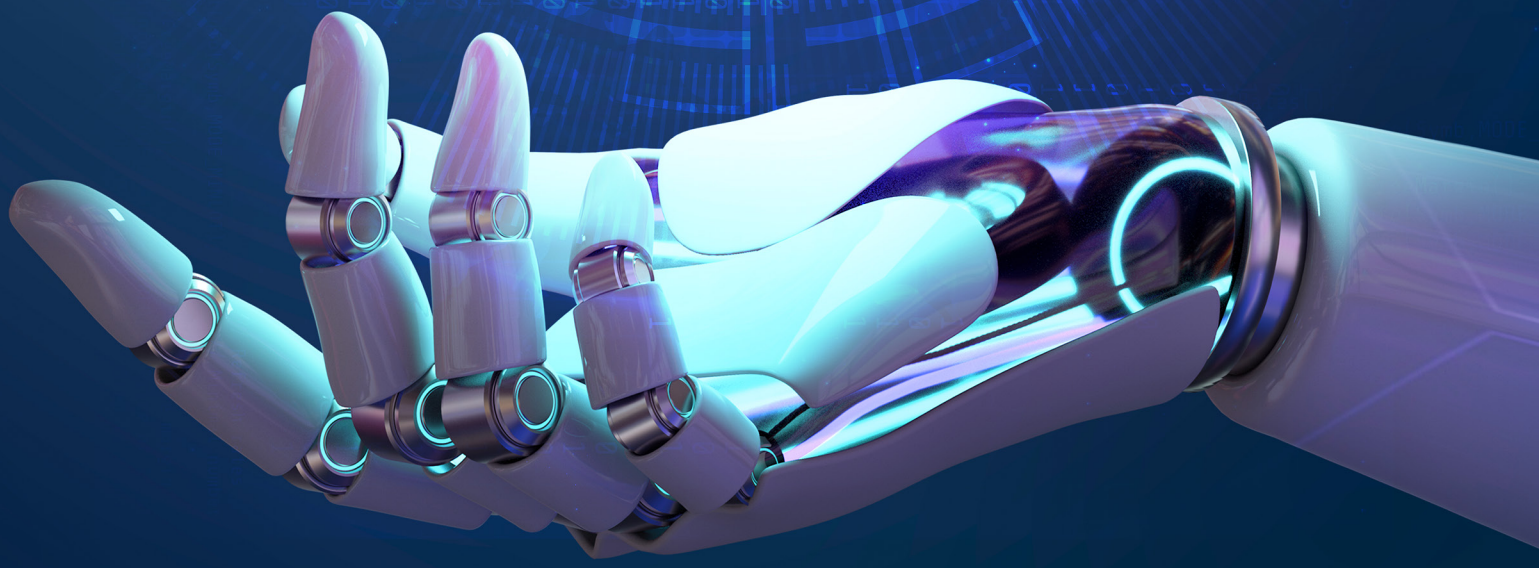
Sztuczna inteligencja

O CO W TYM CHODZI?



PWN

WSTĘP

Każdy programista w swojej pracy spotka się prędzej czy później z zagadnieniem sztucznej inteligencji, uczeniem się maszyn czy Big Data. Warto jest więc wiedzieć, jak działają systemy wykorzystujące te technologie. Publikacja *Sztuczna inteligencja. O co w tym chodzi?* zawiera omówienie podstawowych kwestii z zakresu sztucznej inteligencji. Joanathan E. Steinhart stara się w przystępny sposób omówić te skomplikowane zagadnienia, tak aby każdy – nawet osoba bez zaawansowanej wiedzy matematycznej i informatycznej – mógł je zrozumieć.

W książce zostały poruszone m.in. poniższe tematy:

- jak rozwijała się dziedzina nauki jaką jest sztuczna inteligencja,
- na czym polega uczenie się maszyn i jakie ma zastosowanie w praktyce,
- jakie zastosowanie w uczeniu się maszyn ma twierdzenie Bayesa,
- jak przeprowadzić ekstrakcję cech,
- jak w sztucznej inteligencji są wykorzystywane sieci neuronowe,
- jak są przetwarzane dane w ramach Big Data.

Sztuczna inteligencja w najbliższych latach będzie się dalej prężnie rozwijać. Poznaj najważniejsze zagadnienia tej technologii.

Wioleta Szczygielska-Dybciak

Wioleta Szczygielska-Dybciak
Wydawca Redakcji IT

SZTUCZNA INTELIGENCJA

Ile razy wpisywaliście do wyszukiwarki coś w rodzaju „jak odróżnić kota od paszteta" i otrzymywaliśmy w zamian „Czy chodziło ci o *pasztetu?*" Teraz zapewne wydaje nam się to na tyle oczywiste, że nawet nie zatrzymujemy się, by pomyśleć, w jaki sposób właściwie wyszukiwarka nie tylko rozpoznała błąd we wprowadzonych przez nas danych, ale jeszcze w dodatku wiedziała, jak go poprawić. Nie wydaje się możliwe, by ktoś napisał program zawierający wszystkie możliwe błędy wraz z ich korektą. To oczywiste, że musi tu działać jakaś *sztuczna inteligencja*.

Sztuczna inteligencja jest zaawansowanym tematem zawierającym pokrewne dziedziny, takie jak *uczenie się maszyn*, czy *big data* (dosł. wielkie dane). Z tymi wszystkimi zagadnieniami z pewnością zetkniemy się jako programiści, więc ten rozdział przedstawi ogólny przegląd tych zagadnień.

Termin *sztuczna inteligencja* został ukuty w 1956 roku w Dartmouth College. *Uczenie się maszyn* przyszło zaraz potem, w 1957 roku wraz z *perceptronem*

– przyjrzymy mu się wkrótce. Dziś uczenie się maszyn dokonało olbrzymich postępów, po części za sprawą dwóch trendów. Po pierwsze, postęp technologiczny astronomicznie zwiększył ilość dostępnej pamięci, jednocześnie drastycznie obniżają jej cenę. Dał nam również szybsze procesory i sieci komputerowe. Po drugie, Internet ułatwił gromadzenie ogromnej ilości danych, a ludzie nie mogą się powstrzymać przed grzebaniem w nich. Na przykład dane z projektu skanowania książek i tłumaczenia książek jednej z dużych firm zostały wykorzystane do znacznej poprawy innego projektu translatorskiego. Innym przykładem są dane z projektów kartograficznych używane przez samochody bezzałogowe. Wyniki tych i innych projektów były na tyle przekonujące, że uczenie się maszyn stosuje się teraz w ogromnej liczbie rozwiązań. Ludzie zorientowali się też, że te same dwa trendy – tańsze nośniki i więcej mocy obliczeniowej oraz gromadzenie ogromnych ilości danych – napędzają rozwój sztucznej inteligencji. W rezultacie dzisiaj przeżywamy kolejny rozkwit tej dziedziny.

Pośpiech, z jakim wprowadza się aktualnie sztuczną inteligencję, przypomina niestety podejście „Cóż złego mogłoby się stać?", które przenika świat bezpieczeństwa komputerowego. Nadal nie wiemy jeszcze, jak uniknąć zbudowania psychotycznych systemów przypominających HAL z filmu *2001: Odyseja kosmiczna* z 1968 roku.

Ogólny zarys

Sądzę, że większość z was zorientowała się już do tej pory, że programowanie to żmudna praca niezbędna do implementacji rozwiązań stawianych przed nami problemów. Zdefiniowanie problemu i jego rozwiązania jest znacznie bardziej interesującą i trudniejszą częścią pracy. Wielu wolałoby poświęcić całość swojej kariery na próby zaprzęgnięcia komputerów do programowania, zamiast robić to samemu (jeszcze jeden przykład „dziwnego lenistwa inżynierów", o którym wspominaliśmy w rozdziale 5).

Jak wcześniej wspomniano, wszystko zaczęło się od sztucznej inteligencji – uczenie się maszyn i wielkie dane przyszły później. Chociaż termin ten został ukuty dopiero w 1956 roku na warsztatach w Dartmouth, samo pojęcie sztucznej inteligencji sięga korzeniami mitów greckich. Wielu filozofów i matematyków od tych czasów pracowało nad stworzeniem systemu formalnego, który mógłby skodyfikować myśli ludzkie. Chociaż nie doprowadziło to prawdziwej sztucznej inteligencji, przygotowało od nią grunt. Na przykład George Boole, który w rozdziale 1 dał nam swoją algebrę, opublikował w 1854 roku książkę *Badanie praw myślenia, na których oparte są matematyczne teorie logiki i prawdopodobieństwa*.

Do tej pory zgromadziliśmy mnóstwo informacji na temat ludzkich procesów decyzyjnych, ale nadal tak naprawdę nie wiemy, jak ludzie myślą. Na przykład czytelnicy i ich znajomi zapewne potrafią odróżniać koty od pasztetów. Każdy z nas jednak mógłby osiągać pożądany efekt różnymi drogami. Sam fakt, że dla tych samych danych osiągamy te same rezultaty, wcale nie oznacza jeszcze, że robimy to w ten sam sposób. Znamy dane wejściowe i wyjściowe, poznaliśmy sporo sprzętu, ale nadal nie mamy za bardzo pojęcia o „oprogramowaniu", które zamienia te pierwsze w drugie. Wynika z tego, że

sposób, w jaki komputer odróżnia koty od pasztetów, także zapewne będzie inny od naszego.

Jedną z rzeczy, jakie wiemy o ludzkich procesach myślowych jest to, że podświadomie jesteśmy bardzo dobrzy ze statystyki – co niestety nie jest tym samym, co świadomie przetrwać zajęcia z „sadystyki" na studiach. Lingwiści na przykład badali ludzki sposób nauki języka. Niemowlęta przeprowadzają ogromne ilości analizy statystycznej jako część procesu nauki języka – odróżniają istotne dźwięki od dźwięków otoczenia, grupują je w fonemy, a te następnie grupują w słowa i zdania. To, czy ludzie dysponują w tym celu wyspecjalizowaną maszynerią, czy może używają maszyn obliczeniowych o przeznaczeniu ogólnym, to nadal przedmiot otwartej debaty.

To, że niemowlęta mogą uczyć się języka dzięki analizie statystycznej, jest możliwe po części również przez to, że dysponują one ogromnymi ilościami danych do analizy. Z wyjątkiem bardzo szczególnych okoliczności, niemowlęta są wystawione na działanie ciągłego strumienia dźwięków. Podobnie jest z nieustannym potokiem danych wizualnych i innych danych zmysłowych. Niemowlęta uczą się przez analizę wielkich ilości danych, innymi słowy, dzięki *big data*.

Ogromny wzrost mocy obliczeniowej, pojemności nośników pamięci i liczby połączeń sieciowych między różnymi komputerami (wliczając w to telefony komórkowe) doprowadził do zgromadzenia astronomicznych ilości danych. I to nie tylko przez typków spod ciemnej gwiazdy z poprzedniego rozdziału. Część tych danych jest zorganizowana, a część nie. Przykładem danych zorganizowanych mogłyby być wyniki pomiarów wysokości fal zbierane z boi przybrzeżnych. Przykładem danych niezorganizowanych mogłyby być dźwięki otoczenia. Co my z tym wszystkim chcemy zrobić?

Statystyka jest jedną z dobrze zdefiniowanych gałęzi matematyki, co oznacza, że można napisać program wykonujący analizę statystyczną. Możemy też napisać program, który moglibyśmy trenować z użyciem danych. Jednym ze sposobów wdrożenia filtrów antyspamowych (którym przyjrzymy się wkrótce szczegółowo) jest wprowadzenie do analizatora statystycznego dużej liczby maili wraz ze wskazaniem, które z nich są spamem, a które nie. Innymi słowy dajemy mu zorganizowane dane i mówimy programowi, co te dane znaczą (w tym przypadku, co jest spamem, a co nie). Podejście takie nazywamy *uczeniem maszynowym* (*machine learning*, ML). Trochę tak jak Huckleberry Finn, „nauczymy *się* tej maszyny" – zamiast uczyć *ją*.

Pod wieloma względami systemy uczenia się maszyn są analogiczne do funkcji ludzkiego autonomicznego układu nerwowego. U ludzi mózg nie jest aktywnie zaangażowany w większość funkcji niższego rzędu, takich jak oddychanie – jest zarezerwowany dla funkcji wyższego rzędu, jak na przykład wykombinowanie, co dzisiaj będzie na obiad. Funkcje niższego rzędu są załatwiane przez autonomiczny system nerwowy, a ten angażuje mózg tylko wtedy, gdy wydarzy się coś wymagającego jego uwagi. Uczenie się maszyn aktualnie świetnie nadaje się do rozpoznawania różnych rzeczy, ale to nie to samo co podejmowanie działań.

Dane niezorganizowane to zupełnie inna para kaloszy. Mówimy tutaj o wielkich danych, co oznacza, że nie jest to coś, co z czym ludzie mogliby się zapoznać bez wspomagania maszynowego. W takim przypadku używa

się różnych statycznych technik, by znaleźć powtarzające się wzorce i relacje. Takiego podejścia – zwanego niekiedy *data miningiem* (dosł. wydobywaniem danych) – można użyć do wydobycia muzyki z dźwięków otoczenia (bądź co bądź, jak zauważył francuski kompozytor Edgard Varèse, muzyka to zaaranżowany dźwięk).

Wszystko to zasadniczo polega na znalezieniu sposobu na przekształcenie skomplikowanych danych w coś prostszego. Na przykład można wytrenować system uczenia się maszyn tak, by odróżniał koty od pasztetów. Taki system przekształca złożone dane obrazków na proste bity kotów i pasztetów. Ten proces nazywamy *klasyfikacją*.

W rozdziale 5 omawialiśmy oddzielenie instrukcji od danych. W rozdziale 13, ze względu na bezpieczeństwo, ostrzegałem przed pozwoleniem, by dane były traktowane jak instrukcje. Są jednak sytuacje, w których to ma sens, ponieważ klasyfikacja na podstawie danych ma swoje ograniczenia.

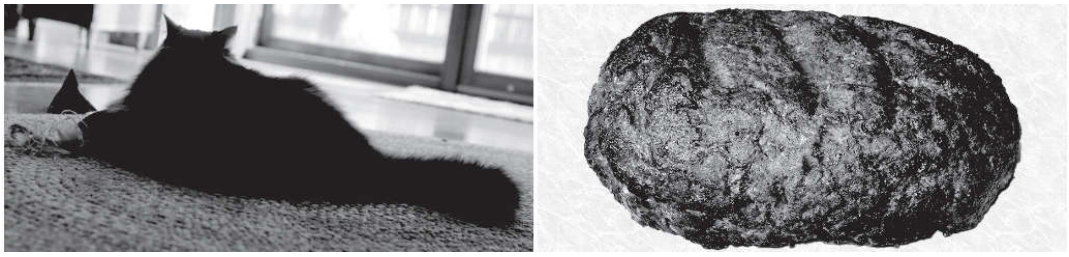
Na podstawie jedynie klasyfikacji nie można na przykład zbudować samochodu bezzałogowego. Na dane wyjściowe klasyfikatorów musi być nałożony zestaw skomplikowanych instrukcji implementujących pewne zachowania, na przykład „Nie przejeżdżaj kotów, ale przejechanie pasztetu jest nie tylko dozwolone, czasami jest wręcz korzystne dla społeczeństwa". Istnieje wiele sposobów, na jakie można wdrożyć takie zachowania, łącznie z kombinacją skrętów i zmian prędkości. Istnieje ogromna ilość zmiennych, które trzeba wziąć pod uwagę – ruch na drodze, położenie przeszkód względem samochodu i tak dalej.

Ludzie nie uczą się prowadzić na podstawie skomplikowanego zestawu szczegółowych instrukcji, takich jak „obróć kierownicę o jeden stopień w lewo i naciśnij pedał hamulca z siłą jednego grama przez trzy sekundy, by ominąć kota" albo „skręć trzy stopnie w prawo i gaz do dechy, aż przejedziesz pasztet". Zamiast tego pracują z celami takimi jak „nie przejeżdżaj kotów". Ludzie „programują" sami siebie i jak wspomnieliśmy wcześniej, nie mamy sposobu na poznanie szczegółów tego programowania, aby określić, w jaki sposób wybieramy drogę do celu. Jeśli poobserwuje się przez parę chwil ruch uliczny, łatwo dojść do wniosku, że istnieje wiele sposobów na osiągnięcie tych samych podstawowych celów.

Ludzie w takich przypadkach nie tylko dostosowują swoje klasyfikatory; ludzie piszą nowe programy. Gdy robią to komputery, mówimy o *sztucznej inteligencji* (*artificial intelligence,* AI). Systemy AI piszą swoje własne programy w celu osiągnięcia zamierzonych celów. Jednym ze sposobów na osiągnięcie tego bez konieczności poświęcania żadnych prawdziwych kotów jest zaopatrzenie systemów AI w symulowane dane wejściowe. Filozofujące termostaty, takie jak Bomba 20 z filmu *Dark Star* z 1974 roku, pokazują nam jednak, że to nie zawsze świetny pomysł.

Ogromną różnicą między uczeniem maszynowym a sztuczną inteligencją jest możliwość przejrzenia systemu i „zrozumienia" wykonywanego „procesu myślowego". Jest to aktualnie niemożliwe w przypadku systemów uczenia się maszyn, ale całkiem możliwe w przypadku AI. Nie jest pewne, czy ta sytuacja się utrzyma w miarę jak systemy AI będą się powiększać, zwłaszcza że nie wydaje się prawdopodobne, by zachodzące w takich systemach procesy przypominały pod jakimś względem myśli ludzkie.

Zobaczmy, czy uda nam się wymyślić sposób na odróżnienie fotografii kota od fotografii pasztetu. Fotografia zawiera o wiele mniej informacji niż to, co jest dostępne dla ludzi, jeśli chcą odróżniać prawdziwe koty od prawdziwych pasztetów. Na przykład ludzie odkryli, że koty zazwyczaj uciekają, gdy się je goni, podczas gdy zachowanie to nie jest powszechne wśród pasztetów. Spróbujemy stworzyć proces, który po okazaniu mu fotografii powie nam, czy „widzi" kota, czy pasztet, czy może żadne z powyższych. Rysunek 13.1 przedstawia oryginalne zdjęcia pasztetopodobnego Antoniego Kota (po lewej stronie) oraz prawdziwego pasztetu (po prawej).



Rysunek 14.1. Antoni Kot i pasztet

Podczas obcowania ze sztuczną inteligencją na każdym poziomie prawdopodobieństwo natknięcia się na statystykę jest wysokie. Zaczniemy więc od powtórki z jej podstaw.

Bayes

Angielski duchowny Thomas Bayes (1701–1761) musiał być zapewne zaniepokojony szansami swojej trzódki na dostanie się do nieba, ponieważ dużo czasu poświęcał na rozważania na temat statystyki. Bayes był w szczególności zainteresowany tym, jak łączą się ze sobą prawdopodobieństwa poszczególnych wydarzeń. Jeśli ktoś jest na przykład graczem w trik-traka, zapewne wiele wie o rozkładzie wyników rzutu dwoma sześciennymi kośćmi. Bayes jest odpowiedzialny za nazwane na jego pamiątkę *twierdzenie Bayesa*.

Tą częścią wiekopomnej spuścizny Bayesa, jaka nas tu będzie interesować, nazywa się *naiwnym klasyfikatorem bayesowskim*. Zostawiwszy na chwilę na boku kota wraz z jego pasztetem, spróbujmy odróżnić wiadomości będące spamem od wiadomości, które nimi nie są. Wiadomości to zbiory słów. Pewne słowa mają większe prawdopodobieństwo pojawienia się w spamie, z czego możemy wywnioskować, że wiadomości pozbawione tych słów mogą spamem nie być.

Zacznijmy od zebrania jakichś prostych statystyk. Załóżmy, że dysponujemy jakąś reprezentatywną próbką wiadomości, z których 100 jest spamem, a 100 nie jest. Podzielimy wiadomości na słowa i policzymy, w ilu wiadomościach dane słowo występuje. Ponieważ mamy po 100 wiadomości każdego typu, nie musimy w ogóle przeliczać procentów. Częściowe wyniki widzimy w tabeli 14.1.

Tabela 14.1. Statystyki wyrazów w wiadomościach

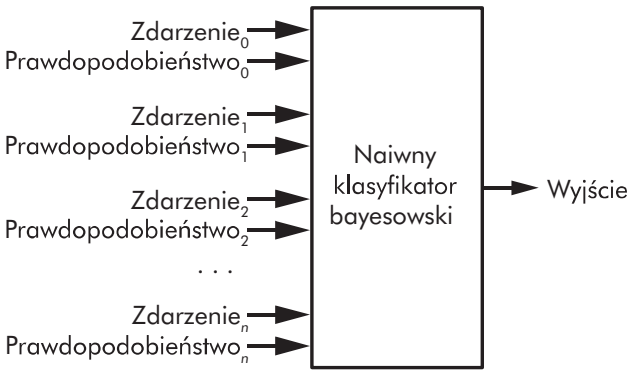
Wyraz	Spam %	Nie-spam %
Pasztet	80	0
Hamburger	75	5
Kocimiętka	0	70
Cebula	68	0
Myszki	1	67
I	99	98
Z	97	99

Widać, że niektóre słowa są powszechnie spotykane zarówno w spamie, jak i w nie-spamie. Zastosujmy naszą tabelę do wiadomości zawierającej frazę „hamburger z cebulą": odpowiednio procenty spamu wynoszą 75, 97 i 68, a nie-spamu 5, 97 i 0. Ile wynosi prawdopodobieństwo, że ta wiadomość jest lub nie jest spamem?

Twierdzenie Bayesa mówi nam, żebyśmy połączyli prawdopodobieństwa (p), gdzie p_0 to prawdopodobieństwo tego, że wiadomość zawierająca słowo *pasztet* jest spamem, p_1 to prawdopodobieństwo, że wiadomość zawierająca słowo *hamburger* jest spamem, i tak dalej:

$$p_{\text{combined}} = \frac{p_0 p_1 p_2 \ldots p_n}{p_0 p_1 p_2 \ldots p_n + (1-p_0)(1-p_1)(1-p_2)\ldots(1-p_n)}$$

Można to zwizualizować jak na rysunku 14.2. Zdarzenia wraz z prawdopodobieństwami podobnymi do tych z tabeli 14.1 wchodzą do klasyfikatora, który z kolei zwraca prawdopodobieństwo, że zdarzenia opisują to, o co nam chodzi.



Rysunek 14.2. Naiwny klasyfikator bayesowski

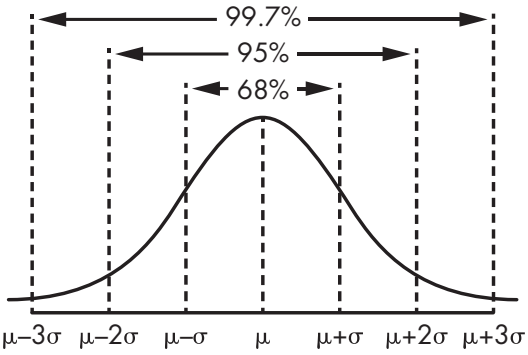
Możemy teraz zbudować dwa klasyfikatory, jeden dla spamu, a drugi dla nie-spamu. Jeśli wrzucimy do nich liczby z powyższego przykładu, okaże się, że wiadomość ma 99,64% szans na bycie spamem i 0% szans, że nim nie jest.

Widać więc, że ta technika działa całkiem dobrze. Niech żyje statystyka! Potrzeba oczywiście jeszcze wielu innych sztuczek, by zbudować przyzwoity

filtr przeciwspamowy. Warto też wiedzieć, co się tu rozumie przez „naiwny". Oczywiście nie to, jakoby Bayes nie wiedział, co robi. Znaczy to, że zupełnie tak jak w przypadku rzutu dwiema kostkami, wszystkie zdarzenia rozpatrywane są jako zdarzenia niezależne. Moglibyśmy ulepszyć nasze filtrowanie spamu przez przyjrzenie się zależnościom między słowami – na przykład moglibyśmy wziąć pod uwagę to, że „i i" pojawia się jedynie w wiadomościach dotyczących algebry boolowskiej. Wielu spamerów stara się ominąć filtry zawierając w swych wiadomościach duże ilości „słownej sałatki", rzadko kiedy poprawnej gramatycznie.

Gauss

Niemiecki matematyk Johann Carl Friedrich Gauss (1777–1855) jest następną statystycznie istotną osobą. Można go obarczyć odpowiedzialnością za *krzywą dzwonową*, zwaną również *rozkładem normalnym* albo *rozkładem Gaussa*. Wygląda tak, jak widać na rysunku 14.3.



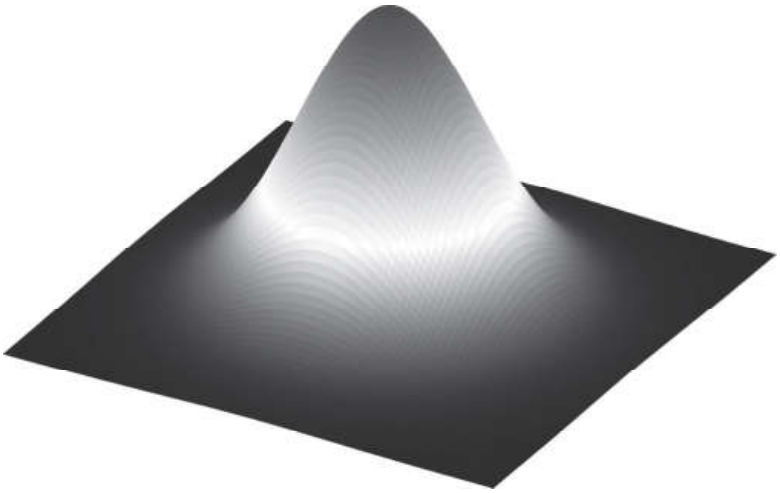
Rysunek 14.3. Krzywa dzwonowa

Krzywa dzwonowa jest bardzo interesująca, ponieważ pasuje do niej wiele zaobserwowanych zjawisk. Jeśli na przykład zmierzymy wysokość każdego koszykarza w parku i określimy średnią wysokość μ, to część graczy będzie niższa, a część wyższa niż μ. (przy okazji, μ czyta się „mi", co czyni z niej najlepszą grecką literę dla kotów). Z wszystkich graczy 68% będzie w granicach *odchylenia standardowego* oznaczanego σ, 95% w granicach dwóch odchyleń standardowych itd. Bardziej dokładne byłoby stwierdzenie, że rozkład wysokości koszykarzy dąży do krzywej dzwonowej w miarę jak rośnie liczba pomiarów – wzrost jednego koszykarza wiele nam nie powie. Ostrożnie próbkowane dane z dobrze zdefiniowanej populacji mogą być użyte do przyjęcia hipotez na temat całości populacji.

Chociaż to już samo w sobie jest bardzo ciekawe, istnieje również wiele innych zastosowań krzywych dzwonowych. Niektórych z nich możemy użyć do rozwiązania naszego problemu z pasztetem. Amerykański rysownik i animator Bernard Kliban (1935–1990) uczy nas, że kot to zasadniczo pasztet z uszami i ogonem. Wynika z tego, że gdybyśmy potrafili wyodrębniać (*extract*) ze zdjęć takie cechy jak posiadanie ogona i uszu i bycie pasztetem, moglibyśmy następnie wrzucić je do klasyfikatora, by odpowiednio zaklasyfikował nasze dane.

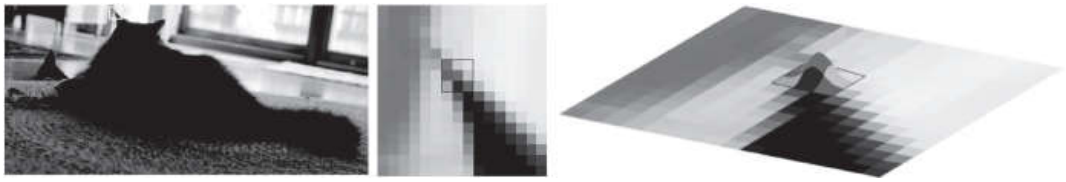
Możemy uczynić cechy przedmiotów nieco łatwiejszymi do rozpoznania, jeśli możemy wykreślić ich kontury. Oczywiście nie możemy tego zrobić bez znalezienia granic. To jest nieco skomplikowane, ponieważ koty (a także spory podzbiór pasztetów) są porośnięte włoskami. Koty w szczególności mają sporo wyróżniających się spośród reszty włosów, które same tworzą swoje kontury, ale nie o takie krawędzie nam chodzi. Choć to wydaje się przeczyć naszym intuicjom, pierwszym krokiem powinno być lekkie rozmazanie obrazu, żeby pozbyć się tych wszystkich niechcianych szczegółów. Rozmywanie obrazu polega na przepuszczeniu go przez filtr dolnoprzepustowy, podobnie jak to widzieliśmy na przykładzie dźwięku w rozdziale 6. Drobne detale na obrazku to odpowiednik „wysokich częstotliwości". Można to zapamiętać, wyobrażając sobie, jak drobne szczegóły zmieniają się szybciej, gdy wodzimy wzrokiem po obrazie.

Zobaczmy, co Gauss może dla nas zrobić. Weźmy krzywą z rysunku 14.3 i obróćmy ją dookoła μ. Otrzymujemy w ten sposób jej trójwymiarową wersję widoczną na rysunku 14.4.



Rysunek 14.4. Trójwymiarowy rozkład gaussowski

Przeciągniemy ten kształt po obrazie, skierowując μ nad każdy piksel po kolei. Można sobie wyobrazić, jak poszczególne części krzywej pokrywają piksele dookoła piksela centralnego. Wygenerujemy nową wartość dla każdego piksela, mnożąc wartości pikseli pod krzywą razy wartość krzywej w tym miejscu, a następnie dodając wyniki. Technikę tę nazywamy *rozmyciem gaussowskim*. Na rysunku 14.5 widzimy, jak to działa. Obrazek na środku jest powiększonym fragmentem obwiedzionym prostokątem na obrazku po lewej. Po prawej stronie widzimy, jak rozmycie gaussowskie waży zbiór pikseli odległych o określone wartości od piksela centralnego.



Rysunek 14.5. Rozmycie gaussowskie

Proces łączenia wartości piksela z wartością jego sąsiadów może się nam wydawać nieco zawiły i rzeczywiście w matematyce nazywamy podobne zabiegi *splotem*. Tablica wag jest znana jako *macierz splotu* lub *kernel*. Spójrzmy na niektóre przykłady.

Rysunek 14.6 pokazuje nam macierze 3×3 oraz 5×5. Zauważmy, że wagi dodają się do 1 – w ten sposób zachowujemy oryginalną jasność.

3×3

$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$
$\frac{2}{16}$	$\frac{4}{16}$	$\frac{2}{16}$
$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$

5×5

$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{6}{256}$	$\frac{24}{256}$	$\frac{36}{256}$	$\frac{24}{256}$	$\frac{6}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$

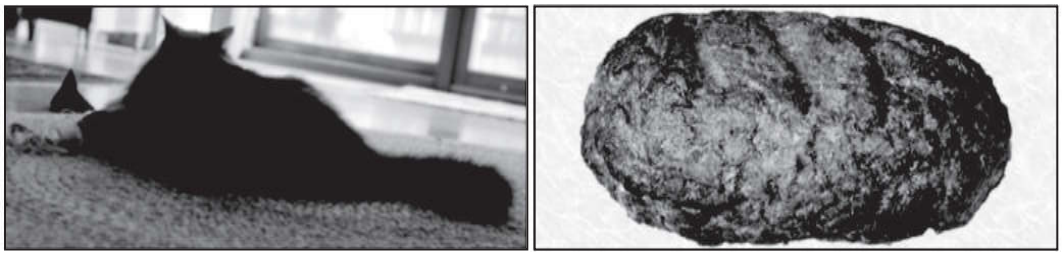
Rysunek 14.6. Gaussowskie macierze splotu

Na rysunku 14.7 oryginalny obraz znajduje się po lewej stronie. Nasze oczy potrafią prześledzić kontury drzewa, mimo że jest na nim wiele luk. Obrazek pośrodku przedstawia rezultat rozmycia gaussowskiego z macierzą 3×3. Choć jest nieco bardziej rozmazany, krawędzie są łatwiejsze do prześledzenia. Obrazek po prawej prezentuje rezultaty rozmycia gaussowskiego z macierzą 5×5.

Można myśleć o obrazku, jakby był on funkcją matematyczną o postaci *jasność* = $f(x, y)$. Wartość tej funkcji stanowi jasność piksela w miejscu oznaczonym współrzędnymi x i y. Zauważmy, że jest to funkcja *dyskretna*, więc wartości x i y muszą być całkowite. Muszą się też oczywiście znajdować w granicach obrazka. W podobny sposób można myśleć o macierzy splotu jako o malutkim obrazku, dla którego wartość pikseli wynosi *waga* = $g(x, y)$. W ten sposób proces wykonywania splotu polega na przejściu przez sąsiednie piksele pokryte przez macierz, mnożeniu ich wartości przez wartość odpowiedniego „piksela" macierzy, a następnie zsumowaniu wszystkich wartości.

Rysunek 14.7. Przykłady rozmycia gaussowskiego

Rysunek 14.8 pokazuje nasze oryginalne zdjęcia rozmyte przy użyciu macierzy gaussowskiej 5×5.



Rysunek 14.8. Rozmyty kot i rozmyty pasztet

Zauważmy, że ponieważ macierze splotu są większe niż jeden piksel, wystają poza brzegi rysunku w trakcie obróbki skrajnych pikseli. Istnieje wiele podejść radzących sobie z tym problemem, takie jak np. niepodchodzenie za blisko brzegu (co czyni wyjściowy obraz ciut mniejszym) lub nieznaczne powiększenie obrazka przez obrysowanie go ramką.

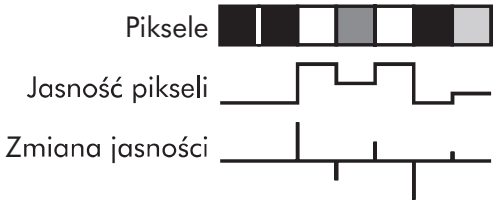
Sobel

Na rysunku 14.1 znajduje się całe mnóstwo informacji, które tak naprawdę nie są niezbędne do osiągnięcia celu – na przykład kolor. Scott McCloud w swojej książce *Understanding Comics: The Invisible Art* (Tundra) udowadnia nam, że możemy rozpoznać twarz na podstawie kółka, dwóch kropek i linii – reszta detali jest niepotrzebna i można ją zignorować. W podobny sposób uprościmy nasze zdjęcia.

Spróbujmy teraz znaleźć krawędzie, skoro ułatwiliśmy sobie to zadanie dzięki rozmyciu obrazu. Istnieje wiele definicji *krawędzi*. Nasze oczy są najbardziej wrażliwe na zmiany jasności, więc tego będziemy się trzymać. Zmiana jasności to po prostu różnica w jasności między danym pikselem a pikselem sąsiadującym.

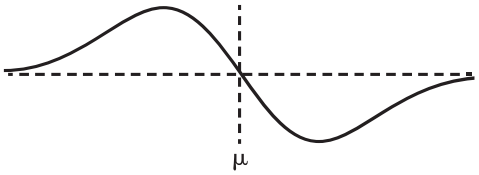
Mniej więcej połowa rachunki różniczkowego mówi o zmianach, więc możemy to tutaj zastosować. *Pochodna* funkcji to nachylenie krzywej będącej wykresem tej funkcji. Jeśli zależy nam na zmianie jasności między pikselami, to właściwym wzorem po prostu $jasność = f(x+1, y) - f(x, y)$.

Przyjrzymy się poziomemu rzędowi pikseli na rysunku 14.9. Poziom jasności jest wykreślony pod nim, a jeszcze niżej znajduje się zmiana jasności. Widać, że wygląda to dość kolczasto. To dlatego, że zmiana przyjmuje niezerową wartość tylko w trakcie zmiany jednego piksela na drugi.



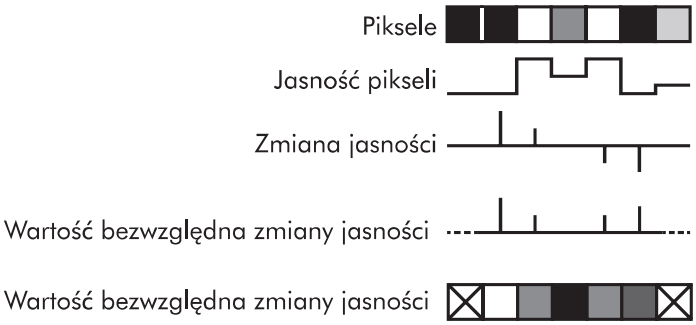
Rysunek 14.9. Krawędzie to zmiany jasności

Jest pewien problem z mierzeniem jasności w ten sposób – zmiana zachodzi tylko na granicy pikseli. Chcemy, by zmiana zachodziła pośrodku piksela. Zobaczmy, czy Gauss może nam tu znowu pomóc. Podczas dokonywania rozmycia centrowaliśmy μ na pikselu. Przyjmiemy to samo podejście, ale zamiast używać krzywej dzwonowej, weźmiemy jej pierwszą pochodną, pokazaną na rysunku 14.10, na którym widzimy wykres nachylenia krzywej z rysunku 14.3.



Rysunek 14.10. Nachylenie krzywej gaussowskiej z rysunku 14.3

Jeśli oznaczymy dodatnie i ujemne nachylenia jako +1 i –1 i umieścimy je centralnie nad sąsiednimi pikselami, zmianą jasności dla piksela będzie $\Delta\text{jasność}_n = 1 \times \text{piksel}_{n-1} \times \text{piksel}_{n+1}$. Rezultat widzimy na rysunku 14.11.

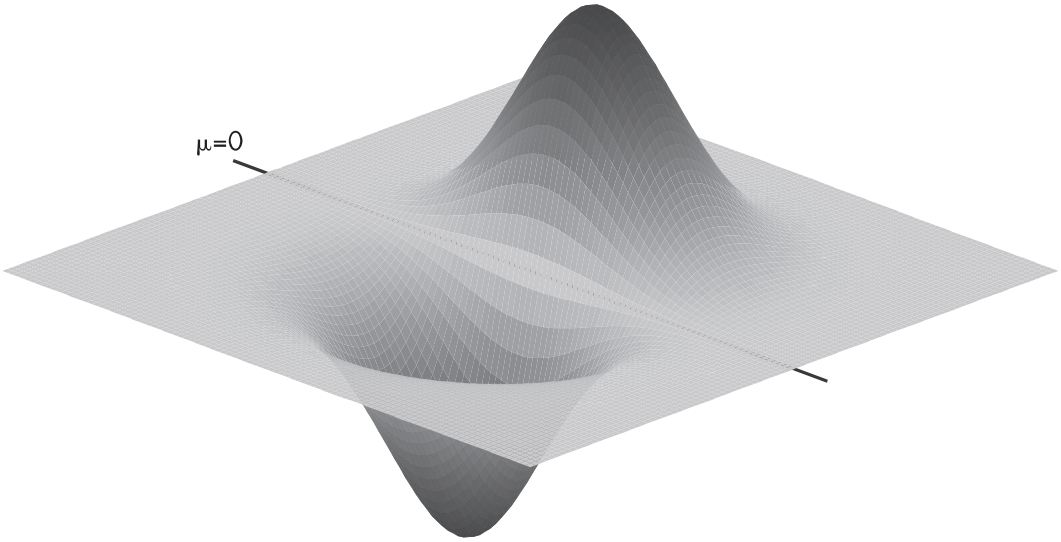


Rysunek 14.11. Zmiana jasności wycentrowana na pikselach

Mamy tu oczywiście ten sam problem z pikselami na brzegach co poprzednio, więc w przykładzie nie mamy wartości dla skrajnych pikseli. W tym momencie nie obchodzi nas kierunek zmiany, więc obliczamy *wartość bezwzględną* zmiany.

Wykrywanie konturów w ten sposób działa dość dobrze, ale wielu ludzi próbowało udoskonalić ten proces. Jedno z najlepszych podejść, *operator Sobela* (*Sobel operator*) został opublikowany przez amerykańskiego naukowca Irwina Sobela w artykule z 1968 roku, którego współautorem był Gary Feldman.

Podobnie jak to miało miejsce w przypadku naszego kernela rozmycia gaussowskiego, generujemy kernel Sobela wykrywający krawędzie na podstawie nachylenia krzywej gaussowskiej. Dwuwymiarową wersję widzieliśmy na rysunku 14.10, rysunek 14.12 przedstawia zaś wersję trójwymiarową.



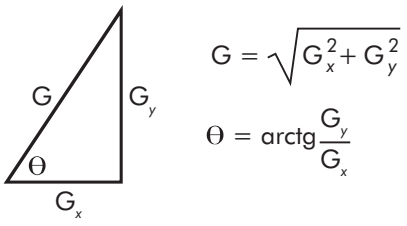
Rysunek 14.12. Trójwymiarowe nachylenie stoku krzywej Gaussa z rysunku 14.10

Ponieważ ten kształt nie jest symetryczny w stosunku do obu osi, Sobel użył dwóch wersji – jednej dla kierunku poziomego, a drugiej dla kierunku pionowego. Rysunek 14.13 prezentuje nam oba kernele.

$Sobel_x$			$Sobel_y$		
+1	0	–1	+1	+2	+1
+2	0	–2	0	0	0
+1	0	–1	–1	–2	–1

Rysunek 14.13. Kernele Sobela

Zastosowanie tych kerneli generuje nam dla każdego piksela parę *gradientów*, G_x i G_y; gradient można traktować jako nachylenie. Skoro mamy gradient w każdym kartezjańskim kierunku, możemy użyć trygonometrii i przekształcić je we współrzędne biegunowe, podając *wielkość* G i *kierunek* θ, tak jak to pokazano na rysunku 14.14.



$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \operatorname{arctg}\frac{G_y}{G_x}$$

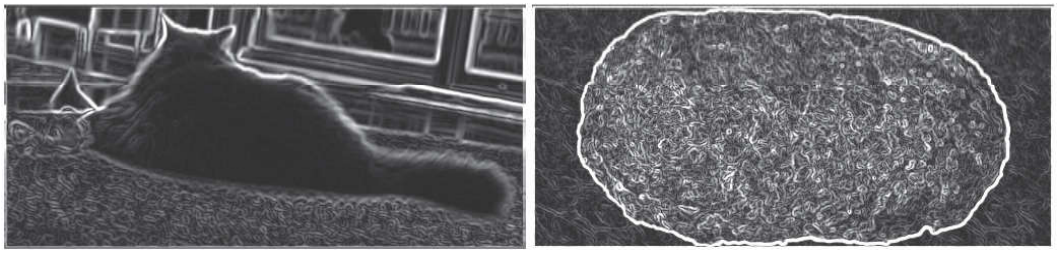
Rysunek 14.14. Wielkość i kierunek gradientu

Wielkość gradientu mówi nam, jak „ostrą" krawędzią dysponujemy, a kierunek daje nam jego położenie. Pamiętajmy, że kierunek jest prostopadły do obiektu – pozioma krawędź ma pionowy gradient.

Można zauważyć, że obliczenie wielkości i kierunku gradientu to nic innego, jak tylko przekształcenie ze współrzędnych kartezjańskich we współrzędne biegunowe, jak to widzieliśmy w rozdziale 11. Zmiana systemu współrzędnych

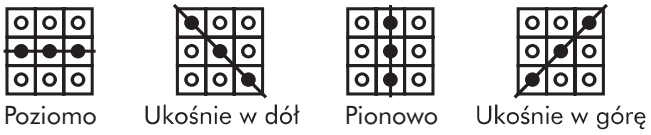
to przydatna sztuczka. W tym przypadku, gdy już jesteśmy we współrzędnych biegunowych, nie musimy się martwić dzieleniem przez zero albo ogromnymi liczbami otrzymywanymi, gdy mianowniki są małe. Większość bibliotek matematycznych zawiera funkcje w rodzaju `atan2(y,x)`, które obliczają arcus tangens bez przeprowadzania dzielenia.

Rysunek 14.15 pokazuje nam wielkości gradientów dla obu obrazków



Rysunek 14.15. Wielkości Sobela dla rozmytego kota i pasztetu

Z kierunkiem mamy dodatkowy problem, bo zawiera on więcej informacji, niż możemy wykorzystać. Spójrzmy na rysunek 14.16.

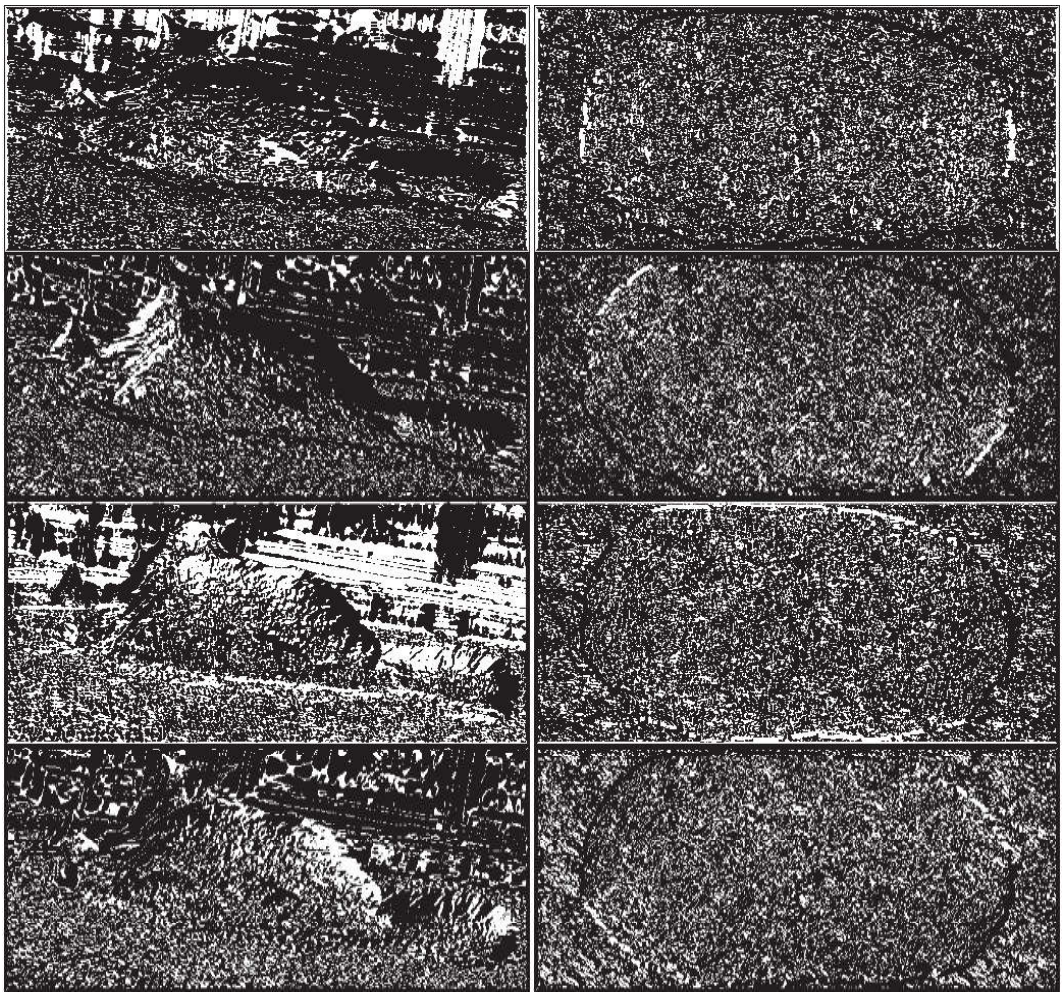


Rysunek 14.16. Piksel i jego sąsiedzi

Jak widać, ponieważ jeden piksel ma tylko ośmiu sąsiadów, istnieją tak naprawdę tylko cztery interesujące nas kierunki. Rysunek 14.17 pokazuje, jak wszystkie kierunku są kwantyfikowane w cztery „skrzynki”.

Rysunek 14.17. Skrzynki dla kierunku gradientu

Rysunek 14.18 przedstawia rozłożone na skrzynki kierunki Sobela dla rozmytych obrazów. Można zauważyć związek między kierunkami a wielkościami. Najwyższy rząd to skrzynka pozioma, następnie mamy skrzynkę skośnie w górę, skrzynkę pionową i skrzynkę skośnie w dół.



Rysunek 14.18. Kierunki Sobela dla rozmytego kota i pasztetu

Jak zatem widać, operator Sobela znajduje krawędzie, jednak nie są to świetne krawędzie. Są grube, przez co łatwo je pomylić z cechami obiektu. Cienkie krawędzie byłyby znacznie lepsze i możemy użyć kierunków Sobela, by je znaleźć.

Canny

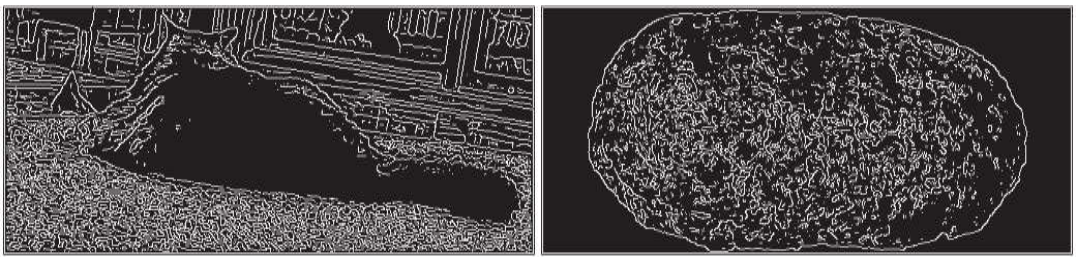
W 1968 roku australijski informatyk John Canny ulepszył wykrywanie krawędzi przez dodanie kilku dodatkowych kroków do wyników Sobela. Pierwszym z nich jest *usuwanie niemaksymalnych pikseli* (*nonmaximum suppression*). Patrząc na rysunek 14.15, można zauważyć, że niektóre krawędzie są grube i rozmazane. Łatwiej będzie później rozpoznać cechy obiektu, jeśli krawędzie będą cieńsze. Usuwanie niemaksymalnych pikseli jest techniką pozwalającą odchudzić krawędzie.

Oto plan. Porównujemy wielkość gradientu każdego piksela z wielkościami gradientu pikseli, które z nim sąsiadują w kierunku tego gradientu. Jeśli jego wielkość jest wyższa niż sąsiadów, zostawiamy tę wartość. W przeciwnym wypadku ustawiamy ją na 0.

Rysunek 14.19. Usuwanie niemaksymalnych pikseli

Widzimy na rysunku 14.19, że środkowy piksel na obrazku po lewej stronie jest pozostawiony, ponieważ ma wyższą wielkość (tzn. jest jaśniejszy) niż jego sąsiedzi, podczas gdy środkowy piksel po prawej stronie jest usuwany (tzn. ustawiamy na 0), ponieważ jego sąsiedzi mają wyższą wielkość.

Rysunek 14.20 pokazuje nam, w jaki sposób usunięcie niemaksymalnych pikseli odchudza krawędzie wygenerowane przez operator Sobela.



Rysunek 14.20. Rezultat usunięcia niemaksymalnych pikseli kota i pasztetu

To już zaczyna wyglądać całkiem dobrze, choć może być dobrym powodem unikania pasztetów. Usunięcie niemaksymalnych pikseli znalazło wiele krawędzi na naszych obrazkach. Jeśli spojrzymy znów na rysunek 14.15, widzimy się, że wiele z tych krawędzi ma niskie wielkości gradientów. Ostatnim krokiem algorytmu Canny'ego jest *progowanie krawędzi z histerezą* (*edge tracking with hysteresis*), dzięki czemu usuniemy „słabe" krawędzie i zostawimy tylko te „silne".

W rozdziale 2 dowiedzieliśmy się, że histereza polega na porównaniu z parą progów. Przeskanujemy wyniki z usuwania niemaksymalnych pikseli w poszukiwaniu pikseli krawędzi (białe na rysunku 14.20), których wielkość jest wyższa od górnego progu. Kiedy taki znajdziemy, zostaje on ostatecznym pikselem krawędzi. Patrzymy potem na jego sąsiadów. Każdy sąsiad, którego wielkość gradientu jest wyższa od dolnego progu, również zostaje zaznaczony jako ostateczny piksel krawędzi. Idziemy każdą z tych ścieżek, używając rekursji, aż znajdziemy wielkość gradientu o wartości niższej od dolnego progu. To trochę tak, jakbyśmy startowali na wyraźnej krawędzi, a potem śledzili jej połączenia, aż powoli znikną. Wyniki widać na rysunku 14.21. Silne krawędzie są białe; a odrzucone, słabsze krawędzie stały się szare.

Rysunek 14.21. Progowanie krawędzi z histerezą

Widać, że wiele krawędzi obecnych po usunięciu pikseli niemaksymalnych teraz zniknęło, a kontury obiektów są bardzo dobrze widoczne.

Istnieje świetna biblioteka typu open source przeznaczona do rozpoznawania obrazów, o nazwie *OpenCV*. Można jej użyć do zabawy z przetwarzaniem obrazów, wraz z tym wszystkim, co robiliśmy w tym rozdziale.

Ekstrakcja cech

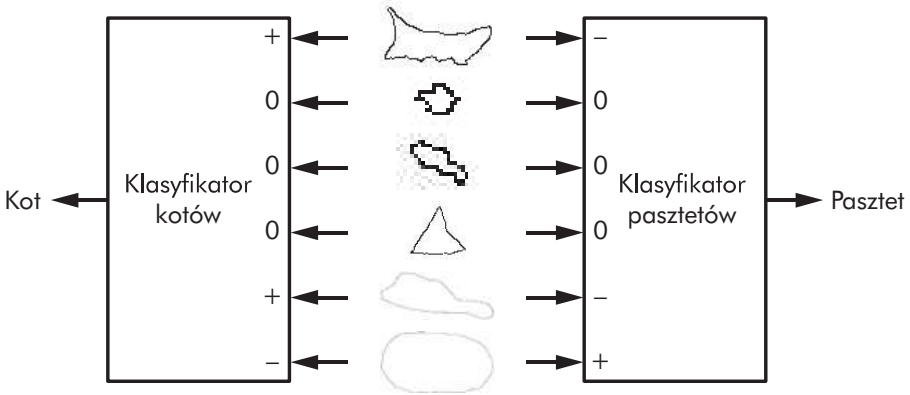
Następny krok jest łatwy dla ludzi, ale trudny dla komputerów. Z obrazków widocznych na rysunku 14.21 chcemy wybrać cechy. Nie będziemy omawiać tego procesu ze wszystkimi szczegółami, ponieważ zawiera on wiele matematyki, z którą większość z was zapewne wcześniej się nie zetknęła. Omówimy jednak podstawy.

Istnieje bardzo wiele algorytmów służących do wykrywania cech w obrazach. Niektóre, jak *transformacja Hougha*, są dobre do wykrywania kształtów geometrycznych, takich jak linie czy okręgi. To nie będzie dla nas zbyt przydatne, ponieważ nie szukamy kształtów geometrycznych. Zróbmy coś prostego. Przeskanujemy nasze obrazki w poszukiwaniu krawędzi, a gdy już jakąś znajdziemy, idziemy za nią, w ten sposób wybierając obiekt. Jeśli krawędzie się krzyżują, obieramy najkrótszą drogę.

Jak widać na rysunku 14.22, daje nam to bąble, uszy, kocie zabawki i bazgrołki. Pokazujemy tu tylko reprezentatywną próbkę.

Rysunek 14.22. Wykryte cechy

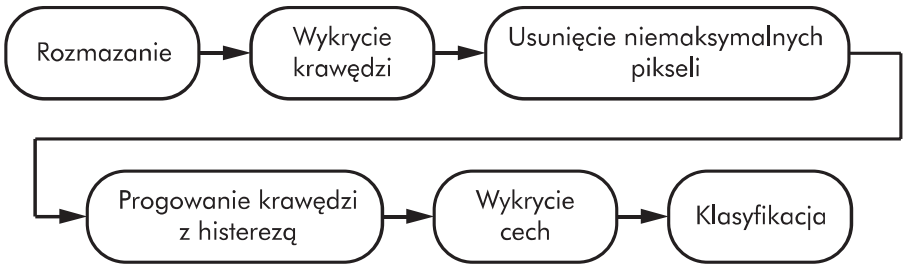
Teraz, kiedy już dysponujemy wykrytymi cechami, możemy zrobić to samo, co wcześniej, na przykład z wykrywaczem spamu. Jak widać na rysunku 14.23, podajemy nasze cechy klasyfikatorom wraz ze wskazaniem +, jeśli jest szansa, że dana cecha wskazuje na rzecz, której dany klasyfikator szuka, – jeśli wskazuje przeciwnie, oraz 0, jeśli nie ma ona znaczenia.



Rysunek 14.23. Klasyfikacja cech

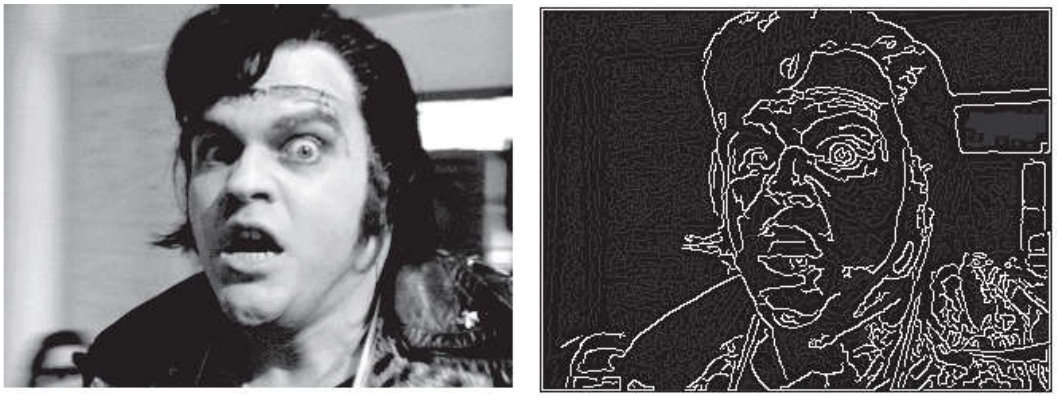
Zauważmy, że na naszych obrazkach znajdzie się nieco informacji, których możemy użyć, by ulepszyć nasze klasyfikacje – na przykład zabawki dla kotów. Przedmioty takie często można znaleźć w pobliżu kota i raczej rzadko można znaleźć ich powiązanie z pasztetami.

Ten przykład nie nadaje się na wiele poza samym pokazaniem etapów klasyfikacji cech, które są podsumowane na rysunku 14.24.



Rysunek 14.24. Potok rozpoznawania obrazów

Podczas gdy pasztety są zazwyczaj stacjonarne, koty poruszają się po okolicy i mają w swym arsenale mnóstwo słodkich póz. Nasz przykład zadziała tylko dla obiektów na naszych przykładowych zdjęciach, lecz nie rozpozna zdjęcia na rysunku 11.44 na stronie 342 jako kota. A ponieważ ma problemy z kontekstem, nie ma większych szans na rozpoznanie, że kociak na rysunku 14.25 jest równocześnie pasztetem… i w dodatku nazywa się Meat Loaf.



Rysunek 14.25. Kociak czy pasztecik?

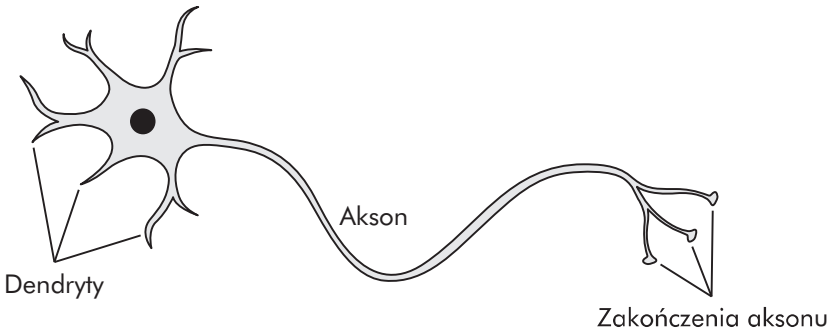
Sieci neuronowe

Pod pewnym względem, patrząc z dostatecznie ogólnego poziomu, nie ma tak naprawdę znaczenia, jakich danych używamy do reprezentacji obiektów. Musimy być w stanie radzić sobie z ogromną liczbą odmian występującą w świecie. Tak samo jak ludzie, komputery nie mogą zmienić danych wejściowych. By radzić sobie z bogactwem typologicznym, potrzebujemy lepszych klasyfikatorów.

Jednym z podejść używanych w dziedzinie sztucznej inteligencji jest naśladowanie ludzkiego zachowania. Jesteśmy całkiem pewni, że *neurony* odgrywają pierwszoplanową rolę. Ludzie mają około 86 miliardów neuronów, chociaż

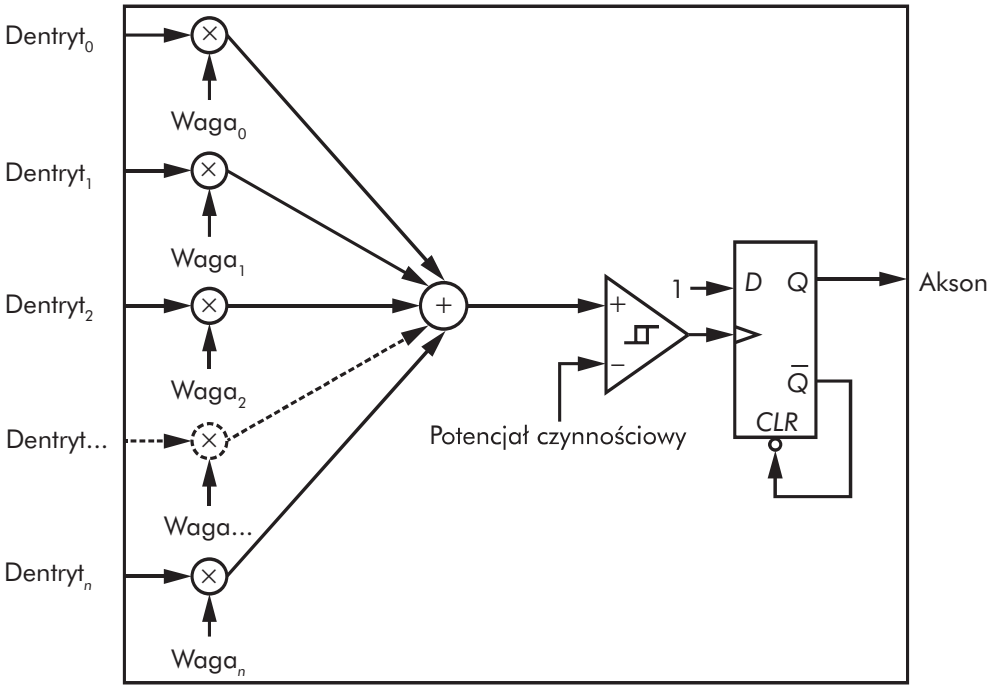
nie wszystkie znajdują się w mózgu. Komórki nerwowe to również neurony, co być może tłumaczy, dlaczego niektórzy ludzie myślą za pomocą brzuchów.

Można myśleć o neuronach jako o czymś między bramkami logicznymi z rozdziału 2 a analogowymi komparatorami z rozdziału 6. Uproszczoną ilustrację neuronu widać na rysunku 14.26.



Rysunek 14.26. Neuron

Dendryty są wejściami, a *akson* jest wyjściem. *Zakończenia aksonu* to po prostu połączenia aksonu z innymi neuronami; neurony mają tylko jedno wyjście. Neurony różnią się od czegoś w rodzaju bramki AND tym, że nie wszystkie wejścia są traktowane na równi. Przyjrzyjmy się rysunkowi 14.27.



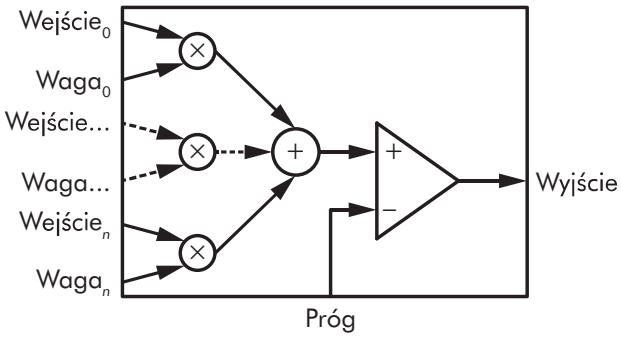
Rysunek 14.27. Uproszczony model bramkowy neuronu

Wartość wejścia każdego dendrytu jest mnożona przez pewną *wagę*, a potem sumuje się wszystkie wejściowe wartości. Pod tym względem neuron jest podobny do klasyfikatora bayesowskiego. Jeśli wartości są mniejsze od *potencjału działania*, na wyjściu komparator daje fałsz, a w przeciwnym razie daje na wyjściu prawdę, co powoduje, że neuron *odpala*, czyli wysyła impuls, przestawiając przerzutnik wyjściowy na wartość prawda. Wyjście neuronu nazywamy *impulsem*, gdy tylko osiągnie wartość prawdy, przerzutnik resetuje się

i wraca do wartości fałszu logicznego. Lub też, jeśli pobieraliśmy nauki od Pana Miyagi, „aks-on" i „aks-off"[21]. Neurolodzy mogą spierać się z opisem komparatora jako posługującego się histerezą; prawdziwe neurony tak robią, ale jest to rozłożone w czasie, czego ten model nie oddaje.

Neurony są trochę jak bramki logiczne również pod tym względem, że są „proste" – ale mogą łączyć się, tworząc skomplikowane „obwody", czy też *sieci neuronowe*. Podstawową własnością neuronu jest to, że wysyłają one impuls na podstawie ważonych wartości wejściowych. Spowodować, że neuron „wystrzeli" może wiele różnych kombinacji wejść.

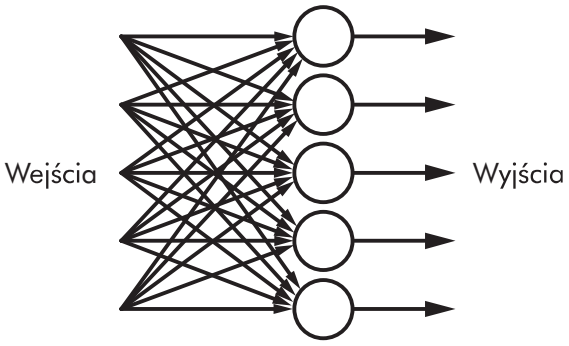
Pierwszą próbą konstrukcji sztucznego neuronu był *perceptron* wynaleziony przez amerykańskiego psychologa Franka Rosenblatta (1928–1971). Jego schemat jest widoczny na rysunku 14.28. Ważną cechą perceptronu jest to, że wejścia i wyjścia są binarne – mogą mieć wartości 0 lub 1. Wagi i progi są liczbami rzeczywistymi.



Rysunek 14.28. Perceptron

Perceptrony wywołały sporo entuzjazmu w świecie sztucznej inteligencji. Ale potem odkryto, że nie działają one dla pewnych klas problemów. To oraz kilka innych czynników doprowadziło do okresu zwanego „zimą sztucznej inteligencji", kiedy to finansowanie badań z tej dziedziny praktycznie się skończyło.

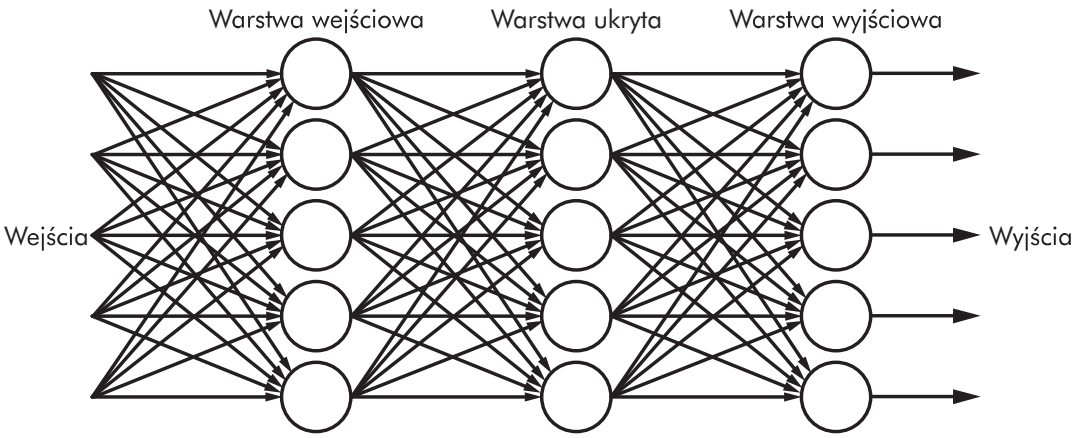
Okazuje się, że problem tkwił w sposobie, w jaki używano perceptronów. Były one uporządkowane w pojedynczą „warstwę", jak to widać na rysunku 14.29 (każdy okrąg symbolizuje jeden perceptron).



Rysunek 14.29. Jednowarstwowa sieć neuronowa

[21] Nawiązanie do postaci z serii „Karate Kid", pana Miyagi i do pojawiającej się w filmach frazy „wax-on", „wax-off" (przyp. tłum.).

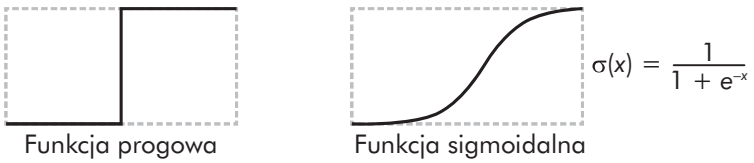
Wartości wejściowe idą do wielu perceptronów, z których każdy podejmuje jedną decyzję i na jej podstawie generuje wyjście. Wiele problemów z perceptronami zostało rozwiązanych przez wynalezienie wielowarstwowych sieci neuronowych – takich jak pokazana na rysunku 14.30.



Rysunek 14.30. Wielowarstwowa sieć neuronowa

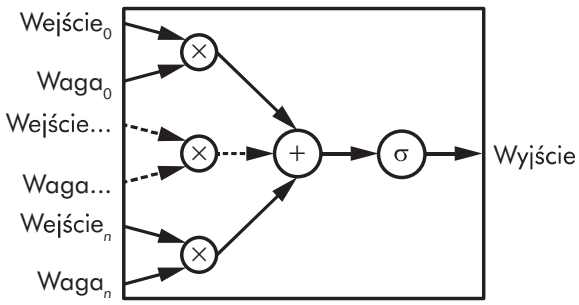
Podobne konstrukcje nazywamy też *sieciami jednokierunkowymi* (*feedforward network*), ponieważ wyjścia jednej warstwy przekazywane są do warstwy następnej i w sieci nie występuje sprzężenie zwrotne. Liczba *warstw ukrytych* może być dowolna, a ich nazwa pochodzi od tego, że nie są podłączone ani do wejść, ani do wyjść. Chociaż na rysunku 14.30 widzimy równą liczbę neuronów w każdej z warstw, nie jest to wymagane. Ustalenie liczby warstw oraz liczby neuronów w poszczególnych warstwach jest czarną magią wykraczającą poza ramy tej książki. Podobne sieci neuronowe potrafią znacznie więcej niż proste klasyfikatory.

Neurolodzy nie wiedzą jeszcze, w jakiś sposób ustalane są wagi poszczególnych dendrytów. Informatycy wymyślili za to sposób działania sztucznego dendrytu, bo inaczej neurony byłyby bezużyteczne. Cyfrowa natura perceptronów sprawia, że nie jest to łatwe, ponieważ małe zmiany w wagach nie odpowiadają proporcjonalnym zmianom na wyjściu – to wszystko albo nic. Inny projekt neuronu, *neuron sigmoidalny*, rozwiązuje ten problem przez zastąpienie komparatora *funkcją sigmoidalną*, co stanowi tylko wydumaną nazwę dla funkcji o wykresie w kształcie litery S. Rysunek 14.31 pokazuje zarówno funkcję przejścia perceptronu, jak i funkcję sigmoidalną. Przywodzi na myśl wspomnienia o naszej dyskusji na temat świata analogowego i cyfrowego z rozdziału 2, prawda?



Rysunek 14.31. Funkcje przejścia sztucznego neuronu

Wnętrze neuronu sigmoidalnego ukazano na rysunku 14.32



Rysunek 14.32. Neuron sigmoidalny

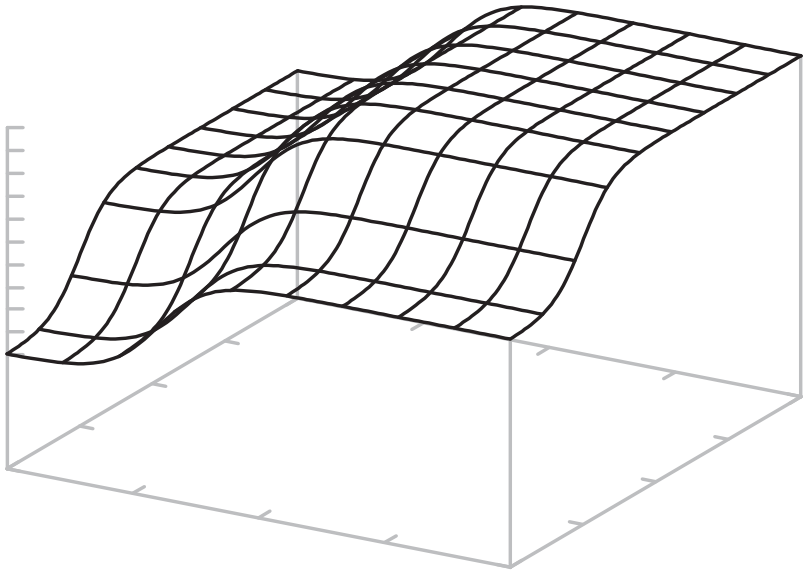
Jedną rzeczą nadal niejasną jest fakt, że wejściami i wyjściami sigmoidalnego neuronu są „analogowe" liczby zmiennoprzecinkowe. Dla ścisłości, w tego typu neuronach używa się również odchylenia (*bias*), ale nie jest to konieczne do zrozumienia, jak on działa.

W sieciach neuronowych zbudowanych z sigmoidalnych neuronów można dostosowywać wagi z użyciem techniki zwanej *propagacją wsteczną* (*backpropagation*), która okresowo zostaje zapomniana i odkrywana na nowo – ostatnio w 1986 roku, w artykule Davida Rumelharta (1942–2011), Geoffreya Hintona i Rolanda Williamsa. Do propagacji wstecznej używa się mnóstwa algebry liniowej, więc pominiemy szczegóły. Czytelnicy prawdopodobnie potrafią rozwiązać układ równań; algebra liniowa wchodzi do gry, gdy mamy dużą liczbę równań z wielką liczbą niewiadomych.

Ogólny pomysł kryjący się za propagacją wsteczną polega na tym, że prezentujemy sieci neuronowej rozpoznane dane wejściowe, na przykład dotyczące cech kotów. Przyglądamy się wartości wyjściowej – w przypadku kota na wejściu spodziewamy się wyjścia 1 lub czegoś zbliżonego. Obliczamy *funkcję błędu*, czyli rzeczywistą wartość wyjściową odjętą od spodziewanej wartości wyjściowej. Następnie zmienia się wagi tak, by następnym razem wartość funkcji błędu była tak bliska 0, jak to tylko możliwe.

Zazwyczaj dokonuje się tego za pomocą algorytmu zwanego *metodą gradientu prostego* (*gradient descent*), która dla kogoś, kto nie lubi się matematyki, wygląda jak prosta droga do piekła Dantego. Spróbujmy ją zrozumieć na prostym przykładzie. Pamiętajmy, że „gradient" to takie inne słowo na określenie nachylenia. Wypróbowujemy różne wartości wag i w zależności od ich zmiany robimy wykres wartości funkcji błędu. Może to wyglądać jak na rysunku 14.33, który przypomina nieco wypukłą mapę z górami, dolinami i tak dalej.

Metoda gradientu prostego polega w skrócie na tym, żeby toczyć piłkę po mapie, aż wyląduje ona w najgłębszej dolinie. Tam właśnie wartość funkcji błędu osiągnie swoje minimum. Ustawiamy wagi na takie wartości, na jakie wskazuje pozycja piłki. Powód, dla którego algorytm ten ma tak mądrze brzmiącą nazwę, polega na tym, że robimy to dla bardzo dużej liczby wag, tymczasem w naszym przykładzie robiliśmy to dla dwóch. A jeśli bierzemy pod uwagę wagi w różnych warstwach, jak na rysunku 14.30, to oczywiście wszystko staje się jeszcze bardziej skomplikowane.



Rysunek 14.33. Topologia gradientu

Mogliśmy zauważyć tajemnicze zniknięcie wyjściowego mechanizmu impulsowego pokazanego wcześniej na rysunku 14.27. Sieci neuronowe, które dotychczas widzieliśmy, to zasadniczo układy kombinatoryczne, a nie sekwencyjne i w rzeczywistości są one DAG-ami. Istnieje odmiana sekwencyjna zwana *rekurencyjną siecią neuronową* (*recurrent neural network*). Ona nie jest DAG-iem, co oznacza, że wyjścia z neuronów w danej warstwie mogą łączyć się z powrotem z wejściami neuronów w poprzedniej warstwie. Przechowywanie wartości wyjściowych i taktowanie tego całego bałaganu to chyba jedyne, co chroni go przed eksplozją. Tego typu sieci neuronowe są świetne do przetwarzania sekwencji wejść, takich jak te, które można znaleźćw rozpoznawaniu pisma ręcznego i mowy.

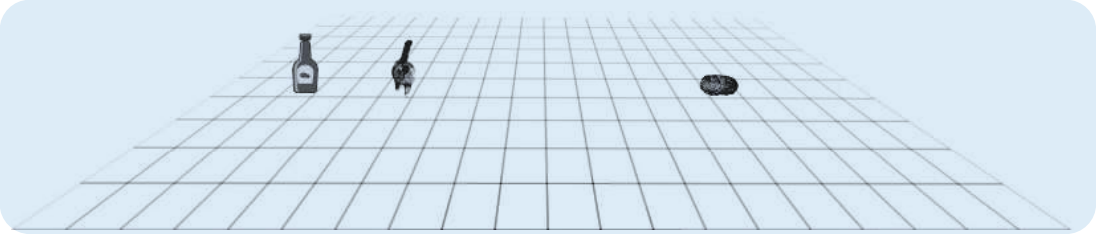
Istnieje jeszcze jedna odmiana sieci neuronowych, szczególnie dobrych w przetwarzaniu danych obrazowych: *splotowe sieci neuronowe* (*convolutional neural network*). Można sobie wyobrazić, że mają one wejścia w postaci tabeli pikseli, podobnie do macierzy splotu, które widzieliśmy powyżej.

Duży problem sieci neuronowych polega na tym, że mogą zostać „zatrute" złymi danymi. Nie wiemy, jakie dziwne zachowania mogą wystąpić u dorosłych, którzy w dzieciństwie oglądali za dużo telewizji i to samo dotyczy systemów uczenia się maszyn. Być może w przyszłości będzie zapotrzebowanie na maszynowych psychoterapeutów, chociaż trudno sobie dziś wyobrazić położenie maszyny na sofie i zapytanie jej, co naprawdę sądzi o zdjęciach kotów.

Najważniejsze w sieciach neuronowych jest to, że stanowią bardzo sprawne klasyfikatory. Mogą je przeszkolić w zakresie przetwarzania ogromnych ilości danych wejściowych na mniejszą ilość danych wyjściowych, które opisują wejścia w wybrany przez nas sposób. Osobniki lubiące zaawansowaną terminologią mogą to nazwać *redukcją wymiarowości* (*reducing dimensionality*). Teraz pozostaje nam tylko wymyślić, co zrobić z właśnie uzyskaną wiedzą.

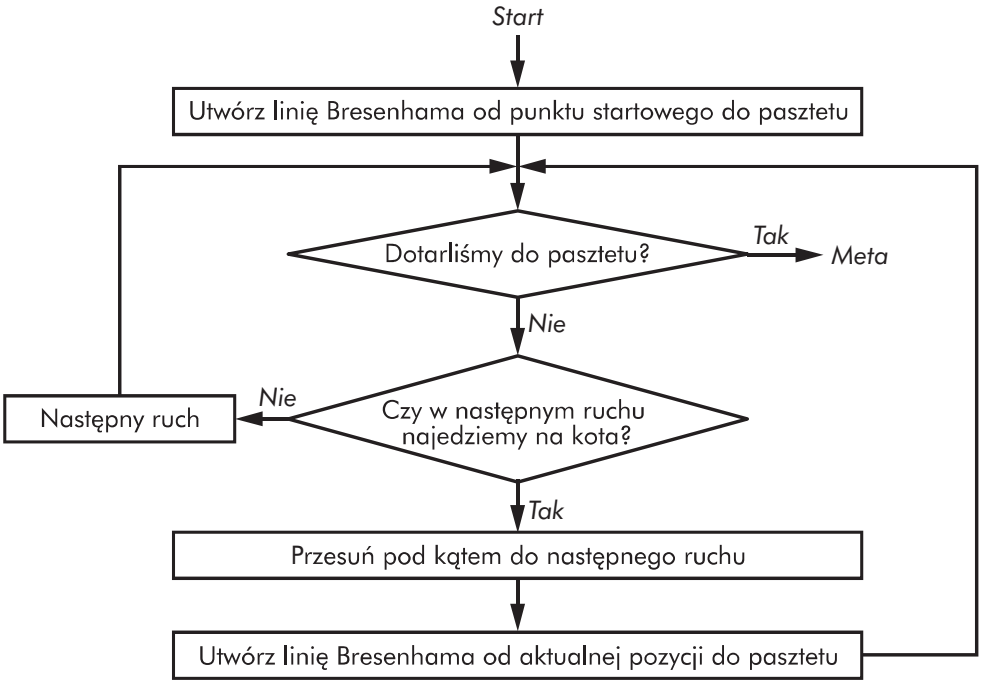
Zastosowanie uczenia się maszyn

Jak moglibyśmy zbudować coś w rodzaju samojezdnej butelki ketchupu z użyciem danych wyjściowych klasyfikatora? Użyjemy scenariusza testowego widocznego na rysunku 14.34. Będziemy poruszać butelką ketchupu po jednej kratce na ruch, jakby to był jakiś rodzaj szachownicy. Celem będzie dotarcie do pasztetu i ominięcie przy tym kota.



Rysunek 14.34. Scenariusz testowy

W tym „podręcznikowym przykładzie" klasyfikatory dają nam pozycję kota i pasztetu. Najkrótszą drogą między dwoma punktami jest linia prosta, a znajdujemy się na siatce wykreślonej liczbami całkowitymi. Skoro tak, to najbardziej wydajnym sposobem na dotarcie do pasztetu będzie użycie algorytmu Bresenhama wykreślającego linie, z którym zapoznaliśmy się już w punkcie „Linie proste" (s. 310). Oczywiście, należy go też zmodyfikować w sposób pokazany na rysunku 14.35, ponieważ koty i sosy szkodzą sobie nawzajem.



Rysunek 14.35. Algorytm samojezdnej butelki ketchupu

Jak widać, jest to dość proste. Walimy prosto do pasztetu, a jeśli kot jest na drodze, omijamy go i wytyczamy następną drogę do celu. Oczywiście w świecie rzeczywistym to o wiele bardziej skomplikowane, bo kot może się poruszać i istnieją też inne przeszkody.

Spójrzmy teraz na ten sam problem z punktu widzenia sztucznej inteligencji i zobaczmy jeszcze inne możliwe rozwiązanie.

Sztuczna inteligencja

Wczesne badania sztucznej inteligencji przyniosły dość szybko dość ekscytujące wyniki, takie jak nauczenie komputera gry w warcaby, czy rozwiązanie kilku problemów logicznych. Wynikiem było hojne i szybkie finansowanie projektów z tej dziedziny. Niestety wczesne sukcesy nie skalowały się dobrze na trudniejsze problemy i po jakimś czasie finansowanie się skończyło.

Termin „sztuczna inteligencja" ukuto w 1956 roku na warsztatach w Dartmouth. Jednym z ich uczestników był John McCarthy (1927–2011), ten sam, który później zaprojektował język LISP podczas pracy na MIT. Język ten był używany z znacznej części wczesnej pracy w tej dziedzinie. Skrót LISP oficjalnie znaczył *List Processor*, czyli „procesor listy" – jednak każdy choć trochę zaznajomiony z jego składnią nabierał powoli graniczącego z pewnością przekonania, że chodzi raczej o *Lots of Insipid Parentheses* – *Mnóstwo Głupawych Nawiasów*.

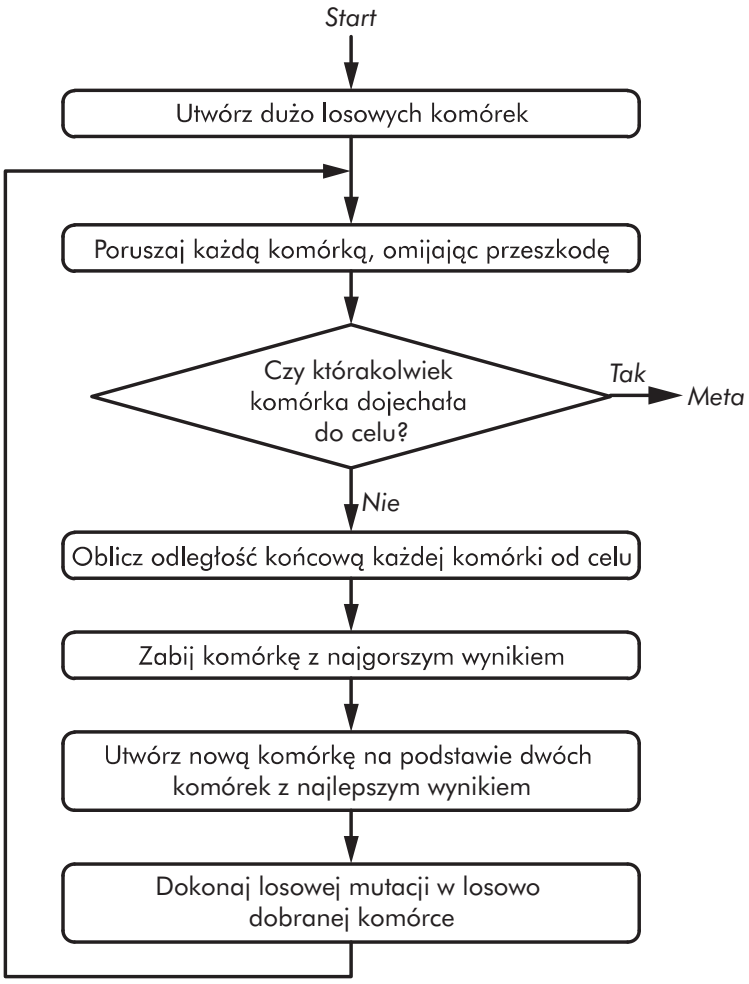
LISP wprowadził kilka nowych idei do wysokopoziomowych języków programowania. Oczywiście nie było to trudne w 1958 roku, ponieważ wtedy istniał tylko jeden inny wysokopoziomowy język programowania – FORTRAN. W szczególności LISP zawierał listy powiązane (patrz rozdz. 7) jako typ danych, co pozwalało budować programy jako listę instrukcji. To oznaczało, że programy mogły modyfikować same siebie, co stanowi główną różnicę między AI a systemami uczenia się maszyn. Sieć neuronowa może modyfikować swoje wagi, ale nie może zmieniać algorytmów. Ponieważ LISP może generować kod, może modyfikować istniejące algorytmy lub tworzyć nowe. JavaScript również wspiera *samomodyfikujący się kod*, chociaż implementacja nie jest aż tak czysta i ogólnie rzecz biorąc nie jest to bezpieczna praktyka w minimalnie ograniczonym środowisku internetowym.

Wczesne systemy sztucznej inteligencji bardzo szybko napotkały ograniczenie w postaci możliwości technicznych sprzętu. Jedną z najpopularniejszych maszyn używanych wtedy do celów badawczych było DEC PDP-10 z przestrzenią adresową ograniczoną początkowo do 256 K słów 36-bitowych. Później rozszerzono to do 4 M. To nie wystarcza, by uruchomić przykładowe systemy uczenia się maszyn opisane w tym rozdziale. We wczesnych latach 70. XX wieku amerykański programista Richard Greenblatt, wraz z inżynierem komputerowym Tomem Knightem, opracowali na MIT *Lisp-maszyny*, czyli komputery zoptymalizowane do uruchamiania programów napisanych w LISP. Nawet jednak w czasie największego ich rozkwitu nie zbudowano więcej niż kilka tysięcy takich maszyn. Możliwą przyczyną był szybki rozwój komputerów przeznaczenia ogólnego.

W latach 80. XX wieku sztuczna inteligencja zaczęła powracać na scenę wraz z wprowadzeniem *systemów ekspertowych* (*expert system*). Takie systemy

wspomagają użytkowników, np. zawodowych lekarzy, przez zadawanie pytań i przeprowadzanie ich przez bazę wiedzy. To powinno brzmieć znajomo – w istocie jest to poważniejsze zastosowanie naszej gry „Zgadnij co to za zwierzę" z rozdziału 10. Niestety, systemy ekspertowe najwyraźniej rozwinęły się w irytujące menu w telefonach.

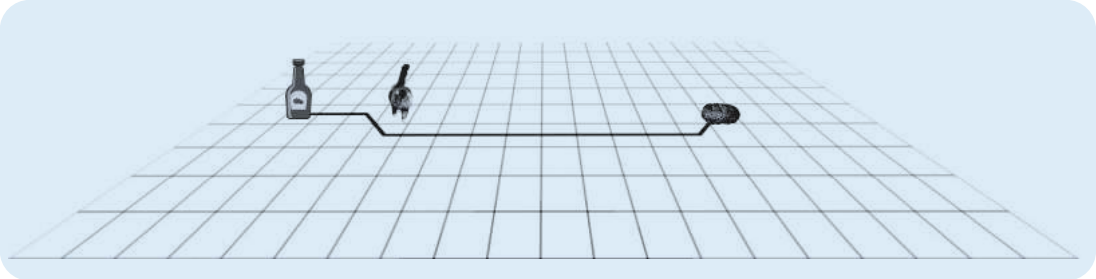
Zamierzamy zaatakować nasz problem samojezdnej butelki ketchupu za pomocą *algorytmu genetycznego*. Technika ta naśladuje ewolucję (patrz rys. 14.36).



Rysunek 14.36. Algorytm genetyczny.

Generujemy losowo zbiór samochodowych *komórek*, z których każda ma pozycję i kierunek ruchu. Każda z komórek jest przesuwana o jeden krok, po czym obliczamy wynik „dobroci" – w tym przypadku z użyciem wzoru na odległość. Rozmnażamy dwie komórki z najlepszym wynikiem, aby stworzyć nową i zabijamy komórkę z najgorszym wynikiem. A ponieważ to ewolucja, dokonujemy także losowej mutacji w którejś z komórek. Całość powtarzamy, krok za krokiem, aż któraś z komórek dotrze do celu. Kroki, jakie wykonała zwycięska komórka po drodze do celu, są naszym wygenerowanym programem.

Przyjrzyjmy się wynikom otrzymanym po wykonaniu powyższego algorytmu dla 20 komórek. Mieliśmy szczęście i otrzymaliśmy nasz wynik (pokazany na rys. 14.37) tylko w 36 iteracjach.



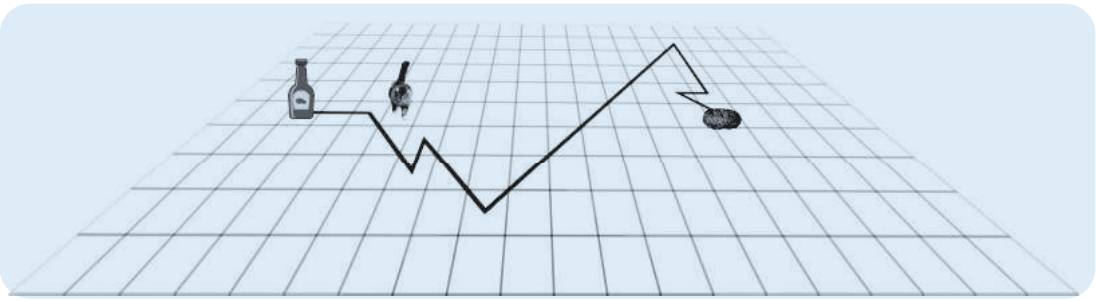
Rysunek 14.37. Dobre wyniki algorytmu genetycznego

Nasz algorytm wygenerował prosty program (pokazany na listingu 14.1), który osiąga zamierzony cel. Najważniejsze jest to, że to nie programista go napisał – nasza sztuczna inteligencja zrobiła to za nas. Zmienne x i y reprezentują położenie butelki ketchupu.

```
x++;
x++;
x++; y++;
x++;
x++;
x++;
x++;
x++;
x++;
x++;
x++;
x++;
x++; y--;
```

Listing 14.1. Wygenerowany kod

Oczywiście, skoro to genetyka, to wyniki nie zawsze będą takie same i nie zawsze będą tak zgrabne. Inne wykonanie tego samego programu zajęło 82 iteracje i znalazło rozwiązanie widoczne na rysunku 14.38. To może być świetny przykład do wykorzystania dla zwolenników „inteligentnego projektu".



Rysunek 14.38. Dziwne wyniki algorytmu genetycznego

Widać, że programy AI potrafią wygenerować dość zaskakujące wyniki. Nie różni się to jednak tak bardzo od tego, co wymyślają dzieci w czasie od-

krywania świata; różnica polega na tym, że teraz ludzie zwracają na ten proces znacznie większą uwagę. W istocie wiele rezultatów sztucznej inteligencji, które dziś zaskakują niektórych, zostało przewidzianych już dawno temu. Sensację swego czasu wzbudził fakt, że systemy AI pewnej firmy wykształciły własny, prywatny język do komunikacji między sobą. To nie jest żadna nowość dla kogokolwiek, kto oglądał film science-fiction z 1970 roku pt. *Projekt Forbina*.

Big Data

Jeśli nadal nie jest to oczywiste na podstawie dotychczasowych przykładów, trzeba to powiedzieć wprost: przetwarzamy mnóstwo danych. Kamera HD (1920×1080) generuje mniej więcej jedną trzecią gigabajtu na sekundę. Wielki Zderzacz Hadronów (LHC) generuje około 25 GiB/s. Szacuje się, że wszystkie urządzenia podłączone do sieci generują łącznie 50GiB/s – ćwierć wieku temu ta liczba wynosiła około 1 MiB/s. Większość z tych informacji to śmieci, a wyzwanie polega na tym, żeby wydobyć z tego tę użyteczną część.

Wielkie dane (*Big Data*) to ruchomy cel: termin ten określa dane zbyt wielkie i zbyt skomplikowane, by przetworzyć je technikami siłowymi z użyciem aktualnie dostępnej technologii. Ilość danych generowanych 25 lat temu dziś jest prosta do przetworzenia, ale wtedy nie była. Możliwości gromadzenia danych prawdopodobnie zawsze będą przewyższać możliwości ich analizy, więc potrzeba tu sprytu.

Termin *big data* odnosi się nie tylko do analizy danych, ale również do ich zbierania, przechowywania i zarządzania nimi. Dla naszych celów skupimy się tu jednak wyłącznie na ich analizie.

Duża część zbieranych danych jest z natury osobista. Nie jest najlepszym pomysłem dzielenie się informacjami o swoim koncie internetowym czy dokumentacji medycznej z obcymi. Ponadto dane, które są zbierane w jednym celu, często są wykorzystywane w innym. Żeby posłużyć się drastycznym przykładem, naziści wykorzystywali dane ze spisu ludności, by lokalizować i prześladować Żydów. Dane zebrane w ramach amerykańskiego spisu ludności zostały natomiast wykorzystane do zlokalizowania Japończyków mieszkających w Stanach Zjednoczonych i internowania ich – i to mimo że w mocy było prawo gwarantujące poufność wszelkich identyfikacyjnych danych osobowych przez okres 75 lat.

Dla celów badawczych wiele danych udostępnia się w formie „anonimowej", co oznacza że wszelkie dane osobowe zostały z nich usunięte. Nie jest to jednak takie proste. Techniki stosowane w big data często pozwalają na ponowną identyfikację jednostek na podstawie anonimowych danych. Co gorsza, wiele zasad mających rzekomo na celu utrudnienie takiej powtórnej identyfikacji w rzeczywistości uczyniły ją łatwiejszą.

W Stanach Zjednoczonych często nadużywa się SSN (*Social Security Number* – numer ubezpieczenia społecznego) w celu identyfikacji jednostki. Tymczasem SSN nie był tworzony z myślą o takim zastosowaniu. Karta ubezpieczenia społecznego zawiera nawet frazę „nie do celów identyfikacji", co było wymogiem oryginalnej ustawy. Teraz jednak rzadko się ją egzekwuje i dorobiono do niej mnóstwo wyjątków.

Numer SSN składa się z trzech pól: trójcyfrowego numeru obszaru (AN – *area number*), dwucyfrowego numeru grupy (GN – *group number*) i czterocyfrowego numeru seryjnego (SN – *serial number*). Numer obszaru przypisuje się na podstawie kodu pocztowego podanego na formularzu zgłoszeniowym. Numery grupowe przyznawane są w określonej kolejności (nie w kolejności rosnącej). Numery seryjne przypisuje się w kolejności rosnącej.

Grupa badaczy z Carnegie Mellon w 2009 roku opublikowała artykuł prezentujący metodę na skuteczne odgadnięcie numerów SSN. Dwie rzeczy sprawiły, że jest to łatwe zadanie. Pierwsza z nich to istnienie DMF – *Death Master File* (główny spis zgonów). Jest to lista zmarłych upubliczniana przez Administrację Opieki Społecznej pod pozorem zapobiegania oszustwom. W wygodny sposób na liście zestawione mamy nazwiska, daty urodzenia, daty śmierci, numery SSN i kody pocztowe. W jaki sposób lista zmarłych umożliwia nam odgadywanie numerów SSN żywych?

No cóż, nie są to jedyne dane, jakimi dysponujemy. Lista rejestracyjna wyborców zawiera dane o urodzeniu, zresztą tak samo jak wiele profili online.

Analiza statystyczna numerów obszaru (AN) i kodów pocztowych w głównym rejestrze zgonów może posłużyć do połączenia AN z danymi geograficznymi. Zasady przyznawania GN i SN są powszechnie znane. W rezultacie informacje z głównego rejestru zgonów pozwalają nam odwzorować AN na kod pocztowy. Niezależnie uzyskane daty urodzenia również mogą zostać połączone z kodami pocztowymi. Oba te źródła danych można połączyć, sortując po dacie urodzenia. Każda luka w numerach SSN widocznych w głównym rejestrze zgonów to żywa osoba o numerze SSN znajdującym się między dwoma numerami widocznymi w kolejnych rekordach w głównym rejestrze zgonów. Przykład pokazano w tabeli 14.2.

Oczywiście nie jest to aż tak proste, jak w tym przykładzie, ale nie jest też dużo bardziej skomplikowane. Jeśli w danych z głównego rejestru zgonów jest większa luka – jak u nas między John Chief Crier a John Small Berries – znamy tylko pewien zakres SN w zgadywanych numerach SSN. Wiele instytucji pyta jednak o ostatnią cyfrę numeru SSN. Jak widać na powyższym przykładzie, właśnie tę cyfrę jest najtrudniej zgadnąć, więc nie ułatwiajmy nikomu zadania i nie podawajmy jej każdemu, kto o nią zapyta. Zawsze można potwierdzić swoją tożsamość w jakiś inny sposób.

Oto następny przykład. GIC (Group Insurance Commision – Komisja Ubezpieczeń Grupowych) w Massachusetts w celu poprawy opieki zdrowotnej i kontroli kosztów udostępniła anonimowe dane. Gubernator Massachusets William Weld zapewniał opinię publiczną, że prywatność pacjentów jest chroniona. Można się zapewne domyślić, dokąd to zmierza – w każdym razie powinien był trzymać gębę na kłódkę.

Gubernator Weld w czasie ceremonii 18 maja 1996 roku zemdlał i został odwieziony do szpitala. Doktorantka MIT Latanya Sweeney wiedziała, że gubernator rezyduje w Cambridge i wydała 20 dolarów na pełną listę obywateli uprawnionych do głosowania w tym mieście. Połączyła dane GIC z danymi wyborców, podobnie jak to zrobiliśmy z tabelą 14.2 i z łatwością zidentyfikowała dane gubernatora. Wysłała do jego biura komplet jego danych medycznych, wliczając w to diagnozy i recepty.

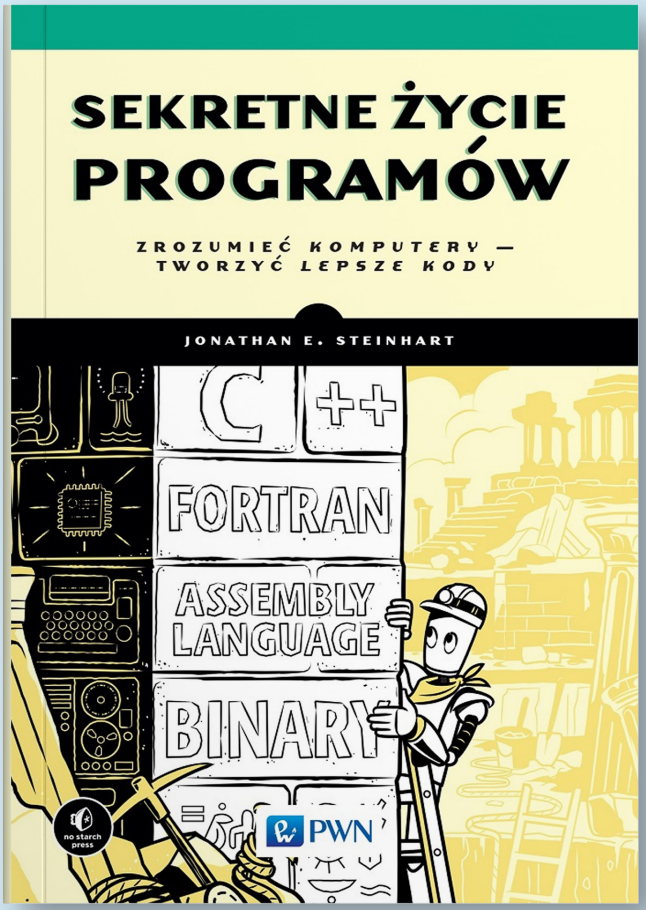
Tabela 14.2. Łączenie danych dla kodu pocztowego 89044

Główny rejestr zgonów			Odgadnięty SSN	Dane urodzenia	
Nazwisko	**Data urodzenia**	**SSN**		**Data urodzenia**	**Nazwisko**
John Many Jars	1984-01-10	051-51-1234			
John Fish	1984-02-01	051-51-1235			
John Two Horns	1984-02-12	051-51-1236			
			051-51-1237	1984-02-14	Jon Steinhart
John Worfin	1984-02-20	051-51-1238			
John Bigboote	1984-03-15	051-51-1239			
John Ya Ya	1984-04-19	051-51-1240			
			051-51-1241	1984-04-20	John Gilmore
John Fledgling	1984-05-21	051-51-1242			
			051-51-1243	1984-05-22	John Perry Barlow
John Grim	1984-06-02	051-51-1244			
John Littlejohn	1984-06-03	051-51-1245			
John Chief Crier	1984-06-12	051-51-1246			
			051-51-1247	1984-07-05	John Jacob Jingleheimer Schmidt
John Small Berries	1984-08-03	051-51-1250			

Ten przypadek był względnie prosty, ponieważ gubernator to jednak osoba publiczna. Jednak telefon komórkowy każdego z nas ma do dyspozycji więcej mocy obliczeniowej, niż mogła uzyskać Sweeney w 1996 roku. Zasoby dzisiejszych komputerów sprawiają, że spokojnie można znaleźć rozwiązania dla bardziej skomplikowanych przypadków.

Podsumowanie

W tym rozdziale omówiliśmy wiele naprawdę skomplikowanego materiału. Dowiedzieliśmy się, że uczenie się maszyn, big data i sztuczna inteligencja są ze sobą powiązane. Nauczyliśmy się również, że jeśli chcemy zagłębić się w te dziedziny, musimy przestudiować o wiele więcej matematyki. Żaden kot nie ucierpiał w czasie tworzenia tego rozdziału.



Ebook pochodzi z książki:

Sekretne życie programów. Zrozumieć komputery – tworzyć lepsze kody

Autor: Joanathan E. Steinhart

Wydawnictwo Naukowe PWN, 2021

SPRAWDŹ

Zainteresowały Cię nasze książki?

Znajdziesz je na:

IBUK libra

Ibuk Libra to czytelnia on-line czynna całą dobę. Dostępne w niej są tysiące e-booków oraz e-czasopism z niemal każdej dziedziny. Do IBUKA Libry możesz zalogować się z dowolnego miejsca, o każdej porze. Korzystanie z IBUKA Libry jest bezpłatne - poproś o dostęp w swojej bibliotece.

Przejdź do IBUK Libra

IBUK

IBUK.pl jest platformą pozwalającą kupować i wypożyczać e-booki. Można je wypożyczać zarówno pojedynczo - już od 4,92 PLN na dobę oraz w abonamentach - ceny zaczynają się nawet od 19,90 PLN miesięcznie. W ofercie dostępne sa także audiobooki.

Przejdź do Ibuk.pl

PWN KSIĘGARNIA INTERNETOWA

Księgarnia Internetowa PWN oferuje szeroki zakres publikacji: podręczniki akademickie, książki naukowe i popularnonaukowe, słowniki języka polskiego i słowniki języków obcych. Znajdziesz w niej zarówno publikacje papierowe, jak i książki w wersji elektronicznej – e-booki i audiobooki.

Przejdź do Księgarni Internetowej PWN

Śledź nas na Facebooku:

PWN STEM