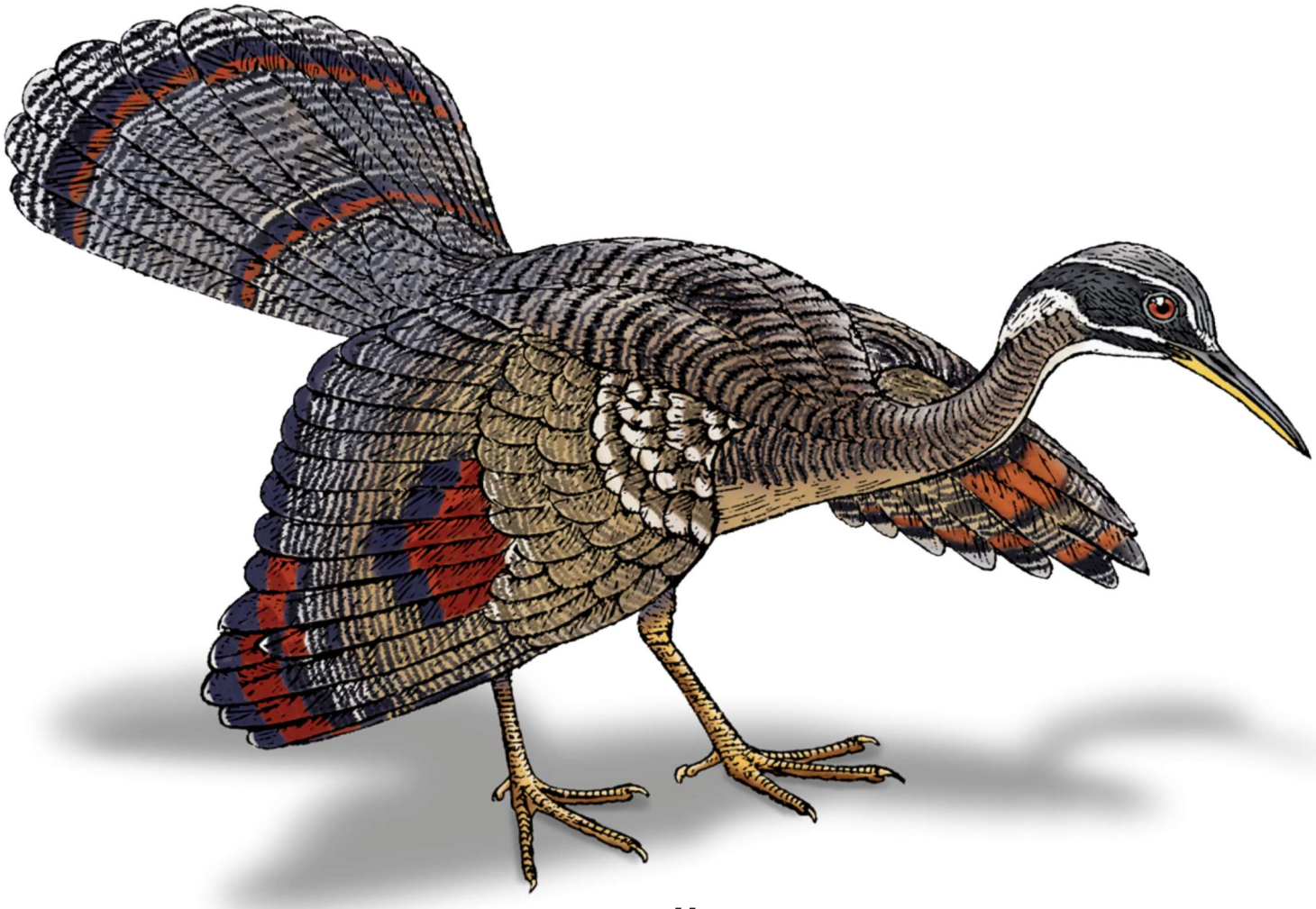
O'REILLY®

Wzorce projektowe uczenia maszynowego

Rozwiązania typowych problemów dotyczących przygotowania danych, konstruowania modeli i MLOps



Valliappa Lakshmanan,
Sara Robinson, Michael Munn

Wzorce projektowe uczenia maszynowego

Rozwiązania typowych problemów dotyczących przygotowania danych, konstruowania modeli i MLOps

Valliappa Lakshmanan,
Sara Robinson, Michael Munn

przekład: Maria Chaniewska

APN Promise
Warszawa 2021

O'REILLY®

Wzorce projektowe uczenia maszynowego

Authorized translation of English edition of
Machine Learning Design Patterns
ISBN 978-1-098-11578-4

APN PROMISE SA, ul. Domaniewska 44a, 02-672 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

ISBN: 978-83-7541-441-7 (wyd. drukowane), 978-83-7541-445-5 (ebook)

Projekt okładki: Karen Montgomery
Ilustracje: Kate Dullea

Przekład: Maria Chaniewska
Redakcja: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści

Wstęp

Dla kogo jest ta książka?

Książki stanowiące wprowadzenie do uczenia maszynowego (*Machine Learning*, ML) zwykle są ukierunkowane na pytania, *co* i *jak* robić. Następnie zawierają objaśnienia matematycznych aspektów nowych metod opracowanych w laboratoriach badawczych sztucznej inteligencji (AI) i instrukcje używania frameworków AI do implementacji tych metod. Natomiast w niniejszej książce przedstawiono wynikające z doświadczenia odpowiedzi na pytanie „dlaczego", obejmujące wskazówki i porady stosowane przez praktyków ML do rozwiązywania rzeczywistych problemów przy użyciu uczenia maszynowego.

Zakładamy, że Czytelnicy znają podstawy uczenia maszynowego i przetwarzania danych. Nie jest to podręcznik dla początkujących. Niniejsza książka jest przeznaczona dla naukowców danych, inżynierów danych oraz inżynierów uczenia maszynowego, którzy poszukują kolejnych wskazówek i porad na temat praktyki uczenia maszynowego. Skatalogowaliśmy tu pomysły, z których część (jako praktyk ML) możesz już znać, i nadaliśmy tym pomysłom nazwy, aby można było śmiało po nie sięgać.

Jeśli jesteś studentem informatyki i zamierzasz pracować w branży, książka pozwoli uzupełnić wiedzę i pomoże przygotować się do wejścia w świat profesjonalistów. Dzięki tej książce nauczysz się budować systemy uczenia maszynowego wysokiej jakości.

Czego nie ma w tej książce

Ta książka przeznaczona jest głównie dla inżynierów ML w przedsiębiorstwach, a nie naukowców ML na uniwersytetach bądź w branżowych laboratoriach badawczych.

Celowo nie omawiamy obszarów aktywnych badań – znajdziesz tu bardzo mało informacji np. o architekturze modeli uczenia maszynowego (np. koderach dwukierunkowych, mechanizmie uwagi lub warstwach zwarciowych), ponieważ zakładamy, że Czytelnicy będą używać architektury przygotowanego wstępnie modelu (takiego jak ResNet-50 lub GRUCell), a nie pisać własny klasyfikator obrazów czy rekurencyjną sieć neuronową.

Oto kilka konkretnych przykładów dziedzin, od których celowo trzymaliśmy się z daleka, ponieważ uważamy, że te tematy bardziej nadają się jako treść wykładów akademickich i obszary zainteresowań badaczy ML:

Algorytmy uczenia maszynowego
: Na przykład nie opisujemy różnic między lasami losowymi a sieciami neuronowymi. Jest to opisane w licznych wprowadzających podręcznikach uczenia maszynowego.

Bloki konstrukcyjne
: Nie opisujemy różnych typów optymalizatorów spadku gradientu ani funkcji aktywacji. Zalecamy użycie optymalizatora Adam i funkcji ReLU – z naszego doświadczenia wynika, że potencjał poprawy wydajności wynikający z wyboru innych algorytmów jest w praktyce niewielki.

Architektury modeli uczenia maszynowego
: Czytelnikom zajmującym się klasyfikacją obrazów zalecamy użycie gotowego modelu, takiego jak ResNet lub innego z ostatnio najpopularniejszych w czasie czytania tej książki. Zalecamy pozostawienie projektowania nowych modeli klasyfikacji obrazów lub modeli klasyfikacji tekstu badaczom, którzy specjalizują się w tym problemie.

Warstwy modelu
: Nie znajdziesz w tej książce konwolucyjnych ani rekurencyjnych sieci neuronowych. Zostały dwojako zdyskwalifikowane – po pierwsze, ponieważ *są* blokami konstrukcyjnymi, a po drugie, ponieważ można użyć gotowych produktów.

Niestandardowe pętle uczenia
: Proste wywołanie metody `model.fit()` w Keras spełnia potrzeby wszystkich praktyków.

W tej książce spróbowaliśmy umieścić tylko takie powszechne wzorce, które inżynierowie uczenia maszynowego w przedsiębiorstwach będą wykorzystywać w codziennej pracy.

Jako analogię możemy rozważyć struktury danych. Choć kurs uniwersytecki dotyczący struktur danych będzie wnikać w implementacje różnych struktur danych, a badacze struktur danych będą musieli uczyć się, jak formalnie przedstawiać własności matematyczne, praktycy mogą być bardziej pragmatyczni. Deweloper oprogramowania w przedsiębiorstwie potrzebuje wiedzieć, jak wydajnie pracować z tablicami, listami, tablicami asocjacyjnymi, zbiorami i drzewami. Podobnie jest w przypadku pragmatycznego praktyka uczenia maszynowego, dla którego jest napisana ta książka.

Przykłady kodu

Udostępniamy kod do uczenia maszynowego (czasami w Keras/TensorFlow, a kiedy indziej w scikit-learn lub BigQuery ML) oraz przetwarzania danych (w SQL) jako sposób przedstawienia praktycznej implementacji opisywanych technik. Cały kod, do którego odwołujemy się w tej książce, jest częścią naszego repozytorium GitHub (*https://github.com/GoogleCloudPlatform/ml-design-patterns*), w którym można znaleźć w pełni

działające modele uczenia maszynowego. Stanowczo zachęcamy do wypróbowania tych przykładów kodu.

Kod jest drugoplanowy względem opisywanych koncepcji i technik. Naszym celem było zachowanie poprawności przedstawionych tematów i zasad, bez względu na (nieuniknione) zmiany w TensorFlow lub Keras. Z łatwością możemy wyobrazić sobie uzupełnienie repozytorium GitHub, aby na przykład uwzględnić inne frameworki uczenia maszynowego przy zachowaniu niezmiennego tekstu książki. Dlatego książka powinna być równie pouczająca, jeśli głównym framewokiem uczenia maszynowego jest PyTorch, a nawet framework niezwiązany z językiem Python, taki jak H20.ai lub R. W istocie zachęcamy Czytelników do współtworzenia repozytorium GitHub przez implementowanie opisanych wzorców w ulubionym frameworku uczenia maszynowego.

W razie pytań technicznych lub problemów z używaniem przykładów kodu zachęcamy do wysłania wiadomości e-mail do *bookquestions@oreilly.com*.

Ta książka powstała, aby pomóc Czytelnikom wykonać swoją pracę. W ogólności, jeśli przykład kodu jest udostępniony w tej książce, Czytelnicy mogą go używać w swoich programach i dokumentacji. Nie muszą kontaktować się z nami, aby uzyskać pozwolenie, o ile nie powielają znacznej części kodu. Na przykład napisanie programu korzystającego z kilku fragmentów kodu z tej książki nie wymaga pozwolenia. Sprzedaż lub dystrybucja przykładów z książek wydawnictwa O'Reilly wymaga pozwolenia. Odpowiedź na pytania z cytatem z tej książki i przykładowym kodem nie wymaga pozwolenia. Uwzględnienie znacznej liczby przykładów kodu z tej książki w dokumentacji produktu wymaga pozwolenia.

Doceniamy atrybucje, ale w ogólności ich nie wymagamy. Atrybucja zwykle obejmuje tytuł, autora, wydawcę i ISBN. Na przykład: „*Machine Learning Design Patterns*, Valliappa Lakshmanan, Sara Robinson Michael Munn (O'Reilly). Copyright 2021 Valliappa Lakshmanan, Sara Robinson i Michael Munn, 978-1-098-11578-4". Jeśli uważasz, że użycie przykładów kodu wykracza poza dozwolone użycie lub pozwolenie podane powyżej, zachęcamy do kontaktu z nami pod adresem *permissions@oreilly.com*.

Konwencje użyte w tej książce

W tej książce użyto następujących konwencji typograficznych:

Kursywa
: Oznacza nowe terminy, adresy URL, adresy e-mail, nazwy plików i rozszerzenia plików.

`Stała szerokość`
: Stosowana w listingach programów, a także wewnątrz akapitów w celu odwoływania się do elementów programu, takich jak nazwy zmiennych lub funkcji, bazy danych, typy danych, zmienne środowiskowe, instrukcje i słowa kluczowe.

`Stała szerokość z pogrubieniem`
: Przedstawia polecenia i inne teksty, które powinny być wpisane dosłownie przez użytkownika.

`Stała szerokość z kursywą`

Przedstawia tekst, który powinien być zastąpiony przez wartości podawane przez użytkownika lub wartości wynikające z kontekstu, a także komentarze w przykładach kodu.

Ten element oznacza wskazówkę lub sugestię.

Ten element oznacza uwagę ogólną.

Ten element wskazuje ostrzeżenie lub przestrogę.

Platforma szkoleniowa O'Reilly online

Przez ponad 40 lat wydawnictwo *O'Reilly Media* dostarcza szkolenia technologiczne i biznesowe, wiedzę oraz spostrzeżenia, które pomagają firmom osiągać sukces.

Nasza unikalna sieć ekspertów i innowatorów dzieli się wiedzą i umiejętnościami w książkach, artykułach i na naszej platformie szkoleniowej online. Platforma szkoleniowa O'Reilly online daje Czytelnikom na żądanie dostęp do szkoleń na żywo, głębokich ścieżek szkoleniowych, interaktywnych środowisk programistycznych oraz obszernej kolekcji tekstu i wideo z wydawnictwa O'Reilly i ponad 200 innych wydawnictw. Aby uzyskać więcej informacji, odwiedź witrynę *http://oreilly.com*.

Jak się z nami skontaktować

Zachęcamy do przesyłania komentarzy i pytań dotyczących niniejszej książki do wydawcy:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (w USA lub Kanadzie)

707-829-0515 (międzynarodowo lub lokalnie)

707-829-0104 (faks)

Przygotowaliśmy stronę internetową dla tej książki, na której umieszczamy erratę, przykłady i dowolne dodatkowe informacje. Dostęp do tej strony można uzyskać pod adresem *https://oreil.ly/MLDP*.

Wyślij e-mail na adres *bookquestions@oreilly.com*, aby skomentować lub zadać pytania techniczne dotyczące tej książki.

Aby poznać wiadomości i informacje o naszych książkach i kursach, odwiedź *http://oreilly.com*.

Znajdź nas na Facebooku: *http://facebook.com/oreilly*

Śledź nas na Twitterze: *http://twitter.com/oreillymedia*

Oglądaj nas na YouTube: *http://youtube.com/oreillymedia*

Podziękowania

Książka taka jak ta nie mogłaby powstać bez wspaniałomyślności wielu pracowników Google, szczególnie naszych kolegów z zespołów Cloud AI, Solution Engineering, Professional Services oraz Developer Relations. Jesteśmy im wdzięczni za zgodę na obserwowanie, analizowanie i zadawanie pytań dotyczących rozwiązań trudnych problemów, które napotykali podczas uczenia i poprawiania modeli uczenia maszynowego oraz operowania nimi. Dziękujemy naszym menedżerom, którymi są Karl Weinmeister, Steve Cellini, Hamidou Dia, Abdul Razack, Chris Hallenbeck, Patrick Cole, Louise Byrne oraz Rochana Golani, za sprzyjanie duchowi otwartości w Google, co dało nam swobodę skatalogowania wzorców i opublikowania książki.

Salem Haykal, Benoit Dherin i Khalid Salama przejrzeli każdy wzorzec i każdy rozdział. Sal wskazał na niuanse, które pominęliśmy, Benoit zawęził nasze twierdzenia, a Khalid wskazał nam odpowiednie badania. Ta książka nie byłaby równie dobra bez Waszego wkładu. Dziękujemy! Amy Unruh, Rajesh Thallam, Robbie Haertel, Zhitao Li, Anusha Ramesh, Ming Fang, Parker Barnes, Andrew Zaldivar, James Wexler, Andrew Sellergren oraz David Kanter przejrzeli części tej książki, które dotyczą obszarów ich wiedzy i dokonali wielu sugestii dotyczących wpływu krótkoterminowego harmonogramu na nasze zalecenia. Nitin Aggarwal i Matthew Yeager rzucili czytelniczym okiem na rękopis i poprawili jego przejrzystość. Na specjalne podziękowania zasłużył Rajesh Thallam za prototypowanie projektu ostatniego rysunku w rozdziale 8. Odpowiedzialność za ewentualne błędy bierzemy oczywiście na siebie.

O'Reilly jest wydawcą wybieranym dla książek technicznych, a profesjonalizm naszego zespołu to uzasadnia. Rebecca Novak przeprowadziła nas przez składanie fascynującego konspektu, Kristen Brown w opanowany sposób zarządzała całym wytwarzaniem treści, Corbin Collins na każdym etapie udzielał nam pomocnych wskazówek, praca z Elizabeth Kelly podczas produkcji byłą przyjemnością, a Charles Roumeliotis rzucił czujnym okiem podczas korekty. Dziękujemy Wam wszystkim za pomoc!

Michael: Dziękuję rodzicom za to, że zawsze we mnie wierzyli i zachęcali do rozwijania zainteresowań, zarówno akademickich, jak i innych. Będziecie w stanie docenić tak samo

jak ja dyskretne wsparcie. Phil, dziękuję Ci za cierpliwe znoszenie mojego harmonogramu, który był nie do zniesienia, gdy pracowałem nad tą książką. Teraz czas się wyspać.

Sara: Jon – jesteś głównym powodem istnienia tej książki. Dziękuję za zachęcanie mnie do jej napisania, za nieustanne rozśmieszanie, docenianie mojego bzika oraz za wierzenie we mnie, szczególnie, kiedy wątpiłam. Moim rodzicom dziękuję za to, że jesteście moimi największymi fanami od pierwszego dnia i popieraliście moje zamiłowanie do technologii i pisania, od kiedy sięgam pamięcią. Ally, Katie, Randi i Sophie – dziękuję za to, że jesteście stałym źródłem światła i radości w tych niepewnych czasach.

Lak: Zabrałem się za tę książkę z myślą, że będę nad nią pracować, czekając na lotniskach. COVID-19 sprawił, że większość pracy wykonałem w domu. Abirami, Sidharth i Sarada dziękuję Wam za wyrozumiałość, gdy zamykałem się, aby znowu pisać. Teraz czas na więcej wycieczek w weekendy!

Wszyscy troje przekazaliśmy 100% honorarium z tej książki dla organizacji Girls Who Code (*https://girlswhocode.com*), której misją jest zwiększenie liczby przyszłych kobiet-inżynierów. Różnorodność, równość i integracja są szczególnie ważne w uczeniu maszynowym, aby zapewnić, że modele sztucznej inteligencji nie będą utrwalały istniejących nierówności w społeczeństwie.

Rozdział 1

Zapotrzebowanie na wzorce projektowe uczenia maszynowego

Wzorce projektowe w inżynierii określają najlepsze praktyki i rozwiązania powszechnie występujących problemów. Kodyfikują wiedzę i doświadczenie ekspertów w postaci rad, z których mogą korzystać wszyscy praktycy. Niniejsza książka jest katalogiem wzorców uczenia maszynowego, które zaobserwowaliśmy pracując z setkami zespołów zajmujących się uczeniem maszynowym.

Co to są wzorce projektowe?

Idea wzorców oraz katalogowania wypróbowanych wzorców została wprowadzona w dziedzinie architektury przez Christophera Alexandra i pięciu współautorów w bardzo wpływowej książce zatytułowanej *Język wzorców* (Oxford University Press, 1977). Książka ta zawiera katalog 253 wzorców przedstawionych w następujący sposób:

> Każdy wzorzec opisuje pewien problem, który stale pojawia się w naszym otoczeniu. Wskazuje następnie istotę rozwiązania tego problemu w taki sposób, abyś mógł korzystać z owego rozwiązania wielokrotnie, nigdy nie powtarzając działania.
>
> …
>
> Każde rozwiązanie jest tak sformułowane, że podaje istotne pole relacji, niezbędne do rozwiązania problemu. Jednak relacje te są opisane na tyle ogólnie i abstrakcyjnie, abyś mógł rozwiązać dany problem dla siebie, na swój sposób, zgodnie z własnymi upodobaniami oraz lokalnymi warunkami w miejscu, w którym to robisz.

Na przykład para wzorców uwzględniających ludzkie preferencje przy projektowaniu domu to *Światło z dwóch stron w każdym pokoju* oraz *Dwumetrowy balkon*. Pomyśl

o ulubionym pokoju w swoim mieszkaniu oraz najmniej lubianym. Czy ulubiony pokój ma okna na dwóch ścianach? A co z najmniej lubianym pokojem? Zdaniem Alexandra:

> W pokojach, w których naturalne światło pada z dwóch stron, nie jest ono tak jaskrawe i w mniejszym stopniu występują olśnienia, gdy patrzymy na ludzi i przedmioty. Pozwala to na bardziej precyzyjne oglądanie tego, co się znajduje w pomieszczeniu, a co ważniejsze, umożliwia również dokładne odczytanie nieznacznych ruchów rąk czy też przelotnych zmian wyrazu twarzy...

Nazwanie tego wzorca oszczędziło architektom konieczności stałego ponownego odkrywania tej zasady. Jednak to, gdzie i w jaki sposób uzyskamy dwa źródła światła w konkretnych warunkach lokalowych, zależy od umiejętności architekta. Podobnie, jaka powinna być wielkość projektowanego balkonu? Alexander zaleca 6 stóp (około 2 m) razy 6 stóp za wystarczający dla 2 (niedobranych!) krzeseł i małego stolika, a 12 stóp razy 12 stóp, jeśli chcemy mieć zarówno zacienione miejsce siedzenia, jak i miejsce siedzenia w słońcu.

Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides[1] przenieśli tę ideę do oprogramowania i skatalogowali 23 zorientowane obiektowo wzorce projektowe w wydanej w 1994 roku książce *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* (Addison-Wesley, 1995, wyd. polskie Helion, 2010). Ich katalog obejmuje wzorce, takie jak Proxy, Singleton i Dekorator, aby wymienić tylko te najważniejsze. Wywarł trwały wpływ na dziedzinę programowania zorientowanego obiektowo. W roku 2005 stowarzyszenie Association of Computing Machinery (ACM) nagrodziło autorów doroczną nagrodą Programming Languages Achievement Award, uznając wpływ ich pracy „na praktykę programowania i projektowanie języków programowania".

Konstruowanie produkcyjnych modeli uczenia maszynowego coraz bardziej staje się dziedziną inżynierii, wykorzystującą metody uczenia maszynowego, których poprawność została dowiedziona w badaniach naukowych, w celu zastosowania ich do problemów biznesowych. Ponieważ uczenie maszynowe coraz bardziej należy do głównego nurtu informatyki, ważne jest, aby praktycy korzystali z wypróbowanych i udowodnionych metod rozwiązywania powtarzalnych problemów.

Nasza praca w części Google Cloud bezpośrednio obsługującej klientów ma tę zaletę, że zapewnia kontakt z różnorodnymi zespołami i indywidualnymi deweloperami zajmującymi się uczeniem maszynowym i nauką o danych z całego świata. W tym samym czasie każde z nas blisko współpracuje z wewnętrznymi zespołami Google rozwiązującymi nowatorskie problemy uczenia maszynowego. Ponadto mieliśmy szczęście pracować z zespołami TensorFlow, Keras, BigQuery ML, TPU oraz Cloud AI Platform, które doprowadziły do demokratyzacji badań i udostępniania infrastruktury uczenia maszynowego. Wszystko to dało nam raczej unikalną perspektywę, z jakiej katalogowaliśmy zaobserwowane najlepsze rozwiązania realizowane przez te zespoły.

Niniejsza książka jest katalogiem wzorców projektowych, czyli powtarzalnych rozwiązań problemów występujących powszechnie w inżynierii ML. Na przykład wzorzec

1 Ten zespół autorów często określany jest mianem „Gang of Four" (Gang czterech).

Przekształcenie (rozdział 6) wymusza rozdzielenie danych wejściowych, cech i przekształceń oraz sprawia, że przekształcenia są trwałe, co ma na celu uproszczenie przenoszenia modelu ML do produkcji. Podobnie *Predykcje z kluczami*, omawiane w rozdziale 5, są wzorcem, który umożliwia dystrybucję na dużą skalę predykcji realizowanych partiami, takich jak modele rekomendacji.

Dla każdego wzorca opisujemy odpowiedni, powszechnie występujący problem, a następnie przechodzimy przez różnorodne potencjalne rozwiązania problemu, kompromisy tych rozwiązań oraz rekomendacje wyboru między tymi rozwiązaniami. Kod implementacji dla tych rozwiązań jest podany w SQL (przydatny podczas przeprowadzania wstępnego przetwarzania danych i innych operacji ETL w Spark SQL, BigQuery itp.), scikit-learn oraz/lub Keras z zapleczem TensorFlow.

Jak używać tej książki

To jest katalog wzorców zaobserwowanych w praktyce, wśród wielu zespołów. W niektórych przypadkach leżące u podstaw koncepcje były znane od wielu lat. Nie przypisujemy sobie wynalezienia ani odkrycia tych wzorców. Zamiast tego mamy nadzieję przedstawić wspólny układ odniesienia i zbiór narzędzi dla praktyków ML. Osiągniemy sukces, jeśli dzięki tej książce zespoły zyskają słownictwo do omawiania koncepcji, które już intuicyjnie wcielają w projektach ML.

Nie oczekujemy, że ta książka będzie czytana po kolei (chociaż jest to możliwe). Zamiast tego zakładamy, że będzie przeglądana, a niektóre fragmenty będą czytane dokładniej niż pozostałe. Czytelnicy będą odwoływać się do tych pomysłów w rozmowach z kolegami i wracać z powrotem do książki, kiedy napotkają problemy, o których przypomną sobie, że już czytali. Osobom planującym czytać fragmentami, zalecamy rozpoczęcie od rozdziałów 1 i 8 przed zagłębieniem się w poszczególne wzorce.

Każdy wzorzec zawiera krótką definicję, kanoniczne rozwiązanie, wyjaśnienie, dlaczego to rozwiązanie działa, a także wieloczęściową dyskusję o kompromisach i alternatywach. Zalecamy przeczytanie punktu dyskusji z myślą o rozwiązaniu kanonicznym, dla porównania i kontrastu. Opis wzorca będzie zawierać fragmenty kodu pochodzące z implementacji rozwiązania kanonicznego. Pełen kod znajduje się w naszym repozytorium GitHub[2]. Zdecydowanie zachęcamy do uważnego zapoznania się z kodem podczas czytania opisu wzorca.

Terminologia uczenia maszynowego

Ponieważ praktycy uczenia maszynowego zazwyczaj wywodzą się z innej dziedziny głównej specjalizacji – takiej jak inżynieria oprogramowania, analiza danych, DevOps lub statystyka – mogą występować drobne różnice w sposobie używania pewnych terminów przez różne osoby. W tym punkcie definiujemy terminologię używaną w całej książce.

2 *https://github.com/GoogleCloudPlatform/mldesign-patterns*

Modele i frameworki

Uczenie maszynowe w istocie jest procesem budowania modeli, które uczą się na podstawie danych. Jest to przeciwne do tradycyjnego programowania, w którym mówimy programom, jak mają działać. *Modele* uczenia maszynowego są algorytmami, które uczą się wzorców na podstawie danych. Aby zilustrować ten punkt widzenia, wyobraź sobie, że mamy firmę przeprowadzkową, która chce oszacować koszty dla potencjalnych klientów. W tradycyjnym programowaniu moglibyśmy to rozwiązać taką instrukcją warunkową:

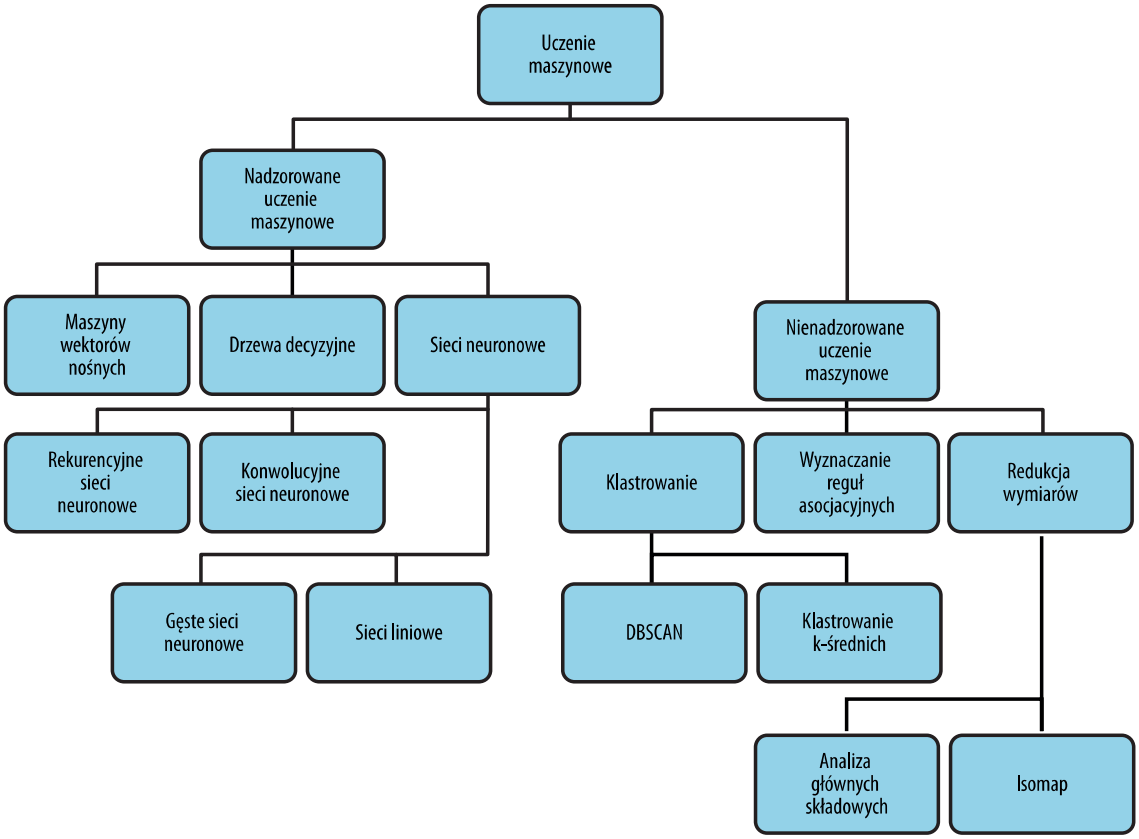
```
if num_bedrooms == 2 and num_bathrooms == 2:
  estimate = 1500
elif num_bedrooms == 3 and sq_ft > 2000:
  estimate = 2500
```

Można sobie wyobrazić, jak szybko to by się skomplikowało, gdy dodawalibyśmy więcej zmiennych (liczbę dużych mebli, ilość ubrań, delikatne przedmioty itp.) i spróbowalibyśmy obsłużyć skrajne przypadki. Co ważniejsze, pytanie klientów z góry o te wszystkie informacje może spowodować, że zrezygnują z procesu szacowania, a w konsekwencji z usług naszej firmy. Zamiast tego możemy wytrenować model uczenia maszynowego, aby szacował koszty przeprowadzki na podstawie danych z poprzednich gospodarstw domowych przeprowadzanych przez naszą firmę.

W tej książce w przykładach używamy głównie modeli skierowanych (jednokierunkowych) sieci neuronowych, ale odwołujemy się także do modeli regresji liniowej, drzew decyzyjnych, modeli klastrowania i innych. *Skierowane sieci neuronowe*, które będziemy zwykle nazywać krócej *sieciami neuronowymi* to typ algorytmu uczenia maszynowego, gdzie wiele warstw, każda z wieloma neuronami, analizuje i przetwarza informacje, a następnie wysyła te informacje do następnej warstwy, co w efekcie daje końcową warstwę, która wytwarza predykcję jako wyjście. Chociaż nie są w żaden sposób identyczne, sieci neuronowe są często porównywane do neuronów w naszym mózgu z powodu łączności między węzłami i sposobu, jak generalizują i formułują nowe predykcje na podstawie przetwarzanych danych. Sieci neuronowe z więcej niż jedną *warstwą ukrytą* (warstwami innymi niż wejściowa i wyjściowa) są klasyfikowane jako *uczenie głębokie* (patrz rysunek 1-1).

Modele uczenia maszynowego, bez względu na ich wizualne przedstawienie, są funkcjami matematycznymi, a zatem mogą być implementowane od podstaw przy użyciu pakietu oprogramowania numerycznego. Jednak inżynierowie ML w branży mają tendencje do używania jednego z wielu frameworków uczenia maszynowego zaprojektowany, aby zapewniać intuicyjne API do konstruowania modeli. W większości naszych przykładów wykorzystujemy *TensorFlow*, framework uczenia maszynowego open source utworzony przez Google i zorientowany na modele uczenia głębokiego. Wewnątrz biblioteki TensorFlow będziemy w naszych przykładach używać interfejsu API *Keras*, który możemy zaimportować za pomocą `tensorflow.keras`. Keras jest interfejsem API wysokiego poziomu do konstruowania sieci neuronowych. Chociaż Keras obsługuje wiele zapleczy, będziemy używać jego zaplecza TensorFlow. W innych przykładach będziemy używać

scikit-learn, *XGBoost* oraz *PyTorch*, które są innymi popularnymi frameworkami open source, udostępniającymi narzędzia do przygotowywania danych, wraz z interfejsami API do konstruowania modeli liniowych i głębokich. Uczenie maszynowe staje się coraz bardziej dostępne, a jednym z ekscytujących postępów jest dostępność modeli uczenia maszynowego, które mogą być wyrażane w języku SQL. Jako takiego przykładu będziemy używać *BigQuery ML*, szczególnie w sytuacjach, gdzie chcemy połączyć wstępne przetwarzanie danych i tworzenie modelu.



Rysunek 1-1 *Podział różnych typów uczenia maszynowego z kilkoma przykładami każdego z nich. Zauważ, że chociaż nie zostało to uwzględnione na tym diagramie, sieci neuronowe, takie jak autokodery, mogą być używane do uczenia nienadzorowanego.*

Odwrotnie, sieci neuronowe zawierające tylko warstwę wejściową i wyjściową są innym podzbiorem uczenia maszynowego, nazywanym *modelami liniowymi*. Modele liniowe reprezentują wzorce, których nauczyły się na podstawie danych przy użyciu funkcji liniowej. *Drzewa decyzyjne* to modele uczenia maszynowego, które używają danych do tworzenia podzbioru ścieżek z różnymi odgałęzieniami. Te odgałęzienia przybliżają rezultaty różnych wyników na podstawie danych. W końcu modele *klastrowania* szukają podobieństw między podzbiorami danych i używają tych rozpoznanych wzorców do grupowania danych w klastry.

Problemy uczenia maszynowego (patrz rysunek 1-1) można podzielić na dwa rodzaje: uczenie nadzorowane i nienadzorowane. *Uczenie nadzorowane* definiuje problemy, w których z góry znamy etykietę prawdy empirycznej dla danych. Na przykład może

to obejmować etykietowanie obrazu jako „kot" lub etykietowanie masy urodzeniowej dziecka jako 2.3 kg. Zasilamy swój model tymi zaetykietowanym danymi w nadziei, że może nauczyć się wystarczająco, aby etykietować nowe przykłady. Przy *uczeniu nienadzorowanym* nie znamy z góry etykiet dla danych, a celem jest zbudowanie modelu, który może znaleźć naturalne skupienia danych (nazywane *klastrowaniem*), skompresować treść informacji (*redukcja liczby wymiarów*) lub znaleźć reguły asocjacyjne. Większość tej książki skupia się na uczeniu nadzorowanym, ponieważ większość modeli uczenia maszynowego używanych w produkcji jest nadzorowanych. W uczeniu nadzorowanym problemy mogą typowo być definiowane jako klasyfikacja lub regresja. Modele *klasyfikacji* przypisują danym wejściowym etykietę (lub etykiety) z dyskretnego, predefiniowanego zbioru kategorii. Przykładami problemów klasyfikacyjnych są m.in., wyznaczanie rasy zwierzęcia na podstawie zdjęcia, tagowanie dokumentu lub przewidywanie, czy transakcja jest nieuczciwa. Modele *regresji* przypisują ciągłe, numeryczne wartości do danych wejściowych. Przykładami modeli regresji są m.in. przewidywanie czasu trwania wycieczki rowerowej, przyszłego dochodu firmy lub ceny produktu.

Inżynieria danych i cech

Dane są sercem dowolnego problemu uczenia maszynowego. Kiedy mówimy o *zbiorach danych*, odwołujemy się do danych używanych do trenowania, walidacji i testowania modelu uczenia maszynowego. Ogrom danych będzie *danymi uczącymi*: danymi zasilającymi model podczas procesu trenowania. *Dane walidacyjne* to dane, które pochodzą ze zbioru uczącego i służą do oceny działania modelu po każdej *epoce* uczenia (czyli przejściu przez dane uczące). Na podstawie działania modelu na danych walidacyjnych podejmowane są decyzje, kiedy skończyć przebieg trenowania i jak dobrać *hiperparametry*, czyli np. liczbę drzew w modelu lasu losowego. *Dane testowe* to dane, których nie używamy wcale w procesie uczenia i służą do oceny działania wytrenowanego modelu. Raporty wydajności modelu uczenia maszynowego muszą być obliczone na niezależnych danych testowych, a nie zbiorach uczących ani walidacyjnych. Ważne jest także, że dane są dzielone w taki sposób, że trzy zbiory danych (uczący, testowy, walidacyjny) mają podobne własności statystyczne.

Dane używane do uczenia modelu mogą mieć wiele form w zależności od typu modelu. Definiujemy *dane strukturalne* jako dane numeryczne i kategorialne. Dane numeryczne zawierają wartości w postaci liczb całkowitych i zmiennoprzecinkowych, a dane kategorialne zawierają dane, które można podzielić na skończoną liczbę grup, takie jak typ samochodu lub poziom edukacji. Można także myśleć o danych strukturalnych jako o danych, które można powszechnie znaleźć w arkuszu kalkulacyjnym. W tej książce będziemy używać terminu *dane tabelaryczne* wymienne z terminem dane strukturalne. *Dane niestrukturalne* obejmują natomiast dane, których nie możemy reprezentować tak wygodnie. Typowo obejmują one dowolne teksty, obrazy, wideo i audio.

Dane numeryczne mogą być często przekazywane bezpośrednio do modelu uczenia maszynowego, a inne dane wymagają różnorodnego *wstępnego przetwarzania danych*, zanim będą gotowe do wysłania do modelu. Ten etap wstępnego przetwarzania danych zawiera typowo skalowanie wartości liczbowych lub konwersję danych nienumerycznych

na format liczbowy, który model może zrozumieć. Innym terminem dla wstępnego przetwarzania danych jest *inżynieria cech*. Tych dwóch terminów będziemy wymiennie używać w tej książce.

Istnieją dwa różnie terminy służące do opisania danych, które przechodzą przez proces inżynierii cech. *Wejście* opisuje pojedynczą kolumnę w zbiorze danych, zanim zostanie przetworzona, a *cecha* opisuje pojedynczą kolumnę *po* jej przetworzeniu. Na przykład sygnatura czasowa byłaby wejściem, a cechą byłby dzień tygodnia. Aby konwertować dane sygnatury czasowej na dzień tygodnia, potrzebujemy pewnego przetworzenia danych. Ten krok przetwarzania może być nazywany także *przekształcaniem danych*.

Wystąpienie to element, który chcemy wysłać do modelu w celu predykcji. Wystąpienie może być wierszem w zbiorze testowym (bez kolumny etykiet), obrazem, który chcemy sklasyfikować, lub dokumentem tekstowym wysyłanym do modelu analizy sentymentu. Na podstawie zbioru cech dotyczących wystąpienia model oblicza prognozowaną wartość. W tym celu model jest trenowany na *przykładach uczących*, które kojarzą wystąpienie z *etykietą*. *Przykład uczący* oznacza pojedyncze wystąpienie (wiersz) danych ze zbioru danych, który będzie przekazywany do modelu. Dla przypadku użycia opierającym się na sygnaturze czasowej, pełen przykład uczący może zawierać: „dzień tygodnia", „miasto" oraz „typ samochodu". *Etykieta* to kolumna wyjściowa w zbiorze danych – element prognozowany przez model. *Etykieta* może odwoływać się zarówno do kolumny docelowej w zbiorze danych (nazywanej także *etykietą prawdy empirycznej*), jak i do danych wyjściowych modelu (nazywanych także *predykcją*). Etykietą próbki dla przykładu uczącego nakreślonego powyżej mogłoby być „czas trwania wycieczki" – w tym przypadku liczba zmiennoprzecinkowa oznaczająca minuty.

Po skompletowaniu zbioru danych i wyznaczeniu cech dla modelu następuje *walidacja danych*, czyli proces obliczania statystyk dotyczących danych, interpretacja schematu oraz ocena zbioru danych w celu rozpoznania problemów, takich jak dryfowanie i skośność zbioru uczącego. Ocena różnych statystyk może pomóc w zapewnieniu, że zbiór danych zawiera zrównoważoną reprezentację każdej cechy. W przypadkach, gdzie nie jest możliwe zebranie większej liczby danych, interpretacje zrównoważenia danych może pomóc zaprojektować model, który bierze to pod uwagę. Interpretacja schematu obejmuje definicję typu danych dla każdej cechy i rozpoznanie przykładów uczących, gdzie pewne wartości mogą być nieprawidłowe lub brakujące. W końcu walidacje danych mogą rozpoznać niespójności, które mogą wpływać na jakość zbiorów uczącego i testowego. Przykładem może być sytuacja, gdy większość danych zbioru uczącego zawiera próbki dla *dni roboczych*, a zbiór testowy zawiera głównie przykłady z *weekendu*.

Proces uczenia maszynowego

Pierwszym krokiem w typowym przepływie pracy uczenia maszynowego jest *trenowanie* – proces przekazywania danych uczących do modelu, aby mógł uczyć się w celu rozpoznawania wzorców. Po uczeniu następnym krokiem procesu jest testowanie, jak model działa na danych spoza zbioru uczącego. Jest to nazywane *ewaluacją* modelu. Możemy uruchamiać trenowanie i ewaluację wiele razy, dokonując dodatkowej inżynierii cech

i dopracowując architekturę modelu. Kiedy jesteśmy zadowoleni z działania modelu podczas ewaluacji, będziemy chcieć serwować model, aby inni mogli uzyskiwać od niego dostęp w celu dokonywania predykcji. Używamy terminu *serwowanie*, aby oznaczyć przyjmowanie przychodzących żądań i wysyłanie z powrotem predykcji poprzez wdrożenie modelu jako mikrousługi. Infrastruktura serwowania może znajdować się w chmurze, w lokalnym centrum danych lub na urządzeniu.

Proces wysyłania nowych danych do modelu i wykorzystywania jego danych wyjściowych jest nazywane *predykcją*. Może to dotyczyć zarówno generowania predykcji z modeli lokalnych, które nie były jeszcze wdrożone, jak i uzyskiwanie predykcji z modeli wdrożonych. W przypadku modeli wdrożonych będziemy się odwoływać zarówno do predykcji online, jak i predykcji partiami. *Predykcja online* jest używana, gdy chcemy otrzymać predykcje dla niewielu przykładów w czasie niemal rzeczywistym. W przypadku predykcji online nacisk jest położony na niskie opóźnienie. *Predykcja partiami* natomiast odwołuje się do generowania predykcji na dużym zbiorze danych w trybie offline. Zadania predykcji partiami trwają dłużej niż predykcja online i są przydatne do wstępnego obliczania predykcji (np. w systemach rekomendacji) oraz w analizie predykcji modelu na dużej próbie nowych danych.

Słowo *predykcja* jest trafne, gdy chodzi o przewidywanie przyszłych wartości, takich jak przewidywanie czasu trwania wycieczki rowerowej lub przewidywanie, czy koszyk sklepowy zostanie porzucony. Jest ono mniej intuicyjne w przypadku modeli klasyfikacji obrazu i tekstu. Jeśli model ML na podstawie recenzji teksu daje na wyjściu informację, że sentyment jest pozytywny, nie jest to w rzeczywistości „predykcja" (nie ma żadnego przyszłego wyniku). Stąd będziemy używać także słowa *wnioskowanie* w odniesieniu do predykcji. Jest tutaj wykorzystany statystyczny termin wnioskowanie, chociaż nie oznacza faktycznego rozumowania.

Proces zbierania danych, inżynieria cech, trenowanie i ewaluacja modelu są często obsługiwane oddzielnie od potoku produkcyjnego. W takim przypadku będziemy ponownie ewaluować rozwiązanie za każdym razem, gdy zdecydujemy, że mamy wystarczająco dużo dodatkowych danych do wytrenowania nowej wersji modelu. W innych sytuacjach możemy mieć nowe dane przyswajane w sposób ciągły i potrzebujemy przetwarzać te dane natychmiast przed wysłaniem ich do modelu w celu uczenia lub predykcji. Jest to nazywane *przesyłaniem strumieniowym*. Aby obsłużyć przesyłanie strumieniowe danych, potrzebujemy wieloetapowego rozwiązania w celu wykonywana inżynierii cech, trenowania, ewaluacji i predykcji. Takie wieloetapowe rozwiązania nazywane są *potokami ML*.

Narzędzia do danych i modelowania

Będziemy odwoływać się do różnorodnych produktów Google Cloud, które zapewniają narzędzia do rozwiązywania problemów związanych z danymi i uczeniem maszynowym. Te produkty są zaledwie jedną z opcji implementowania wzorców projektowych, do których będziemy odwoływać się w tej książce, i nie mają stanowić wyczerpującej listy. Wszystkie produkty uwzględnione tutaj są bezserwerowe, co pozwala nam skupić

się bardziej na implementacji wzorców projektowych uczenia maszynowego, a nie na infrastrukturze, która za nimi stoi.

BigQuery[3] to hurtownia danych klasy przedsiębiorstwa, zaprojektowana do analizowania dużych zbiorów danych za pomocą języka SQL. Będziemy używać BigQuery w naszych przykładach do zbierania danych i inżynierii cech. Dane w BigQuery są uporządkowane w zbiory danych, a zbiory danych mogą mieć wiele tabel. Wiele z naszych przykładów będzie używać danych z witryny *Google Cloud Public Datasets*[4], zestawu darmowych, publicznie dostępnych danych hostowanych w BigQuery. Google Cloud Public Datasets składa się z setek różnych zbiorów danych, w tym danych pogody NOAA od roku 1929, pytań i odpowiedzi z witryny Stack Overflow, otwartego kodu źródłowego z witryny GitHub, danych urodzeniowych itp. Do budowania niektórych modeli w naszych przykładach będziemy używać *BigQuery Machine Learning*[5], czyli BigQuery ML. BigQuery ML to narzędzie do budowania modeli z danych przechowywanych w BigQuery. Dzięki BigQuery ML możemy trenować, ewaluować i generować predykcje na naszych modelach przy użyciu języka SQL. Big Query ML obsługuje modele klasyfikacji i regresji, wraz z nienadzorowanymi modelami klastrowania. Jest także możliwe importowanie wcześniej wytrenowanych modeli TensorFlow do BigQuery ML w celu predykcji.

Witryna *Cloud AI Platform*[6] obejmuje różnorodne produkty do trenowania i serwowania niestandardowych modeli uczenia maszynowego na platformie Google Cloud. W naszych przykładach będziemy używać usług AI Platform Training oraz AI Platform Prediction. AI Platform Training zapewnia infrastrukturę do trenowania modelu uczenia maszynowego w Google Cloud. Dzięki AI Platform Prediction można wdrażać własne wytrenowane modele i generować na nich predykcje przy użyciu API. Obie usługi obsługują modele TensorFlow, scikit-learn oraz XGBoost, wraz z niestandardowymi kontenerami dla modeli zbudowanych w innych frameworkach. Odwołujemy się także do *Explainable AI*[7], czyli narzędzia do interpretacji wyników predykcji modeli, dostępnego dla modeli wdrożonych w AI Platform.

Role

W organizacjach występuje wiele różnych ról związanych z danymi i uczeniem maszynowym. Poniżej definiujemy kilka popularnych ról, do których odwołujemy się często w książce. Niniejsza książka jest przeznaczona głównie dla naukowców zajmujących się danymi, inżynierów danych oraz inżynierów ML, zatem od nich zaczniemy.

Naukowiec zajmujący się danymi (*data scientist*), w skrócie *naukowiec danych*, to ktoś skupiony na zbieraniu, interpretowaniu i przetwarzaniu zbiorów danych. Naukowcy uruchamiają statystyczne i objaśniające analizy danych. W związku z uczeniem maszynowym

3 *https://oreil.ly/7PnVj*
4 *https://oreil.ly/AbTaJ*
5 *https://oreil.ly/_VjVz*
6 *https://oreil.ly/90KLs*
7 *https://oreil.ly/lDocn*

naukowiec danych może pracować nad zbieraniem danych, inżynierią cech, konstruowaniem modeli itp. Naukowcy danych często pracują w językach Python lub R w środowisku notesu i zwykle są pierwszymi osobami konstruującymi modele uczenia maszynowego w organizacji.

Inżynier danych skupia się na infrastrukturze i przepływach pracy zasilających dane w organizacji. Może pomagać w zarządzaniu sposobem przyjmowania danych przez firmę, potokami danych oraz sposobem przechowywania i przesyłania danych. Inżynierowie danych implementują infrastrukturę i potoki wokół danych.

Inżynierowie uczenia maszynowego wykonują zadania podobne, jak inżynierowie danych, ale dla modeli ML. Biorą modele opracowane przez naukowców zajmujących się danymi i zarządzają infrastrukturą i operacjami wokół uczenia i wdrażania tych modeli. Inżynierowie ML pomagają budować systemy produkcyjne w celu obsługi aktualizacji modeli, wersjonowania modeli oraz serwowania predykcji dla użytkowników końcowych.

Im mniejszy jest zespół nauki o danych w firmie i im bardziej zwinny jest ten zespół, tym bardziej prawdopodobne jest, że ta sama osoba odgrywa wiele ról. Jeśli jesteś w takiej sytuacji, bardzo prawdopodobne jest, że czytając powyższe trzy opisy dostrzegasz swój częściowy udział we wszystkich trzech kategoriach. Możesz zaczynać projekt uczenia maszynowego jako inżynier danych i budować potoki danych w celu tworzenia operacji przyjmowania danych. Następnie przemieniasz się w naukowca danych i konstruujesz modele ML. W końcu nakładasz kapelusz inżyniera ML i wprowadzasz model do środowiska produkcyjnego. W większych organizacjach projekty uczenia maszynowego będą przechodzić przez te same etapy, ale z różnymi zespołami zajmującymi się każdym etapem.

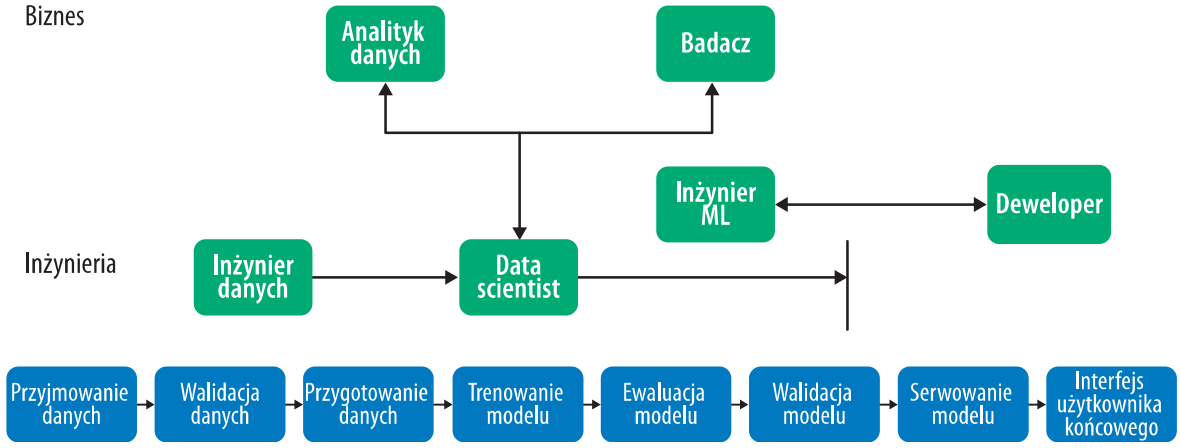
Naukowcy-badacze, analitycy danych i deweloperzy również mogą budować modele AI i ich używać, ale te role zawodowe nie są grupą docelową tej książki.

Naukowcy-badacze (*research scientist*) skupiają się głównie na znajdowaniu i rozwijaniu nowych algorytmów, wprowadzając postęp w dziedzinie ML. Może to obejmować różnorodne dziedziny podrzędne uczenia maszynowego, takie jak architektury modelu, przetwarzania języka naturalnego, rozpoznawanie obrazów, strojenie hiperparametrów, interpretowalność modeli itp.. W przeciwieństwie do innych ról opisanych tutaj, naukowcy-badacze spędzają większość czasu na prototypowaniu i ewaluacji nowych podejść do uczenia maszynowego, zamiast konstruować produkcyjne systemy ML.

Analitycy danych ewaluują i zbierają spostrzeżenia wynikające z danych, a następnie podsumowują te spostrzeżenia dla innych zespołów w swojej organizacji. Mają tendencję do pracy w języku SQL i arkuszach kalkulacyjnych oraz do używania narzędzi inteligencji biznesowej do tworzenia wizualizacji w celu udostępniania swoich odkryć. Analitycy danych blisko współpracują z zespołami produktowymi, aby zrozumieć, jak ich spostrzeżenia mogą pomóc rozwiązać problemy biznesowe i wytworzyć wartość. Podczas gdy analitycy danych skupiają się na identyfikacji tendencji w istniejących danych i wyprowadzania z nich spostrzeżeń, naukowcy danych skupiają się na używaniu tych danych do generowania przyszłych predykcji i automatyzacji lub skalowania generowania spostrzeżeń. Wraz z wzrastającą demokratyzacją uczenia maszynowego analitycy danych podnoszą swoje umiejętności, aby stać się naukowcami danych.

Deweloperzy są odpowiedzialni za budowanie systemów produkcyjnych, które pozwalają użytkownikom końcowym uzyskiwać dostęp do modeli ML. Często zajmują się projektowaniem interfejsów API, które odpytują modele i zwracają predykcje w przyjaznym dla użytkownika formacie przez aplikację webową lub mobilną. To może obejmować modele hostowane w chmurze lub modele serwowane na urządzeniu. Deweloperzy wykorzystują infrastrukturę udostępniania usług modelu implementowaną przez inżynierów ML do budowania aplikacji i interfejsów użytkownika w celu serwowania predykcji użytkownikom modelu.

Rysunek 1-2 ilustruje, jak te różnie role współpracują ze sobą w całym procesie rozwoju modelu uczenia maszynowego w organizacji.



Rysunek 1-2 *Istnieje wiele różnych ról zawodowych związanych z danymi i uczeniem maszynowym, a te role współpracują nad przepływem pracy ML, od przyjmowania danych do serwowania modelu oraz interfejsu użytkownika końcowego. Na przykład inżynier danych pracuje nad przyjmowaniem danych i walidacją danych i współpracuje blisko z naukowcami danych.*

Typowe wyzwania w uczeniu maszynowym

Dlaczego potrzebujemy książki o wzorcach projektowych uczenia maszynowego? Proces konstruowania systemów ML prezentuje różnorodne unikalne wyzwania, które wpływają na projektowanie ML. Zrozumienie tych wyzwań pomoże praktykom ML rozwinąć układ odniesienia dla rozwiązań wprowadzonych w treści książki.

Jakość danych

Modele uczenia maszynowego są tylko tak wiarygodne, jak dane użyte do ich trenowania. Jeśli model uczenia maszynowego jest trenowany na niekompletnym zbiorze danych ze słabo wybranymi cechami lub na danych, które nie reprezentują dokładnie populacji używającej modelu, wtedy predykcje modelu będą bezpośrednio odzwierciedlać te dane. W efekcie modele uczenia maszynowego są często określane przy użyciu popularnego

powiedzenia „śmieci na wejściu, śmieci na wyjściu". Dlatego trzeba podkreślić cztery ważne składowe jakości danych: dokładność, kompletność, spójność i aktualność.

Dokładność danych odnosi się zarówno do cech zbioru uczącego, jak i etykiet prawdy empirycznej odpowiadających tym cechom. Zrozumienie pochodzenia danych i ewentualnych potencjalnych błędów w procesie zbierania danych może pomóc w zapewnieniu dokładności cech. Po zebraniu danych ważne jest przeprowadzenie dokładnej analizy, aby wykryć literówki, duplikaty wpisów, niespójności miar w danych tabelarycznych, brakujące cechy i ewentualne inne błędy, które mogą wpływać na jakość danych. Na przykład duplikaty w zbiorze uczącym mogą powodować, że model będzie nieprawidłowo przypisywał większą wagę do tych punktów danych.

Dokładne etykiety danych są równie ważne, jak dokładność cech. Model polega wyłącznie na etykietach prawdy empirycznej w danych uczących, aby aktualizować swoje wagi i minimalizować stratę. W efekcie nieprawidłowo zaetykietowane przykłady uczące mogą prowadzić do mylącej dokładności modelu. Powiedzmy na przykład, że konstruujemy model analizy sentymentu, a 25% „pozytywnych" przykładów uczących zostało błędnie zaetykietowanych jako „negatywne". Model będzie miał niedokładny obraz tego, co powinien uważać za sentyment negatywny, co zostanie bezpośrednio odzwierciedlone w jego predykcjach.

Aby zrozumieć *kompletność* danych, powiedzmy, że trenujemy model, aby rozpoznawał rasy kotów. Trenujemy model na wyczerpującym zbiorze danych zdjęć kotów, a wynikowy model jest w stanie klasyfikować obraz do 1 z 10 możliwych kategorii („bengalski", „syjamski" itd.) z 99% dokładnością. Po wdrożeniu modelu do produkcji, zauważasz, że oprócz przekazywania zdjęć kotów do klasyfikacji, wielu użytkowników przekazuje zdjęcia psów i jest rozczarowanych wynikami modelu. Ponieważ model został wytrenowany do rozpoznawania 10 różnych ras kotów, jest to wszystko, co potrafi. Te 10 kategorii ras jest istotnie całym „widzianym światem" modelu. Cokolwiek wyślesz do modelu, możesz oczekiwać, że zostanie zakwalifikowane do jednej z tych 10 kategorii. Może nawet zrobić to z wysoką pewnością dla obrazu, który nawet nie przypomina kota. Ponadto nie ma sposobu, aby model zwrócił odpowiedź „to nie kot", jeśli takie dane i etykieta nie zostały uwzględnione w zbiorze uczącym.

Innym aspektem kompletności danych jest zapewnienie, że dane uczące zawierają różne reprezentacje każdej z etykiet. W przykładzie wykrywania rasy kotów, jeśli wszystkie obrazy są zbliżeniami pyska kota, model nie będzie w stanie rozpoznawać obrazu kota z boku ani całej postaci kota. Aby spojrzeć na przykład danych tabelarycznych, jeśli konstruujemy model do prognozowania ceny nieruchomości w konkretnym mieście, ale zamieścimy tylko przykłady uczące domów większych niż 2 000 stóp kwadratowych (około 185 m^2), wynikowy model będzie działał słabo dla mniejszych domów.

Trzecim aspektem jakości danych jest ich *spójność*. W przypadku dużych zbiorów danych typowo dzieli się pracę nad zbieraniem i etykietowaniem danych pomiędzy grupę osób. Wypracowanie zbioru standardów dla tego procesu może pomóc zapewnić spójność w zbiorze danych, ponieważ każda osoba do tego zatrudniona będzie nieuchronnie wnosić własne obciążenia do procesu. Podobnie jak w przypadku kompletności, niespójności

danych można znaleźć zarówno w cechach, jak i etykietach. Aby pokazać przykład niespójnych cech, powiedzmy, że zbieramy dane atmosferyczne z czujników temperatury. Jeśli każdy czujnik został skalibrowany w innym standardzie, będzie to skutkować niedokładanymi i niewiarygodnymi predykcjami modelu. Niespójności mogą odnosić się także do formatu danych. Jeśli przechwytujemy dane lokalizacji, niektóre osoby mogą napisać pełen adres ulicy jako „ulica Długa", a inni skrócić go do „ul. Długa". Jednostki miar także mogą się różnić, czego przykładem są mile i kilometry.

Jeśli chodzi o niespójności etykietowania, wróćmy do przykładu analizy sentymentu. W tym przypadku jest prawdopodobne, że podczas etykietowania danych uczących różne osoby nie zawsze zgadzają się, co jest uważane z pozytywne, a co za negatywne. Aby to rozwiązać, można kazać wielu osobom zaetykietowanie każdego przykładu w zbiorze danych, a następnie wybrać najczęściej stosowane etykiety dla poszczególnych elementów. Świadomość potencjalnego obciążenia osoby etykietującej oraz implementowanie systemów biorących to pod uwagę, zapewni spójność danych w zbiorze. Zbadamy koncepcję obciążenie w punkcie „Wzorzec projektowy 30: Rzetelny obiektyw" w rozdziale 7.

Aktualność danych odnosi się do opóźnienia między czasem wystąpienia zdarzenia a dodaniem go do bazy danych. Jeśli na przykład zbieramy dane w dziennikach aplikacji, dziennik błędów może pojawić się dopiero po kilku godzinach w bazie danych dzienników. W przypadku zbioru danych rejestrującego transakcje kartami kredytowymi od wystąpienia transakcji do jej zarejestrowania w systemie może upłynąć cały dzień. Aby radzić sobie z aktualnością, przydatne jest rejestrowanie możliwie największej ilości informacji o konkretnych punktach danych i upewnianie się, że te informacje są odzwierciedlane podczas przekształcania danych w cechy dla modelu uczenia maszynowego. Mówiąc dokładniej, można śledzić sygnatury czasowe wystąpienia zdarzenia oraz dodania go do zbioru danych. Następnie podczas przeprowadzania inżynierii danych można odpowiednio uwzględnić te różnice.

Odtwarzalność

W tradycyjnym programowaniu dane wyjściowe programu są powtarzalne i gwarantowane. Na przykład, jeśli piszemy program w języku Python, który odwraca ciąg znaków, wiemy, że dla podanego na wejściu słowa „banan" zawsze otrzymamy na wyjściu „nanab". Podobnie, jeśli w programie jest błąd powodujący nieprawidłowe odwracanie ciągów zawierających liczby, możemy wysłać program do kolegi i oczekiwać, że będzie w stanie odtworzyć ten błąd z tymi samymi danymi wyjściowymi, których użyliśmy (o ile błąd nie ma czegoś wspólnego z utrzymywaniem przez program jakiegoś nieprawidłowego stanu wewnętrznego, różnic w architekturze, takich jak precyzja liczb zmiennoprzecinkowych lub różnic w działaniu, związanych np. z wątkami).

Z drugiej strony, modele uczenia maszynowego zawierają nieodłączny element losowości. Podczas trenowania modelu ML wagi są inicjowane wartościami losowymi. Te wagi następnie podlegają konwergencji podczas trenowania, gdy model iteruje i uczy się na podstawie danych. Z tego powodu ten sam kod modelu z podanymi tymi samymi

danymi uczącymi będzie wytwarzać w przebiegach trenowania trochę różniące się wyniki. Wprowadza to problem odtwarzalności. W przypadku trenowania modelu do 98.1% dokładności, powtórzony przebieg trenowania nie gwarantuje osiągnięcia tego samego wyniku. Sprawia to trudności w porównywaniu różnych eksperymentów.

W celu rozwiązania problemu odtwarzalności często ustawia się wartość ziarna losowego używanego przez model w celu zapewniania, że ta sama losowość zostanie zastosowana za każdym przebiegiem treningu. W TensorFlow możemy to zrobić przez uruchomienie instrukcji `tf.random.set_seed(value)` na początku programu.

Dodatkowo w scikit-learn wiele funkcji narzędziowych do tasowania danych pozwala również na ustawianie wartości ziarna losowego:

```
from sklearn.utils import shuffle
data = shuffle(data, random_state=value)
```

Należy pamiętać, że trzeba użyć tych samych danych *oraz* tego samego ziarna losowego podczas trenowania modelu w celu zapewnienia powtarzalnych, odtwarzalnych wyników w różnych eksperymentach.

Trenowanie modelu ML obejmuje kilka artefaktów, które muszą być ustalone w celu zapewnienia odtwarzalności: użyte dane, mechanizm podziału używany do generowania zbiorów danych uczącego i walidacyjnego, przygotowanie danych i hiperparametry modelu, a także takie zmienne, jak rozmiar partii i harmonogram współczynnika uczenia.

Odtwarzalność dotyczy również zależności frameworków uczenia maszynowego. Oprócz ręcznego ustawiania ziarna losowego frameworki implementują także wewnętrznie elementy losowości, które są wykonywane przy wywoływaniu funkcji do trenowania modelu. Jeśli ta wewnętrzna implementacja zmienia się pomiędzy różnymi wersjami frameworków, odtwarzalność nie jest gwarantowana. Konkretnym przykładem jest sytuacja, gdy jedna wersja metody `train()` frameworku wykonuje 13 wywołań `rand()`, a nowsza wersja tego samego frameworku wykonuje 14 wywołań. Wtedy użycie różnych wersji w różnych eksperymentach spowoduje nieco inne wyniki, nawet z tymi samymi danymi i kodem modelu. Uruchamianie obciążeń pracą ML w kontenerach i standaryzowanie wersji bibliotek może pomóc w zapewnieniu odtwarzalności. W rozdziale 6 przedstawiamy serię wzorców zapewniających odtwarzalność procesów ML.

W końcu odtwarzalność może odnosić się do środowiska do trenowania modelu. Często z powodu dużych zbiorów danych i złożoności, wiele modeli potrzebuje znacznej ilości czasu do wytrenowania. Można to przyspieszyć, stosując takie strategie dystrybucji, jak równoległość danych lub modelu (patrz rozdział 5). Z takim przyspieszeniem wiąże się jednak dodatkowy problem powtarzalności, gdy ponownie uruchamiamy kod, który wykorzystuje rozproszone trenowanie.

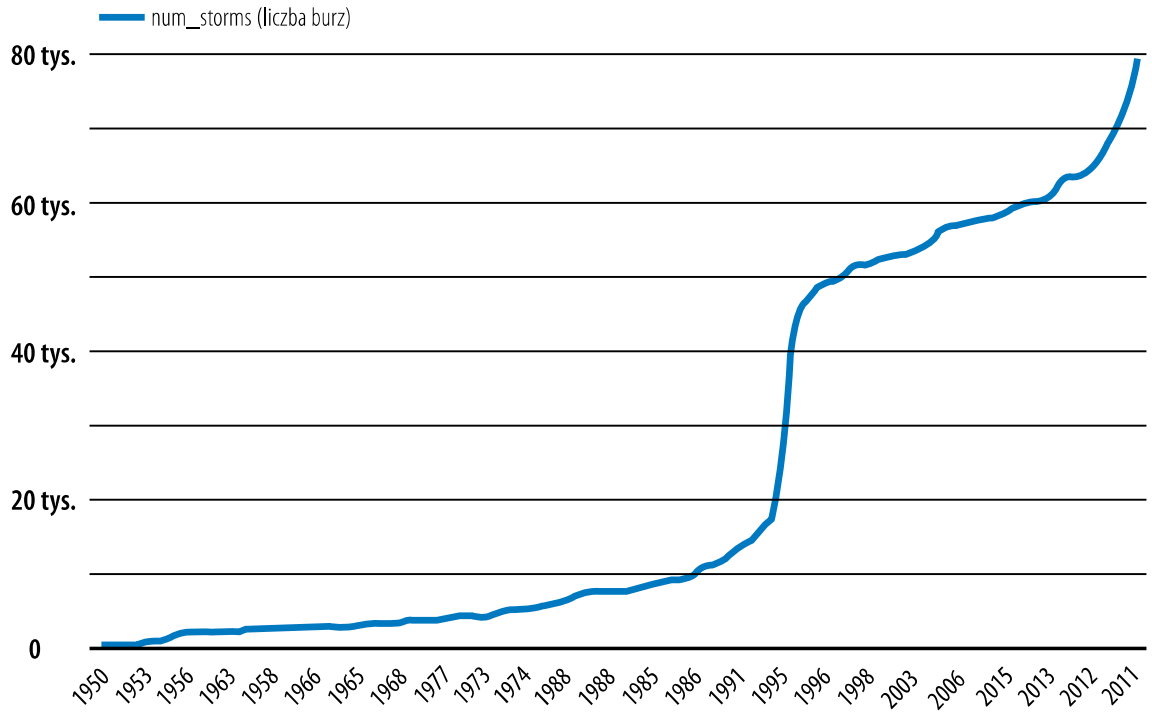
Dryfowanie danych

O ile modele uczenia maszynowego typowo reprezentują statyczne związki między danymi wejściowymi a wyjściowymi, dane mogą zmieniać się znacznie z upływem czasu. Dryfowanie danych odnosi się do problemu zapewnienia, że modele uczenia maszynowego

pozostają odpowiednie, a predykcje modelu są dokładnym odzwierciedleniem środowiska, w którym są używane.

Na przykład powiedzmy, że trenujemy model do klasyfikacji nagłówków artykułów informacyjnych do takich kategorii, jak „polityka", „biznes" oraz „technologia". Jeśli trenujemy i ewaluujmy model na historycznych artykułach informacyjnych z XX w., prawdopodobnie nie będzie on działać równie dobrze na bieżących danych. Dzisiaj wiemy, że artykuł ze słowem „smartfon" w nagłówku prawdopodobnie dotyczy technologii. Jednak model trenowany na danych historycznych nie miałby żadnej wiedzy na temat tego słowa. W celu rozwiązania tego dryfowania ważne jest stałe aktualizowanie zbioru uczącego, ponowne trenowanie modelu i modyfikowanie wag nieprzypisywanych przez model do konkretnych grup danych wejściowych.

Aby zobaczyć mniej oczywisty przykład dryfowania, przyjrzymy się zbiorowi danych NOAA[8] w BigQuery, który dotyczy silnych burz. W przypadku trenowania modelu do prognozowania prawdopodobieństwa burzy na danym obszarze, trzeba wziąć pod uwagę sposób, w jaki raportowanie pogody zmieniało się z czasem[9]. Możemy zobaczyć na rysunku 1-3, że liczba rejestrowanych silnych burz zwiększała się stale od roku 1950.



Rysunek 1-3 *Liczba silnych burz raportowanych rocznie, zgodnie z rejestrem NOAA od roku 1950 do 2011.*

Z tego trendu możemy zobaczyć, że trenowanie modelu na danych sprzed roku 2000 do generowania predykcji burz w dzisiejszych czasach prowadziłoby do niedokładnych

8 *https://oreil.ly/obzvn*

9 *https://github.com/GoogleCloudPlatform/ml-designpatterns/blob/master/01_need_for_design_patterns/ml_challenges.ipynb*

prognoz. Oprócz rosnącej liczby raportowanych burz, ważne jest także rozważenie innych czynników, które mogły wpływać na dane z rysunku 1-3. Na przykład technologia obserwowania burz poprawiła się z upływem czasu, najmocniej z wprowadzeniem radarów pogodowych w latach 90. XX w. W kontekście cech może to oznaczać, że nowe dane zawierają więcej informacji o każdej burzy, a cecha dostępna w dzisiejszych danych mogła nawet nie być obserwowana w roku 1950. Objaśniająca analiza danych może pomóc rozpoznać ten typ dryfowania i wskazać prawidłowe okno danych do użycia w celu trenowania. W punkcie „Wzorzec projektowy 23: Schemat mostkowy" w rozdziale 6 przedstawiono sposób obsługi zbiorów danych, w których dostępność cech poprawia się z upływem czasu.

Skala

Problem skalowania jest obecny na wielu etapach typowego przepływu pracy uczenia maszynowego. Prawdopodobnie napotkamy problemy skalowania w zbieraniu danych i wstępnym przetwarzaniu, trenowaniu i serwowaniu. Podczas przyswajania i przygotowywania danych do modelu uczenia maszynowego rozmiar zbioru danych będzie dyktował narzędzia wymagane do użycia w rozwiązaniu. Często zadaniem inżynierów danych jest konstruowanie potoków danych, które mogą skalować się do obsługi milionów wierszy.

W przypadku trenowania modelu inżynierowie ML są odpowiedzialni za wyznaczanie niezbędnej infrastruktury do konkretnego zadania trenowania. W zależności od typu i rozmiaru zbioru danych, trenowanie modelu może zajmować dużo czasu lub być kosztowne obliczeniowo, wymagać infrastruktury zaprojektowanej specjalnie do obciążeń pracą ML (np. procesorów graficznych – GPU). Na przykład modele z obrazami wymagają zwykle znacznie więcej infrastruktury trenowania, niż modele trenowane całkowicie na danych tabelarycznych.

W kontekście świadczenia usług modelu, infrastruktura wymagana do obsługi zespołu naukowców danych uzyskujących predykcje z prototypu modelu jest zupełnie odmienna od infrastruktury niezbędnej do obsługi modelu produkcyjnego, otrzymującego miliony żądań predykcji co godzinę. Deweloperzy i inżynierowie ML typowo są odpowiedzialni za obsługę problemów skalowania, związanych z wdrażaniem modelu i serwowaniem żądań predykcji.

Większość wzorców ML w tej książce jest przydatnych bez względu na dojrzałość organizacji. Jednak kilka wzorców w rozdziałach 6 i 7 w różny sposób dotyczy problemów elastyczności i odtwarzalności, a wybór między nimi sprowadza się do przypadku użycia i zdolności organizacji do asymilacji złożoności.

Wiele celów

Chociaż często zdarza się, że pojedynczy zespół jest odpowiedzialny za konstruowanie modelu uczenia maszynowego, wiele zespołów w organizacji będzie wykorzystywać ten model w pewien sposób. Nieuchronnie te zespoły mogą mieć różne pomysły na to, co definiuje udany model.

Aby zrozumieć, jak można to rozegrać w praktyce, powiedzmy, że budujemy model do rozpoznawania uszkodzonych produktów na podstawie zdjęć. Celem naukowca danych jest minimalizacja straty entropii krzyżowej modelu. Menedżer produktu natomiast może chcieć zredukować liczbę uszkodzonych produktów, które zostały błędnie sklasyfikowane i wysłane do klientów. Na koniec celem zespołu zarządzającego może być zwiększenie dochodu o 30%. Każdy z tych celów różni się pod względem tego, co jest optymalizowane, a równoważenie tych różnych potrzeb w organizacji może stanowić wyzwanie.

Naukowiec danych mógłby przełożyć potrzeby zespołu produktu w kontekście modelu, mówiąc, że fałszywe wyniki negatywne są pięć razy bardziej kosztowne niż fałszywe wyniki pozytywne. Dlatego należy optymalizować pod kątem czułości, przed precyzją, w celu spełnienia tego założenia podczas projektowania modelu. Następnie możemy znaleźć równowagę między celem zespołu produktowego (optymalizacji precyzji) a własnym celem minimalizacji straty modelu.

Podczas definiowania celów dla modelu ważne jest rozważenie potrzeb różnych zespołów w organizacji oraz sposobu przenoszenia potrzeb każdego zespołu z powrotem do modelu. Przez analizę tego, co każdy zespół optymalizuje, jeszcze przed skonstruowaniem rozwiązania można znaleźć obszary kompromisu, aby optymalnie wyważyć te różne cele.

Podsumowanie

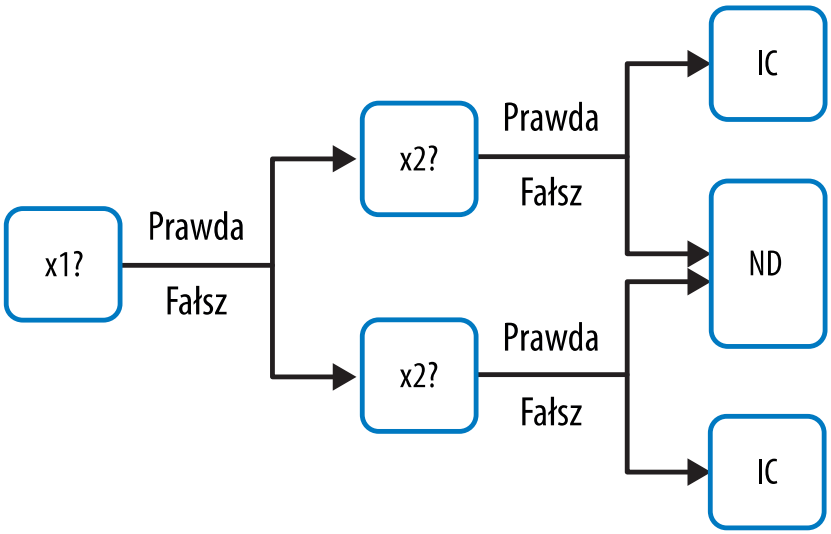
Wzorce projektowe są sposobem kodyfikowania wiedzy i doświadczenia ekspertów w postaci rady, którą wszyscy praktycy mogą wykorzystywać. Wzorce projektowe w tej książce ujmują najlepsze praktyki i rozwiązania często występujących problemów w projektowaniu, budowaniu i wdrażaniu systemów uczenia maszynowego. Najczęstsze problemy w uczeniu maszynowym dotyczą jakości danych, odtwarzalności, dryfowania danych, skalowania i konieczności spełnienia wielu celów.

Mamy tendencję do używania różnych wzorców ML na różnych etapach cyklu życia ML. Istnieją wzorce, które są przydatne w formułowaniu problemu i ocenie wykonalności. Większość wzorców dotyczy etapów wytwarzania lub wdrażania, a tylko nieliczne zajmują się wzajemnymi zależnościami między tymi etapami.

Rozdział 2

Wzorce projektowe reprezentacji danych

Sercem dowolnego modelu uczenia maszynowego jest funkcja matematyczna, która z definicji operuje tylko na konkretnych typach danych. Jednocześnie modele uczenia maszynowego w świecie rzeczywistym muszą operować na danych, które niekoniecznie nadają się do bezpośredniego wprowadzania do funkcji matematycznej. Na przykład matematyczny rdzeń drzewa decyzyjnego operuje na zmiennych logicznych. Zauważ, że mówimy tutaj o matematycznym rdzeniu drzewa decyzyjnego – oprogramowanie uczenia maszynowego drzewa decyzyjnego będzie typowo zawierało także funkcje uczenia optymalnego drzewa na podstawie danych i sposoby odczytywania i przetwarzania różnych typów danych liczbowych i kategorialnych. Jednak funkcja matematyczna (patrz rysunek 2-1), która stanowi fundament drzewa decyzyjnego, operuje na zmiennych logicznych i używa takich operacji jak AND (&& na rysunku 2-1) oraz OR (+ na rysunku 2-1).



$$f(x1, x1) = ((1 \;\&\&\; x2) + (!x1 \;\&\&\; !x2))*IC + ((x1 \;\&\&\; !x2) + (!x1 \;\&\&\; x2))*ND$$

Rysunek 2-1 *Sercem modelu uczenia maszynowego drzewa decyzyjnego służącego do prognozowania, czy dziecko wymaga intensywnej opieki, jest modelem matematycznym, który operuje na zmiennych logicznych.*

Załóżmy, że mamy drzewo decyzyjne do prognozowania, czy dziecko wymaga intensywnej opieki (IC) lub czy może otrzymać normalny wypis (ND), a drzewo decyzyjne przyjmuje na wejściu dwie zmienne, *x1* oraz *x2*. Wytrenowany model może wyglądać podobnie, jak na rysunku 2-1.

Jest dość oczywiste, że zmienne *x1* i *x2* muszą być zmiennymi logicznymi, aby funkcja *f(x1, x2)* działała. Załóżmy, że dwie informacje brane pod uwagę podczas klasyfikacji dziecka jako wymagającego lub niewymagającego intensywnej opieki, to szpital, w którym urodziło się dziecko, i masa dziecka. Czy możemy użyć szpitala, w którym urodziło się dziecko, jako wejścia do drzewa decyzyjnego? Nie, ponieważ szpital nie przyjmuje wartości Prawda ani Fałsz i nie może być podany do operatora && (AND). Jest matematycznie niezgodny. Oczywiście możemy „przerobić" wartość szpitala na wartość logiczną za pomocą takiej operacji:

```
x1 = (hospital IN France)
```

zatem *x1* ma wartość Prawda, gdy szpital jest we Francji, a wartość Fałsz, jeśli nie jest. Podobnie masa dziecka nie może być podana bezpośrednio do modelu, ale dzięki wykonaniu takiej operacji:

```
x2 = (babyweight < 3 kg)
```

możemy użyć szpitala lub masy dziecka jako wejścia do modelu. Jest to przykład, jak dane wejściowe (szpital, czyli złożony obiekt, lub masa dziecka, czyli liczba zmiennoprzecinkowa) mogą być reprezentowane w formie (logicznej) oczekiwanej przez model. To właśnie określamy *reprezentacją danych*.

W tej książce będziemy używać terminu *wejście* na określenie reprezentacji danych świata rzeczywistego przekazywanych do modelu (na przykład ciężaru dziecka), a terminu *cecha* na określenie reprezentacji przekształconych danych, na których model faktycznie operuje (na przykład, czy masa dziecka jest mniejsza niż 3 kilogramy). Proces tworzenia cech do reprezentowania danych wejściowych nazywamy *inżynierią cech*, a zatem możemy myśleć o inżynierii cech jako o sposobie wyboru reprezentacji danych.

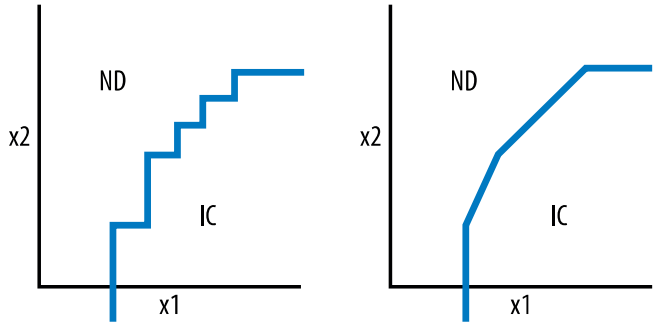
Oczywiście, zamiast kodować na sztywno parametry, takie jak wartość progowa 3 kilogramy, preferujemy, aby model uczenia maszynowego uczył się, jak tworzyć każdy z węzłów przez wybór zmiennej wejściowej i progu. Drzewa decyzyjne są przykładem modelu uczenia maszynowego, które są zdolne nauczenia się reprezentacji[1]. Wiele z wzorców, którym przyjrzymy się w tym rozdziale, obejmuje podobne *wyuczalne reprezentacje danych*.

Wzorzec projektowy *Osadzanie* jest kanonicznym przykładem reprezentacji danych, którego głębokie sieci neuronowe są w stanie nauczyć się samodzielnie. W osadzaniu nauczona reprezentacja danych jest gęsta i ma mniej wymiarów niż wejście, które może być rzadkie. Algorytm uczący potrzebuje wyodrębnić najbardziej istotne informacje na podstawie wejścia i reprezentować je w bardziej zwięzły sposób w postaci cechy. Proces uczenia cech do reprezentacji danych wejściowych jest nazywany *wyodrębnianiem cech*

1 Tutaj wyuczona reprezentacja danych składa się ze zmiennej wejściowej *baby weight* (waga dziecka), operatora *mniejszy niż* i progu 3 kg.

i możemy myśleć o wyuczalnych reprezentacjach danych (takich jak osadzanie) jako o automatycznej inżynierii cech.

Reprezentacja danych nie musi nawet być pojedynczą zmienną wejściową – ukośne drzewo decyzyjne na przykład tworzy cechy logiczne przez tworzenie progów liniowych kombinacji dwóch lub więcej zmiennych wejściowych. Drzewo decyzyjne, w którym każdy węzeł może reprezentować tylko jedną zmienną wejściową, redukuje się do schodkowej funkcji liniowej, a ukośne drzewo decyzyjne, gdzie każdy węzeł może reprezentować liniową kombinację zmiennych wejściowych, redukuje się do funkcji przedziałami liniowej (patrz rysunek 2-2). Biorąc pod uwagę liczbę kroków do wyuczenia w celu odpowiedniego reprezentowania linii, model przedziałami liniowy jest prostszy i szybszy do nauczenia. Rozszerzeniem tej idei jest wzorzec projektowy *Krzyżowanie cech*, który upraszcza uczenie relacji AND między wielowartościowymi zmiennymi kategorialnymi.



Rysunek 2-2 *Klasyfikator drzewa decyzyjnego, w którym każdy węzeł może stanowić próg dla tylko jednej wartości wejściowej (x1 lub x2), będzie skutkować schodkową liniową funkcją graniczną, a ukośny klasyfikator drzewa, gdzie węzeł może stanowić próg dla liniowej kombinacji zmiennych wejściowych, może dać w wyniku przedziałami liniową funkcję graniczną. Przedziałami linowa funkcja graniczna wymaga mniej węzłów i może osiągnąć większą dokładność.*

Reprezentacja danych nie musi być wyuczona ani ustalona – możliwe jest także rozwiązanie hybrydowe.Wzorzec projektowy *Cecha haszowana* jest deterministyczny, ale nie wymaga, aby model znał potencjalne wartości, które może przyjmować konkretne wejście.

Wszystkie reprezentacje danych, którym przyglądaliśmy się do tej pory, były jeden do jednego. Chociaż możemy reprezentować dane wyjściowe różnych typów oddzielnie lub reprezentować każdą daną po prostu jako jedną cechę, możemy skorzystać z zalet użycia *Wejścia wielomodalnego*. Jest to czwarty wzorzec projektowy, który zbadamy w tym rozdziale.

Proste reprezentacje danych

Zanim sięgniemy po wyuczalne reprezentacje danych, krzyżowanie cech itp. przyjrzymy się prostym reprezentacjom danych. Możemy myśleć o tych prostych reprezentacjach danych jako o powszechnych *idiomach* w uczeniu maszynowym – raczej nie wzorcach, niemniej jednak powszechnie stosowanych rozwiązaniach.

Wejścia liczbowe

Większość nowoczesnych, wielkoskalowych modeli uczenia maszynowego (lasy losowe, maszyny wektorów nośnych, sieci neuronowe) operują na wartościach liczbowych, a zatem jeśli dane wejściowe są numeryczne, możemy przekazać je do modelu bez zmian.

Dlaczego skalowanie jest wskazane

Ponieważ framework ML używa optymalizatora, który jest dobrze dostrojony do pracy z liczbami z zakresu [– 1, 1], skalowanie wartości liczbowych, aby leżały w tym zakresie, może być często korzystne.

Dlaczego skalować wartości liczbowe, aby leżały w zakresie [– 1, 1]?

Optymalizatory spadku gradientu wymagają więcej kroków, aby osiągnąć zbieżność, gdy krzywa funkcji straty się zwiększa. Jest tak, ponieważ pochodne cech z większymi modułami względnymi będą miały tendencję, aby również być większe, co prowadzi do anormalnych aktualizacji wag. Anormalnie duże aktualizacje wag będą wymagać większej liczby kroków do osiągnięcia zbieżności, a zatem zwiększają obciążenie obliczeniowe.

„Centrowanie" danych, aby leżały w zakresie [– 1, 1], powoduje, że funkcja błędu jest bardziej sferyczna. Wtedy modele trenowane z przekształconymi danymi mają tendencję do szybszej zbieżności, a zatem szybszego/tańszego trenowania. Dodatkowo zakres [– 1, 1] oferuje większą precyzję liczb zmiennoprzecinkowych.

Szybki test na jednym z wbudowanych zbiorów danych scikit-learn może to udowodnić (ten fragment kodu dostępny jest w repozytorium dla tej książki[2]):

```
from sklearn import datasets, linear_model
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
raw = diabetes_X[:, None, 2]
max_raw = max(raw)
min_raw = min(raw)
scaled = (2*raw - max_raw - min_raw)/(max_raw - min_raw)

def train_raw():
    linear_model.LinearRegression().fit(raw, diabetes_y)

def train_scaled():
    linear_model.LinearRegression().fit(scaled, diabetes_y)

raw_time = timeit.timeit(train_raw, number=1000)
scaled_time = timeit.timeit(train_scaled, number=1000)
```

2 *https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb*

Po uruchomieniu tego kodu uzyskaliśmy prawie 9% poprawę w przypadku modelu, który używał tylko jednej cechy wejściowej. Biorąc pod uwagę liczbę cech w typowym modelu uczenia maszynowego, oszczędności mogą się sumować.

Innym ważnym powodem skalowania jest to, że pewne algorytmy i techniki uczenia maszynowego są bardzo wrażliwe na względne moduły różnych cech. Na przykład algorytm klastrowania k-średnich, który używa odległości euklidesowej jako miary przybliżenia, dojdzie do polegania mocno na cechach z większymi modułami. Brak skalowania wpływa także na skuteczność regularyzacji L1 lub L2, ponieważ moduł wag dla cechy zależy od modułu wartości tej cechy, a zatem regularyzacja będzie wpływać różnie na różne cechy. Dzięki skalowaniu wszystkich cech, aby leżały między [– 1, 1] zapewniamy, że nie ma zbyt wielkiej różnicy między względnymi modułami różnych cech.

Skalowanie liniowe

Powszechnie stosowane są cztery formy skalowania:

Skalowanie min-max
: Wartość liczbowa jest skalowana liniowo, aby minimalna wartość, którą można przyjąć na wejściu, była skalowana do – 1, a możliwa maksymalna wartość do 1:

```
x1_scaled = (2*x1 - max_x1 - min_x1)/(max_x1 - min_x1)
```

Problem ze skalowaniem min-max polega na tym, że maksymalna i minimalna wartość (max_x1 i min_x1) muszą być oszacowane na podstawie zbioru uczącego, a często są to wartości odstające. Rzeczywiste dane często kurczą się do bardzo małego zakresu w paśmie [– 1, 1].

Przycinanie (w połączeniu ze skalowaniem min-max)
: Pomaga rozwiązać problem z elementami odstającymi przez użycie „rozsądnych" wartości zamiast szacowania minimum i maksimum ze zbioru uczącego. Wartość liczbowa jest liniowo skalowana między dwiema rozsądnymi granicami, a następnie przycinana do zakresu [– 1, 1]. W efekcie elementy odstające są traktowane jako – 1 lub 1.

Standaryzacja Z
: Rozwiązuje problem z elementami odstającymi bez wymagania wcześniejszej wiedzy o rozsądnym zakresie, dzięki linowemu skalowaniu wejścia przy użyciu średniej i odchylenia standardowego, oszacowanych dla zbioru uczącego:

```
x1_scaled = (x1 - mean_x1)/stddev_x1
```

Nazwa metody odzwierciedla fakt, że skalowana wartość ma średnią równą zero i jest normalizowana przez odchylenie standardowe, tak aby miała jednostkową wariancję na zbiorze uczącym. Skalowana wartość jest nieograniczona, ale mieści się w przedziale [– 1, 1] większość razy (67%, jeśli używamy rozkładu normalnego). Wartości poza tym zakresem są coraz rzadsze, gdy zwiększa się ich wartość bezwzględna, ale nadal obecne.

Winsoryzacja

Empiryczny rozkład stwierdzony w zbiorze uczącym jest stosowany do przycięcia zbioru danych do granic ustawionych na 10. i 90. percentyl wartości danych (lub 5. i 95. percentyl, itp.). Wartość poddana winsoryzacji podlega skalowaniu min-max.

Wszystkie opisane do tej pory metody skalują dane liniowo (w przypadku przycinania i winsoryzacji – liniowo w typowym zakresie). Min-max i przycinanie zwykle działają najlepiej dla jednolitych rozkładów danych, a standaryzacja Z sprawdza się najlepiej dla rozkładu normalnego. Wpływ różnych funkcji skalowania na kolumnę `mother_age` (wiek matki) w przykładzie predykcji masy dziecka jest pokazany na rysunku 2-3 (zobacz pełen kod[3]).

Bez wyrzucania „elementów odstających"

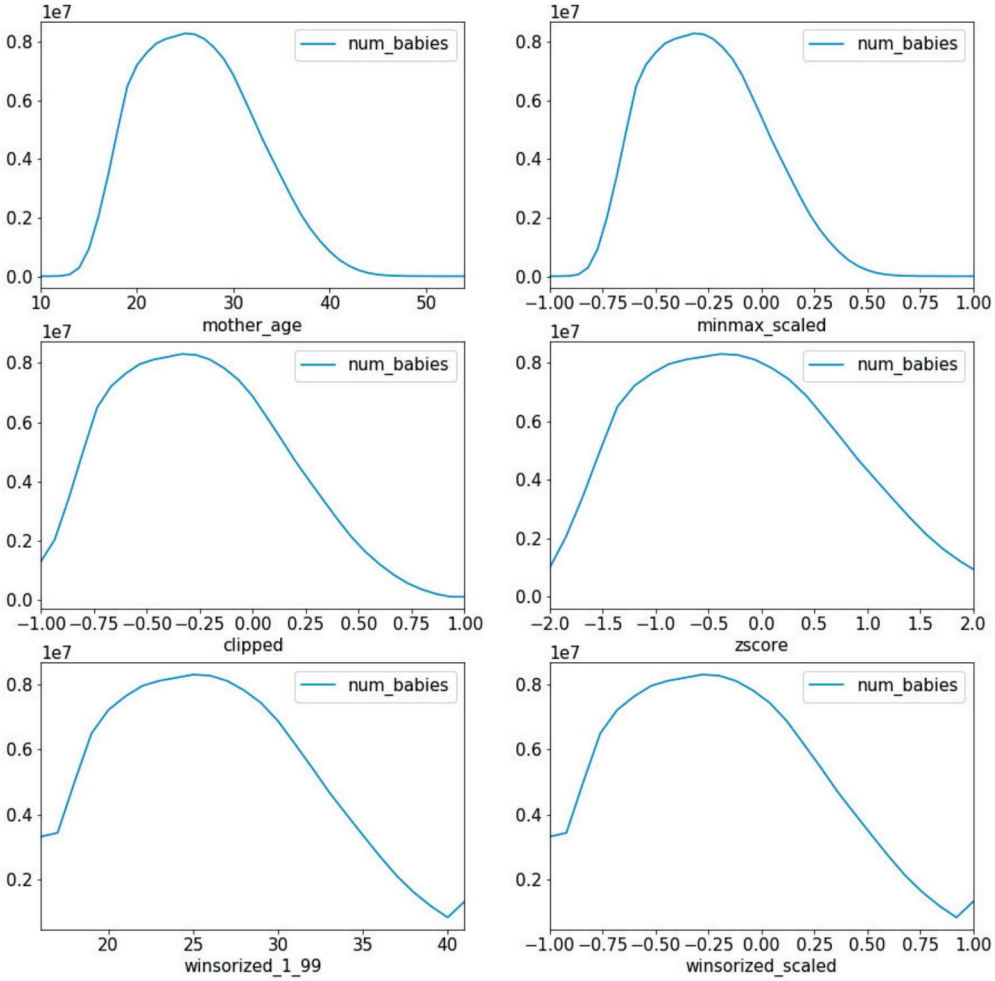
Zauważ, że zdefiniowaliśmy przycinanie jako branie przeskalowanych wartości mniejszych od – 1 i traktowanie ich jako – 1, a przeskalowanych wartości większych od 1 i traktowanie ich jako 1. Nie odrzucamy po prostu takich „elementów odstających", ponieważ oczekujemy, że model uczenia maszynowego będzie napotykać takie elementy odstające w środowisku produkcyjnym. Weźmy na przykład dzieci urodzone przez 50-letnie matki. Ponieważ nie ma wystarczająco dużo starszych matek w naszym zbiorze danych, przycinanie kończy się traktowaniem wszystkich matek starszych niż 45-letnie (na przykład) jako 45-letnich. To samo traktowanie zostanie zastosowane w środowisku produkcyjnym, a zatem nasz model będzie w stanie obsługiwać starsze matki. Model nie nauczyłby się, jak odzwierciedlać elementy odstające, jeśli po prostu wyrzucilibyśmy wszystkie przykłady uczące dzieci urodzonych przez matki 50-letnie i starsze!

Inny sposób myślenia o tym problemie polega na tym, że chociaż akceptowalne jest wyrzucenie *nieprawidłowych danych wejściowych*, nie jest akceptowalne wyrzucenie *danych prawidłowych*. Zatem można usprawiedliwić wyrzucenie wierszy, gdzie mother_age (wiek matki) jest liczbą ujemną, ponieważ jest to prawdopodobnie błąd wprowadzania danych. W środowisku produkcyjnym walidacja formularza wejściowego zapewni, że rejestratorka musi wprowadzić wiek matki. Jednak nie da się usprawiedliwić wyrzucenia wierszy, gdzie wartość mother_age jest równa 50, ponieważ 50 jest doskonale prawidłowym wejściem i oczekujemy uwzględniania 50-letnich matek po wdrożeniu modelu do środowiska produkcyjnego.

Na rysunku 2-3 można zauważyć, że minmax_scaled sprowadza wartości x do żądanego przedziału [– 1, 1], ale nadal zachowuje wartości na ekstremalnych końcach rozkładu, gdzie nie ma wystarczającej liczby przykładów. Przycinanie zawija wiele problematycznych wartości, ale wymaga, aby użyte progi przycinania były dokładnie prawidłowe – tutaj

3 *https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb*

powolne zmniejszanie liczby dzieci matek w wieku powyżej 40 lat stanowi problem z ustawieniem sztywnego progu. Winsoryzacja, podobnie jak przycinanie, wymaga dokładnie prawidłowego pobrania progów percentylowych. Normalizacja Z poprawia zakres (ale nie ogranicza wartości, aby były w zakresie [– 1, 1]), i odsuwa dalej problematyczne wartości. Z tych trzech metod normalizacja Z działa najlepiej dla `mother_age`, ponieważ surowe wartości wieku przypominają krzywą dzwonową. W przypadku innych problemów skalowanie min-max, przycinanie lub winsoryzacja mogą być lepsze.

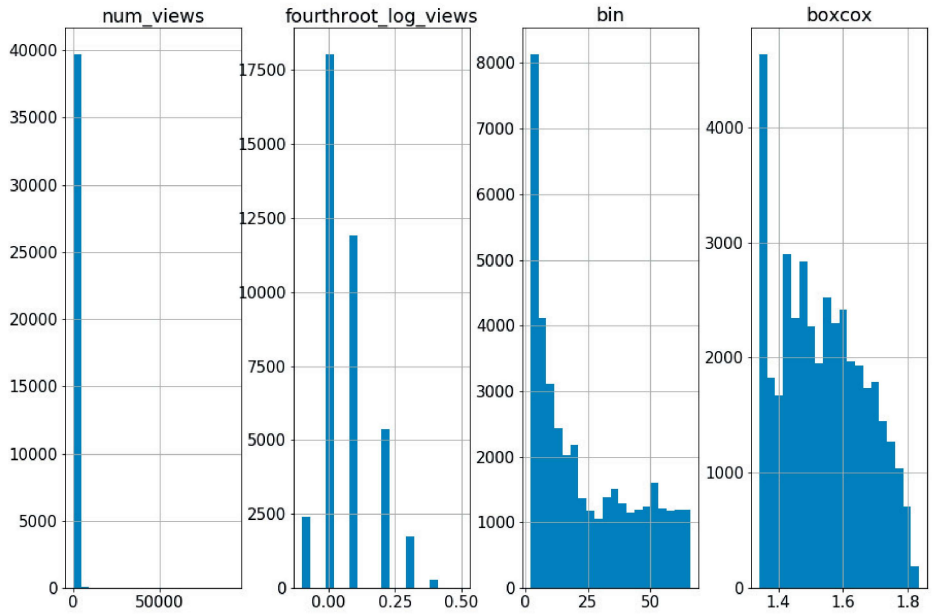


Rysunek 2-3 *Histogram mother_age w przykładzie predykcji masy urodzeniowej dziecka jest pokazany w panelu po lewej u góry, a różne funkcje skalowania (patrz etykieta osi x) są pokazane w pozostałych panelach.*

Przekształcenia nieliniowe

Co w przypadku, gdy nasze dane są skośne i nie mają ani rozkładu jednostajnego, ani przypominającego krzywą dzwonową? W takim przypadku lepiej jest zastosować *przekształcenie nieliniowe* wejścia przed jego skalowaniem. Popularnym sposobem jest obliczenie logarytmu wartości wejściowej przed jej skalowaniem. Inne popularne przekształcenia

obejmują funkcję sigmoidalną i rozwinięcia wielomianowe (kwadrat, pierwiastek kwadratowy, sześcian, pierwiastek sześcienny itd.). To, że mamy dobrą funkcję przekształcenia, poznamy po tym, gdy przekształcona wartość uzyska rozkład jednostajny lub normalny.



Rysunek 2-4 *Lewy panel: rozkład liczby wyświetleń stron Wikipedii jest mocno skośny i obejmuje duży zakres dynamiki. Na drugim panelu zademonstrowano, że te problemy można rozwiązać przez przekształcenie liczby wyświetleń przy użyciu kolejno logarytmu, funkcji potęgowej oraz skalowania liniowego. Na trzecim panelu pokazano efekty wyrównania histogramu, a na czwartym efekt przekształcenia Boxa-Coxa.*

Załóżmy, że konstruujemy model do predykcji sprzedaży książki z kategorii literatury faktu. Jednym z wejść modelu jest popularność strony Wikipedii związanej z tym tematem. Liczba wyświetleń stron w Wikipedii jest jednak mocno skośna i zajmuje duży zakres dynamiki (patrz lewy panel na rysunku 2-4: rozkład jest mocno skośny w kierunku rzadko wyświetlanych stron, ale najpopularniejsze strony są wyświetlane miliony razy). Dzięki obliczeniu logarytmu wyświetleń, a następnie obliczeniu pierwiastka czwartego stopnia z wartości tego logarytmu i liniowemu przeskalowaniu wyników, otrzymujemy coś, co jest żądanym zakresem i ma kształt trochę przypominający dzwon. W celu uzyskania szczegółowego kodu do odpytywania danych z Wikipedii, zastosowania przekształceń i generowania wykresu, zobacz znajdujące się w witrynie GitHub w repozytorium do tej książki[4].

Wynalezienie funkcji linearyzacji, która sprawia, że rozkład wygląda jak krzywa dzwonowa, może być trudne. Łatwiejszym podejściem jest utworzenie zasobników dla wielu widoków i takie dobranie granic zasobników, aby pasowały do żądanego rozkładu wyjściowego. Pryncypialnym podejściem do wyboru tych zasobników jest wykonanie *wyrównania histogramu*, gdzie zasobniki histogramu są wybrane na bazie kwantyli w surowym

4 *https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb*

rozkładzie (patrz trzeci panel na rysunku 2-4). W idealnej sytuacji wyrównanie histogramu skutkuje rozkładem jednostajnym (chociaż nie w tym przypadku, z powodu powtarzalnych wartości w kwantylach). Aby wykonać wyrównanie histogramu w BigQuery, możemy użyć instrukcji:

```
ML.BUCKETIZE(num_views, bins) AS bin
```

gdzie zmienna bins (zasobniki) jest uzyskana z:

```
APPROX_QUANTILES(num_views, 100) AS bins
```

Zobacz notes w repozytorium kodu do tej książki, aby uzyskać pełne szczegóły[5].

Inną metodą do obsłużenia skośnych rozkładów jest użycie techniki przekształcenia parametrycznego, takiego jak *przekształcenie Boxa-Coxa*. Polega ono na wybraniu pojedynczego parametru, lambda, w celu kontroli „heteroskedastyczności", aby wariancja nie zależała już od modułu. Tutaj wariancja wśród rzadko odwiedzanych stron Wikipedii była znacznie mniejsza niż wariancja wśród często odwiedzanych stron, a przekształcenie Boxa-Coxa próbuje wyrównać wariancję wśród wszystkich zakresów wielu widoków. Można to zrobić przy użyciu pakietu SciPy języka Python:

```
traindf['boxcox'], est_lambda = (
    scipy.stats.boxcox(traindf['num_views']))
```

Parametr szacowany na podstawie zbioru uczącego (est_lambda) jest następnie używany do przekształcenia innych wartości:

```
evaldf['boxcox'] = scipy.stats.boxcox(evaldf['num_views'], est_lambda)
```

Tablica liczb

Czasami dane wejściowe są tablicą liczb. Jeśli tablica ma stałą długość, reprezentacja danych może być dość prosta: spłaszczenie tablicy i traktowanie każdej pozycji jako odmiennej cechy. Ale często tablica będzie miała zmienną długość. Na przykład jednym z wejść modelu do prognozowania sprzedaży książki niefikcyjnej może być sprzedaż wszystkich poprzednich książek na ten temat. Przykładem może być:

```
[2100, 15200, 230000, 1200, 300, 532100]
```

Oczywiście długość tej tablicy będzie różna w każdym wierszu, ponieważ mamy różne liczby książek publikowanych na różne tematy.

Popularne idiomy do obsługi tablic liczbowych to m.in.:

- Reprezentacja tablicy wejściowej w terminach jej statystyk zbiorczych. Na przykład możemy użyć długości (czyli liczby wcześniejszych książek na ten temat), średniej, mediany, minimum, maksimum itd.

5 *https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/simple_data_representation.ipynb*

- Reprezentacja tablicy wejściowej w terminach rozkładu empirycznego – tj. według 10./20./... percentyla itd.
- Jeśli tablica jest uporządkowana w konkretny sposób (np. w kolejności czasu lub według rozmiaru), reprezentacja tablicy wejściowej następuje według ostatnich trzech lub jakiejś innej stałej liczby elementów. Dla tablic o długości mniejszej niż trzy, cechy są dopełniane do długości trzy brakującymi wartościami.

Wszystkie te sposoby kończą się reprezentacją tablicy danych o zmiennej długości jako cechy o stałej długości. Możemy także sformułować ten problem jako problem prognozowania szeregów czasowych, ponieważ problem przewidywania sprzedaży następnej książki na dany temat jest oparty na historii sprzedaży poprzednich książek w czasie. Traktując sprzedaż poprzednich książek jako tablicę wejściową, zakładamy, że najważniejszymi czynnikami w predykcji sprzedaży książki jest sama książka (autor, wydawca, recenzje itp.), a nie kontynuacja w czasie wielkości sprzedaży.

Wejścia kategorialne

Ponieważ większość nowoczesnych modeli uczenia maszynowego (lasy losowe, maszyny wektorów nośnych, sieci neuronowe) operuje na wartościach liczbowych, wejścia kategorialne muszą być reprezentowane jako liczby.

Proste ponumerowanie możliwych wartości i odwzorowanie ich na skali porządkowej będzie działać słabo. Załóżmy, że jednym z wejść do modelu, który prognozuje sprzedaż książki, jest język, w jakim została ona napisana. Nie możemy po prostu utworzyć takiej tabeli odwzorowań:

Wejście kategorialne	Cecha liczbowa
angielski	1.0
chiński	2.0
niemiecki	3.0

Jest to niewłaściwe, ponieważ model uczenia maszynowego próbowałby dokonać interpolacji między popularnością książek niemieckich i angielskich, aby uzyskać popularność książki po chińsku! Ponieważ nie ma porządkowej relacji między językami, trzeba użyć takiego odwzorowania kategorii na liczby, które pozwala, aby model uczył się z rynku książek napisanych w tych językach w sposób niezależny.

Kodowanie 1 z n

Najprostszą metodą odwzorowania zmiennych kategorialnych przy zapewnieniu że zmienne są niezależne, jest *kodowanie 1 z n*. W naszym przypadku kategorialna zmienna wejściowa zostałaby skonwertowana na trójelementowy wektor cech przy użyciu następującego odwzorowania:

Wejście kategorialne	Cecha liczbowa
angielski	[1.0, 0.0, 0.0]

Wejście kategorialne	Cecha liczbowa
chiński	[0.0, 1.0, 0.0]
niemiecki	[0.0, 0.0, 1.0]

Kodowanie 1 z n wymaga od nas znajomości z wyprzedzeniem *słownika* wejścia kategorialnego. Tutaj słownik składa się z trzech tokenów (angielski, chiński i niemiecki), a długość wynikowej cechy jest rozmiarem tego słownika.

Dummy coding czy kodowanie 1 z n?

Technicznie rzecz biorąc, 2-elementowy wektor cech jest wystarczający do zapewnienia unikalnego odwzorowania słownika o rozmiarze 3:

Wejście kategorialne	Cecha liczbowa
angielski	[0.0, 0.0]
chiński	[1.0, 0.0]
niemiecki	[0.0, 1.0]

Podejście to jest nazwane *dummy coding* (głupie kodowanie). Ponieważ jest bardziej zwięzłą reprezentacją, jest preferowane w modelach statystycznych, które działają lepiej, gdy wejścia są liniowo niezależne.

Jednak nowoczesne algorytmy uczenia maszynowego nie wymagają, aby ich wejścia były liniowo niezależne i używają takich metod, jak regularyzacja L1, do przycinania nadmiarowych wejść. Dodatkowy stopień swobody pozwala, aby framework przejrzyście obsługiwał brakujące wejścia w środowisku produkcyjnym jako same zera:

Wejście kategorialne	Cecha liczbowa
angielski	[1.0, 0.0, 0.0]
chiński	[0.0, 1.0, 0.0]
niemiecki	[0.0, 0.0, 1.0]
(brakujący)	[0.0, 0.0, 0.0]

Zatem wiele frameworków uczenia maszynowego często obsługuje tylko kodowanie 1 z n.

W pewnych okolicznościach może być przydatne traktowanie wejścia liczbowego jako kategorialnego i odwzorowanie go w kolumnie z kodowaniem 1 z n:

Kiedy wejście liczbowe jest indeksem

Na przykład, jeżeli próbujemy prognozować poziomy ruchu i jednym z wejść jest dzień tygodnia, możemy traktować dzień tygodnia jako liczbę (1, 2, 3, …, 7), ale warto

zauważyć, że numer dnia tygodnia nie tworzy ciągłej skali, ale faktycznie jest po prostu indeksem. Lepiej traktować go jako zmienną kategorialną (niedziela, poniedziałek, …, sobota), ponieważ indeksowanie jest arbitralne. Czy tydzień powinien zaczynać się od niedzieli (jak w USA), poniedziałku (jak we Francji) czy soboty (jak w Egipcie)?

Kiedy relacje między wejściem a etykietą są nieciągłe
: To, co powinno przechylić szalę w kierunku traktowania dnia tygodnia jako cechy kategorialnej, to fakt, że na natężenie ruchu w piątek nie wpływa natężenie ruchu w czwartek ani sobotę.

Kiedy traktowanie zmiennej liczbowej jako zasobnika jest korzystne
: W większości miast natężenie ruchu zależy od tego, czy jest to weekend, a to z kolei zależy od lokalizacji (sobota i niedziela w większości świata, czwartek i piątek w niektórych krajach islamskich). Byłoby zatem pomocne traktowanie dnia tygodnia jako cechy logicznej (weekend lub dzień roboczy). Takie odwzorowanie, gdzie liczba rozdzielnych wejść (tutaj siedem) jest większa niż liczba oddzielnych wartości cech (tutaj dwie) jest nazywana stosowaniem zasobników. Typowe stosowanie zasobników odbywa się na podstawie zakresów – na przykład możemy podzielić na zasobniki zakresy zmiennej mother_age, które dzielą się przy 20, 25, 30 itd. i traktować każdy z tych zasobników jako kategorialny, ale powinniśmy zdawać sobie sprawę, że tracimy w ten sposób porządkową naturę zmiennej wejściowej mother_age.

Kiedy chcemy traktować różne wartości wejścia liczbowego jako niezależne, gdy chodzi o ich wpływ na etykietę
: Na przykład masa dziecka zależy od mnogości[6] ciąży, ponieważ bliźnięta i trojaczki mają zwykle mniejszą masę niż dzieci urodzone pojedynczo. Zatem dziecko, które ma niższą masę, ale należy do trojaczków, może być zdrowsze niż bliźniak o tej samej masie. W takim przypadku możemy odwzorować mnogość jako zmienną kategorialną, ponieważ zmienna kategorialna pozwala, aby model uczył się niezależnie strojonych parametrów dla różnych wartości mnogości. Oczywiście możemy zrobić to tylko wtedy, gdy mamy wystarczająco dużo przykładów bliźniąt i trojaczków w zbiorze danych.

Tablica wartości kategorialnych

Czasami dane wejściowe są tablicą kategorii. Jeśli tablica ma stałą długość, możemy traktować każdą pozycję jako oddzielną cechę. Jednak często tablica będzie miała zmienną długość. Na przykład jednym z wejść modelu dotyczącego urodzeń mogą być typy poprzednich porodów tej matki:

```
[Induced, Induced, Natural, Cesarean]
```

Oczywiście długość tej tablicy będzie różna w każdym wierszu, ponieważ mamy różne liczby starszego rodzeństwa dla każdego dziecka.

Typowe idiomy do obsługi tablic wartości kategorialnych to m.in.:

6 Dla bliźniąt mnogość to 2. Dla trojaczków mnogość to 3.

- *Zliczanie* liczby wystąpień każdego elementu słownika. Zatem reprezentacja dla naszego przykładu byłaby `[2, 1, 1]` przy założeniu, że słownik to `Induced, Natural`, oraz `Cesarean` (wywoływany, naturalny, cesarski – w tej kolejności). Jest to teraz tablica liczb o stałej długości, którą możemy spłaszczyć i używać w kolejności pozycyjnej. Jeśli mamy tablicę, gdzie element może wystąpić tylko raz (na przykład języki, którym mówi osoba), lub jeśli funkcja po prostu wskazuje obecność, a nie liczbę (na przykład, czy matka kiedykolwiek miała cesarskie cięcie), wtedy liczba na każdej pozycji jest 0 lub 1, co jest nazywane *kodowaniem wiele z n*.
- W celu uniknięcia dużych liczb można użyć *względnej częstotliwości* zamiast zliczania. Reprezentacją dla naszego przykładu byłoby [0.5, 0.25, 0.25] zamiast [2, 1, 1]. Puste tablice (pierworodne dziecko bez wcześniejszego rodzeństwa) są reprezentowane jako [0, 0, 0]. W przetwarzaniu języka naturalnego względna częstotliwość słowa jest ogólnie normalizowana przez względną częstotliwość w dokumentach, które zawierają to słowo, aby zwrócić *częstość termu* × *odwrotną częstość w dokumentach*, czyli TF-IDF[7]. TF-IDF (skrót od *term frequency* × *inverse document frequency*) odzwierciedla, jak unikalne jest słowo w danym dokumencie.
- W przypadku tablicy uporządkowanej w konkretny sposób (np. w kolejności czasowej) reprezentacja tablicy wejściowej wg trzech ostatnich elementów. Tablice krótsze niż trzy są dopełniane brakującymi wartościami.
- Reprezentacja tablicy przez zbiorcze statystyki, np. długość tablicy, modę (najczęściej występujący wpis), medianę, 10./20./… percentyl itp.

Spośród nich idiom zliczania/względnej częstości jest najczęstszy. Zauważ, że oba te sposoby są uogólnieniem kodowania 1 z n – jeśli dziecko nie ma starszego rodzeństwa, reprezentacją byłoby `[0, 0, 0]`, a jeśli dziecko ma jedno starsze rodzeństwo urodzone naturalnie, reprezentacją byłoby `[0, 1, 0]`.

Po obejrzeniu prostych reprezentacji danych omówimy wzorce projektowe, które pomagają w reprezentacji danych.

Wzorzec projektowy 1: Cecha haszowana

Wzorzec projektowy Cecha haszowana (Hashed Feature) rozwiązuje trzy możliwe problemy związane z cechami kategorialnymi: niepełny słownik, rozmiar modelu związany z mocą zbioru i zimny rozruch. Robi to przez grupowanie cech kategorialnych i zgodę na kompromis dotyczący występowania kolizji w reprezentacji danych.

Problem

Kodowanie 1 z n kategorialnej zmiennej wejściowej wymaga znajomości słownika z wyprzedzeniem. Nie jest to problemem, jeśli zmienna wejściowa jest czymś takim, jak język, w którym napisano książkę, lub dzień tygodnia, kiedy prognozowany jest poziom ruchu

7 *https://oreil.ly/kNYHr*

drogowego. Co jednak, jeśli zmienna kategorialna, o której mowa, jest czymś takim jak identyfikator szpitala (`hospital_id`), gdzie urodziło się dziecko, lub identyfikator lekarza (`physician_id`), czyli osoby odbierającej poród? Takie zmienne kategorialne stanowią kilka problemów:

- Znajomość słownika wymaga wyodrębnienia go z danych uczących. Z powodu losowego próbkowania jest możliwe, że dane uczące nie będą zawierać wszystkich możliwych szpitali lub lekarzy. Słownik może być *niekompletny*.
- Zmienne kategorialne mają *wielką moc*. Zamiast wektorów cech z trzema językami lub siedmioma dniami mamy wektory cech, których długość liczona jest w tysiącach lub milionach. Takie wektory cech w praktyce generują wiele problemów. Obejmują tyle wag, że dane uczące mogą być niewystarczające. Jeśli nawet uczenie modelu będzie możliwe, wytrenowany model będzie wymagać dużo miejsca w pamięci masowej, ponieważ w czasie serwowania potrzebny jest cały słownik. Zatem wdrożenie modelu na mniejszych urządzeniach może być niemożliwe.
- Po wprowadzeniu modelu do środowiska produkcyjnego mogą zostać zbudowane nowe szpitale i zatrudnieni nowi lekarze. Model nie będzie w stanie dokonywać dla nich predykcji, a zatem oddzielna infrastruktura serwowania będzie wymagana do obsłużenia takich problemów *zimnego rozruchu*.

Nawet w przypadku prostych reprezentacji, takich jak kodowanie 1 z n, warto przewidzieć problem zimnego rozruchu i jawnie zarezerwować same zera dla wejść spoza słownika.

Jako konkretny przykład weźmiemy problem prognozowania opóźnienia lądowania samolotu. Jednym z wejść modelu jest lotnisko odlotu. W czasie zbierania danych w USA było 347 lotnisk:

```
SELECT
   DISTINCT(departure_airport)
FROM `bigquery-samples.airline_ontime_data.flights`
```

Niektóre lotniska miały niewiele lotów, od jednego do trzech, w całym rozpatrywanym przedziale czasu, a zatem oczekiwaliśmy, że słownik danych uczących będzie niekompletny. 347 jest wystarczająco dużą liczbą, żeby cecha była dość rzadka, a w tym przypadku z pewnością nowe lotniska będą budowane. Wszystkie trzy problemy (niepełny słownik, duża moc zbioru i zimny rozruch) będą występować, jeśli zapiszemy lotnisko odlotu za pomocą kodowania 1 z n.

Zbiór danych linii lotniczych, podobnie jak zbiór danych urodzeniowych i prawie wszystkie inne zbiory danych używane w tej książce dla ilustracji, jest publicznym zbiorem danych w BigQuery[8], na którym można wypróbować własne zapytania. Podczas pisania tej książki można było bezpłatnie wykonać 1 TB zapytań miesięcznie i dostępne

8 *https://oreil.ly/lgcKA*

jest środowisko testowe, które pozwala używać BigQuery do tego limitu bez obciążania karty kredytowej. Zachęcamy do dodania do zakładek naszego repozytorium GitHub. Jako przykład, można wykorzystać notes w witrynie GitHub zawierający pełen kod[9].

Rozwiązanie

Wzorzec projektowy Cecha haszowana reprezentuje kategorialną zmienną wejściową w następujący sposób:

1. Konwersja wejścia kategorialnego na unikalny ciąg. W przypadku lotniska odlotu, możemy użyć trzyliterowego kodu IATA[10] danego lotniska.
2. Wywołanie deterministycznego (bez ziaren losowych ani soli) i przenośnego (aby ten sam algorytm mógł być używany do uczenia i serwowania) algorytmu haszowania ciągu.
3. Wzięcie reszty z dzielenia wyniku haszowania przez żądaną liczbę zasobników. Typowo algorytm haszowania zwraca liczbę całkowitą, która może być ujemna, a wynik modulo liczby całkowitej ujemnej jest ujemny. Zatem brana jest wartość bezwzględna wyniku.

W BigQuery SQL te kroki są osiągane następująco:

```
ABS(MOD(FARM_FINGERPRINT(airport), numbuckets))
```

Funkcja `FARM_FINGERPRINT` używa rodziny algorytmów FarmHash, które są deterministyczne, mają dobry rozkład, a także implementacje dostępne w wielu językach programowania[11].

W TensorFlow te kroki są implementowane przez funkcję `feature_column`:

```
tf.feature_column.categorical_column_with_hash_bucket(
    airport, num_buckets, dtype=tf.dtypes.string)
```

Na przykład w tabeli 2-1 przedstawiono wynik FarmHash niektórych kodów lotnisk IATA po haszowaniu na 3, 10 i 1000 zasobników.

Tabela 2-1 *Wynik funkcji FarmHash dla niektórych kodów lotnisk IATA po haszowaniu na różną liczbę zasobników.*

Wiesz	departure_airport	hash3	hash10	hash1000
1	DTW	1	3	543
2	LBB	2	9	709

9 *https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/hashed_feature.ipynb*

10 *https://oreil.ly/B8nLw*

11 *https://github.com/google/farmhash/blob/master/Understanding_Hash_Functions*, *https://github.com/google/farmhash*

Wiesz	departure_airport	hash3	hash10	hash1000
3	SNA	2	7	587
4	MSO	2	7	737
5	ANC	0	8	508
6	PIT	1	7	267
7	PWM	1	9	309
8	BNA	1	4	744
9	SAF	1	2	892
10	IPL	2	1	591

Dlaczego to działa

Załóżmy, że wybraliśmy haszowanie kodu lotniska przy użyciu 10 zasobników (hash10 w tabeli 2-1). Jak rozwiązuje to rozpoznany problem?

Wejście spoza słownika

Nawet jeśli lotnisko z garścią lotów nie jest częścią zbioru uczącego, jego haszowana wartość cechy będzie w zakresie [0 – 9]. Zatem nie ma problemu elastyczności podczas serwowania – nieznane lotnisko będzie otrzymywać predykcje odpowiadające innym lotniskom w zasobniku haszowania. Model nie wykaże błędu.

Jeśli mamy 347 lotnisk, średnio 35 otrzyma ten sam kod zasobnika haszowania, jeśli dokonamy haszowania na 10 zasobników. Lotnisko, którego brakuje w zbiorze uczącym, będzie „pożyczać" swoje charakterystyki z innych podobnych ~35 lotnisk w zasobniku haszowania. Oczywiście predykcja dla brakującego lotniska nie będzie dokładna (nieuzasadnione byłoby oczekiwanie dokładnych predykcji dla nieznanych wejść), ale będzie mieścić się we właściwym zakresie.

Liczbę zasobników haszowania wybieramy przez równoważenie potrzeby rozsądnej obsługi wejść spoza słownika i potrzeby, aby model dokładnie odzwierciedlał wejście kategorialne. Przy 10 zasobnikach haszowania ~35 zostanie pomieszanych. Dobrą regułą praktyczną jest wybór takiej liczby zasobników haszowania, aby każdy zasobnik otrzymał pięć wpisów. W tym przypadku oznaczałoby to, że 70 zasobników haszowania jest dobrym kompromisem.

Wielka moc

Łatwo zauważyć, że problem wielkiej mocy zbioru jest rozwiązywany, o ile wybierzemy odpowiednio małą liczbę zasobników haszowania. Nawet jeśli mamy miliony lotnisk, szpitali czy lekarzy, możemy haszować te cechy na kilkaset zasobników, a zatem utrzymać praktyczne wymagania dotyczące pamięci systemu i rozmiaru modelu.

Nie potrzebujemy przechowywać słownika, ponieważ kod przekształcenia jest niezależny od rzeczywistej wartości danych, a rdzeń modelu radzi sobie z wejściami `num_buckets`, a nie z pełnym słownikiem.

Prawdą jest, że haszowanie jest stratne – ponieważ mamy 347, średnio 35 lotnisk otrzyma ten sam kod zasobnika haszowania, jeśli będziemy haszować na 10 zasobników. Kiedy alternatywą jest odrzucenie zmiennej z powodu zbyt rozległych wartości, kodowanie stratne jest akceptowalnym kompromisem.

Zimny rozruch

Sytuacja zimnego rozruchu przypomina sytuację danej spoza słownika. Jeśli nowe lotnisko jest dodawane do systemu, początkowo otrzymuje predykcje odpowiadające pozostałym lotniskom w zasobniku haszowania. W miarę, jak lotnisko staje się popularne, będzie więcej lotów z tego lotniska. O ile będziemy okresowo ponownie trenować model, jego predykcje zaczną odzwierciedlać opóźnienia przylotu z tego nowego lotniska. Jest to opisane bardziej szczegółowo w podrozdziale „Wzorzec projektowy 18: Ciągła ewaluacja modelu" w rozdziale 5. Dzięki wybraniu takiej liczby zasobników haszowania, że każdy zasobnik otrzymuje około pięciu wpisów, zapewniamy, że dowolny zasobnik będzie miał rozsądne wyniki początkowe.

Kompromisy i alternatywy

Większość wzorców projektowych uwzględnia pewnego rodzaju kompromis, a wzorzec projektowy Cecha haszowana nie jest wyjątkiem. Kluczowym kompromisem jest tutaj strata dokładności modelu.

Kolizje zasobników

Obliczanie modulo w implementacji Cechy haszowanej jest operacją stratną. Wybór rozmiaru haszowania równego 100 oznacza, że postanawiamy, aby 3 – 4 lotniska współdzieliły zasobnik. Jawnie rezygnujemy ze zdolności reprezentacji danych (ze stałym słownikiem i kodowaniem 1 z n) w celu obsłużenia wejść spoza słownika, ograniczeń mocy/rozmiaru modelu oraz problemów zimnego rozruchu. Nie ma nic za darmo. Nie należy wybierać Cechy haszowanej, jeśli słownik jest znany z wyprzedzeniem, jeśli rozmiar słownika jest względnie mały (liczony w tysiącach jest akceptowalny dla zbioru danych z milionami próbek), a zimny start nie stanowi problemu.

Należy zauważyć, że nie możemy po prostu zwiększyć liczby zasobników do ekstremalnie wysokiej liczby w nadziei całkowitego uniknięcia kolizji. Nawet jeśli zwiększymy liczbę zasobników do 100 000 przy tylko 347, prawdopodobieństwo, że co najmniej dwa lotniska współdzielą ten sam zasobnik haszowania, wynosi 45% – co jest nieakceptowalnie wysokie (patrz tabela 2-2). Zatem powinniśmy używać Cech haszowanych tylko wtedy, gdy chcemy tolerować wiele wejść kategorialnych współdzielących tą samą wartość zasobnika haszowania.

Tabela 2-2 *Oczekiwana liczba wpisów na zasobnik i prawdopodobieństwo co najmniej jednej kolizji, gdy kody lotnisk IATA są haszowane na różną liczbę zasobników*

num_hash_buckets	entries_per_bucket	collision_prob
3	115.666667	1.000000
10	34.700000	1.000000
100	3.470000	1.000000
1000	0.347000	1.000000
10000	0.034700	0.997697
100000	0.003470	0.451739

Skośność

Utrata dokładności jest szczególnie dotkliwa, gdy rozkład wejścia kategorialnego jest bardzo skośny. Rozważmy przykład zasobnika haszowania, który zawiera ORD (Chicago, jedno z najbardziej ruchliwych lotnisk na świecie). Możemy je znaleźć w następujący sposób:

```
CREATE TEMPORARY FUNCTION hashed(airport STRING, numbuckets INT64) AS (
   ABS(MOD(FARM_FINGERPRINT(airport), numbuckets))
);

WITH airports AS (
SELECT
   departure_airport, COUNT(1) AS num_flights
FROM `bigquery-samples.airline_ontime_data.flights`
GROUP BY departure_airport
)

SELECT
   departure_airport, num_flights
FROM airports
WHERE hashed(departure_airport, 100) = hashed('ORD', 100)
```

Wyniki pokazują, że chociaż w bazie jest ~3.6 milionów lotów z ORD, znajdziemy tam tylko ~67000 lotów z BTV (Burlington, Vermont):

departure_airport	num_flights
ORD	3610491
BTV	66555
MCI	597761

To wskazuje, że we wszystkich praktycznych zastosowaniach model będzie uwzględniać długie czasy dojazdu taksówką i opóźnienia pogodowe doświadczane w Chicago przy prognozach dla miejskiego lotniska w Burlington, Vermont! Dokładność modelu dla BTV czy MCI (lotnisko w Kansas City) będzie dość słaba z powodu ogromnej liczby odlotów z Chicago.

Funkcja agregująca

W przypadkach, gdy rozkład zmiennej kategorialnej jest skośny lub gdy liczba zasobników jest tak mała, że kolizje zasobników są częste, możemy uznać za przydatne dodanie funkcji agregującej jako wejścia do naszego modelu. Na przykład dla każdego portu możemy znaleźć prawdopodobieństwo lotów bez opóźnień w zbiorze uczącym i dodać to jako cechę do naszego modelu Pozwoli to uniknąć straty informacji związanych z poszczególnymi lotniskami, gdy haszujemy kody lotnisk. W niektórych przypadkach możemy być w stanie całkowicie uniknąć używania nazwy lotniska jako cechy, ponieważ względna częstotliwość lotów bez opóźnień może być wystarczająca.

Strojenie hiperparametrów

Z powodu kompromisów związanych z częstotliwością kolizji zasobników wybór liczby zasobników może być trudny. Często zależy od samego problemu. Zatem zalecamy traktowanie liczby zasobników jako hiperparametru do strojenia:

```
- parameterName: nbuckets
      type: INTEGER
      minValue: 10
      maxValue: 20
      scaleType: UNIT_LINEAR_SCALE
```

Należy upewnić się, że liczba zasobników pozostaje w rozsądnym zakresie względem mocy zmiennej kategorialnej, która jest haszowana.

Haszowanie kryptograficzne

To, co sprawia, że Cecha haszowana jest stratna, to część obliczania modulo w jej implementacji. A jakby tak uniknąć całkowicie obliczania modulo? Ostatecznie algorytm Farm Fingerprint ma stałą długość (INT64 ma 64 bity), a zatem może być reprezentowany przy użyciu 64 wartości cech, z których każda wynosi 0 lub 1. Jest to nazywane *kodowaniem binarnym*.

Jednak kodowanie binarne nie rozwiązuje problemu wejść spoza słownika ani zimnego rozruchu (jedynie problem wielkiej mocy). Faktyczne kodowanie bitowe jest fałszywym tropem. Jeśli nie dokonujemy operacji modulo, możemy uzyskać unikalną reprezentację przez proste zastosowanie trzech znaków kodu IATA (zatem używając cechy o długości 3*26=78). Problem z taką reprezentacją jest natychmiast widoczny: lotniska, których nazwy zaczynają się na literę O, nie mają ze sobą nic wspólnego, jeśli chodzi o charakterystyki opóźnień lotów – kodowanie utworzyło *pozorną korelację* między lotniskami zaczynającymi się na tę samą literę. To samo spostrzeżenie zachodzi również w przestrzeni binarnej. Z tego powodu nie zalecamy binarnego kodowania wartości algorytmu Farm Fingerprint.

Kodowanie binarne funkcji haszującej MD5 nie wykazuje tego problemu pozornej korelacji, ponieważ wynik funkcji haszujacej MD5 ma rozkład jednostajny, a zatem wynikowe bity również będą miały rozkład jednostajny. Jednak w przeciwieństwie do algorytmu

Farm Fingerprint, algorytm haszujący MD5 nie jest deterministyczny ani unikatowy – jest haszowaniem jednokierunkowym i może mieć wiele nieoczekiwanych kolizji.

We wzorcu projektowym Cecha haszowana musimy używać algorytmu haszowania odcisku palca, a nie kryptograficznego algorytmu haszowania. Jest tak, ponieważ celem funkcji odcisku palca jest wytworzenie deterministycznej i unikalnej wartości. Jeśli się nad tym zastanowimy, uznamy to wymaganie za kluczowe w funkcjach wstępnego przetwarzania w uczeniu maszynowym, ponieważ potrzebujemy stosować tę samą funkcję podczas serwowania modelu i otrzymywać tę samą zahaszowaną wartość. Funkcja Fingerprint nie wytwarza wyjścia o rozkładzie jednostajnym. Algorytmy kryptograficzne, takie jak MD5 lub SHA1 wytwarzają wyjście o rozkładzie jednostajnym, ale nie są deterministyczne i celowo są kosztowne obliczeniowo. Dlatego haszowanie kryptograficzne nie jest użyteczne w kontekście inżynierii cech, gdzie wartość haszowana obliczona dla danego wejścia podczas predykcji musi być taka sama, jak obliczona podczas trenowania, a funkcja haszująca nie powinna spowalniać modelu uczenia maszynowego.

Powodem, dla którego algorytm MD5 nie jest deterministyczny, jest typowe dodawanie "soli" do ciągu, który ma być poddany haszowaniu. Sól[12] to losowy ciąg dodawany do każdego hasła w celu zapewnienia, że nawet jeśli dwoje użytkowników przypadkowo użyje tego samego hasła, zahaszowane wartości w bazie danych będą różne. Jest to potrzebne, aby udaremnić ataki oparte na „tęczowych tabelach", czyli ataki polegające na słownikach typowo wybieranych haseł, które porównują hasz znanego hasła z haszami w bazie danych. Dzięki wzrostowi mocy obliczeniowej możliwe jest przeprowadzenie również brutalnego ataku na każdą możliwą sól, dlatego nowoczesne implementacje kryptograficzne dokonują haszowania w pętli, aby zwiększyć koszt obliczeniowy. Nawet jeśli wyłączymy sól i zredukujemy liczbę iteracji do jednej, haszowanie MD5 jest jednokierunkowe. Wartość nie będzie unikalna.

Podsumowując, potrzebujemy użyć algorytmu haszowania odcisków palców. Potrzebujemy też obliczać modulo z wyniku haszowania.

Kolejność operacji

Należy zauważyć, że najpierw dokonujemy operacji modulo, a następnie obliczenia wartości bezwzględnej:

```
CREATE TEMPORARY FUNCTION hashed(airport STRING, numbuckets INT64) AS (
   ABS(MOD(FARM_FINGERPRINT(airport), numbuckets))
);
```

Kolejność funkcji `ABS`, `MOD` oraz `FARM_FINGERPRINT` w powyższym fragmencie kodu jest ważna, ponieważ zakres `INT64` nie jest symetryczny. Mówiąc ściślej, zakres ten rozciąga

12 *https://oreil.ly/cv7PS*

się od `-9 223 372 036 854 775 808` do `9 223 372 036 854 775 807` (włącznie). Zatem gdybyśmy zrobili tak:

```
ABS(FARM_FINGERPRINT(airport))
```

mógłby wystąpić rzadki i prawdopodobnie nieodtwarzalny błąd przepełnienia, gdy operacja FARM_FINGERPRINT zwróciłaby `-9 223 372 036 854 775 808`, ponieważ jej wartość bezwzględna nie może być reprezentowana przy użyciu INT64!

Puste zasobniki haszowania

Chociaż jest to mało prawdopodobne, istnieje możliwość, że nawet jeśli wybierzemy 10 zasobników haszowania do reprezentacji 347 lotnisk, jeden z zasobników haszowania będzie pusty. Dlatego podczas używania haszowanych kolumn cech korzystne może być użycie także regularyzacji L2[13], aby wagi skojarzone z pustym zasobnikiem były sprowadzone prawie do zera. W ten sposób, jeśli lotnisko spoza słownika wpadnie w pusty zasobnik, nie spowoduje numerycznej niestabilności modelu.

Wzorzec projektowy 2: Osadzanie

Osadzanie (Embeddings) to ucząca się reprezentacja danych, która odwzorowuje dane o wielkiej mocy na przestrzeń o mniejszej liczbie wymiarów w taki sposób, że informacje znaczące dla problemu uczenia są zachowywane. Osadzanie jest sercem nowoczesnego uczenia maszynowego i ma różne wcielenia w tej dziedzinie.

Problem

Modele uczenia maszynowego systematycznie szukają w danych wzorców, które przechwytują sposób relacji cech wejściowych modelu z etykietą wyjściową. W efekcie reprezentacja danych dla cech wejściowych bezpośrednio wpływa na jakość końcowego modelu. Chociaż obsługa strukturalnych, numerycznych wejść jest dość prosta, dane potrzebne do trenowania modelu uczenia maszynowego mogą występować w niezliczonych odmianach, takich jak zmienne kategorialne, tekst, obrazy, dźwięk, szeregi czasowe i wiele innych. Dla tych reprezentacji danych potrzebujemy przekazać do naszego modelu uczenia maszynowego znaczącą wartość liczbową, aby te cechy mogłyby pasować do typowego paradygmatu trenowania. Osadzanie zapewnia sposób obsługi niektórych spośród tych różnych typów danych, zachowujący podobieństwo między elementami, a zatem poprawia zdolność naszego modelu do nauczenia się istotnych wzorców.

Kodowanie 1 z n jest powszechnym sposobem reprezentacji kategorialnych zmiennych wejściowych. Na przykład rozważmy wejście mnogości w zbiorze danych urodzeniowych[14]. Jest to zmienna kategorialna, która ma sześć możliwych wartości: `['Single(1)'` (pojedyncza), `'Multiple(2+)'` (mnoga), `'Twins(2)'` (bliźnięta), `'Triplets(3)'` (trojaczki),

13 *https://oreil.ly/xlwAH*

14 Ten zbiór danych jest dostępny w BigQuery: *bigquery-public-data.samples.natality*.

`'Quadruplets(4)'` (czworaczki), `'Quintuplets(5)']` (pięcioraczki). Możemy obsłużyć to wejście kategorialne przy użyciu kodowania 1 z n, które odzwierciedla każdą potencjalną wartość ciągu wejściowego na wektor jednostkowy w R^6, jak pokazano w tabeli 2-3.

Tabela 2-3 *Przykład kodowania 1 z n wejść kategorialnych do zbioru danych urodzeniowych*

Mnogość	Kodowanie 1 z n
Single(1) (pojedyncza)	[1,0,0,0,0,0]
Multiple(2+) (mnoga)	[0,1,0,0,0,0]
Twins(2) (bliźnięta)	[0,0,1,0,0,0]
Triplets(3) (trojaczki)	[0,0,0,1,0,0]
Quadruplets(4) (czworaczki)	[0,0,0,0,1,0]
Quintuplets(5) (pięcioraczki)	[0,0,0,0,0,1]

Po zakodowaniu w ten sposób potrzebujemy sześciu wymiarów do reprezentacji każdej z różnych kategorii. Sześć wymiarów to może nie być bardzo źle, ale co z sytuacją, gdy mamy dużo więcej kategorii do rozważenia?

Co jeśli na przykład nasz zbiór danych składa się z historii oglądania przez klientów filmów z bazy danych, a zadaniem jest sugerowanie listy nowych filmów na podstawie wcześniejszych interakcji? W tym scenariuszu pole `customer_id` (identyfikator klienta) mogłoby mieć tysiące unikalnych wejść. Podobnie pole `video_id` (identyfikator filmu) dla wcześniej obejrzanych filmów równie dobrze mogłoby zawierać tysiące wpisów. Kodowanie 1 z n kategorialnych cech o *wielkiej mocy*, takich jak `video_ids` lub `customer_ids` jako wejść do modelu uczenia maszynowego prowadzi do powstania rzadkiej macierzy, która niezbyt dobrze pasuje do wielu algorytmów uczenia maszynowego.

Drugim problemem z kodowaniem 1 z n jest to, że traktuje zmienne kategorialne jako *niezależne*. Jednak reprezentacja danych bliźniąt powinna być bliska reprezentacji danych trojaczków i dość odległa od reprezentacji pięcioraczków. Ciąża mnoga najprawdopodobniej oznacza bliźnięta, ale może też oznaczać trojaczki. Jako przykład w tabeli 2-4 pokazano alternatywną reprezentację kolumny mnogości w mniejszym wymiarze, który przechwytuje taką relację *bliskości*.

Tabela 2-4 *Użycie osadzania o mniejszej liczbie wymiarów do reprezentacji kolumny mnogości w zbiorze danych urodzeniowych.*

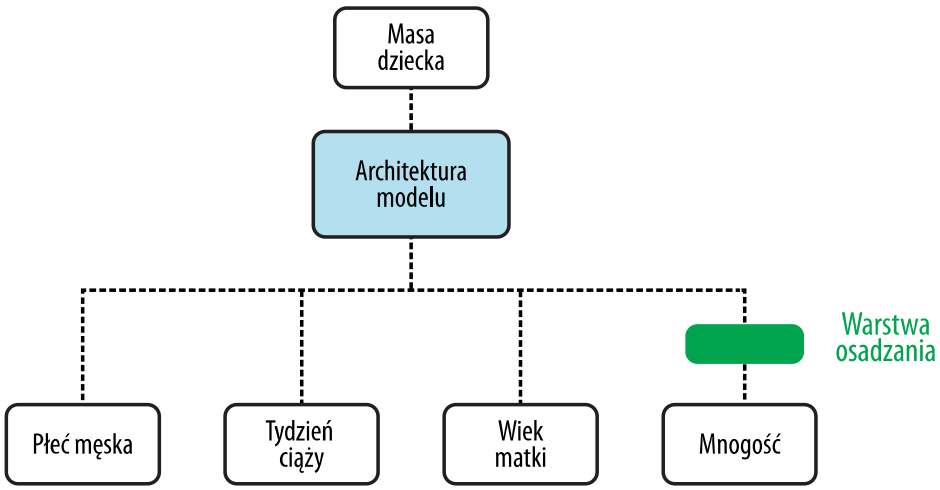
Mnogość	Kandydujące kodowanie
Single(1) (pojedyncza)	[1.0,0.0]
Multiple(2+) (mnoga)	[0.0,0.6]
Twins(2) (bliźnięta)	[0.0,0.5]
Triplets(3) (trojaczki)	[0.0,0.7]
Quadruplets(4) (czworaczki)	[0.0,0.8]
Quintuplets(5) (pięcioraczki)	[0.0,0.9]

Te liczby są oczywiście wybrane arbitralnie. Czy jest jednak możliwe nauczenie się najlepszej możliwej reprezentacji kolumny mnogości przy użyciu zaledwie dwóch wymiarów dla problemu urodzeń? To jest właśnie problem rozwiązywany przez wzorzec projektowy Osadzanie.

Te same trudności związane z wielką mocą i zależnościami danych występują w obrazach i tekście. Obrazy składają się z tysięcy pikseli, które nie są wzajemnie niezależne. Tekst w języku naturalnym pochodzi ze słownika obejmującego dziesiątki tysięcy słów, a słowo takie jak `walk` (spacerować) jest bliższe słowu `run` (biegać) niż słowu `book` (książka).

Rozwiązanie

Wzorzec projektowy Osadzanie rozwiązuje problem reprezentacji danych o wielkiej mocy bardziej gęsto w mniejszej liczbie wymiarów, dzięki przekazaniu danych przez warstwę osadzania, której wagi można wytrenować. W ten sposób zmienna wejściowa kategorialna o dużej liczbie wymiarów jest mapowana na wektor o wartościach rzeczywistych w pewnej przestrzeni o mniejszej liczbie wymiarów. Wagi do tworzenia gęstej reprezentacji są uczone w trakcie optymalizacji modelu (patrz rysunek 2-5). W praktyce osadzanie ostatecznie przechwytuje relacje bliskości w danych wejściowych.



Rysunek 2-5 *Wagi warstwy osadzania są wyuczane jako parametry podczas trenowania.*

Ponieważ osadzanie przechwytuje relacje bliskości w danych wejściowych w reprezentacji o mniejszej liczbie wymiarów, możemy użyć warstwy osadzania jako zamiennika dla technik klastrowania (np. segmentacji klientów) i metod redukcji wymiarów, takich jak analiza głównych składowych (PCA). Wagi osadzania są wyznaczane w głównej pętli trenowania modelu, zatem eliminują potrzebę uprzedniego klastrowania lub analizy PCA.

Wagi w warstwie osadzania byłyby uczone jako część procedury spadku gradientu podczas trenowania modelu urodzeń.

Na koniec trenowania wagi warstwy osadzania mogłyby kodować zmienne kategorialne, jak pokazano w tabeli 2-5.

Tabela 2-5 *Kodowanie 1 z n dla kolumny mnogości w zbiorze danych urodzeniowych*

Mnogość	Kodowanie 1 z n	Nauczone kodowanie
Single(1) (pojedyncza)	[1,0,0,0,0,0]	[0.4, 0.6]
Multiple(2+) (mnoga)	[0,1,0,0,0,0]	[0.1, 0.5]
Twins(2) (bliźnięta)	[0,0,1,0,0,0]	[-0.1, 0.3]
Triplets(3) (trojaczki)	[0,0,0,1,0,0]	[-0.2, 0.5]
Quadruplets(4) (czworaczki)	[0,0,0,0,1,0]	[-0.4, 0.3]
Quintuplets(5) (pięcioraczki)	[0,0,0,0,0,1]	[-0.6, 0.5]

Osadzanie odwzorowuje rzadkie, zakodowane 1 z n wektory na gęste dane wektorowe w R^2.

W TensorFlow najpierw konstruujemy kolumnę cechy kategorialnej, następnie opakowujemy ją w kolumnie cechy osadzonej. Na przykład dla cechy mnogości ciąż mielibyśmy:

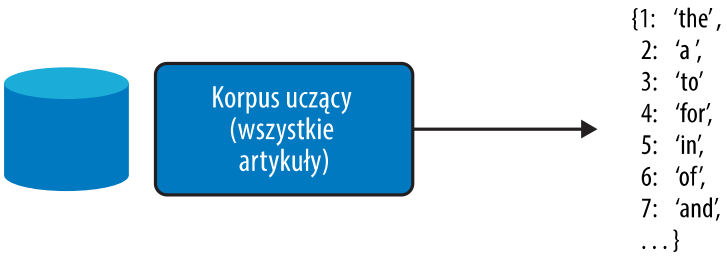
```
plurality = tf.feature_column.categorical_column_with_vocabulary_list(
            'plurality', ['Single(1)', 'Multiple(2+)', 'Twins(2)',
            'Triplets(3)', 'Quadruplets(4)', 'Quintuplets(5)'])
plurality_embed = tf.feature_column.embedding_column(plurality, dimension=2)
```

Wynikowa kolumna cechy osadzonej mnogości (plurality_embed) służy jako wejście do dalszych węzłów sieci neuronowej, zamiast kolumny cechy zakodowanej w 1 z n (plurality).

Osadzanie tekstu

Tekst stanowi naturalne środowisko, w którym korzystne jest używanie warstwy osadzania. Biorąc pod uwagę moc słownika (często rzędu dziesiątków tysięcy słów), kodowanie 1 z n każdego słowa nie jest praktyczne. Prowadziłoby do utworzenia niezwykle dużej (o ogromnej liczbie wymiarów) i jednocześnie rzadkiej macierzy do trenowania. Ponadto chcielibyśmy, aby podobne znaczeniowo słowa miały osadzenie bliskie, a niezwiązane dalekie w przestrzeni osadzania. Dlatego używamy gęstego osadzania słów do wektoryzacji dyskretnych wejść tekstowych przed przekazaniem do modelu.

W celu implementacji osadzania tekstu w Keras najpierw tworzymy tokenizację dla każdego słowa w słowniku, jak pokazano na rysunku 2-6. Następnie używamy tej tokenizacji do odwzorowania warstwy osadzania, podobnie jak zostało to zrobione dla kolumny mnogości.



Rysunek 2-6 *Tokenizator tworzy tabelę wyszukiwania, która odwzorowuje każde słowo na indeks.*

Tokenizacja jest tablicą wyszukiwania (*lookup table*), która odwzorowuje każde słowo w naszym słowniku na indeks. Możemy myśleć o tym jako o kodowaniu 1 z n każdego słowa, gdzie tokenizowany indeks jest lokalizacją niezerowego elementu w kodowaniu 1 z n. To wymaga pełnego przejścia przez cały zbiór danych (załóżmy, że składa się z tytułów artykułów[15]), aby utworzyć tabelę wyszukiwania i może być to dokonane w Keras. Pełny kod można znaleźć w repozytorium do tej książki[16]:

```
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(titles_df.title)
```

Tutaj możemy użyć klasy `Tokenizer` z biblioteki *keras.preprocessing.text*. Wywołanie do `fit_on_texts` tworzy tabelę wyszukiwania, która odzwierciedla każdy wyraz znaleziony w naszych tekstach na indeks. Wywołanie `tokenizer.index_word` umożliwia bezpośrednie zbadanie tej tabeli wyszukiwania:

```
tokenizer.index_word
{1: 'the',
 2: 'a',
 3: 'to',
 4: 'for',
 5: 'in',
 6: 'of',
 7: 'and',
 8: 's',
 9: 'on',
 10: 'with',
 11: 'show', ...
```

Możemy następnie wywołać to odwzorowanie z metodą `texts_to_sequences` naszego tokenizatora. To odwzorowuje każdą sekwencję słów reprezentowaną w wejściu tekstowym (tutaj zakładamy, że są to tytuły artykułów) na sekwencję tokenów odpowiadających poszczególnym słowom (patrz rysunek 2-7):

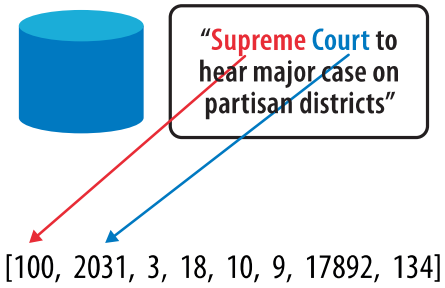
```
integerized_titles = tokenizer.texts_to_sequences(titles_df.title)
```

Tokenizator zawiera odpowiednie informacje, których użyjemy później do tworzenia warstwy osadzania. W szczególności `VOCAB_SIZE` przechwytuje liczbę elementów w tabeli wyszukiwania indeksu, a `MAX_LEN` zawiera maksymalną długość ciągów tekstowych w zbiorze danych:

```
VOCAB_SIZE = len(tokenizer.index_word)
MAX_LEN = max(len(sequence) for sequence in integerized_titles)
```

15 Ten zbiór danych jest dostępny w BigQuery: *bigquery-public-data.hacker_news.stories*.

16 *https://github.com/GoogleCloudPlatform/ml-design-patterns/blob/master/02_data_representation/embeddings.ipynb*



Rysunek 2-7 *Użycie tokenizatora sprawia, że każdy tytuł jest odwzorowany na sekwencję wartości indeksów całkowitoliczbowych.*

Przed utworzeniem modelu niezbędne jest wcześniejsze przetworzenie tytułów w zbiorze danych. Będzie trzeba dopełnić elementy tytułu w celu załadowania do modelu. Keras ma służące do tego pomocnicze funkcje `pad_sequence` ponad metodami tokenizatora. Funkcja `create_sequences` przyjmuje zarówno tytuły, jak i maksymalną długość zdania jako wejście i zwraca listę liczb całkowitych odpowiadających naszym tokenom dopełnionym do maksymalnej długości zdania:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

def create_sequences(texts, max_len=MAX_LEN):
    sequences = tokenizer.texts_to_sequences(texts)
    padded_sequences = pad_sequences(sequences,
                                     max_len,
                                     padding='post')
    return padded_sequences
```

Następnie będziemy budować model głębokiej sieci neuronowej (DNN, *deep neural network*) w Keras, który implementuje prostą warstwę osadzania do przekształcania liczb całkowitych słów na gęste wektory. Warstwa `Embedding` Keras może być uważana za odwzorowanie z indeksów całkowitoliczbowych konkretnych słów na gęste wektory (ich osadzenia). Liczba wymiarów osadzenia jest determinowana przez wymiar wyjścia (`output_dim`). Argument `input_dim` wskazuje rozmiar słownika, a `input_shape` wskazuje długość sekwencji wejściowych. Ponieważ tytuły zostały dopełnione przed przekazaniem do modelu, ustawiamy `input_shape=[MAX_LEN]`:

```
model = models.Sequential([layers.Embedding(input_dim=VOCAB_SIZE + 1,
                                            output_dim=embed_dim,
                                            input_shape=[MAX_LEN]),
                           layers.Lambda(lambda x: tf.reduce_mean(x,axis=1)),
                           layers.Dense(N_CLASSES, activation='softmax')])
```

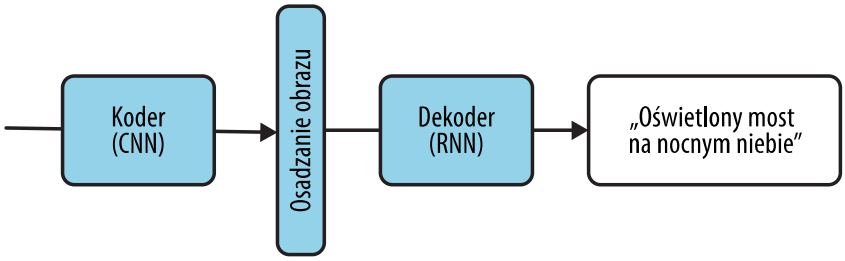
Zauważ, że trzeba umieścić niestandardową warstwę Keras Lambda między warstwą osadzania a gęstą warstwą softmax, aby uśrednić wektory słów zwracane przez warstwę osadzania. Jest to średnia, która jest przekazywana do gęstej warstwy softmax. W ten sposób tworzymy model, który jest prosty, ale traci informację o kolejności słów, tworząc model, który widzi zdanie jako „worek słów".

Osadzanie obrazów

Problemy tekstowe muszą radzić sobie z bardzo rzadkim wejściem, natomiast inne typy danych, takie jak obrazy lub audio, składają się z gęstych, wielowymiarowych wektorów, zwykle z wieloma kanałami zawierającymi surowe piksele lub informacje o częstotliwości. W tym środowisku Osadzanie przechwytuje odpowiednią niskowymiarową reprezentację wejścia.

W przypadku osadzania obrazów złożone konwolucyjne sieci neuronowe – takie jak Inception lub ResNet – są najpierw trenowane na dużym zestawie danych obrazów, takim jak ImageNet, zawierającym miliony obrazów i tysiące możliwych etykiet klasyfikujących. Następnie ostatnia warstwa softmax jest usuwana z modelu. Bez końcowej warstwy klasyfikatora softmax model może być używany do wyodrębniania wektora cech dla danego wejścia. Ten wektor cech zawiera wszystkie istotne informacje z obrazu, więc jest istotnie niskowymiarowym osadzeniem obrazu wejściowego.

Podobnie można rozważyć zadanie podpisywania obrazów, to jest generowania tekstowego podpisu dla danego obrazu, pokazane na rysunku 2-8.



Rysunek 2-8 *W przypadku zadania translacji obrazów koder wytwarza niskowymiarową reprezentację osadzenia obrazu.*

Przez trenowanie architektury tego modelu na olbrzymim zbiorze danych par obraz/podpis, koder uczy się skutecznej reprezentacji wektorowej obrazów. Dekoder uczy się, jak tłumaczyć ten wektor na podpis tekstowy. W tym sensie koder staje się maszyną osadzania Image2Vec.

Dlaczego to działa

Warstwa osadzania jest po prostu inną ukrytą warstwą w sieci neuronowej. Wagi są następnie przypisywane do każdego z wymiarów o wielkiej mocy, a wyjście jest przekazywane do reszty sieci. Dlatego wagi tworzące osadzanie są wyuczane w procesie spadku gradientu, tak jak dowolne inne wagi w sieci neuronowej. Oznacza to, że wynikowe osadzenia wektorowe reprezentują najbardziej skuteczną reprezentację niskowymiarową tych wartości cech z uwzględnieniem celu uczenia.

Chociaż to poprawne osadzanie ostatecznie pomaga modelowi, same osadzenia są wartościowe i pozawalają nam zyskać dodatkowy wgląd w zbiór danych.

Rozważmy ponownie zbiór danych klientów filmów. Użycie tylko kodowania 1 z n sprawia, że dowolni dwaj oddzielni użytkownicy user_i i user_j, będą mieli tę samą miarę podobieństwa. Podobnie iloczyn skalarny lub podobieństwo cosinusowe dla dowolnych