O'REILLY®

Helion

Matematyka i sztuczna inteligencja

Kluczowe koncepcje zwiększania skuteczności i wydajności systemów

Hala Nelson

Tytuł oryginału: Essential Math for AI: Next-Level Mathematics for Efficient and Successful AI Systems

Tłumaczenie: Radosław Meryk

ISBN: 978-83-289-1445-2

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 3

Dopasowywanie funkcji do danych

Dzisiaj wszystko pasuje. Co będzie jutro?

— H.

W tym rozdziale przedstawię podstawowe pojęcia matematyczne leżące u podstaw wielu aplikacji sztucznej inteligencji, w tym matematycznych silników sieci neuronowych. Moim celem jest zaprezentowanie czytelnikom następującej struktury części problemu sztucznej inteligencji, na którą składa się uczenie maszynowe:

Identyfikacja problemu

Problem zależy od konkretnego przypadku użycia: klasyfikowanie obrazów, klasyfikowanie dokumentów, przewidywanie cen domów, wykrywanie oszustw lub anomalii, rekomendowanie następnego produktu, przewidywanie prawdopodobieństwa ponownego popełnienia przestępstwa, przewidywanie wewnętrznej struktury budynku na podstawie zewnętrznych zdjęć, konwertowanie mowy na tekst, generowanie dźwięku, generowanie obrazów, generowanie wideo itp.

Pozyskiwanie odpowiednich danych

Chodzi o szkolenie modeli tak, by postępowały właściwie. Mówimy, że modele *uczą się* na podstawie danych. Należy zadbać o to, aby dane były czyste, kompletne a w razie potrzeby, w zależności od konkretnego modelu, przekształcone (znormalizowane, ustandaryzowane, z agregacją niektórych cech itp.). Ten krok jest zwykle znacznie bardziej czasochłonny niż implementacja i szkolenie modeli uczenia maszynowego.

Tworzenie funkcji hipotezy

Terminów *funkcja hipotezy*, *funkcja uczenia się*, *funkcja przewidywania*, *funkcja szkolenia* i *model* używamy zamiennie. Głównym założeniem jest to, że funkcja matematyczna wejścia-wyjścia wyjaśnia obserwowane dane i można ją później wykorzystać do przewidywania nowych danych. Przekazujemy modelowi określone cechy, np. codzienne nawyki osoby, a model zwraca prognozę, np. prawdopodobieństwo bezproblemowej spłaty pożyczki przez tę osobę. W tym rozdziale przekażemy modelowi długość ryby, a on zwróci jej wagę.

Znalezienie liczbowych wartości wag

Istnieje wiele modeli (włącznie z sieciami neuronowymi), w których funkcja szkoleniowa ma nieznane parametry zwane **wagami**. Celem jest znalezienie na podstawie danych wartości liczbowych tych wag. Po znalezieniu wartości wag można użyć przeszkolonej funkcji do tworzenia prognoz. Wystarczy przekazać do wzoru przeszkolonej funkcji cechy nowego punktu danych.

Utworzenie funkcji błędu

Aby znaleźć wartości nieznanych wag, tworzymy kolejną funkcję zwaną *funkcją błędu*, *funkcją kosztu*, *funkcją celu* lub *funkcją straty* (w dziedzinie sztucznej inteligencji każde pojęcie ma trzy lub więcej nazw). Taka funkcja powinna mierzyć pewien rodzaj odległości między źródłem prawdy a wygenerowanymi prognozami. Naturalnie chcemy, aby prognozy były jak najbliżej prawdy, więc szukamy wartości wag, które minimalizują funkcję straty. Z perspektywy matematyki rozwiązujemy problem minimalizacji. W istocie dziedzina *optymalizacji matematycznej* jest dla sztucznej inteligencji niezbędna.

Wybór wzorów matematycznych

W tym procesie to my jesteśmy inżynierami, więc to my decydujemy o wzorach matematycznych dla funkcji szkoleniowych, funkcji strat, metod optymalizacji i komputerowych implementacji. Różni inżynierowie stosują różne procesy i mają różne wyniki wydajności. Nie ma w tym niczego niewłaściwego. Ostatecznie sędzią jest wydajność wdrożonego modelu, a wbrew powszechnemu przekonaniu modele matematyczne są elastyczne i gdy zajdzie potrzeba, można je dostosowywać i modyfikować. Kluczowe znaczenie ma monitorowanie wydajności po wdrożeniu.

Znajdź sposób na wyszukiwanie minimalizatorów

Ponieważ celem jest znalezienie wartości wag, które minimalizują błąd między przewidywaniami modelu a prawdą, trzeba znaleźć skuteczny matematyczny sposób wyszukiwania **minimalizatorów**: specjalnych wartości wag, które generują najmniejszy błąd. Kluczową rolę w tym procesie odgrywa metoda **zstępowania gradientowego**. Jest to prosta metoda polegająca na obliczeniu *pierwszej pochodnej* funkcji błędu. Daje to jednak spore możliwości. Jest to jeden z powodów, dla których spędziliśmy połowę zajęć z rachunku różniczkowego na obliczaniu pochodnych (i gradientu — jest to pierwsza pochodna w wyższych wymiarach). Istnieją również inne metody, które wymagają obliczenia dwóch pochodnych. Spotkamy się z nimi w dalszej części książki. Kiedy to nastąpi, wymienię korzyści i wady korzystania z metod wyższego rzędu.

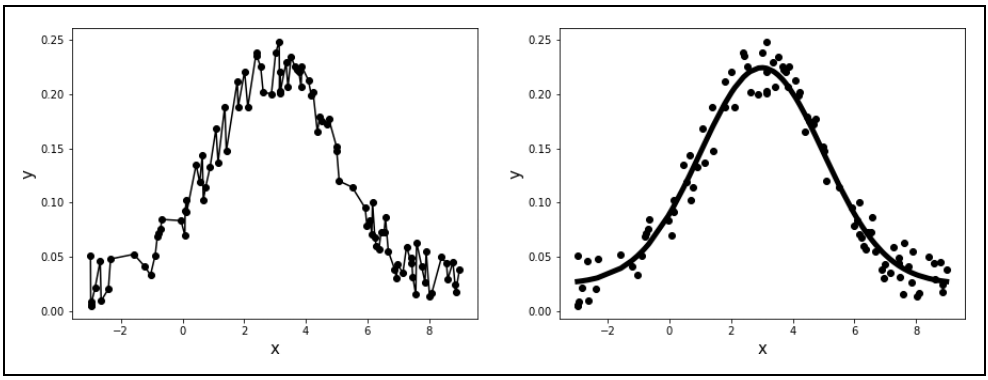
Wykorzystanie algorytmu propagacji wstecznej

Kiedy zbiory danych są ogromne, a model jest warstwową siecią neuronową, trzeba znaleźć skuteczny sposób na obliczenie tej jednej pochodnej. Do tego celu można wykorzystać *algorytm wstecznej propagacji*. Zagadnienia związane ze zstępowaniem gradientowym i propagacją wsteczną omówię w rozdziale 4.

Regularyzacja funkcji

Jeśli funkcja uczenia zbyt dobrze pasuje do danych, to nie będzie dobrze działać na nowych danych. Funkcja o zbyt dobrym dopasowaniu do danych wykrywa szumy w danych razem

z sygnałem (na przykład funkcja po lewej stronie na rysunku 3.1). Dążymy do tego, aby nie wykrywać szumów. Do tego celu potrzebna jest **regularyzacja**. Istnieje wiele matematycznych sposobów na regularyzację funkcji, co oznacza jej wygładzenie oraz wyeliminowanie oscylacji i nieregularności. Ogólnie rzecz biorąc, funkcja, która podąża za szumami w danych, ma zbyt wiele oscylacji. Potrzebne są bardziej regularne funkcje. Techniki regularyzacji omówię w rozdziale 4.



Rysunek 3.1. Po lewej: funkcja dopasowania idealnie pasuje do danych; nie jest to jednak dobra funkcja prognozowania, ponieważ dopasowuje szumy w danych zamiast głównego sygnału. Po prawej: bardziej regularna funkcja dopasowana do tego samego zbioru danych. Użycie tej funkcji zamiast funkcji z lewej części wykresu zapewni lepsze prognozy, nawet jeśli funkcja po lewej lepiej pasuje do punktów danych

W kolejnych podrozdziałach zbadamy strukturę problemu sztucznej inteligencji z wykorzystaniem rzeczywistych, ale prostych zestawów danych. W kolejnych rozdziałach przekonamy się, jak te same pojęcia uogólniają się na znacznie bardziej skomplikowane zadania.

Tradycyjne i bardzo przydatne modele uczenia maszynowego

Wszystkie dane użyte w tym rozdziale są *oznaczone etykietami* z wykorzystaniem źródeł prawdy, a celem tworzonych modeli jest *przewidywanie* etykiet nowych (niezaobserwowanych) i nieoznaczonych danych. Jest to uczenie *nadzorowane*.

W kilku kolejnych podrozdziałach spróbuję dopasować funkcje szkoleniowe do oznaczonych danych przy użyciu wymienionych poniżej popularnych modeli uczenia maszynowego. Chociaż wiele słyszy się o najnowszych i najlepszych osiągnięciach w dziedzinie sztucznej inteligencji, w typowym środowisku biznesowym najlepiej zacząć od następujących, bardziej tradycyjnych modeli:

Regresja liniowa

Prognozowanie wartości liczbowych.

Regresja logistyczna

Przyporządkowanie do dwóch klas (klasyfikacja binarna).

Regresja softmax

Przyporządkowanie do wielu klas.

Maszyny wektorów nośnych

Przyporządkowanie do dwóch klas lub regresja (przewidywanie wartości liczbowej).

Drzewa decyzyjne

Przyporządkowywanie do dowolnej liczby klas lub regresja (przewidywanie wartości liczbowej).

Lasy losowe

Przyporządkowywanie do dowolnej liczby klas lub regresja (przewidywanie wartości liczbowej).

Zespoły modeli

Łączenie wyników wielu modeli poprzez uśrednianie wartości prognoz, głosowanie na najpopularniejszą klasę lub zastosowanie innego mechanizmu łączenia.

Klasteryzacja k-średnich

Przyporządkowywanie do dowolnej liczby klas lub regresja.

W celu porównania wydajności wypróbowujemy wiele modeli na tych samych zestawach danych. W rzeczywistym świecie rzadko się zdarza, aby jakikolwiek model został wdrożony bez uprzedniego porównania go z wieloma innymi modelami. Taka jest natura branży sztucznej inteligencji. Obliczenia wymagają dużej mocy obliczeniowej, dlatego trzeba korzystać z obliczeń równoległych, które pozwalają na szkolenie wielu modeli jednocześnie (z wyjątkiem modeli, które są budowane i ulepszane na podstawie innych modeli, jak w przypadku *stackingu*; w takiej sytuacji nie można korzystać z obliczeń równoległych).

Zanim zagłębimy się w jakiekolwiek modele uczenia maszynowego, chcę zwrócić uwagę, że według relacji wielu osób zaledwie 5% czasu analityków danych czy badaczy sztucznej inteligencji zajmuje szkolenie modeli uczenia maszynowego. Większość czasu pochłaniają czynności wykonywane *przed* wprowadzeniem danych do modeli uczenia maszynowego: pozyskiwanie danych, oczyszczanie ich, organizowanie, tworzenie odpowiednich potoków dla danych itp. Uczenie maszynowe jest zatem tylko jednym z etapów procesu produkcyjnego i gdy dane są gotowe do szkolenia modelu, jest to łatwy krok. W dalszej części tej książki pokażę, jak działają modele uczenia maszynowego: większość matematyki, której trzeba użyć, dotyczy właśnie tych modeli. Badacze sztucznej inteligencji zawsze starają się ulepszać modele uczenia maszynowego i automatycznie dopasowywać je do potoków produkcyjnych. Dlatego ważne jest dokładne poznanie całego potoku, począwszy od surowych danych (włącznie z magazynowaniem, sprzętem, protokołami zapytań itp.), a skończywszy na wdrożeniu i monitorowaniu. Zapoznanie się z uczeniem maszynowym to tylko część większej i bardziej interesującej historii.

Zacznę od *regresji*, ponieważ idee regresji mają podstawowe znaczenie dla większości omówionych później modeli sztucznej inteligencji i ich zastosowań. W przypadku *regresji liniowej* minimalizujące wagi znajdujemy przy użyciu metody *analitycznej*. W ten sposób wyznacza się wyraźny wzór na pożądane wagi bezpośrednio w kategoriach zestawu danych szkoleniowych

i jego docelowych etykiet. Takie jednoznaczne rozwiązanie analityczne jest możliwe dzięki prostocie modelu regresji liniowej. Większość innych modeli nie ma tak jednoznacznych rozwiązań, a znalezienie minimalizatorów wymaga zastosowania metod numerycznych, wśród których niezwykle popularne jest zstępowanie gradientowe.

W regresji i wielu innych modelach, włącznie z sieciami neuronowymi, którymi zajmę się w kilku następnych rozdziałach, należy zwrócić uwagę na następującą sekwencję procesu modelowania:

1. Funkcja szkoleniowa.
2. Funkcja straty.
3. Optymalizacja.

Rozwiązania numeryczne a rozwiązania analityczne

Należy zdać sobie sprawę z różnic między rozwiązaniami numerycznymi i analitycznymi problemów matematycznych. Problemem matematycznym może być cokolwiek, np.:

- Znalezienie minimalizatora określonej funkcji.
- Znalezienie najlepszego sposobu na podróż z miejsca *A* do miejsca *B* przy ograniczonym budżecie.
- Znalezienie najlepszego sposobu zaprojektowania i odpytywania hurtowni danych.
- Znalezienie rozwiązania równania matematycznego (gdzie lewa strona wyrażenia matematycznego jest równa jego prawej stronie). Mogą to być równania algebraiczne, równania różniczkowe zwyczajne, równania różniczkowe cząstkowe, równania całkowo-różniczkowe, układy równań lub dowolne równania matematyczne. Ich rozwiązania mogą być statyczne lub mogą ewoluować w czasie. Można w nich modelować wszystko ze świata fizycznego, biologicznego, społeczno-ekonomicznego lub naturalnego.

Oto słownictwo:

Numeryczne
: Dotyczy liczb.

Analityczne
: Dotyczy analizy.

Z reguły rozwiązania numeryczne są znacznie łatwiejsze do uzyskania i znacznie bardziej dostępne niż rozwiązania analityczne, pod warunkiem że dysponujemy wystarczającą mocą obliczeniową do symulacji i obliczenia tych rozwiązań. Aby uzyskać rozwiązanie numeryczne, zazwyczaj wystarczy zdyskretyzować pewne ciągłe przestrzenie i (lub) funkcje, czyli zamienić kontinuum na zbiór punktów (co czasami wymaga zastosowania dość wymyślnych sposobów) i obliczyć wartość funkcji na podstawie tych dyskretnych wielkości. Jedynym problemem związanym z rozwiązaniami numerycznymi jest to, że są one jedynie rozwiązaniami przybliżonymi. Jeśli nie są poparte oszacowaniami dotyczącymi tego, jak mocno różnią się od

rzeczywistych rozwiązań analitycznych i jak szybko zbiegają się do tych rzeczywistych rozwiązań (co z kolei wymaga matematycznego zaplecza i analizy), rozwiązania numeryczne nie są dokładne. Dostarczają jednak niezwykle przydatnych informacji związanych z rzeczywistymi rozwiązaniami. W wielu przypadkach rozwiązania numeryczne są jedynymi dostępnymi, a wiele dziedzin nauki i inżynierii w ogóle nie posunęłoby się do przodu, gdyby nie polegały na numerycznych rozwiązaniach złożonych problemów. Gdyby w tych obszarach czekano na rozwiązania analityczne i dowody — lub mówiąc inaczej: aż teoria matematyczna nadrobi zaległości — postęp w nich byłby bardzo powolny.

Z drugiej strony rozwiązania analityczne są dokładne, solidne i poparte rozbudowaną teorią matematyczną. Towarzyszą im twierdzenia i dowody. Gdy dostępne są rozwiązania analityczne, są one bardzo wydajne. Nie są one jednak łatwo dostępne, czasem wręcz niemożliwe do zdobycia, a ponadto wymagają głębokiej wiedzy i doświadczenia w takich dziedzinach jak rachunek różniczkowy, analiza matematyczna, algebra, teoria równań różniczkowych itp. Metody analityczne są jednak niezwykle cenne w opisywaniu ważnych właściwości rozwiązań (nawet jeśli rozwiązania jawne nie są dostępne), sterowaniu technikami numerycznymi i dostarczaniu podstawowych prawd do porównywania przybliżonych metod numerycznych (w tych szczęśliwych przypadkach, gdy te rozwiązania analityczne są dostępne).

Niektórzy badacze są ściśle analityczni i teoretyczni, inni ściśle numeryczni i obliczeniowi, a najlepszą postawą jest zajęcie miejsca gdzieś pomiędzy, kiedy mamy przyzwoite rozumienie analitycznych i numerycznych aspektów problemów matematycznych.

Regresja — przewidywanie wartości liczbowej

Szybkie wyszukiwanie w witrynie Kaggle (*https://www.kaggle.com/*) zestawów danych do regresji zwraca wiele doskonałych zbiorów danych i powiązanych z nimi notatników. Losowo wybrałam prosty zestaw danych Fish Market (*https://www.kaggle.com/datasets/vipullrathod/fish-market*), którego użyjemy do wyjaśnienia zastosowanych poniżej reguł matematycznych. Naszym celem jest zbudowanie modelu, który przewiduje wagę ryby na podstawie pięciu różnych pomiarów jej długości lub cech, oznaczonych w zestawie danych jako *Length1* (długość), *Length2*, *Length3*, *Height* (wysokość) i *Width* (szerokość) — patrz rysunek 3.2. Dla uproszczenia zdecydowałam się nie włączać do tego modelu cechy kategorialnej, *Species* (gatunek), mimo że można by to zrobić (wpłynęłoby to na lepsze prognozy, ponieważ gatunek ryby jest dobrym predyktorem jej wagi). Gdybym zdecydowała się uwzględnić cechę *Species*, musielibyśmy przekonwertować jej wartości na wartości liczbowe z wykorzystaniem **kodowania one-hot**, co oznacza dokładnie to, na co wygląda: przypisanie każdemu gatunkowi ryby kodu składającego się z jedynek i zer. Cecha *Species* obejmuje siedem kategorii: *Perch* (okoń), *Bream* (leszcz), *Roach* (płoć), *Pike* (szczupak), *Smelt* (stynka), *Parkki* i *Whitefish* (sieja). Zatem szczupaka zakodowalibyśmy jako (0,0,0,1,0,0), natomiast leszcza jako (0,1,0,0,0,0). Oczywiście dodałoby to siedem dodatkowych wymiarów do przestrzeni cech modelu i siedem dodatkowych wag do przeszkolenia.

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

Rysunek 3.2. Pierwsze pięć wierszy zestawu danych Fish Market pobranych z serwisu Kaggle. Kolumna Weight *(waga) jest cechą docelową. Chcemy zbudować model, który przewiduje wagę ryby na podstawie pomiarów jej długości*

Aby zaoszczędzić pisania, oznaczymy pięć cech jako x_1, x_2, x_3, x_4 i x_5, a następnie zapiszemy wagę ryb jako funkcję tych pięciu cech, $y = f(x_1, x_2, x_3, x_4, x_5)$. W ten sposób po ustaleniu akceptowalnego wzoru dla funkcji trzeba jedynie wprowadzić wartości cech dla określonej ryby, a funkcja wyświetli przewidywaną wagę tej ryby.

Ten podrozdział buduje fundament dla pozostałej jego części, należy zatem zwrócić baczną uwagę na to, jak jest zorganizowany:

Funkcja szkoleniowa

- Modele parametryczne kontra modele nieparametryczne.

Funkcja straty

- Wartość przewidywana a wartość rzeczywista.
- Odległość wartości bezwzględnej w stosunku do odległości podniesionej do kwadratu.
- Funkcje z osobliwościami (punktami krytycznymi).
- W przypadku regresji liniowej funkcją straty jest błąd średniokwadratowy.
- Wektory w tej książce są zawsze wektorami kolumnowymi.
- Podzbiory szkoleniowe, walidacyjne i testowe.
- Gdy dane szkoleniowe mają wysoce skorelowane cechy.

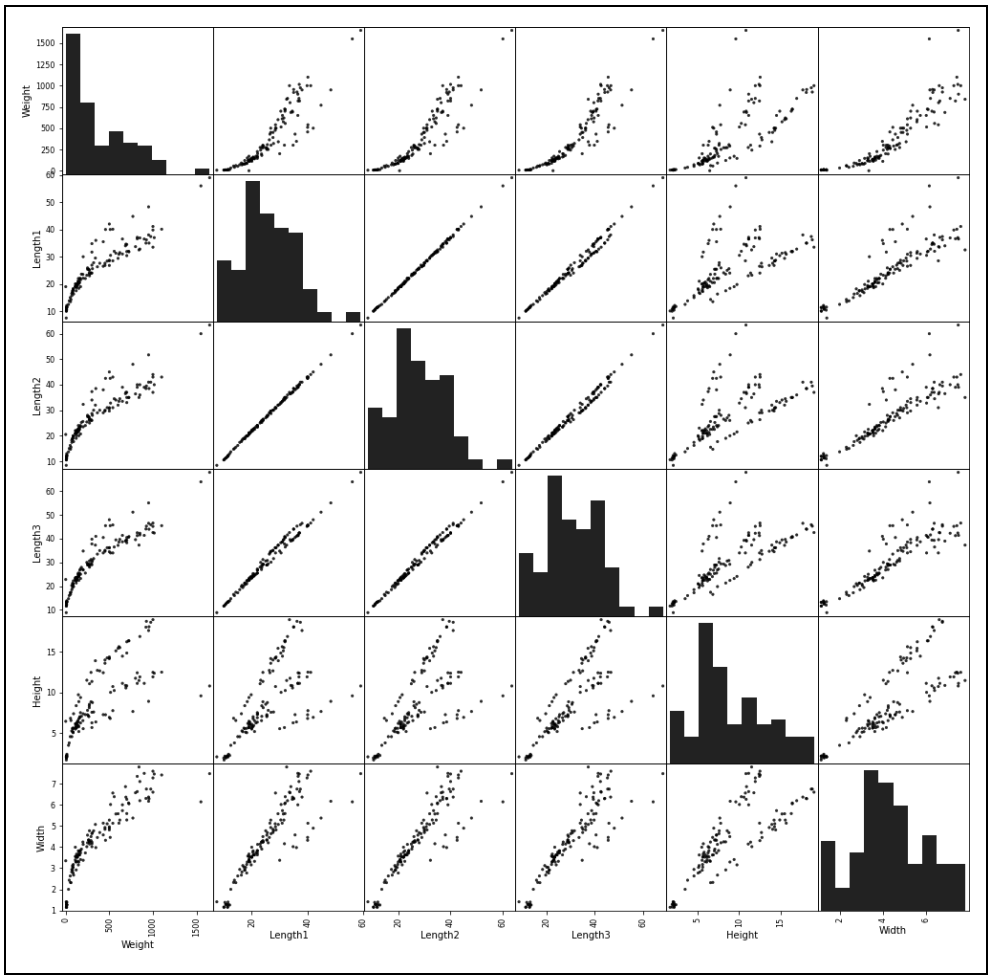
Optymalizacja

- Krajobrazy wypukłe kontra krajobrazy niewypukłe.
- Jak zlokalizować minimalizatory funkcji?
- Rachunek w pigułce.
- Przykład optymalizacji jednowymiarowej.
- Często używane pochodne wyrażeń algebry liniowej.
- Minimalizacja funkcji straty błędu średniokwadratowego.

- Uwaga: mnożenie dużych macierzy przez siebie jest bardzo kosztowne obliczeniowo — zamiast tego należy mnożyć macierze przez wektory.
- Uwaga: nigdy nie należy zbyt ściśle dopasowywać danych szkoleniowych.

Funkcja szkoleniowa

Szybka eksploracja danych, jak w przypadku wykreślania wagi względem różnych cech długości, pozwala założyć model liniowy (chociaż w tym przypadku model nieliniowy mógłby być lepszy). Oznacza to, że przyjęliśmy założenie, że waga zależy liniowo od cech długości (rysunek 3.3).



*Rysunek 3.3. Wykresy rozrzutu numerycznych cech zbioru danych Fish Market. Więcej szczegółów można znaleźć na stronie GitHub książki (*https://github.com/halanelson/Essential-Math-For-AI*) lub w niektórych publicznych notatnikach w serwisie Kaggle (*https://www.kaggle.com/aungpyaeap/fish-market/code*) powiązanych z tym zbiorem danych*

Oznacza to, że wagę ryby, y, można obliczyć za pomocą **liniowej kombinacji** jej pięciu różnych pomiarów długości, plus wyraz błędu systematycznego ω_0, co daje następującą funkcję szkoleniową:

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$$

Gdy po podjęciu głównej decyzji w procesie modelowania chce się użyć liniowej funkcji szkoleniowej $f(x_1,x_2,x_3,x_4,x_5)$, trzeba jedynie znaleźć odpowiednie wartości parametrów ω_0, ω_1, ω_2, ω_3, ω_4 i ω_5. Najlepsze wartości dla parametrów ω poznamy na podstawie danych. Proces korzystania z danych w celu znalezienia odpowiednich wartości ω nazywany jest **szkoleniem** modelu. *Przeszkolony* model to taki, w którym znaleziono wartości ω.

Ogólnie rzecz biorąc, funkcje szkoleniowe, zarówno liniowe, jak i nieliniowe, w tym te reprezentujące sieci neuronowe, mają nieznane parametry ω, których model musi się nauczyć na podstawie danych. W przypadku modeli liniowych każdy parametr nadaje każdej z cech określoną wagę w procesie przewidywania. Jeśli więc wartość ω_2 jest większa niż wartość ω_5, to przy założeniu, że druga i piąta cecha mają porównywalne skale, druga cecha odgrywa w przewidywaniu ważniejszą rolę niż piąta. Jest to jeden z powodów, dla których przed przystąpieniem do uczenia modelu należy skalować lub normalizować dane. Z drugiej strony, jeśli wartość ω_3 związana z trzecią cechą zaniknie, czyli stanie się zerowa lub nieistotna, to trzecią cechę w zbiorze danych można pominąć, ponieważ nie odgrywa ona w przewidywaniu żadnej roli. Dlatego uczenie się parametrów ω na podstawie danych pozwala matematycznie obliczyć wkład każdej cechy w prognozy (albo znaczenie kombinacji cech, jeśli przed szkoleniem, na etapie przygotowania danych, niektóre cechy zostały połączone). Inaczej mówiąc, modele uczą się, w jaki sposób cechy danych oddziałują na siebie i jak silne są te interakcje. Wniosek jest taki, że dzięki przeszkolonej funkcji uczenia można ilościowo określić, w jaki sposób cechy łączą się ze sobą w celu uzyskania zarówno zaobserwowanych, jak i jeszcze niezaobserwowanych wyników.

Modele parametryczne kontra modele nieparametryczne

Model, który ma wbudowane we wzór parametry (nazywamy je wagami), takie jak ω w bieżącym modelu regresji liniowej:

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$$

(a później ω w sieciach neuronowych), nazywany jest **modelem parametrycznym**. Oznacza to, że wzór funkcji szkoleniowej ustalamy przed faktycznym szkoleniem, a całe szkolenie polega na wyszukaniu parametrów wykorzystywanych we wzorze. Ustalenie wzoru z wyprzedzeniem to analogiczne działanie do określenia *rodziny*, do której należy funkcja szkoleniowa, a znalezienie wartości parametrów określa członka tej rodziny, który najlepiej wyjaśnia dane.

Modele nieparametryczne, takie jak drzewa decyzyjne i lasy losowe, które omówię w dalszej części tego rozdziału, nie określają z wyprzedzeniem wzoru na funkcję szkoleniową wraz z jej parametrami. Kiedy więc szkolimy model nieparametryczny, nie wiemy, ile parametrów będzie miał przeszkolony model. Model dostosowuje się do danych i określa wymaganą liczbę parametrów w zależności od danych. Ostrożnie, już słychać dzwony nadmiernego

dopasowania! Przypomnę, że zbytnie dopasowanie modeli do danych jest niekorzystne, ponieważ takie modele mogą nie być właściwie uogólnione w odniesieniu do danych niezaobserwowanych. Modelom nieparametrycznym zwykle towarzyszą techniki, które pomagają uniknąć nadmiernego dopasowania.

Zarówno modele parametryczne, jak i nieparametryczne mają inne parametry zwane **hiperparametrami**, które również trzeba dostroić podczas procesu uczenia. Nie są one jednak wbudowane we wzór funkcji szkoleniowej (i nie są również częścią wzoru w modelu nieparametrycznym). W tej książce spotkamy się z wieloma hiperparametrami.

Funkcja straty

Przekonaliśmy się, że następnym logicznym krokiem jest znalezienie odpowiednich wartości dla parametrów ω pojawiających się w funkcji szkoleniowej (liniowego modelu parametrycznego) przy użyciu danych, które posiadamy. Aby to zrobić, trzeba *zoptymalizować odpowiednią funkcję straty*.

Wartość przewidywana a wartość rzeczywista

Załóżmy, że przypisujemy losowe wartości liczbowe dla każdej z nieznanych wartości ω_0, ω_1, ω_2, ω_3, ω_4 i ω_5 — na przykład $\omega_0 = -3$, $\omega_1 = 4$, $\omega_2 = 0{,}2$, $\omega_3 = 0{,}03$, $\omega_4 = 0{,}4$ i $\omega_5 = 0{,}5$. W takim przypadku wzór na liniową funkcję szkoleniową $y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$ przyjmuje następującą postać:

$$y = -3 + 4x_1 + 0{,}2x_2 + 0{,}03x_3 + 0{,}4x_4 + 0{,}5x_5$$

a model jest gotowy do prognozowania. Aby uzyskać przewidywaną wartość wagi ryby i^{th}, wystarczy podstawić wartości liczbowe dla cech długości tej ryby. Na przykład pierwsza ryba w zestawie danych to leszcz, dla którego pomiary długości wynoszą $x_1^1 = 23{,}2$, $x_2^1 = 25{,}4$, $x_3^1 = 30$, $x_4^1 = 11{,}52$, i $x_5^1 = 4{,}02$. Po podstawieniu tych wartości do funkcji szkoleniowej otrzymujemy prognozę wagi tej ryby:

$$\begin{aligned} y^1_{\text{prognoza}} &= \omega_0 + \omega_1 x_1^1 + \omega_2 x_2^1 + \omega_3 x_3^1 + \omega_4 x_4^1 + \omega_5 x_5^1 \\ &= -3 + 4(23{,}2) + 0{,}2(25{,}4) + 0{,}03(30) + 0{,}4(11{,}52) + 0{,}5(4{,}02) \\ &= 102{,}4 \text{ grama} \end{aligned}$$

Ogólnie rzecz biorąc, wzór na wagę i-tej ryby jest następujący:

$$y^i_{\text{rzeczywista}} = \omega_0 + \omega_1 x_1^i + \omega_2 x_2^i + \omega_3 x_3^i + \omega_4 x_4^i + \omega_5 x_5^i$$

Analizowana ryba ma jednak pewną prawdziwą wagę, $y^i_{\text{rzeczywista}}$, która w przypadku przynależności do oznaczonego zbioru danych jest jej etykietą. W przypadku pierwszej ryby w zestawie danych rzeczywista waga wynosi $y^i_{\text{rzeczywista}} = 242$ gramy. Model liniowy z losowo wybranymi wartościami ω przewidywał wagę 102,4 grama. Prognozowana wartość jest oczywiście dość odległa od rzeczywistej, ponieważ w ogóle nie skalibrowaliśmy wartości ω. W każdym przypadku można zmierzyć błąd między wagą przewidzianą przez model a wagą rzeczywistą, a następnie znaleźć sposób na uzyskanie lepszych wartości parametrów ω.

Odległość bezwzględna a odległość podniesiona do kwadratu

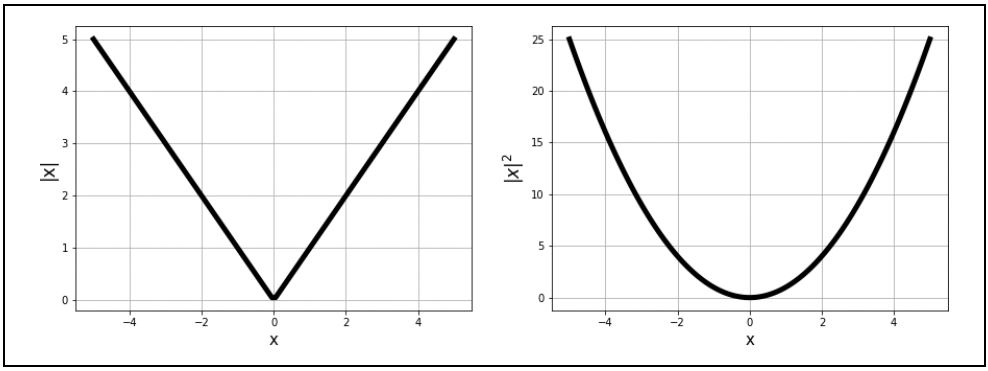
Jedną z interesujących własności matematyki jest istnienie wielu sposobów mierzenia odległości między przedmiotami, z wykorzystaniem różnych miar odległości. Na przykład można naiwnie zmierzyć odległość między dwiema wielkościami jako 1, jeśli są różne, i 0, jeśli są takie same, poprzez zakodowanie słów: *różne* = 1, *podobne* = 0. Oczywiście, gdy skorzystamy z takiej naiwnej metryki, stracimy mnóstwo informacji, ponieważ odległość między takimi wielkościami jak 2 i 10 będzie równa odległości między 2 a 1 milionem, czyli będzie wynosiła 1.

W uczeniu maszynowym istnieje kilka popularnych metryk odległości. Najpierw przedstawimy dwie najczęściej używane:

- Odległość bezwzględna: $|y_{prognoza} - y_{rzeczywista}|$ wynikająca z funkcji wartości bezwzględnej $|x|$.
- Odległość kwadratowa: $|y_{prognoza} - y_{rzeczywista}|^2$ wynikająca z funkcji rachunku różniczkowego $|x|^2$ (która dla wartości skalarnych jest równoważna funkcji x^2). Oczywiście spowoduje to również podniesienie jednostek do kwadratu.

Wystarczy spojrzeć na wykresy funkcji $|x|$ i x^2 na rysunku 3.4, aby zauważyć dużą różnicę w gładkości funkcji w punkcie (0, 0). Funkcja $|x|$ załamuje się w tym punkcie, co czyni ją nieróżniczkowalną w punkcie $x = 0$. Ta *osobliwość* funkcji $|x|$ w punkcie $x = 0$ powoduje, że wiele osób (i matematyków!) odrzuca możliwość uwzględnienia tej funkcji lub funkcji z podobnymi osobliwościami w swoich modelach. Należy jednak zapamiętać poniższą zasadę.

Modele matematyczne są elastyczne. Kiedy napotkasz przeszkodę, spróbuj przeanalizować problem dokładniej, aby zrozumieć, co się dzieje, a następnie obejdź tę przeszkodę.



Rysunek 3.4. Po lewej: wykres funkcji |x| *załamuje się w punkcie* x = *0, przez co jego pochodna jest w tym punkcie niezdefiniowana. Po prawej: wykres funkcji* $|x|^2$ *w punkcie* x = *0 jest gładki, więc nie ma problemów z wyznaczeniem w tym punkcie pochodnej*

Oprócz różnicy w regularności funkcji $|x|$ i $|x^2|$ (co oznacza możliwość lub brak możliwości wyznaczenia pochodnej we wszystkich punktach) przed podjęciem decyzji o włączeniu którejkolwiek z tych funkcji do wzoru na błąd trzeba uwzględnić jeszcze jedną kwestię: *jeśli liczba*

ma dużą wartość, to jej kwadrat jest jeszcze większy. Z tej prostej obserwacji wynika, że jeśli zdecydujemy się zmierzyć błąd za pomocą kwadratów odległości między wartościami rzeczywistymi a przewidywanymi, to taka metoda będzie *bardziej wrażliwa na* wartości odstające w danych. Jedna wartość odstająca może przechylić całą funkcję predykcji w kierunku tej wartości, a tym samym z dala od bardziej poprawnych wzorców w danych. Byłoby idealnie, gdyby było można zająć się wartościami odstającymi i zdecydować, czy powinny być zachowane, czy nie, na etapie przygotowywania danych, jeszcze przed ich wprowadzeniem do modelu uczenia maszynowego.

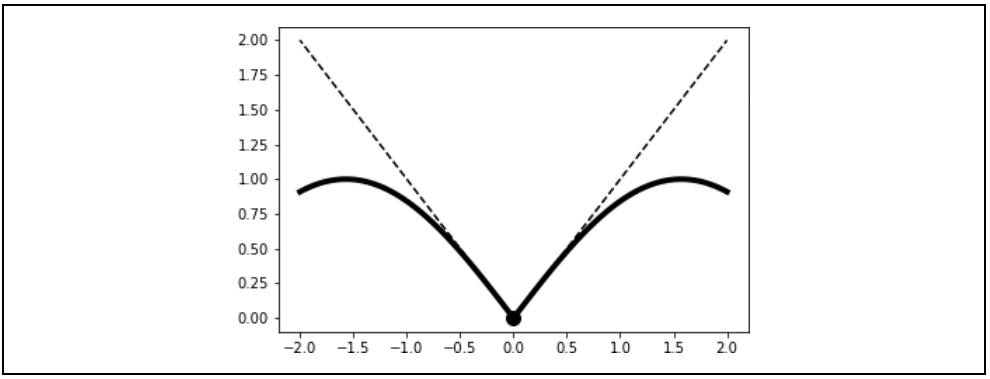
Ostatnia różnica między funkcją $x|$ (i podobnymi funkcjami liniowymi) a $|x|^2$ (i podobnymi funkcjami nieliniowymi, ale różniczkowalnymi) polega na tym, że pochodna funkcji $x|$ jest bardzo prosta:

1 dla $x > 0$; –1 dla $x < 0$ (i niezdefiniowana dla $x = 0$)

W modelu obejmującym miliardy kroków obliczeniowych cecha polegająca na *braku konieczności oceny czegokolwiek* w przypadku korzystania z pochodnej $|x|$ okazuje się niezwykle cenna. Pochodne funkcji, które nie są ani liniowe, ani częściowo liniowe, zwykle wymagają obliczeń (ponieważ również mają x we wzorach, a nie tylko stałe, jak w przypadku funkcji częściowo liniowych), co w przypadku dużych zbiorów danych może wiązać się z kosztami obliczeniowymi.

Funkcje z osobliwościami

Ogólnie rzecz biorąc, wykresy funkcji *różniczkowalnych* nie mają wcięć, załamań, narożników ani żadnych elementów spiczastych. Jeśli mają takie *osobliwości*, to funkcja w tych punktach nie ma pochodnej. Wynika to z faktu, że w punkcie spiczastym można narysować dwie różne linie styczne do wykresu funkcji, w zależności od tego, czy zdecydujemy się narysować linię styczną po lewej, czy po prawej stronie punktu (rysunek 3.5). Przypomnę, że pochodną funkcji w punkcie jest nachylenie linii stycznej do wykresu funkcji w tym punkcie. Jeśli dla określonego punktu istnieją dwa *różne* nachylenia, to w takim punkcie nie można zdefiniować pochodnej.



Rysunek 3.5. W punktach osobliwych pochodna nie istnieje. W takich punktach istnieje więcej niż jedno możliwe nachylenie stycznej

Nieciągłość w nachyleniu stycznej stwarza problem dla metod, które opierają się na ocenie pochodnej funkcji, takich jak metoda zstępowania gradientowego. Występują tu dwa problemy:

Niezdefiniowana pochodna

Jakiej wartości pochodnej należy użyć? Jeśli zdarzy Ci się znaleźć dziwny, spiczasty punkt na wykresie, metoda nie będzie wiedziała, co zrobić, ponieważ nie będzie w tym punkcie zdefiniowanej pochodnej. Niektórzy przypisują wartość pochodnej w tym punkcie (zwaną **subgradientem** lub **subróżnicą**) i przechodzą dalej. Jakie są szanse, że pechowo będziemy szukać wartości funkcji dokładnie w tym jednym, niewygodnym punkcie? O ile krajobraz funkcji nie wygląda jak trudny teren Alp (w rzeczywistości wiele z nich tak wygląda), metoda numeryczna może ominąć wszystkie trudne punkty.

Niestabilność

Innym problemem jest niestabilność. Ponieważ podczas analizowania punktów na wykresie funkcji wokół punktu osobliwego wartość pochodnej gwałtownie skacze, metoda wykorzystująca pochodną również gwałtownie zmienia wartość, co w przypadku poszukiwania zbieżności spowoduje niestabilność. Wyobraź sobie, że wędrujesz po szwajcarskich Alpach pokazanych na rysunku 3.6 (krajobraz funkcji straty), a Twoim celem jest to ładne miasteczko, które można zobaczyć w dolinie (miejsce o najniższej wartości błędu). Nagle zostajesz przeniesiony przez jakiegoś kosmitę (kosmitą jest matematyczna metoda wyszukiwania wykorzystująca nagle zmieniającą się pochodną) na *drugą* stronę góry, gdzie nie widzisz już swojego celu. Teraz widzisz w dolinie tylko parę brzydkich krzewów i bardzo wąski kanion, który może Cię uwięzić, jeśli Twoja metoda Cię tam zaprowadzi. Zbieżność z pierwotnym celem jest teraz niestabilna, a być może całkiem ją utraciłeś.

Rysunek 3.6. Szwajcarskie Alpy. Optymalizacja przypomina wędrówkę po wykresie funkcji

Niemniej jednak funkcje z osobliwościami takimi jak opisana powyżej ciągle są wykorzystywane w uczeniu maszynowym. Spotykamy je we wzorach niektórych funkcji szkoleniowych sieci neuronowych (rektyfikowana liniowa funkcja jednostkowa; kto wymyśla takie nazwy?), w niektórych funkcjach strat (odległość wartości bezwzględnej) i w niektórych wyrażeniach regularyzujących (regresja lasso; znów te nazwy).

W przypadku regresji liniowej funkcją straty jest błąd średniokwadratowy

Wracając do głównego celu tego podrozdziału: skonstruowania funkcji błędu, zwanej również *funkcją straty*, która koduje liczbę błędów popełnianych przez model podczas dokonywania prognoz i musi mieć niskie wartości.

W przypadku regresji liniowej używamy **funkcji błędu średniokwadratowego**. Wspomniana funkcja oblicza średnią błędów kwadratów odległości między wartością prognozowaną a rzeczywistą dla m punktów danych (wkrótce wskażę te punkty danych, które należy uwzględnić):

$$\text{Średni błąd kwadratowy} = \frac{1}{m}\left(\left|y_{\text{prognoza},1} - y_{\text{rzeczywista},1}\right|^2 + \left|y_{\text{prognoza},2} - y_{\text{rzeczywista},2}\right|^2 + \cdots + \left|y_{\text{prognoza},m} - y_{\text{rzeczywista},m}\right|^2\right)$$

Spróbujmy zapisać powyższe wyrażenie w bardziej zwięzły sposób z wykorzystaniem notacji sumy:

$$\text{Średni błąd kwadratowy} = \frac{1}{m}\sum_{i=1}^{m}\left|y_{\text{prognoza},i} - y_{\text{rzeczywista},i}\right|^2$$

Nabieramy nawyku używania jeszcze bardziej kompaktowej notacji algebry liniowej wektorów i macierzy. Ten nawyk okazuje się niezwykle przydatny w praktycznych zadaniach (nie chcemy utonąć przy próbie śledzenia indeksów). Indeksy mogą wkraść się w nasze beztroskie sny, w których wszystko jest zrozumiałe, i szybko przekształcić je w przerażające koszmary. Innym ważnym powodem, dla którego warto używać kompaktowej notacji algebry liniowej, jest to, że zarówno oprogramowanie, jak i sprzęt zbudowany dla modeli uczenia maszynowego są zoptymalizowane pod kątem obliczeń macierzowych i *tensorowych* (pomyśl o obiekcie zbudowanym z warstwowych macierzy jak o trójwymiarowym pudełku, a nie jak o płaskim kwadracie). Co więcej, piękna dziedzina numerycznej algebry liniowej rozwiązała wiele potencjalnych problemów i utorowała drogę do korzystania z szybkich metod wykonywania wszelkiego rodzaju obliczeń macierzowych.

Jeśli skorzystamy z notacji algebry liniowej, będziemy mogli zapisać średni błąd kwadratowy następująco:

$$\text{Średni błąd kwadratowy} = \frac{1}{m}\left(\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right)^t\left(\vec{y}_{prognoza} - \vec{y}_{\text{rzeczywista}}\right)$$
$$= \frac{1}{m}\left\|\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right\|_{l^2}^2$$

W ostatnim równaniu wprowadziliśmy normę l^2 wektora, która z definicji to po prostu pierwiastek z sumy kwadratów jego składowych.

Pomysł na przyszłość: *funkcja straty, którą skonstruowaliśmy, koduje różnicę między przewidywaniami a prawdą dla punktów danych wykorzystanych w procesie uczenia, mierzoną z wykorzystaniem pewnej normy — jednostki matematycznej, która spełnia rolę odległości*. Można było skorzystać z wielu innych norm, ale *norma l^2* jest dość popularna.

Notacja — wektory w tej książce zawsze są kolumnowe

Aby zachować spójność notacji w całej książce, *wszystkie* wektory są kolumnowe. Zatem jeśli wektor $\vec{v}$ ma cztery składowe, symbol $\vec{v}$ oznacza $\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$.

Transpozycja wektora $\vec{v}$ jest więc zawsze wektorem wierszowym. Transpozycją powyższego wektora z czterema składowymi jest $\vec{v} = (v_1\ v_2\ v_3\ v_4)$.

Możemy również skorzystać z notacji iloczynu kropkowego (zwanego również iloczynem skalarnym, ponieważ mnożymy dwa wektory, ale wynik jest wartością skalarną). Iloczyn skalarny dwóch wektorów $\vec{a} \cdot \vec{b}$ jest tym samym co $\vec{a}^t\vec{b}$. W gruncie rzeczy w zapisie $\vec{a}^t\vec{b}$ wektor kolumnowy oznacza macierz o kształcie *długość wektora* × *1*, a jego transpozycja to macierz o kształcie *1* × *długość wektora*.

Załóżmy teraz, że $\vec{a}$ i $\vec{b}$ mają cztery składniki. W takim przypadku:

$$\vec{a}^t\vec{b} = (a_1\ a_2\ a_3\ a_4)\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4 = \sum_{i=1}^{4} a_ib_i$$

Ponadto:

$$\| \vec{a} \|_{l^2}^2 = \vec{a}^t\vec{a} = a_1^2 + a_2^2 + a_3^2 + a_4^2$$

Podobnie:

$$\| \vec{b} \|_{l^2}^2 = \vec{b}^t\vec{b} = b_1^2 + b_2^2 + b_3^2 + b_4^2$$

W ten sposób używamy notacji macierzowej. Strzałka nad literą oznacza, że mamy do czynienia z wektorem kolumnowym.

Podzbiory szkoleniowe, walidacyjne i testowe

Które punkty danych uwzględniamy w funkcji straty? Czy uwzględniamy cały zestaw danych, niektóre jego małe partie, czy nawet tylko jeden punkt? Czy mierzymy średni błąd kwadratowy dla punktów danych w *podzbiorze szkoleniowym*, *podzbiorze walidacyjnym*, czy *podzbiorze testowym*? Czym właściwie są te podzbiory?

W praktyce zestaw danych dzielimy na trzy podzbiory:

Podzbiór szkoleniowy

Podzbiór danych, którego używamy do dopasowania funkcji szkoleniowej. Oznacza to, że punkty danych w tym podzbiorze są uwzględniane w funkcji straty (poprzez uwzględnienie wartości ich cech i etykiet w wyrazach $y_{prognoza}$ i $y_{rzeczywista}$ funkcji straty).

Podzbiór walidacyjny

Punkty danych należące do tego podzbioru są wykorzystywane na wiele sposobów:

- Powszechnie spotyka się opis, zgodnie z którym używamy tego podzbioru do *dostrojenia hiperparametrów modelu uczenia maszynowego*. Hiperparametry to wszelkiego rodzaju parametry w modelu uczenia maszynowego niebędące parametrami ω funkcji szkoleniowej, którą próbujemy rozwiązać. W uczeniu maszynowym jest ich wiele, a ich wartości wpływają na wyniki i wydajność modelu. Wśród przykładów hiperparametrów znajdują się następujące elementy, które warto poznać (choć nie musisz jeszcze wiedzieć, czym dokładnie są):
  - Szybkość uczenia się w metodzie zstępowania gradientowego.
  - Hiperparametr określający szerokość marginesu w metodach maszyny wektorów nośnych.
  - Procentowy udział danych wejściowych podzielonych na podzbiory szkoleniowy, walidacyjny i testowy.
  - Rozmiar partii przy stosowaniu metody randomizowanego zstępowania gradientowego z losowymi partiami.
  - Hiperparametry związane z rozkładem wag, jakie stosuje się w regresji grzbietowej, lasso i elastycznej sieci.
  - Hiperparametry towarzyszące metodom z wykorzystaniem pędu, takim jak zstępowanie gradientowe z pędem (ang. *gradient descent with momentum*) i ADAM (ang. *Adaptive Moment Estimation*). Te metody zawierają elementy przyspieszające zbieganie metody do minimum. Wspomniane elementy są mnożone przez hiperparametry, które należy dostroić przed fazą testowania i wdrożenia.
  - Liczba *epok* w procesie optymalizacji, czyli liczba przebiegów przez cały podzbiór szkoleniowy obserwowana przez optymalizator.
  - Architektura sieci neuronowej, w tym liczba warstw oraz szerokość każdej warstwy.
  - Podzbiór walidacyjny pomaga także ustalić, kiedy należy przerwać optymalizację, aby nie dopuścić do nadmiernego dopasowania podzbioru szkoleniowego.
  - Spełnia także rolę zbioru testowego do porównywania wydajności różnych modeli uczenia maszynowego na tym samym zbiorze danych, na przykład porównywania wydajności modelu regresji liniowej z metodą losowego lasu i siecią neuronową.

Podzbiór testowy

Po zdecydowaniu o najlepszym modelu do wykorzystania (lub uśrednieniu bądź zagregowaniu wyników wielu modeli) i przeszkoleniu modelu używamy tego nietkniętego podzbioru danych jako testu wykonywanego na ostatnim etapie przed wdrożeniem modelu do produkcji. Ponieważ model nie obserwował wcześniej żadnego z punktów danych z tego podzbioru (co oznacza, że nie uwzględnił żadnego z nich w procesie optymalizacji), można go uznać za najbliższą analogię rzeczywistej sytuacji. Pozwala to ocenić wydajność modelu przed zastosowaniem go do zupełnie nowych rzeczywistych danych.

Podsumowanie

Zanim przejdę dalej, spróbuję podsumować wiedzę, którą zaprezentowałam dotychczas:

- Bieżący model uczenia maszynowego nazywamy *regresją liniową*.
- Funkcja szkoleniowa jest liniowa, zgodnie ze wzorem:

  $$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$$

  Wyrazy x w tym wzorze to cechy, natomiast ω to nieznane wagi, czyli parametry.
- Jeśli podstawimy do wzoru funkcji szkoleniowej wartości cech konkretnego punktu danych — na przykład dziesiątego — otrzymamy prognozy modelu dla tego punktu:

  $$y^{10}_{\text{prognoza}} = \omega_0 + \omega_1 x^{10}_1 + \omega_2 x^{10}_2 + \omega_3 x^{10}_3 + \omega_4 x^{10}_4 + \omega_5 x^{10}_5$$

  Indeks górny 10 wskazuje, że są to wartości odpowiadające dziesiątemu punktowi danych.
- W roli funkcji straty wykorzystamy funkcję błędu średniokwadratowego o następującym wzorze:

  $$\text{Średni błąd kwadratowy} = \frac{1}{m}\left(\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right)^t \left(\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right)$$
  $$= \frac{1}{m}\left\|\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right\|^2_{l^2}$$
- Chcemy znaleźć wartości ω, które minimalizują tę funkcję straty. Następnym krokiem musi być więc rozwiązanie problemu minimalizacji (optymalizacji).

Aby znacznie ułatwić optymalizację, po raz kolejny zastosujemy wygodną notację algebry liniowej (z wykorzystaniem wektorów i macierzy). Dzięki temu można uwzględnić we wzorze funkcji straty cały podzbiór danych szkoleniowych jako macierz i natychmiast wykonać obliczenia na całym podzbiorze szkoleniowym, w przeciwieństwie do obliczeń na każdym punkcie danych z osobna. Ten prosty manewr pozwala uniknąć wielu błędów, problemów i żmudnych obliczeń z wieloma komponentami, które są trudne do śledzenia na bardzo dużych zbiorach danych.

Po pierwsze zapiszemy prognozę modelu odpowiadającą poszczególnym punktom danych z podzbioru szkoleniowego:

$$\begin{aligned}
y^1_{\text{prognoza}} &= 1\omega_0 + \omega_1 x^1_1 + \omega_2 x^1_2 + \omega_3 x^1_3 + \omega_4 x^1_4 + \omega_5 x^1_5 \\
y^2_{\text{prognoza}} &= 1\omega_0 + \omega_1 x^2_1 + \omega_2 x^2_2 + \omega_3 x^2_3 + \omega_4 x^2_4 + \omega_5 x^2_5 \\
&\vdots \\
y^m_{\text{prognoza}} &= 1\omega_0 + \omega_1 x^m_1 + \omega_2 x^m_2 + \omega_3 x^m_3 + \omega_4 x^m_4 + \omega_5 x^m_5
\end{aligned}$$

Powyższy układ równań można z łatwością zapisać w postaci:

$$\begin{pmatrix} y^1_{\text{prognoza}} \\ y^2_{\text{prognoza}} \\ \vdots \\ y^m_{\text{prognoza}} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \omega_0 + \begin{pmatrix} x^1_1 \\ x^2_1 \\ \vdots \\ x^m_1 \end{pmatrix} \omega_1 + \begin{pmatrix} x^1_1 \\ x^2_2 \\ \vdots \\ x^m_2 \end{pmatrix} \omega_2 + \begin{pmatrix} x^1_3 \\ x^2_3 \\ \vdots \\ x^m_3 \end{pmatrix} \omega_3 + \begin{pmatrix} x^1_4 \\ x^2_4 \\ \vdots \\ x^m_4 \end{pmatrix} \omega_4 + \begin{pmatrix} x^1_5 \\ x^2_5 \\ \vdots \\ x^m_5 \end{pmatrix} \omega_5$$

Albo jeszcze lepiej:

$$\begin{pmatrix} y^1_{\text{prognoza}} \\ y^2_{\text{prognoza}} \\ \vdots \\ y^m_{\text{prognoza}} \end{pmatrix} = \begin{pmatrix} 1 & x^1_1 & x^1_2 & x^1_3 & x^1_4 & x^1_5 \\ 1 & x^2_1 & x^2_2 & x^2_3 & x^2_4 & x^2_5 \\ \vdots & & & & & \\ 1 & x^m_1 & x^m_2 & x^m_3 & x^m_4 & x^m_5 \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \omega_5 \end{pmatrix}$$

Wektor po lewej stronie tego równania to $\vec{y}_{\text{prognoza}}$, macierz po prawej stronie to podzbiór szkoleniowy X powiększony o wektor jedynek, a ostatni wektor po prawej stronie zawiera wszystkie nieznane wagi. Oznaczmy ten wektor przez $\vec{\omega}$. Następnie możemy kompaktowo zapisywać $\vec{y}_{\text{prognoza}}$ w kategoriach podzbioru szkoleniowego i $\vec{\omega}$ w następujący sposób:

$$\vec{y}_{\text{prognoza}} = X\vec{\omega} \text{ T}$$

Teraz wzór funkcji straty błędu średniokwadratowego, który wcześniej zapisaliśmy jako:

$$\text{Średni błąd kwadratowy} = \frac{1}{m}\left(\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right)^t\left(\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right)$$
$$= \frac{1}{m}\left\|\vec{y}_{\text{prognoza}} - \vec{y}_{\text{rzeczywista}}\right\|^2_{l^2}$$

przyjmie postać:

$$\text{Średni błąd kwadratowy} = \frac{1}{m}\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right)^t\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right)$$
$$= \frac{1}{m}\left\|X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right\|^2_{l^2}$$

Jesteśmy teraz gotowi do znalezienia wartości $\vec{\omega}$, która minimalizuje zgrabnie zapisaną funkcję straty. W tym celu musimy odwiedzić bogatą i piękną matematyczną dziedzinę optymalizacji.

Gdy dane szkoleniowe mają wysoce skorelowane cechy

Gdy przyjrzymy się macierzy szkoleniowej (powiększonej o wektor jedynek):

$$X = \begin{pmatrix} 1 & x^1_1 & x^1_2 & x^1_3 & x^1_4 & x^1_5 \\ 1 & x^2_1 & x^2_2 & x^2_3 & x^2_4 & x^2_5 \\ \vdots & & & & & \\ 1 & x^m_1 & x^m_2 & x^m_3 & x^m_4 & x^m_5 \end{pmatrix}$$

która pojawia się w wektorze $\vec{y}_{\text{prognoza}} = X\vec{\omega}$, wzorowi funkcji straty błędu średniokwadratowego, a następnie wzorowi określającemu niewiadomą $\vec{\omega}$ (zwanemu również *równaniem normalnym*):

$$\vec{\omega} = (X^tX)^{-1}X^t\vec{y}_{\text{rzeczywista}}$$

zauważymy, że model może mieć problem w przypadku, gdy między dwiema lub większą liczbą cech (kolumn x) danych występują silne korelacje. Oznacza to, że istnieje silna liniowa zależność między cechami, dzięki czemu jedną z tych cech można obliczyć za pomocą liniowej

kombinacji pozostałych. W związku z tym odpowiednie kolumny cech *nie są liniowo niezależne* (czyli są bliskie bycia liniowo zależnymi). W przypadku macierzy jest to problem, ponieważ oznacza, że macierz albo nie może być odwrócona, albo jest *źle uwarunkowana*. Źle uwarunkowane macierze powodują duże niestabilności w obliczeniach, ponieważ niewielkie zmiany w danych szkoleniowych (co należy założyć) powodują duże zmiany w parametrach modelu, a tym samym sprawiają, że jego prognozy stają się niewiarygodne.

Macierze wykorzystywane w obliczeniach powinny być dobrze uwarunkowane, więc trzeba się pozbyć źródeł złego uwarunkowania. W przypadku wysoce skorelowanych cech jedną z możliwości jest uwzględnienie w modelu tylko jednej z nich, ponieważ pozostałe nie wprowadzają zbyt wielu informacji. Innym rozwiązaniem jest zastosowanie technik redukcji wymiaru, np. analizy głównych składowych, z którymi spotkamy się w rozdziale 11. Zbiór danych Fish Market zawiera cechy wysoce skorelowane, a towarzyszący mu Jupyter Notebook się do nich odnosi.

Trzeba jednak pamiętać, że na niektóre modele uczenia maszynowego, np. drzewa decyzyjne i losowe lasy (które omówię wkrótce), skorelowane cechy nie mają wpływu, podczas gdy na inne, np. model regresji liniowej oraz inne modele regresji logistycznej i maszyny wektorów nośnych, mają one negatywny wpływ. W przypadku modeli sieci neuronowych, nawet jeśli mogą one *nauczyć się* korelacji powiązanych z cechami danych podczas szkolenia, to działają lepiej, gdy takie nadmiarowości zostaną uwzględnione z wyprzedzeniem. Nie bez znaczenia są także oszczędności kosztów obliczeniowych i czasu.

Optymalizacja

Optymalizacja oznacza znalezienie optymalnego, najlepszego, maksymalnego, minimalnego lub ekstremalnego rozwiązania.

Zapisaliśmy liniową funkcję szkoleniową:

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$$

i pozostawiliśmy nieznane wartości jej sześciu parametrów ω_0, ω_1, ω_2, ω_3, ω_4 i ω_5. Celem optymalizacji jest znalezienie wartości, dzięki którym funkcja szkoleniowa *będzie najlepiej pasować do podzbioru danych szkoleniowych*, przy czym słowo „najlepiej" jest określone ilościowo za pomocą funkcji straty. Funkcja ta zapewnia miarę określającą, jak daleko odbiegają od prawdy prognozy wykonane przez funkcję szkoleniową modelu. Chcemy, aby ta funkcja straty była jak najmniejsza, więc rozwiązujemy problem minimalizacji.

W tym celu nie zamierzamy jednak wypróbowywać każdej możliwej wartości ω, dopóki nie znajdziemy kombinacji, która zapewni najmniejsze straty. Nawet gdybyśmy tak zrobili, nie wiedzielibyśmy, kiedy przestać, ponieważ nie wiedzielibyśmy, czy istnieją inne *lepsze* wartości. Musimy mieć wcześniejszą wiedzę na temat wykresu funkcji straty i wykorzystać jej właściwości matematyczne. Analogią może być wędrówka po szwajcarskich Alpach z opaską na oczach w porównaniu do wędrówki bez opaski i ze szczegółową mapą (trudny teren szwajcarskich Alp przedstawiono na rysunku 3.7). Zamiast przeszukiwać wykres funkcji straty w poszukiwaniu minimalizatorów z opaską na oczach, wykorzystamy techniki *optymalizacji*.

Optymalizacja to piękna dziedzina matematyki, dostarczająca różnych metod skutecznego wyszukiwania i lokalizowania optymalizatorów funkcji oraz odpowiadających im optymalnych wartości.

Rysunek 3.7. Szwajcarskie Alpy. Optymalizacja przypomina wędrówkę po wykresie funkcji. Miejscem docelowym jest dno najniższej doliny (minimalizacja) lub szczyt najwyższego wzniesienia (maksymalizacja). Potrzebujemy dwóch rzeczy: współrzędnych punktów minimalizacji lub maksymalizacji oraz wysokości wykresu w tych punktach

Problem optymalizacyjny w tym rozdziale i w kilku następnych można sformułować następująco:

$$min_{\vec{\omega}} funkcja\ straty$$

W przypadku obecnego modelu regresji liniowej funkcję optymalizacji można zapisać następująco:

$$min_{\vec{\omega}} \frac{1}{m}\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right)^t\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right) = min_{\vec{\omega}} \frac{1}{m}\left\|X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right\|_{l^2}^2$$

Kiedy zajmujemy się matematyką, nigdy nie powinniśmy tracić z oczu tego, co wiemy i czego szukamy. W przeciwnym razie ryzykujemy wpadnięcie w pułapkę logiki cyklicznej. We wspomnianym wzorze znamy następujące informacje:

m
: Liczba egzemplarzy w podzbiorze szkoleniowym.

X
: Podzbiór szkoleniowy uzupełniony o wektor jedynek.

$\vec{y}_{\text{rzeczywista}}$
: Wektor etykiet odpowiadający podzbiorowi szkoleniowemu.

Szukamy następujących informacji:

- Wartość $\vec{\omega}$, przy której funkcja osiąga minimum.
- Wartość funkcji straty przy wartości $\vec{\omega}$ gwarantującej osiągnięcie minimum.

Wykresy wypukłe kontra wykresy niewypukłe

Najłatwiejszymi funkcjami i równaniami do rozwiązania są funkcje liniowe. Niestety, większość funkcji (i równań), z którymi mamy do czynienia, jest nieliniowa. Jednocześnie nie jest to takie złe, ponieważ liniowe życie jest płaskie, nudne, pozbawione inspiracji i wydarzeń. Gdy funkcja, którą analizujemy, jest w pełni nieliniowa, czasami *linearyzujemy* ją w pobliżu określonych punktów, na których nam zależy. Chodzi o to, że nawet jeśli pełny obraz funkcji jest nieliniowy, możemy aproksymować go funkcją liniową w miejscu, na którym się skupiamy. Mówiąc inaczej, w ograniczonym sąsiedztwie funkcja nieliniowa może wyglądać i zachowywać się liniowo, choć wspomniane sąsiedztwo może być nieskończenie małe. W ramach analogii pomyśl o tym, jak wygląda Ziemia (i jak zachowuje się w kategoriach obliczania odległości itp.). Z lokalnej perspektywy wydaje się płaska, a jej nieliniowy kształt można zauważyć tylko z wysoka. Gdy chcemy zlinearyzować funkcję w pobliżu punktu, przybliżamy ją za pomocą jej przestrzeni stycznej w pobliżu tego punktu (jest to linia styczna, gdy mówimy o funkcji jednej zmiennej, płaszczyzna styczna, gdy mamy do czynienia z funkcją dwóch zmiennych i hiperpłaszczyzna styczna, jeśli jest to funkcja trzech lub więcej zmiennych). W tym celu trzeba obliczyć jedną pochodną funkcji względem wszystkich jej zmiennych, ponieważ w ten sposób możemy wyznaczyć nachylenie (które mierzy inklinację) przybliżonej płaskiej przestrzeni.

Niestety linearyzacja w pobliżu jednego punktu może nie wystarczyć i może wystąpić konieczność zastosowania liniowych przybliżeń w wielu miejscach. Na szczęście jest to wykonalne, ponieważ w tym celu wystarczy jedynie oszacować jedną pochodną w kilku punktach. Prowadzi to nas do *kolejnych najłatwiejszych* funkcji (po funkcjach liniowych): funkcji odcinkowo liniowych, które są liniowe, ale tylko w pewnych fragmentach lub liniowe z wyjątkiem odizolowanych punktów bądź lokalizacji. Takimi funkcjami zajmuje się dziedzina *programowania liniowego*, w której funkcje do optymalizacji są liniowe, a granice dziedzin, w których jest wykonywana optymalizacja, są fragmentarycznie liniowe (są przecięciami półprzestrzeni).

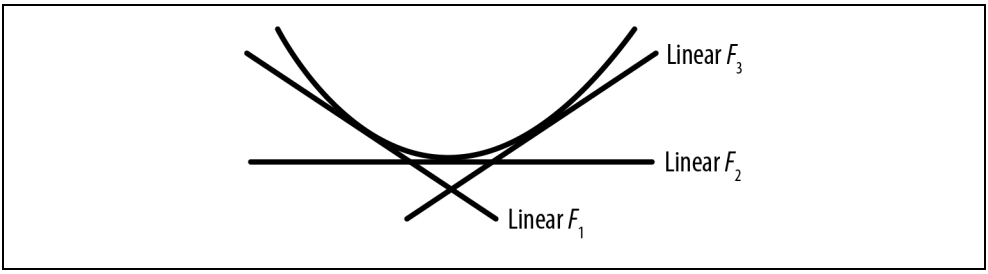
Najlepsze do optymalizacji są funkcje liniowe (wtedy można skorzystać z technik programowania liniowego) lub wypukłe (wtedy nie musimy obawiać się utknięcia w lokalnych minimach i możemy skorzystać z nierówności, które pomagają w procesie analizy).

Jednym z ważnych typów funkcji, o którym należy pamiętać, a który występuje w uczeniu maszynowym, jest funkcja będąca maksimum dwóch lub większej liczby funkcji wypukłych. Takie funkcje zawsze są wypukłe. Przypomnę, że funkcje liniowe są płaskie, więc są jednocześnie wypukłe i wklęsłe. Jest to przydatne, ponieważ niektóre funkcje są zdefiniowane jako maksima funkcji liniowych: nie ma gwarancji, że są liniowe (są fragmentarycznie liniowe), ale jest gwarancja, że są wypukłe. Oznacza to, że nawet gdy podczas wyznaczania maksimum funkcji liniowych tracimy liniowość, to rekompensujemy to wypukłością.

Przykładem funkcji zdefiniowanej jako maksimum dwóch funkcji liniowych jest funkcja ReLU (ang. *Rectified Linear Unit*), wykorzystywana jako nieliniowa funkcja aktywacji w sieciach neuronowych: *ReLU (x) = max (0,x)*. Innym przykładem jest funkcja straty zawiasowej (ang. *hinge loss function*) stosowana w maszynach wektorów nośnych: *H(x) = max(0,1 – tx)*, gdzie *t* wynosi 1 lub –1.

Należy pamiętać, że minimum rodziny funkcji wypukłych nie musi być wypukłe; może mieć formę podwójnej studni. Jednak ich maksimum jest zdecydowanie wypukłe.

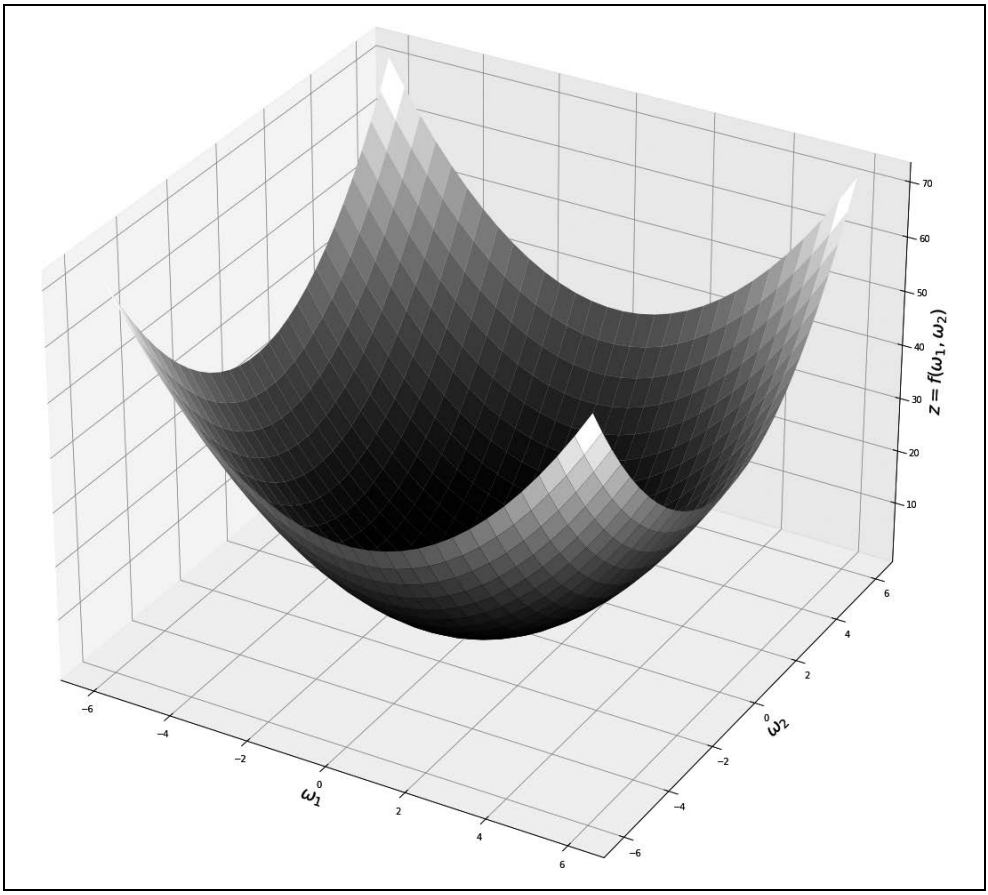
Istnieje jeszcze jeden związek między liniowością a wypukłością. Jeśli mamy funkcję wypukłą (nieliniową, ponieważ liniowa byłaby trywialna), to maksimum wszystkich funkcji liniowych, które pozostają poniżej tej funkcji, jest dokładnie równe wartości tej funkcji. Mówiąc inaczej, wypukłość zastępuje liniowość w tym sensie, że gdy liniowość nie jest dostępna, ale jest dostępna wypukłość, można zastąpić funkcję wypukłą maksimum wszystkich funkcji liniowych, których wykres leży poniżej wykresu funkcji (rysunek 3.8). Przypomnę, że wykres funkcji wypukłej leży powyżej wykresu jej stycznej w dowolnym punkcie, a styczne są liniowe. Daje to bezpośredni sposób wykorzystania prostoty funkcji liniowych, gdy analizujemy funkcje wypukłe. Gdy rozważamy maksimum *wszystkich* stycznych, mamy równanie, natomiast gdy rozważamy maksimum stycznych w kilku punktach, mamy jedynie przybliżenie.



Rysunek 3.8. Funkcja wypukła jest równa maksimum wszystkich swoich stycznych

Na rysunkach 3.9 i 3.10 przedstawiono ogólne wykresy nieliniowych funkcji wypukłych i niewypukłych. Ogólnie rzecz biorąc, wykres funkcji wypukłej dobrze się nadaje do rozwiązywania problemów minimalizacji. Nie trzeba się obawiać utknięcia w lokalnych minimach, ponieważ dla funkcji wypukłej każde lokalne minimum jest również minimum globalnym. Wykres funkcji niewypukłej obejmuje szczyty, doliny i punkty siodłowe. Problem minimalizacji na takim wykresie wiąże się z ryzykiem utknięcia w minimach lokalnych. Wtedy minima globalne mogą nigdy nie zostać znalezione.

Na koniec warto zdać sobie sprawę z różnicy pomiędzy funkcją wypukłą, zbiorem wypukłym i problemem optymalizacji funkcji wypukłej, który optymalizuje funkcję wypukłą w zbiorze wypukłym.

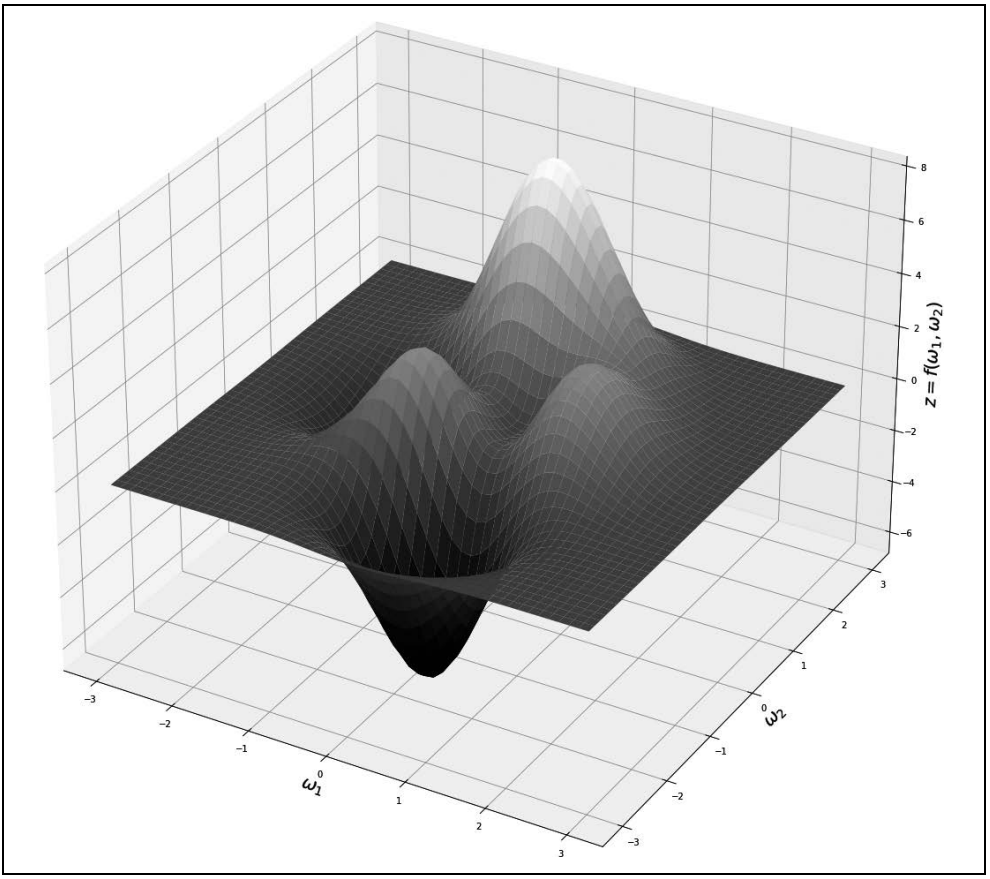


Rysunek 3.9. Wykres funkcji wypukłej jest dobry do rozwiązywania problemów minimalizacji. Nie obawiamy się utknięcia w lokalnych minimach, ponieważ w przypadku funkcji wypukłej każde lokalne minimum jest również minimum globalnym.

Jak zlokalizować minimalizatory funkcji?

Ogólnie rzecz biorąc, istnieją dwa podejścia do lokalizowania minimalizatorów (lub maksymalizatorów) funkcji. Zazwyczaj wiąże się to z wykorzystaniem następujących sposobów:

1. Obliczenie tylko jednej pochodnej i powolna zbieżność do minimum (choć istnieją metody pozwalające przyspieszyć zbieżność). Są to tak zwane metody **gradientowe**. Gradient jest pierwszą pochodną funkcji wielu zmiennych. Na przykład funkcja straty jest funkcją kilku parametrów ω (lub jednego wektora $\vec{\omega}$).
2. Obliczanie dwóch pochodnych (znacznie wyższe koszty obliczeniowe, co jest dużym utrudnieniem, zwłaszcza gdy mamy tysiące parametrów) i szybsza zbieżność do minimum. Koszty obliczeniowe można nieco obniżyć dzięki wykorzystaniu przybliżenia drugiej pochodnej zamiast obliczania jej dokładnie. Metody drugiej pochodnej są nazywane metodami **Newtona**. W tych metodach pojawia się *hesjan* (macierz drugich pochodnych) lub jej przybliżenie.



Rysunek 3.10. Wykres funkcji niewypukłej obejmuje szczyty, doliny i punkty siodłowe. Problem minimalizacji na takim wykresie wiąże się z ryzykiem utknięcia w minimach lokalnych i nieznalezienia minimów globalnych

Nigdy nie trzeba obliczać więcej niż dwóch pochodnych.

Dlaczego pierwsza i druga pochodna funkcji są tak ważne dla zlokalizowania jej optymalizatorów? Zwięzła odpowiedź brzmi: pierwsza pochodna zawiera informacje o tym, jak szybko funkcja rośnie lub maleje w danym punkcie (więc jeśli podążasz za jej kierunkiem, możesz wznieść się do maksimum lub zejść do minimum). Z kolei druga pochodna zawiera informacje o kształcie *krzywizny* funkcji — czy funkcja zakrzywia się w górę, czy w dół.

Podstawowe znaczenie ma w tym przypadku kluczowe pojęcie rachunku różniczkowego: minimalizatory (lub maksymalizatory) występują w *punktach krytycznych* (zdefiniowanych jako punkty, w których jedna z pochodnych funkcji jest równa zero lub nie istnieje) lub w punktach granicznych. Aby zlokalizować te optymalizatory, trzeba przeszukać *zarówno* punkty graniczne (jeśli przestrzeń wyszukiwania ma granice), jak *i* wewnętrzne punkty krytyczne.

Jak zlokalizować punkty krytyczne wewnątrz przestrzeni poszukiwań?

Podejście 1.

Należy wykonać następujące czynności:

- Znajdź pierwszą pochodną funkcji (nie jest źle, wszyscy robiliśmy to w szkole).
- Przyrównaj ją do zera (wszyscy potrafią zapisać symbol równości i zero).
- Rozwiąż równanie względem wartości ω, przy których pochodna przyjmie wartość zero (to jest zły krok!).

W przypadku funkcji, których pochodne są liniowe, takich jak funkcja straty błędu średniokwadratowego, znalezienie rozwiązania dla tych wartości ω jest łatwe. Dziedzina algebry liniowej została specjalnie stworzona w celu rozwiązywania liniowych układów równań. Dziedzinę numerycznej algebry liniowej stworzono, aby pomóc w rozwiązywaniu realistycznych i dużych układów równań liniowych, w których dominują złe warunki. Do rozwiązywania liniowych układów równań mamy wiele narzędzi (i pakietów oprogramowania).

Z drugiej strony znajdowanie rozwiązań dla równań nieliniowych to zupełnie inna historia. Staje się ono grą typu chybił trafił z przewagą chybień! Oto krótki przykład ilustrujący różnicę między rozwiązywaniem równań liniowych i nieliniowych:

Rozwiązywanie równań liniowych

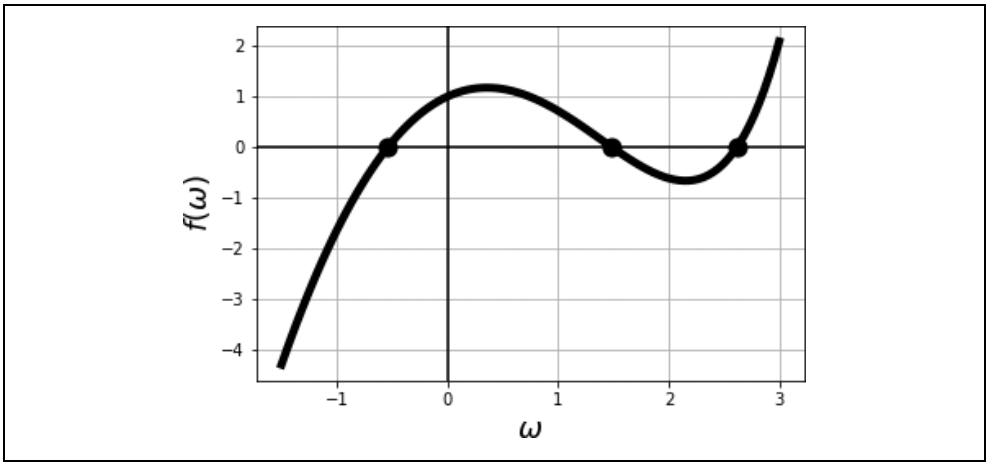
Znajdź takie ω, że $0{,}002\omega - 5 = 0$.

Rozwiązanie: Po przeniesieniu 5 na drugą stronę równania, a następnie podzieleniu przez 0,002 otrzymujemy $\omega = 5/0{,}002 = 2500$. Gotowe.

Rozwiązywanie równań nieliniowych

Znajdź ω takie, że $0{,}002 \sin(\omega) - 5\omega^2 + e^{\omega} = 0$.

Rozwiązanie: Nie ma prostej metody. Potrzebna jest metoda numeryczna! (Graficzne przybliżenie rozwiązania tego równania nieliniowego można znaleźć na rysunku 3.11).



Rysunek 3.11. Rozwiązywanie równań nieliniowych jest trudne. W tym przypadku wykreślamy funkcję $f(\omega) = 0{,}002 \sin(\omega) - 5\omega^2 + e^{\omega}$ i wyznaczamy przybliżenie jej trzech pierwiastków (punktów, w których na wykresie $f(\omega) = 0$)

Jest wiele metod numerycznych poświęconych wyłącznie znajdowaniu rozwiązań równań nieliniowych (oraz całe dziedziny poświęcone numerycznemu rozwiązywaniu nieliniowych równań różniczkowych zwyczajnych i cząstkowych). Metody te pozwalają znaleźć przybliżone rozwiązania, a następnie ustalić granice określające, jak daleko rozwiązania numeryczne odbiegają od dokładnych rozwiązań analitycznych. Zazwyczaj polegają one na skonstruowaniu sekwencji, która pod pewnymi warunkami zbiega się do rozwiązania analitycznego. Niektóre metody zapewniają szybszą zbieżność niż inne i są lepiej dostosowane do określonych problemów niż inne.

Podejście 2.

> Inną opcją jest podążanie za kierunkiem nachylenia, które opada w stronę minimum lub wznosi się w stronę maksimum.

Aby zrozumieć te metody gradientowe, pomyśl o wędrówce w dół góry (lub zjeżdżaniu z góry na nartach dla metod z akceleracją). Zaczynamy od losowego punktu w przestrzeni poszukiwań, co ustawia nas na początkowym poziomie wysokości na wykresie funkcji. Następnie przechodzimy do nowego punktu w przestrzeni wyszukiwania w nadziei, że w tej nowej lokalizacji znajdziemy się na nowym poziomie wysokości, który będzie *niższy* niż poziom wysokości, z którego wyszliśmy. W takim przypadku będzie to oznaczać, że *schodzimy*. Czynność tę powtarzamy. W idealnym przypadku, jeśli wykres funkcji odpowiednio współpracuje, sekwencja punktów zbiega się w kierunku minimalizatora funkcji, którego szukamy.

Oczywiście, w przypadku funkcji z wykresami obejmującymi wiele szczytów i dolin miejsce rozpoczęcia — lub mówiąc inaczej, sposób inicjalizacji — ma znaczenie, ponieważ można zejść w dół zupełnie innej doliny niż ta, do której dążymy. Możemy znaleźć minimum lokalne zamiast globalnego.

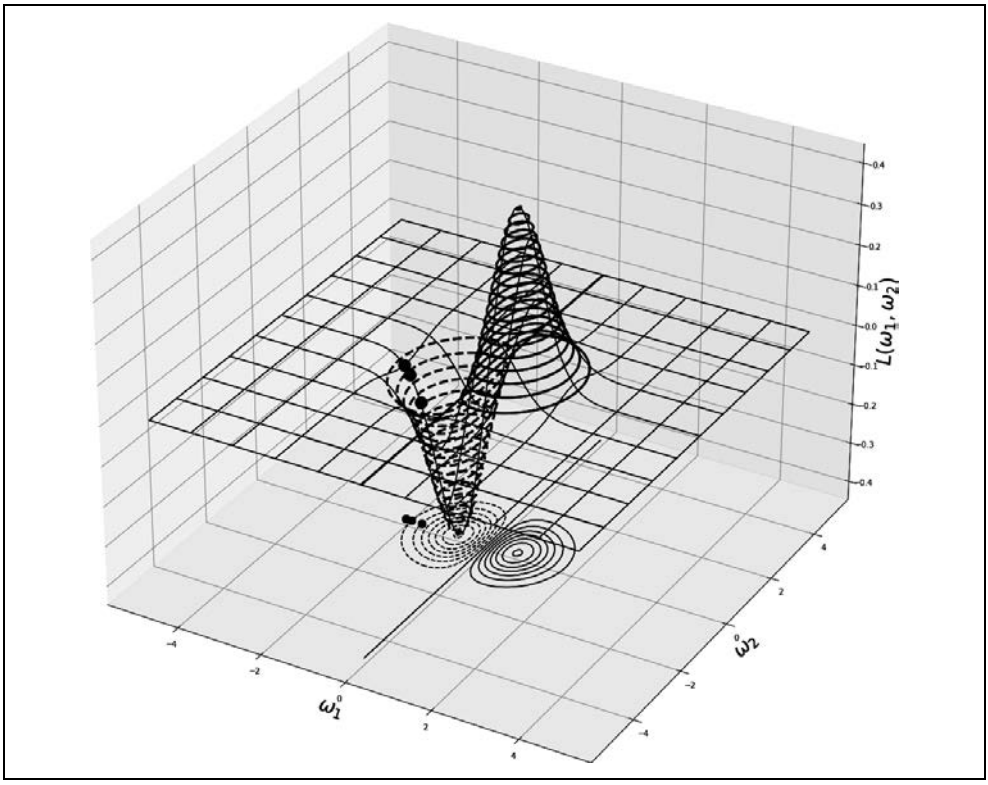
Funkcje, które są wypukłe i ograniczone poniżej, mają kształt miski do sałatki, zatem w ich przypadku nie trzeba się martwić o utknięcie w minimach lokalnych i oddalenie od minimów globalnych. W przypadku funkcji wypukłych możemy mieć inne źródło zmartwień: gdy kształt czaszy funkcji jest zbyt wąski, zbieżność metody może być bardzo wolna. Problem ten omówię szczegółowo w rozdziale 4.

Przydatne i popularne są zarówno podejście 1., jak i podejście 2. Czasami nie ma innego wyjścia, jak użycie jednej lub drugiej metody, w zależności od tego, jak szybko każda z nich zbiega się dla konkretnego ustawienia, jak *regularna* jest funkcja, którą próbujemy zoptymalizować (ile ma dobrze zachowanych pochodnych), itp. W innych przypadkach jest to po prostu kwestia indywidualnych preferencji. W przypadku funkcji straty regresji liniowej — błędu średniokwadratowego — sprawdzają się oba typy metod. Spróbujemy użyć podejścia 1. tylko dlatego, że będziemy używać metod zstępowania gradientowego dla *wszystkich* innych funkcji straty wykorzystanych w tej książce.

Trzeba wspomnieć, że analogia do wędrówki w dół góry dla metod poszukiwania minimum jest doskonała, ale nieco myląca. Kiedy my, ludzie, schodzimy z góry, fizycznie znajdujemy się w tej samej trójwymiarowej przestrzeni, w której istnieje górski krajobraz, co oznacza, że znajdujemy się na pewnej wysokości i możemy zejść do miejsca znajdującego się na niższej wysokości nawet z opaską na oczach, a nawet wtedy, gdy jest mglisto i możemy zejść tylko o jeden mały krok na raz. Wyczuwamy wzniesienie, a następnie poruszamy się w dół. Z drugiej strony,

numeryczne metody zstępowania nie szukają minimum w tym samym wymiarze przestrzeni, w którym osadzony jest krajobraz funkcji. Zamiast tego szukają na poziomie podstawy, o jeden wymiar *poniżej* wykresu (rysunek 3.12). Z tego powodu schodzenie w kierunku minimum jest znacznie trudniejsze. Na poziomie podstawy można przejść z dowolnego punktu do dowolnego innego bez wiedzy o tym, jaki poziom wysokości znajduje się nad nami. Dowiemy się o tym dopiero wtedy, gdy ocenimy wartość funkcji w punkcie i znajdziemy tę wysokość. Zatem wykorzystana metoda numeryczna może przypadkowo przenieść nas z jednego punktu o pewnej wysokości nad podstawą do innego punktu o *większej* wysokości, a więc dalej od minimum. Dlatego ważne jest, aby na poziomie podstawy zlokalizować kierunek, który powoduje szybkie *zmniejszanie* wysokości funkcji, i ustalić, jak daleko można się poruszać w tym kierunku na poziomie podstawy (jaka jest wielkość kroku), która *zapewni zmniejszanie wysokości funkcji nad podstawą*. Rozmiar kroku jest nazywany również *hiperparametrem szybkości uczenia się*. Będziemy się z nim spotykać każdorazowo podczas wykorzystywania metody zstępującej.

Wróćmy do głównego celu: chcemy znaleźć najlepszą wartość $\vec{\omega}$ dla funkcji szkoleniowej. Chcemy zminimalizować funkcję straty błędu średniokwadratowego przy użyciu podejścia 1., czyli oblicz pierwszą pochodną funkcji straty i przyrównaj ją do zera, a następnie rozwiąż dla wektora $\vec{\omega}$. W tym celu trzeba opanować rachunek wyrażeń algebry liniowej. Wróćmy do pierwszych lekcji z analizy matematycznej.



Rysunek 3.12. Wyszukiwanie minimum odbywa się na poziomie podstawy, a nie bezpośrednio na wykresie funkcji

Analiza matematyczna w pigułce

Na podstawowym kursie analizy matematycznej poznajemy funkcje jednej zmiennej ($f(\omega)$), ich wykresy oraz o obliczaniu wartości funkcji w określonych punktach. Następnie poznajemy najważniejszą operację w analizie matematycznej: wyznaczanie granic. Na podstawie pojęcia granicy definiujemy ciągłość i nieciągłość funkcji, pochodną w punkcie $f'(\omega)$ (granica nachylenia siecznych przechodzących przez punkt) oraz całkę w dziedzinie (granica sum miniobszarów wyznaczonych przez funkcję w dziedzinie). Kurs kończy się na prezentacji podstawowego twierdzenia rachunku różniczkowego, odnoszącego się do całkowania i różniczkowania jako operacji odwrotnych. Jedną z kluczowych właściwości pochodnej jest to, że określa ona, jak szybko funkcja rośnie lub maleje w określonym punkcie. Z tego powodu pochodna odgrywa kluczową rolę w lokalizowaniu minimum i (lub) maksimum funkcji wewnątrz jej dziedziny (punkty graniczne są traktowane oddzielnie).

Na kursie rachunku różniczkowego wielu zmiennych, który jest zwykle trzecim kursem rachunku różniczkowego, wiele pojęć przenosi się z rachunku pojedynczej zmiennej. Dotyczy to także pochodnej, obecnie nazywanej gradientem, ponieważ dotyczy kilku zmiennych i jej roli w lokalizowaniu wewnętrznych minimów i (lub) maksimów. Gradient $\nabla(f(\vec{\omega}))$ z $f(\vec{\omega})$ jest pochodną funkcji względem wektora $\vec{\omega}$ zmiennych.

W uczeniu głębokim nieznane wagi są zorganizowane w macierze, a nie w wektory, więc trzeba obliczyć pochodną funkcji $f(W)$ względem macierzy zmiennych W.

Dla celów w sztucznej inteligencji funkcją, której pochodną trzeba obliczyć, jest funkcja straty, która ma wbudowaną funkcję szkoleniową. Zgodnie z *regułą łańcuchową dla pochodnych* trzeba również obliczyć *pochodną* funkcji szkoleniowej względem ω.

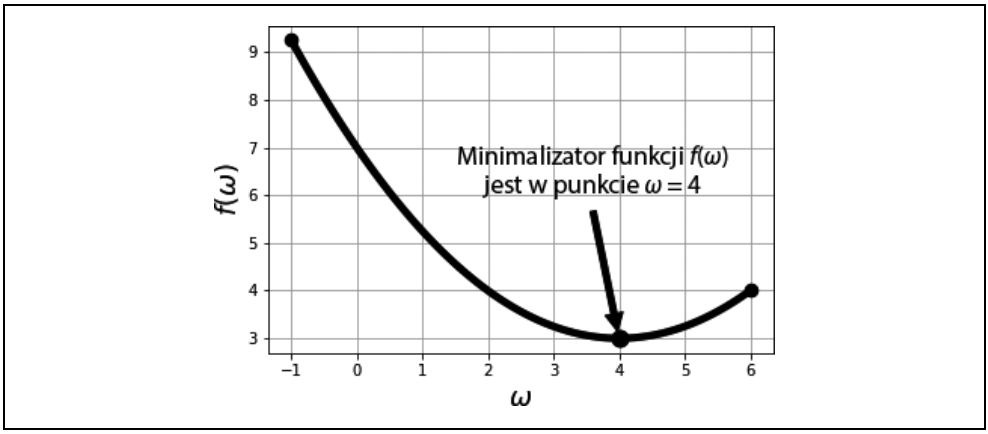
Spróbujmy to zademonstrować na prostym przykładzie z rachunku jednej zmiennej, a następnie przejdziemy do obliczania pochodnych wyrażeń algebry liniowej.

Przykład optymalizacji jednowymiarowej

Problem: znajdź minimalizatora(y) i wartość minimalną (jeśli istnieje) funkcji $f(\omega) = 3 + (0{,}5\omega - 2)^2$ w przedziale [–1, 6].

Jednym z bardzo czasochłonnych sposobów jest wypróbowanie nieskończenie wielu wartości ω pomiędzy –1 a 6 i wybranie takiej wartości parametru ω, dla której funkcja f przyjmuje najmniejszą wartość. Innym sposobem jest wykorzystanie wiedzy z algebry, zgodnie z którą optymalizatory (minimalizatory lub maksymalizatory) występują albo w punktach krytycznych (gdzie pochodna nie istnieje bądź wynosi zero), albo w punktach granicznych. Zilustrowano to na rysunku 3.13.

Punkty graniczne w tym przykładzie to –1 i 6, więc najpierw oszacujemy funkcję w tych punktach: $f(-1) = 3 + (0{,}5(-1) - 2)^2 = 9{,}25$ i $f(6) = 3 + (0{,}5(6) - 2)^2 = 4$. Oczywiście –1 nie jest minimalizatorem, ponieważ $f(6) < f(-1)$, więc ten punkt graniczny odpada z rywalizacji i teraz z wewnętrznymi punktami krytycznymi konkuruje tylko punkt graniczny 6. Aby znaleźć punkty krytyczne, sprawdzamy pochodną funkcji wewnątrz przedziału $[-1,6]$: $f'(\omega) = 0 + 2(0{,}5\omega - 2) \cdot 0{,}5 = 0{,}25(0{,}5\omega - 2)$. Gdy przyrównamy tę pochodną do zera, otrzymamy $0{,}25\,(0{,}5\omega - 2) = 0$,



Rysunek 3.13. Minimalna wartość funkcji $f(\omega) = 3 + (0{,}5\omega - 2)^2$ w przedziale [–1, 6] wynosi 3 i występuje w punkcie krytycznym $\omega = 4$. W tym punkcie krytycznym pochodna funkcji wynosi zero, co oznacza, że jeśli narysujemy linię styczną, będzie ona pozioma

co oznacza, że $\omega = 4$. W ten sposób wewnątrz przedziału *[–1, 6]* znaleźliśmy tylko jeden punkt krytyczny $\omega = 4$. W tym szczególnym punkcie wartość funkcji wynosi $f(4) = 3 + (0{,}5(4) - 2)^2 = 3$. Ponieważ wartość f jest tutaj najmniejsza, to oczywiście znaleźliśmy zwycięzcę konkursu minimalizacji, a mianowicie $\omega = 4$ z minimalną wartością f równą 3.

Pochodne często używanych wyrażeń algebry liniowej

Najskuteczniejsze jest obliczanie pochodnych bezpośrednio na wyrażeniach obejmujących wektory i macierze, bez konieczności rozkładania ich na czynniki. Popularne są następujące dwa podejścia:

1. Gdy a i ω są skalarami, a a jest stałe, pochodną funkcji $f(\omega) = a\omega$ jest $f'(\omega) = a\omega$. Gdy $\vec{a}$ i $\vec{\omega}$ są wektorami (o tej samej długości), a elementy wektora $\vec{a}$ są stałe, to gradient $f(\vec{\omega}) = \vec{a}^{t \to \vec{\omega}}$ wynosi $\nabla f(\vec{\omega}) = \vec{a}$. Podobnie gradient $f(\vec{\omega}) = \vec{a}^t\vec{\omega}$ wynosi $\nabla f(\vec{\omega}) = \vec{a}$
2. Gdy s jest skalarne i stałe, a ω jest skalarne, to pochodną funkcji kwadratowej $f(\omega) = s\omega^2$ jest $f'(\omega) = 2s\omega$. Analogiczny przypadek dla funkcji wielowymiarowej to sytuacja, gdy S jest macierzą symetryczną o stałych elementach, funkcja $f(\vec{\omega}) = \vec{\omega}^t S\vec{\omega}$ jest kwadratowa, a jej gradient wynosi $\nabla f(\vec{\omega}) = 2S\vec{\omega}$.

Minimalizacja funkcji straty błędu średniokwadratowego

W końcu możemy zminimalizować funkcję straty błędu średniokwadratowego:

$$L(\vec{\omega}) = \frac{1}{m}\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right)^t\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right)$$

Przed wyznaczeniem gradientu wyrażenia i przyrównaniem go do zera spróbujmy je przekształcić:

$$
\begin{aligned}
L(\vec{\omega}) &= \frac{1}{m}\left((X\vec{\omega})^t - \vec{y}^t_{\text{rzeczywista}}\right)\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right) \\
&= \frac{1}{m}\left(\vec{\omega}^t X^t - \vec{y}^t_{\text{rzeczywista}}\right)\left(X\vec{\omega} - \vec{y}_{\text{rzeczywista}}\right) \\
&= \frac{1}{m}\left(\vec{\omega}^t X^t X\vec{\omega} - \vec{\omega}^t X^t \vec{y}_{\text{rzeczywista}} - \vec{y}^t_{\text{rzeczywista}} X\vec{\omega} + \vec{y}^t_{\text{rzeczywista}}\vec{y}_{\text{rzeczywista}}\right) \\
&= \frac{1}{m}\left(\vec{\omega}^t S\vec{\omega} - \vec{\omega}^t\vec{a} - \vec{a}^t\vec{\omega} + \vec{y}^t_{\text{rzeczywista}}\vec{y}_{\text{rzeczywista}}\right),
\end{aligned}
$$

gdzie w ostatnim kroku podstawiamy $X^tX = \text{i } X^t\vec{y}_{\text{rzeczywista}} = \vec{a}$. Następnie obliczamy gradient ostatniego wyrażenia względem $\vec{\omega}$ i przyrównujemy je do zera. Podczas obliczania gradientu wykorzystujemy to, czego właśnie nauczyliśmy się o różniczkowaniu wyrażeń algebry liniowej:

$$
\begin{aligned}
\nabla L(\vec{\omega}) &= \frac{1}{m}(2S\vec{\omega} - \vec{a} - \vec{a} + 0) \\
&= \vec{0}
\end{aligned}
$$

Teraz znalezienie rozwiązania dla $\vec{\omega}$ jest łatwe:

$$
\frac{1}{m}(2S\vec{\omega} - 2\vec{a}) = \vec{0}
$$

zatem:

$$
2\vec{S} = 2\vec{a}
$$

otrzymujemy więc:

$$
\vec{\omega} = S^{-1}\vec{a}
$$

Przypomnę, że użyliśmy podstawień $S = X^tX$ i $\vec{a} = X^t y_{\text{rzeczywista}}$. Spróbujmy zatem przepisać funkcję minimalizacji $\vec{\omega}$ w kategoriach zbioru szkoleniowego X (rozszerzonego o jedynki) i odpowiadającego mu wektora etykiet $\vec{y}_{\text{rzeczywista}}$.

$$
\vec{\omega} = (X^tX)^{-1}X^t\vec{y}_{\text{rzeczywista}}
$$

W przypadku zestawu danych Fish Market (patrz załączony notatnik Jupyter) otrzymujemy:

$$
\vec{\omega} = \begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \omega_5 \end{pmatrix} = \begin{pmatrix} -475.19929130109716 \\ 82.84970118 \\ -28.85952426 \\ -28.50769512 \\ 29.82981435 \\ 30.97250278 \end{pmatrix}
$$

Właśnie zlokalizowaliśmy wektor wag $\vec{\omega}$, który daje najlepsze dopasowanie między danymi szkoleniowymi a funkcją szkoleniową regresji liniowej:

$$
f(\vec{\omega}; \vec{x}) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5
$$

Aby uzyskać rozwiązanie określone przez równanie normalne, zastosowaliśmy metodę analityczną (obliczenie gradientu funkcji straty i przyrównanie go do zera). Jest to jeden z bardzo rzadkich przypadków, w których jesteśmy w stanie wyprowadzić rozwiązanie analityczne. Wszystkie inne metody znajdowania minimalizacji $\vec{\omega}$ będą numeryczne.

Mnożenie dużych macierzy przez siebie jest bardzo kosztowne obliczeniowo. Zamiast tego mnóż macierze przez wektory

Za wszelką cenę staraj się unikać mnożenia macierzy przez siebie; zamiast tego mnóż macierze przez wektory. Na przykład w równaniu $\vec{\omega} = (X^t X)^{-1} X^t \vec{y}_{\text{rzeczywista}}$, najpierw oblicz $X^t \vec{y}_{\text{rzeczywista}}$ i unikaj obliczania $(X^t X)^{-1}$. Sposobem na obejście tego problemu jest rozwiązanie równania liniowego $X\vec{\omega} = \vec{y}_{\text{rzeczywista}}$ z wykorzystaniem pseudoinwersji X (zobacz dołączony notatnik Jupyter). Pseudoinwersję omówię w rozdziale 11. Na razie zapamiętaj, że jest to działanie, które pozwala odwracać macierze (co jest równoważne dzieleniu przez nie), które nie mają odwrotności.

Nie należy zbytnio dopasowywać danych szkoleniowych

Równanie $\vec{\omega} = (X^t X)^{-1} X^t \vec{y}_{\text{rzeczywista}}$ pozwala wyznaczyć wartości ω, przy których funkcja szkoleniowa *najlepiej pasuje* do danych szkoleniowych, ale w przypadku zbyt ścisłego dopasowania funkcja szkoleniowa może zamiast sygnału w danych przechwytywać również szum. Zatem aby *nie uzyskać zbyt dobrego dopasowania*, wspomniane rozwiązanie, a nawet sam problem minimalizacji, trzeba zmodyfikować. W tym przypadku *pomocna jest regularyzacja* lub **wczesne zatrzymanie** (ang. *early stopping*). Tym zagadnieniom poświęcimy trochę czasu w rozdziale 4.

To była długa droga do regresji. Po drodze musieliśmy przejść przez rachunek różniczkowy i algebrę liniową, bo to są podstawy. Prezentowanie trudniejszych modeli uczenia maszynowego — regresji logistycznej, maszyn wektorów nośnych, drzew decyzyjnych i lasów losowych — będzie szybsze, ponieważ zadania te sprowadzają się do stosowania dokładnie tych samych pomysłów do różnych funkcji.

Regresja logistyczna — klasyfikacja do dwóch klas

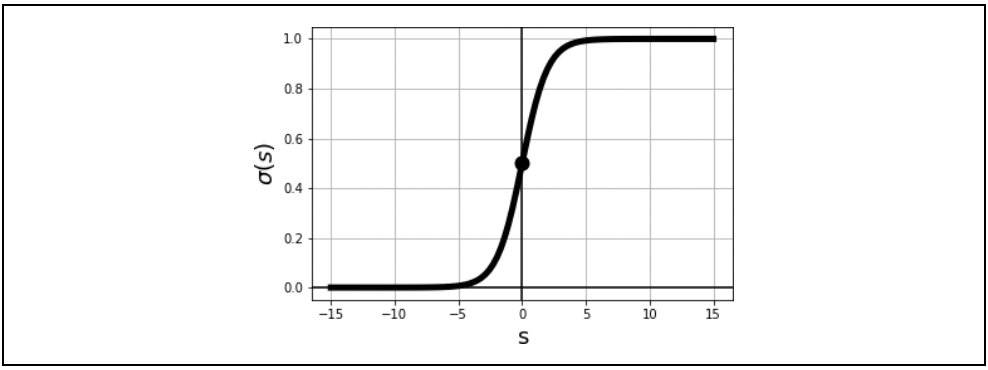
Regresja logistyczna jest używana głównie do zadań klasyfikacyjnych. Najpierw wyjaśnię, w jaki sposób można wykorzystać ten model do zadań klasyfikacji binarnej (klasyfikowania do dwóch klas, np. rak – nie rak, bezpieczny dla dzieci – niebezpieczny, prawdopodobnie spłaci pożyczkę – prawdopodobnie jej nie spłaci itp.) Następnie uogólnię ten model do klasyfikowania na wiele klas (na przykład przyporządkowywanie odręcznych rysunków cyfr do wartości 0, 1, 2, 3, 4, 5, 6, 7, 8 lub 9). W tym przypadku również mamy tę samą konfigurację matematyczną:

1. Funkcja szkoleniowa.
2. Funkcja straty.
3. Optymalizacja.

Funkcja szkoleniowa

Podobnie jak w przypadku regresji liniowej, funkcja szkoleniowa regresji logistycznej oblicza kombinację liniową cech i dodaje stały wyraz odchylenia. Jednak zamiast wyprowadzać wynik w takiej postaci, przekazuje go do *funkcji logistycznej*, której wykres przedstawiono na rysunku 3.14 i która ma następujący wzór:

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$



Rysunek 3.14. Wykres funkcji logistycznej $\sigma(s) = \frac{1}{1+e^{-s}}$. Należy zauważyć, że funkcję tę można obliczyć dla dowolnego s i zawsze uzyskamy wynik z przedziału od 0 do 1. W związku z tym wynik funkcji można interpretować jako prawdopodobieństwo

Jest to funkcja, która przyjmuje tylko wartości od 0 do 1, więc jej wynik można interpretować jako prawdopodobieństwo przynależności punktu danych do określonej klasy. Jeśli wynik jest mniejszy niż 0,5, to klasyfikujemy przynależność punktu danych do pierwszej klasy, a jeśli wynik przekracza 0,5, to klasyfikujemy jego przynależność do drugiej klasy. Liczba 0,5 to *próg*, przy którym podejmujemy decyzję o sklasyfikowaniu punktu danych.

Tak więc funkcja szkoleniowa jest liniową kombinacją cech plus odchylenie, złożoną z funkcji logistycznej w połączeniu z funkcją progową:

$$y = Próg\big(\sigma(\omega_0 + \omega_1 x_1 + \cdots + \omega_n x_n)\big)$$

Podobnie jak w przypadku regresji liniowej, parametry ω są niewiadomymi, dla których trzeba zoptymalizować funkcję straty. Tak samo jak jest w przypadku regresji liniowej, liczba tych niewiadomych jest równa liczbie cech danych plus jedna dla wyrazu określającego systematyczny błąd. W przypadku takich zadań jak klasyfikacja obrazów każdy piksel jest cechą, więc może ich być tysiące.

Funkcja straty

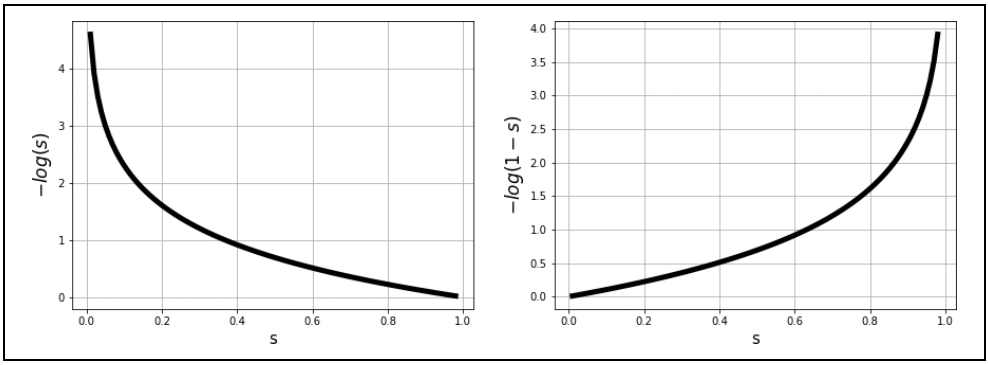
Spróbujmy zaprojektować dobrą **funkcję straty** dla klasyfikacji. Jesteśmy inżynierami i chcemy „karać" błędnie sklasyfikowane punkty danych szkoleniowych. Jeśli punkt danych w oznakowanym zestawie danych należy do klasy, to jego $y_{rzeczywista} = 1$, a jeśli nie, to jego $y_{rzeczywista} = 0$.

Chcemy, aby funkcja szkoleniowa zwracała wynik $y_{prognoza} = 1$ dla punktów szkoleniowych należących do klasy dodatniej (dla których $y_{rzeczywista}$ również wynosi 1). Skuteczne wartości ω powinny gwarantować przekazywanie do funkcji logistycznej wysokiej wartości t (wynik kroku kombinacji liniowej), a tym samym przypisanie wysokiego prawdopodobieństwa dla dodatnich punktów danych i przekazanie progu 0,5, aby uzyskać $y_{prognoza} = 1$. W związku z tym, jeśli suma kombinacji liniowej i stronniczości daje niską wartość t, podczas gdy $y_{rzeczywista} = 1$, należy ukarać model.

Na podobnej zasadzie skuteczne wartości wag powinny gwarantować przekazywanie niskich wartości t dla punktów szkoleniowych, które nie należą do klasy (ich $y_{rzeczywista} = 0$). Dlatego jeśli suma kombinacji liniowej i stronniczości daje wysoką wartość t, podczas gdy $y_{rzeczywista} = 0$, należy ukarać model.

Jak więc znaleźć funkcję straty, która karze błędnie sklasyfikowany punkt danych szkoleniowych? Karane powinny być zarówno wyniki fałszywie dodatnie, jak i fałszywie ujemne. Przypomnę, że wynik tego modelu klasyfikacji to 1 lub 0:

- Pomyśl o funkcji algebraicznej $-log(s)$, która nagradza jedynki i karze zera (rysunek 3.15).
- Pomyśl o funkcji algebraicznej $-log(1-s)$, która karze jedynki i nagradza zera (rysunek 3.15).



Rysunek 3.15. Po lewej: wykres funkcji f(s) = – log(s). Ta funkcja przypisuje wysokie wartości liczbom bliskim 0 i niskie wartości liczbom bliskim 1. Po prawej: wykres funkcji f(s) = – log(1 – s). Ta funkcja przypisuje wysokie wartości liczbom bliskim 1 i niskie wartości liczbom bliskim 0

Skupmy się teraz na wyniku funkcji logistycznej $\sigma(s)$ dla bieżącego zestawu parametrów ω:

- Jeśli $\sigma(s)$ jest mniejsze niż 0,5 (prognoza modelu to $y_{prognoza} = 0$), ale rzeczywiste $y_{rzeczywista} = 1$ (wynik fałszywie ujemny), trzeba nałożyć na model karę w wysokości – log ($\sigma(s)$). Jeśli z kolei $\sigma(s) > 0{,}5$, prognoza modelu wynosi $y_{prognoza} = 1$) (wynik prawdziwie dodatni), to wartość – log($\sigma(s)$) jest mała, więc model nie będzie ukarany.
- Na podobnej zasadzie, jeśli $\sigma(s)$ jest większe niż 0,5, ale rzeczywista wartość $y_{rzeczywista} = 0$ (wynik fałszywie dodatni), na model zostanie nałożona kara w wysokości – log(1 – $\sigma(s)$). Tak jak w przypadku wyniku prawdziwie dodatniego, również za wynik prawdziwie ujemny model nie ponosi wysokiej kary.

W związku z tym koszt błędnej klasyfikacji jednego szkoleniowego punktu danych $\left(x_1^i, x_2^i, \cdots, x_n^i; y_{\text{rzeczywista}}\right)$ można zapisać następująco:

$$koszty = \begin{cases} -\log\left(\sigma(s)\right) \text{ jeśli } y_{\text{rzeczywista}} = 1 \\ -\log\left(1-\sigma(s)\right) \text{ jeśli } y_{\text{rzeczywista}} = 0 \end{cases} = -y_{\text{rzeczywista}} \log\left(\sigma(s)\right) - \left(1 - y_{\text{rzeczywista}}\right) \log\left(1-\sigma(s)\right)$$

Na koniec funkcja straty oznacza średni koszt dla *m* szkoleniowych punktów danych. W ten sposób możemy uzyskać wzór na popularną *funkcję straty w entropii krzyżowej*:

$$L(\vec{\omega}) = -\frac{1}{m}\sum_{i=1}^{m} y_{\text{rzeczywista}}^i \log\left(\sigma\left(\omega_0 + \omega_1 x_1^i + \cdots + \omega_n x_n^i\right)\right) + \left(1 - y_{\text{rzeczywista}}^i\right)\log\left(1 - \sigma\left(\omega_0 + \omega_1 x_1^i + \cdots + \omega_n x_n^i\right)\right)$$

Optymalizacja

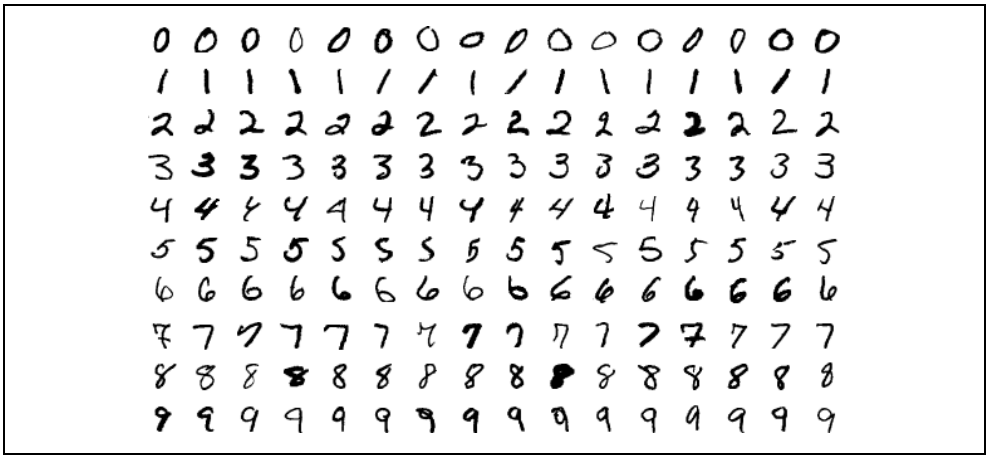
Jeśli zdecydujemy się zminimalizować funkcję straty poprzez ustawienie $\nabla L(\omega) = 0$, to w przeciwieństwie do przypadku regresji liniowej nie będziemy mieć analitycznego wzoru rozwiązania dla parametrów ω. Dobrą wiadomością jest to, że funkcja ta jest wypukła, więc zastosowanie metody zstępowania gradientowego omówionej w rozdziale 4. (lub zstępowania gradientowego stochastycznego bądź z wykorzystaniem minipartii) gwarantuje znalezienie minimum (o ile *szybkość uczenia* nie jest zbyt duża i o ile czekamy wystarczająco długo).

Regresja softmax — przyporządkowanie do wielu klas

Pojęcie regresji logistycznej można łatwo uogólnić do klasyfikacji na wiele klas. Znanym przykładem takiego zadania klasyfikacji niebinarnej jest przyporządkowanie obrazów 10 odręcznie zapisanych cyfr 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9 za pomocą zestawu danych MNIST (*https://yann.lecun.com/exdb/mnist/*)[1]. Ten zbiór danych zawiera 70 000 obrazów odręcznie zapisanych cyfr (próbki tych obrazów zaprezentowano na rysunku 3.16), podzielonych na podzbiór szkoleniowy złożony z 60 000 obrazów i podzbiór testowy złożony z 10 000 obrazów. Każdy obraz oznaczono klasą, do której ten obraz należy, czyli jedną z 10 cyfr.

Wspomniany zbiór danych zawiera również wyniki wielu modeli klasyfikacyjnych, w tym klasyfikatorów liniowych, *k-najbliższych sąsiadów*, *drzew decyzyjnych*, *maszyn wektorów nośnych* z różnymi *jądrami* i sieci neuronowych o różnych architekturach, wraz z odwołaniami do odpowiednich artykułów i latami ich publikacji. Warto zwrócić uwagę na postępy w wydajności na przestrzeni lat i ewolucję stosowanych metod.

[1] W chwili pisania tego tekstu ta strona wymagała poświadczeń, ale można uzyskać do niej dostęp za pomocą instrukcji zamieszczonych w serwisie Hacker News (*https://news.ycombinator.com/item?id=31077780*).



Rysunek 3.16. Przykładowe obrazy ze zbioru danych MNIST (źródło obrazu: https://en.wikipedia.org/wiki/MNIST_database*)*

Nie należy mylić klasyfikacji do wielu klas z modelami wielowyjściowymi

Regresja softmax prognozuje jedną klasę na raz, więc nie można jej wykorzystać do sklasyfikowania na przykład pięciu osób na tym samym zdjęciu. Zamiast tego można jej użyć do sprawdzenia, czy określone zdjęcie na Facebooku jest zdjęciem moim, mojej siostry, mojego brata, mojego męża lub mojej córki. Zdjęcie przekazane do modelu regresji softmax może przedstawiać tylko jedno z nas pięciorga. W przeciwnym razie klasyfikacja modelu byłaby mniej oczywista. Oznacza to, że klasy w tego rodzaju zadaniu muszą się wzajemnie wykluczać. Jeśli zatem na zdjęciu na Facebooku zostanie automatycznie oznaczonych pięć osób, oznacza to, że użyto modelu wielowyjściowego, a nie modelu regresji softmax.

Załóżmy, że znamy cechy punktu danych i chcemy wykorzystać te informacje w celu przyporządkowania punktu danych do jednej z k możliwych klas. Zaprezentowane poniżej funkcja szkoleniowa, funkcja strat i proces optymalizacji nie powinny teraz budzić wątpliwości.

Cechy danych zdjęć

W przypadku zdjęć w skali szarości każdy piksel intensywności jest cechą, więc zdjęcia zwykle mają tysiące cech. Obrazy w skali szarości są zwykle reprezentowane jako dwuwymiarowe macierze liczb, gdzie poszczególne elementy macierzy reprezentują intensywność pikseli. Zdjęcia kolorowe są reprezentowane za pomocą trzech kanałów: czerwonego, zielonego i niebieskiego, przy czym każdy kanał jest reprezentowany jako dwuwymiarowa macierz liczb, a kanały są ułożone jeden na drugim, co tworzy trzy warstwy dwuwymiarowych macierzy. Taka struktura jest nazywana **tensorem**. Sposoby przetwarzania zdjęć w skali szarości i kolorowych w Pythonie zaprezentowano w notatniku poświęconym tej tematyce na stronie GitHub repozytorium powiązanego z tą książką (*https://github.com/halanelson/Essential-Math-For-AI*).

Funkcja szkoleniowa

Pierwszy krok jest zawsze taki sam: liniowe wyrażenie z wykorzystaniem cech zsumowane ze stałym wyrazem reprezentującym błąd systematyczny. W regresji logistycznej, gdy mieliśmy tylko dwie klasy, przekazywaliśmy wynik do funkcji logistycznej o następującym wzorze:

$$\sigma(s) = \frac{1}{1+e^{-s}} = \frac{1}{1+\frac{1}{e^{s}}} = \frac{e^{s}}{1+e^{s}} = \frac{e^{s}}{e^{0}+e^{s}}$$

Wyniki tej funkcji interpretowaliśmy jako prawdopodobieństwo przynależności punktu danych do interesującej nas klasy. Zwróć uwagę, że w celu podkreślenia, iż uwzględnia ona dwa prawdopodobieństwa, po jednym dla każdej klasy, przepisaliśmy wzór funkcji logistycznej do postaci $\sigma(s) = \frac{e^{s}}{e^{0}+e^{s}}$. Mówiąc inaczej, $\sigma(s)$ określa prawdopodobieństwo, że punkt danych należy do interesującej nas klasy, a $1 - \sigma(s) = \frac{e^{s}}{e^{0}+e^{s}}$ określa prawdopodobieństwo, że punkt danych nie należy do tej klasy.

Gdy zamiast tylko dwóch mamy wiele klas, to dla tego samego punktu danych powtarzamy ten sam proces wiele razy: jeden raz dla każdej klasy. Każda klasa charakteryzuje się własnym błędem i zbiorem wag, które tworzą liniową zależność pomiędzy cechami. W związku z tym dla punktu danych o wartościach cech x_1, x_2,..., i x_n, obliczamy k różnych kombinacji liniowych plus odchylenia:

$$\begin{aligned} s^1 &= \omega_0^1 + \omega_1^1 x_1 + \omega_2^1 x_2 + \cdots + \omega_n^1 x_n \\ s^2 &= \omega_0^2 + \omega_1^2 x_1 + \omega_2^2 x_2 + \cdots + \omega_n^2 x_n \\ &\vdots \\ s^k &= \omega_0^k + \omega_1^k x_1 + \omega_2^k x_2 + \cdots + \omega_n^k x_n \end{aligned}$$

Dobre nawyki

Warto wyrobić sobie dobry nawyk śledzenia liczby niewiadomych wartości ω występujących we wzorze funkcji szkoleniowej. Przypomnijmy, że są to wartości ω, które znajdujemy poprzez minimalizację funkcji straty. Innym dobrym nawykiem jest zastosowanie skutecznego i spójnego sposobu organizowania ich w całym modelu (w wektorze, macierzy itp.). W przypadku funkcji softmax, gdy mamy k klas i n cech dla każdego punktu danych, otrzymujemy $k \times n$ parametrów ω dla kombinacji liniowych oraz k odchyleń, co daje łącznie $k \times n + k$ nieznanych parametrów ω. Na przykład, jeśli używamy modelu regresji softmax do klasyfikacji obrazów w zbiorze danych MNIST (*https://en.wikipedia.org/wiki/MNIST_database*) odręcznie zapisanych cyfr, każdy obraz ma 28×28 pikseli, co oznacza 784 cechy. Chcemy przyporządkować je do 10 klas, więc ostatecznie musimy wykonać optymalizację dla 7850 wartości ω. W przypadku zarówno modelu regresji liniowej, jak i logistycznej mieliśmy tylko $n + 1$ nieznanych wartości ω, które trzeba było zoptymalizować.

Następnie przekazujemy każdy z tych k wyników do *funkcji softmax*, która uogólnia funkcję logistyczną z dwóch do wielu klas i jej wyniki także interpretujemy jako prawdopodobieństwo. Wzór na funkcję softmax wygląda następująco:

$$\sigma(s^j) = \frac{e^{s^j}}{e^{s^1} + e^{s^2} + \cdots + e^{s^k}}$$

W ten sposób dla tego samego punktu danych otrzymamy *k* wyników prawdopodobieństwa, po jednym wyniku odpowiadającym każdej klasie. Na koniec klasyfikujemy punkt danych jako należący do tej klasy, w której uzyskaliśmy największy wynik prawdopodobieństwa.

Podsumowując wszystko, co napisałam powyżej, otrzymujemy ostateczny wzór funkcji szkoleniowej, którą można teraz wykorzystać do klasyfikacji (tzn. po znalezieniu optymalnych wartości ω poprzez minimalizację odpowiedniej funkcji straty):

$y = j$ takie, że $\sigma\left(\omega_0^j + \omega_1^j x_1 + \cdots + \omega_n^j x_n\right)$ ma wartość maksymalną

Należy zauważyć, że w przypadku tej funkcji szkoleniowej trzeba jedynie wprowadzić cechy danych (wartości *x*), a w wyniku otrzymujemy jeden numer klasy: *j*.

Funkcja logistyczna i funkcja softmax a mechanika statystyczna

Czytelnicy znający dziedzinę mechaniki statystycznej być może zauważyli, że funkcje logistyczna i softmax obliczają prawdopodobieństwa w ten sam sposób, w jaki *funkcja partycji* z dziedziny mechaniki statystycznej oblicza prawdopodobieństwo znalezienia się systemu w określonym stanie.

Funkcja straty

Funkcję straty entropii krzyżowej w przypadku regresji logistycznej można zapisać następująco:

$$L(\vec{\omega}) = -\frac{1}{m}\sum_{i=1}^{m} y^i_{\text{rzeczywista}} \log\left(\sigma\left(\omega_0 + \omega_1 x_1^i + \cdots + \omega_n x_n^i\right)\right) + \left(1 - y^i_{\text{rzeczywista}}\right)$$
$$\log\left(1 - \sigma\left(\omega_0 + \omega_1 x_1^i + \cdots + \omega_n x_n^i\right)\right)$$

Można ją wyprowadzić z następujących zależności:

$$\text{cost} = \begin{cases} -\log\left(\sigma(s)\right) \text{ if } y_{\text{rzeczywista}} = 1 \\ -\log\left(1 - \sigma(s)\right) \text{ if } y_{\text{rzeczywista}} = 0 \end{cases} = -y_{\text{rzeczywista}} \log\left(\sigma(s)\right) - \left(1 - y_{\text{rzeczywista}}\right)$$
$$\log\left(1 - \sigma(s)\right)$$

Spróbujmy uogólnić tę samą logikę na wiele klas. Użyjemy notacji $y_{rzeczywista,\ i} = 1$, jeśli dany punkt danych należy do *i*-tej klasy, w przeciwnym razie $y_{rzeczywista,\ i} = 0$. Następnie możemy określić koszt związany z błędną klasyfikacją określonego punktu danych jako:

$$\text{koszt} = \begin{cases} -\log\left(\sigma(s^1)\right) \text{ jeśli } y_{\text{rzeczywista},1} = 1 \\ -\log\left(\sigma(s^2)\right) \text{ jeśli } y_{\text{rzeczywista},2} = 1 \\ -\log\left(\sigma(s^3)\right) \text{ jeśli } y_{\text{rzeczywista},3} = 1 \\ \vdots \\ -\log\left(\sigma(s^k)\right) \text{ jeśli } y_{\text{rzeczywista},k} = 1 \end{cases}$$
$$= -y_{\text{rzeczywista},1} \log\left(\sigma(s^1)\right) - \cdots - y_{\text{rzeczywista},k}$$
$$\log\left(\sigma(s^k)\right)$$

Po obliczeniu średniej wszystkich m punktów danych w zestawie szkoleniowym otrzymamy *uogólnioną funkcję straty entropii krzyżowej*, która opisuje przypadek wielu klas:

$$L(\vec{\omega}) = -\frac{1}{m}\sum_{i=1}^{m} y^i_{\text{rzeczywista},1}\log\left(\sigma\left(\omega_0^1 + \omega_1^1 x_1^i + \cdots + \omega_n^1 x_n^i\right)\right) + y^i_{\text{rzeczywista},2}$$
$$\log\left(\sigma\left(\omega_0^2 + \omega_1^2 x_1^i + \cdots + \omega_n^2 x_n^i\right)\right) + \cdots + y^i_{\text{rzeczywista},k}\log\left(\sigma\left(\omega_0^k + \omega_1^k x_1^i + \cdots + \omega_n^k x_n^i\right)\right)$$

Optymalizacja

Po wyprowadzeniu wzoru na funkcję straty możemy poszukać wartości ω, przy których ta funkcja osiąga minimum. Podobnie jak w przypadku większości funkcji straty, z którymi się spotykamy, nie ma jawnego wzoru na minimalizatory tej funkcji straty w odniesieniu do zbioru szkoleniowego i związanych z nim docelowych etykiet. Z tego względu zadowolimy się znalezieniem minimalizatorów za pomocą metod numerycznych. W szczególności zastosujemy zstępowanie gradientowe, stochastyczne zstępowanie gradientowe lub zstępowanie gradientowe z wykorzystaniem minipartii (patrz rozdział 4.). Trzeba pamiętać, że uogólniona funkcja straty między entropiami ma swoją wypukłość, która w procesie minimalizacji działa na naszą korzyść, zatem mamy gwarancję znalezienia poszukiwanych wartości ω.

Entropia krzyżowa a teoria informacji

Pojęcie entropii krzyżowej zostało zapożyczone z teorii informacji. Omówię to dokładniej przy okazji opisywania drzew decyzyjnych w dalszej części tego rozdziału. Na razie należy pamiętać o poniższej wielkości, w której p oznacza prawdopodobieństwo wystąpienia zdarzenia:

$$\log\left(\frac{1}{p}\right) = -\log(p)$$

Wielkość ta jest duża, gdy p jest małe, dlatego określa ilościowo większe *zaskoczenie dla mniej prawdopodobnych* zdarzeń.

Wykorzystanie omówionych modeli do ostatniej warstwy sieci neuronowej

Model regresji liniowej pozwala generować prognozy poprzez odpowiednie liniowe połączenie cech danych, a następnie dodanie odchylenia. Modele regresji logistycznej i regresji softmax dokonują klasyfikacji poprzez odpowiednie liniowe połączenie cech danych, dodanie odchylenia, a następnie przekazanie wyniku do funkcji oceny prawdopodobieństwa. W tych prostych modelach cechy danych są łączone tylko liniowo, dlatego wymienione modele słabo sprawdzają się w zadaniach wychwytywania potencjalnie ważnych nieliniowych interakcji między cechami danych. Modele sieci neuronowych uwzględniają w swoich funkcjach szkoleniowych nieliniowe *funkcje aktywacji*. Robią to w wielu warstwach, a zatem są lepiej przygotowane do wykrywania relacji nieliniowych i bardziej złożonych. Ostatnią warstwą sieci neuronowej jest jej warstwa wyjściowa. Warstwa tuż przed ostatnią warstwą generuje pewne

cechy wyższego rzędu i wprowadza je do ostatniej warstwy. Jeśli chcemy, aby sieć neuronowa przyporządkowywała dane do wielu klas, możemy przekształcić ostatnią warstwę w warstwę softmax. Jeśli chcemy, aby klasyfikowała do dwóch klas, to ostatnią warstwą może być warstwa regresji logistycznej. Z kolei gdy sieć ma prognozować wartości liczbowe, możemy uczynić jej ostatnią warstwę warstwą regresji. Przykłady takich rozwiązań zaprezentuję w rozdziale 5.

Inne popularne techniki i zestawy technik uczenia maszynowego

Po omówieniu regresji i regresji logistycznej przedstawię pojęcia dotyczące niektórych najpopularniejszych technik klasyfikacji i regresji. *Maszyny wektorów nośnych*, *drzewa decyzyjne* i *lasy losowe* oferują duże możliwości i można je wykorzystać zarówno do wykonywania zadań klasyfikacji, jak i regresji. Powstaje więc naturalne pytanie, kiedy używamy konkretnej metody uczenia maszynowego, w tym regresji liniowej i logistycznej, a później sieci neuronowych? Skąd mamy wiedzieć, której metody użyć i na której oprzeć uzyskane wnioski i przewidywania? W celu uzyskania odpowiedzi na te pytania warto skorzystać z technik analizy matematycznej modeli uczenia maszynowego.

Analiza matematyczna każdej metody, w tym rodzajów zestawów danych, dla których jest ona zwykle najlepsza, zyskała poważną uwagę dopiero niedawno, po zwiększeniu przydziału zasobów na badania w dziedzinie sztucznej inteligencji, uczenia maszynowego i nauki o danych. Obecna praktyka polega na próbowaniu każdej metody na tym samym zestawie danych i na użyciu tej z najlepszymi wynikami. Oznacza to, że o ile mamy wymagane zasoby obliczeniowe i czasowe, powinniśmy wypróbować różne techniki uczenia maszynowego. Co więcej, jeśli masz czas i zasoby na szkolenie różnych modeli uczenia maszynowego (idealne są tutaj obliczenia równoległe), dobrze jest wykorzystać **metody zespołowe** (ang. *ensemble methods*). Łączą one wyniki różnych modeli uczenia maszynowego poprzez uśrednianie lub głosowanie. Nieco zaskakujące, choć matematycznie uzasadnione, jest to, że często w ten sposób uzyskuje się lepsze wyniki w porównaniu ze stosowaniem najlepszych modeli pojedynczo, *a nawet wtedy, gdy wyniki tych najlepszych modeli są słabe*!

Jednym z przykładów metody zespołowej jest las losowy, czyli zespół drzew decyzyjnych.

Przy generowaniu prognoz z metod zespołowych pojawiają się terminy branżowe, takie jak *bagging* (czyli *agregacja bootstrapowa*), *pasting*, *boosting* (np. *ADA boost* lub *boosting gradientowy*), *stacking* oraz *losowe łatki* (ang. *random patches*). W metodach **bagging** i **pasting** szkolimy *ten sam* model uczenia maszynowego na różnych losowych podzbiorach zbioru szkoleniowego. Punkty danych w metodzie baggingu są próbkowane ze zbioru szkoleniowego z powtórzeniami, natomiast w metodzie pastingu punkty danych są próbkowane ze zbioru szkoleniowego bez powtórzeń. W metodzie **losowych łatek** próbkowanie dotyczy również przestrzeni cech, a model uczenia maszynowego jest szkolony na losowym podzbiorze cech w danym momencie. Jest to bardzo pomocne w przypadku, gdy zbiór danych obejmuje bardzo wiele cech. Przykładem są obrazy (gdzie każdy piksel jest cechą). **Stacking** uczy się mechanizmu przewidywania zespołu zamiast stosowania prostego głosowania lub uśredniania.

Maszyny wektorów nośnych

Maszyna wektorów nośnych to niezwykle popularna metoda uczenia maszynowego, która pozwala na wykonywanie zadań klasyfikacji i regresji zarówno z liniowymi (płaskimi), jak i nieliniowymi (zakrzywionymi) granicami decyzyjnymi.

W przypadku zadań klasyfikacji metoda ta ma na celu oddzielenie oznaczonych danych przy użyciu możliwie najszerszego marginesu, co skutkuje optymalną *autostradą* podziału, w przeciwieństwie do cienkiej linii podziału. Spróbuję wyjaśnić, w jaki sposób maszyny wektorów nośnych klasyfikują oznakowane egzemplarze danych w kontekście opisanej w tym rozdziale struktury funkcji szkoleniowej, funkcji straty i optymalizacji.

Funkcja szkoleniowa

Ponownie łączymy liniowo cechy punktu danych z nieznanymi wagami ω i dodajemy odchylenie ω_0. Następnie przekazujemy odpowiedź przez funkcję *znaku*. Jeśli liniowa kombinacja cech plus odchylenie jest liczbą dodatnią, funkcja zwraca 1 (czyli przyporządkowuje do pierwszej klasy), a jeśli jest liczbą ujemną, zwraca –1 (czyli przyporządkowuje do drugiej klasy). Tak więc wzór na funkcję szkoleniową przyjmuje taką oto postać:

$$f(\vec{\omega}; \vec{x}) = \text{znak}(\vec{\omega}^t \vec{x} + \omega_0)$$

Funkcja straty

Trzeba zaprojektować funkcję straty, która karze błędnie sklasyfikowane punkty. W przypadku regresji logistycznej wykorzystaliśmy funkcję straty entropii krzyżowej. W przypadku maszyn wektorów nośnych funkcja straty opiera się na funkcji zwanej *zawiasową funkcją straty*:

$$\max\left(0{,}1 - y_{\text{rzeczywista}}\,(\vec{\omega}^t \vec{x} + \omega_0)\right)$$

Zobaczmy, w jaki sposób zawiasowa funkcja straty karze błędy w klasyfikacji. Po pierwsze przypomnijmy, że w zależności od tego, czy punkt danych należy do klasy dodatniej, czy ujemnej, $y_{rzeczywista}$ wynosi 1 lub –1.

- Jeśli dla pewnego punktu danych $y_{rzeczywista}$ wynosi 1, ale $\vec{\omega}^t \vec{x} + \omega_0 < 0$, funkcja szkoleniowa sklasyfikuje go błędnie i uzyskamy $y_{prognoza} = -1$, a wartość zawiasowej funkcji straty wyniesie $1 - (1)(\vec{\omega}^t \vec{x} + \omega_0) > 1$, co w przypadku, gdy celem jest minimalizacja, jest wysoką karą.
- Z drugiej strony, jeśli $y_{rzeczywista}$ wynosi 1, a $\vec{\omega}^t \vec{x} + \omega_0 > 0$, funkcja szkoleniowa sklasyfikuje go poprawnie i uzyskamy wynik $y_{prognoza} = 1$. Zawiasowa funkcja straty jest jednak zaprojektowana w taki sposób, że nadal będzie nas karać, jeśli $\vec{\omega}^t \vec{x} + \omega_0 < 1$, a jej wartość będzie wynosić $1 - (1)(\vec{\omega}^t \vec{x} + \omega_0)$. Teraz jest ona mniejsza od 1, ale nadal większa od 0.
- Tylko wtedy, gdy $y_{rzeczywista}$ wynosi 1 i $\vec{\omega}^t \vec{x} + \omega_0 > 1$ (funkcja szkoleniowa nadal poprawnie sklasyfikuje ten punkt i zwróci wynik $y_{prognoza} = 1$), wartość zawiasowej funkcji straty będzie wynosić 0, ponieważ będzie to maksimum między 0 a wartością ujemną.

- Ta sama logika ma zastosowanie w przypadku, gdy $y_{rzeczywista}$ wynosi –1. Zawiasowa funkcja straty nałoży wysoką karę za błędną prognozę i niezbyt wysoką za poprawną prognozę, gdy nie będzie wystarczającego *marginesu* od *dzielnika 0* (margines większy niż 1). Zawiasowa funkcja straty zwróci 0 tylko wtedy, gdy prognoza będzie prawidłowa, a punkt danych będzie się znajdował w odległości większej niż 1 od dzielnika 0.
- Zauważmy, że dzielnik 0 ma równanie $\vec{\omega}^t\vec{x} + \omega_0 = 0$, a krawędzie brzegowe mają równania $\vec{\omega}^t\vec{x} + \omega_0 = -1$ i $\vec{\omega}^t\vec{x} + \omega_0 = 1$. Odległość między krawędziami marginesu można łatwo obliczyć jako $\frac{2}{\|\omega\|_2}$. Jeśli więc chcemy zwiększyć tę szerokość marginesu, musimy zmniejszyć $\|\omega\|_2$; zatem ten wyraz musi wejść do funkcji straty wraz z zawiasową funkcją straty, która karze zarówno błędnie sklasyfikowane punkty, jak i punkty w granicach marginesu.

Gdy teraz uśrednimy stratę zawiasową dla wszystkich m punktów danych w zbiorze szkoleniowym i dodamy $\| \omega \|_2^2$, otrzymamy wzór na funkcję straty, która jest powszechnie stosowana w maszynach wektorów nośnych:

$$L(\vec{\omega}) = \frac{1}{m}\sum_{i=1}^{m} \max\left(0{,}1 - y^i_{\text{rzeczywista}}(\vec{\omega}^t\vec{x}^i + \omega_0)\right) + \lambda \| \omega \|_2^2$$

Optymalizacja

Celem jest znalezienie wektora $\vec{\omega}$, który minimalizuje funkcję straty. Przyjrzyjmy się tej funkcji straty nieco dokładniej:

- Funkcja składa się z dwóch wyrazów: $\frac{1}{m}\sum_{i=1}^{m} \max\left(0{,}1 - y^i_{\text{rzeczywista}}(\vec{\omega}^t\vec{x}^i + \omega_0)\right)$ i $\lambda \| \vec{\omega} \|_2^2$. Ilekroć w problemie optymalizacyjnym występuje więcej niż jeden wyraz, najprawdopodobniej są to wyrazy konkurujące, w tym sensie, że te same wartości ω, które sprawiają, że pierwszy wyraz ma niską wartość (co jest korzystne), mogą sprawić, że drugi wyraz będzie miał wysoką wartość (co jest niekorzystne). Zatem podczas poszukiwań wektora $\vec{\omega}$, który optymalizuje sumę tych dwóch wyrazów, toczy się między nimi gra typu *pchaj-i-ciągnij*.
- Wartość λ występująca w wyrazie $\lambda \| \vec{\omega} \|_2^2$ jest przykładem hiperparametru modelu, który można dostroić na etapie walidacji procesu uczenia. Zauważmy, że kontrolowanie wartości λ pomaga kontrolować szerokość marginesu w następujący sposób: jeśli wybierzemy dużą wartość λ, to aby zrekompensować dużą wartość $\vec{\omega}$, optymalizator będzie zajęty wybieraniem wartości $\| \omega \|_2^2$ z bardzo niską wartością wyrazu λ, a pierwszy wyraz funkcji straty będzie mieć mniejsze znaczenie. Należy jednak pamiętać, że mniejsza wartość $\|\omega\|_2$ oznacza większy margines!
- Wyraz $\lambda \| \vec{\omega} \|_2^2$ można również traktować jako *wyraz regularyzacji*, o którym opowiem w rozdziale 4.
- Omawiana funkcja straty jest wypukła i ograniczona poniżej przez 0, więc rozwiązanie problemu jej minimalizacji nie jest zbyt trudne: nie ma obaw o utknięcie w lokalnych minimach. Pierwszy wyraz ma osobliwość, ale jak wspomniano wcześniej, można zdefiniować jego subgradient w punkcie osobliwym, a następnie zastosować metodę zstępowania gradientu.

Formułę niektórych problemów optymalizacyjnych można zmodyfikować, dzięki czemu zamiast rozwiązywania problemu *pierwotnego* możemy rozwiązywać dla niego problem *wtórny*! Zazwyczaj jeden z nich jest łatwiejszy do rozwiązania niż drugi. O problemie wtórnym można pomyśleć jak o innym problemie optymalizacyjnym, który żyje w równoległym wszechświecie problemu pierwotnego. Wszechświaty spotykają się w optymalizatorze. Z tego względu rozwiązanie jednego problemu automatycznie daje rozwiązanie drugiego. Gdy uczymy się optymalizacji, badamy dualność. Szczególnym zainteresowaniem i ogromnym zastosowaniem cieszy się optymalizacja liniowa i kwadratowa, znana również jako programowanie liniowe i kwadratowe. Bieżący problem minimalizacji:

$$\min \vec{\omega} \frac{1}{m} \sum_{i=1}^{m} \max\left(0{,}1 - y^{i}_{\text{rzeczywista}} \left(\vec{\omega}^{t} \vec{x}^{i} + \omega_0\right)\right) + \lambda \parallel \omega \parallel_2^2$$

jest przykładem programowania kwadratowego i odpowiada mu problem wtórny, który okazuje się łatwiejszy do optymalizacji niż pierwotny (zwłaszcza gdy liczba cech jest wysoka):

$$\max \vec{a} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{j=1}^{m} \sum_{k=1}^{m} \alpha_j \alpha_k y^{j}_{\text{rzeczywista}} y^{k}_{\text{rzeczywista}} \left((\vec{x}^{j})^{t} \vec{x}^{k}\right)$$

z zastrzeżeniem ograniczeń $\alpha_j \geq 0$ i $\sum_{j=1}^{m} \alpha_j y^{j}_{\text{rzeczywista}} = 0$. Zapisanie tego wzoru, gdy znamy problemy pierwotne i wtórne, jest zwykle proste, dlatego pominę jego wyprowadzenie, aby nie przerywać płynności opisu.

Programowanie kwadratowe jest bardzo dobrze rozwiniętą dziedziną. Istnieje wiele pakietów oprogramowania, które pozwalają rozwiązać ten problem. Po znalezieniu maksymalizującego $\vec{\alpha}$ można znaleźć wektor $\vec{\omega}$, który minimalizuje problem pierwotny przy użyciu wyrażenia $\vec{\omega} = \sum_{j=1}^{m} \alpha_j y^{i}_{\text{rzeczywista}} \vec{x}^{j}$. Po wyznaczeniu $\vec{\omega}$ możemy sklasyfikować nowe punkty danych za pomocą przeszkolonej funkcji:

$$\begin{aligned} f\left(\vec{x}_{\text{nowy}}\right) &= \text{znak}\left(\vec{\omega}^{t} \vec{x}_{\text{nowy}} + \omega_0\right) \\ &= \text{znak}\left(\sum_{j} \alpha_j y^{i} (\vec{x}^{j})^{t} \vec{x}_{\text{nowy}} + \omega_0\right) \end{aligned}$$

Jeśli chcesz uniknąć stosowania programowania kwadratowego, możesz skorzystać z innej bardzo szybkiej metody zwanej **zstępowaniem współrzędnych**, która rozwiązuje problem wtórny i bardzo dobrze się sprawdza w przypadku dużych zbiorów danych z dużą liczbą cech.

Sztuczka z jądrem

Możemy teraz przenieść te same pomysły na klasyfikację nieliniową. Zauważmy najpierw ważną uwagę dotyczącą problemu wtórnego: punkty danych występują tylko parami, a dokładniej tylko w iloczynie skalarnym, mianowicie $(\vec{x}^{j})^{t} \vec{x}^{k}$. Na podobnej zasadzie w przeszkolonej funkcji także występują tylko w postaci iloczynu skalarnego. Ta prosta obserwacja pozwala na zastosowanie magii:

- Jeśli znajdziesz funkcję $K(\vec{x}^j, \vec{x}^j)$, którą można zastosować do par punktów danych, i okazuje się, że ta funkcja zwraca iloczyn skalarny par przekształconych punktów danych w pewnej przestrzeni o wyższym wymiarze (bez znajomości rzeczywistej transformacji), to można rozwiązać ten sam problem wtórny w przestrzeni o wyższym wymiarze poprzez zastąpienie iloczynu skalarnego we wzorze problemu wtórnego wyrażeniem

  $K(\vec{x}^j, \vec{x}^j)$
- Intuicja w tym przypadku podpowiada, że dane, które można odseparować nieliniowo w niższych wymiarach, prawie zawsze są separowalne liniowo w wymiarach wyższych. Należy zatem przekształcić wszystkie punkty danych do wyższych wymiarów, a następnie je rozdzielić. Problem klasyfikacji liniowej w wyższych wymiarach *bez* przekształcania każdego punktu danych rozwiązuje sztuczka jądra. Samo jądro oblicza iloczyn skalarny przekształconych danych *bez* przekształcania danych. To dość interesujące.

Do przykładów funkcji jądra można zaliczyć:

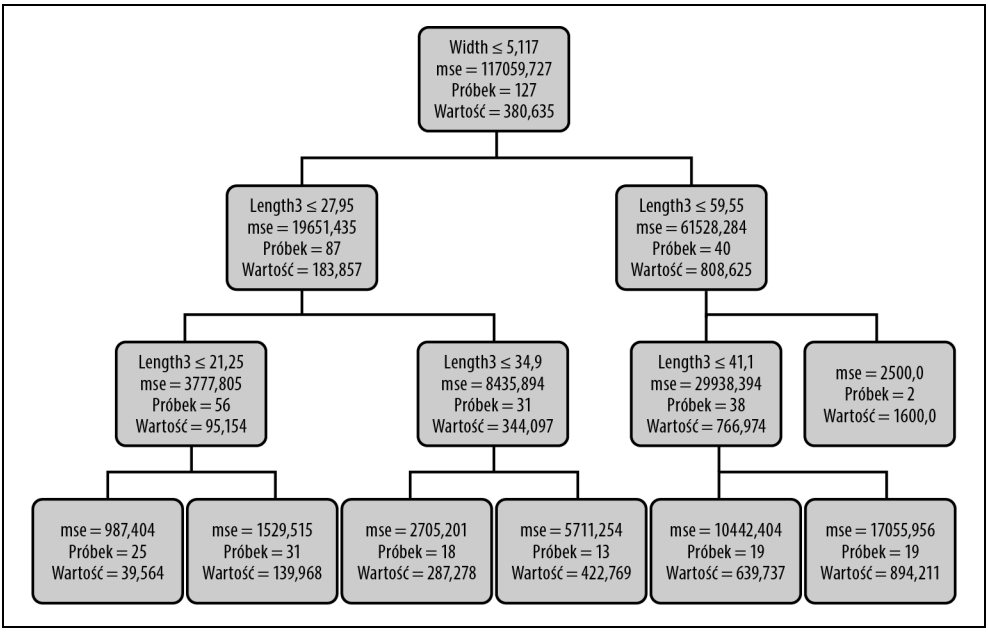
- $K(\vec{x}^j, \vec{x}^j) = ((\vec{x}^j)^t \vec{x}^j)^2$.
- Jądro wielomianowe: $K(\vec{x}^j, \vec{x}^j) = (1 + (\vec{x}^j)^t \vec{x}^j)^d$.
- Jądro gaussowskie: $K(\vec{x}^j, \vec{x}^j) = e^{-\gamma|x_j - x_k|^2}$.

Drzewa decyzyjne

Trzymając się motywu przewodniego w tym rozdziale, że wszystko jest funkcją, drzewo decyzyjne w gruncie rzeczy jest funkcją, która przyjmuje zmienne logiczne jako dane wejściowe (są to zmienne, które mogą przyjmować tylko wartości `true` [lub 1] bądź `false` [lub 0]), np. cecha > 5, cecha = słoneczny, cecha = mężczyzna itp. Drzewo generuje *decyzje*, np. zgodę na pożyczkę, sklasyfikowanie jako *covid19*, zwrot 25 itd. Zamiast dodawać lub mnożyć zmienne logiczne, używamy w odniesieniu do nich operatorów logicznych `or`, `and` i `not`.

Co jednak należy zrobić, jeśli cechy w wejściowym zestawie danych nie zostały podane jako zmienne logiczne? W takim przypadku przed wprowadzeniem ich do modelu w celu wygenerowania prognoz trzeba je przekształcić w zmienne logiczne. Na przykład drzewo decyzyjne przedstawione na rysunku 3.17 zostało przeszkolone na zbiorze danych Fish Market. Jest to drzewo regresji. Drzewo pobiera nieprzetworzone dane, ale funkcja reprezentująca drzewo faktycznie działa na nowych zmiennych, które są oryginalnymi cechami danych przekształconymi na zmienne logiczne:

```
a1 = (Width ≤ 5.117).

a2 = (Length3 ≤ 59.55).

a3 = (Length3 ≤ 41.1).

a4 = (Length3 ≤ 34.9).

a5 = (Length3 ≤ 27.95).

a6 = (Length3 ≤ 21.25).
```



Rysunek 3.17. Regresyjne drzewo decyzyjne skonstruowane na zbiorze danych Fish Market. Szczegółowe informacje można znaleźć w dołączonym notatniku Jupyter

Funkcja reprezentująca drzewo decyzyjne pokazane na rysunku 3.17 to:

$$f(a1, a2, a3, a4, a5, a6) = (a1 \text{ and } a5 \text{ and } a6) \times 39{,}584 + (a1 \text{ i } a5 \text{ and not } a6) \times 139{,}968 + (a1 \text{ and not } a5 \text{ and } a4) \times 287{,}278 + (a1 \text{ and not } a5 \text{ and not } a4) \times 422{,}769 + (\text{not } a1 \text{ and } a2 \text{ and } a3) \times 639{,}737 + (\text{not } a1 \text{ and } a2 \text{ and not } a3) \times 824{,}211 + (\text{not } a1 \text{ and not } a2) \times 1600$$

Warto zauważyć, że w przeciwieństwie do funkcji szkoleniowych, z którymi mieliśmy do czynienia do tej pory w tym rozdziale, ta funkcja nie zawiera parametrów ω, które trzeba znaleźć. Nazywa się to **modelem nieparametrycznym**, dla którego nie ustala się z wyprzedzeniem kształtu funkcji. Dzięki temu uzyskujemy elastyczność polegającą na *dorastaniu* wraz z danymi lub mówiąc inaczej, dostosowywaniu się do danych. Oczywiście, z tą wysoką zdolnością adaptacji do danych wiąże się wysokie ryzyko nadmiernego dopasowania danych. Na szczęście istnieją sposoby na obejście tego problemu. Niektóre wymienię tutaj bez żadnego rozwinięcia: przycinanie drzewa po jego wzroście, ograniczenie liczby warstw, ustawienie minimalnej liczby punktów danych na węzeł lub wykorzystanie zbioru drzew zamiast jednego, co określa się jako *las losowy* (omówię go później).

Jedna bardzo ważna obserwacja: drzewo *decyzyjne* zdecydowało się rozdzielić tylko dwie cechy oryginalnego zestawu danych, a mianowicie cechy `Width` i `Length3`. Drzewa decyzyjne są zaprojektowane w taki sposób, aby ważniejsze cechy (te, które dostarczają najwięcej informacji mających wpływ na prognozę) znajdowały się bliżej korzenia. Z tego względu drzewa decyzyjne mogą pomóc w selekcji cech polegającej na wyborze najważniejszych cech, które przyczyniają się do prognoz ostatecznego modelu.

Nic dziwnego, że cechy `Width` (szerokość) i `Length3` (długość3) okazały się najważniejsze dla prognozy wagi ryb. Macierz korelacji pokazana na rysunku 3.18 i wykresy rozrzutu z rysunku 3.3 pokazują niezwykle silną korelację między wszystkimi cechami długości. Oznacza to, że dostarczane przez nie informacje są nadmiarowe, a uwzględnienie ich wszystkich w modelach predykcyjnych zwiększy koszty obliczeniowe i obniży wydajność.

	Weight	Length1	Length2	Length3	Height	Width
Weight	1.000000	0.908678	0.911888	0.917883	0.747700	0.896036
Length1	0.908678	1.000000	0.999493	0.991731	0.637844	0.870414
Length2	0.911888	0.999493	1.000000	0.993869	0.653291	0.877268
Length3	0.917883	0.991731	0.993869	1.000000	0.716450	0.882716
Height	0.747700	0.637844	0.653291	0.716450	1.000000	0.802115
Width	0.896036	0.870414	0.877268	0.882716	0.802115	1.000000

Rysunek 3.18. Macierz korelacji dla zestawu danych Fish Market. Istnieje niezwykle silna korelacja między wszystkimi cechami długości

Wybór cech

Właśnie wprowadziłam bardzo ważny temat wyboru cech. Rzeczywiste zbiory danych zawierają wiele cech, a niektóre z nich mogą dostarczać nadmiarowych informacji, podczas gdy inne w ogóle nie mają znaczenia dla prognozowania docelowej etykiety. Uwzględnienie w modelu uczenia maszynowego nieistotnych i nadmiarowych cech zwiększa koszty obliczeniowe i obniża jego wydajność. Właśnie pokazałam, że jednym ze sposobów na wybranie ważnych cech są drzewa decyzyjne. Innym sposobem jest technika regularyzacji zwana regresją lasso, którą przedstawię w rozdziale 4. Istnieją testy statystyczne, które testują wzajemne zależności cech. *Test F* testuje zależności liniowe (daje wyższe wyniki dla skorelowanych cech, ale same korelacje są zwodnicze), natomiast *informacje wzajemne* testują zależności nieliniowe. Wymienione testy zapewniają miarę określającą stopień, w jakim określona cecha przyczynia się do określenia etykiety docelowej, a tym samym pomagają w wyborze cech poprzez zachowanie tych, które są najbardziej obiecujące. Można również przetestować zależności cech od siebie nawzajem, a także ich korelacje i wykresy rozrzutu. **Wyznaczanie progu wariancji** usuwa cechy o niewielkiej lub zerowej wariancji. Jest to zgodne z założeniem, że jeśli cecha nie różni się zbytnio między różnymi punktami danych, niezbyt dobrze nadaje się do generowania prognoz.

W jaki sposób szkoli się drzewo decyzyjne na zbiorze danych? Jaką funkcję należy zoptymalizować? Są dwie funkcje, które są zwykle optymalizowane podczas *wzrostu* drzew decyzyjnych: entropia i nieczystość Giniego. Użycie jednej bądź drugiej nie sprawia dużej różnicy w wynikowych drzewach. Poniżej omówię to bardziej szczegółowo.

Entropia i nieczystość Giniego

Funkcje te wspomagają podjęcie decyzji o podzieleniu węzła drzewa względem cechy, która jest oceniana jako *najważniejsza*. Entropia i nieczystość Giniego to dwa popularne sposoby pomiaru ważności cechy. Nie są one matematycznie równoważne, ale oba działają i zapewniają stworzenie rozsądnych drzew decyzyjnych. Nieczystość Giniego wiąże się zwykle z niższymi kosztami obliczeniowymi, więc jest to metoda domyślna w wielu pakietach oprogramowania. Mamy jednak możliwość zmiany ustawienia domyślnego na entropię. Zastosowanie funkcji nieczystości Giniego często prowadzi do tworzenia mniej zrównoważonych drzew w przypadku, gdy w danych występują klasy o znacznie wyższej częstości niż inne. Klasy te są odizolowane we własnych gałęziach. Jednak w wielu przypadkach użycie entropii lub nieczystości Giniego nie powoduje dużej różnicy w wynikowych drzewach decyzyjnych.

W podejściu polegającym na wykorzystaniu **entropii** szukamy podziału funkcji, który zapewnia *maksymalny zysk informacyjny* (wkrótce podam jego wzór). Pojęcie zysku informacyjnego zostało zapożyczone z teorii informacji i ma związek z pojęciem entropii. Entropia z kolei jest zapożyczona z termodynamiki i fizyki statystycznej i określa ilościowo wielkość nieporządku w określonym systemie.

W podejściu **nieczystości Giniego** szukamy takiego podziału funkcji, który zapewnia węzłom potomnym najniższą średnią wartość nieczystości Giniego (wzór na tę wielkość również zaprezentuję wkrótce).

Aby zmaksymalizować zysk informacyjny (lub zminimalizować nieczystość Giniego), algorytm rozwijający drzewo decyzyjne musi przejrzeć wszystkie cechy podzbioru danych szkoleniowych i obliczyć zysk informacyjny (lub nieczystość Giniego), ustalić, czy drzewo wykorzystuje tę konkretną cechę jako węzeł do podziału, a następnie wybrać cechę, która zapewnia najwyższy zysk informacyjny (bądź węzły potomne o najniższej średniej wartości nieczystości Giniego). Co więcej, jeśli cechę opisują rzeczywiste wartości liczbowe, algorytm musi zdecydować, *jakie pytanie zadać węzłowi*, czyli według jakiej wartości cechy dokonać podziału, np. czy $x_5 < 0{,}1$? Algorytm musi to zrobić sekwencyjnie w każdej warstwie drzewa: obliczyć zysk informacyjny (lub nieczystość Giniego) dla cech punktów danych w każdym węźle, a czasem dla każdego możliwego podziału wartości. Łatwiej to zrozumieć na przykładach.

Najpierw jednak zapiszemy wzory na entropię, zysk informacyjny i nieczystość Giniego.

Entropia i zysk informacyjny

Najprostszym sposobem na zrozumienie wzoru na entropię jest wykorzystanie intuicji. Zgodnie z nią, jeśli zdarzenie jest wysoce prawdopodobne, to jego wystąpienie wiąże się z niewielkim zaskoczeniem. Tak więc, gdy *p*(*zdarzenie*) jest duże, związane z nim zaskoczenie jest niskie. Matematycznie można to zakodować za pomocą funkcji, która maleje wraz ze wzrostem prawdopodobieństwa. Do tego celu nadaje się funkcja rachunku logarytmicznego $\log \frac{1}{x}$, która ma dodatkową cechę, zgodnie z którą zaskoczenia dotyczące zdarzeń niezależnych sumują się. W związku z tym można zapisać następującą definicję:

$$Zaskoczenie(\text{zdarzenie}) = \log \frac{1}{p(\text{zdarzenie})} = -\log\left(p(\text{zdarzenie})\right)$$

Teraz entropię zmiennej losowej (która w tym przypadku jest konkretną cechą w zestawie danych szkoleniowych) można zdefiniować jako *oczekiwane zaskoczenie* związane ze zmienną losową. Trzeba więc zsumować zaskoczenia każdego możliwego wyniku zmiennej losowej (zaskoczenie dla każdej wartości określonej cechy) pomnożone przez odpowiadające im prawdopodobieństwa. W ten sposób otrzymujemy:

$$\text{Entropia}(X) = -p(\text{wynik}_1)\log\left(p(\text{wynik}_1)\right) - p(\text{wynik}_2)$$
$$\log\left(p(\text{wynik}_2)\right) - \cdots - p(\text{wynik}_n)\log\left(p(\text{wynik}_n)\right)$$

Entropię dla jednej cechy danych szkoleniowych, która przyjmuje kilka wartości, można zapisać w następujący sposób:

$$\text{Entropia}\left(\text{Cecha}^2\right) = -p(\text{wartość}_1)\log\left(p(\text{wartość}_1)\right) - p(\text{wartość}_2)\log\left(p(\text{wartość}_2)\right) - \cdots - p(\text{wartość}_n)\log\left(p(\text{wartość}_n)\right)$$

Ponieważ celem w tym przykładzie jest wybranie takiego podziału według cechy, który zapewnia duży zysk informacji o wyniku (etykieta lub cecha docelowa), najpierw obliczmy entropię cechy wyniku.

Wyjście binarne

Przyjmijmy dla uproszczenia, że mamy do czynienia z problemem klasyfikacji binarnej, więc cecha wyniku ma tylko dwie wartości: dodatnią (należy do klasy) i ujemną (nie należy do klasy).

Jeśli przyjmiemy, że p oznacza liczbę punktów danych z wynikami dodatnimi docelowej funkcji, a n jest liczbą wyników ujemnych, to $p + n = m$ oznacza liczbę punktów danych w podzbiorze danych szkoleniowych. Teraz prawdopodobieństwo wybrania punktu danych o wartości dodatniej z tej kolumny docelowej będzie wynosić $\frac{p}{m} = \frac{p}{p+n}$. Podobnie prawdopodobieństwo wybrania punktu danych o wartości ujemnej będzie wynosić:

$$\frac{n}{m} = \frac{n}{p+n}$$

Zatem entropia cechy wynikowej (bez korzystania z jakichkolwiek informacji z innych cech) wynosi:

$$\textit{Entropia}(\text{cecha wyniku})$$
$$= -p(\text{dodatni})\log\left(p(\text{dodatni})\right) - p(\text{ujemny})\log\left(p(\text{ujemny})\right)$$
$$= -\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right)$$

Następnie wykorzystujemy informacje z jednej innej cechy i obliczamy różnicę w entropii cechy wyniku, która jak można oczekiwać, zmniejszy się wraz z uzyskaniem większej ilości informacji (ogólnie rzecz biorąc, im więcej informacji, tym mniejsze zaskoczenie).

Załóżmy, że do podziału węzła drzewa decyzyjnego wybraliśmy cechę A. Załóżmy, że cecha A przyjmuje cztery wartości i ma k_1 punktów danych z wartością *wartość*$_1$, z których dla p_1 oznaczono wynik docelowy jako dodatni, a dla n_1 oznaczono docelowy wynik jako ujemny. Tak więc $p_1 + n_1 = k_1$. Podobnie cecha A ma k_2 punktów danych z wartością *wartość*$_2$, z których dla p_2

oznaczono wynik docelowy jako dodatni, a dla n_2 oznaczono wynik docelowy jako ujemny. Tak więc $p_2 + n_2 = k_2$. To samo dotyczy wartości *wartość*$_3$ i *wartość*$_4$ cechy *A*. Zauważ, że całkowita liczba punktów danych w podzbiorze szkoleniowym zbioru danych wynosi $k_1 + k_2 + k_3 + k_4 = m$.

O każdej wartości dodatniej *wartość*$_k$ cechy *A* można pomyśleć jako o zmiennej losowej w swoim własnym aspekcie, z p_k wynikami dodatnimi i n_k wynikami ujemnymi. Dzięki temu możemy obliczyć jej entropię (oczekiwane zaskoczenie):

$$Entropia(\text{ wartość }_1) = -\frac{p_1}{p_1+n_1}\log\left(\frac{p_1}{p_1+n_1}\right) - \frac{n_1}{p_1+n_1}\log\left(\frac{n_1}{p_1+n_1}\right)$$
$$Entropia\ (\text{ wartość }_2) = -\frac{p_2}{p_2+n_2}\log\left(\frac{p_2}{p_2+n_2}\right) - \frac{n_2}{p_2+n_2}\log\left(\frac{n_2}{p_2+n_2}\right)$$
$$Entropia\ (\text{ wartość }_3) = -\frac{p_3}{p_3+n_3}\log\left(\frac{p_3}{p_3+n_3}\right) - \frac{n_3}{p_3+n_3}\log\left(\frac{n_3}{p_3+n_3}\right)$$
$$Entropia\ (\text{ wartość }_4) = -\frac{p_4}{p_4+n_4}\log\left(\frac{p_4}{p_4+n_4}\right) - \frac{n_4}{p_4+n_4}\log\left(\frac{n_4}{p_4+n_4}\right)$$

Na podstawie tych informacji można obliczyć oczekiwaną entropię po podziale według cechy *A* — poprzez dodanie czterech wspomnianych entropii, każdej pomnożonej przez właściwe prawdopodobieństwo: $p(\text{ wartość }_1) = \frac{k_1}{m}, p(\text{ wartość }_2) = \frac{k_2}{m}, p(\text{ wartość }_3) = \frac{k_3}{m}$, i $p(\text{ wartość }_4) = \frac{k_4}{m}$

W związku z tym oczekiwana entropia po podziale wg cechy *A* wyniesie:

$$\begin{aligned}&\text{Oczekiwana entropia}(\text{cecha A}^{\,2})\\ &= p(\text{wartość }_1)Entropia(\text{wartość}_1) + p(\text{wartość }_2)\ Entropia(\text{wartość }_2)\\ &+p(\text{wartość }_3)\ Entropia(\text{wartość}_3) + p(\text{wartość }_4)\ Entropia(\text{wartość }_4)\\ =\frac{k_1}{m}&Entropia(\text{wartość}_1) + \frac{k_2}{m}Entropia(\text{wartość }_2) + \frac{k_3}{m}Entropia(\text{wartość }_3)\\ &+\frac{k_4}{m}Entropia(\text{wartość }_4)\end{aligned}$$

Jakie uzyskamy informacje, gdy do podziału użyjemy cechy *A*? Będzie to różnica między entropią cechy wynikowej bez żadnych informacji z cechy *A* a oczekiwaną entropią cechy *A*. Oznacza to, że mamy wzór na *zysk informacyjny*, pod warunkiem że zdecydowaliśmy się na dokonanie podziału według cechy *A*:

Zysk informacyjny

Entropia(cecha wynikowa) – Oczekiwana entropia(cecha A) =

$$-\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right) - \text{Oczekiwana entropia(cecha A)}$$

Teraz można łatwo skorzystać z każdej z cech należących do podzbioru danych szkoleniowych i obliczyć zysk informacyjny wynikający z użycia do podziału tej cechy. Ostatecznie algorytm drzewa decyzyjnego decyduje o podziale według cechy o najwyższym zysku informacyjnym.

Algorytm wykonuje to rekurencyjnie dla każdego węzła i każdej warstwy drzewa, aż do wyczerpania cech do podziału lub punktów danych. W ten sposób uzyskujemy drzewo decyzyjne oparte na entropii.

Wyjście wieloklasowe

Nie jest zbyt trudno uogólnić tę logikę na przypadek, w którym mamy wieloklasowy wynik, na przykład rozwiązujemy problem klasyfikacji z trzema lub większą liczbą docelowych etykiet. Doskonałym przykładem zbioru z trzema etykietami docelowymi jest klasyczny zestaw danych Iris (*https://archive.ics.uci.edu/dataset/53/iris*) z repozytorium UCI Machine Learning Repository (*https://archive.ics.uci.edu/*). Ten zestaw danych zawiera cztery cechy dla każdego rodzaju irysa: długość i szerokość działki kielicha oraz długość i szerokość płatka. Należy pamiętać, że każda z tych cech jest ciągłą, a nie dyskretną zmienną losową. Trzeba więc opracować test, który przed zastosowaniem *wcześniej* wspomnianej logiki dokona podziału według wartości każdej cechy. Jest to część etapu inżynierii cech projektu nauki o danych. Krokiem inżynieryjnym jest w tym przypadku przekształcenie cechy o wartości ciągłej w cechę logiczną. Na przykład czy długość płatka > 2,45? Nie będę omawiać, w jaki sposób została wybrana liczba 2,45, ale jak łatwo się domyślić, w tym przypadku trzeba również zastosować proces optymalizacji.

Nieczystość Giniego

Każde drzewo decyzyjne zawiera węzły, gałęzie i liście. Węzeł jest uważany za *czysty*, jeśli zawiera wyłącznie takie punkty danych z podzbioru danych szkoleniowych, które mają tę samą etykietę docelową (oznacza to, że należą do tej samej klasy). Łatwo zauważyć, że czysty węzeł jest stanem pożądanym, ponieważ znamy jego klasę. Z tego powodu algorytm powinien rozwijać drzewo w sposób, który minimalizuje nieczystość węzłów: jeśli punkty danych w węźle nie należą do tej samej klasy, to węzeł jest *nieczysty*. Nieczystość Giniego określa tę wielkość ilościowo, w sposób opisany poniżej.

Załóżmy, że w problemie klasyfikacji występują trzy klasy, tak jak w przypadku zestawu danych Iris (*https://archive.ics.uci.edu/dataset/53/iris*). Załóżmy również, że określony węzeł w drzewie decyzyjnym utworzonym w celu dopasowania do tego zestawu danych zawiera n szkoleniowych punktów danych, z których n_1 należy do pierwszej klasy, n_2 do drugiej klasy i n_3 do trzeciej klasy (zatem $n_1 + n_2 + n_3 = n$). W tym przypadku wzór na nieczystość Giniego dla tego węzła można zapisać w następujący sposób:

$$\text{Nieczystość Giniego} = 1 - \left(\frac{n_1}{n}\right)^2 - \left(\frac{n_2}{n}\right)^2 - \left(\frac{n_3}{n}\right)^2$$

Zatem dla każdego węzła wyznaczamy liczbę punktów danych należących do każdej klasy, podnosimy tę wartość do kwadratu, a następnie sumę tych wartości odejmujemy od 1. Należy zauważyć, że jeśli wszystkie egzemplarze danych węzła należą do tej samej klasy, to z tego wzoru otrzymujemy nieczystość Giniego równą 0.

Algorytm wzrostu drzewa decyzyjnego szuka teraz cechy i punktu podziału, które generują węzły potomne o najniższej średniej wartości nieczystości Giniego. Oznacza to, że węzły potomne danego węzła średnio muszą być czystsze niż ich węzeł nadrzędny. Tak więc algorytm

stara się zminimalizować średnią ważoną nieczystości Giniego dwóch węzłów potomnych (drzewo binarne). Nieczystość Giniego każdego węzła potomnego jest ważona z wykorzystaniem jego względnego rozmiaru, oznaczającego stosunek liczby jego punktów danych do całkowitej liczby punktów danych w tej warstwie drzewa (która jest taka sama jak liczba punktów danych jego rodzica). W związku z tym trzeba szukać kombinacji cech i punktów podziału (dla każdej cechy), które rozwiązują następujący problem minimalizacji:

$$min_{\text{Cecha, wartość podziału cechy}} \frac{n_{\text{lewy}}}{n} \text{ Gini(węzeł lewy)} + \frac{n_{\text{prawy}}}{n} \text{ Gini(węzeł prawy)}$$

gdzie n_{lewy} i n_{prawy} to liczba punktów danych, które ostatecznie trafiają do lewego i prawego węzła potomnego, a n to liczba punktów danych, które znajdują się w węźle nadrzędnym (należy pamiętać, że wartości n_{lewy} i n_{prawy} muszą sumować się do n).

Regresyjne drzewa decyzyjne

Należy podkreślić, że drzewa decyzyjne można wykorzystywać zarówno do regresji, jak i klasyfikacji. Regresyjne drzewo decyzyjne zwraca prognozowaną wartość, a nie klasę, ale ma do niego zastosowanie podobny proces jak do drzew klasyfikacyjnych.

Zamiast zastosowania podziału węzła ze względu na cechę i wartość tej cechy (na przykład czy wysokość > 1 metr?), co maksymalizuje zysk informacyjny lub minimalizuje nieczystość Giniego, wybieramy taką cechę i jej wartość, które minimalizują średnią kwadratową odległość między rzeczywistymi etykietami a średnią etykiet wszystkich punktów danych w każdym lewym i prawym węźle potomnym. Oznacza to, że algorytm wybiera cechę i jej wartość do podziału, a następnie analizuje lewy i prawy węzeł potomny wynikający z tego podziału i oblicza następujące wielkości:

- Średnia wartość wszystkich etykiet punktów danych szkoleniowych w lewym węźle. Ta średnia będzie wartością lewego węzła y_{lewy} i jest wartością prognozowaną przez drzewo decyzyjne w przypadku, gdy ten węzeł zostanie węzłem liścia.
- Średnia wartość wszystkich etykiet punktów danych szkoleniowych w prawym węźle. Ta średnia będzie wartością prawego węzła y_{prawy}. Podobnie jak w przypadku lewego węzła, jest to wartość prognozowana przez drzewo decyzyjne w przypadku, gdy ten węzeł stanie się węzłem liścia.
- Suma kwadratów odległości między wartością lewego węzła a rzeczywistą etykietą każdego punktu danych w lewym węźle: $\sum_{PunktyDanychLewegoWęzła} \left| y^i_{\text{rzeczywista}} - y_{\text{lewy}} \right|^2$.
- Suma kwadratów odległości między wartością prawego węzła a rzeczywistą etykietą każdego egzemplarza w prawym węźle: $\sum_{PunktyDanychPrawegoWęzła} \left| y^i_{\text{rzeczywista}} - y_{\text{lewy}} \right|^2$.
- Średnia ważona wspomnianych dwóch sum, gdzie dla każdego z węzłów jest stosowana waga względem rozmiaru w stosunku do węzła nadrzędnego, podobnie jak w przypadku nieczystości Giniego:

$$\frac{n_{\text{lewy}}}{n}\sum_{PunktyDanychLewegoWęzła}\left|y^{i}_{\text{rzeczywista}}-y_{\text{lewy}}\right|^{2}+\frac{n_{\text{prawy}}}{n}\sum_{PunktyDanychPrawegoWęzła}\left|y^{i}_{\text{rzeczywista}}-y_{\text{prawy}}\right|^{2}\Bigg|\ y^{i}_{\text{rzeczywista}}\Bigg)$$

Algorytm ten jest **zachłanny** i kosztowny obliczeniowo, w tym sensie, *że musi to zrobić dla każdej cechy i każdej możliwej wartości podziału cechy*, a następnie wybrać cechę i podział cechy, które zapewniają najmniejszą średnią ważonego błędu kwadratowego między lewym i prawym węzłem potomnym.

Pakiety oprogramowania, w tym biblioteka `scikit-learn` Pythona, której używamy w notatnikach Jupyter uzupełniających tę książkę, stosują słynny algorytm CART (ang. *Classification and Regression Tree*). Ten algorytm tworzy drzewa zawierające węzły złożone tylko z dwóch potomków (drzewa binarne), gdzie test w każdym węźle zwraca tylko dwie odpowiedzi: *Tak* lub *Nie*. Inne algorytmy, np. ID3, pozwalają tworzyć drzewa z węzłami zawierającymi dwa bądź większą liczbę węzłów potomnych.

Wady drzew decyzyjnych

Drzewa decyzyjne są bardzo łatwe w interpretacji i są popularne z wielu istotnych powodów: dostosowują się do dużych zbiorów danych, różnych typów danych (cechy dyskretne i ciągłe, nie ma potrzeby skalowania danych) i pozwalają wykonywać zarówno zadania regresji, jak i klasyfikacji. Mogą jednak być niestabilne, w tym sensie, że dodanie zaledwie jednego punktu do zbioru danych może zmienić korzeń drzewa, a tym samym mogą skutkować zupełnie innym drzewem decyzyjnym. Są one również wrażliwe na rotacje w danych, ponieważ ich granice decyzyjne są zwykle poziome i pionowe (a nie ukośne, jak w przypadku maszyn wektorów nośnych). Dzieje się tak dlatego, że podziały zwykle występują przy określonych wartościach cech, zatem granice decyzyjne przebiegają równolegle do osi cech. Jedna z poprawek polega na przekształceniu zestawu danych przy użyciu przedstawionej w rozdziale 6. *metody dekompozycji pojedynczej wartości* w celu dopasowania go do *głównych osi*. Drzewa decyzyjne mają tendencję do nadmiernego dopasowywania się do danych, więc trzeba je przycinać. Do tego celu zazwyczaj są stosowane testy statystyczne. Ze względu na wykorzystanie do konstrukcji drzew, w których wyszukiwanie odbywa się na podstawie wszystkich cech i ich wartości, algorytmów zachłannych, takie algorytmy są kosztowne obliczeniowo i mniej dokładne. Omówione poniżej lasy losowe pozwalają rozwiązać problemy związane z niektórymi spośród tych niedociągnięć.

Lasy losowe

Kiedy po raz pierwszy dowiedziałam się o drzewach decyzyjnych, najbardziej kłopotliwe były dla mnie następujące aspekty:

- Czym zainicjować drzewo, czyli jak zdecydować, która cecha danych ma być cechą główną?
- Przy jakiej konkretnej wartości cechy decydujemy się podzielić węzeł?

- Kiedy należy przestać?
- Jak bardzo problematyczny, ogólnie rzecz biorąc, był rozwój drzewa?

(Zauważ, że na niektóre z tych pytań odpowiedziałam w poprzednim podpunkcie). Sprawy nie ułatwiało surfowanie po internecie w poszukiwaniu odpowiedzi, gdzie jedynie spotykałam deklaracje, że drzewa decyzyjne są łatwe do zbudowania i zrozumienia. Czułam się więc, jakbym była jedyną osobą głęboko zdezorientowaną w kwestii drzew decyzyjnych.

Moje zdziwienie natychmiast zniknęło, gdy dowiedziałam się o *lasach losowych*. Niesamowitą rzeczą w lasach losowych jest możliwość uzyskania niewiarygodnie dobrych wyników regresji lub klasyfikacji *bez* odpowiadania na żadne z nurtujących mnie pytań. Dzięki randomizacji całego procesu możliwe jest zbudowanie wielu drzew decyzyjnych, a jednocześnie na wszystkie moje pytania można odpowiedzieć dwoma słowami: *wybieraj losowo*. Zespołowa agregacja prognoz za pomocą lasów losowych daje bardzo dobre wyniki, nawet lepsze niż jedno starannie opracowane drzewo decyzyjne. Można się spotkać z poglądem, że *randomizacja często daje niezawodność*!

Inną bardzo przydatną właściwością lasów losowych jest generowana przez nie miara *ważności* cech, pomagająca określić te cechy, które znacząco wpływają na prognozy, a także pomagają w wyborze cech.

Klasteryzacja k-średnich

Jednym z powszechnych celów analityków danych jest podział danych na klastry, z których każdy podkreśla pewne wspólne cechy. Klasteryzacja *k*-średnich jest powszechną metodą uczenia maszynowego, która dzieli *n* punktów danych (wektorów) na *k* klastrów, przy czym każdy punkt danych jest przypisywany do klastra o najbliższej wartości średniej. Średnia każdego klastra lub jego centroid spełnia rolę prototypu klastra. Ogólnie rzecz biorąc, grupowanie *k*-średnich minimalizuje wariancję (kwadrat odległości euklidesowej do średniej) w obrębie każdego klastra. Najpopularniejszy algorytm *k*-średnich to algorytm iteracyjny:

1. Rozpocznij od początkowego zestawu *k* średnich. Oznacza to określenie liczby klastrów z góry, co rodzi następujące pytanie: Jak go zainicjować? W jaki sposób należy wybrać lokalizacje pierwszych *k* centroidów? Odpowiedzi na te pytania można znaleźć w literaturze.
2. Przypisz każdy punkt danych do klastra o najbliższej średniej w kategoriach kwadratu odległości euklidesowej.
3. Ponownie oblicz średnie dla każdego klastra.

Algorytm wykazuje zbieżność, gdy przypisania punktów danych do każdego z klastrów się nie zmieniają.

Miary wydajności dla modeli klasyfikacji

Opracowanie modeli matematycznych, które coś obliczają i generują wyniki, jest stosunkowo łatwe. Opracowanie modeli, które dobrze się sprawdzają w pożądanych przez nas zadaniach, to zupełnie inna historia. Co więcej, modele, które osiągają dobre wyniki według niektórych

wskaźników, według innych wskaźników zachowują się źle. Podczas opracowywania wskaźników wydajności i podejmowania decyzji, na których z nich polegać w zależności od konkretnych przypadków użycia, trzeba zachować szczególną ostrożność.

Mierzenie wydajności modeli prognozujących wartości liczbowe, takich jak modele regresji, jest łatwiejsze niż modeli klasyfikacji, ponieważ istnieje wiele sposobów obliczania odległości między liczbami (prognozy dobre i złe). Z drugiej strony, gdy zadaniem jest klasyfikacja (do której można użyć takich modeli jak regresja logistyczna, regresja softmax, maszyny wektorów nośnych, drzewa decyzyjne, lasy losowe lub sieci neuronowe), ocena wydajności wymaga nieco więcej uwagi. Co więcej, zwykle potrzebne są pewne kompromisy. Na przykład, jeśli zadanie polega na klasyfikowaniu filmów z YouTube'a jako bezpiecznych dla dzieci (wynik dodatni) lub niebezpiecznych dla dzieci (wynik ujemny), to czy należy dostosować model w taki sposób, aby zmniejszyć liczbę wyników fałszywie dodatnich, czy trzeba raczej dążyć do zmniejszenia liczby wyników fałszywie ujemnych? Oczywiście bardziej problematyczne jest sklasyfikowanie filmu jako bezpiecznego, podczas gdy w rzeczywistości jest on niebezpieczny (wynik fałszywie dodatni), niż odwrotnie, więc wybrany wskaźnik wydajności powinien to odzwierciedlać.

Poniżej zestawiłam miary wydajności powszechnie stosowane dla modeli klasyfikacji. Nie staraj się zapamiętywać ich nazw, ponieważ nie mają one logicznego sensu. Zamiast tego lepiej poświęcić czas na zrozumienie ich znaczenia:

Dokładność

Odsetek przypadków, w których model predykcyjny wygenerował poprawną klasyfikację:

$$\text{Dokładność} = \frac{\text{prawdziwie dodatnie} + \text{prawdziwie ujemne}}{\text{wszystkie prognozowane wyniki dodatnie} + \text{wszystkie prognozowane wyniki ujemne}}$$

Macierz pomyłek (ang. *confusion matrix*)

Zliczanie wszystkich wyników prawdziwie dodatnich, fałszywie dodatnich, prawdziwie ujemnych i fałszywie ujemnych:

Prawdziwie ujemne	Fałszywie dodatnie
Fałszywie ujemne	Prawdziwie dodatnie

Współczynnik precyzji

Dokładność prognoz dodatnich:

$$\text{Precyzja} = \frac{\text{prawdziwie dodatnie}}{\text{wszystkie prognozowane wyniki dodatnie}} = \frac{\text{prawdziwie dodatnie}}{\text{prawdziwie dodatnie} + \text{fałszywie dodatnie}}$$

Współczynnik czułości (ang. *recall score*)

Stosunek dodatnich punktów danych, które zostały poprawnie sklasyfikowane:

$$\text{Czułość} = \frac{\text{prawdziwie dodatnie}}{\text{wszystkie etykiety dodatnie}} = \frac{\text{prawdziwie dodatnie}}{\text{prawdziwie dodatnie} + \text{fałszywie ujemne}}$$

Specyficzność

Stosunek ujemnych punktów danych, które zostały poprawnie sklasyfikowane:

$$\text{Specyficzność} = \frac{\text{prawdziwie ujemne}}{\text{wszystkie etykiety ujemne}} = \frac{\text{prawdziwie ujemne}}{\text{prawdziwie ujemne + fałszywie dodatnie}}$$

Współczynnik F_1

Wielkość, która jest wysoka tylko wtedy, gdy zarówno współczynnik precyzji, jak i czułości są wysokie:

$$F_1 = \frac{2}{\frac{1}{\text{precyzja}} + \frac{1}{\text{czułość}}}$$

Krzywe AUC (ang. *area under the curve* — dosłownie: *obszar pod krzywą*) i *ROC* (ang. *receiver operating characteristics* — dosłownie: *charakterystyka operacyjna odbiornika*)

Wspomniane krzywe zapewniają miarę wydajności modelu klasyfikacji przy różnych wartościach progowych. Można wykorzystać te krzywa do zmierzenia, jak dobrze określona zmienna prognozuje określony wynik; na przykład jak dobrze wynik testu przedmiotowego GRE prognozuje zdanie egzaminu kwalifikacyjnego do szkoły wyższej za pierwszym razem?

Doskonały przewodnik po najlepszych praktykach w zakresie pomiarów wydajności stanowi książka Andrew Nga *Machine Learning Yearning* (*https://github.com/ajaymache/machine-learning-yearning*, wydana własnym nakładem). Przeczytaj uważnie, zanim zagłębisz się rzeczywiste aplikacje AI. Receptury zawarte w tej książce opierają się na wielu próbach, sukcesach i porażkach.

Podsumowanie i perspektywy na przyszłość

W tym rozdziale przeanalizowałam wybrane najpopularniejsze modele uczenia maszynowego z naciskiem na konkretne struktury matematyczne, które występują w całej książce: funkcję szkoleniową, funkcję straty i optymalizację. Omówiłam regresję liniową, logistyczną i softmax, a następnie poświęciłam nieco miejsca maszynom wektorów nośnych, drzewom decyzyjnym, metodom zespołowym i lasom losowym.

Ponadto przedstawiłam przyzwoite argumenty przemawiające za studiowaniem następujących matematycznych pojęć:

Rachunek różniczkowy

Minima i maksima występują na granicy lub w punktach, w których jedna z pochodnych przyjmuje wartość zero lub nie istnieje.

Algebra liniowa

- Liniowe łączenie cech: $\omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n$.
- Zapisywanie różnych wyrażeń matematycznych przy użyciu notacji macierzowej i wektorowej.
- Iloczyn skalarny dwóch wektorów $\vec{a}^t \vec{b}$.

- Norma l^2 wektora.
- Unikaj korzystania ze źle uwarunkowanych macierzy. Pozbądź się cech zależnych liniowo. Ma to również związek z wyborem cech.
- Unikaj mnożenia macierzy przez siebie; takie działanie jest zbyt kosztowne. Zamiast tego mnóż macierze przez wektory.

Optymalizacja

- W przypadku funkcji wypukłych nie trzeba się martwić o utknięcie w minimach lokalnych, ponieważ minima lokalne są jednocześnie minimami globalnymi. Trzeba się jednak martwić wąskimi dolinami (patrz rozdział 4.).
- W metodach gradientowych potrzebna jest tylko jedna pochodna (patrz rozdział 4.).
- W metodach Newtona trzeba stosować dwie pierwsze pochodne lub ich przybliżenie (co w przypadku dużych zbiorów danych jest niewygodne).
- Programowanie kwadratowe, problem wtórny i zstępowanie współrzędnych (wszystkie występują w maszynach wektorów nośnych).

Statystyki

- Macierz korelacji i wykresy punktowe.
- Test F i test informacji wzajemnej dla selekcji cech.
- Standaryzacja cech danych (odjęcie średniej i podzielenie przez odchylenie standardowe).

Dodatkowe kroki, których nie wykonaliśmy i których nie opisałam (jeszcze):

- Walidacja modeli — dostrojenie wartości wag i hiperparametrów w celu uniknięcia nadmiernego dopasowania.
- Testowanie przeszkolonego modelu na testowym podzbiorze danych, którego model nie używał (ani nie zaobserwował) podczas etapów uczenia i walidacji (robimy to w dołączonym notatniku Jupyter).
- Wdrażanie i monitorowanie gotowego modelu.
- Nigdy nie należy przestawać myśleć o sposobach ulepszania modeli i ich lepszej integracji z całym procesem produkcyjnym.

W rozdziale 4. wkroczymy w nową i ekscytującą erę sieci neuronowych.

Skorowidz

Ł

M

N

O

R

S

Ś

T

U

W

Z

Ta książka w zachwycający sposób sprawia, że matematyka staje się zabawą dla licznych uczestników przyszłości opartej na sztucznej inteligencji!

Adri Purkayastha, analityk oceny ryzyka, BNP Paribas

Sztuczna inteligencja i technologie oparte na danych są coraz częściej integrowane z istniejącymi systemami i operacjami. Ta tendencja dotyczy licznych branż. Dziś przy budowaniu systemów SI można korzystać z gotowych bibliotek, jeżeli jednak zależy Ci na w pełni świadomym tworzeniu doskonalszych aplikacji, musisz dobrze opanować matematykę leżącą u podstaw sztucznej inteligencji.

Nawet jeśli nie darzysz królowej nauk płomiennym uczuciem, dzięki temu kompleksowemu opracowaniu z łatwością poradzisz sobie z jej lepszym poznaniem. Nie znajdziesz tu skomplikowanych teorii naukowych, tylko przystępnie podane koncepcje matematyczne niezbędne do rozwoju w dziedzinie sztucznej inteligencji, w szczególności do praktycznego stosowania najnowocześniejszych modeli. Poznasz takie zagadnienia jak regresja, sieci neuronowe, sieci konwolucyjne, optymalizacja, prawdopodobieństwo, procesy Markowa, równania różniczkowe i wiele innych w ekskluzywnym kontekście sztucznej inteligencji. Książkę docenią pasjonaci nowych technologii, twórcy aplikacji, inżynierowie i analitycy danych, a także matematycy i naukowcy.

W książce:

- wyjaśnienie pojęć z zakresu uczenia maszynowego, inżynierii danych i matematyki
- ujednolicanie modeli w ramach jednej struktury matematycznej
- grafy i dane sieciowe
- eksploracja rzeczywistych danych, zmniejszanie liczby wymiarów i przetwarzanie obrazów
- korzystanie z modeli w różnych projektach opartych na danych
- implikacje i ograniczenia sztucznej inteligencji

Dr Hala Nelson wykłada matematykę na James Madison University. Specjalizuje się w modelowaniu matematycznym i konsultacjach dla sektora publicznego, w szczególności służb ratunkowych. Dorastała w Libanie podczas brutalnej wojny domowej, to ukształtowało jej zainteresowanie ludzkim zachowaniem, naturą inteligencji i sztuczną inteligencją.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1445-2
9 788328 914452
Cena: 129,00 zł