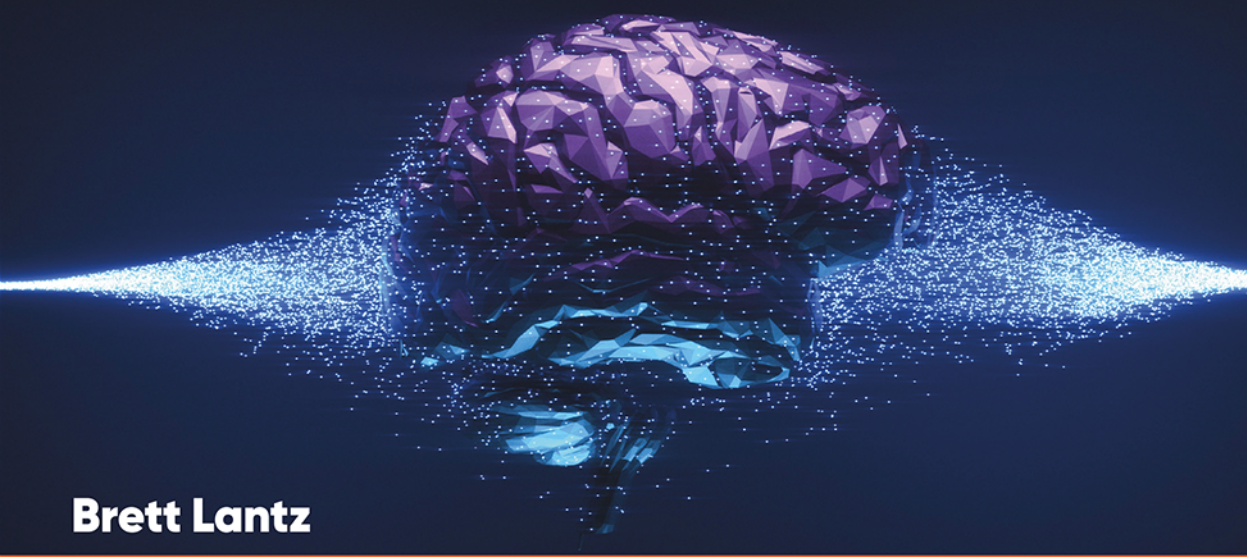
OKIEM EKSPERTA

Uczenie maszynowe w języku R

Tworzenie i doskonalenie modeli – od przygotowania danych po dostrajanie, ewaluację i pracę z big data

Wydanie IV



Brett Lantz

Helion

packt

Tytuł oryginału: Machine Learning with R: Learn techniques for building and improving machine learning models, from data preparation to model tuning, evaluation, and working with big data, 4th Edition

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-0899-4

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/uczjr4
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

Uczenie leniwe — klasyfikacja metodą najbliższych sąsiadów

W miastach na całym świecie pojawił się ciekawy rodzaj doznania kulinarnego. Goście są obsługiwani w całkowicie zaciemnionej restauracji przez kelnerów, którzy poruszają się po zapamiętanych trasach, posługując się wyłącznie zmysłami dotyku i dźwięku. Urok tych lokali tkwi w przekonaniu, że odcięcie się od wrażeń wzrokowych wyostrza smak i węch, co pozwala doświadczać potraw w nowy sposób. Każdy kęs budzi zachwyt podczas odkrywania smakowych kompozycji przygotowanych przez szefa kuchni.

Czy potrafisz sobie wyobrazić, jak goście doświadczają niewidocznego jedzenia? Już pierwszy kęs przytłacza ich zmysły. Jakie smaki dominują? Czy potrawa jest pikantna, czy słodka? Czy smakuje jak coś, co jedli wcześniej? Osobiście wyobrażam sobie ten proces w kategoriach nieco zmodyfikowanego powiedzenia — jeśli coś pachnie jak kaczka i smakuje jak kaczka, to prawdopodobnie jest kaczką.

Ilustruje to ideę, którą można wykorzystać w uczeniu maszynowym, podobnie jak inna angielska maksyma związana z drobiem — „ptaki o takim samym upierzeniu trzymają się razem", czyli „ciągnie swój do swego". Ujmując to inaczej, rzeczy, które są podobne, zwykle mają podobne właściwości. W uczeniu maszynowym używa się tej zasady do klasyfikowania danych poprzez umieszczanie ich w tej samej kategorii, do której zaliczają się podobni lub „najbliżsi" sąsiedzi. Niniejszy rozdział jest poświęcony klasyfikatorom, które wykorzystują to podejście. Poznasz w nim:

- kluczowe koncepcje, które definiują klasyfikatory oparte na sąsiedztwie i sprawiają, że uważa się je za „leniwych" uczniów;
- metody pomiaru podobieństwa dwóch przykładów z wykorzystaniem odległości;
- popularny klasyfikator oparty na sąsiedztwie i nazywany k-NN.

Jeśli całe to gadanie o jedzeniu sprawiło, że zgłodniałeś, naszym pierwszym zadaniem będzie zrozumienie podejścia k-NN poprzez wykorzystanie go do rozstrzygnięcia wieloletniego sporu kulinarnego.

Klasyfikacja metodą najbliższych sąsiadów

Najprościej mówiąc, klasyfikatory wykorzystujące metodę **najbliższych sąsiadów** klasyfikują nieznane przykłady w ten sposób, że przypisują im klasę podobnych znanych przykładów. Przypomina to posiłek opisany we wstępie do tego rozdziału, podczas którego goście identyfikują nowe potrawy poprzez porównywanie ich z tymi, które jedli poprzednio. W klasyfikacji metodą najbliższych sąsiadów komputery wykorzystują „ludzką" zdolność przywoływania przeszłych doświadczeń, aby wnioskować na temat bieżących okoliczności. Pomimo prostoty tego pomysłu metoda najbliższych sąsiadów okazuje się imponująco skuteczna. Stosowano ją z sukcesem w:

- Wizji komputerowej, m.in. optycznym rozpoznawaniu znaków (OCR) i rozpoznawaniu twarzy na zdjęciach i filmach wideo.
- Systemach rekomendacji, które przewidują, czy komuś spodoba się pewien film albo piosenka.
- Identyfikowaniu wzorców w danych genetycznych w celu wykrywania konkretnych białek lub chorób.

Ogólnie rzecz biorąc, klasyfikatory oparte na sąsiedztwie dobrze nadają się do zadań klasyfikacyjnych, w których relacje między cechami a docelowymi klasami są liczne, skomplikowane albo z innych powodów bardzo trudne do zrozumienia, ale obiekty podobnego typu są dość homogeniczne. Innymi słowy, jeśli jakieś pojęcie jest trudne do zdefiniowania, ale łatwo rozpoznać je na przykładzie, to metoda najbliższych sąsiadów prawdopodobnie będzie odpowiednia. Z drugiej strony, jeśli dane są „zaszumione" i nie istnieje klarowne rozróżnienie między grupami, algorytm najbliższych sąsiadów może mieć kłopoty z identyfikowaniem granic klas.

Algorytm k-NN

Przykładem klasyfikacji opartej na sąsiedztwie jest algorytm ***k* najbliższych sąsiadów** (ang. *k-nearest neighbors*, k-NN). Choć jest to prawdopodobnie jeden z najprostszych algorytmów uczenia maszynowego, to wciąż cieszy się dużą popularnością. Ma on następujące zalety i wady:

Zalety	Wady
■ Prosty i efektywny ■ Nie wymaga żadnych założeń odnośnie do dystrybucji danych ■ Szybka faza treningu	■ Nie tworzy modelu, ograniczając możliwość zrozumienia związku cech z klasą ■ Wymaga wyboru odpowiedniej wartości *k* ■ Powolna faza klasyfikacji ■ Cechy nominalne i brakujące dane wymagają dodatkowego przetwarzania

Nazwa k-NN bierze się stąd, że algorytm wykorzystuje informacje o *k* najbliższych sąsiadach do klasyfikowania nieoznaczonych przykładów. Litera *k* jest zmienną, co sugeruje, że można użyć dowolnej liczby najbliższych sąsiadów. Po wybraniu *k* algorytm wymaga treningowego zbioru danych złożonego z przykładów, które zostały podzielone na kilka kategorii oznaczonych zmienną nominalną. Następnie dla każdego nieoznaczonego rekordu w testowym zbiorze danych k-NN identyfikuje *k* rekordów w danych treningowych, które są „najbliżej" tego rekordu pod względem podobieństwa. Nieoznaczonemu egzemplarzowi testowemu przypisuje się następnie klasę, którą ma większość spośród *k* najbliższych sąsiadów.

Aby zilustrować ten proces, wróćmy do smakowania na ślepo opisanego na początku rozdziału. Przypuśćmy, że przed zjedzeniem tajemniczej potrawy utworzyliśmy zbiór danych i zapisaliśmy w nim wrażenia dotyczące pewnego zbioru składników, których spróbowaliśmy poprzednio. Dla prostoty oceniliśmy tylko dwie cechy każdego składnika. Pierwsza mierzy w skali od 1 do 10, jak chrupiący jest składnik, a druga mierzy w skali od 1 do 10, jak jest słodki. Następnie oznaczyliśmy każdy składnik jako jeden z trzech typów żywności: owoce, warzywa lub białka, ignorując inne rodzaje żywności, takie jak ziarna i tłuszcze.

Pierwszych kilka wierszy takiego zbioru danych mogłoby mieć następującą strukturę:

Składnik	Słodkość	Chrupkość	Typ żywności
Jabłko	10	9	Owoc
Bekon	1	4	Białko
Banan	10	1	Owoc
Marchewka	7	10	Warzywo
Seler	3	10	Warzywo

Algorytm k-NN traktuje cechy jako współrzędne w wielowymiarowej przestrzeni cech, czyli przestrzeni, która obejmuje wszystkie możliwe kombinacje wartości cech.

Ponieważ zbiór danych o składnikach ma tylko dwie cechy, jego przestrzeń cech jest dwuwymiarowa. Dane dwuwymiarowe możemy przedstawić na wykresie punktowym, na którym oś *x* wskazuje słodkość składnika, a oś *y* jego chrupkość. Po dodaniu kilku innych składników do zbioru danych nasz wykres punktowy mógłby wyglądać tak jak na rysunku 3.1.

Rysunek 3.1. Wykres punktowy słodkości i chrupkości wybranych produktów

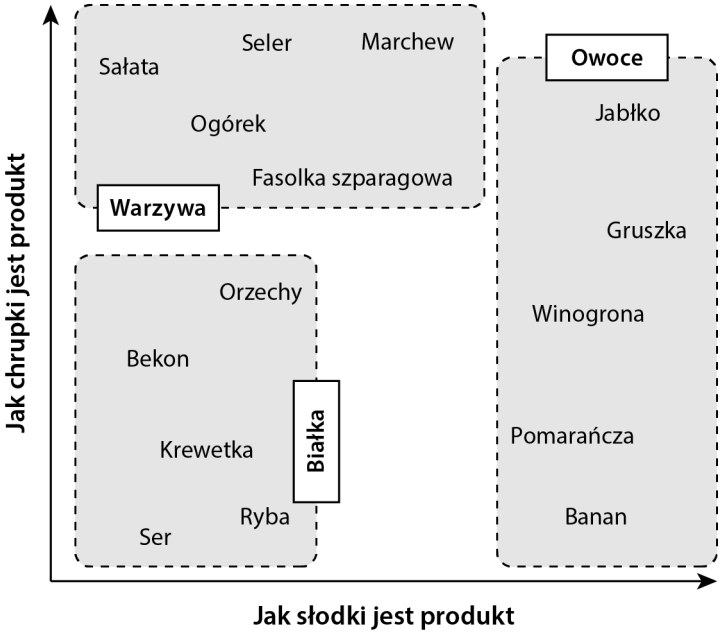
Czy dostrzegasz pewien wzór? Podobne typy żywności są dość ściśle zgrupowane. Jak pokazano na rysunku 3.2, warzywa zwykle są chrupiące, ale nie słodkie; owoce zwykle są słodkie i albo chrupiące, albo nie; a białka nie są ani chrupiące, ani słodkie.

Przypuśćmy, że po skonstruowaniu tego zbioru danych postanawiamy wykorzystać go, aby rozstrzygnąć odwieczne pytanie: czy pomidor jest owocem, czy warzywem? Żeby ustalić najlepiej pasującą klasę, możemy użyć metody najbliższych sąsiadów, jak pokazano na rysunku 3.3.

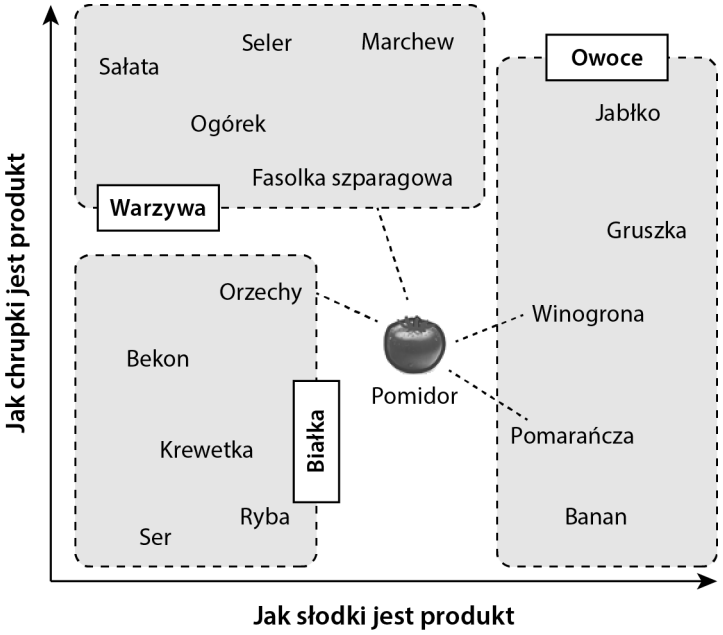
Mierzenie podobieństwa za pomocą odległości

Lokalizowanie najbliższych sąsiadów wymaga **funkcji odległości**, czyli wzoru na obliczanie podobieństwa między dwoma przykładami.

Istnieje wiele sposobów obliczania odległości. Wybór funkcji odległości może znacząco wpłynąć na działanie modelu, choć trudno jest stwierdzić, której należy użyć, inaczej



Rysunek 3.2. Podobnie klasyfikowane rodzaje żywności zwykle mają podobne atrybuty



Rysunek 3.3. Najbliżsi sąsiedzi pomidora wskazują, czy jest on owocem, czy warzywem

niż poprzez bezpośrednie porównanie ich w danym zadaniu. Tradycyjnie algorytm k-NN używa **odległości euklidesowej**, czyli odległości, którą byśmy zmierzyli, gdybyśmy mogli użyć linijki do połączenia dwóch punktów. Odległość euklidesową mierzy się „lotem ptaka", co sugeruje najkrótszą bezpośrednią trasę. Na rysunku 3.3 zilustrowano to kropkowanymi liniami, które łączą pomidor z jego sąsiadami.

Wskazówka

Inną popularną miarą odległości jest **odległość manhattańska** oparta na trasie, którą przeszedłby pieszy idący między budynkami. Jeśli chciałbyś dowiedzieć się więcej o innych miarach odległości, przeczytaj dokumentację funkcji odległości w języku R poprzez wydanie polecenia `?dist`.

Odległość euklidesowa jest określona poniższym wzorem, w którym p i q to przykłady do porównania, każdy mający n cech. Wyraz p_1 odnosi się do wartości pierwszej cechy przykładu p, a wyraz q_1 — do wartości pierwszej cechy przykładu q:

$$\text{dist}(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Wzór na odległość wiąże się z porównywaniem wartości cech każdego przykładu. Aby na przykład obliczyć odległość między pomidorem (słodkość = 6, chrupkość = 3) a fasolką szparagową (słodkość = 3, chrupkość = 7), użylibyśmy wzoru w następujący sposób:

$$\text{dist}(pomidor, fasolka) = \sqrt{(6-3)^2 + (4-7)^2} = 4{,}2$$

W podobny sposób możemy obliczyć odległość między pomidorem a kilkoma jego najbliższymi sąsiadami:

Składnik	Słodkość	Chrupkość	Typ żywności	Odległość od pomidora
Winogrona	8	5	Owoc	`sqrt((6 - 8)^2 + (4 - 5)^2) = 2,2`
Fasolka	3	7	Warzywa	`sqrt((6 - 3)^2 + (4 - 7)^2) = 4,2`
Orzechy	3	6	Białka	`sqrt((6 - 3)^2 + (4 - 6)^2) = 3,6`
Pomarańcza	7	3	Owoc	`sqrt((6 - 7)^2 + (4 - 3)^2) = 1,4`

Aby sklasyfikować pomidor jako warzywo, białko lub owoc, zaczniemy od przypisania mu tego samego typu żywności, który ma jego jeden najbliższy sąsiad. Nazywa się to klasyfikacją 1-NN, ponieważ $k = 1$. Najbliższym sąsiadem pomidora jest pomarańcza z odległością 1,4. Ponieważ pomarańcza jest owocem, algorytm 1-NN sklasyfikowałby pomidor jako owoc.

Jeśli zamiast tego użyjemy algorytmu k-NN z parametrem *k* = 3, przeprowadzi on „głosowanie" między trzema najbliższymi sąsiadami: pomarańczą, winogronem i orzechami. Ponieważ wśród tych sąsiadów klasą większościową jest owoc (dwa spośród trzech głosów), pomidor zostanie ponownie sklasyfikowany jako owoc.

Wybieranie odpowiedniej wartości k

Od wyboru liczby sąsiadów w algorytmie k-NN zależy to, jak dobrze model będzie generalizować się na przyszłe dane. Równowaga między nadmiernym a niedostatecznym dopasowaniem do danych treningowych jest znana jako kompromis między obciążeniem a wariancją (ang. *bias-variance tradeoff*). Wybór dużej wartości *k* zmniejsza wpływ wariancji spowodowanej zaszumionymi danymi, ale może obciążyć model tak, że będzie on ignorował subtelne, ale ważne wzorce.

Przypuśćmy, że posuwamy się do skrajności i ustawiamy bardzo dużą wartość *k*, równą liczbie obserwacji w danych treningowych. W przypadku każdego przykładu treningowego reprezentowanego w ostatecznym głosowaniu najczęściej występująca klasa zawsze będzie zdobywać większość głosów. W rezultacie model zawsze będzie przewidywał klasę większościową, bez względu na najbliższych sąsiadów.

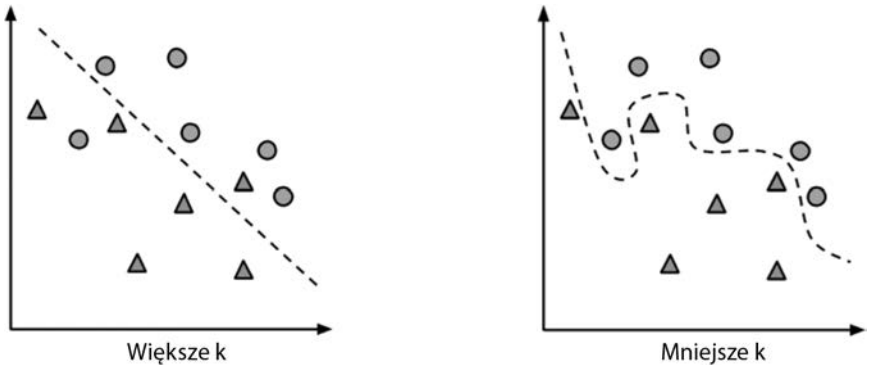
Przeciwna skrajność, czyli używanie tylko jednego najbliższego sąsiada, sprawia, że zaszumione dane i wartości odstające nadmiernie wpływają na klasyfikację przykładów. Przypuśćmy na przykład, że niektóre przykłady treningowe zostały oznaczone błędnymi etykietami. Każdy nieoznaczony etykietą przykład, który znajdzie się najbliżej błędnie oznaczonego sąsiada, zostanie przypisany do niewłaściwej klasy, nawet jeśli dziewięciu innych najbliższych sąsiadów zagłosowałoby inaczej.

Oczywiście najlepsza wartość *k* leży gdzieś między tymi dwiema skrajnościami.

Rysunek 3.4 ilustruje ogólnie, jak większe lub mniejsze wartości *k* wpływają na granicę decyzyjną (oznaczoną kreskowaną linią). Mniejsze wartości prowadzą do bardziej skomplikowanych granic decyzyjnych, które dokładniej pasują do danych treningowych. Problem w tym, że nie wiemy, czy to prosta, czy też zakrzywiona granica lepiej reprezentuje rzeczywistą podstawową koncepcję, której ma się nauczyć model.

W praktyce wybór *k* zależy od stopnia trudności koncepcji, której ma się nauczyć model, oraz liczby rekordów w zbiorze treningowym. Często zaczyna się od wartości *k* równej pierwiastkowi kwadratowemu z liczby przykładów treningowych. W opracowanym poprzednio klasyfikatorze produktów spożywczych moglibyśmy ustawić *k* = 4, ponieważ w danych treningowych było 15 przykładowych składników, a pierwiastek kwadratowy z 15 wynosi 3,87.

Jednak takie reguły nie zawsze dają najlepszą wartość *k*. Alternatywnym podejściem jest przetestowanie kilku wartości *k* na różnych testowych zbiorach danych i wybranie



Rysunek 3.4. Większa wartość k ma wyższe obciążenie i mniejszą wariancję, niż ma je mniejsza wartość k

tej, która zapewnia największą trafność klasyfikacji. Z drugiej strony, jeśli dane nie są bardzo zaszumione, duży zbiór treningowy może sprawić, że wybór wartości *k* będzie mniej istotny. Wynika to stąd, że nawet subtelne koncepcje będą miały wystarczająco dużą pulę przykładów, które będą głosować jako najbliżsi sąsiedzi.

Wskazówka

Mniej typowym, ale interesującym rozwiązaniem tego problemu jest wybranie większej wartości *k* i używanie ważonego głosowania, w którym głos bliższych sąsiadów ma większe znaczenie niż dalszych. Niektóre implementacje algorytmu k-NN oferują tę opcję.

Przygotowywanie danych do użytku w algorytmie k-NN

Przed zastosowaniem algorytmu k-NN zwykle normalizuje się cechy do standardowego zakresu. Powodem jest to, że wzór na odległość jest bardzo wrażliwy na sposób pomiaru cech. Zwłaszcza gdy niektóre cechy mają znacznie większy zakres wartości niż inne, pomiar odległości zostanie zdominowany przez cechy o większych zakresach. Nie było to problemem w przykładzie z próbowaniem jedzenia, ponieważ zarówno słodkość, jak i chrupkość były mierzone w skali od 1 do 10.

Przypuśćmy jednak, że dodajemy kolejną cechę, która reprezentuje ostrość jedzenia mierzoną w skali Scoville'a. Jest to standardowa miara pikantności, która rozciąga się od zera (zupełnie nieostre) do ponad miliona (najostrzejsze papryki chili). Ponieważ różnica między jedzeniem ostrym a nieostrym może wynosić ponad milion, a między jedzeniem słodkim a niesłodkim oraz chrupkim i niechrupkim wynosi najwyżej 10, stopień ostrości wpływa na funkcję odległości znacznie bardziej niż pozostałe dwa czynniki. Jeśli nie wyregulujemy danych, możemy odkryć, że funkcja odległości rozróżnia produkty spożywcze wyłącznie na podstawie ostrości; wpływ chrupkości i słodkości zostanie zdominowany przez wpływ pikantności.

Rozwiązaniem jest przeskalowanie cech poprzez zwężenie lub rozszerzenie ich zakresów w taki sposób, żeby każda miała względnie równy wkład w funkcję odległości. Jeśli na przykład słodkość i chrupkość są mierzone w skali od 1 do 10, chcielibyśmy, żeby ostrość również była mierzona w skali od 1 do 10. Takie skalowanie można osiągnąć kilkoma sposobami.

Tradycyjną metodą skalowania cech na użytek algorytmu k-NN jest **normalizacja min-max**. Proces ten przekształca cechę tak, że wszystkie jej wartości przypadają na zakres od 0 do 1. Wzór na normalizację cechy jest następujący:

$$X_{nowe} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

W celu przekształcenia każdej wartości cechy *X* formuła odejmuje od niej minimalną wartość *X* i dzieli różnicę przez zakres *X*. Wynikowe znormalizowane wartości cechy wskazują, jak daleko, od 0 do 100%, pierwotna wartość była usytuowana w zakresie od pierwotnego minimum do maksimum.

Inną typową transformacją jest **standaryzacja Z**. Poniższy wzór odejmuje średnią wartość cechy X i dzieli różnicę przez odchylenie standardowe X:

$$X_{nowe} = \frac{X - \mu}{\sigma} = \frac{X - \text{Średnia}(X)}{\text{OdchStd}(X)}$$

Ta formuła, oparta na właściwościach rozkładu normalnego opisanych w rozdziale 2. „Zarządzanie danymi", przeskalowuje każdą wartość cechy w zależności od tego, o ile odchyleń standardowych jest ona oddalona od średniej. Uzyskaną w ten sposób wartość nazywa się wynikiem z (ang. *z-score*). Wyniki z mieszczą się w nieograniczonym zakresie liczb ujemnych i dodatnich. W przeciwieństwie do wartości znormalizowanych nie mają wstępnie określonego minimum ani maksimum.

Wskazówka

Ta sama metoda skalowania, którą zastosowano na treningowym zbiorze danych k-NN, musi zostać zastosowana również do przykładów testowych, które algorytm będzie później klasyfikował. W przypadku normalizacji min-max może to prowadzić do problematycznych sytuacji, ponieważ minimalne lub maksymalne wartości przyszłych przykładów mogą wykraczać poza zakres wartości zaobserwowanych w danych treningowych. Jeśli znasz teoretyczne minima lub maksima, możesz użyć tych stałych zamiast zaobserwowanych wartości minimalnych i maksymalnych. Możesz też użyć standaryzacji Z przy założeniu, że rozkład przyszłych przykładów będzie miał tę samą średnią i odchylenie standardowe, co przykłady treningowe.

Wzór na odległość euklidesową jest niezdefiniowany dla danych nominalnych. Aby zatem obliczyć odległość między cechami nominalnymi, musimy przekształcić je w format

liczbowy. Typowym rozwiązaniem jest **kodowanie dychotomiczne** (ang. *dummy coding*), w którym wartość 1 wskazuje jedną kategorię, a 0 drugą. Na przykład kodowanie dychotomiczne zmiennej płci można przeprowadzić następująco:

$$\text{męska} = \begin{cases} 1 & \text{jeśli x = męska} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Zauważ, jak kodowanie dychotomiczne dwuklasowej (binarnej) zmiennej płci daje w wyniku jedną nową cechę o nazwie „męska". Nie trzeba konstruować oddzielnej cechy dla płci niemęskiej. Ponieważ cechy te wzajemnie się wykluczają, wystarczy znać jedną lub drugą.

Dotyczy to również bardziej ogólnego przypadku. *N*-klasową zmienną nominalną można zakodować dychotomicznie poprzez utworzenie binarnych zmiennych sygnalizacyjnych dla *n*–1 poziomów cechy. Na przykład trójklasową zmienną temperatury (gorąco, średnio, zimno), można zakodować za pomocą (3–1) = 2 cech, jak pokazano poniżej:

$$\text{gorąco} = \begin{cases} 1 & \text{jeśli x = gorąco} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

$$\text{średnio} = \begin{cases} 1 & \text{jeśli x = średnio} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Kiedy wiemy, że zarówno cecha „gorąco", jak i „średnio" mają wartość 0, mamy wystarczająco dużo informacji, aby stwierdzić, że jest zimno, a zatem trzecia zmienna binarna dla kategorii „zimno" jest niepotrzebna. Jednak pokrewna metoda, znana jako **kodowanie z gorącą jedynką** (ang. *one-hot encoding*), tworzy cechy binarne dla wszystkich *n* poziomów cechy, a nie *n–1*, jak w kodowaniu dychotomicznym. „Gorąca jedynka" odnosi się do faktu, że tylko jeden atrybut jest kodowany jako 1, a wszystkie inne są ustawiane na 0.

W praktyce te dwie metody prawie się nie różnią, a wybór kodowania nie ma wpływu na wyniki uczenia maszynowego. Warto jednak zauważyć, że kodowanie z gorącą jedynką może powodować problemy w modelach liniowych, takich jak opisane w rozdziale 6. „Prognozowanie danych liczbowych — metody regresji", więc często unika się go w statystyce albo dziedzinach (takich jak ekonomia) które intensywnie korzystają z tego rodzaju modeli. Z drugiej strony stało się ono bardzo popularne w dziedzinie uczenia maszynowego i jest często traktowane synonimicznie z kodowaniem dychotomicznym z tej prostej przyczyny, że wybór nie sprawia praktycznie żadnej różnicy, jeśli chodzi o dopasowanie modelu; jednak w kodowaniu z gorącą jedynką sam model może być bardziej zrozumiały, ponieważ wszystkie poziomy cech kategorycznych są określone jawnie. W tej książce używam tylko kodowania dychotomicznego, ponieważ jest bardziej uniwersalne, ale prędzej czy później z pewnością napotkasz również kodowanie z gorącą jedynką.

Użytecznym aspektem zarówno kodowania dychotomicznego, jak i kodowania z gorącą jedynką jest to, że odległość między cechami zakodowanymi dychotomicznie zawsze wynosi jeden lub zero, a zatem wartości leżą na tej samej skali, co dane liczbowe znormalizowane metodą min-max. Nie są potrzebne żadne dodatkowe przekształcenia.

Wskazówka

Jeśli cecha nominalna ma charakter porządkowy (jak w przypadku temperatury), alternatywą dla kodowania dychotomicznego jest ponumerowanie kategorii i zastosowanie normalizacji. Na przykład cechy „gorąco", „średnio" i „zimno" można ponumerować 1, 2, 3 i znormalizować do postaci 0, 0,5 i 1. Jednak podejście to należy stosować tylko wtedy, gdy odstępy między kategoriami są równoważne. Na przykład, choć kategorie przychodu dla biedoty, klasy średniej i bogaczy są uporządkowane, różnica między biedotą a klasą średnią może być inna niż różnica między klasą średnią a bogaczami. Ponieważ odstępy między grupami nie są równe, bezpieczniejszym podejściem będzie kodowanie dychotomiczne.

Dlaczego algorytm k-NN jest „leniwy"?

Algorytmy klasyfikacji oparte na sąsiedztwie uważa się za „leniwe" algorytmy uczenia maszynowego, ponieważ formalnie rzecz biorąc, nie przeprowadzają one żadnej abstrakcji. Całkowicie pomijają procesy abstrakcji i generalizacji, co podważa definicję uczenia się zaproponowaną w rozdziale 1. „Wprowadzenie do uczenia maszynowego".

Zgodnie ze ścisłą definicją uczenia się leniwy uczeń tak naprawdę niczego się nie uczy, a po prostu zapamiętuje dosłownie dane treningowe. Dzięki temu faza treningowa, podczas której w rzeczywistości niczego się nie trenuje, może trwać bardzo krótko. Oczywiście, wadą uczenia leniwego jest to, że proces dokonywania prognoz jest względnie powolny. Ponieważ polegamy przede wszystkim na przykładach treningowych, a nie na abstrakcyjnym modelu, uczenie leniwe nazywa się również **uczeniem na przykładach** lub **uczeniem na pamięć**.

Ponieważ algorytmy uczące się na przykładach nie budują modelu, należą do klasy metod **uczenia nieparametrycznego** — nie poznają żadnych parametrów cechujących dane. Metody nieparametryczne nie generują teorii dotyczących danych wejściowych, więc ograniczają naszą możliwość zrozumienia, jak klasyfikator wykorzystuje dane, a mimo to mogą dokonywać użytecznych prognoz. Uczenie nieparametryczne pozwala uczniowi odnajdować naturalne wzorce, zamiast próbować wpasować dane we wstępnie przyjętą i potencjalnie stronniczą formę funkcjonalną (rysunek 3.5).

Choć klasyfikatory k-NN można uznać za leniwe, to nadal są one całkiem efektywne. Jak się wkrótce przekonasz, prostą zasadę najbliższego sąsiedztwa można wykorzystać do zautomatyzowania procesu wykrywania raka.

Rysunek 3.5. Algorytmy uczenia maszynowego mają różne uprzedzenia i mogą dochodzić do różnych wniosków!

Przykład — diagnozowanie raka piersi za pomocą algorytmu k-NN

Rutynowe badania przesiewowe w kierunku raka piersi umożliwiają zdiagnozowanie choroby i podjęcie leczenia przed wystąpieniem zauważalnych objawów. Proces wczesnego wykrywania obejmuje badanie tkanki piersi pod kątem nieprawidłowych grudek lub guzów. W przypadku wykrycia guza wykonuje się biopsję aspiracyjną cienkoigłową, podczas której za pomocą wydrążonej igły pobiera się z guza niewielką próbkę komórek. Następnie lekarz bada komórki pod mikroskopem, aby określić, czy guz jest złośliwy, czy łagodny.

Gdyby uczenie maszynowe mogło zautomatyzować identyfikację komórek nowotworowych, przyniosłoby to duże korzyści systemowi opieki zdrowotnej. Zautomatyzowane procesy zwiększyłyby efektywność procesu detekcji, dzięki czemu lekarze poświęcaliby mniej czasu na diagnozowanie, a więcej na leczenie choroby. Zautomatyzowany system badań przesiewowych mógłby również zwiększyć dokładność detekcji poprzez wyeliminowanie subiektywnego ludzkiego osądu.

Zbadajmy użyteczność uczenia maszynowego w wykrywaniu raka poprzez stosowanie algorytmu k-NN do pomiarów komórek pobranych od kobiet z nietypowymi zmianami w piersiach.

Etap 1. Zbieranie danych

Wykorzystamy zbiór danych Breast Cancer Wisconsin (Diagnostic) z witryny UCI Machine Learning Repository pod adresem *http://archive.ics.uci.edu/ml*. Dane te zostały udostępnione przez badaczy z Uniwersytetu Wisconsin i zawierają pomiary wykonane na cyfrowych zdjęciach próbek z biopsji aspiracyjnej cienkoigłowej piersi. Wartości reprezentują charakterystykę jąder komórkowych widocznych na obrazie cyfrowym.

Odsyłacz

Więcej informacji o tym zbiorze danych można znaleźć w artykule „Breast Cancer Diagnosis and Prognosis via Linear Programming", O.L. Mangasarian, W.N. Street, W.H. Wolberg, *Operations Research*, 1995, tom 43, s. 570 – 577.

Dane obejmują 569 przykładowych biopsji, każdą z 32 cechami. Jedną z cech jest numer identyfikacyjny, drugą diagnoza nowotworu, a 30 to liczbowe wartości pomiarów laboratoryjnych. Diagnoza jest zakodowana jako „M", co wskazuje nowotwór złośliwy (ang. *malignant*), albo „B", co wskazuje nowotwór łagodny (ang. *benign*).

30 pomiarów liczbowych reprezentuje wartość średnią, błąd standardowy oraz wartość najgorszą (tzn. największą) dla 10 różnych cech jądra komórkowego, takich jak promień, tekstura, obszar, gładkość i zwartość. Na podstawie tych nazw cech wydaje się, że zbiór danych reprezentuje kształty i rozmiary jądra komórkowego, ale jeśli nie jesteś onkologiem, prawdopodobnie nie wiesz, jaki związek mają te parametry ze złośliwością lub łagodnością nowotworu. Wiedza ta jest jednak niepotrzebna, ponieważ komputer wykryje ważne wzorce w procesie uczenia maszynowego.

Etap 2. Badanie i przygotowywanie danych

Eksploracja danych pozwoli nam rzucić nieco światła na związek między cechami a diagnozą raka. Przy okazji przygotujemy dane do użytku z metodą uczenia k-NN.

Wskazówka

Aby samodzielnie wykonywać przykłady w miarę lektury, przejdź do podkatalogu *Rozdział 03* w katalogu z pobranym kodem źródłowym i ustaw go jako katalog roboczy. Dane znajdują się w pliku *wisc_bc_data.csv.csv*. Na potrzeby tej książki pierwotny zbiór danych został nieco zmodyfikowany. W szczególności dodano wiersz nagłówka i losowo uporządkowano wiersze danych.

Zacznijmy od zaimportowania pliku danych CSV, tak jak zrobiliśmy to w poprzednich rozdziałach, wczytując dane o raku piersi do ramki danych `wbcd`:

```
> wbcd <- read.csv("wisc_bc_data.csv")
```

Za pomocą polecenia `str(wbcd)` możemy upewnić się, że zgodnie z oczekiwaniami dane obejmują 569 przykładów i 32 cechy. Oto kilka pierwszych wierszy wyników:

```
> str(wbcd)
'data.frame':      569 obs. of  32 variables:
 $ id               : int  87139402 8910251 905520 ...
 $ diagnosis        : chr  "B" "B" "B" "B" ...
 $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
```

```
$ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
$ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
$ area_mean        : num  464 346 373 385 712 ...
```

Pierwszą cechą jest zmienna całkowita o nazwie `id`. Jest to po prostu unikatowy identyfikator (ID) każdego pacjenta, który nie dostarcza żadnych użytecznych informacji, więc trzeba będzie wykluczyć go z modelu.

Wskazówka

Niezależnie od metody uczenia maszynowego zawsze należy wykluczać zmienne identyfikacyjne. Zaniedbanie tego może prowadzić do błędnych wniosków, ponieważ identyfikator można wykorzystać do prawidłowego przewidzenia każdego przykładu. Dlatego model obejmujący kolumnę ID niemal na pewno będzie nadmiernie dopasowany, więc będzie słabo uogólniał się na przyszłe dane.

Usuńmy cechę `id` z naszej ramki danych. Ponieważ znajduje się ona w pierwszej kolumnie, możemy ją wykluczyć poprzez utworzenie kopii ramki danych bez kolumny numer 1:

```
> wbcd <- wbcd[-1]
```

Następna cecha, `diagnosis`, jest szczególnie interesująca, ponieważ reprezentuje docelowy wynik, który chcemy przewidywać. Cecha ta wskazuje, czy próbka pochodzi z guza złośliwego, czy łagodnego. Wyniki polecenia `table()` pokazują, że 357 guzów jest łagodnych, a 212 jest złośliwych:

```
> table(wbcd$diagnosis)
  B   M
357 212
```

Wiele klasyfikatorów języka R wymaga, aby docelowa cecha była zakodowana jako czynnik, więc będziemy musieli przekodować kolumnę `diagnosis`. Przy okazji wykorzystamy parametr `labels`, aby nadać wartościom „B” i „M” bardziej czytelne nazwy:

```
> wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"),
                           labels = c("Benign", "Malignant"))
```

Przyglądając się wynikom polecenia `prop.table()`, widzimy teraz, że wartości zostały oznaczone etykietami `Benign` oraz `Malignant` i reprezentują odpowiedni 62,7% i 37,3% guzów:

```
> round(prop.table(table(wbcd$diagnosis)) * 100, digits = 1)
   Benign Malignant
     62.7      37.3
```

Wszystkie pozostałe cechy są liczbowe i zgodnie z oczekiwaniami składają się z trzech różnych pomiarów dziesięciu charakterystyk. Dla celów ilustracyjnych przyjrzyjmy się bliżej trzem spośród tych cech:

```
> summary(wbcd[c("radius_mean", "area_mean", "smoothness_mean")])
  radius_mean       area_mean      smoothness_mean
 Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
 1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
 Median :13.370   Median : 551.1   Median :0.09587
 Mean   :14.127   Mean   : 654.9   Mean   :0.09636
 3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
 Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

Czy oglądając te wartości wyświetlone jedna obok drugiej, dostrzegasz jakiś problem? Przypomnij sobie, że obliczenia odległości w algorytmie k-NN są bardzo wrażliwe na skalę pomiarową cech wejściowych. Ponieważ gładkość (ang. *smoothness*) ma zakres od 0,05 do 0,16, a pole (ang. *area*) ma zakres od 143,5 do 2501,0, pole będzie miało znacznie większy wpływ na obliczenia odległości niż gładkość. Może to powodować problemy w naszym klasyfikatorze, więc zastosujmy normalizację, aby przeskalować cechy do standardowego zakresu wartości.

Przekształcanie — normalizacja danych liczbowych

Aby znormalizować te cechy, utworzymy funkcję R o nazwie `normalize()`. Funkcja ta przyjmuje wektor `x` z wartościami liczbowymi i dla każdej wartości w `x` odejmuje minimalną wartość `x` oraz dzieli różnicę przez zakres wartości `x`. Na koniec funkcja zwraca znormalizowany wektor. Kod funkcji jest następujący:

```
> normalize <- function(x) {
    return ((x - min(x)) / (max(x) - min(x)))
}
```

Po wykonaniu poprzedniego kodu możemy używać funkcji `normalize()` w środowisku R. Przetestujmy ją na kilku wektorach:

```
> normalize(c(1, 2, 3, 4, 5))
[1] 0.00 0.25 0.50 0.75 1.00
> normalize(c(10, 20, 30, 40, 50))
[1] 0.00 0.25 0.50 0.75 1.00
```

Wydaje się, że funkcja działa poprawnie. Choć wartości w drugim wektorze są 10 razy większe niż w pierwszym, po normalizacji stają się identyczne.

Możemy teraz zastosować funkcję `normalize()` do cech liczbowych w naszej ramce danych. Zamiast osobno normalizować każdą z 30 zmiennych liczbowych, użyjemy funkcji R do zautomatyzowania tego procesu.

Funkcja `lapply()` przyjmuje listę i stosuje określoną funkcję do każdego elementu listy. Ponieważ ramka danych jest listą wektorów o równej długości, możemy użyć funkcji `lapply()`, aby zastosować funkcję `normalize()` do każdej cechy w ramce danych. Ostatnim etapem będzie przekształcenie listy zwróconej przez `lapply()` w ramkę danych za pomocą funkcji `as.data.frame()`. Cały proces wygląda tak:

```
> wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
```

Mówiąc po ludzku, polecenie to stosuje funkcję `normalize()` do kolumn 2 – 31 w ramce danych `wbcd`, przekształca wynikową listę w ramkę danych i przypisuje tej ramce nazwę `wbcd_n`. Przyrostek `_n` ma przypominać, że wartości zawarte w `wbcd` zostały znormalizowane.

Aby sprawdzić, czy przekształcenie przebiegło poprawnie, przyjrzyjmy się statystykom zbiorczym jednej zmiennej:

```
> summary(wbcd_n$area_mean)
 Min.    1st Qu. Median  Mean    3rd Qu. Max.
 0.0000  0.1174  0.1729  0.2169  0.2711  1.0000
```

Zgodnie z oczekiwaniami zmienna `area_mean`, która pierwotnie miała zakres od 143,5 do 2501,0, teraz ma zakres od 0 do 1.

Wskazówka

Aby uprościć przygotowywanie danych na potrzeby tego przykładu, zastosowaliśmy normalizację min-max do całego zbioru danych — łącznie z wierszami, które później staną się zbiorem testowym. W pewnym sensie narusza to naszą symulację przyszłych, wcześniej niewidzianych danych, bo w praktyce w czasie trenowania modelu rzeczywiste wartości minimalne i maksymalne są zwykle nieznane, a przyszłe wartości mogą wykraczać poza poprzednio zaobserwowany zakres. Lepszym podejściem było znormalizowanie zbioru testowego wyłącznie na podstawie wartości minimalnych i maksymalnych zaobserwowanych w danych treningowych, a może nawet ograniczanie przyszłych wartości do uprzednich maksimów i minimów. Jednak to, czy normalizacja zostanie zastosowana do zbiorów treningowego i testowego osobno, czy łącznie, zwykle nie ma znacznego wpływu na trafność modelu i nie sprawia różnicy w omawianym tu przykładzie.

Przygotowywanie danych — tworzenie treningowego i testowego zbioru danych

Choć każdą z 569 biopsji oznaczono jako łagodną lub złośliwą, przewidywanie tego, co już wiemy, nie jest szczególnie interesujące. Ponadto miary trafności uzyskane podczas treningu mogą być zwodnicze, ponieważ nie wiemy, czy model nadmiernie nie dopasowuje się do danych i jak dobrze uogólni się na przyszłe przypadki. Dlatego ciekawszym pytaniem jest to, czy model będzie dobrze działać na zbiorze danych zawierającym poprzednio niewidziane przykłady. Gdybyśmy mieli dostęp do laboratorium, moglibyśmy zastosować nasz model na pomiarach następnych stu guzów o nieznanym statusie i sprawdzić, jak trafne są jego prognozy w porównaniu z diagnozami uzyskanymi konwencjonalnymi metodami.

Pod nieobecność nowych danych możemy zasymulować ten scenariusz poprzez podział naszych danych na dwie części: zbiór treningowy, który posłuży do zbudowania modelu k-NN, oraz zbiór testowy, który wykorzystamy do oszacowania predykcyjnej dokładności modelu. Pierwsze 469 rekordów wykorzystamy do treningu, a pozostałe 100 — do symulowania nowych pacjentów.

Korzystając z metod ekstrakcji danych, zaprezentowanych w rozdziale 2. „Zarządzanie danymi", podzielimy ramkę danych `wbcd_n` na `wbcd_train` i `wbcd_test`:

```
> wbcd_train <- wbcd_n[1:469, ]
> wbcd_test <- wbcd_n[470:569, ]
```

Jeśli powyższe polecenia są niezrozumiałe, przypomnij sobie, że dane wyodrębnia się z ramek danych z wykorzystaniem składni [*wiersz*, *kolumna*]. Pusta wartość wiersza lub kolumny oznacza wszystkie wiersze lub kolumny. Zatem pierwsze polecenie nakazuje zwrócić wiersze od 1 do 469 i wszystkie kolumny, a drugie — wiersze od 470 do 569 i wszystkie kolumny.

Wskazówka

Podczas tworzenia treningowego i testowego zbioru danych trzeba zadbać o to, żeby każdy z nich był reprezentatywnym podzbiorem pełnego zbioru danych. Rekordy `wbcd` zostały zawczasu uporządkowane losowo i dlatego możemy po prostu wyodrębnić 100 kolejnych rekordów w celu utworzenia reprezentatywnego zbioru testowego. Nie byłoby to właściwe, gdyby dane były uporządkowane chronologicznie albo w grupy o podobnych wartościach. W takich przypadkach potrzebne są metody próbkowania losowego. Próbkowanie losowe zostanie omówione w rozdziale 5. „Dziel i zwyciężaj — klasyfikacja z wykorzystaniem drzew decyzyjnych i reguł".

Podczas konstruowania znormalizowanego zbioru treningowego i testowego wykluczyliśmy docelową zmienną `diagnosis`. Na potrzeby treningu modelu k-NN będziemy musieli zapisać te etykiety klas w wektorach czynnikowych rozdzielonych na zbiór treningowy i testowy:

```
> wbcd_train_labels <- wbcd[1:469, 1]
> wbcd_test_labels <- wbcd[470:569, 1]
```

Kod ten pobiera czynnik `diagnosis` z pierwszej kolumny ramki danych `wbcd` i tworzy wektory `wbcd_train_labels` oraz `wbcd_test_labels`. Użyjemy ich w następnych etapach treningu i ewaluacji naszego klasyfikatora.

Etap 3. Trenowanie modelu na danych

Dysponując danymi treningowymi i wektorem etykiet, jesteśmy gotowi do sklasyfikowania naszych rekordów testowych. W algorytmie k-NN faza treningu nie wiąże się z budowaniem modelu; proces trenowania „leniwego" ucznia, takiego jak k-NN, polega po prostu na zapisaniu danych wejściowych w ustrukturyzowanym formacie.

Do klasyfikowania przykładów testowych użyjemy implementacji k-NN z pakietu `class`, który oferuje zbiór podstawowych funkcji R służących do klasyfikacji. Jeśli pakiet ten nie jest jeszcze zainstalowany w Twoim systemie, zainstaluj go poleceniem:

```
> install.packages("class")
```

Aby wczytać pakiet podczas dowolnej sesji, w której chcesz użyć jego funkcji, po prostu wydaj polecenie `library(class)`.

Funkcja `knn()` z pakietu `class` oferuje standardową, tradycyjną implementację algorytmu k-NN. Dla każdego przykładu w danych testowych funkcja identyfikuje *k* najbliższych sąsiadów na podstawie odległości euklidesowej, gdzie *k* jest parametrem określonym przez użytkownika. Przykład testowy jest klasyfikowany poprzez „głosowanie" *k* najbliższych sąsiadów — mówiąc ściślej, przypisuje mu się klasę większości sąsiadów. Ewentualne remisy rozstrzyga się losowo.

Wskazówka

W innych pakietach R dostępnych jest kilka innych funkcji k-NN, które oferują bardziej zaawansowane lub efektywniejsze implementacje. Jeśli natrafisz na ograniczenia, korzystając z funkcji `knn()`, wyszukaj frazę „k-NN" w witrynie CRAN: *https://cran.r-project.org*.

Trenowanie i klasyfikację z wykorzystaniem funkcji `knn()` przeprowadza się za pomocą jednego polecenia, które wymaga czterech parametrów, jak pokazano na rysunku 3.6.

Teraz mamy niemal wszystko, czego potrzebujemy, aby zastosować algorytm k-NN do naszych danych. Podzieliliśmy je na zbiór treningowy i testowy, z których każdy ma takie same cechy liczbowe. Etykiety dla zbioru treningowego są przechowywane w oddzielnym wektorze czynnikowym. Jedynym pozostałym parametrem jest *k*, który określa liczbę sąsiadów uwzględnianych w głosowaniu.

Ponieważ nasz zbiór treningowy zawiera 469 przykładów, moglibyśmy wypróbować parametr `k = 21`, liczbę nieparzystą mniej więcej równą pierwiastkowi kwadratowemu z 469. Kiedy wynik może należeć do jednej z dwóch kategorii, użycie nieparzystej liczby eliminuje możliwość remisu podczas głosowania.

Składnia klasyfikacji kNN

Użycie funkcji `knn()` z pakietu `class`

Budowanie klasyfikatora i dokonywanie prognoz:

```
p <- knn(train, test, class, k)
```

- `train` to ramka danych zawierająca liczbowe dane treningowe
- `test` to ramka danych zawierająca liczbowe dane testowe
- `class` to wektor czynnikowy z klasami każdego wiersza w zbiorze treningowym
- k to liczba całkowita wskazująca liczbę najbliższych sąsiadów

Funkcja zwraca wektor czynnikowy klas przewidzianych dla każdego wiersza w testowej ramce danych.

Przykład:

```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

Rysunek 3.6. Składnia klasyfikacji kNN

Teraz możemy użyć funkcji `knn()` do sklasyfikowania danych testowych:

```
> wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
                        cl = wbcd_train_labels, k = 21)
```

Funkcja `knn()` zwraca wektor czynnikowy przewidzianych etykiet dla każdego przykładu w zbiorze danych `wbcd_test`. Wyniki tych prognoz przypisaliśmy wektorowi `wbcd_test_pred`.

Etap 4. Ewaluacja modelu

Następnym etapem procesu jest sprawdzenie, jak dobrze przewidziane klasy w wektorze `wbcd_test_pred` pasują do rzeczywistych wartości w wektorze `wbcd_test_labels`. W tym celu możemy użyć funkcji `CrossTable()` z pakietu `gmodels`, który omówiliśmy w rozdziale 2. „Zarządzanie danymi". Jeśli jeszcze nie zainstalowałeś tego pakietu, zrób to teraz poleceniem `install.packages("gmodels")`.

Po wczytaniu pakietu poleceniem `library(gmodels)` możemy utworzyć tabelę krzyżową pokazującą zgodność między wektorami przewidzianych i rzeczywistych etykiet. Podanie parametru `prop.chisq = FALSE` usuwa z wyników niepotrzebne nam wartości chi kwadrat:

```
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
           prop.chisq = FALSE)
```

Wynikowa tabela wygląda tak:

```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.968 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        37 |        39 |
                 |     0.051 |     0.949 |     0.390 |
                 |     0.032 |     1.000 |           |
                 |     0.020 |     0.370 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        63 |        37 |       100 |
                 |     0.630 |     0.370 |           |
-----------------|-----------|-----------|-----------|
```

Wartości procentowe w tabeli wskazują proporcje wartości, które należą do czterech kategorii. Lewa górna komórka wskazuje wyniki **prawdziwie negatywne**: 61 spośród 100 wartości to przypadki, w których guz był łagodny, a algorytm k-NN poprawnie zidentyfikował go jako taki. Prawa dolna komórka wskazuje wyniki **prawdziwie pozytywne**, w których klasyfikator i ustalona klinicznie etykieta zgadzają się, że guz jest złośliwy. Prawdziwie pozytywnych było 37 spośród 100 prognoz.

Komórki leżące na drugiej przekątnej zawierają liczby przypadków, w których prognoza k-NN nie zgadzała się z rzeczywistą etykietą. Dwa przykłady w lewej dolnej komórce to wyniki **fałszywie negatywne**; w tym przypadku algorytm k-NN przewidział guz łagodny, który w rzeczywistości był złośliwy. Błędy w tym kierunku mogą być niezwykle kosztowne, ponieważ skłaniają pacjenta do myślenia, że nie ma raka, a w rzeczywistości choroba może postępować.

Górna prawa komórka zawierałaby wyniki **fałszywie pozytywne**, gdybyśmy jakieś mieli. Wartości te pojawiają się wtedy, kiedy model sklasyfikuje guz jako złośliwy, podczas gdy w rzeczywistości jest on łagodny. Choć takie błędy są mniej niebezpieczne od wyników fałszywie negatywnych, ich również należy unikać, ponieważ mogą stanowić dodatkowe obciążenie finansowe dla systemu opieki zdrowotnej i powodować stres u pacjenta poddawanego niepotrzebnym testom lub zabiegom.

Podejście k-NN niepoprawnie sklasyfikowało 2 ze 100, czyli 2% guzów. Choć 98-procentowa dokładność wydaje się imponująca, jak na kilka wierszy kodu R, możemy wypróbować kolejną iterację modelu i sprawdzić, czy możemy zwiększyć trafność i ograniczyć liczbę błędnie sklasyfikowanych wartości, zwłaszcza że błędy były niebezpiecznymi „fałszywymi negatywami".

Wskazówka

Gdybyśmy chcieli, moglibyśmy wyeliminować wszystkie wyniki fałszywie negatywne poprzez klasyfikowanie każdego guza jako złośliwego. Oczywiście, taka strategia nie jest realistyczna, ale ilustruje fakt, że w przewidywaniu chodzi o znalezienie równowagi między wskaźnikami wyników fałszywie pozytywnych i fałszywie negatywnych. W rozdziale 10. „Ewaluacja działania modelu" poznasz metody oceniania dokładności predykcyjnej, które pozwalają optymalizować model pod kątem kosztu błędów każdego typu.

Etap 5. Poprawianie działania modelu

Wypróbujemy dwie proste wariacje na temat poprzedniego klasyfikatora. Najpierw zastosujemy alternatywną metodę skalowania cech liczbowych. Następnie wypróbujemy kilka różnych wartości *k*.

Przekształcanie — standaryzacja Z

Choć do klasyfikacji k-NN często używa się normalizacji, standaryzacja Z może być bardziej odpowiednim sposobem skalowania cech w zbiorze danych dotyczących raka.

Ponieważ wartości poddane standaryzacji Z nie mają wstępnie zdefiniowanego minimum i maksimum, wartości skrajne nie są przesuwane w kierunku środka. Nawet bez wykształcenia medycznego można podejrzewać, że nowotwór złośliwy może prowadzić do wartości skrajnie odstających w miarę niekontrolowanego wzrostu guza. Rozsądnie będzie więc pozwolić, aby wartości odstające miały większy wkład w obliczenia odległości. Zobaczmy, czy standaryzacja Z poprawi dokładność predykcyjną.

Aby ustandaryzować wektor, możemy użyć wbudowanej funkcji R o nazwie `scale()`, która domyślnie skaluje wartości z wykorzystaniem standaryzacji Z. Funkcję `scale()` można zastosować bezpośrednio do ramki danych, więc nie trzeba używać funkcji `lapply()`. Aby utworzyć ustandaryzowaną wersję danych `wbcd`, wydaj polecenie:

```
> wbcd_z <- as.data.frame(scale(wbcd[-1]))
```

Spowoduje to przeskalowanie wszystkich cech z wyjątkiem `diagnosis` w pierwszej kolumnie i zapisanie wyników w ramce danych `wbcd_z`. Przyrostek `_z` ma przypominać, że wartości zostały poddane standaryzacji Z.

Aby upewnić się, że przekształcenie przebiegło poprawnie, możemy przyjrzeć się statystykom zbiorczym:

```
> summary(wbcd_z$area_mean)
Min.    1st Qu. Median   Mean    3rd Qu. Max.
-1.4532 -0.6666 -0.2949  0.0000  0.3632  5.2459
```

Średnią zmiennej poddanej standaryzacji Z powinno zawsze być zero, a zakres powinien być dość zwarty. Wynik *z* mniejszy niż –3 lub większy niż 3 wskazuje niezwykle rzadką wartość. Badając statystyki zbiorcze według tych kryteriów, dochodzimy do wniosku, że przekształcenie prawdopodobnie było poprawne.

Jak wcześniej, tak i teraz musimy podzielić dane poddane standaryzacji Z na zbiory treningowy i testowy oraz sklasyfikować przykłady testowe za pomocą funkcji `knn()`. Następnie porównamy przewidziane i rzeczywiste etykiety za pomocą funkcji `Cross` ↪`Table()`:

```
> wbcd_train <- wbcd_z[1:469, ]
> wbcd_test <- wbcd_z[470:569, ]
> wbcd_train_labels <- wbcd[1:469, 1]
> wbcd_test_labels <- wbcd[470:569, 1]
> wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
                        cl = wbcd_train_labels, k = 21)
>    CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
               prop.chisq = FALSE)
```

Niestety, wyniki w zamieszczonej niżej tabeli wskazują niewielki spadek dokładności. Używając tych samych danych, w których poprzednio sklasyfikowaliśmy poprawnie 98% przykładów, teraz klasyfikujemy poprawnie tylko 95%. Co gorsza, nie poradziliśmy sobie lepiej z klasyfikowaniem groźnych fałszywych negatywów.

```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.924 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         5 |        34 |        39 |
                 |     0.128 |     0.872 |     0.390 |
                 |     0.076 |     1.000 |           |
                 |     0.050 |     0.340 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        66 |        34 |       100 |
                 |     0.660 |     0.340 |           |
-----------------|-----------|-----------|-----------|
```

Testowanie różnych wartości k

Możemy spróbować zoptymalizować model k-NN poprzez zbadanie jego dokładności przy kilku różnych wartościach *k*. Używając znormalizowanego zbioru treningowego i testowego, sklasyfikujmy 100 tych samych rekordów z kilkoma wybranymi wartościami *k*. Ponieważ będziemy testować tylko sześć wartości *k*, moglibyśmy po prostu kilkakrotnie wyciąć i wkleić poprzednie wywołania funkcji `knn()` i `CrossTable`. Możemy

jednak również napisać pętlę `for`, która wykona te dwie funkcje dla każdej wartości w wektorze o nazwie `k_values`, jak pokazano w poniższym kodzie:

```
> k_values <- c(1, 5, 11, 15, 21, 27)
> for (k_val in k_values) {
    wbcd_test_pred <- knn(train = wbcd_train,
                          test = wbcd_test,
                          cl = wbcd_train_labels,
                          k = k_val)
    CrossTable(x = wbcd_test_labels,
               y = wbcd_test_pred,
               prop.chisq = FALSE)
  }
```

Pętlę `for` można czytać jak proste zdanie: dla każdej wartości nazwanej `k_val` w wektorze `k_values` wykonaj funkcję `knn()`, ustawiając parametr `k` na bieżącą wartość `k_val`, a następnie wygeneruj tabelę `CrossTable()` dla wynikowych prognoz.

Wskazówka

Bardziej zaawansowana metoda przetwarzania danych w pętli z wykorzystaniem jednej z funkcji `apply()` języka R jest opisana w rozdziale 7. „Czarne skrzynki — sieci neuronowe i maszyny wektorów nośnych". Pozwala ona testować różne wartości parametru kosztu i wykreślać wyniki.

Poniżej pokazano wyniki fałszywie negatywne, fałszywie pozytywne oraz ogólny wskaźnik błędu dla każdej iteracji:

Wartość *k*	Wyniki fałszywie negatywne	Wyniki fałszywie pozytywne	Wskaźnik błędu
1	1	3	4%
5	2	0	2%
11	3	0	3%
15	3	0	3%
21	2	0	2%
27	4	0	4%

Choć klasyfikator nigdy nie działał idealnie, metoda 1-NN uniknęła części wyników fałszywie negatywnych kosztem dodania wyników fałszywie pozytywnych. Warto jednak pamiętać, że nie należy zbyt ściśle dopasowywać podejścia do danych testowych; bądź co bądź, inny zbiór 100 rekordów pacjentów prawdopodobnie będzie nieco różnić się od tego, którego użyliśmy do ewaluacji naszego modelu.

Wskazówka

Jeśli musisz mieć pewność, że uczeń uogólni się na przyszłe dane, możesz losowo utworzyć kilka zbiorów pacjentów i wielokrotnie powtarzać testy. Takie metody uważnej ewaluacji modeli uczenia maszynowego zostaną omówione dokładniej w rozdziale 10. „Ewaluacja działania modelu".

Podsumowanie

W tym rozdziale poznałeś klasyfikację metodą k-NN. W przeciwieństwie do wielu innych algorytmów klasyfikacji, algorytm *k* najbliższych sąsiadów niczego się nie uczy — przynajmniej nie według formalnej definicji uczenia maszynowego. Zamiast tego po prostu dosłownie zapamiętuje dane treningowe. Następnie za pomocą funkcji odległości dopasowuje nieoznaczone przykłady testowe do najbardziej podobnych rekordów w zbiorze treningowym i przypisuje każdemu nieoznaczonemu przykładowi etykietę, którą ma najwięcej jego najbliższych sąsiadów.

Choć k-NN jest bardzo prostym algorytmem, można wykorzystywać go w bardzo skomplikowanych zadaniach, takich jak identyfikowanie zmian nowotworowych. W kilku wierszach kodu R zdołaliśmy poprawnie określić złośliwość 98% guzów na podstawie rzeczywistych danych. Choć ten edukacyjny zbiór danych został specjalnie zaprojektowany tak, aby ułatwić budowanie modelu, ćwiczenie to pokazało, że algorytmy uczenia maszynowego mogą dokonywać dokładnych prognoz w sposób podobny do ludzi.

W następnym rozdziale zbadamy metodę klasyfikacji, która używa prawdopodobieństwa, aby oszacować, czy obserwacja należy do pewnej kategorii. Ciekawie będzie porównać, czym ta metoda różni się od algorytmu k-NN. W rozdziale 9. „Znajdowanie grup danych — klasteryzacja metodą k-średnich" poznasz algorytm blisko spokrewniony z k-NN, który wykorzystuje miary odległości do zupełnie innego zadania.

Naucz się przekształcać surowe dane w wiedzę!

Uczenie maszynowe polega na przekształcaniu danych w informacje ułatwiające podejmowanie decyzji. W erze big data umożliwia pracę z ogromnymi strumieniami napływających informacji — pozwala na ich zrozumienie i efektywne zastosowanie. Ulubionym narzędziem analityków danych jest bezpłatne wieloplatformowe środowisko programowania statystycznego o nazwie R, oferujące potężne, intuicyjne i łatwe do opanowania narzędzia.

To czwarte, zaktualizowane wydanie znakomitego przewodnika poświęconego zastosowaniu uczenia maszynowego do rozwiązywania rzeczywistych problemów w analizie danych. Dzięki książce dowiesz się wszystkiego, co trzeba wiedzieć o wstępnym przetwarzaniu danych, znajdowaniu kluczowych spostrzeżeń, prognozowaniu i wizualizowaniu odkryć. W tym wydaniu dodano kilka nowych rozdziałów dotyczących data science i niektórych trudniejszych zagadnień, takich jak zaawansowane przygotowywanie danych, budowanie lepiej uczących się modeli i praca z big data. Znalazło się tu także omówienie etycznych aspektów uczenia maszynowego i wprowadzenie do uczenia głębokiego. Treść została zaktualizowana do wersji 4.0.0 języka R.

W książce:

- kompleksowa realizacja procesu uczenia maszynowego
- przeprowadzenie predykcji za pomocą drzew decyzyjnych, reguł i maszyn wektorów nośnych
- szacowanie wartości finansowych przy użyciu regresji
- modelowanie złożonych procesów z wykorzystaniem sztucznych sieci neuronowych
- ocenia modeli i poprawa ich trafność
- łączenie R z bazami danych SQL i nowymi technologiami big data

Brett Lantz korzysta z innowacyjnych metod analizy danych, aby lepiej zrozumieć ludzkie zachowanie. Jest z wykształcenia socjologiem i instruktorem DataCamp, prowadzi warsztaty uczenia maszynowego na całym świecie. Interesuje się między innymi zastosowaniami data science w sporcie, grach wideo, pojazdach autonomicznych i nauce języków obcych.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-0899-4
9 788328 908994
Cena: 139,00 zł
packt