O'REILLY®

Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow

POJĘCIA, TECHNIKI I NARZĘDZIA SŁUŻĄCE DO TWORZENIA INTELIGENTNYCH SYSTEMÓW

Helion

Aurélien Géron

Tytuł oryginału: Hands-On Machine Learning with Scikit-Learn and TensorFlow

Tłumaczenie: Krzysztof Sawka

ISBN: 978-83-283-4373-3

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Krajobraz uczenia maszynowego

Większość osób, słysząc o uczeniu maszynowym, wyobraża sobie robota: jedni myślą o posłusznym kamerdynerze, inni o zabójczym Terminatorze. Zjawisko to jednak nie stanowi futurystycznej fantazji, lecz jest już elementem współczesnego świata. W rzeczywistości istnieje już od dziesiątek lat w pewnych wyspecjalizowanych zastosowaniach, takich jak techniki **optycznego rozpoznawania znaków** (ang. *Optical Character Recognition* — OCR). Jednak techniki uczenia maszynowego trafiły pod strzechy, poprawiając komfort życia milionów osób, dopiero w latach dziewięćdziesiątych — w postaci **filtrów spamu**. Niekoniecznie przypominają one Skynet[1], ale pod względem technicznym są klasyfikowane jako aplikacje wykorzystujące uczenie maszynowe (istotnie, filtry te uczą się tak skutecznie, że bardzo rzadko musisz własnoręcznie oznaczać wiadomości jako spam). W następnych latach nastąpił wysyp różnych innych zastosowań uczenia maszynowego, stanowiących trzon wielu aplikacji i usług, z których korzystamy na co dzień, począwszy od polecanych produktów aż do wyszukiwania głosowego.

Gdzie się zaczyna i kończy uczenie maszynowe? Co to właściwie oznacza, że dana maszyna **uczy się** określonego zagadnienia? Czy jeśli pobiorę na dysk artykuł z Wikipedii, to mój komputer „nauczył się" jego treści? Czy stał się nagle mądrzejszy? W tym rozdziale wyjaśnię, co uznajemy za uczenie maszynowe oraz dlaczego ta dziedzina może Cię zainteresować.

Następnie, zanim wyruszymy na wyprawę w głąb kontynentu Uczenia Maszynowego, zerkniemy na mapę i przyjrzymy się jego najważniejszym regionom oraz punktom orientacyjnym: porównamy uczenie nadzorowane z nienadzorowanym, przyrostowe ze wsadowym, a także metody uczenia z przykładów z metodami uczenia z modelu. Następnie przeanalizujemy cykl tworzenia typowego projektu uczenia maszynowego, zastanowimy się nad głównymi wyzwaniami, jakim należy stawić czoło, a także przeanalizujemy sposoby oceny i strojenia szybkości modelu.

W niniejszym rozdziale wprowadzam wiele podstawowych pojęć (i terminów), które musi znać każdy szanujący się analityk danych. Informacje tu zawarte są bardzo ogólne (jest to jedyny rozdział pozbawiony listingów kodu) i raczej nieskomplikowane, zanim jednak przejdziesz do następnego rozdziału, upewnij się, że doskonale je rozumiesz. Zatem kawa w dłoń i do dzieła!

[1] Samoświadoma sztuczna inteligencja dążąca do zniszczenia ludzkości, istniejąca w serii filmów *Terminator* — *przyp. tłum.*

Jeśli znasz już podstawy uczenia maszynowego, możesz przejść od razu do rozdziału 2. Jeżeli jednak nie masz pewności, spróbuj najpierw odpowiedzieć na wszystkie pytania umieszczone na końcu rozdziału.

Czym jest uczenie maszynowe?

Uczeniem maszynowym nazywamy dziedzinę nauki (i sztukę) programowania komputerów w sposób umożliwiający im **uczenie się z danych**.

Poniżej przedstawiam nieco ogólniejszą definicję:

> [Uczenie maszynowe to] dziedzina nauki dająca komputerom możliwość uczenia się bez konieczności ich jawnego programowania.
>
> — Arthur Samuel, *1959*

A tu bardziej techniczna:

> Mówimy, że program komputerowy uczy się na podstawie doświadczenia E w odniesieniu do jakiegoś zadania T i pewnej miary wydajności P, jeśli jego wydajność (mierzona przez P) wobec zadania T wzrasta wraz z nabywaniem doświadczenia E.
>
> — Tom Mitchell, *1997*

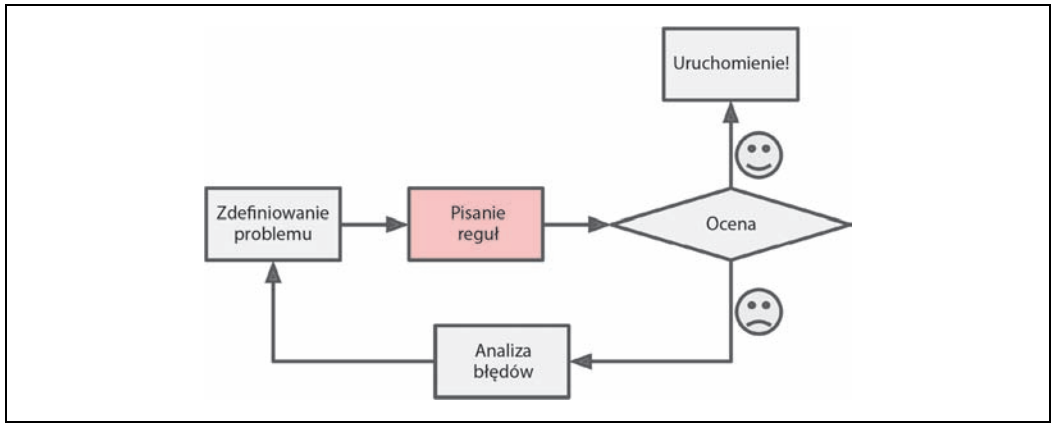
Na przykład filtr spamu jest programem wykorzystującym algorytmy uczenia maszynowego do rozpoznawania spamu na podstawie przykładowych wiadomości e-mail (tj. spamu oznaczonego przez użytkowników oraz zwykłych wiadomości niebędących spamem, znanych jako „ham”[2]). Przykładowe dane używane do trenowania systemu noszą nazwę **zbioru/zestawu uczącego** (ang. *training set*). Każdy taki element uczący jest nazywany **przykładem uczącym** (**próbką uczącą**). Zgodnie z powyższą definicją naszym zadaniem T jest tu oznaczanie spamu, doświadczeniem E — **dane uczące**; pozostaje nam wyznaczyć miarę wydajności P. Może być nią, na przykład, stosunek prawidłowo oznaczonych wiadomości do przykładów nieprawidłowo sklasyfikowanych. Ta konkretna miara wydajności jest nazywana **dokładnością** (ang. *accuracy*) i znajduje często zastosowanie w zadaniach klasyfikujących.

Nawet jeśli pobierzesz całą Wikipedię na dysk, Twój komputer będzie przechowywał mnóstwo informacji, ale nie wykorzysta ich w żaden sposób. Dlatego nie uznajemy tego za uczenie maszynowe.

Dlaczego warto korzystać z uczenia maszynowego?

Załóżmy, że chcemy napisać filtr spamu za pomocą tradycyjnych technik programistycznych (rysunek 1.1):

[2] Angielska nazwa niechcianych wiadomości pocztowych — „spam” (mielonka) — została zapożyczona ze słynnego skeczu grupy Monty Python pt. „Spam”. Nazwa standardowych wiadomości („ham”) oznacza w dosłownym tłumaczeniu szynkę — *przyp. tłum.*



Rysunek 1.1. Tradycyjne podejście

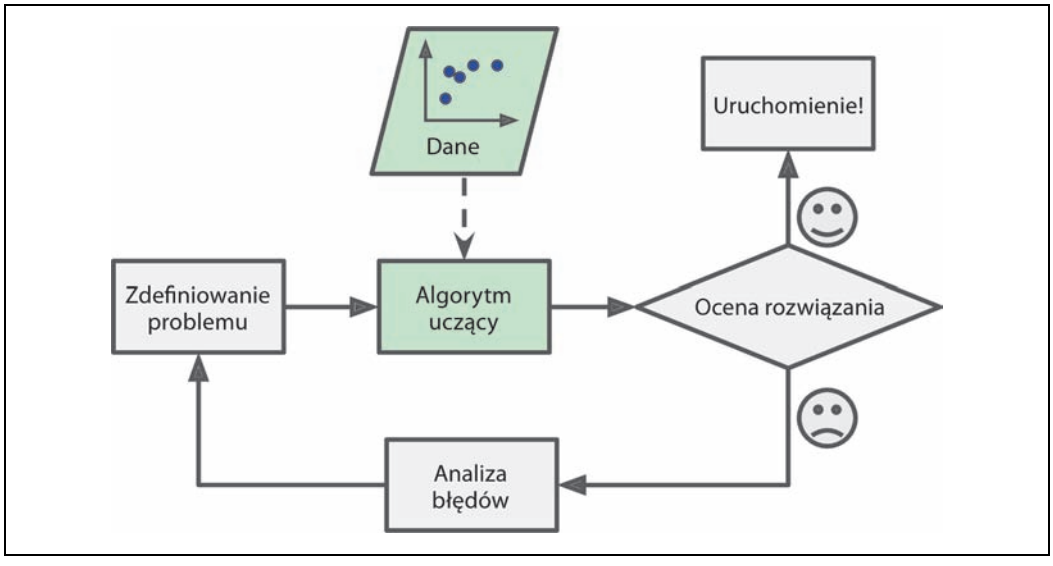
1. Najpierw musielibyśmy zastanowić się, jak wygląda klasyczny spam. Pewnie zauważysz, że niektóre wyrazy lub zwroty (np. „karta kredytowa", „darmowe", „zdumiewające", „bez limitów") bardzo często występują w temacie wiadomości. Być może odkryjesz również kilka innych szablonów w nazwie nadawcy, ciele wiadomości itd.
2. Następnym etapem byłoby napisanie algorytmu wykrywającego każdy z zaobserwowanych szablonów; program oznaczałby wiadomości jako spam, jeśli wykrywałby w nich kilka spośród zdefiniowanych wzorców.
3. Teraz należałoby przetestować ten program i usprawniać go poprzez ciągłe powtarzanie kroków 1. i 2.

Problem ten nie jest trywialny i omawiany program będzie się rozrastał o coraz większą liczbę skomplikowanych reguł — jego własnoręczne utrzymanie stałoby się w końcu bardzo uciążliwe.

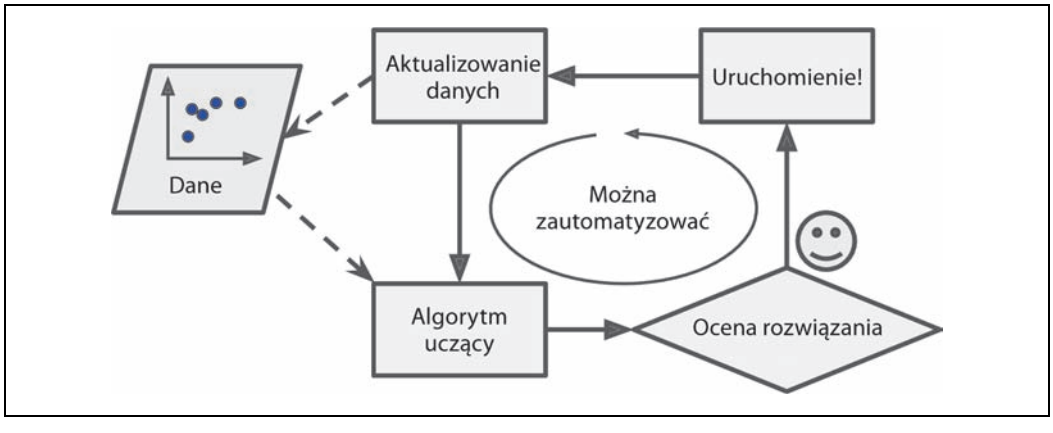
Z drugiej strony filtr spamu bazujący na algorytmach uczenia maszynowego automatycznie rozpoznaje, które słowa i wyrażenia stanowią dobre czynniki prognostyczne, poprzez wykrywanie podejrzanie często powtarzających się wzorców wyrazów w przykładowym spamie (w porównaniu do przykładów standardowych wiadomości); patrz rysunek 1.2. Kod takiej aplikacji jest znacznie krótszy, łatwiejszy do utrzymywania i prawdopodobnie dokładniejszy.

Poza tym jeśli spamerzy zauważą, że blokowane są wszystkie wiadomości zawierające zwrot „bez limitu", mogą go zmodyfikować na „bez ograniczeń". Filtr spamu stworzony za pomocą tradycyjnych technik programistycznych musiałby wtedy zostać zaktualizowany tak, aby oznaczał wiadomości zawierające nowy zwrot. Gdyby spamerzy próbowali cały czas w jakiś sposób oszukiwać Twój filtr spamu, musiałabyś/musiałbyś bez przerwy dopisywać nowe reguły.

Natomiast filtr spamu stworzony za pomocą technik uczenia maszynowego automatycznie „zauważy" nagły wzrost częstotliwości pojawiania się zwrotu „bez ograniczeń" w spamie oznaczanym przez użytkowników i zacznie go blokować bez Twojego udziału (rysunek 1.3).



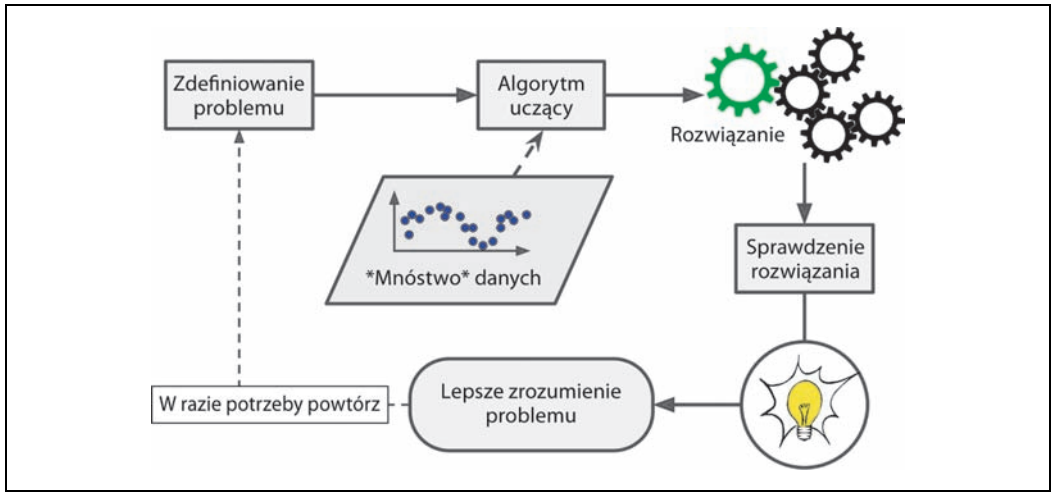
Rysunek 1.2. Podejście wykorzystujące uczenie maszynowe



Rysunek 1.3. Automatyczne dostosowywanie się do zmian

Kolejnym obszarem, w którym wyróżnia się mechanizm uczenia maszynowego, są problemy, które są zbyt złożone dla tradycyjnych metod lub dla których nie istnieją gotowe algorytmy. Przyjrzyjmy się, na przykład, rozpoznawaniu mowy: załóżmy, że chcesz stworzyć prostą aplikację zdolną do rozróżniania wyrazów „raz" i „trzy". Zwróć uwagę, że słowo „trzy" zaczyna się od dźwięku o wysokim tonie („T'"), dlatego moglibyśmy stworzyć algorytm wykrywający wysokość dźwięku i użyć go do rozróżniania obydwu wyrazów. Oczywiście, technika ta zupełnie nie nadaje się do rozpoznawania tysięcy słów wypowiadanych w hałaśliwych miejscach przez miliony różnych ludzi, porozumiewających się w dziesiątkach języków. Najlepszym rozwiązaniem (przynajmniej przy obecnym stanie naszej wiedzy) jest napisanie samouczącego się algorytmu na podstawie przykładowych nagrań danego słowa.

Uczenie maszynowe pomaga również w trenowaniu ludzi (rysunek 1.4): możemy przeglądać algorytmy, aby sprawdzić, czego się nauczyły (chociaż nie zawsze jest to łatwo sprawdzić). Przykładowo, po wytrenowaniu filtru spamu do rozpoznawania odpowiedniej ilości spamu, można z łatwością sprawdzić listę wyuczonych słów i kombinacji wyrazów uznawanych za najlepsze czynniki prognostyczne. Czasem w ten sposób możemy wykryć niespodziewane korelacje lub nowe trendy, dzięki czemu jesteśmy w stanie lepiej pojąć dany problem.



Rysunek 1.4. Uczenie maszynowe pomaga w trenowaniu ludzi

Wykorzystanie technik uczenia maszynowego do analizowania olbrzymich ilości danych może pomóc w wykrywaniu nieoczywistych wzorców. Proces ten nazywamy **wydobywaniem danych** (ang. *data mining*).

Podsumowując, uczenie maszynowe nadaje się znakomicie do:

- problemów, których rozwiązanie wymaga mnóstwo ręcznego dostrajania algorytmu lub korzystania z długich list reguł; często jeden algorytm uczenia maszynowego upraszcza aplikację i poprawia jej szybkość,
- złożonych problemów, których nie można rozwiązać tradycyjnymi metodami; najlepsze algorytmy uczenia maszynowego są w stanie znaleźć rozwiązanie,
- zmiennych środowisk; mechanizm uczenia maszynowego potrafi dostosować się do nowych danych,
- pomagania człowiekowi w analizowaniu skomplikowanych zagadnień i olbrzymich ilości danych.

Rodzaje systemów uczenia maszynowego

Istnieje tak wiele typów uczenia maszynowego, że warto podzielić je na ogólne kategorie na podstawie takich czynników, jak:

- nadzór człowieka w procesie trenowania (uczenie nadzorowane, nienadzorowane, półnadzorowane i uczenie przez wzmacnianie),

- możliwość uczenia się w czasie rzeczywistym (uczenie przyrostowe i wsadowe),
- sposób pracy: proste porównywanie nowych punktów danych ze znanymi punktami lub, podobnie do naukowców, wykrywanie wzorców w danych uczących i tworzenie modelu predykcyjnego (uczenie z przykładów i uczenie z modelu).

Kryteria te nie wykluczają się wzajemnie; możesz łączyć je w dowolny sposób. Na przykład nowoczesny filtr spamu może uczyć się na bieżąco przy użyciu modelu głębokiej sieci neuronowej, korzystając z przykładowych wiadomości e-mail; w ten sposób zaprzęgamy do pracy przyrostowy, bazujący na modelu i nadzorowany system uczenia maszynowego.

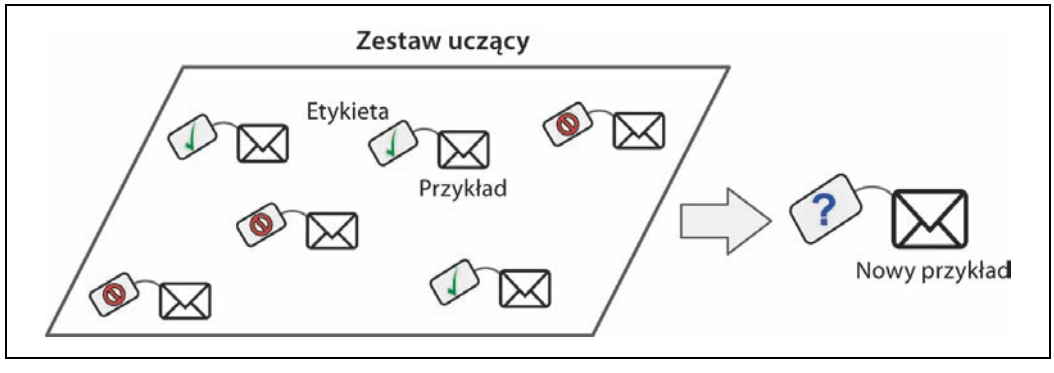
Przyjrzyjmy się nieco uważniej każdemu z wymienionych kryteriów.

Uczenie nadzorowane/nienadzorowane

Systemy uczenia maszynowego możemy podzielić na podstawie stopnia i rodzaju nadzorowania procesu uczenia. Pod tym względem uczenie maszynowe dzielimy na cztery zasadnicze rodzaje: uczenie nadzorowane, nienadzorowane, półnadzorowane i uczenie przez wzmacnianie).

Uczenie nadzorowane

W **uczeniu nadzorowanym** (ang. *supervised learning*) dane uczące przekazywane algorytmowi zawierają dołączone rozwiązania problemu, tzw. **etykiety** (ang. *labels*); patrz rysunek 1.5.



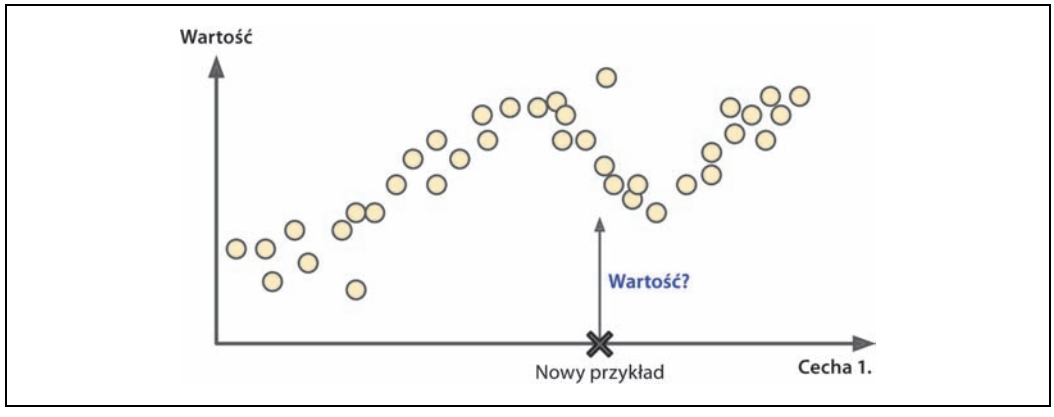
Rysunek 1.5. Zbiór danych uczących zaopatrzonych w etykiety, stosowany w uczeniu nadzorowanym (np. do klasyfikowania spamu)

Klasycznym zadaniem uczenia nadzorowanego jest **klasyfikacja** (ang. *classification*). Filtr spamu stanowi tu dobry przykład: jest on trenowany za pomocą dużej liczby przykładowych wiadomości e-mail należących do danej **klasy** (spamu lub hamu), dzięki którym musi być w stanie klasyfikować nowe wiadomości.

Innym typowym zadaniem uczenia nadzorowanego jest przewidywanie **docelowej** wartości numerycznej, takiej jak cena samochodu, przy użyciu określonego zbioru cech (przebieg, wiek, marka itd.), zwanych **czynnikami prognostycznymi/predykcyjnymi** (ang. *predictors*). Ten typ zadania nosi nazwę

regresji (rysunek 1.6)[3]. Aby wyuczyć dany system, należy mu podać wiele przykładów samochodów, w tym zarówno etykiety (np. ceny), jak i czynniki prognostyczne.

W terminologii uczenia maszynowego **atrybutem** (ang. *attribute*) nazywamy typ danych (np. „Przebieg"), natomiast **cecha** (ang. *feature*) w zależności od kontekstu ma kilka różnych znaczeń, zazwyczaj jednak mówiąc o cesze, mamy na myśli atrybut wraz z jego wartością (np. „Przebieg = 15 000"). Wiele osób korzysta jednak naprzemiennie z wyrazów **atrybut** i **cecha**.



Rysunek 1.6. Regresja

Zwróć uwagę, że niektóre algorytmy regresyjne mogą być również używane do klasyfikowania danych i odwrotnie. Na przykład algorytm **regresji logistycznej** jest powszechnie stosowany w zadaniach klasyfikacyjnych, ponieważ może podawać wynikową wartość odpowiadającą prawdopodobieństwu przynależności do danej klasy (np. 20% szans na to, że dana wiadomość jest spamem).

Poniżej wymieniam niektóre z najważniejszych algorytmów nadzorowanego uczenia maszynowego (opisywanych w tej książce):

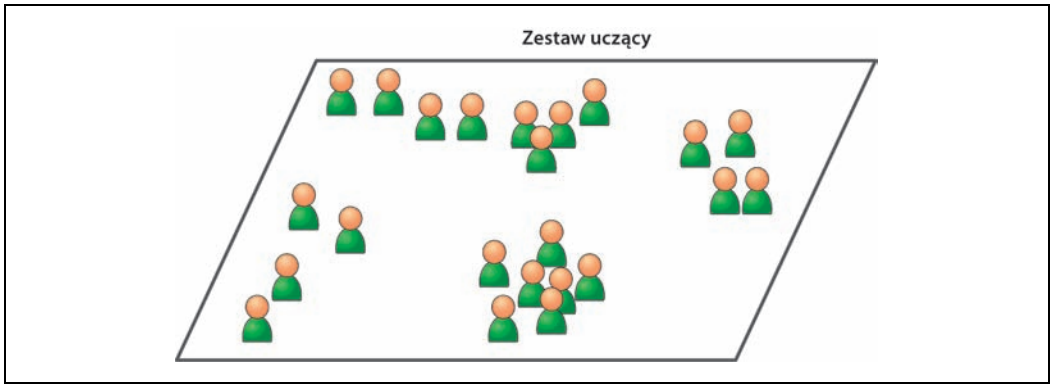
- metoda k-najbliższych sąsiadów,
- regresja liniowa,
- regresja logistyczna,
- maszyny wektorów nośnych,
- drzewa decyzyjne i losowe lasy,
- sieci neuronowe[4].

[3] Ciekawostka: ta dziwnie brzmiąca nazwa wywodzi się z teorii statystyki i została zaproponowana przez Francisa Galtona, gdy analizował regułę, zgodnie z którą dzieci wysokich rodziców często bywają od nich niższe. Zjawisko redukcji wzrostu w kolejnym pokoleniu badacz ten nazwał **regresją do średniej**. Nazwą tą zostały następnie określone użyte przez niego metody służące do analizowania korelacji pomiędzy zmiennymi.

[4] Architektury niektórych sieci neuronowych mogą być nienadzorowane (np. autokodery czy ograniczone maszyny Boltzmanna). Istnieją także architektury półnadzorowane, takie jak głębokie sieci przekonań i nienadzorowane uczenie wstępne.

Uczenie nienadzorowane

Jak można się domyślać, w **uczeniu nienadzorowanym** (ang. *unsupervised learning*) dane uczące są nieoznakowane (rysunek 1.7). System próbuje uczyć się bez nauczyciela.

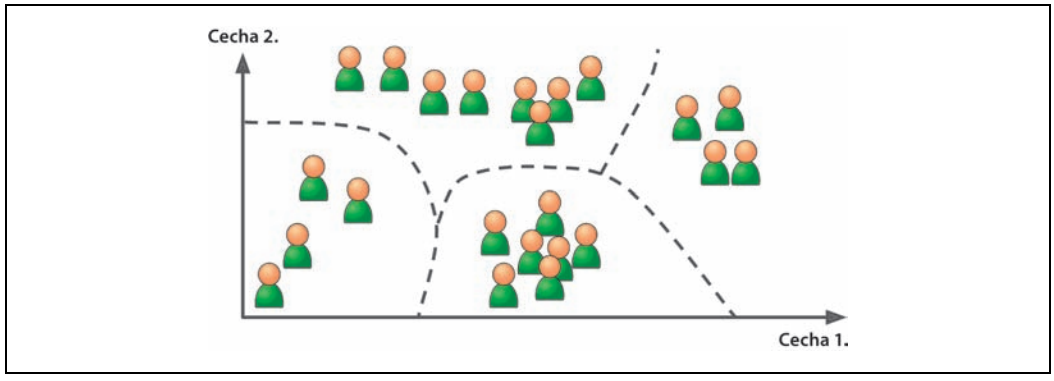


Rysunek 1.7. Zbiór nieoznakowanych danych uczących używanych w uczeniu nienadzorowanym

Poniżej wymieniamy kilka najważniejszych algorytmów uczenia nienadzorowanego (w rozdziale 8. zajmiemy się zagadnieniem redukcji wymiarowości):

- analiza skupień:
  - metoda k-średnich,
  - hierarchiczna analiza skupień (HCA),
  - algorytm oczekiwanie-maksymalizacja,
- wizualizacja i redukcja wymiarowości:
  - analiza głównych składowych (PCA),
  - jądrowa analiza głównych składowych,
  - lokalnie liniowe zanurzanie (ang. *locally linear embedding* — LLE),
  - stochastyczne zanurzanie sąsiadów przy użyciu rozkładu t (ang. *t-Distributed Stochastic Neighbor Embedding* — t-SNE),
- uczenie przy użyciu reguł asocjacyjnych:
  - algorytm Apriori,
  - algorytm Eclat.

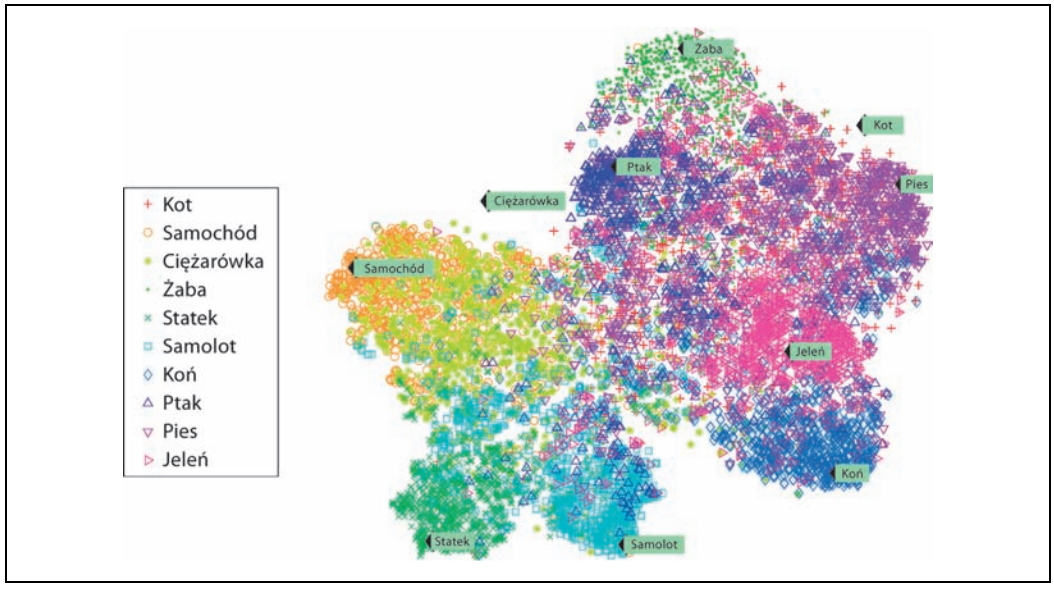
Załóżmy, na przykład, że masz do dyspozycji mnóstwo danych dotyczących osób odwiedzających Twój blog. Możesz chcieć skorzystać z **analizy skupień** (ang. *clustering*), aby spróbować określić grupy podobnych użytkowników (rysunek 1.8). W dowolnym momencie możesz sprawdzić, do której grupy zalicza się dana odwiedzająca osoba: powiązania pomiędzy poszczególnymi użytkownikami są określane bez Twojej pomocy. Algorytm ten może, przykładowo, zauważyć, że 40% odwiedzających osób to męscy miłośnicy komiksów przeglądający Twoją stronę głównie wieczorami, podczas gdy grupa 20% innych użytkowników to młodociani czytelnicy fantastyki naukowej, którzy zaglądają na Twój blog wyłącznie w weekendy itd. Jeśli użyjesz algorytmu **hierarchicznej analizy skupień**



Rysunek 1.8. Analiza skupień

(ang. *hierarchical clustering*), może on dodatkowo rozdzielić grupy na mniejsze podjednostki. W ten sposób możesz łatwiej określić, jakie wpisy umieszczać dla poszczególnych grup.

Dobrym przykładem algorytmów uczenia nienadzorowanego są również algorytmy **wizualizujące**: wprowadzasz mnóstwo złożonych, nieoznakowanych danych, które są następnie wyświetlane w postaci punktów na dwu- lub trójwymiarowym wykresie (rysunek 1.9). Algorytmy te starają się w maksymalnym stopniu zachować pierwotną strukturę danych (np. próbując rozdzielać poszczególne skupienia w przestrzeni wejściowej, redukując zjawiska nakładania się poszczególnych klastrów na siebie), dzięki czemu możesz łatwiej przeanalizować te dane i być może odkryć jakieś nieprzewidziane wzorce.



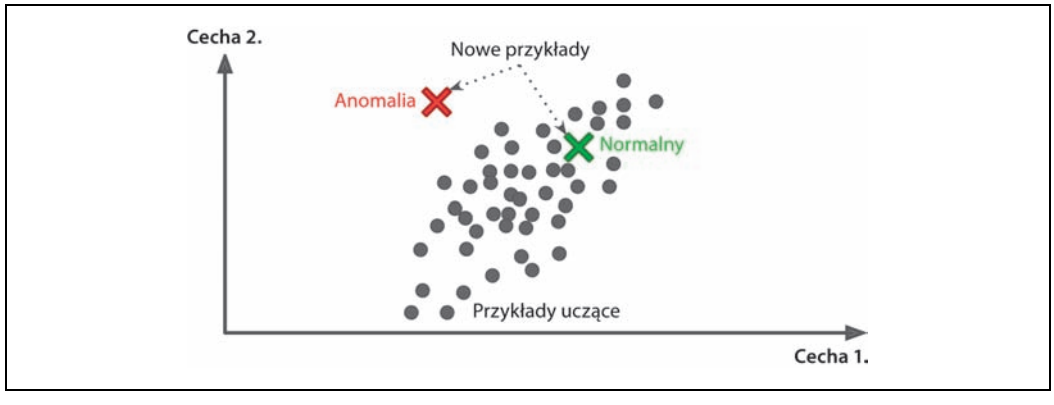
Rysunek 1.9. Przykład wizualizacji t-SNE wskazującej semantyczny podział skupień[5]

[5] Zwróć uwagę, jak ładnie są rozdzielone zwierzęta od pojazdów, konie znajdują się blisko jeleni, ale daleko od ptaków itd. Rysunek wykorzystany za pozwoleniem Sochera, Ganjoo, Manninga i Ng (2013), *T-SNE visualization of the semantic word space*.

Podobnym zadaniem jest **redukcja wymiarowości** (ang. *dimensionality reduction*), której celem jest uproszczenie danych bez utraty nadmiernej ilości informacji. Możemy to osiągnąć między innymi poprzez scalenie kilku skorelowanych cech w jedną. Przykładowo, możemy skorelować przebieg samochodu z jego wiekiem, dzięki czemu algorytm będzie w stanie połączyć je w jedną cechę reprezentującą zużycie pojazdu. Proces ten jest nazywany **wydobywaniem cech** (ang. *feature extraction*).

Zawsze warto spróbować zredukować wymiarowość danych uczących przed dostarczeniem ich do algorytmu uczenia maszynowego (np. algorytmu uczenia nadzorowanego). Algorytm będzie działał wtedy szybciej, dane będą zabierały mniej miejsca na dysku i w pamięci operacyjnej, a w niektórych przypadkach wzrośnie także wydajność uczenia maszynowego.

Kolejnym ważnym nienadzorowanym zadaniem jest **wykrywanie anomalii** (ang. *anomaly detection*), takich jak np. nietypowe transakcje wykorzystujące kartę kredytową, co pozwala zapobiegać nielegalnym operacjom. Służy ono także do wyłapywania usterek produkcyjnych czy też automatycznego usuwania elementów odstających w danych uczących przed ich przekazaniem algorytmowi uczenia maszynowego. Taki system jest trenowany za pomocą standardowych przykładów, a w przypadku zaprezentowania nowego przykładu jest w stanie stwierdzić, czy na przykład jest on standardowy, czy też stanowi anomalię (rysunek 1.10).

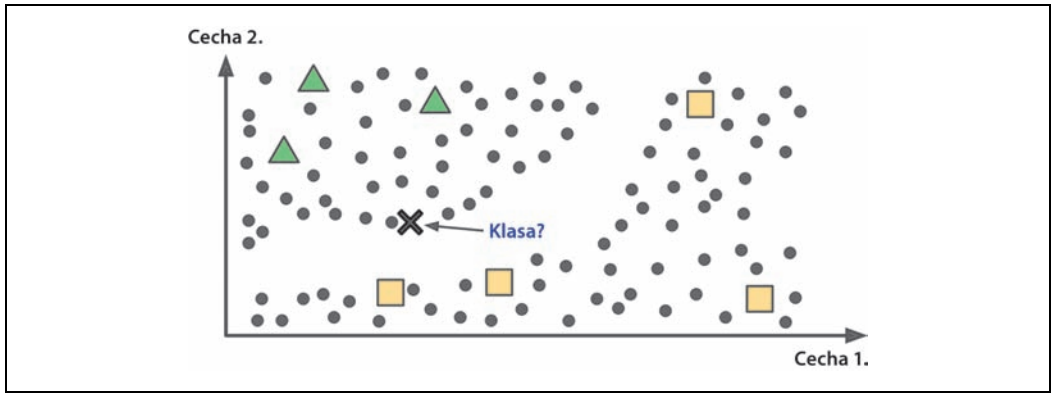


Rysunek 1.10. Wykrywanie anomalii

Jeszcze jednym powszechnie stosowanym zadaniem uczenia nienadzorowanego jest **uczenie przy użyciu reguł asocjacyjnych** (ang. *association rule learning*), którego celem jest analiza ogromnej ilości danych i wykrycie interesujących zależności pomiędzy atrybutami. Załóżmy, że jesteś właścicielką/właścicielem supermarketu. Przetworzenie danych dotyczących obrotów za pomocą algorytmu reguł asocjacyjnych pozwoliłoby nam odkryć, że osoby kupujące keczup i czipsy zazwyczaj zaopatrują się również w steki. Dzięki tej wiedzy możesz chcieć, na przykład, umieścić te produkty niedaleko siebie.

Uczenie półnadzorowane

Niektóre algorytmy są w stanie przetwarzać częściowo oznakowane dane uczące, najczęściej składające się z większości nieoznakowanych i tylko odrobiny oznakowanych przykładów. Taką sytuację nazywamy **uczeniem półnadzorowanym** (ang. *semisupervised learning*; rysunek 1.11).



Rysunek 1.11. Uczenie półnadzorowane

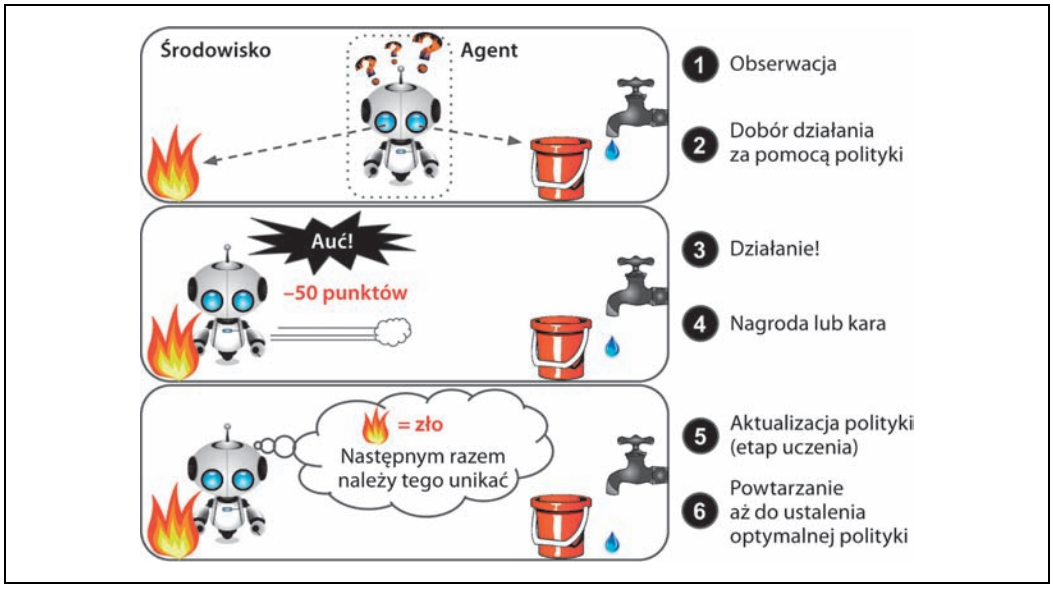
Dobrym przykładem zastosowania tego typu metod są niektóre usługi przechowywania obrazów, np. Zdjęcia Google. Po przesłaniu do takiej usługi wszystkich zdjęć rodzinnych rozpoznaje ona automatycznie, że na zdjęciach 1., 5. i 11. pojawia się osoba A, natomiast do fotografii 2., 5. i 7. pozuje osoba B. Na tym etapie mamy do czynienia z uczeniem nienadzorowanym (analizą skupień). Teraz wystarczy, że podasz imiona poszczególnych osób. Każda osoba potrzebuje tylko jednej etykiety[6], co wystarczy do oznakowania wszystkich osób na każdym zdjęciu i znacznie ułatwia wyszukiwanie zdjęć.

Większość algorytmów półnadzorowanych stanowi kombinację algorytmów uczenia nadzorowanego i nienadzorowanego. Na przykład **głębokie sieci przekonań** (ang. *deep belief networks* — DBN) bazują na ułożonych warstwowo elementach nienadzorowanych, zwanych **ograniczonymi maszynami Boltzmanna** (ang. *restricted Boltzmann machines* — RBM). Maszyny te są uczone sekwencyjnie w nienadzorowany sposób, a następnie cały system zostaje dostrojony przy użyciu technik uczenia nadzorowanego.

Uczenie przez wzmacnianie

Uczenie przez wzmacnianie (ang. *reinforcement learning*) to zupełnie inna bajka. System uczący, zwany w tym kontekście **agentem**, może obserwować środowisko, dobierać i wykonywać czynności, a także odbierać **nagrody** (lub **kary** będące ujemnymi nagrodami; patrz rysunek 1.12). Musi następnie samodzielnie nauczyć się najlepszej strategii (nazywanej **polityką**; ang. *policy*), aby uzyskiwać jak największą nagrodę. Polityka definiuje rodzaj działania, jakie agent powinien wybrać w danej sytuacji.

[6] Tak wygląda wyidealizowana sytuacja. W praktyce usługa ta często tworzy kilka skupień dla jednej osoby, a czasami myli dwie podobne do siebie osoby, dlatego należy każdej osobie przydzielić kilka etykiet oraz własnoręcznie oczyścić niektóre skupienia.



Rysunek 1.12. Uczenie przez wzmacnianie

Na przykład metody uczenia przez wzmacnianie są używane w wielu modelach robotów do nauki chodzenia. Również program AlphaGo firmy DeepMind był trenowany przez wzmacnianie: zrobiło się o nim głośno w marcu 2016 roku, gdy pokonał dotychczasowego mistrza świata w grę *Go* — Lee Sedola. Aplikacja ta zdefiniowała politykę gwarantującą zwycięstwo, analizując miliony rozgrywek i rozgrywając mnóstwo partii przeciwko samej sobie. Warto zauważyć, że algorytmy uczące zostały wyłączone podczas partii z ludzkim przeciwnikiem; program AlphaGo korzystał jedynie z wyuczonej polityki.

Uczenie wsadowe/przyrostowe

Innym kryterium stosowanym do klasyfikowania systemów uczenia maszynowego jest możliwość trenowania przyrostowego przy użyciu strumienia nadsyłanych danych.

Uczenie wsadowe

W mechanizmie **uczenia wsadowego** (ang. *batch learning*) system nie jest w stanie trenować przyrostowo: do jego nauki muszą wystarczyć wszystkie dostępne dane. Wymaga to zazwyczaj poświęcenia dużej ilości czasu i zasobów, dlatego najczęściej proces ten jest przeprowadzany w trybie offline. System najpierw jest uczony, a następnie zostaje wdrożony do cyklu produkcyjnego i już więcej nie jest trenowany; korzysta jedynie z dotychczas zdobytych informacji. Zjawisko to bywa nazywane **uczeniem offline** (ang. *offline learning*).

Jeśli chcesz, aby system uczenia wsadowego brał pod uwagę nowe dane (np. nowe rodzaje spamu), musisz od podstaw wytrenować nową wersję systemu przy użyciu wszystkich dostępnych danych (nowych i starych), następnie wyłączyć stary system i zastąpić go nowym.

Na szczęście cały proces trenowania, oceniania i uruchamiania systemu uczenia maszynowego można dość łatwo zautomatyzować (co widać na rysunku 1.3), dlatego nawet układ uczenia wsadowego jest w stanie dostosowywać się do zmian. Wystarczy zaktualizować dane i z odpowiednią częstotliwością trenować system od podstaw.

Rozwiązanie to jest proste i często skuteczne, ale trenowanie za pomocą pełnego zbioru danych uczących nieraz trwa wiele godzin, dlatego zazwyczaj trenujemy nowy system raz na dobę, a czasami nawet raz w tygodniu. Jeśli Twój system musi dostosowywać się do szybko zmieniających się danych (np. podczas prognozowania cen akcji giełdowych), będziesz potrzebować dynamiczniejszego mechanizmu.

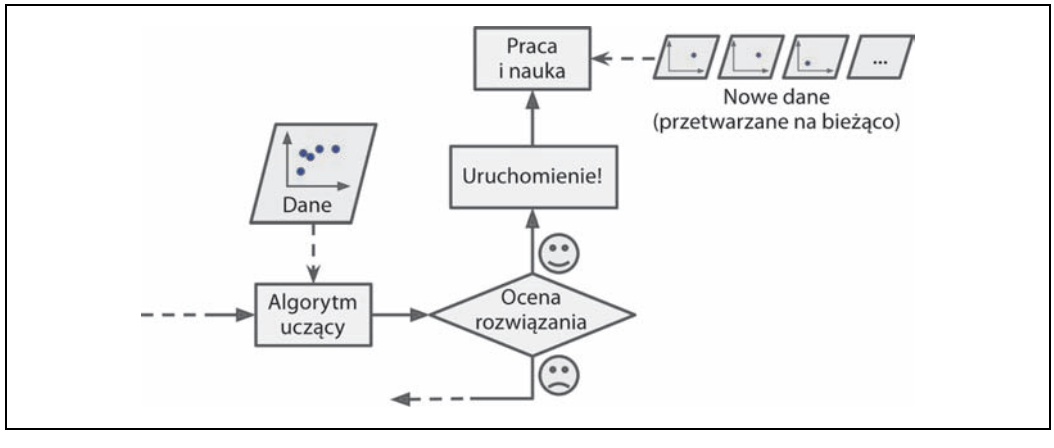
Ponadto uczenie się na pełnym zbiorze danych pochłania mnóstwo zasobów obliczeniowych (mocy procesora, miejsca i szybkości dysku, szybkości sieci itd.). Jeśli masz dużo danych i trenujesz system codziennie od podstaw, będzie Cię to kosztowało wiele pieniędzy. W przypadku danych o naprawdę pokaźnych rozmiarach korzystanie z algorytmu uczenia wsadowego może być wręcz niemożliwe.

Zwróćmy na koniec uwagę, że jeśli system ma być zdolny do autonomicznej nauki przy użyciu ograniczonych zasobów (np. aplikacja na smartfona albo marsjański łazik), to magazynowanie olbrzymich ilości danych i przeznaczanie zasobów na codzienne wielogodzinne trenowanie systemu mogą zwyczajnie uniemożliwiać normalną pracę urządzenia.

Na szczęście istnieje lepsze rozwiązanie dla wymienionych powyżej przykładów: algorytmy zdolne do uczenia przyrostowego.

Uczenie przyrostowe

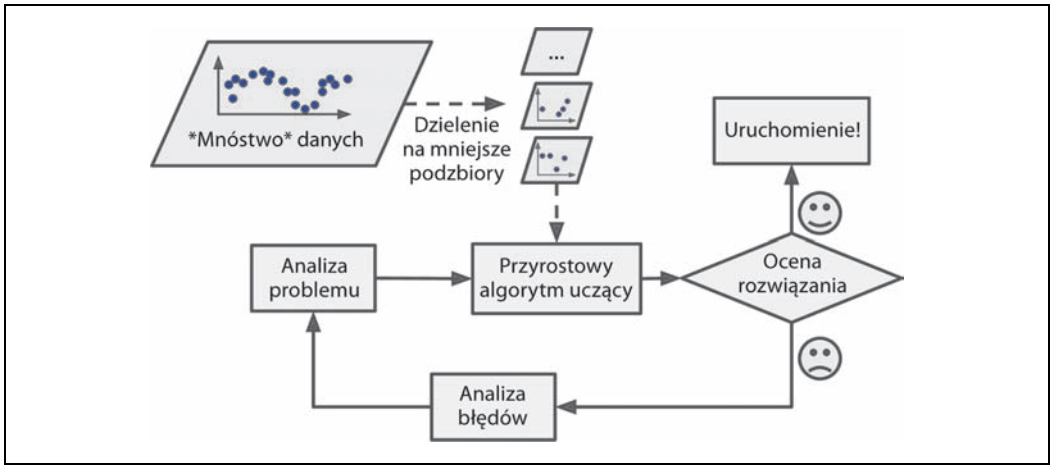
W procesie **uczenia przyrostowego** (ang. *online learning*) system jest trenowany na bieżąco poprzez sekwencyjne dostarczanie danych; mogą być one pojedyncze lub przyjmować postać tzw. **minipakietów/minigrup** (ang. *mini-batches*), czyli niewielkich zbiorów. Każdy krok uczący jest szybki i niezbyt kosztowny, dlatego system jest w stanie uczyć się na bieżąco przy użyciu nowych danych, gdy tylko się pojawiają (patrz rysunek 1.13).



Rysunek 1.13. Uczenie przyrostowe

Uczenie przyrostowe sprawdza się znakomicie w systemach odbierających ciągły strumień nowych danych (np. ceny akcji) oraz wymagających szybkiego lub autonomicznego dostosowywania się do nowych warunków. Metody te przydają się również przy ograniczonych zasobach obliczeniowych: dane użyte do nauki nie są już potrzebne i można się ich pozbyć (chyba że chcesz zachować możliwość cofnięcia się do wcześniejszego stanu i „odtworzyć ponownie" dane). W ten sposób możesz mocno zaoszczędzić na przestrzeni dyskowej.

Algorytmy uczenia przyrostowego są w stanie również trenować systemy za pomocą dużych zbiorów danych niemieszczących się w pamięci urządzenia (jest to tzw. **uczenie pozakorowe** — ang. *out-of-core learning*). Następuje wczytanie części danych i trenowanie systemu za ich pomocą, po czym cały proces zostanie powtórzony dla całego zbioru danych uczących (rysunek 1.14).



Rysunek 1.14. Przetwarzanie dużych ilości danych za pomocą algorytmu uczenia przyrostowego

Jednym z najważniejszych parametrów systemów uczenia przyrostowego jest szybkość, z jaką dostosowują się do zmieniających się danych: jest to tak zwany **współczynnik uczenia** (ang. *learning rate*). Wysoka wartość tego współczynnika oznacza, że system będzie szybko dostosowywał się do nowych danych, ale jednocześnie dość szybko zapominał o starych danych (raczej nie chcesz, aby filtr spamu oznaczał jedynie najnowsze rodzaje spamu). Z drugiej strony, system o niskim współczynniku uczenia będzie cechował się większą inercją; oznacza to, że będzie uczył się wolniej, ale jednocześnie będzie bardziej odporny na zaszumienie nowych danych oraz na sekwencje niereprezentatywnych punktów danych.

Z kolei dużym problemem uczenia przyrostowego jest stopniowy spadek wydajności systemu w przypadku dostarczenia mu nieprawidłowych danych. Klienci korzystający z danego systemu na pewno to zauważą. Przykładowo, nieprawidłowe dane mogą pochodzić z uszkodzonego czujnika w robocie lub od osoby zasypującej silnik przeglądarki danymi w celu podbicia swojego rankingu w wynikach wyszukiwania. Aby zmniejszyć to ryzyko, musisz uważnie obserwować swój system i świadomie wyłączyć proces uczenia (a także ewentualnie przywrócić stan do wcześniejszego stanu) po wykryciu spadku szybkości algorytmu. Możesz także śledzić przychodzące dane i reagować na wszelkie nieprawidłowości (np. za pomocą algorytmu wykrywania anomalii).

Uczenie z przykładów/z modelu

Możemy również podzielić systemy uczenia maszynowego pod względem **uogólniania**. Większość zadań uczenia maszynowego polega na sporządzaniu prognoz. Oznacza to, że na podstawie określonej liczby próbek uczących system musi być w stanie generalizować wyniki na niewidziane wcześniej przykłady. Uzyskanie dobrej szybkości algorytmu wobec danych uczących jest pożądane, ale niewystarczające; prawdziwy cel stanowi dobra wydajność wobec nowych przykładów.

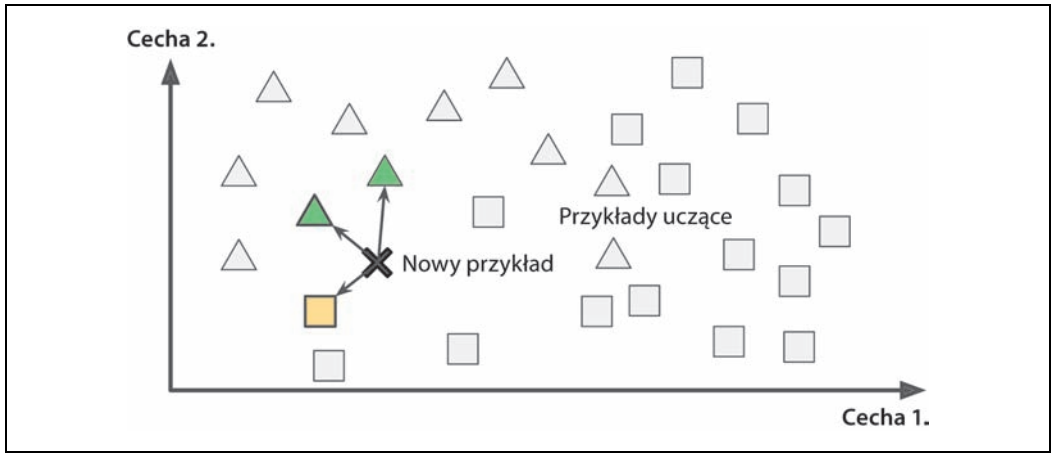
Znamy dwa główne mechanizmy uogólniania: uczenie z przykładów i uczenie z modelu.

Uczenie z przykładów

Prawdopodobnie najprostszą formą nauki jest „wykuwanie na blachę". Gdybyśmy stworzyli w ten sposób filtr spamu, oznaczałby po prostu wszystkie wiadomości identyczne z oznakowanymi przez użytkowników — rozwiązanie nie najgorsze, ale zarazem nie najlepsze.

Zamiast oznaczać wiadomości identyczne z rozpoznanym wcześniej spamem, możemy zaprogramować filtr w taki sposób, aby znakował również wiadomości bardzo podobne do powszechnie rozpoznawanych wzorców. Potrzebna staje się jakaś **miara podobieństwa** dwóch wiadomości e-mail. Może ją stanowić (w wielkim uproszczeniu) porównanie liczby takich samych słów występujących w obydwu wiadomościach. Filtr mógłby oznaczać wiadomość jako spam, jeśli będzie miała wiele słów wspólnych ze znanym spamem.

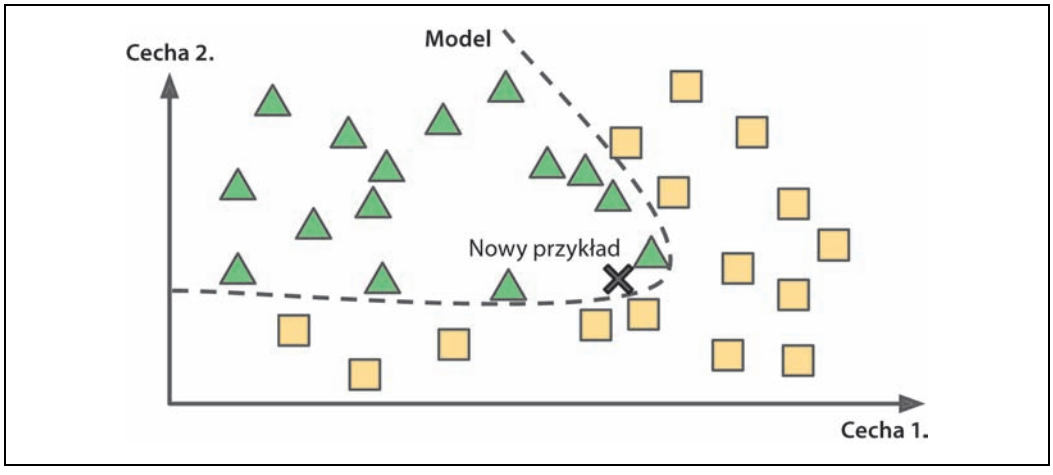
Mamy tu do czynienia z **uczeniem z przykładów** (ang. *instance-based learning*): system uczy się przykładów „na pamięć", a następnie porównuje (generalizuje) je z nowymi przypadkami za pomocą miary podobieństwa (rysunek 1.15).



Rysunek 1.15. Uczenie z przykładów

Uczenie z modelu

Innym sposobem uogólniania wyników uzyskiwanych z danych uczących jest stworzenie modelu z tych przykładów i użycie go do **przewidywania** (**prognozowania**; ang. *prediction*). Jest to **uczenie z modelu** (ang. *model-based learning*; rysunek 1.16).



Rysunek 1.16. Uczenie z modelu

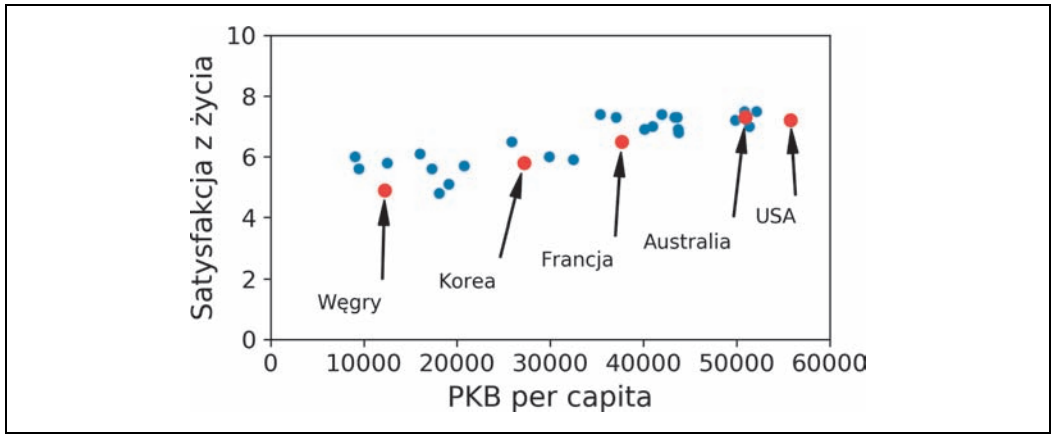
Załóżmy, na przykład, że chcemy sprawdzić, czy pieniądze dają szczęście; w tym celu moglibyśmy pobrać dane *Better Life Index* ze strony Organizacji Współpracy Gospodarczej i Rozwoju (OECD, *http://stats.oecd.org/index.aspx?DataSetCode=BLI*), a także statystyki PKB per capita z witryny Międzynarodowego Funduszu Walutowego (IMF, *http://goo.gl/j1MSKe*). Teraz wystarczyłoby połączyć obydwie tabele i posortować dane dotyczące PKB. Tabela 1.1 prezentuje fragment uzyskanych informacji.

Tabela 1.1. Czy pieniądze przynoszą szczęście?

Państwo	PKB per capita (w dolarach amerykańskich)	Satysfakcja z życia
Węgry	12 240	4,9
Korea	27 195	5,8
Francja	37 675	6,5
Australia	50 962	7,3
Stany Zjednoczone	55 805	7,2

Wygenerujmy teraz wykres danych dla kilku przykładowych krajów (rysunek 1.17).

Możemy dostrzec tu pewien trend! Pomimo tego, że dane są **zaszumione** (tzn. częściowo losowe), wygląda na to, że poziom satysfakcji z życia wzrasta w sposób mniej więcej liniowy wraz ze wzrostem produktu krajowego brutto na osobę. Postanawiasz zatem stworzyć model satysfakcji z życia jako funkcji liniowej wobec parametru PKB per capita. Etap ten nazywamy **doborem modelu**: wybraliśmy **model liniowy** satysfakcji z życia wykorzystujący tylko jeden atrybut — PKB per capita (równanie 1.1).

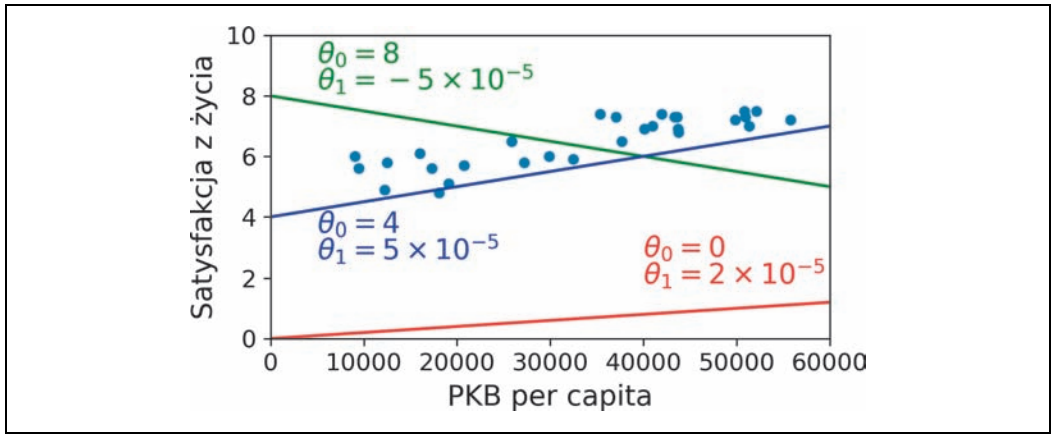


Rysunek 1.17. Czy widzisz tu trend?

Równanie 1.1. Prosty model liniowy

$$satysfakcja\_z\_życia = \theta_0 + \theta_1 \times PKB\_per\_capita$$

Model ten zawiera dwa **parametry**, θ_0 i θ_1[7]. Poprzez modyfikowanie tych parametrów możesz za ich pomocą uzyskać dowolną funkcję liniową, co zostało ukazane na rysunku 1.18.



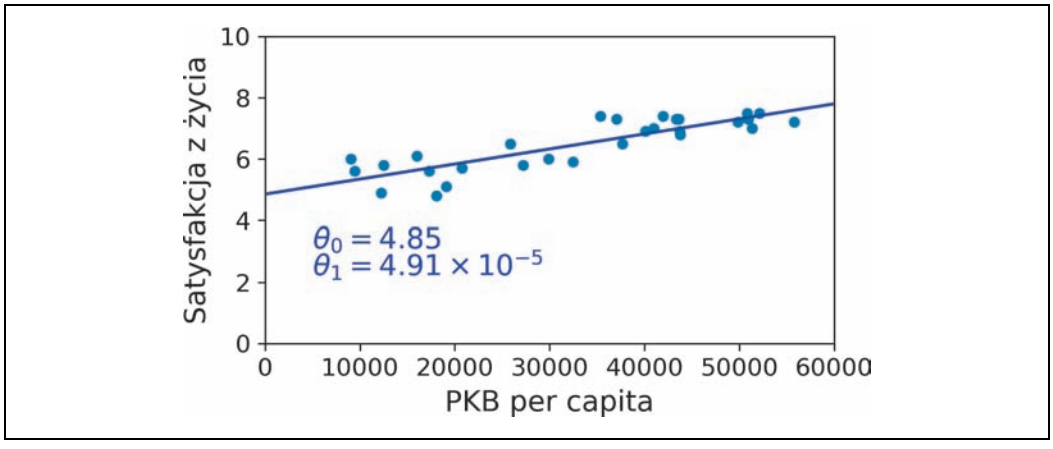
Rysunek 1.18. Kilka wybranych modeli liniowych

Zanim zaczniesz korzystać z modelu, musisz zdefiniować wartości jego parametrów θ_0 i θ_1. Skąd mamy wiedzieć, które wartości nadają się najlepiej dla danego modelu? Aby móc odpowiedzieć na to pytanie, należy zdefiniować miarę szybkości. Możesz wyznaczyć **funkcję użyteczności** (zwaną także **funkcją dopasowania**), mówiącą nam, jak **dobry** jest dany model, lub **funkcję kosztu**, mającą przeciwne zastosowanie. W zagadnieniach wykorzystujących regresję liniową zazwyczaj jest stosowana funkcja kosztu mierząca odległości pomiędzy przewidywaniami modelu liniowego a przykładami uczącymi; naszym zadaniem jest zminimalizowanie tego dystansu.

[7] Zgodnie z konwencją grecka litera θ (teta) często symbolizuje parametry modelu.

W tym właśnie miejscu przydaje się algorytm regresji liniowej: dostarczamy mu dane uczące, a on określa parametry najlepiej pasujące do danego modelu liniowego. Proces ten nosi nazwę uczenia (trenowania) modelu. W naszym przykładzie algorytm regresji liniowej wyznacza następujące optymalne wartości parametrów: $\theta_0 = 4{,}85$ i $\theta_1 = 4{,}91 \times 10^{-5}$.

Teraz nasz model jest maksymalnie dopasowany do danych uczących (jak na model liniowy), o czym możemy się przekonać na rysunku 1.19.



Rysunek 1.19. Model liniowy najlepiej dopasowany do danych uczących

Możemy w końcu uruchomić model, aby przeprowadził prognozy. Załóżmy, na przykład, że chcemy się dowiedzieć, jak szczęśliwi są Cypryjczycy; informacji tej nie znajdziemy w bazie danych OECD. Na szczęście, możemy posłużyć się modelem, aby uzyskać wiarygodne przewidywania: sprawdzamy wartość PKB per capita dla Cypru (22 587 dolarów), a po uruchomieniu modelu dowiadujemy się, że współczynnik satysfakcji z życia oscyluje wokół wartości $4{,}85 + 22\,587 \times 4{,}91 \times 10^{-5} = 5{,}96$.

Zaostrzymy Ci apetyt, pokazując w przykładzie 1.1 kod Python służący do wczytywania danych, ich przygotowania[8], utworzenia wykresu, a także wyuczenia modelu liniowego i prognozowania wyników[9].

Przykład 1.1. Uczenie i uruchamianie modelu liniowego za pomocą biblioteki Scikit-Learn

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn

# Wczytuje dane
oecd_bli = pd.read_csv("oecd_bli_2015.csv", thousands=',')
```

[8] Zakładamy tu, że funkcja `prepare_country_stats()` została już zdefiniowana: łączy ona dane dotyczące PKB i satysfakcji z życia w jeden obiekt `Dataframe` biblioteki Pandas.

[9] Nie szkodzi, jeżeli nie do końca rozumiesz ukazany kod; w następnych rozdziałach zajmiemy się omówieniem biblioteki Scikit-Learn.

```
gdp_per_capita = pd.read_csv("gdp_per_capita.csv",thousands=',',delimiter='\t',
                                                  encoding='latin1', na_values="bd")

# Przygotowuje dane
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["PKB per capita"]]
y = np.c_[country_stats["Satysfakcja z życia"]]

# Wizualizuje dane
country_stats.plot(kind='scatter', x="PKB per capita", y='Satysfakcja z życia')
plt.show()

# Dobiera model liniowy
model = sklearn.linear_model.LinearRegression()

# Trenuje model
model.fit(X, y)

# Prognozuje wyniki dla Cypru
X_new = [[22587]] # PKB per capita dla Cypru
print(model.predict(X_new)) # generuje wynik [[ 5.96242338]]
```

Gdybyś użyła/użył algorytmu uczenia z przykładów, zauważyłabyś/zauważyłbyś, że wartość PKB najbardziej zbliżoną do Cypru ma Słowenia (20 732 dolary), a skoro dzięki danym OECD wiemy, że współczynnik satysfakcji z życia wynosi u Słoweńców 5,7, moglibyśmy w drodze analogii stwierdzić, że jego wartość dla Cypryjczyków powinna być bardzo podobna. Po nieznacznym „oddaleniu skali" i przyjrzeniu się dwóm kolejnym państwom o zbliżonej wartości satysfakcji z życia okaże się, że są to Portugalia (5,1) i Hiszpania (6,5). Średnia tych trzech wartości wynosi 5,77, co stanowi wartość całkiem zbliżoną do wyliczonej przez nasz model. Wspomniany tu prosty algorytm nosi nazwę regresji **k-najbliższych sąsiadów** (ang. *k-nearest neighbors*); w tym przykładzie $k = 3$.

Aby zastąpić algorytm regresji liniowej algorytmem k-najbliższych sąsiadów, wystarczy podmienić wiersz:

```
model = sklearn.linear_model.LinearRegression()
```

na następujący:

```
model = sklearn.neighbors.KNeighborsRegressor(n_neighbors=3)
```

Jeśli wszystko zostało dobrze wykonane, Twój model powinien wyliczać dobre prognozy. W przeciwnym wypadku może być konieczne wykorzystanie większej liczby atrybutów (stopy zatrudnienia, zdrowia społeczeństwa, zanieczyszczenia powietrza itd.), zdobycie więcej danych uczących lub danych lepszej jakości, ewentualnie skorzystanie z wydajniejszego modelu (np. algorytmu regresji wielomianowej).

Podsumowując:

- Przeanalizowaliśmy dane.
- Wybraliśmy model.
- Wytrenowaliśmy go na danych uczących (np. algorytm uczący wyszukał wartości parametrów pozwalających na zminimalizowanie funkcji kosztu).

- Na końcu wykorzystaliśmy wytrenowany model do prognozowania wyników dla nowych przypadków (jest to tzw. **wnioskowanie**) z nadzieją, że będzie on skutecznie generalizował zdobytą wiedzę.

Tak właśnie wygląda typowy projekt uczenia maszynowego. W rozdziale 2. doświadczysz tego na własnej skórze podczas przygotowywania projektu od podstaw.

Poruszyliśmy do tej pory wiele zagadnień: wiesz już, czym jest uczenie maszynowe, dlaczego okazuje się przydatne i jakie są podstawowe kategorie tej dziedziny, a także znasz już podstawowy cykl pracy nad projektem uczenia maszynowego. Spójrzmy teraz, co może pójść nieprawidłowo w procesie uczenia i uniemożliwić uzyskiwanie dokładnych prognoz.

Główne problemy uczenia maszynowego

Skoro naszym głównym zadaniem jest dobór algorytmu uczącego i wyuczenie go za pomocą określonych danych, to dwoma największymi zagrożeniami są „zły algorytm" i „złe dane". Zacznijmy od przyjrzenia się niewłaściwym danym.

Niedobór danych uczących

Aby nauczyć berbecia, czym jest jabłko, wystarczy wskazać mu ten owoc i powiedzieć „jabłko" (najprawdopodobniej musisz to jeszcze kilkakrotnie powtórzyć). Dziecko teraz będzie w stanie rozpoznawać jabłka w najróżniejszych kolorach i kształtach. Mały geniusz.

Niestety, techniki uczenia maszynowego znajdują się pod tym względem daleko w tyle; żeby większość algorytmów działała prawidłowo, należy im zapewnić mnóstwo danych. Nawet w przypadku bardzo prostych problemów należy zazwyczaj dostarczyć tysiące przykładów, natomiast bardziej zaawansowane zagadnienia, takie jak rozpoznawanie mowy, wymagają nawet milionów próbek (chyba że jesteś w stanie wykorzystać fragmenty już istniejącego modelu).

Niedorzeczna efektywność danych

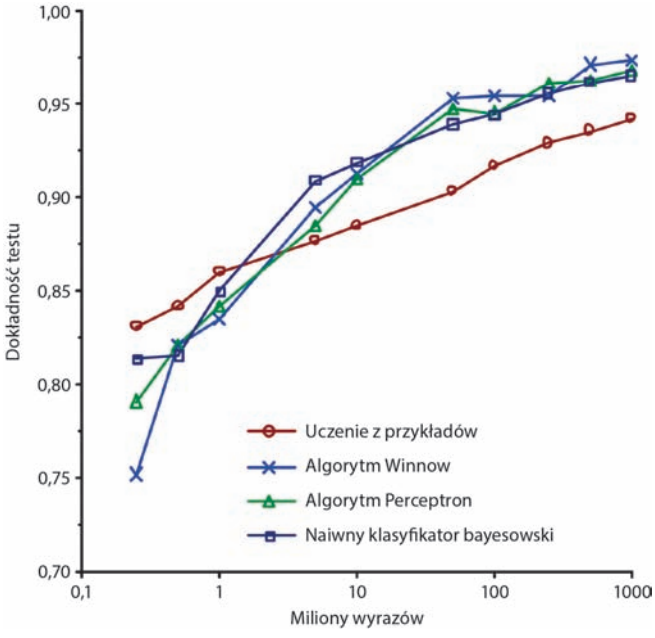
W słynnym dokumencie (*http://www.aclweb.org/anthology/P01-1005*) z 2001 roku badacze z firmy Microsoft, Michele Banko i Eric Brill, udowodnili, że różne algorytmy uczenia maszynowego, włącznie z najprostszymi, po przesłaniu im odpowiedniej ilości danych osiągają niemal identyczną wydajność wobec złożonych problemów ujednoznacznień w języku naturalnym[10] (co zaprezentowano na rysunku 1.20).

Autorzy ujmują to następująco: „Wyniki te sugerują, że warto zastanowić się nad przeniesieniem środka ciężkości w kompromisie pomiędzy poświęcaniem czasu i pieniędzy na tworzenie algorytmów a przeznaczaniem zasobów na rozwój korpusu językowego".

Koncepcja przewagi znaczenia danych nad algorytmami w rozwiązywaniu złożonych problemów została spopularyzowana przez Petera Nordviga i in. w dokumencie *The Unreasonable Effectiveness of Data* (*https://research.google.com/pubs/archive/35179.pdf*), opublikowanym w 2009 roku[11]. Należy jednak zauważyć, że ciągle są popularne niewielkie i średniej wielkości zbiory danych, a pozyskanie ich większej ilości często bywa bardzo trudne lub kosztowne, dlatego nie bagatelizujmy znaczenia algorytmów.

[10] Na przykład rozróżnianie znaczeń wyrazów „buk", „Bug" i „bóg" w zależności od kontekstu.

[11] *The Unreasonable Effectiveness of Data*, Peter Norvig i in. (2009).

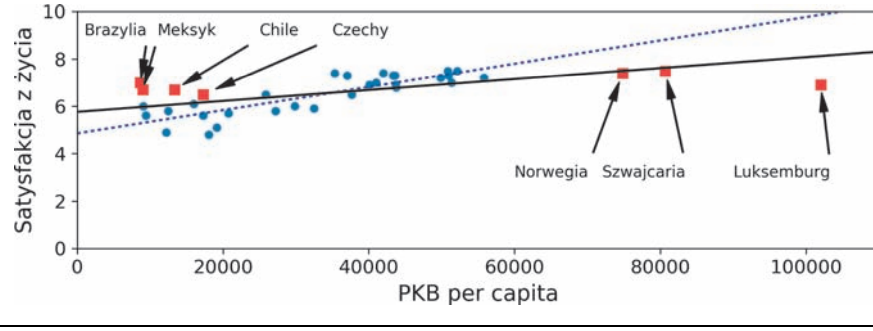


Rysunek 1.20. Porównanie znaczenia danych i algorytmów[12]

Niereprezentatywne dane uczące

Aby proces uogólniania przebiegał skutecznie, dane uczące muszą być reprezentatywne wobec nowych danych, wobec których nastąpi generalizacja. Jest to prawdziwe zarówno dla uczenia z przykładów, jak i dla uczenia z modelu.

Na przykład zbiór krajów użyty przez nas do trenowania modelu liniowego nie był doskonale reprezentatywny; brakuje w nim kilku państw. Na rysunku 1.21 widzimy, jak będą wyglądać dane po dodaniu tych kilku brakujących krajów.



Rysunek 1.21. Bardziej reprezentatywny zbiór przykładów uczących

[12] Rysunek umieszczony za pozwoleniem Banko i Brilla (2001), *Learning Curves for Confusion Set Disambiguation*.

Model wyuczony za pomocą tych danych został ukazany w postaci linii ciągłej, natomiast stary model jest reprezentowany przez linię przerywaną. Jak widać, dodanie kilku brakujących krajów nie tylko w znaczny sposób modyfikuje dany model, ale pozwala nam również uzmysłowić sobie, że prosty model liniowy raczej nigdy nie będzie bardzo dokładny. Wygląda na to, że mieszkańcy bardzo bogatych państw nie są szczęśliwsi od ich odpowiedników w średnio zamożnych krajach (wydaje się wręcz, że są bardziej nieszczęśliwi), a z kolei ludność wielu ubogich państw wydaje się znacznie szczęśliwsza od populacji bogatych krajów.

Za pomocą niereprezentatywnego zbioru danych wyuczyliśmy model, który raczej nie będzie generował dokładnych prognoz, zwłaszcza w przypadku krajów ubogich i bardzo zamożnych.

Jest niezwykle istotne, aby zbiór danych uczących był reprezentatywny wobec generalizowanych przypadków. Często jest to trudniej osiągnąć, niż się wydaje: jeżeli przykład będzie zbyt mały, musisz liczyć się z **zaszumieniem próbkowania** (ang. *sampling noise*; niereprezentatywne dane pojawiają się tu w wyniku przypadku), ale nawet olbrzymie zbiory danych mogą nie być reprezentatywne, jeśli użyta zostanie niewłaściwa metoda próbkowania. W tym przypadku mamy do czynienia z **obciążeniem próbkowania** (ang. *sampling bias*).

Słynny przykład obciążenia próbkowania

Prawdopodobnie najsłynniejszy przykład wpływu obciążenia próbkowania można było zaobserwować podczas wyborów na prezydenta Stanów Zjednoczonych w 1936 roku, w których przeciwko Rooseveltowi stanął Landon: magazyn „Literary Digest" przeprowadził ankietę, wysyłając pocztą ulotki do około 10 milionów obywateli. Zostało odesłanych 2,4 miliona odpowiedzi, dzięki którym z dużą dozą pewności prognozowano, że Landon uzyska 57% głosów.

Ostatecznie okazało się, że to Roosevelt zebrał 62% głosów. Problem polegał na użytej metodzie próbkowania:

- Po pierwsze, magazyn „Literary Digest" zebrał adresy osób, którym została wysłana ankieta, z książek telefonicznych, list abonentów i członków klubów itd. We wszystkich tych źródłach dominowały majętne osoby, które chętniej głosowały na republikanów (czyli na Landona).
- Po drugie, odpowiedzi odesłało mniej niż 25% osób. Również ten czynnik odpowiada za obciążenie próbkowania, ponieważ nie są brane pod uwagę osoby apolityczne, przeciwnicy magazynu „Literary Digest" i inne kluczowe grupy. Jest to specjalny rodzaj obciążenia próbkowania, zwany **błędem braku odpowiedzi** (ang. *nonresponse bias*).

Weźmy pod uwagę inny przykład: powiedzmy, że chcesz stworzyć system rozpoznający teledyski z muzyką funkową. Jednym ze sposobów przygotowania zbioru danych uczących jest wyszukanie wyników „funk music" w serwisie YouTube i skorzystanie z nich. Zakładamy jednak w tym przypadku, że silnik wyszukiwania zwróci zestaw teledysków stanowiący przekrój wszystkich piosenek funkowych umieszczonych w serwisie. W rzeczywistości wyniki wyszukiwania są często tendencyjnie nastawione wobec najpopularniejszych artystów (natomiast jeśli mieszkasz w Brazylii, zostanie wyświetlonych wiele teledysków z nurtu „funk carioca", który w ogóle nie brzmi jak James Brown). Z drugiej strony, jak inaczej można pozyskać duży zbiór danych uczących?

Dane kiepskiej jakości

Jest oczywiste, że jeśli dane uczące zawierają mnóstwo błędów, elementów odstających i szumu (np. z powodu niskiej jakości pomiarów), to systemowi będzie znacznie trudniej wykryć wzorce, przez co nie osiągnie optymalnej wydajności. Warto często poświęcić czas na oczyszczenie takich danych. Prawda jest taka, że większość analityków danych poświęca na tę czynność wiele czasu. Na przykład:

- jeżeli niektóre elementy wyraźnie odstają od reszty przykładów, wystarczy je po prostu odrzucić albo spróbować własnoręcznie poprawić błędy,
- jeśli niektórym przykładom brakuje kilku cech (np. 5% klientów nie podało wieku), musisz zadecydować, czy należy takie cechy całkowicie ignorować, ignorować te przykłady, wypełnić brakujące wartości (np. korzystając z mediany wieku), wytrenować jeden model przy użyciu tej cechy, a drugi bez niej itd.

Nieistotne cechy

Zgodnie ze słynnym powiedzeniem: z gipsu tortu nie ulepisz. Twój system będzie w stanie uczyć się jedynie za pomocą danych zawierających wystarczającą liczbę istotnych cech i niezaśmieconych nadmiarem cech nieistotnych. Elementem krytycznym dla sukcesu w projekcie uczenia maszynowego jest wybór dobrego zbioru cech uczących. Proces ten, zwany **inżynierią cech** (ang. *feature engineering*), składa się z następujących elementów:

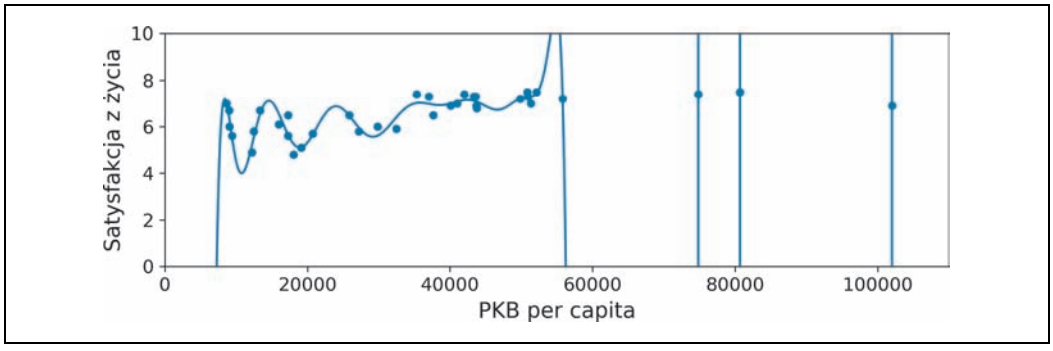
- **dobór cechy** (ang. *feature selection*): dobór najprzydatniejszych spośród dostępnych cech,
- **odkrywanie cechy** (ang. *feature extraction*): łączenie ze sobą istniejących cech w celu uzyskania przydatniejszej cechy (jak już wiemy, pomóc nam w tym może algorytm redukcji wymiarowości),
- uzyskiwanie nowych cech z nowych danych.

Skoro już wiemy, jak wyglądają złe dane, przeanalizujmy kwestię nieprawidłowych algorytmów.

Przetrenowanie danych uczących

Załóżmy, że na zagranicznej wycieczce zostałaś okradziona/zostałeś okradziony przez taksówkarza. Być może na Twoje usta ciśnie się stwierdzenie, że *wszyscy* taksówkarze w danym kraju są złodziejami. Ludzie często mają tendencję do nadmiernego generalizowania i, niestety, maszyny równie łatwo wpadają w tę pułapkę, jeśli nie zachowamy ostrożności. Zjawisko to w terminologii uczenia maszynowego nosi nazwę **przetrenowania** albo **nadmiernego dopasowania** (ang. *overfitting*). Termin ten oznacza, że model sprawdza się w przypadku danych uczących, ale sam proces uogólniania nie sprawuje się zbyt dobrze.

Na rysunku 1.22 widzimy przykład wielomianowego modelu satysfakcji z życia o olbrzymim stopniu przetrenowania za pomocą danych uczących. Nawet jeśli sprawuje się on znacznie lepiej wobec danych uczących niż zwykły model liniowy, to czy moglibyśmy zawierzyć jego prognozom?



Rysunek 1.22. Przetrenowanie danych uczących

Takie złożone modele, jak głębokie sieci neuronowe, są w stanie wykrywać subtelne wzorce w danych, ale jeżeli zbiór danych uczących jest zaszumiony lub zbyt mały (ryzyko zaszumienia próbkowania), to model ten prawdopodobnie będzie wykrywał wzorce nie w użytecznych danych, lecz w szumie. Oczywiście. szablonów tych nie da się generalizować na nowe próbki. Powiedzmy, że dostarczamy modelowi satysfakcji z życia wiele dodatkowych atrybutów, w tym takich nieprzydatnych, jak nazwy państw. W takiej sytuacji złożony model może wykrywać takie wzorce, jak np. wskaźnik satysfakcji z życia przekraczający wartość 7 w krajach mających w nazwie literę „w": Nowa Zelandia (7,3), Norwegia (7,4), Szwecja (7,2) czy Szwajcaria (7,5). Czy masz pewność, że ta „reguła litery »w«" dotyczy również takich państw, jak Rwanda lub Zimbabwe? Oczywiście, wzorzec ten wystąpił w danych uczących zupełnie przypadkowo, ale model nie jest w stanie stwierdzić, czy taki szablon jest rzeczywisty, czy stanowi wynik zaszumienia danych.

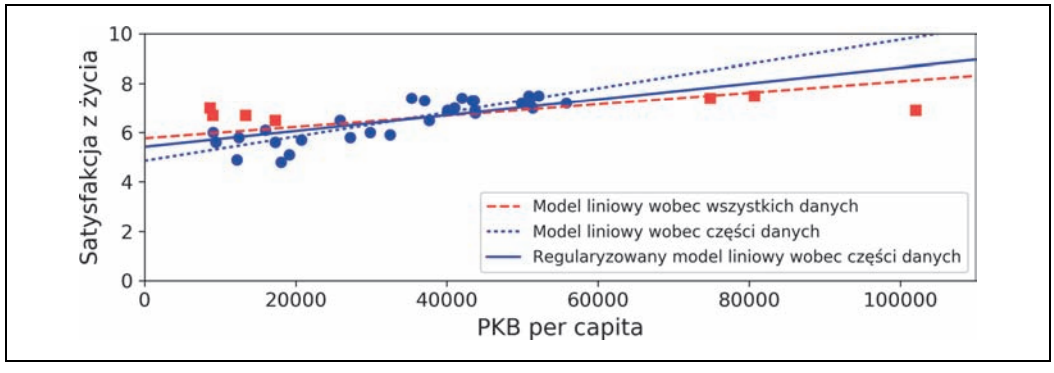
Zjawisko przetrenowania występuje, gdy model jest zbyt skomplikowany w porównaniu do ilości lub zaszumienia danych uczących. W takich przypadkach możliwe są następujące rozwiązania:

- uproszczenie modelu poprzez wybór zawierającego mniej parametrów (np. modelu liniowego w miejsce modelu wielomianowego), zmniejszenie liczby atrybutów w danych uczących lub ograniczenie modelu,
- uzyskanie większej ilości danych uczących,
- zmniejszenie zaszumienia danych uczących (np. poprzez usunięcie błędnych danych lub elementów odstających).

Ograniczenie modelu w celu jego uproszczenia i zmniejszenia ryzyka przetrenowania nosi nazwę **regularyzacji** (ang. *regularization*). Przykładowo, zdefiniowany przez nas wcześniej model liniowy zawiera dwa parametry: θ_0 i θ_1. Algorytm uczący uzyskuje w ten sposób dwa **stopnie swobody**, za pomocą których możemy dostosowywać model do danych uczących: mamy wpływ zarówno na wysokość (θ_0), jak i nachylenie (θ_1) funkcji. Gdybyśmy wymusili wartość $\theta_1 = 0$, algorytm zostałby ograniczony do jednego stopnia swobody i jego prawidłowe wytrenowanie stałoby się znacznie trudniejsze: moglibyśmy jedynie zmieniać położenie funkcji wzdłuż osi y i starać się umieścić jej przebieg jak najbliżej punktów uczących, co w tym przypadku oznaczałoby określenie wartości średniej. Zaiste, bardzo prosty model! Jeżeli pozwolimy algorytmowi na nieznaczne zmiany wartości parametru θ_1, to w istocie algorytm będzie znajdował się gdzieś pośrednio pomiędzy jednym a dwoma

stopniami swobody. Uzyskamy w ten sposób model prostszy od mającego dwa stopnie swobody, ale bardziej złożony od jednostopniowego. Musimy wyszukiwać odpowiednią równowagę pomiędzy perfekcyjnym dopasowywaniem danych a zachowaniem odpowiedniej prostoty modelu pozwalającej na generalizowanie wyników.

Rysunek 1.23 prezentuje trzy modele: linia kropkowana symbolizuje nasz oryginalny model wytrenowany pod nieobecność kilku krajów, linia kreskowana ukazuje nasz model wyuczony przy użyciu pełnego zakresu danych, natomiast linia ciągła reprezentuje model wytrenowany za pomocą takich samych danych, jak w pierwszym przypadku, ale przy wprowadzonej regularyzacji. Widzimy, że w wyniku regularyzacji funkcja ma trochę mniejsze nachylenie, co sprawia, że jest nieco mniej dopasowana do danych uczących, ale za to sprawuje się lepiej w procesie uogólniania nowych przykładów.



Rysunek 1.23. Regularyzacja zmniejsza ryzyko przetrenowania

Stopień regularyzacji przeprowadzanej na etapie nauki możemy kontrolować za pomocą **hiperparametrów** (ang. *hyperparameters*). Najprościej mówiąc, hiperparametrami nazywamy parametry algorytmu uczącego (nie całego modelu). Nie są one modyfikowane przez sam algorytm uczący; należy je wyznaczyć tuż przed rozpoczęciem procesu uczenia i w jego trakcie ich wartości pozostają niezmienne. Jeśli wyznaczysz bardzo dużą wartość hiperparametru regularyzacji, uzyskana funkcja będzie niemal płaska (będzie miała niemal zerowe nachylenie); algorytm uczący prawie na pewno nie ulegnie przetrenowaniu, ale jednocześnie trudniej będzie mu znaleźć prawidłowe rozwiązanie. Strojenie hiperparametrów stanowi istotną część tworzenia systemu uczenia maszynowego (w następnym rozdziale przekonasz się o tym na własnej skórze).

Niedotrenowanie danych uczących

Jak łatwo się domyślić, **niedotrenowanie** (ang. *underfitting*) stanowi przeciwieństwo przetrenowania: występuje ono wtedy, gdy model jest zbyt prosty, aby wyuczyć się struktur danych uczących. Na przykład, liniowy model satysfakcji z życia jest podatny na niedotrenowanie; rzeczywistość jest po prostu bardziej skomplikowana od modelu, zatem jego prognozy nie będą dokładne, nawet w przypadku danych uczących.

Głównymi sposobami rozwiązania tego problemu są:

- wybór potężniejszego modelu wykorzystującego większą liczbę parametrów,
- dołączenie większej liczby cech do algorytmu uczącego (inżyniera cech),
- zmniejszenie ograniczeń modelu (np. zredukowanie hiperparametru regularyzacji).

Podsumowanie

Wiemy już coraz więcej na temat uczenia maszynowego. Wspomnieliśmy jednak o tak wielu pojęciach, że możesz poczuć zagubienie, dlatego zatrzymajmy się na chwilę i sprawdźmy, co już wiemy:

- Uczenie maszynowe polega na ciągłym usprawnianiu wykonywania zadań przez urządzenie poprzez jego uczenie się z danych, a nie jawne dopisywanie nowych reguł do kodu.
- Istnieje wiele różnych metod uczenia maszynowego: nadzorowane, nienadzorowane, wsadowe, przyrostowe, z przykładów, z modelu itd.
- W projekcie uczenia maszynowego gromadzimy dane w zbiorze uczącym, który następnie dostarczamy algorytmowi uczącemu. Jeśli wykorzystujemy technikę uczenia z modelu, pewne parametry zostają dostrojone do zbioru danych uczących (np. w celu uzyskiwania dokładnych prognoz wobec samych danych uczących), dzięki czemu model ten może również dobrze przewidywać wyniki wobec nowych przykładów. Z kolei w metodach uczenia z przykładów próbki zostają po prostu wyuczone „na pamięć", a model wykorzystuje miarę podobieństwa do uogólniania wyników na nowe przykłady.
- System nie będzie dobrze spełniał swojego zadania, jeżeli zbiór danych testowych jest zbyt mały, same dane są niereprezentatywne, zaszumione lub zaśmiecone nieistotnymi cechami (z gipsu tortu nie ulepisz). Model nie może być również zbyt prosty (niedotrenowanie) ani zbyt skomplikowany (przetrenowanie).

Musimy omówić jeszcze jedno istotne zagadnienie: po wyuczeniu modelu nie wystarczy sama nadzieja na skuteczne uogólnianie nowych próbek. Chcesz ocenić wydajność modelu i w razie potrzeby dostroić go do swoich potrzeb. Dowiesz się teraz, jak tego dokonać.

Testowanie i ocenianie

Jedynym sposobem sprawdzenia, jak dobrze model generalizuje wyniki, jest wypróbowanie go na nowych danych. Możemy, na przykład, zaimplementować go w środowisku produkcyjnym i monitorować jego wydajność. Jest to skuteczne rozwiązanie, ale jeśli model okaże się beznadziejny, użytkownicy będą narzekać — a tego lepiej unikać.

Lepszym pomysłem jest podział danych na dwa zestawy: **zbiór uczący** (ang. *training set*) i **zbiór testowy** (ang. *test set*). Zgodnie z tymi nazwami, model trenujemy za pomocą zbioru uczącego, a sprawdzamy przy użyciu zbioru testowego. Współczynnik błędu uzyskiwany dla nowych przykładów nosi nazwę **błędu uogólniania** (lub **błędu generalizacji**), a dzięki zbiorowi testowemu możemy oszacować jego wartość. Parametr ten mówi nam również, jak dobrze model będzie się spisywał wobec nieznanych danych.

Jeśli wartość błędu uczenia jest niewielka (tzn. model rzadko się myli wobec zbioru uczącego), ale błąd uogólniania jest duży, oznacza to, że model jest przetrenowany.

Zazwyczaj na zbiór uczący składa się 80% danych, a pozostałe 20% *przechowujemy* w celu testowania.

Zatem ocena modelu nie jest wcale taka trudna: wystarczy użyć zbioru testowego. Załóżmy teraz, że wahamy się przy wyborze jednego z dwóch modeli (np. pomiędzy wyborem modelu liniowego a wielomianowego): w jaki sposób mamy zdecydować? Jednym z rozwiązań jest wyuczenie obydwu modeli i sprawdzenie, jak sobie radzą z uogólnianiem wobec zbioru testowego.

Przyjmijmy teraz, że model liniowy radzi sobie lepiej z uogólnianiem, ale chcesz wprowadzić regularyzację w celu uniknięcia przetrenowania. Rodzi się w tym momencie pytanie: w jaki sposób dobrać wartość hiperparametru regularyzacji? Możesz na przykład wytrenować 100 różnych modeli, z których każdy będzie miał inną wartość tego hiperparametru. Załóżmy, że znajdziesz optymalną wartość tego hiperparametru, dającą model o najniższym błędzie generalizacji (np. rzędu 5%).

Zatem wprowadzasz ten model do środowiska produkcyjnego, ale niestety, sprawuje się on poniżej oczekiwań i generuje 15% błędów. Co się właściwie stało?

Problem polega na tym, że wielokrotnie mierzyłaś/mierzyłeś błąd uogólniania wobec zbioru testowego i dostosowałaś/dostosowałeś go *do tych danych*. Oznacza to, że teraz prawdopodobnie model ten nie będzie sobie radził zbyt dobrze z nowymi próbkami.

Powszechnie stosowanym rozwiązaniem tego problemu jest odłożenie jeszcze jednego zbioru danych, zwanego **zbiorem walidacyjnym** (ang. *validation set*). Trenujemy wiele modeli mających różne wartości hiperparametrów za pomocą zbioru uczącego, dobieramy model i hiperparametry najlepiej sprawujące się wobec zbioru walidacyjnego, a gdy uzyskiwane wyniki będą Cię zadowalać, wystarczy przeprowadzić ostatni sprawdzian wobec zbioru testowego, aby oszacować wartość błędu uogólniania.

Aby uniknąć „zużywania" zbyt dużej ilości danych uczących na zbiór walidacyjny, możesz skorzystać z techniki zwanej **sprawdzianem krzyżowym** lub **kroswalidacją** (ang. *cross-validation*): zbiór uczący zostaje rozdzielony na wzajemnie uzupełniające się podzbiory; każdy model jest uczony za pomocą różnych kombinacji tych podzbiorów i oceniany przy użyciu pozostałych, nieużytych podzestawów. Po dobraniu rodzaju i hiperparametrów modelu ostateczny model zostaje wytrenowany wobec pełnego zbioru uczącego, a błąd uogólniania określamy przy użyciu zbioru testowego.

Twierdzenie o nieistnieniu darmowych obiadów

Model stanowi uproszczoną postać obserwacji. Uproszczenia wynikają z konieczności unikania niepotrzebnych szczegółów, które nie ułatwiają procesu uogólniania. Podczas dobierania i odrzucania danych musimy jednak przyjmować **założenia**. Na przykład w modelu liniowym zakładamy, że dane są, nomen omen, liniowe, a odległość pomiędzy przykładami i wykresem funkcji stanowi wyłącznie szum, który możemy bezpiecznie zignorować.

W słynnej publikacji z 1996 roku[13] (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.390.9412&rep=rep1&type=pdf*) David Wolpert udowodnił, że jeśli nie przyjmiemy jakichkolwiek założeń dotyczących danych, to okaże się, że żaden model nie będzie lepszy od pozostałych. Jest to tak zwane **twierdzenie o nieistnieniu darmowych obiadów** (ang. *No Free Lunch Theorem* — NFL). Dla pewnych zbiorów danych najlepiej nadaje się model liniowy, natomiast dla innych — sieci neuronowe. Nie istnieje żaden model, który z założenia będzie działał lepiej (stąd nazwa twierdzenia). Jedynie poprzez ocenę działania każdego modelu możemy przekonać się, który z nich będzie sprawował się najlepiej. Jest to niewykonalne, dlatego w praktyce przyjmujemy rozsądne założenia dotyczące danych i oceniamy działanie tylko kilku rozsądnie dobranych modeli. Na przykład wobec prostych zadań możemy ocenić działanie modeli liniowych różniących się stopniem regularyzacji, a bardziej skomplikowane problemy możemy przetestować przy użyciu sieci neuronowych.

Ćwiczenia

W rozdziale tym poznaliśmy część najważniejszych koncepcji opisujących dziedzinę uczenia maszynowego. W kolejnych rozdziałach skoncentrujemy się na szczegółach i pisaniu właściwego kodu, zanim jednak przejdziesz dalej, upewnij się, że jesteś w stanie odpowiedzieć na poniższe pytania:

1. Podaj definicję uczenia maszynowego.
2. Wymień cztery rodzaje problemów, z których rozwiązaniem najlepiej sobie radzi proces uczenia maszynowego.
3. Czym jest oznakowany zbiór danych uczących?
4. Jakie są dwa najczęstsze zastosowania uczenia nadzorowanego?
5. Wymień cztery najpowszechniejsze zastosowania uczenia nienadzorowanego.
6. Jakiego rodzaju algorytmu uczenia maszynowego użyłabyś/użyłbyś w robocie przeznaczonym do poruszania się po nieznanym terenie?
7. Jakiego rodzaju algorytmu uczenia maszynowego użyłabyś/użyłbyś do rozdzielenia klientów na kilka różnych grup?
8. Czy problem wykrywania spamu stanowi część mechanizmu uczenia nadzorowanego lub nienadzorowanego?
9. Czym jest system uczenia przyrostowego?
10. Czym jest uczenie pozakorowe?
11. W jakim algorytmie uczenia maszynowego jest wymagana miara podobieństwa do uzyskiwania prognoz?
12. Wyjaśnij różnicę pomiędzy parametrem modelu a hiperparametrem algorytmu uczącego.
13. Czego poszukują algorytmy uczenia z modelu? Z jakiej strategii najczęściej korzystają? W jaki sposób przeprowadzają prognozy?

[13] *The Lack of A Priori Distinctions Between Learning Algorithms*, D. Wolpert (1996).

14. Wymień cztery główne problemy związane z uczeniem maszynowym.
15. Z czym mamy do czynienia, jeżeli model sprawuje się znakomicie wobec danych uczących, ale nie radzi sobie z uogólnianiem wobec nowych próbek? Wymień trzy możliwe rozwiązania.
16. Czym jest zbiór testowy i dlaczego powinniśmy z niego korzystać?
17. Do czego służy zbiór walidacyjny?
18. Co nam grozi w przypadku strojenia hiperparametru wobec zbioru testowego?
19. Czym jest sprawdzian krzyżowy oraz dlaczego warto go stosować wobec zbioru walidacyjnego?

Odpowiedzi znajdziesz w dodatku A.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

W

Z

O'REILLY®

Już dziś zacznij tworzyć inteligentne systemy!

W ciągu ostatnich lat uczenie maszynowe stało się sercem wielu nowoczesnych produktów, takich jak zaawansowane techniki wyszukiwania w przeglądarkach, rozpoznawanie mowy w smartfonach czy proponowanie treści w zależności od indywidualnych preferencji użytkownika. Być może niedługo taki inteligentny system zastąpi nas za kierownicą samochodu. Uczenie głębokie wprowadziło nową jakość do uczenia maszynowego. Daje niesamowite możliwości, jednak wymaga olbrzymiej mocy obliczeniowej i potężnych ilości danych. Programiści implementujący takie rozwiązania są poszukiwanymi specjalistami i mogą liczyć na ekscytujące oferty!

Ta książka jest praktycznym podręcznikiem tworzenia systemów inteligentnych. Przedstawiono tu najważniejsze zagadnienia teoretyczne dotyczące uczenia maszynowego i sieci neuronowych. W zrozumiały sposób zaprezentowano koncepcje i narzędzia służące do tworzenia systemów inteligentnych. Opisano Scikit-Learn i TensorFlow — środowiska produkcyjne języka Python — i pokazano krok po kroku, w jaki sposób wykorzystuje się je do implementacji sieci neuronowych. Liczne praktyczne przykłady i ćwiczenia pozwolą na pogłębienie i utrwalenie zdobytej wiedzy. Jeśli tylko potrafisz posługiwać się Pythonem, dzięki tej przystępnie napisanej książce szybko zaczniesz implementować systemy inteligentne.

W tej książce między innymi:

- podstawowe koncepcje uczenia maszynowego, uczenia głębokiego i sieci neuronowych
- przygotowywanie zbiorów danych i zarządzanie nimi
- algorytmy uczenia maszynowego
- rodzaje architektury sieci neuronowych
- uczenie głębokich sieci neuronowych

Aurélien Géron — specjalizuje się w dziedzinie uczenia maszynowego. Posiada również doświadczenie w branży finansowej, obronnej i medycznej. Był wykładowcą akademickim, pracował w Google, jest założycielem firmy Wifirst (wiodącego dostawcy bezprzewodowych usług internetowych we Francji). Napisał kilka książek o języku C++, sieciach Wi-Fi oraz architekturze sieci.

ISBN 978-83-283-4373-3
9 788328 343733

Cena: 89,00 zł