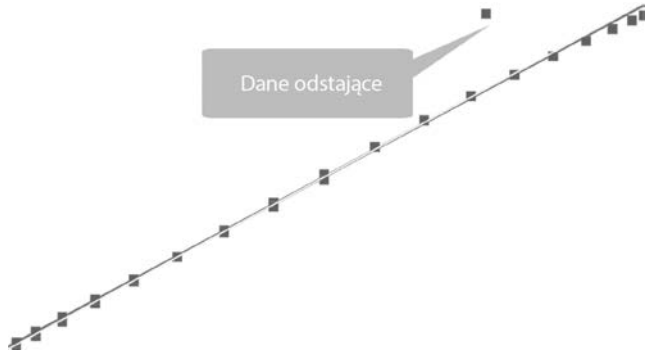
Zbieranie danych

Do swojej dyspozycji mamy niezliczone ilości danych, zaczynając od baz danych SQL i NoSQL, poprzez pliki Excela, bazy danych Accessa, aż po pliki tekstowe i wiele, wiele innych. Musisz zatem zdecydować, gdzie mają znajdować się Twoje dane, jak zostaną sformatowane oraz jak będziesz je importować i przetwarzać. Trzeba zawsze pamiętać, że niczym nie da się zastąpić dużych ilości danych testowych i treningowych. Nie można też zapomnieć o odpowiedniej jakości tych danych. Metody typu „śmieci na wejściu, śmieci na wyjściu" w ogóle nie sprawdzają się w przypadku uczenia maszynowego!

Przygotowanie danych

Jak już wcześniej mówiłem, niczym nie można zastąpić danych o odpowiedniej jakości. Czy jakichś danych w zbiorze brakuje? Czy są one źle przygotowane albo całkowicie niepoprawne? Nie można też zapomnieć o kolejnym pojęciu, z którym często będziemy się spotykać: dane odstające (ang. *data outliers*). Są to nieprzyjemne elementy naszych danych, które w żaden sposób nie pasują do pozostałych danych. Zdarza Ci się takie widywać? Jeżeli zdarza się, to trzeba sobie zadać pytanie, czy powinny się one tam znajdować, a jeżeli tak, to w jaki sposób należałoby je traktować. Jeżeli nie masz pewności, to na rysunku na następnej stronie przedstawiam takie odstające dane w zbiorze wykorzystanym do przygotowania wykresu.

W statystyce dane odstające są punktem obserwacji, który znajduje się daleko od pozostałych obserwacji — czasami jest bardzo, a czasami mniej odległy. Takie odstające wartości mogą być wynikiem zmienności procesu pomiarowego, wskazywać na błędy w wykonaniu eksperymentu, ale mogą też być całkowicie poprawnymi wynikami. Jeżeli w swoich danych dostrze-

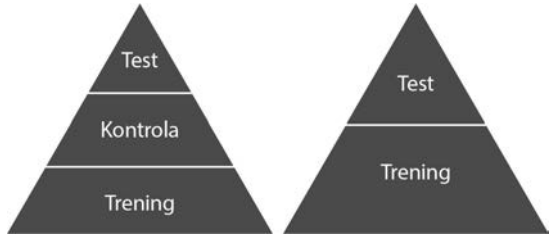


gasz elementy odstające, to musisz postarać się zrozumieć, z czego one wynikają. Mogą wskazywać na istnienie pewnych błędów w wykonywanych pomiarach, a stosowane algorytmy mogą nie być na tyle stabilne, żeby poradzić sobie z istnieniem elementów odstających.

Wybranie modelu i trening

Podczas tworzenia i trenowania modelu musisz zwracać uwagę na kilka ważnych spraw:

- Musisz wybrać algorytm uczenia maszynowego właściwy dla danego zadania, który będzie odpowiednio reprezentował dostępne dane. Dostępne dane musisz następnie podzielić na trzy podzbiory: **treningowy**, **kontrolny** i **testowy**. Reguły proporcji podziału są różne w zależności od wielkości zbioru danych, który masz do dyspozycji. Na przykład jeżeli dysponujesz 10 000 wierszy, to dobrym podziałem byłoby 20% na dane treningowe i 80% na testowe. Jeżeli jednak możesz skorzystać z 10^8 wierszy danych, to na dane treningowe wystarczy 5% z nich, a pozostałe 95% możesz przydzielić do puli testowej.
- Jednej zasady musisz trzymać się co do joty. Niezależnie od tego, jakich metod użyjesz do testowania, treningu i kontroli danych, to *zawsze całość tych danych musi pochodzić z tego samego zbioru*! To jest naprawdę bardzo ważne. Nigdy nie wolno wykorzystywać do treningu danych z jednego zbioru, a potem w ramach testów używać danych z innego zbioru. To będzie powodować jedynie frustrację. Zawsze musisz przygotować sobie wielki zbiór danych na potrzeby treningu, testów i kontroli.
- Dane kontrolne mogą zostać użyte do sprawdzenia poprawności danych testowych jeszcze przed ich wykorzystaniem w ramach testów. Niezależnie od tego, jak podzielisz swoje dane, zawsze uzyskasz zbiór danych do treningu i zbiór danych do testów. Jednym z celów stosowanego algorytmu powinno być uzyskanie elastyczności wystarczającej do poradzenia sobie z danymi, których wcześniej nie widzieliśmy. Tego nie da się uzyskać, jeżeli będziemy testować z wykorzystaniem tych samych danych, których używaliśmy do treningu algorytmu. Na rysunku na następnej stronie przedstawiam dwie metody podziału zbioru danych, na podzbiór testowy i treningowy (w jednej metodzie wyznaczany jest jeszcze zbiór kontrolny, a w drugiej nie).



Ocena modelu

Po wykorzystaniu danych treningowych możesz przejść do testowania i kontrolowania swojego modelu, używając do tego przygotowanego wcześniej zbioru danych. To właśnie w tym momencie możesz się przekonać, jak dobrze Twój model radzi sobie z danymi, których wcześniej nie widział. Jeżeli nie działa on tutaj właściwie, to wracasz na start, nie pobierasz 200 dolarów i zaczynasz poprawiać cały proces!

Poprawianie modelu

Podczas dokonywania oceny swojego modelu na pewnym etapie możesz stwierdzić, że konieczne będzie zastosowanie innego modelu albo uzupełnienie istniejącego o nowe funkcje lub zmienne albo też hiperparametry, aby w ten sposób poprawić jego skuteczność i wydajność. Dobrą metodą unikania takich przypadków jest poświęcenie większej ilości czasu na *zbieranie* i *przygotowywanie* danych. Jak już mówiłem wcześniej, niczym nie da się zastąpić dużych ilości poprawnych danych.

Jeżeli mimo wszystko musisz poprawić swój model (bo na pewno kiedyś to nastąpi), to masz tutaj do dyspozycji kilka możliwych rozwiązań:

- przeszukiwanie siatki,
- wyszukiwanie losowe,
- optymalizacja Bayesa,
- optymalizacja z użyciem gradientów,
- optymalizacja ewolucyjna.

Teraz przyjrzyjmy się przykładowemu zbiorowi danych — słynnemu i powszechnie używanemu zbiorowi danych Iris.

Zbiór danych o irysach

Zbiór danych irysów jest zbiorem opisującym różne kwiaty, który został utworzony w 1936 roku przez biologa Ronalda Fishera. W zbiorze tym znajduje się 50 próbek dla każdego gatunku irysów (*Iris setosa*, *Iris virginica*, *Iris versicolor*). Każda z próbek opisywana jest czterema

cechami (długością i szerokością dwóch rodzajów płatków). Połączenie tych danych tworzy liniowo dyskryminujący model, który umożliwia rozróżnienie poszczególnych gatunków.

Przejście od kwiatu do danych można zobaczyć na poniższym rysunku:

	A	B	C	D	E
1	Długość płatka	Szerokość płatka	Długość mniejszego płatka	Szerokość mniejszego płatka	Klasa
2	5,10	3,50	1,40	0,20	Iris-setosa
3	4,90	3,00	1,40	0,20	Iris-setosa
4	4,70	3,20	1,30	0,20	Iris-setosa
5	4,60	3,10	1,50	0,20	Iris-setosa
6	5,00	3,60	1,40	0,20	Iris-setosa
7	5,40	3,90	1,70	0,40	Iris-setosa
8	4,60	3,40	1,40	0,30	Iris-setosa
9	5,00	3,40	1,50	0,20	Iris-setosa
10	4,40	2,90	1,40	0,20	Iris-setosa
11	4,90	3,10	1,50	0,10	Iris-setosa
12	5,40	3,70	1,50	0,20	Iris-setosa
13	4,80	3,40	1,60	0,20	Iris-setosa
14	4,80	3,00	1,40	0,10	Iris-setosa
15	4,30	3,00	1,10	0,10	Iris-setosa
16	5,80	4,00	1,20	0,20	Iris-setosa
17	5,70	4,40	1,50	0,40	Iris-setosa
18	5,40	3,90	1,30	0,40	Iris-setosa
19	5,10	3,50	1,40	0,30	Iris-setosa
20	5,70	3,80	1,70	0,30	Iris-setosa
21	5,10	3,80	1,50	0,30	Iris-setosa
22	5,40	3,40	1,70	0,20	Iris-setosa
23	5,10	3,70	1,50	0,40	Iris-setosa
24	4,60	3,60	1,00	0,20	Iris-setosa
25	5,10	3,30	1,70	0,50	Iris-setosa

Teraz musimy wziąć dostępne nam informacje na temat wizualnej reprezentacji obiektu naszych prac (kwiatu) i przekształcić je w coś, co będzie zrozumiałe dla komputera. W tym celu musimy podzielić znane nam informacje na temat kwiatów na kolumny (cechy) i wiersze (elementy danych), tak jak na pierwszym rysunku na następnej stronie.

Dzięki temu, że wszystkie wartości pomiarów zapisane są w formacie zrozumiałym dla komputera, w pierwszym kroku powinniśmy się upewnić, że w całym zbiorze nie ma brakujących lub zniekształconych danych, które mogłyby powodować kłopoty. Jeżeli spojrzysz na komórki wyróżnione kolorem (zobacz pierwszy rysunek na następnej stronie), to zauważysz, że są to miejsca z brakującymi danymi. Zanim będziemy mogli podać taki zbiór danych aplikacji, musimy uzupełnić wszystkie brakujące informacje. Gdy dane zostaną już odpowiednio przygotowane, a ich poprawność potwierdzona, będziemy mogli przystąpić do pracy. Jeżeli teraz uruchomimy program kontrolny dla naszych danych, dostępny w frameworku **Encog3**$_4$, to jego odpowiedź powinna potwierdzić, że mamy do dyspozycji 150 zbiorów danych (zobacz drugi rysunek na następnej stronie).

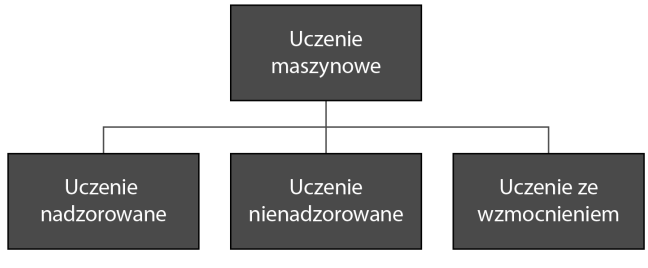
1	Długość płatka	Szerokość płatka	Długość mniejszego płatka	Szerokość mniejszego płatka	Klasa
2	5,10	3,50	1,40	0,20	Iris-setosa
3	4,90	3,00	1,40	0,20	Iris-setosa
4	4,70	3,20	1,30	0,20	Iris-setosa
5	4,60	3,10	1,50	0,20	Iris-setosa
6	5,00	3,60	1,40	0,20	Iris-setosa
7	5,40	3,90	1,70		Iris-setosa
8	4,60	3,40	1,40	0,30	Iris-setosa
9	5,00	3,40	1,50	0,20	Iris-setosa
10	4,40	2,90	1,40	0,20	Iris-setosa
11	4,90	3,10	1,50	0,10	Iris-setosa
12	5,40		1,50	0,20	Iris-setosa
13	4,80	3,40	1,60	0,20	Iris-setosa
14	4,80	3,00	1,40	0,10	Iris-setosa
15	4,30	3,00	1,10	0,10	Iris-setosa
16	5,80	4,00	1,20	0,20	Iris-setosa
17	5,70	4,40	1,50	0,40	
18	5,40	3,90	1,30	0,40	Iris-setosa
19	5,10	3,50	1,40	0,30	Iris-setosa
20	5,70	3,80	1,70	0,30	Iris-setosa
21	5,10	3,80	1,50	0,30	Iris-setosa
22	5,40	3,40	1,70	0,20	Iris-setosa
23	5,10	3,70	1,50	0,40	Iris-setosa
24	4,60	3,60	1,00	0,20	Iris-setosa
25	5,10	3,30	1,70	0,50	Iris-setosa

```
0 : Analyzing file
150 Datasets validated
```

Rodzaje uczenia maszynowego

Teraz pokrótce zapoznamy się z różnymi rodzajami uczenia maszynowego, o których będziemy mówić w tej książce, zaczynając już od następnego rozdziału. Ważne jest, żeby znać poszczególne pojęcia, ponieważ wcześniej czy później pojawią się one w naszej pracy, a im więcej będziemy wiedzieć i rozumieć, tym łatwiej przyjdzie nam rozwiązywanie pojawiających się problemów i wyjaśnianie ich swoim kolegom.

Oto prosty diagram przedstawiający trzy kategorie uczenia maszynowego:



Uczenie nadzorowane

Ten rodzaj uczenia maszynowego jest używany do przewidywania wyników na podstawie przedstawionych danych. Przedłożone instrukcje są (a przynajmniej powinny być) jasne i dokładne, przez co ta klasa metod otrzymała etykietę uczenia **nadzorowanego**. W zasadzie uczymy tutaj funkcję, która odwzorowuje dane wejściowe na dane wyjściowe, wykorzystując w tym procesie pary wejścia i wyjścia. Taka funkcja wywodzona jest z danych treningowych (**etykiet**), które jednoznacznie informują funkcję o oczekiwanym wyniku. W uczeniu nadzorowanym zawsze mamy dane wejściowe i przypisane im dane wyjściowe (a dokładniej — oczekiwaną wartość wyjścia). Mówiąc bardziej formalnie, ten rodzaj algorytmów stosuje technikę nazywaną **odchyleniem wywiedzionym** (ang. *inductive bias*), co oznacza, że istnieje pewien zbiór założeń, które algorytm będzie wykorzystywał do przewidywania danych wyjściowych na podstawie danych wejściowych, z którymi mógł, choć nie musiał się już zetknąć.

W uczeniu nadzorowanym zazwyczaj mamy dostęp do zbioru X właściwości (X_1, X_2, X_3, …, X_n), zmierzonych w ramach obserwacji, oraz odpowiedzi Y, która również została zmierzona w ramach tych samych n obserwacji. Na tej podstawie próbujemy przewidzieć wartość Y, wykorzystując do tego wartości X_1, X_2, X_3, …, X_n.

Przykładami metod uczenia maszynowego mogą być takie modele jak maszyny wektorów nośnych (ang. *Support Vector Machines* — SVM), regresja liniowa, naiwny klasyfikator bayesowski albo metody korzystające ze struktur drzewiastych.

Następnie pokrótce omówimy kilka szczegółów, o których musimy pamiętać, zajmując się metodami uczenia nadzorowanego. Podaję je tutaj w kolejności losowej:

- kompromis odchylenie – wariancja,
- ilość danych treningowych,
- wymiarowość przestrzeni wejścia,
- nieprawidłowe wartości wyjścia,
- heterogeniczność danych.

Kompromis odchylenie – wariancja

Zanim opowiem o kompromisach między odchyleniem a wariancją, dobrze będzie przypomnieć sobie znaczenie poszczególnych pojęć.

Mówiąc o kompromisie odchylenia i wariancji, za odchylenie uznajemy błędy wynikające z niewłaściwych założeń algorytmu uczenia. Wysokie odchylenie powoduje powstanie **słabego dopasowania** (ang. *under-fitting*), czyli zjawiska sprawiającego, że algorytm będzie pomijał ważne dane na temat związków między wejściem i wyjściem.

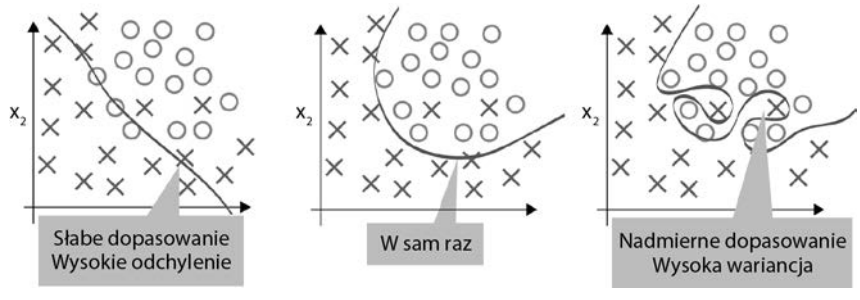
Z kolei wariancja jest błędem czułości na niewielkie fluktuacje w zbiorze treningowym. Wysoka wariancja może sprawić, że algorytm będzie interpretował szumy jako ważne dane, zmieniając oczekiwane wartości wyjścia. To zjawisko nazywane jest **nadmiernym dopasowaniem** (ang. *over-fitting*).

Pomiędzy odchyleniem a wariancją istnieje związek, z którego musi zdawać sobie sprawę każdy programista zajmujący się uczeniem maszynowym. Jest to bezpośrednio powiązane ze słabym i nadmiernym dopasowaniem naszych danych. Mówimy, że algorytm uczący ma wysoką wariancję dla danego wejścia, jeżeli przewiduje on inne wartości wyjściowe przy zastosowaniu innego zestawu treningowego. Oczywiście nie jest to sytuacja pożądana.

Algorytm uczenia maszynowego o **niskim odchyleniu** musi być na tyle elastyczny, żeby dobrze dostosowywać się do danych. Jeżeli jednak algorytm będzie zbyt elastyczny, to każdy zbiór danych treningowych i testowych będzie inaczej dopasowywany, co będzie oznaczało wysoką wariancję.

Nasz algorytm musi być na tyle elastyczny, żeby możliwe było skorygowanie tego kompromisu, realizowane albo na podstawie informacji algorytmicznych, albo za pomocą parametru nastawianego przez użytkownika.

Na poniższym rysunku przedstawiam prosty model o wysokim odchyleniu (po lewej) oraz znacznie bardziej złożony model o wysokiej wariancji (po prawej).



Ilość danych treningowych

Jak już wcześniej wielokrotnie mówiłem, jeżeli swoje zadanie chcemy zrealizować poprawnie i w pełni, to niczym nie da się zastąpić odpowiedniej ilości danych. To z kolei ma bezpośredni związek ze złożonością stosowanego algorytmu uczącego. Mniej złożony algorytm o wysokim odchyleniu i niskiej wariancji może być lepiej uczony za pomocą mniejszych ilości danych. Jeżeli jednak algorytm jest bardzo skomplikowany (wiele wejściowych właściwości, parametrów itp.), to będzie trzeba zastosować znacznie większe zbiory treningowe, pozwalające na uczenie się przy niskim odchyleniu i wysokiej wariancji.

Wymiarowość przestrzeni wejścia

W każdym zadaniu związanym z uczeniem dane wejściowe będą miały postać **wektora**. Dokładniej, będzie to **wektor właściwości**, co oznacza, że już same właściwości danych mogą w dużym stopniu wpływać na algorytm uczący. Jeżeli wejściowe wektory właściwości są bardzo duże, co nazywane jest wielkowymiarowością (ang. *high-dimensionality*), to uczenie może być bardzo kłopotliwe, nawet jeżeli interesować będzie nas tylko część z tych właściwości. Czasami dodatkowe wymiary dezorientują algorytm uczący, przez co wzrasta poziom wariancji. To z kolei będzie oznaczało konieczność takiego dostosowania algorytmu, żeby miał on mniejszą wariancję, a większe odchylenie. Czasami łatwiejszym rozwiązaniem (o ile jest to w ogóle możliwe) jest usunięcie z danych dodatkowych właściwości, co spowoduje poprawienie sprawności uczenia funkcji.

Po tym wstępie mogę powiedzieć, że istnieje technika nazywana **redukcją wymiarowości**, która jest wykorzystywana przez niektóre algorytmy uczenia maszynowego. Takie algorytmy samodzielnie wykrywają i usuwają z danych nieistotne właściwości.

Nieprawidłowe wartości wyjścia

Należy zadać tutaj sobie pytanie, ile błędów znajduje się w oczekiwanych danych wyjściowych z naszego algorytmu uczenia maszynowego. Jeżeli jest ich zbyt dużo, to oznacza, że algorytm uczący próbuje zbyt dokładnie dopasować dane, co prowadzi do powstania wspomnianego wcześniej efektu *nadmiernego dopasowania*. Nadmierne dopasowanie może być wynikiem nieprawidłowych danych albo skutkiem zastosowania algorytmu o zbyt dużej złożoności w stosunku do potrzeb zadania. W takiej sytuacji trzeba albo poprawić stosowany algorytm, albo poszukać innego, który charakteryzowałby się wyższym odchyleniem i niższą wariancją.

Heterogeniczność danych

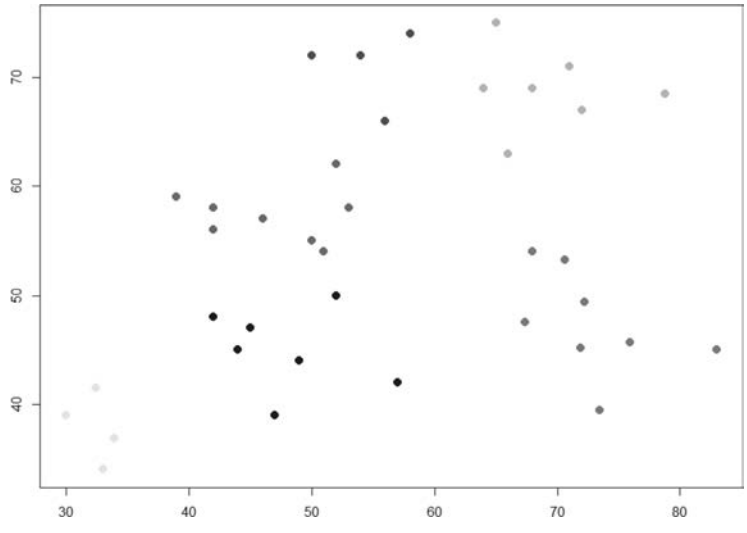
Zgodnie ze słownikiem języka polskiego „heterogeniczność" oznacza *cechę polegającą na składaniu się z niejednakowych lub zróżnicowanych elementów*. Z naszego punktu widzenia oznacza to, że wektory właściwości zawierają właściwości wielu różnych rodzajów. Jeżeli dotyczy to naszej aplikacji, to być może lepszym rozwiązaniem byłoby zastosowanie innego algorytmu uczącego, lepiej dostosowanego do wybranego zadania. Niektóre algorytmy uczące wymagają też przeskalowania danych do określonego przedziału wartości, takiego jak *[0 – 1]*, *[–1 – 1]* itd. Gdy przejdziemy do algorytmów uczących, wykorzystujących jako bazy funkcje odległości, takich jak najbliższy sąsiad albo metody wektorów wspierających, zauważysz, że są one wyjątkowo czułe pod tym względem. Z drugiej strony algorytmy korzystające ze struktur drzewiastych (drzewa decyzji itd.) całkiem dobrze sobie z tym radzą.

Na zakończenie tego opisu powiem jeszcze, że pracę należy zawsze zaczynać z najmniej złożonym, ale najlepiej dopasowanym algorytmem i zapewnić mu prawidłowo zebrane i przygotowane dane. Na takiej podstawie można prowadzić eksperymenty z różnymi algorytmami uczącymi i dopasowywać je tak, żeby jak najlepiej działały w wybranej sytuacji. Nie należy

tutaj mieć złudzeń. Dostosowywanie algorytmów nie jest prostym zadaniem i ostatecznie zajmuje znacznie więcej czasu, niż możemy na to poświęcić. Dlatego zawsze na samym początku trzeba zapewnić sobie odpowiednią ilość danych!

Uczenie nienadzorowane

W przeciwieństwie do uczenia nadzorowanego wersja nienadzorowana ma więcej swobody w określaniu oczekiwanego wyniku. Algorytmy z tej kategorii traktują dane jako całość i nie nadają żadnej właściwości większego znaczenia niż pozostałym. Algorytmy te uczą się na podstawie zbiorów danych wejściowych, dla których dane wyjściowe nie otrzymują żadnych etykiet. Przykładem modelu nienadzorowanego może być algorytm centroidów (z analizy skupień). Świetnie nadaje się on do wyszukiwania wzorców w danych, które są w jakiś sposób powiązane z danymi wejściowymi. Główna różnica w stosunku do tego, czego dowiedzieliśmy się o modelach nadzorowanych, jest taka, że mamy tu tylko właściwości X_1, X_2, X_3, …, X_n uzyskane w ramach n obserwacji. Tym razem nie chcemy przewidywać wartości Y, ponieważ tutaj taka wartość nie istnieje. W tym przypadku interesuje nas tylko wykrywanie wzorców występujących we właściwościach danych, tak jak poniżej:



Na powyższym rysunku można zauważyć, że takie dane skłaniają nas do używania rozwiązań nieliniowych, w których dane grupowane są zależnie od poziomu ważności. Takie rozwiązania nazywane są **nieliniowymi**, ponieważ nie ma tu możliwości poprowadzenie linii prostej, która dzieliłaby dane na poszczególne kategorie. Uczenie nienadzorowane umożliwia nam zajęcie się problemem, choć nie mamy żadnego pojęcia o tym, jak wygląda wynik, ani nawet jak powinien wyglądać. Struktury wywodzone są z otrzymanych danych, w przeciwieństwie do przydzielania etykiet danym wyjściowym, stosowanego w uczeniu nadzorowanym. Takie struktury powstają zwykle poprzez **grupowanie** powiązanych ze sobą danych.

Załóżmy na przykład, że mamy dane 10^8 genów, pochodzące z przeprowadzonego wcześniej eksperymentu nad genami. Chcielibyśmy teraz pogrupować te dane w podobnych segmentach, takich jak kolor włosów, długość życia, waga.

Drugi przykład odnosi się do efektu, który często nazywany jest **efektem przyjęcia koktajlowego**. Chodzi tutaj o umiejętność naszego mózgu polegającą na skupieniu się na jednej rzeczy i odfiltrowaniu pozostałego **szumu**.

W obu przykładach możemy wykorzystać grupowanie w celu osiągnięcia zakładanych celów.

Uczenie ze wzmocnieniem

Uczenie ze wzmocnieniem to sytuacja, w której maszyna trenowana jest w celu uzyskania określonego wyniku. Jedynym celem takiego działania jest maksymalizacja wydajności i/lub skuteczności. Algorytm jest wtedy **nagradzany** za podjęcie właściwej decyzji i **karany** za niewłaściwą. Ciągły trening stosowany jest w celu nieustającego podnoszenia wydajności. Taki nieprzerwany proces uczenia zmniejsza liczbę niezbędnych interakcji z człowiekiem. Modele Markowa są przykładem mechanizmów uczenia ze wzmocnieniem, a świetnym przykładem ich zastosowania są samochody autonomiczne. Cały czas współpracują one ze swoim otoczeniem, szukają przeszkód, ograniczeń prędkości albo pieszych, mierzą odległości od tych obiektów i na tej podstawie podejmują (taką mamy nadzieję) właściwe decyzje.

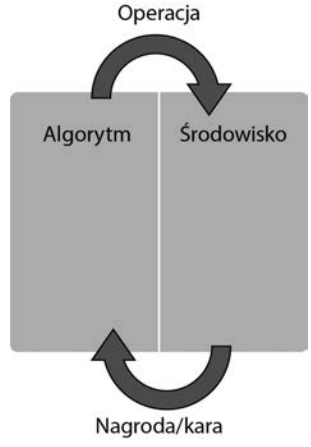
Tym, co wyróżnia uczenie ze wzmocnieniem, jest fakt, że nie pracujemy tu z poprawnymi danymi wejściowymi i wyjściowymi, ale koncentrujemy się na zwiększeniu wydajności. Oznacza to, że w jakiś sposób staramy się balansować między tym, czego algorytm nauczył się do tej pory, a pojawiającymi się nowymi danymi.

Algorytm wykonuje operację na swoim środowisku i otrzymuje za nią nagrodę lub karę. Następnie proces się powtarza i trwa w nieskończoność, tak jak na rysunku na następnej stronie. Możesz sobie tylko wyobrażać, ile razy na sekundę powtarza się ten cykl w niepozornej autonomicznej taksówce, która podwiozła Cię do hotelu.

Lepiej kupić, zbudować czy skorzystać z otwartych źródeł?

Teraz możemy zadać sobie to najważniejsze pytanie: „Lepiej będzie kupić, zbudować samodzielnie czy skorzystać z otwartych źródeł?”.

Moja rekomendacja jest jednocześnie powodem, dla którego tworzę tę książkę — chcę Cię zachęcić do zagłębienia się w świat otwartych źródeł. Zdaję sobie sprawę z tego, że wielu programistów cierpi na syndrom „niepowstałego tutaj”, jednak powinniśmy spojrzeć realnie na siebie i nie trwać w tym stanie. Czy na pewno uważasz, że masz doświadczenie pozwalające



zrobić to lepiej i szybciej, a także przetestować powstały produkt w rozsądnym czasie, a potem porównać go z tym, co już istnieje? Najpierw powinniśmy rozejrzeć się i sprawdzić, jakie narzędzia już powstały i z jakich możemy skorzystać. Istnieje wiele wspaniałych, otwartoźródłowych zestawów narzędziowych, których twórcy włożyli ogromny wysiłek i poświęcili wiele godzin ciężkiej pracy w ich przygotowanie i przetestowanie. Oczywiście pakiety otwartoźródłowe nie będą rozwiązaniem dla każdego, ale nawet jeżeli nie użyjesz ich w swoich aplikacjach, to z pewnością możesz z nich wydobyć wiele informacji poprzez samo eksperymentowanie.

Kupowanie zwykle nie jest żadnym rozwiązaniem. Jeżeli nawet uda Ci się znaleźć coś wartego zainteresowania, to zapewne nie uzyskasz pozwolenia na zakup, ponieważ takie pakiety kosztują niemałe pieniądze. A co będzie, jeżeli zajdzie potrzeba zmodyfikowania takiego produktu, żeby robił to, czego potrzebujesz? Życzę powodzenia przy próbach uzyskania dostępu do kodu źródłowego albo zmiany priorytetów zespołu rozwijającego ten produkt. To się nie zdarzy, a przynajmniej nie na tyle wcześnie, żeby mogło mieć wpływ na Twoją pracę.

A co z samodzielnym przygotowaniem odpowiednich narzędzi? Jesteśmy w końcu programistami, więc powinniśmy być w stanie to zrobić! Zanim jednak uruchomisz Visual Studio i przystąpisz do tworzenia, zastanów się dobrze nad tym, w co się pakujesz.

W tym świetle pierwszym wyborem powinny być zawsze rozwiązania otwartoźródłowe. Możesz je wykorzystać w swojej pracy (o ile pozwalają na to warunki licencji) i dostosować je do swoich potrzeb (użyć kontraktów, przygotować więcej testów jednostkowych, poprawić dokumentację itp.).

Dodatkowa lektura

Co prawda prezentowany tam kod zapisany jest w językach Python i R, ale i tak zachęcam do poszerzenia wiedzy na omawiane w tym rozdziale tematy poprzez przejrzenie zawartości witryny Jasona Brownlee, znajdującej się pod adresem *https://machinelearningmastery.com/*. Dostępne tam wyjaśnienia tajników uczenia maszynowego nie mają sobie równych. Można tam

znaleźć ogromną ilość doskonałych informacji. Wszystkie opisy są jasne i naprawdę dogłębnie prezentują każdy temat. Polecam zapoznanie się z tą witryną i blogiem oraz pozyskanie z nich tyle, ile tylko będzie możliwe.

Podsumowanie

W tym rozdziale omówiliśmy wiele zagadnień związanych z uczeniem maszynowym w języku C#, różnymi strategiami implementowania kodu (kupić, zbudować czy skorzystać z otwartych źródeł), a dodatkowo pokrótce przedstawiliśmy kilka ważnych definicji. Mam nadzieję, że to wszystko przygotowało Cię na kolejne rozdziały.

Zanim zajmiemy się tworzeniem kodu i całych aplikacji, chciałbym przedyskutować jeszcze pewną sprawę, która jest dla mnie niezmiernie ważna: protokołowanie. To coś, co wszyscy robimy (a przynajmniej powinniśmy robić), dlatego chciałem zaprezentować fenomenalne narzędzie do protokołowania. W tej książce będziemy z niego korzystać bardzo często, dlatego warto zapoznać się z nim wcześniej, co zrobimy już w następnym rozdziale.

Odwołania

- Praca własna użytkownika Nicoguaro, CC BY 4.0, *https://commons.wikimedia.org/w/index.php?curid=46257808*.
- Creative Commons Attribution-ShareAlike 3.0 Unported.
- *https://en.wikipedia.org/wiki/Cocktail_party_effect*.
- Framework Encog — własność Jeff Heaton/Heaton Research.

Skorowidz

Q

R

S

T

U

W

Z

Uczenie maszynowe — najlepiej z wydajnym C#!

Uczenie maszynowe weszło już do kanonu technologii informatycznych. Praktyczne umiejętności w tej dziedzinie powinien posiadać każdy programista i analityk. Standardowo do rozwiązań związanych z machine learning stosuje się Pythona i opracowane dla niego biblioteki, niemniej równie skutecznie można do tego celu używać innych języków programowania. Trzeba jedynie dobrze zaznajomić się z wdrożeniami algorytmów uczenia maszynowego. Niezwykle ciekawym rozwiązaniem jest pisanie takich implementacji w C#. Przemawiają za tym nie tylko zalety samego języka, ale i to, że większość aplikacji dla profesjonalistów jest pisana w C# przy użyciu takich narzędzi jak Visual Studio, SQL Server, Unity czy Microsoft Azure.

Ta książka jest przeznaczona dla doświadczonych programistów C#, którzy chcą nauczyć się technik machine learning, deep learning i sztucznej inteligencji. Opisano tu dostępne narzędzia do uczenia maszynowego, dzięki którym można łatwo budować inteligentne aplikacje .NET wykorzystujące takie rozwiązania jak wykrywanie obrazów lub ruchu, wnioskowanie bayesowskie, głębokie uczenie i głęboka wiara. Omówiono zasady implementacji algorytmów uczenia nadzorowanego i nienadzorowanego oraz ich zastosowanie w budowie modeli predykcji. Przedstawiono różne techniki, od prostej regresji liniowej, przez drzewa decyzyjne i SVM, po zaawansowane rozwiązania, takie jak sztuczne sieci neuronowe, autoenkodery lub uczenie ze wzmocnieniem.

Najciekawsze zagadnienia przedstawione w książce:

- podstawy uczenia maszynowego
- wykorzystywanie logiki rozmytej
- mapy samoorganizujące się
- framework Kelp.Net i jego integracja z systemem ReflectInsight
- realia obliczeń kwantowych

Matt R. Cole — od 30 lat programuje dla systemu Windows; biegle posługuje się językami: C, C++, C# oraz platformą .NET. Napisał system generowania mowy oraz system VOIP dla NASA, którego używano na promach kosmicznych i stacji kosmicznej. Przygotował pierwszy framework mikrousług klasy enterprise (napisany w całości w C# i .NET), wykorzystywany przez jeden z głównych funduszy hedgingowych. Napisał też framework sztucznej inteligencji, w którym zintegrowane zostały neurony lustrzane i kanoniczne.

ISBN 978-83-283-5233-9

Cena: 49,00 zł

Packt