Helion

Algorytmy uczenia maszynowego

Zaawansowane techniki implementacji

Packt›

Giuseppe Bonaccorso

Tytuł oryginału: Mastering Machine Learning Algorithms: Expert techniques to implement popular machine learning algorithms and fine-tune your models

Tłumaczenie: Krzysztof Sawka

ISBN: 978-83-283-5245-2

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/alguma.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Podstawy modelu uczenia maszynowego

Modele uczenia maszynowego są układami matematycznymi zawierającymi wiele cech wspólnych. Nawet jeśli są czasami definiowane jedynie z perspektywy teoretycznej, postępy badań pozwalają nam wykorzystywać różne koncepcje służące do lepszego zrozumienia mechanizmów działania złożonych systemów, takich jak głębokie sieci neuronowe. W tym rozdziale poznamy i omówimy pewne podstawowe elementy, które mogą być już doskonale znane doświadczonym Czytelnikom, ale które jednocześnie można różnie interpretować i które mają różne zastosowania.

W szczególności omówimy następujące elementy:

- procesy generowania danych;
- skończone zestawy danych;
- strategie uczenia i rozdzielania danych;
- sprawdzian krzyżowy;
- pojemność, obciążenie i wariancja modelu;
- teoria Vapnika-Chervonenkisa;
- granica Craméra-Rao;
- niedotrenowanie i przetrenowanie;
- funkcje straty i kosztu;
- regularyzacja.

Modele a dane

Algorytmy uczenia maszynowego pracują na danych. Tworzą powiązania, wyszukują relacje, odkrywają wzorce, tworzą nowe przykłady itd. na podstawie dobrze zdefiniowanych zestawów danych. Niestety, czasami przyjmowane założenia lub nakładane warunki nie są wyraźnie sprecyzowane, a długotrwały proces uczenia może zakończyć się całkowitym niepowodzeniem weryfikacji. Nawet jeśli taki warunek jest wyraźniej zarysowany w kontekście uczenia głębokiego, możemy wyobrazić sobie model jako szarą skrzynkę (pewną przejrzystość gwarantuje prostota wielu popularnych algorytmów), w której wektor wejściowy $\overline{X}$ zostaje przekształcony w wektor wyjściowy $\overline{Y}$ (rysunek 1.1).

Rysunek 1.1. Schemat typowego modelu parametryzowanego za pomocą wektora θ

Na rysunku 1.1 model został ukazany jako pseudofunkcja zależna od zestawu parametrów zdefiniowanych w wektorze θ. W tym podrozdziale bierzemy pod uwagę wyłącznie modele **parametryczne**, ale istnieje również rodzina modeli **nieparametrycznych**, bazujących wyłącznie na strukturze danych. Niektórymi z nich zajmiemy się w następnych rozdziałach.

Zadaniem parametrycznego procesu uczenia jest znalezienie najlepszego zestawu parametrów maksymalizującego funkcję docelową, której wartość jest proporcjonalna do dokładności (lub błędu, jeżeli staramy się ją minimalizować) modelu dla określonych danych wejściowych X i wyjściowych Y. Definicja ta nie jest zbyt ścisła, ale zawęzimy ją w kolejnych podrozdziałach, gdyż okazuje się jednak przydatna do zrozumienia kontekstu, w jakim przyjdzie nam pracować.

Pojawia się teraz pierwsze pytanie: jaka jest natura danych wejściowych X? Problem uczenia maszynowego sprowadza się do wyszukiwania abstrakcyjnych związków pozwalających na spójne uogólnianie wobec nowo dostarczanych przykładów. Mówiąc konkretniej, możemy zdefiniować stochastyczny **proces generowania danych** wykorzystujący wspólny rozkład prawdopodobieństwa:

$$p_{dane}\left(\overline{x}, \overline{y}\right) = p\left(\overline{x} \mid \overline{y}\right) p\left(\overline{x}\right)$$

Część wspólną prawdopodobieństwa $p(x, y)$ warto czasami wyrażać jako iloczyn prawdopodobieństwa warunkowego $p(x \mid y)$, czyli prawdopodobieństwa wystąpienia etykiety danego przykładu, i prawdopodobieństwa brzegowego przykładów $p(x)$. Wyrażenie to jest szczególnie użyteczne, jeżeli prawdopodobieństwo wstępne $p(x)$ jest znane w kontekstach półnadzorowanych lub jeśli interesuje nas rozwiązywanie problemów za pomocą algorytmu **oczekiwania-maksymalizacji**. Wrócimy do tego zagadnienia w dalszej części książki.

W wielu przypadkach nie jesteśmy w stanie określić precyzyjnie rozkładu, zajmując się jednak zestawem danych, zawsze przyjmujemy, że został on wygenerowany za pomocą pierwotnego rozkładu. Warunek ten nie jest wyłącznie założeniem teoretycznym, ponieważ, jak się przekonamy, za każdym razem gdy nasze punkty danych będą generowane przy użyciu innego rozkładu, dokładność modelu może gwałtownie zmaleć.

Jeżeli weźmiemy N wartości **niezależnych i pochodzących z tego samego rozkładu prawdopodobieństwa** (ang. *independent and identically distributed — i. i. d.*) z procesu p_{dane}, możemy utworzyć skończony zestaw danych X składający się z k-wymiarowych wektorów rzeczywistych:

$$X = \left\{ \overline{x}_0, \overline{x}_1, \ldots, \overline{x}_N \right\} \quad gdzie\ \overline{x}_i \in \mathbb{R}^k$$

W przypadku uczenia nadzorowanego potrzebne nam również będą odpowiednie etykiety (z t wartości wyjściowych):

$$Y = \left\{ \overline{y}_0, \overline{y}_1, \ldots, \overline{y}_N \right\} \quad gdzie\ \overline{y}_i \in \mathbb{R}^t$$

Gdy dane wyjściowe zawierają więcej niż dwie klasy, możemy skorzystać z różnych strategii zarządzania tym problemem. W przypadku klasycznego uczenia maszynowego jedną z najpopularniejszych technik jest **jeden-przeciw-wszystkim** (ang. *One-versus-All — OvA*), która polega na uczeniu N różnych klasyfikatorów binarnych, gdzie każda etykieta jest oceniana w stosunku do wszystkich pozostałych. W ten sposób do określenia prawidłowej klasy wykonywanych jest *N-1* przebiegów. Z kolei w płytkich i głębokich sieciach neuronowych używamy często funkcji **softmax** określającej wyjściowy rozkład prawdopodobieństwa dla wszystkich klas:

$$\tilde{y}_i = \left(\frac{e^{z_0}}{\sum e^z}, \frac{e^{z_1}}{\sum e^z}, \ldots, \frac{e^{z_N}}{\sum e^z} \right)$$

Możemy łatwo zarządzać takim rodzajem danych wyjściowych (z_i symbolizuje wartości pośrednie, a suma wszystkich członów jest normalizowana do wartości 1) za pomocą specjalnego rodzaju funkcji kosztu — entropii krzyżowej (patrz odpowiedni akapit w podrozdziale „Funkcje straty i kosztu").

Środkowanie i wybielanie

Wiele algorytmów wykazuje większą skuteczność (przede wszystkim w kontekście szybkości uczenia), gdy zestaw danych jest symetryczny (o średniej równej 0). Zatem jednym z najważniejszych etapów wstępnego przetwarzania danych jest tzw. **środkowanie** (ang. *zero-centering*), polegające na odjęciu od wszystkich przykładów średniej wyliczonej z cech $E_x[X]$:

$$\hat{x}_i = \overline{x}_i - E_x\left[X \right]$$

Zazwyczaj w razie potrzeby operacja ta jest odwracalna i nie wpływa na relacje zarówno pomiędzy przykładami, jak i składowymi danego przykładu. W przypadku uczenia głębokiego wyśrodkowany zestaw danych pozwala wykorzystywać symetrię pewnych funkcji aktywacji, co przyśpiesza osiąganie zbieżności (szczegóły omówimy w kolejnych rozdziałach).

Kolejnym bardzo ważnym etapem wstępnej obróbki danych jest **wybielanie** (ang. *whitening*), czyli operacja nakładania jednostkowej macierzy kowariancji na wyśrodkowany zestaw danych:

$$E_x\left[X^T X\right] = I$$

Macierz kowariancji $E_x[X^T X]$ jest rzeczywista i symetryczna, dlatego możemy rozłożyć ją na wektory i wartości własne bez konieczności odwracania macierzy wektorów własnych:

$$E_x\left[X^T X\right] = V\Omega V^T$$

Macierz V zawiera wektory własne (jako kolumny), a macierz diagonalna Ω przechowuje wartości własne. Aby rozwiązać ten problem, musimy znaleźć taką macierz A, że:

$$\hat{x}_i = A\overline{x}_i \quad i \quad E_x\left[\hat{X}^T \hat{X}\right] = I$$

Za pomocą rozkładu do wektorów i wartości własnych uzyskujemy:

$$E_x\left[\hat{X}^T \hat{X}\right] = E_x\left[AX^T XA^T\right] = AE_x\left[X^T X\right]A^T = AV\Omega V^T A^T = I$$

Zatem macierz A przyjmuje postać:

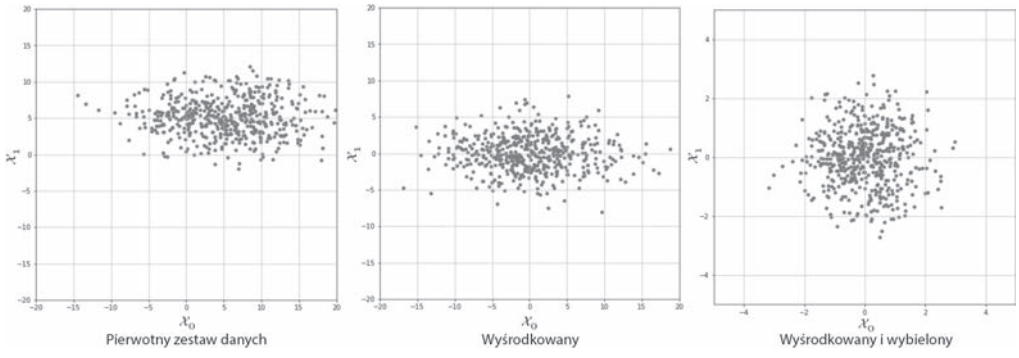
$$AA^T = V\Omega^{-1}V^T \Rightarrow A = V\Omega^{-\frac{1}{2}}$$

Jedną z głównych zalet wybielania jest dekorelacja zestawu danych, co ułatwia nam rozdzielanie składowych. Ponadto jeśli macierz X zostaje wybielona, każde przekształcenie ortogonalne powodowane przez macierz P również jest wybielone:

$$Y = PX \Rightarrow E\left[Y^T Y\right] = PE\left[X^T X\right]P^T = PP^T = I$$

Do tego wiele algorytmów, które wymagają szacowania parametrów ściśle powiązanych z wejściową macierzą kowariancji, również korzysta na tym warunku, ponieważ powoduje on zmniejszenie rzeczywistej liczby zmiennych niezależnych (generalnie algorytmy te działają na macierzach, które po przeprowadzeniu wybielania stają się symetryczne). Kolejną ważną zaletą w kontekście uczenia głębokiego jest fakt, że gradienty są często większe w pobliżu środka układu współrzędnych, a maleją w obszarach, w których funkcje aktywacji (np. tangens hiperboliczny lub sigmoidalna) ulegają nasyceniu ($|x| \rightarrow \infty$). Z tego właśnie powodu modele uzyskują zbieżność szybciej wobec wybielonych (i wyśrodkowanych) zestawów danych.

Na rysunku 1.2 porównujemy **pierwotny zestaw danych** z jego **wyśrodkowaną i wybieloną** wersją.



Rysunek 1.2. Pierwotny zestaw danych (po lewej), jego wersja wyśrodkowana (w środku) i wybielona (po prawej)

Jeżeli chcemy przeprowadzić wybielanie, musimy uwzględnić kilka istotnych szczegółów. Po pierwsze, istnieje różnica w skali pomiędzy rzeczywistą kowariancją przykładu a oszacowaniem X^TX, często przyjmowana w postaci **rozkładu według wartości osobliwych** (ang. *singular value decomposition — SVD*). Druga kwestia wiąże się z pewnymi klasami zaimplementowanymi w wielu środowiskach, np. z klasą `StandardScaler` biblioteki Scikit-Learn. W istocie środkowanie jest operacją na cechach, natomiast filtr wybielający należy obliczyć przy uwzględnieniu całej macierzy kowariancji (klasa `StandardScaler` implementuje jedynie skalowanie cech bazujące na wariancji jednostkowej).

Na szczęście wszystkie algorytmy zawarte w bibliotece Scikit-Learn, które czerpią korzyści z wybielania lub go wymagają, zawierają wbudowane odpowiednie funkcje, dzięki czemu zazwyczaj nie wymagają naszej interwencji. Dla Czytelników, którzy pragną implementować algorytmy bezpośrednio, napisałem dwie funkcje używane do środkowania i wybielania. Zakładam w nich, że macierz X ma wymiary $[N_{przyk} \times n]$. Ponadto funkcja `whiten()` przyjmuje parametr `correct`, pozwalający przeprowadzać korekcję skalowania (jej domyślna wartość to `True`):

```
import numpy as np

def zero_center(X):
    return X - np.mean(X, axis=0)

def whiten(X, correct=True):
    Xc = zero_center(X)
    _, L, V = np.linalg.svd(Xc)
    W = np.dot(V.T, np.diag(1.0 / L))
    return np.dot(Xc, W) * np.sqrt(X.shape[0]) if correct else 1.0
```

Zbiory uczące i walidacyjne

W rzeczywistych sytuacjach liczba przykładów jest ograniczona i zazwyczaj trzeba rozdzielać pierwotny zestaw danych *X* (wraz z etykietami *Y*) na dwa następujące podzbiory:

- **zbiór uczący** używany do uczenia modelu,
- **zbiór walidacyjny** stosowany do obiektywnego oceniania wyniku modelu za pomocą nieznanych przykładów.

W zależności od rodzaju problemu można dobrać stosunek podziału 70% – 30% (jest to dobry kompromis w przypadku, gdy zestaw danych jest względnie mały) lub przeznaczyć więcej przykładów na zbiór uczący (80%, 90% do 99%) w zadaniach uczenia głębokiego, gdzie mamy znaczną liczbę przykładów uczących. Zakładamy w obydwu sytuacjach, że zbiór uczący zawiera wszystkie informacje wymagane do spójnego uogólniania. W wielu prostych przypadkach jest to prawdziwe i łatwe do zweryfikowania; jednak wraz ze wzrostem złożoności zestawu danych problem ten staje się coraz trudniejszy. Nawet jeśli chcemy wykorzystać przykłady z tego samego rozkładu, może się tak zdarzyć, że losowo dobrany zbiór testowy zawiera cechy nieobecne w innych przykładach uczących. Taki warunek może mieć bardzo negatywny wpływ na dokładność globalną i bez używania innych metod może być również bardzo trudny do zidentyfikowania. Jest to jeden z powodów stosowania olbrzymich zestawów danych w uczeniu głębokim: biorąc pod uwagę złożoność cech i strukturę rozkładów generujących dane, wybór dużego zbioru testowego może ograniczać możliwość poznawania określonych powiązań.

W bibliotece Scikit-Learn możliwe jest rozdzielanie pierwotnego zestawu danych za pomocą funkcji `train_test_split()`, w której możemy określać rozmiar zbiorów uczącego/testowego, a także decydujemy, czy przykłady mają być losowo tasowane (domyślnie tak). Na przykład jeśli chcemy rozdzielić zestawy *X* i *Y* na 70% zbioru uczącego i 30% zbioru testowego, możemy napisać:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7,
↪random_state=1)
```

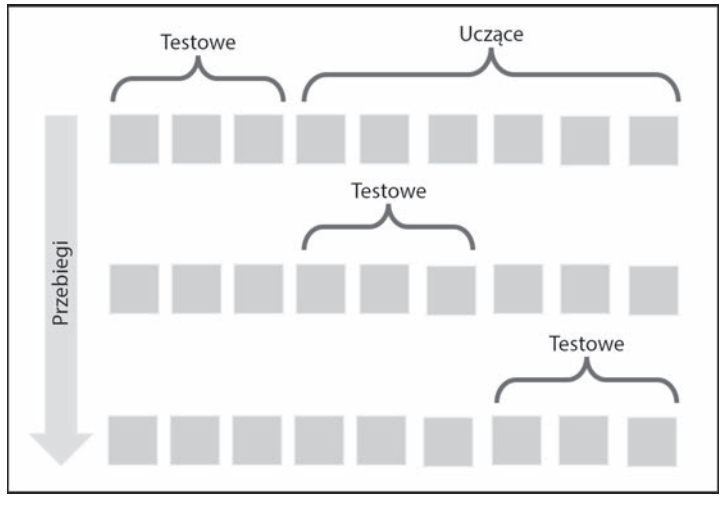
Tasowanie zbiorów jest zawsze dobrym pomysłem, gdyż zmniejszamy korelację pomiędzy przykładami. W rzeczywistości założyliśmy, że zestaw danych X składa się z przykładów niezależnych i pochodzących z tego samego rozkładu prawdopodobieństwa, ale w kilku przypadkach następujące po sobie przykłady wykazują silną korelację, co zmniejsza skuteczność uczenia. W pewnych przypadkach warto również tasować zbiór uczący po każdej epoce, jednakże w większości ćwiczeń będziemy pracować na zbiorze danych przetasowanym tylko raz — na początku. Należy unikać tasowania w przypadku pracy z sekwencjami i modelami bazującymi na pamięci: w takich sytuacjach musimy wykorzystywać istniejącą korelację do określania rozkładu przyszłych przykładów.

> Podczas pracy z bibliotekami NumPy i Scikit-Learn zawsze warto wyznaczyć stałą wartość ziarna losowości, dzięki czemu inne osoby będą w stanie odtworzyć przebieg eksperymentu w tych samych warunkach początkowych. Wystarczy wywołać funkcję `np.random.seed(…)` i użyć parametru `random_state` dostępnego w wielu metodach Scikit-Learn.

Sprawdzian krzyżowy

Dobrym sposobem wykrywania problemów z niewłaściwie dobranymi zbiorami testowymi jest technika **sprawdzianu krzyżowego** (**kroswalidacji**; ang. *cross-validation*). W szczególności interesuje nas **k-krotny** sprawdzian krzyżowy (ang. *K-fold cross-validation*). Mechanizm kroswalidacji polega na podzieleniu zestawu danych *X* na zmienny zbiór testowy i zbiór uczący (pozostałe przykłady). Rozmiar zbioru testowego jest uzależniony od liczby grup — w ciągu *k* przebiegów zbiór testowy wykorzystuje wszystkie przykłady z zestawu danych.

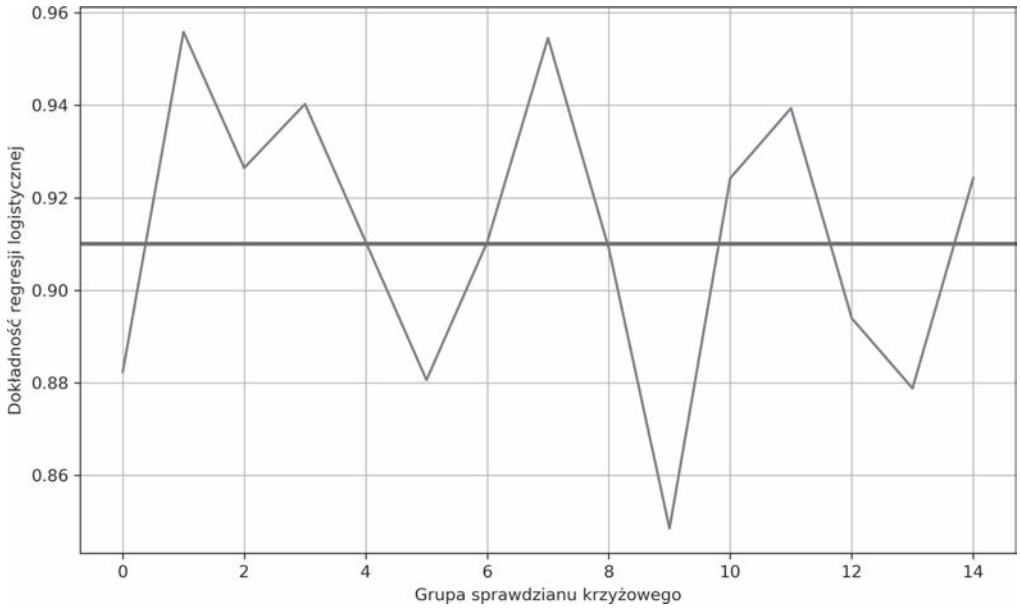
Proces ten został zaprezentowany na rysunku 1.3.



Rysunek 1.3. Schemat k-krotnego sprawdzianu krzyżowego

W ten sposób możliwa staje się ocena dokładności modelu za pomocą różnych podziałów próbkowania, a proces uczenia może być realizowany przy użyciu większych zestawów danych, dokładniej zaś na *(k–1)*N* przykładach. W idealnej sytuacji dokładność powinna być podobna we wszystkich przebiegach, w rzeczywistości jednak osiąga ona wartości poniżej średniej. Oznacza to, że zbiór uczący tworzą przykłady niezawierające cech potrzebnych do właściwego dopasowania hiperpowierzchni separujących modelu przy uwzględnieniu p_{dane}. W dalszej części rozdziału przyjrzymy się uważniej tym zagadnieniom, jeśli jednak odchylenie standardowe dokładności jest zbyt duże (wartość progową ustalamy w zależności od natury problemu/modelu), to oznacza prawdopodobnie, że zestaw *X* nie został utworzony jednorodnie z p_{dane} i na etapie wstępnego przetwarzania danych warto ocenić wpływ elementów odstających. Rysunek 1.4 przedstawia wykres 15-krotnego sprawdzianu krzyżowego wykonanego dla modelu regresji logistycznej.

Przy średniej wynoszącej 0,91 (ciągła, szara linia pośrodku) wartości oscylują w zakresie od 0,84 do 0,95. W tym konkretnym przypadku, biorąc pod uwagę to, że początkowym celem było użycie klasyfikatora liniowego, możemy stwierdzić, iż wszystkie grupy cechują się dużą dokładnością. Stanowi to dowód, że zestaw danych jest rozdzielny liniowo; występują tu jednak pewne przykłady (grupa 9.) wymagane do uzyskania minimalnej dokładności rzędu 0,88.



Rysunek 1.4. Dokładności uzyskane za pomocą sprawdzianu krzyżowego

K-krotny sprawdzian krzyżowy występuje w różnych odmianach dostosowanych do rozwiązywania określonych problemów:

- **Warstwowy k-krotny sprawdzian krzyżowy** (ang. *stratified k-fold*): **standardowy k-krotny sprawdzian krzyżowy** rozdziela zestaw danych bez uwzględniania rozkładu prawdopodobieństwa $p(y \mid x)$, zatem niektóre grupy mogą teoretycznie zawierać jedynie ograniczoną liczbę etykiet. Z kolei warstwowy k-krotny sprawdzian krzyżowy stara się tak rozdzielić zestaw danych X, żeby wszystkie etykiety były rozłożone równomiernie.
- **Metoda LOO** (ang. *leave-one-out* — pozostaw jeden): ta technika jest najbardziej drastyczna, ponieważ generuje N grup, z których każda zawiera $N–1$ przykładów uczących i tylko jeden przykład testowy. W ten sposób maksymalna możliwa liczba przykładów zostaje użyta do uczenia i całkiem łatwo można sprawdzić, czy algorytm jest zdolny do uzyskania wystarczającej dokładności lub czy może lepiej wybrać inną strategię. Główną wadą tego rozwiązania jest konieczność wyuczenia N modeli, a w przypadku dużej wartości N mogą pojawiać się problemy ze skutecznością. Poza tym przy dużej liczbie przykładów wzrasta prawdopodobieństwo występowania podobieństwa pomiędzy dwoma losowymi przykładami, dlatego wiele grup może dawać niemal identyczne rezultaty. Jednocześnie metoda LOO ogranicza możliwości oceniania zdolności generalizacji, ponieważ jeden przykład testowy nie wystarczy do uzyskania rzetelnego oszacowania.
- **Metoda LPO** (ang. *leave-P-out* — pozostaw P): w tym przypadku liczba przykładów testowych wynosi p (zbiory pokrywające się), zatem liczba grup jest równa wartości współczynnika dwumianowego n po p. Unikamy w ten sposób wad techniki LOO; rozwiązanie to stanowi kompromis pomiędzy zwykłym sprawdzianem krzyżowym

a metodą LOO. Liczba grup może być bardzo duża, możemy ją jednak kontrolować, dostosowując liczbę p przykładów testowych, jeśli jednak będzie ona za mała albo za duża, współczynnik dwumianowy może **eksplodować**. Istotnie, jeśli p zawiera około $n/2$ przykładów, otrzymujemy maksymalną liczbę grup:

$$\binom{n}{p} = \frac{n!}{p!(n-p)!} \approx \prod_{t=1}^{p} \frac{n-t}{t} \quad \text{jeśli} \quad p \approx \frac{n}{2} \quad i \quad n \gg 1$$

Biblioteka Scikit-Learn zawiera wszystkie te metody (oraz ich odmiany), sugeruję jednak korzystanie zawsze z funkcji pomocniczej `cross_val_score()`, pozwalającej stosować różne metody wobec określonego problemu. W poniższym fragmencie kodu, który bazuje na **wielomianowej maszynie wektorów nośnych** i zestawie danych MNIST, wprowadzamy tę funkcję, definiując liczbę grup (parametr `cv`). W ten sposób biblioteka Scikit-Learn automatycznie wykorzysta warstwowy k-krotny sprawdzian krzyżowy w klasyfikacji kategorialnej, a w pozostałych sytuacjach **standardową k-krotną kroswalidację**:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

data = load_digits()
svm = SVC(kernel='poly')

skf_scores = cross_val_score(svm, data['data'], data['target'], cv=10)

print(skf_scores)
[ 0.96216216 1.         0.93922652 0.99444444 0.98882682 0.98882682
  0.99441341 0.99438202 0.96045198 0.96590909]

print(skf_scores.mean())
0.978864325583
```

Każda grupa cechuje się bardzo dużą dokładnością ($> 0{,}9$), zatem spodziewamy się, że metodą LOO uzyskamy jeszcze większą jej wartość. Korzystamy z 1797 przykładów, dlatego powinniśmy uzyskać tyle samo wyników:

```
from sklearn.model_selection import cross_val_score, LeaveOneOut

loo_scores = cross_val_score(svm, data['data'], data['target'], cv=LeaveOneOut())

print(loo_scores[0:100])
[  1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
   1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
   1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
   1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
   1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
   1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

print(loo_scores.mean())
0.988870339455
```

Zgodnie z oczekiwaniami uzyskana dokładność jest bardzo duża, jednak pewne przykłady są nadal nieprawidłowo klasyfikowane. Jak się wkrótce przekonamy, taka sytuacja może świadczyć o przetrenowaniu, co oznacza, że model jest doskonale dopasowany do zbioru uczącego, ale traci zdolność uogólniania. Metoda LOO nie nadaje się jednak zbytnio do pomiaru zdolności uogólniania z powodu rozmiaru zbioru walidacyjnego.

Sprawdźmy teraz algorytm przy użyciu techniki LPO. Uwzględniwszy dotychczas uzyskane informacje, wybraliśmy mniejszy zestaw danych Iris i klasyfikator bazujący na regresji logistycznej. Korzystamy ze 150 przykładów ($N=150$), zatem po wyborze wartości $p=3$ uzyskujemy 551 300 grup:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, LeavePOut

data = load_iris()

p = 3
lr = LogisticRegression()

lpo_scores = cross_val_score(lr, data['data'], data['target'], cv=LeavePOut(p))

print(lpo_scores[0:100])
[ 1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.          1.          1.          1.          1.          1.
1.
  1.           0.66666667 ...

print(lpo_scores.mean())
0.955668420098
```

Podobnie jak w poprzednim przykładzie, wyświetliliśmy zaledwie pierwsze 100 wyników, jednak już po kilku pierwszych wartościach możemy zauważyć ogólną tendencję.

Technika sprawdzianu krzyżowego stanowi potężne narzędzie, które przydaje się szczególnie wtedy, gdy koszt skuteczności nie jest zbyt duży. Niestety, nie jest ona jednak najlepszym rozwiązaniem w przypadku modeli uczenia głębokiego, cechujących się znacznym rozmiarem zestawu danych, a proces uczenia może trwać nawet wiele dni. Przekonamy się jednak, że w takich sytuacjach prawidłowy wybór (współczynnika podziału zestawu danych) wraz z dokładną analizą zestawów danych i wprowadzeniem takich technik jak regularyzacja czy normalizacja pozwala uzyskiwać modele wykazujące doskonałą zdolność uogólniania.

Cechy modelu uczenia maszynowego

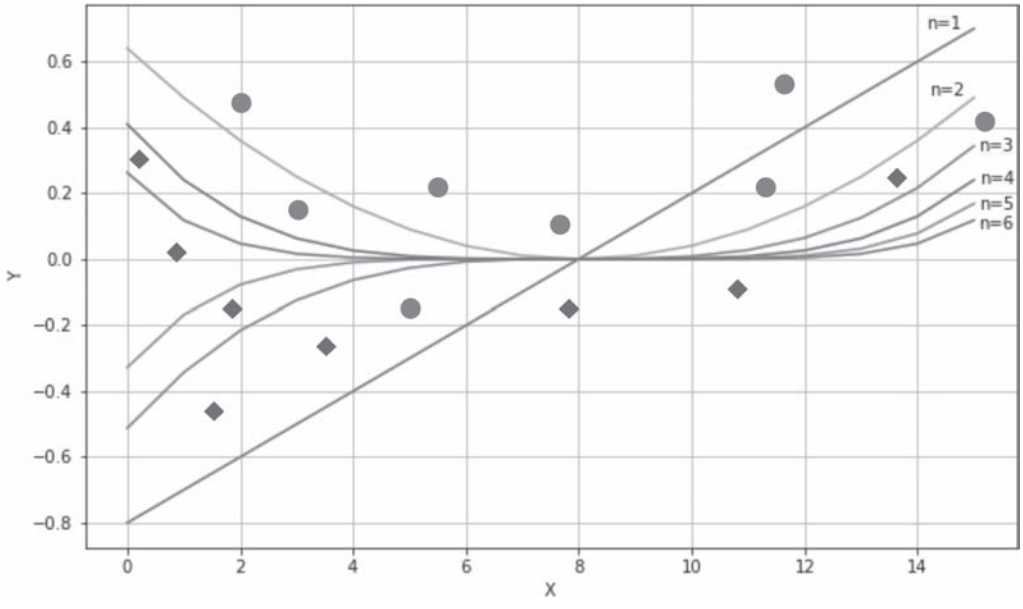
W tym podrozdziale przyjrzymy się modelom nadzorowanym i zastanowimy się, w jaki sposób można mierzyć ich teoretyczną, potencjalną dokładność i zdolność prawidłowego uogólniania dla wszystkich możliwych przykładów zawartych w p_{dane}. Większość omawianych tu koncepcji powstała jeszcze przed nastaniem **ery uczenia głębokiego**, ciągle jednak mają olbrzymi wpływ na projekty badawcze. Na przykład pojęcie **pojemności** (ang. *capacity*) stanowi do dzisiaj otwarte pytanie, jakie neurobiolodzy zadają w kontekście ludzkiego mózgu. Współczesne modele uczenia głębokiego zawierające dziesiątki warstw i miliony parametrów ponownie ożywiły dyskusję w ujęciu matematycznym. Te oraz inne zagadnienia, takie jak granice wariancji estymatora, znowu przyciągają uwagę, ponieważ algorytmy stają się coraz potężniejsze, a skuteczność, która niegdyś pozostawiała wiele do życzenia, daje teraz wielkie możliwości. Współczesny inżynier głębokich sieci neuronowych oczekuje dzisiaj, że jego praca będzie umożliwiać uczenie modelu, a także pełne wykorzystanie jego pojemności, maksymalizację zdolności uogólniania i zwiększenie dokładności w stopniu przewyższającym nawet możliwości ludzkiego mózgu.

Pojemność modelu

Jeżeli będziemy rozważać model nadzorowany jako zbiór parametryzowanych funkcji, to przez dużą **pojemność modelu** (ang. *representational capacity*) rozumiemy, że w zbiorze tym znajdziemy dużą liczbę bardzo różnych rozkładów danych. Aby zrozumieć to pojęcie, weźmy funkcję $f(x)$ różniczkowalną dowolnie wiele razy w pewnym otoczeniu punktu x_0 i rozpiszmy ją w postaci szeregu Taylora:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

Możemy postanowić, aby korzystać z n pierwszych członów, dzięki czemu otrzymamy wielomianową funkcję n-tego stopnia. Na rysunku 1.5 widzimy dwuwymiarowy przykład z sześcioma funkcjami (począwszy od funkcji liniowej). Możemy zobaczyć zmianę ich przebiegu dla małego zestawu punktów danych.



Rysunek 1.5. Przebiegi sześciu różnych, wielomianowych krzywych rozdzielających

Zdolność szybkiego zmieniania krzywizny jest wprost proporcjonalna do stopnia wielomianu. Jeżeli wybierzemy klasyfikator liniowy, będziemy mogli wpływać jedynie na jego nachylenie (cały czas omawiamy przykład w przestrzeni dwuwymiarowej) i punkt przecięcia z osią współrzędnych. Z kolei po wybraniu funkcji wyższego stopnia uzyskujemy większą swobodę w **zaginaniu** krzywizny. Weźmy pod uwagę funkcje o stopniach $n=1$ i $n=2$. W przypadku tej pierwszej możemy uwzględnić punkt (kropkę) widniejący w miejscu $x=11$, jednak ta decyzja będzie miała negatywny wpływ na punkt umieszczony w $x=5$.

Jedynie parametryzowana funkcja nieliniowa jest w stanie skutecznie rozwiązać ten problem, ponieważ wymaga on pojemności potencjalnej większej od zapewnianej przez klasyfikatory liniowe. Innym klasycznym przykładem jest funkcja XOR. Badacze przez długi czas ignorowali koncepcję perceptronów (liniowych sieci neuronowych), ponieważ nie potrafiły one klasyfikować zestawów danych generowanych przy użyciu funkcji XOR. Na szczęście problem ten (i wiele innych, których złożoność wykracza poza możliwości klasycznych modeli uczenia maszynowego) został rozwiązany w momencie wprowadzenia perceptronów wielowarstwowych zawierających funkcje nieliniowe.

Pojemność Vapnika-Chervonenkisa

Pojemność klasyfikatora została matematycznie sformalizowana w postaci **teorii Vapnika-Chervonenkisa**. Zanim przejdziemy do definicji, musimy najpierw wyjaśnić pojęcie **rozbijania** (ang. *shattering*). Jeżeli mamy klasę zbiorów C i zbiór M, to mówimy, że C rozbija M, jeżeli:

$$\forall m_i \subseteq M \exists c_j \in C \Rightarrow m_i = c_j \cap M$$

Innymi słowy dowolny podzbiór M możemy uzyskać jako część wspólną określonego wystąpienia zbioru C (c_j) i samego zbioru M. Jeśli teraz rozważymy model jako funkcję parametryzowaną:

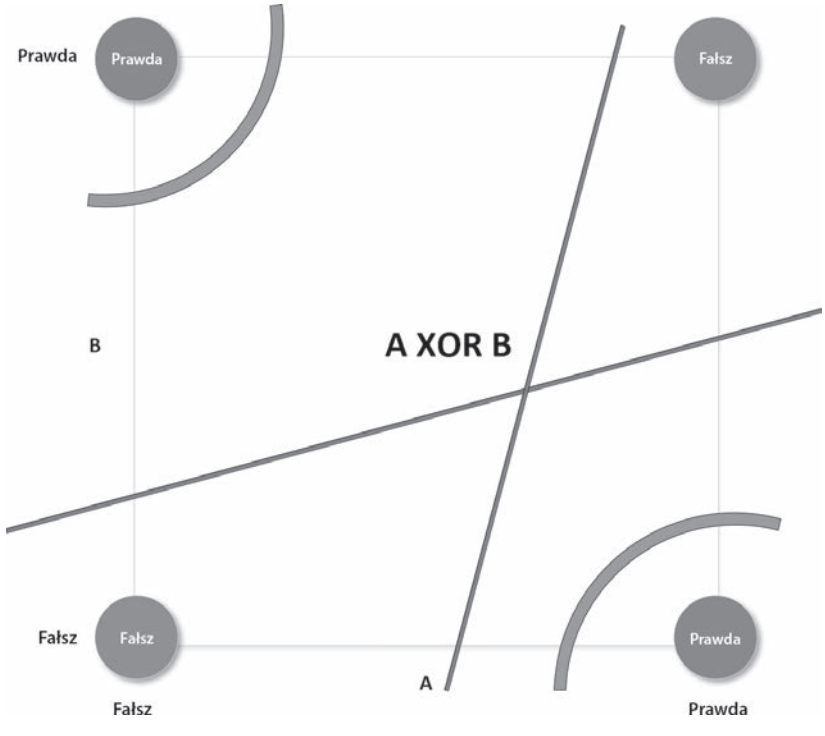
$$C = f\left(\overline{\theta}\right) \quad gdzie \quad \overline{\theta} \in \mathbb{R}^p$$

Chcemy określić jej pojemność w odniesieniu do skończonego zestawu danych X:

$$X = \left\{\overline{x}_0, \overline{x}_1, \ldots, \overline{x}_N\right\} \quad gdzie \quad \overline{x}_i \in \mathbb{R}^k$$

Zgodnie z teorią Vapnika-Chervonenkisa możemy stwierdzić, że model f rozbija zbiór X, jeżeli nie ma błędów klasyfikacji dla każdego możliwego przydziału etykiety. Możemy więc zdefiniować **pojemność Vapnika-Chervonenkisa** (zwaną także **pojemnością VC** lub **wymiarem VC**) jako maksymalną moc podzbioru X, jaką może rozbić model f.

Przykładowo jeśli teraz rozważymy klasyfikator liniowy w przestrzeni dwuwymiarowej, pojemność VC będzie równa 3, ponieważ zawsze możemy oznaczyć trzy przykłady rozbijane przez model f; jest to jednak niemożliwe we wszystkich sytuacjach, gdzie $N > 3$. Kwestia funkcji XOR stanowi przykład modelu wymagającego pojemności VC większej niż 3. Spójrz na rysunek 1.6.



Rysunek 1.6. Problem XOR z ukazanymi różnymi krzywymi rozdzielającymi

Taki określony dobór etykiet sprawia, że ten zestaw staje się nieliniowo rozdzielny. Jedynym sposobem rozwiązania tego problemu jest wprowadzenie funkcji wielomianowych wyższego rzędu (lub funkcji nieliniowych). Krzywe (przynależne do klasyfikatora, którego pojemność VC jest większa od 3) mogą rozdzielać obszary w lewym górnym i prawym dolnym rogu wykresu, co jest niewykonalne w przypadku prostej (może ona zawsze oddzielać jeden punkt od trzech pozostałych).

Obciążenie estymatora

Przyjrzyjmy się teraz parametryzowanemu modelowi zawierającemu jeden parametr wektorowy (nie jest to żadne ograniczenie, lecz wyłącznie nasz wybór w celach dydaktycznych):

$$p\left(X;\overline{\theta}\right)$$

Celem procesu uczenia jest takie oszacowanie parametru θ, żeby (na przykład) zmaksymalizować dokładność klasyfikacji. Zdefiniujmy **obciążenie** (ang. *bias*) estymatora (w odniesieniu do parametru θ):

$$Obciążenie\left[\tilde{\theta}\right] = E_{x|\overline{\theta}}\left[\tilde{\theta}\right] - \overline{\theta} = \left(\sum_{x} \tilde{\theta} p\left(x \mid \overline{\theta}\right)\right) - \overline{\theta}$$

Innymi słowy obciążenie stanowi różnicę pomiędzy wartością oczekiwaną oszacowania a rzeczywistą wartością parametru. Pamiętaj, że oszacowanie jest funkcją zestawu danych X i nie można go tu rozpatrywać jako stałej.

Estymator jest **nieobciążony** (ang. *unbiased*), jeżeli:

$$Obciążenie\left[\tilde{\theta}\right] = 0 \Rightarrow E\left[\tilde{\theta}\right] = \overline{\theta}$$

Co więcej, estymator jest **spójny** (ang. *consistent*), jeżeli sekwencja oszacowań jest zbieżna (przynajmniej z prawdopodobieństwem równym 1) z wartością rzeczywistą, gdy $k \to \infty$:

$$\forall \varepsilon > 0 \quad P\left(\left|\overline{\theta} - \tilde{\theta}_k\right| < \varepsilon\right) \to 1 \quad kiedy \quad k \to \infty$$

Mając dany zestaw danych X, którego przykłady pochodzą z p_{dane}, dokładność estymatora jest odwrotnie proporcjonalna do jego obciążenia. Estymatory nieobciążone (lub o małym obciążeniu) są w stanie odwzorowywać zestaw danych X z dużą precyzją, natomiast estymatory o dużym obciążeniu będą miały prawdopodobnie zbyt małą pojemność, aby rozwiązać dany problem, dlatego ich umiejętność wykrywania wzorców jest niewielka.

Obliczmy teraz pochodną obciążenia w odniesieniu do wektora θ (przyda nam się to w dalszej części książki):

$$\frac{\partial Obciążenie\left[\tilde{\theta}\right]}{\partial\bar{\theta}}=\frac{\partial}{\partial\bar{\theta}}\left(\left(\sum_{x}\tilde{\theta}p\left(x\,|\,\bar{\theta}\right)\right)-\bar{\theta}\right)=\left(\sum_{x}\tilde{\theta}\frac{\partial p\left(x\,|\,\bar{\theta}\right)}{\partial\bar{\theta}}\right)-1=$$

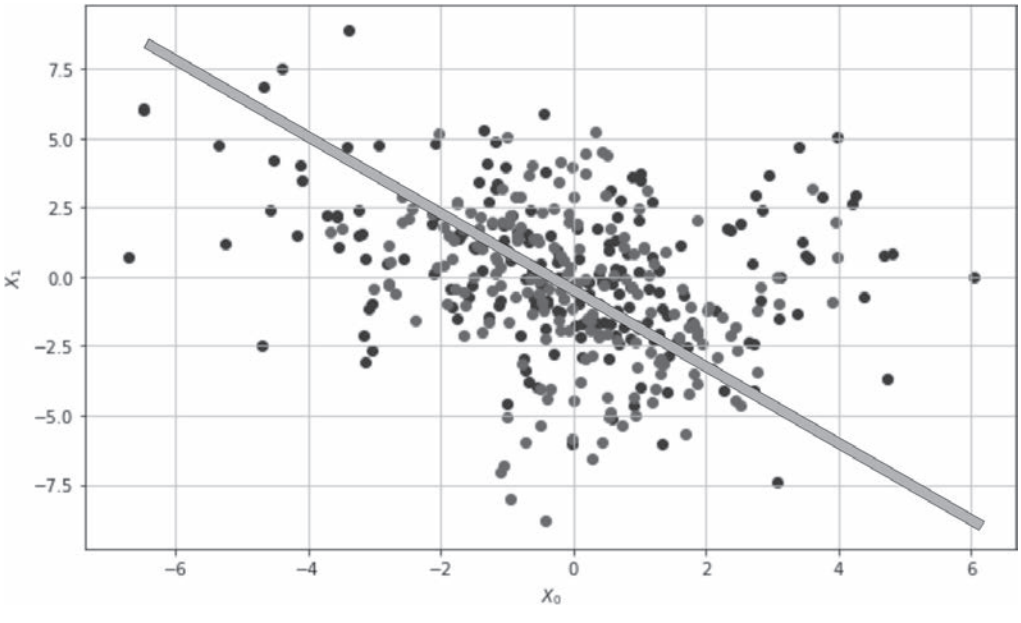
$$=\left(\sum_{x}\tilde{\theta}p\left(x\,|\,\bar{\theta}\right)\frac{\partial\log p\left(x\,|\,\bar{\theta}\right)}{\partial\bar{\theta}}\right)-1=E_{x|\bar{\theta}}\left[\tilde{\theta}\frac{\partial\log p\left(x\,|\,\bar{\theta}\right)}{\partial\bar{\theta}}\right]-1$$

Zauważ, że ostatnie równanie dzięki liniowości $E\left[\bullet\right]$ jest prawdziwe również wtedy, gdy do oszacowania θ dodamy człon niezależny od x. Istotnie zgodnie z prawami prawdopodobieństwa możemy łatwo sprawdzić, że:

$$\sum_{x}\left(\tilde{\theta}+a\right)p\left(x\,|\,\bar{\theta}\right)=\sum_{x}\tilde{\theta}p\left(x\,|\,\bar{\theta}\right)+a\sum_{x}p\left(x\,|\,\bar{\theta}\right)=\sum_{x}\tilde{\theta}p\left(x\,|\,\bar{\theta}\right)$$

Niedotrenowanie

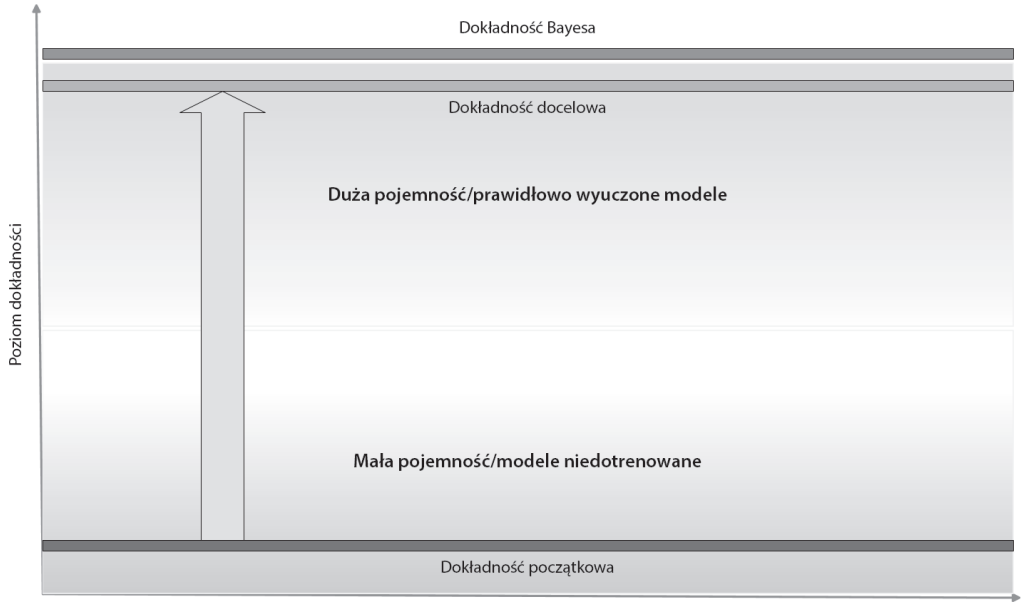
Model o dużym obciążeniu prawdopodobnie będzie niedotrenowany wobec zbioru uczącego. Rozważmy scenariusz zaprezentowany na rysunku 1.7.



Rysunek 1.7. Klasyfikator niedotrenowany: nie jest w stanie prawidłowo rozdzielić obydwu klas

Nawet jeśli problem jest bardzo trudny do rozwiązania, możemy spróbować dostosować model liniowy. Na koniec procesu uczenia nachylenie i punkt przecięcia z osią układu współrzędnych wynoszą mniej więcej odpowiednio: –1 i 0 (co widać na rysunku 1.7). Jeżeli jednak zmierzymy dokładność, okaże się, że jest ona niemal równa 0! Bez względu na liczbę przebiegów model ten nigdy nie rozpozna powiązań pomiędzy zbiorami *X* i *Y*. Zjawisko to nazywamy **niedotrenowaniem** (lub **niedopasowaniem**; ang. *underfitting*) i stanowi ono główny wskaźnik bardzo niskiej dokładności uczenia. Nawet jeśli pewne czynności wstępnego przetwarzania danych mogą ją poprawić, jedynym rozwiązaniem w przypadku niedotrenowanego modelu jest dobór modelu o większej pojemności.

W zadaniach uczenia maszynowego naszym celem jest uzyskanie maksymalnej dokładności, począwszy od zbioru uczącego aż do zbioru walidacyjnego. W bardziej formalnym ujęciu możemy powiedzieć, że chcemy tak usprawniać modele, aby osiągnąć dokładność jak najbardziej zbliżoną do **dokładności Bayesa** (ang. *Bayes accuracy*). Nie jest to wyraźnie zdefiniowana wartość, lecz teoretyczna górna granica dokładności, jaką może osiągnąć dany estymator. Zostało to zilustrowane na rysunku 1.8.



Rysunek 1.8. Schemat poziomów dokładności

Dokładność Bayesa często stanowi czysto teoretyczną granicę i w wielu przypadkach nie można jej osiągnąć nawet za pomocą układów biologicznych, jednak postępy w dziedzinie uczenia głębokiego pozwalają tworzyć modele, których dokładność jest tylko nieco gorsza od dokładności Bayesa. Generalnie nie istnieje jawny wzór pozwalający wyznaczyć dokładność Bayesa, dlatego za wyznacznik służą możliwości ludzkiego mózgu. W poprzednim przykładzie klasyfikacji człowiek błyskawicznie rozróżnia różne typy punktów danych, jednak dla klasyfikatorów o ograniczonej pojemności problem ten może być bardzo skomplikowany. Niektóre z omawianych w tej książce modeli rozwiązują go, osiągając bardzo dużą wartość dokładności. W tym momencie natrafiamy jednak na kolejne wyzwanie, które stanie się dla nas bardziej zrozumiałe po zdefiniowaniu pojęcia wariancji estymatora.

Wariancja estymatora

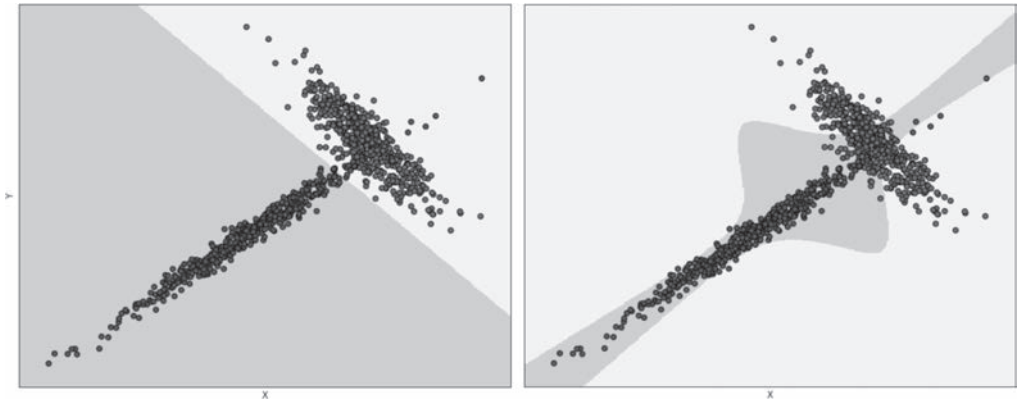
Na początku rozdziału zdefiniowaliśmy proces generowania danych p_{dane} i założyliśmy, że nasz zestaw danych X korzysta z tego rozkładu. Nie chcemy jednak poznawać istniejących związków, ograniczając się wyłącznie do tego zestawu danych, oczekujemy natomiast, że nasz model będzie sprawnie uogólniał prognozy wobec dowolnego podzbioru wygenerowanego za pomocą p_{dane}. Dobrą miarą tego zjawiska jest **wariancja** (ang. *variance*) estymatora:

$$War\left[\tilde{\theta}\right] = BłdStd\left[\tilde{\theta}\right]^2 = E\left[\left(\tilde{\theta} - E\left[\tilde{\theta}\right]\right)^2\right]$$

Możemy również zdefiniować wariancję jako kwadrat błędu standardowego (na drodze analogii z odchyleniem standardowym). Duża wariancja sugeruje znaczne zmiany w dokładności podczas pojawiania się nowych podzbiorów, ponieważ model prawdopodobnie uzyskał bardzo wysoką dokładność uczenia poprzez przetrenowanie na ograniczonym zestawie relacji i niemal całkowicie zatracił umiejętność uogólniania.

Przetrenowanie

Jeżeli niedotrenowanie stanowiło konsekwencję małej pojemności i dużego obciążenia, to **przetrenowanie** (lub **nadmierne dopasowanie**; ang. *overfitting*) jest zjawiskiem powiązanym z dużą wartością wariancji. Generalnie możemy obserwować bardzo dużą dokładność uczenia (być może bliską nawet dokładności Bayesa), która jednak bardzo maleje wobec zbioru walidacyjnego. Oznacza to, że pojemność modelu jest wystarczająco (a może nawet zbyt) duża do danego zadania (wraz ze wzrostem pojemności zwiększa się wartość wariancji), a także że zbiór uczący nie jest wystarczająco reprezentatywny dla p_{dane}. Aby lepiej zrozumieć to zagadnienie, przeanalizuj rysunek 1.9.



Rysunek 1.9. Prawidłowe dopasowanie (po lewej), klasyfikator przetrenowany (po prawej)

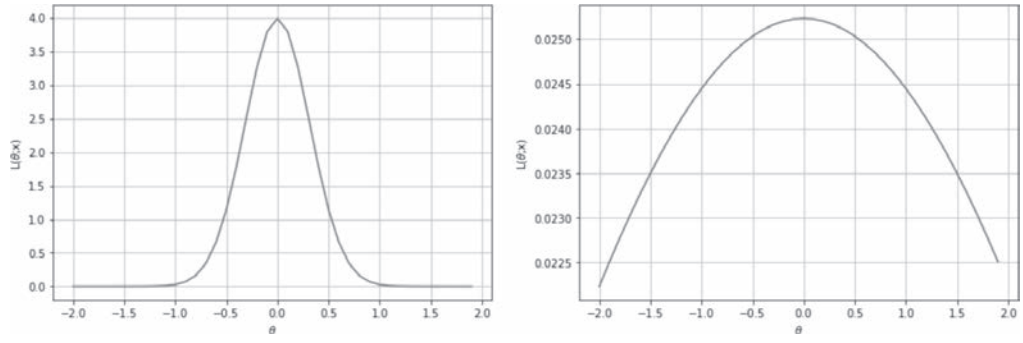
W wykresie po lewej użyliśmy modelu regresji logistycznej, natomiast wykres po prawej uzyskaliśmy za pomocą modelu maszyny wektorów nośnych z jądrem wielomianowym szóstego stopnia. Jeżeli przyjrzymy się temu drugiemu modelowi, zauważymy, że granice decyzyjne są znacznie precyzyjniejsze i odstaje od nich tylko kilka przykładów. Biorąc pod uwagę wymiary obydwu zbiorów danych, moglibyśmy stwierdzić, że nieliniowa maszyna wektorów nośnych lepiej wychwytuje ich dynamikę. Jeśli jednak stworzymy kolejny zestaw danych za pomocą procesu p_{dane}, a diagonalny **ogon** stanie się szerszy, model regresji logistycznej nadal będzie prawidłowo klasyfikować punkty danych, natomiast drastycznie zmaleje dokładność maszyny wektorów nośnych. Drugi model jest bardzo podatny na przetrenowanie i wymagane jest wprowadzanie pewnych poprawek. Gdy dokładność dla zbioru walidacyjnego jest znacznie niższa niż dla zbioru uczącego, dobrym rozwiązaniem staje się zwiększenie liczby przykładów uczących, aby można było uwzględniać rzeczywisty proces p_{dane}. W rzeczywistości bywa czasami tak, że zbiór uczący jest od początku generowany za pomocą hipotetycznego rozkładu mającego niewiele wspólnego z rzeczywistością, ewentualnie liczba przykładów używanych do walidacji jest zbyt duża, co zmniejsza ilość informacji zawartych w pozostałych przykładach. Sprawdzian krzyżowy jest wartościowym sposobem oceniania jakości zestawów danych, zawsze jednak możemy odkryć zupełnie nowe podzbiory (na przykład generowane po wdrożeniu aplikacji do środowiska produkcyjnego), które będą nieprawidłowo klasyfikowane pomimo rzekomej przynależności do p_{dane}. Jeżeli nie mamy możliwości powiększenia zbioru testowego, przydatne może być dogenerowanie danych, ponieważ metoda ta pozwala tworzyć sztuczne przykłady (w przypadku obrazów przeprowadzane są operacje odwracania, obracania lub rozmywania) za pomocą informacji zawartych w istniejących punktach danych. Inne strategie zapobiegające przetrenowaniu bazują na technice zwanej **regularyzacją**, do której powrócimy pod koniec rozdziału. Na razie wystarczy nam informacja, że skutek regularyzacji przypomina częściową linearyzację, co wiąże się ze zmniejszaniem pojemności, a zatem również redukcją wariancji.

Granica Craméra—Rao

Jeżeli istnieje teoretyczna możliwość stworzenia modelu nieobciążonego (nawet asymptotycznie), nie można tego samego powiedzieć o wariancji. Aby zrozumieć tę koncepcję, musimy wprowadzić istotną definicję: **informację Fishera**. Jeśli mamy parametryzowany model i proces generowania danych p_{dane}, możemy wyznaczyć funkcję prawdopodobieństwa przy uwzględnieniu następujących parametrów:

$$L\left(\overline{\theta} \mid X\right) = p\left(X \mid \overline{\theta}\right)$$

Funkcja ta pozwala mierzyć dopasowanie modelu do pierwotnego procesu generowania danych. Przebieg tej funkcji może przyjmować różne kształty, od wyraźnie zarysowanych krzywych wypukłych aż do niemal zupełnych wypłaszczeń. Na rysunku 1.10 widzimy dwa przykłady zależne od jednego parametru.



Rysunek 1.10. Spiczasty (po lewej) i spłaszczony (po prawej) przebieg prawdopodobieństwa

Od razu widzimy, że w pierwszym przypadku możemy łatwo osiągnąć prawdopodobieństwo maksymalne za pomocą wzrostu wzdłuż gradientu, gdyż wykres funkcji jest bardzo stromy. Jednakże w drugiej sytuacji rozmiar gradientu jest mniejszy i szybciej można zatrzymać się przed osiągnięciem maksimum globalnego z powodu nieścisłości numerycznych lub tolerancji. W najgorszym wypadku przebieg funkcji może być niemal płaski na długich odcinkach, co oznacza wartość gradientu zbliżoną do zera. Chcielibyśmy, oczywiście, zawsze pracować na bardzo wyrazistych i stromych funkcjach prawdopodobieństwa, ponieważ zawierają one więcej informacji na temat ich maksimum. Informacja Fishera wyznacza tę wartość w sposób ilościowy. Definiujemy ją następująco dla pojedynczego parametru:

$$I(\theta) = E_{\overline{x}|\theta}\left[\left(\frac{\partial}{\partial\theta}\log p(\overline{x}\,|\,\theta)\right)^2\right]$$

Informacja Fishera jest nieograniczoną, nieujemną liczbą wprost proporcjonalną do ilości informacji przenoszonej przez logarytm prawdopodobieństwa. Wprowadzenie logarytmu nie ma wpływu na wzrost gradientu, ale upraszcza poszczególne wyrażenia, gdyż przekształca iloczyny w sumy. Wartość tę możemy interpretować jako **szybkość** gradientu w dążeniu funkcji do maksimum. Jej większe wartości sugerują zatem lepsze przybliżenia, natomiast hipotetyczna wartość zero wskazywałaby, że również prawdopodobieństwo określenia oszacowania właściwego parametru byłoby zerowe.

Podczas pracy z zestawem K parametrów informacja Fishera staje się dodatnio półokreśloną macierzą:

$$I(\overline{\theta}) = \begin{pmatrix} E_{\overline{x}|\overline{\theta}}\left[\left(\frac{\partial}{\partial\theta_0}\log p(\overline{x}\,|\,\overline{\theta})\right)\left(\frac{\partial}{\partial\theta_0}\log p(\overline{x}\,|\,\overline{\theta})\right)\right] & \cdots & E_{\overline{x}|\overline{\theta}}\left[\left(\frac{\partial}{\partial\theta_0}\log p(\overline{x}\,|\,\overline{\theta})\right)\left(\frac{\partial}{\partial\theta_K}\log p(\overline{x}\,|\,\overline{\theta})\right)\right] \\ \vdots & \ddots & \vdots \\ E_{\overline{x}|\overline{\theta}}\left[\left(\frac{\partial}{\partial\theta_K}\log p(\overline{x}\,|\,\overline{\theta})\right)\left(\frac{\partial}{\partial\theta_0}\log p(\overline{x}\,|\,\overline{\theta})\right)\right] & \cdots & E_{\overline{x}|\overline{\theta}}\left[\left(\frac{\partial}{\partial\theta_K}\log p(\overline{x}\,|\,\overline{\theta})\right)\left(\frac{\partial}{\partial\theta_K}\log p(\overline{x}\,|\,\overline{\theta})\right)\right] \end{pmatrix}$$

Macierz ta jest symetryczna i ma jeszcze jedną ważną własność: gdy wartość wynosi 0, oznacza to, że odpowiednia para parametrów staje się ortogonalna w celu oszacowania maksymalnego prawdopodobieństwa i można je traktować rozdzielnie. W wielu rzeczywistych sytuacjach, jeśli wartość jest zbliżona do zera, oznacza ona małą korelację pomiędzy parametrami i mimo że taka operacja wykracza nieco poza rygor aparatu matematycznego, możliwe okazuje się ich rozłączenie.

W tym momencie możemy wprowadzić **granicę Craméra-Rao** (ang. *Cramér-Rao bound*), zgodnie z którą dla każdego nieobciążonego estymatora, dla którego zbiorem metrycznym jest x (z rozkładem prawdopodobieństwa $p(x; \theta)$), wariancja dowolnego estymatora funkcji θ jest zawsze ograniczona z dołu zgodnie z następującą nierównością:

$$War\left[\tilde{\theta}\right] \geq \frac{1}{I(\theta)}$$

W istocie, jeśli rozważymy początkowo ogólny estymator i powiążemy nierówność Cauchy'ego-Schwarza z wariancją i informacją Fishera (obydwiema wyrażanymi jako wartości oczekiwane), otrzymujemy:

$$E_{\bar{x}|\theta}\left[\left(\tilde{\theta} - E_{\bar{x}|\theta}\left[\tilde{\theta}\right]\right)^2\right] E_{\bar{x}|\theta}\left[\left(\frac{\partial \log p(\bar{x}\,|\,\theta)}{\partial\theta}\right)^2\right] \geq E_{\bar{x}|\theta}\left[\left(\tilde{\theta} - E_{\bar{x}|\theta}\left[\tilde{\theta}\right]\right)\frac{\partial \log p(\bar{x}\,|\,\theta)}{\partial\theta}\right]^2$$

Jeśli teraz wyznaczymy pochodne obciążenia w odniesieniu do θ, biorąc pod uwagę, że wartość spodziewana oszacowania θ nie zależy od x, możemy przekształcić prawą stronę nierówności w poniższy sposób:

$$E_{x|\theta}\left[\left(\tilde{\theta} - E_{x|\theta}\left[\tilde{\theta}\right]\right)\frac{\partial \log p(\bar{x}\,|\,\theta)}{\partial\theta}\right]^2 = \left(\frac{\partial Obciążenie\left[\tilde{\theta}\right]}{\partial\theta} + 1\right)^2$$

W przypadku nieobciążonego estymatora pochodna po prawej stronie jest równa 0, zatem otrzymujemy:

$$War\left[\tilde{\theta}\right] \cdot I(\theta) \geq 1$$

Innymi słowy możemy próbować zmniejszyć wariancję, ale będzie ona zawsze ograniczona z dołu przez odwrotność informacji Fishera. Zatem dla danego zestawu danych i modelu zawsze istnieje granica zdolności uogólniania. W pewnych sytuacjach łatwo jest określić tę miarę. Jej rzeczywista wartość jest jednak teoretyczna, ponieważ wyznacza funkcję prawdopodobieństwa o kolejnej podstawowej własności: przechowuje ona wszystkie informacje potrzebne do oszacowania najgorszego przypadku wariancji. Nie jest to niespodzianką: podczas omawiania pojemności modelu widzieliśmy, w jaki sposób różne funkcje prowadzą do wyższej lub niższej dokładności. Jeżeli dokładność uczenia ma wystarczająco dużą wartość, oznacza to, że pojemność jest wystarczająca lub zbyt duża dla danego problemu. Nie zastanowiliśmy się jednak jeszcze nad rolą prawdopodobieństwa $p(X \mid \theta)$.

Modele o dużej pojemności, zwłaszcza zawierające małe/przechowujące niewiele informacji zestawy danych, mogą prowadzić do płaskich przebiegów funkcji prawdopodobieństwa o wyższej wartości niż w przypadku modeli cechujących się małą pojemnością. Zatem wartość informacji Fishera maleje, ponieważ występuje coraz więcej zbiorów parametrów przechowujących podobne wartości prawdopodobieństwa, a to ostatecznie jest przyczyną wzrostu wariancji i ryzyka przetrenowania. Na koniec tego podrozdziału warto rozważyć ogólną regułę wywodzącą się z zasady zwanej **brzytwą Ockhama**: jeżeli prosty model opisuje dane zjawisko z wystarczającą dokładnością, nie ma sensu zwiększać jego pojemności. Zawsze preferowany jest prostszy model (gdy jego skuteczność jest dobra, a on sam dokładnie reprezentuje dany problem), ponieważ pozwala on zaoszczędzić czas zarówno w fazie uczenia, jak i wnioskowania, a także jest wydajniejszy. Zasadę tę możemy wprowadzić jeszcze precyzyjniej w przypadku sieci uczenia głębokiego, ponieważ łatwiej nam wtedy dostosowywać liczbę warstw i neuronów aż do momentu osiągnięcia pożądanej dokładności.

Funkcje straty i kosztu

Na początku rozdziału przyjrzeliśmy się koncepcji ogólnej funkcji docelowej, którą optymalizujemy w celu rozwiązania problemu uczenia maszynowego. W bardziej formalnym ujęciu (nadzorowanym), gdy mamy skończone zestawy danych X i Y:

$$X = \left\{\overline{x}_0, \overline{x}_1, \ldots, \overline{x}_N\right\} \quad gdzie \quad \overline{x}_i \in \mathbb{R}^k$$
$$Y = \left\{\overline{y}_0, \overline{y}_1, \ldots, \overline{y}_N\right\} \quad gdzie \quad \overline{y}_i \in \mathbb{R}^t$$

możemy zdefiniować ogólną **funkcję straty** (ang. *loss function*) dla pojedynczego przykładu jako:

$$J\left(\overline{x}_i, \overline{y}_i; \overline{\theta}\right) = J\left(f\left(\overline{x}_i, \overline{\theta}\right), \overline{y}_i\right) = J\left(\tilde{y}_i, \overline{y}_i\right)$$

J stanowi funkcję całego zbioru parametrów i musi być ona proporcjonalna do stosunku błędu pomiędzy etykietą rzeczywistą a prognozowaną. Inną jej istotną własnością jest wypukłość. W wielu rzeczywistych sytuacjach warunek ten jest niemal niemożliwy do spełnienia. Zawsze jednak warto szukać wypukłych funkcji straty, ponieważ możemy je z łatwością zoptymalizować metodą gradientu prostego. Przekonamy się o tym w rozdziale 9., „Sieci neuronowe w uczeniu maszynowym". Teraz zaś wystarczy, że będziemy traktować funkcję straty jako produkt pośredni pomiędzy naszym procesem uczenia a idealną optymalizacją matematyczną. Brakującym ogniwem jest tu pełen zestaw danych. Jak już wiemy, zestaw danych X otrzymujemy z procesu p_{dane}, który powinien zawierać rozkład rzeczywisty. Zatem w trakcie minimalizowania funkcji kosztu bierzemy pod uwagę określony podzbiór punktów, a nie pełen rzeczywisty zestaw danych. W wielu przypadkach nie stanowi to ograniczenia, ponieważ przy zerowym obciążeniu i wystarczająco małej wariancji otrzymany model będzie wykazywał wystarczającą zdolność uogólniania (wysoką dokładność dla zbioru uczącego i testowego). Jednak w kontekście procesu generowania danych warto wprowadzić kolejną metrykę — **ryzyko oczekiwane** (ang. *expected risk*):

$$E_{ryzyko}[f] = \int J\left(f\left(\overline{x},\overline{\theta}\right),\overline{y}\right) p_{dane}\left(\overline{x},\overline{y}\right) d\overline{x}d\overline{y}$$

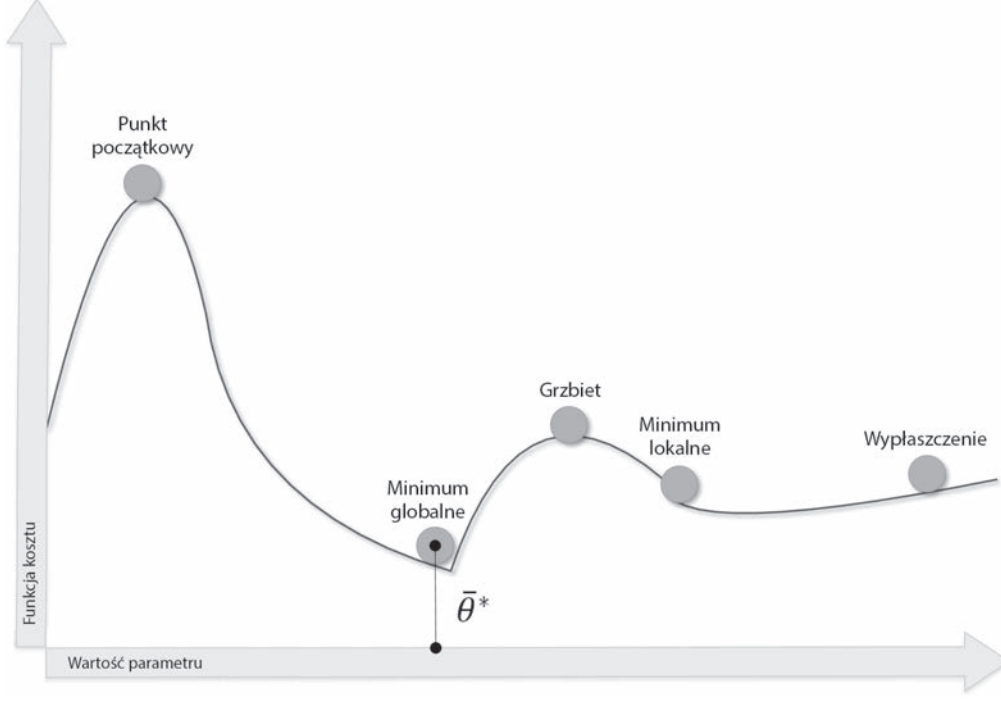
Wartość tę możemy interpretować jako średnią wartość funkcji straty dla wszystkich możliwych przykładów wygenerowanych za pomocą procesu p_{dane}. Minimalizacja ryzyka oczekiwanego prowadzi do maksymalizacji dokładności globalnej. Podczas pracy ze skończoną liczbą przykładów uczących możemy z kolei zdefiniować **funkcję kosztu** (ang. *cost function*; często jest nazywana również funkcją straty i nie należy jej mylić z logarytmiczną funkcją prawdopodobieństwa):

$$L\left(X,Y;\overline{\theta}\right) = \sum_{i=0}^{N} J\left(\overline{x}_i,\overline{y}_i;\overline{\theta}\right)$$

Jest to właściwa funkcja, którą będziemy minimalizować, a po podzieleniu jej przez liczbę przykładów (czynnik ten nie wpływa w żaden sposób na rezultaty) nosi miano **ryzyka empirycznego** (ang. *empirical risk*), ponieważ stanowi ono przybliżenie (bazujące na rzeczywistych danych) ryzyka oczekiwanego. Innymi słowy chcemy znaleźć taki zbiór parametrów, w którym:

$$\overline{\theta}^* = \arg\max_{\overline{\theta}} L\left(X,Y;\overline{\theta}\right)$$

Gdy funkcja kosztu zawiera więcej niż dwa parametry, bardzo trudno (a może wręcz niemożliwe) jest zrozumieć jej strukturę wewnętrzną. Możemy jednak przeanalizować jej pewne warunki potencjalne za pomocą schematu dwuwymiarowego (rysunek 1.11).



Rysunek 1.11. Różne rodzaje punktów na wykresie dwuwymiarowym

Na rysunku 1.11 widzimy następujące obszary:

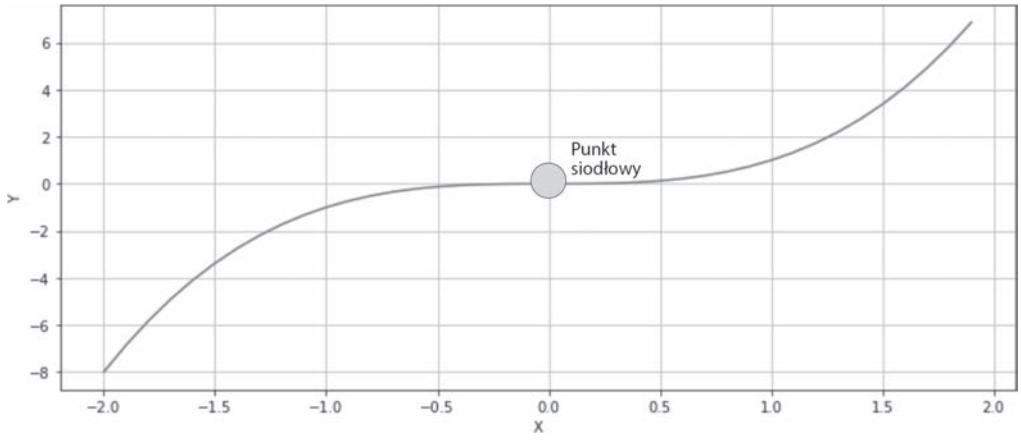
- **punkt początkowy**, w którym z powodu wysokiej wartości błędu funkcja kosztu również jest bardzo duża;
- **minima lokalne**, w których gradient jest zerowy (a druga pochodna przyjmuje wartości dodatnie). Są one kandydatami na optymalne zestawy parametrów, ale niestety, jeśli wypukłość nie jest zbyt duża, ruch wynikający z bezwładności lub jakieś zaszumienie mogą łatwo odsunąć punkt;
- **grzbiety** (**maksima lokalne**), gdzie gradient jest równy 0, a druga pochodna przyjmuje wartości ujemne. Są to niestabilne punkty, ponieważ nawet drobne zaburzenie pozwala uciec z tego obszaru do niżej położonych obszarów;
- **wypłaszczenia** to obszary, w których powierzchnia jest niemal płaska, a gradient ma wartość bliską 0. Jedynym sposobem ucieczki z wypłaszczenia jest utrzymywanie resztowej energii kinetycznej — przyjrzymy się uważniej temu zagadnieniu podczas omawiania neuronowych algorytmów optymalizacji (rozdział 9. „Sieci neuronowe w uczeniu maszynowym”);
- **minimum globalne**, czyli punkt, który chcemy osiągnąć w procesie optymalizowania funkcji kosztu.

Nawet jeśli występowanie minimów lokalnych jest całkiem prawdopodobne w przypadku niewielu parametrów, wraz ze wzrostem liczby tych parametrów maleje szansa na znalezienie tychże obszarów. Istotnie n-wymiarowy punkt θ^* stanowi minimum lokalne funkcji wypukłej (zakładamy tutaj, że L jest funkcją wypukłą) jedynie wtedy, gdy:

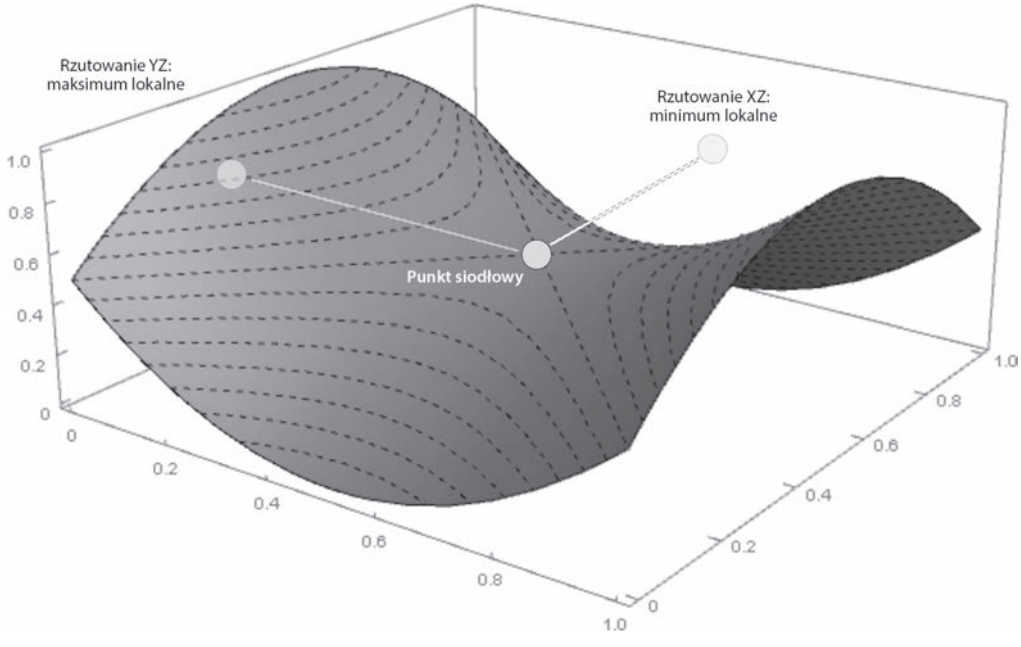
$$\begin{cases} \nabla_\theta L\left(x^*\right) = 0 \\ H_\theta L\left(\theta^*\right) \quad dodatnio \quad półokreślona \end{cases}$$

Drugi warunek wymusza obecność dodatnio półokreślonej macierzy Hessego (równoznacznie wszystkie minory główne H_n z pierwszych n rzędów i n kolumn muszą być nieujemne), zatem wszystkie wartości własne $\lambda_0, \lambda_1, \ldots, \lambda_N$ muszą być nieujemne. Prawdopodobieństwo takiej sytuacji maleje wraz ze wzrostem liczby parametrów (H jest macierzą kwadratową o wymiarach $n \times n$, zawierającą n wartości własnych) i w modelach uczenia głębokiego osiąga wartości bliskie zeru, gdyż liczba wag przyjmuje wartości rzędu 10 000 000 (a nawet więcej). Czytelnik zainteresowany pełnym dowodem matematycznym może zapoznać się z artykułem *High Dimensional Spaces, Deep Learning and Adversarial Examples*, Dube S., arXiv:1801.00634 [cs.CV]. W rezultacie wygodniejszym warunkiem okazuje się obecność **punktu siodłowego** (ang. *saddle point*), w którym wartości własne mają różne znaki, a ortogonalne pochodne kierunkowe są zerowe, nawet jeśli dane punkty nie stanowią lokalnych maksimów ani minimów. Spójrz na rysunek 1.12.

Widoczna na rysunku 1.12 funkcja ma wzór $y=x^3$, a jej pierwsza i druga pochodna przyjmują postać odpowiednio: $y`=3x^2$ i $y``=6x$, zatem $y`(0)=0$ oraz $y``(0)=0$. W takim wypadku (funkcja jednowartościowa) punkt ten nazywany jest **punktem przegięcia** (ang. *point of inflection*), ponieważ w punkcie $x=0$ zmienia się wypukłość funkcji. Łatwiej zrozumieć, dlaczego punkt ten jest tak nazywany, jeżeli przeanalizujemy trójwymiarowy układ współrzędnych. Został on zaprezentowany na rysunku 1.13.



Rysunek 1.12. Punkt siodłowy na wykresie dwuwymiarowym



Rysunek 1.13. Punkt siodłowy na wykresie trójwymiarowym

Widoczna powierzchnia przypomina z kształtu końskie siodło, a jeżeli będziemy rzutować punkt w płaszczyźnie ortogonalnej, w płaszczyźnie XZ będzie on minimum lokalnym, natomiast w wyniku rzutowania go w innej płaszczyźnie (YZ) stanie się on maksimum lokalnym. Punkty siodłowe są dość niebezpieczne, ponieważ wiele prostszych algorytmów może w nich zwolnić, a nawet się zatrzymać, przez co stracą możliwość wyszukiwania właściwego kierunku. W rozdziale 9., „Sieci neuronowe w uczeniu maszynowym", poznamy pewne metody pozwalające uniknąć tego problemu, dzięki czemu modele uczenia głębokiego są w stanie osiągać zbieżność.

Przykładowe funkcje kosztu

W tym punkcie przyjrzymy się najpopularniejszym funkcjom kosztu stosowanym zarówno w zadaniach klasyfikacji, jak i regresji. Niektóre z nich będą mocno eksploatowane w kolejnych rozdziałach, zwłaszcza podczas omawiania procesów uczenia w płytkich i głębokich sieciach neuronowych.

Błąd średniokwadratowy

Błąd średniokwadratowy (ang. *mean-squared error*) to jedna z najczęściej stosowanych funkcji kosztu w regresji. Jej ogólny wzór wygląda następująco:

$$L\left(X,Y;\overline{\theta}\right)=\frac{1}{N}\sum_{i=0}^{N-1}\left(f\left(\overline{x}_i,\overline{\theta}\right)-y_i\right)^2$$

Funkcja ta jest różniczkowalna w każdym punkcie domeny, a także cechuje się wypukłością, dzięki czemu można ją optymalizować za pomocą algorytmu **stochastycznego spadku wzdłuż gradientu** (ang. *stochastic gradient descent* — SGD); traci jednak na skuteczności, jeśli stosujemy ją w zadaniach regresji w obecności przykładów odstających. Jej wartość jest zawsze kwadratowa, dlatego przy dużej odległości pomiędzy przewidywaną a rzeczywistą wartością (czyli przykładem odstającym) względny błąd również jest duży, co może prowadzić do nieakceptowalnych poprawek.

Funkcja Hubera

Jak już wiemy, błąd średniokwadratowy jest wrażliwy na elementy odstające, ponieważ jego wartość jest zawsze kwadratowa niezależnie od odległości pomiędzy wartością rzeczywistą a przewidywaną. Możemy uniknąć tego problemu, stosując **funkcję Hubera** bazującą na progu t_H. Dla odległości mniejszych od tego progu przyjmuje ona postać funkcji kwadratowej, natomiast po przekroczeniu t_H staje się ona liniowa, dzięki czemu redukujemy wartość błędu, a zatem również względną istotność przykładów odstających.

Wzór analityczny prezentuje się następująco:

$$L\left(X,Y;\overline{\theta},t_H\right)=\begin{cases}\frac{1}{2}\sum_{i=0}^{N-1}\left(f\left(\overline{x}_i,\overline{\theta}\right)-y_i\right)^2 & jeżeli \quad \left|f\left(\overline{x}_i,\overline{\theta}\right)-y_i\right|\le t_H\\ t_H\sum_{i=0}^{N-1}\left|f\left(\overline{x}_i,\overline{\theta}\right)-y_i\right|-\frac{t_H}{2} & jeżeli \quad \left|f\left(\overline{x}_i,\overline{\theta}\right)-y_i\right|>t_H\end{cases}$$

Zawiasowa funkcja straty

Ten typ funkcji kosztu jest wykorzystywany w maszynach wektorów nośnych, w których celem jest maksymalizacja odległości pomiędzy granicami rozdzielającymi (na których leżą wektory nośne). Jej wyrażenie analityczne ma następującą postać:

$$L\left(X,Y;\overline{\theta}\right)=\sum_{i=0}^{N-1}\max\left(0,1-f\left(\overline{x}_i,\overline{\theta}\right)\cdot y_i\right)$$

W przeciwieństwie do poprzednich metod ta funkcja kosztu nie jest optymalizowana za pomocą klasycznych technik stochastycznego spadku wzdłuż gradientu, ponieważ nie jest różniczkowalna we wszystkich punktach, dla których:

$$f\left(\overline{x}_i,\overline{\theta}\right)\cdot y_i=1\Rightarrow\max\left(0,0\right)$$

Z tego powodu algorytmy maszyn wektorów nośnych są optymalizowane za pomocą technik programowania kwadratowego.

Kategoryjna entropia krzyżowa

Kategoryjna entropia krzyżowa (ang. *categorical cross-entropy*) stanowi najbardziej rozpowszechnioną funkcję kosztu stosowaną w zadaniach klasyfikacji, gdyż spotykamy ją zarówno w metodach regresji logistycznej, jak i w wielu architekturach neuronowych. Ogólny wzór analityczny wygląda następująco:

$$L\left(X,Y;\overline{\theta}\right)=-\sum_{i=0}^{N-1}y_i\log f\left(\overline{x}_i,\overline{\theta}\right)$$

Ta funkcja kosztu jest wypukła i możemy ją łatwo optymalizować za pomocą technik stochastycznego spadku wzdłuż gradientu (w dodatku istnieje jeszcze jedna istotna jej interpretacja). Jeżeli uczymy klasyfikator, naszym celem jest stworzenie modelu, którego rozkład jest maksymalnie podobny do rozkładu p_{dane}. Warunek ten możemy spełnić poprzez minimalizację rozbieżności Kullbacka-Leiblera pomiędzy dwoma rozkładami:

$$D_{KL}\left(p_{dane}\left\|\tilde{p}_M\right.\right)=\sum_{i=0}^{N-1}p_{dane}\left(\overline{x}_i,y_i\right)\log\frac{p_{dane}\left(\overline{x}_i,y_i\right)}{\tilde{p}_M\left(\overline{x}_i,y_i;\overline{\theta}\right)}$$

W powyższym wzorze p_M stanowi rozkład wygenerowany przez model. Teraz możemy rozpisać rozbieżność w poniższy sposób:

$$D_{KL}\left(p_{dane}\left\|\tilde{p}_M\right.\right)=\sum_{i=0}^{N-1}p_{dane}\left(\overline{x}_i,y_i\right)\log p_{dane}\left(\overline{x}_i,y_i\right)-\sum_{i=0}^{N-1}p_{dane}\left(\overline{x}_i,y_i\right)\log\tilde{p}_M\left(\overline{x}_i,y_i;\overline{\theta}\right)=$$
$$=H\left(p_{dane}\left(\overline{x}_i,y_i\right)\right)+H\left(p_{dane}\left(\overline{x}_i,y_i\right),\tilde{p}_M\left(\overline{x}_i,y_i;\overline{\theta}\right)\right)$$

Pierwszy człon to entropia rozkładu generowanych danych — jest on niezależny od parametrów modelu; z kolei drugim członem jest entropia krzyżowa. Zatem zmniejszając jej wartość, minimalizujemy również rozbieżność Kullbacka-Leiblera, dzięki czemu zmuszamy model do odwzorowania rozkładu bardzo zbliżonego do p_{dane}. Jest to bardzo trafne wyjaśnienie, dlaczego entropia krzyżowa stanowi doskonały wybór w zadaniach klasyfikacji.

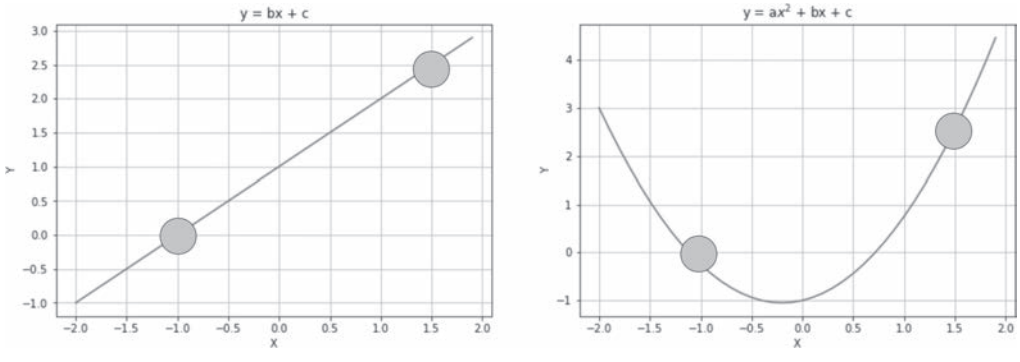
Regularyzacja

Jeśli model ma niewłaściwie dobrane warunki lub jest wrażliwy na przetrenowanie, **regularyzacja** zapewnia pewne metody umożliwiające rozwiązanie tych problemów. Z perspektywy matematycznej regularyzator stanowi karę dodaną do funkcji kosztu, przez co nakłada dodatkowy warunek na ewoluowanie parametrów:

$$L_R\left(X,Y,\overline{\theta}\right)=L\left(X,Y,\overline{\theta}\right)+\lambda g\left(\overline{\theta}\right)$$

Parametr λ reguluje siłę regularyzacji, wyrażaną poprzez funkcję $g(\theta)$. Jej podstawowy warunek wymaga, żeby funkcja ta była różniczkowalna w taki sposób, aby nowa funkcja złożona kosztu była nadal optymalizowana za pomocą algorytmów stochastycznego spadku wzdłuż gradientu. Generalnie możemy używać dowolnej funkcji, w praktyce jednak potrzebujemy funkcji zdolnej do ograniczania nieskończonego powiększania się wartości parametrów.

Zasada ta została ukazana na rysunku 1.14.



Rysunek 1.14. Interpolacja za pomocą prostej (po lewej) i paraboli (po prawej)

Na lewym wykresie model jest liniowy i zawiera dwa parametry, natomiast na prawym wykresie ma on postać kwadratową i opisują go trzy parametry. Wiemy już, że drugi przypadek jest bardziej narażony na przetrenowanie, jeśli jednak wprowadzimy człon regularyzacji, możemy uniknąć wzrostu wartości (pierwszego parametru kwadratowego), dzięki czemu przekształcimy model w formę liniową. Oczywiście, istnieje różnica pomiędzy wyborem modelu o małej pojemności a zastosowaniem regularyzacji. W tej pierwszej sytuacji rezygnujemy z możliwości oferowanych przez większą pojemność i narażamy się na zwiększone ryzyko obciążenia, natomiast w przypadku regularyzacji pozostawiamy wybrany model, ale optymalizujemy go tak, aby zmniejszyć wariancję. Przyjrzyjmy się teraz najczęściej stosowanym technikom regularyzacji.

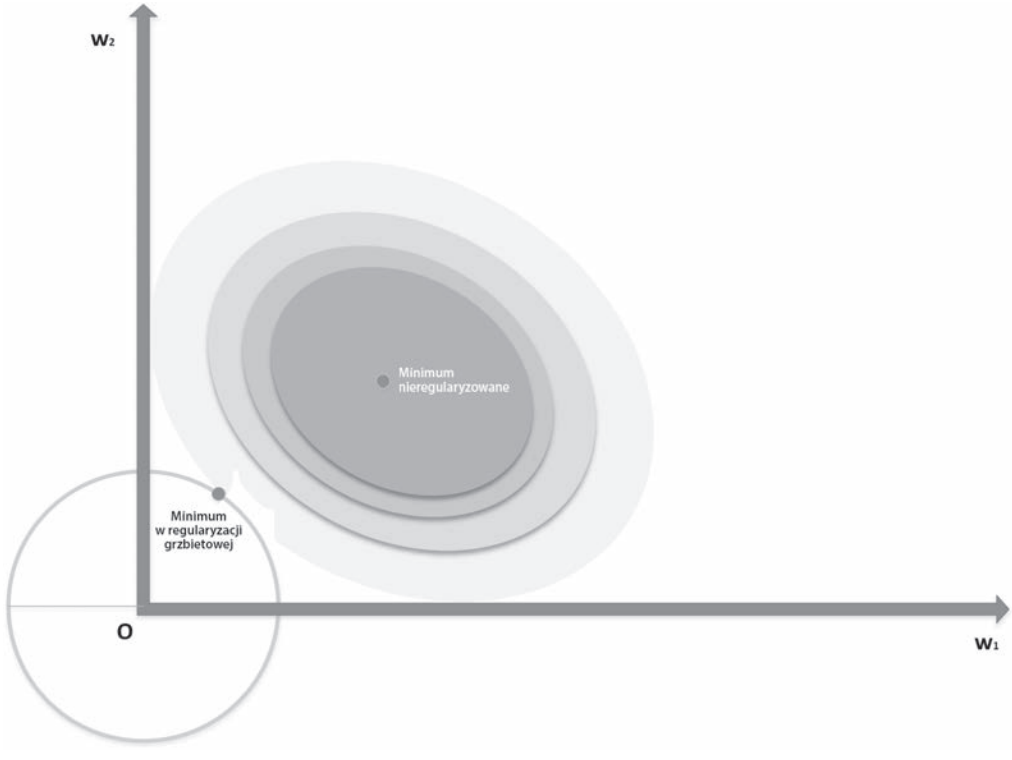
Regularyzacja grzbietowa

Regularyzacja grzbietowa (ang. *ridge regularization*; zwana jest także **regularyzacją Tichonowa**) bazuje na podniesionej do kwadratu normie L2 wektora parametrów:

$$L_R\left(X,Y;\overline{\theta}\right)=L\left(X,Y;\overline{\theta}\right)+\lambda\left\|\overline{\theta}\right\|_2^2$$

Kara ta zapobiega nieskończonemu wzrostowi wartości parametrów (z tego powodu znana jest także jako **kurczenie wag**; ang. *weight shrinkage*) i przydaje się zwłaszcza w źle uwarunkowanych modelach lub w których mamy do czynienia ze znaczną współliniowością wynikającą z występowania przykładów całkowicie niezależnych względem siebie (względnie często spotykana sytuacja).

Rysunek 1.15 prezentuje schematyczną reprezentację regresji grzbietowej na wykresie dwuwymiarowym.



Rysunek 1.15. Regresja grzbietowa (L2)

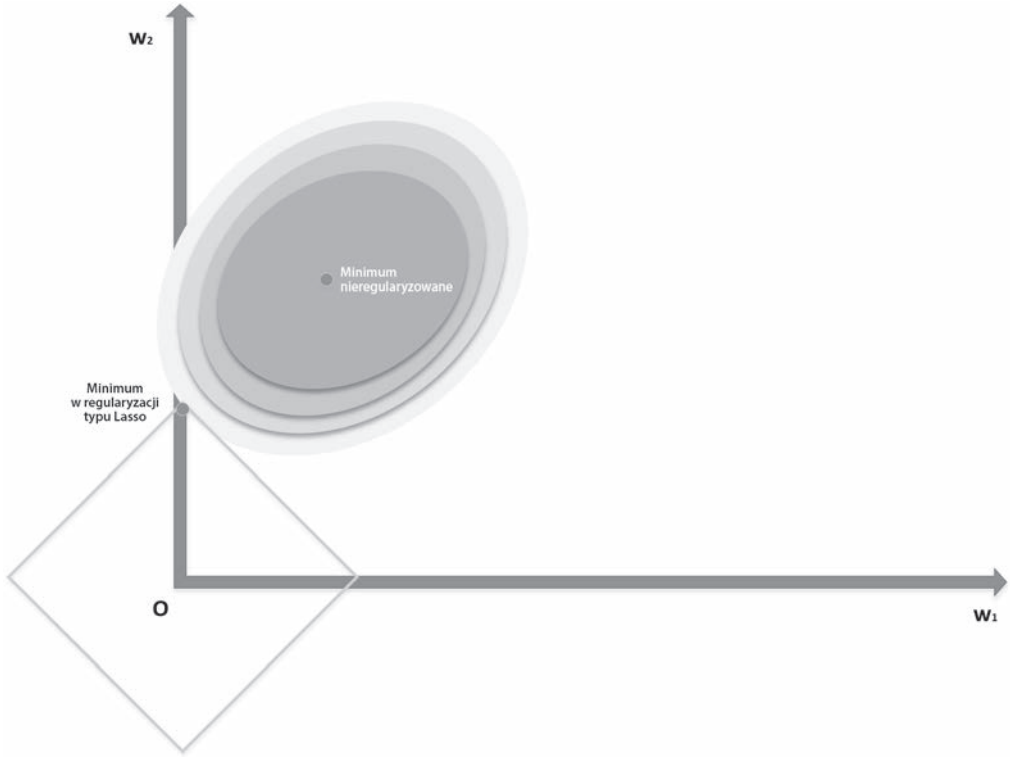
Okrąg wyrysowany wokół środka układu współrzędnych reprezentuje granicę regularyzacji grzbietowej, natomiast wypełniona powierzchnia przedstawia pierwotną funkcję kosztu. Bez stosowania regularyzacji minimum (w_1, w_2) ma wartość (np. odległość od środka układu współrzędnych) niemal dwukrotnie większą od uzyskiwanej za pomocą optymalizacji grzbietowej, co potwierdza oczekiwane kurczenie. Jeżeli wprowadzimy tę regularyzację do metod regresji wyliczanych przy użyciu **metody najmniejszych kwadratów** (ang. *Ordinary Least Squares* — OLS), możemy udowodnić, że zawsze istnieje współczynnik grzbietowy, dzięki któremu wagi są zmniejszane w stosunku do wartości wyliczanych metodą OLS. Taki sam wynik (przy wprowadzeniu pewnych ograniczeń) można rozszerzyć na inne funkcje kosztu.

Regularyzacja typu Lasso

Regularyzacja typu **Lasso** bazuje na normie L1 wektora parametrów:

$$L_R\left(X,Y;\overline{\theta}\right)=L\left(X,Y;\overline{\theta}\right)+\lambda\left\|\overline{\theta}\right\|_1$$

W przeciwieństwie do regularyzacji grzbietowej, w której kurczone są wszystkie wagi, w metodzie Lasso tylko najmniejsza waga zostaje wyzerowana, dzięki czemu uzyskujemy wektor rzadki parametrów. Wyprowadzenie dowodu matematycznego wykracza poza zakres niniejszej książki, możemy jednak zrozumieć intuicyjnie tę koncepcję, spoglądając na rysunek 1.16.



Rysunek 1.16. Regularyzacja typu Lasso (L1)

Kwadrat wyrysowany wokół środka układu współrzędnych symbolizuje granice regularyzacji typu Lasso. Prawdopodobieństwo, że dana prosta będzie styczna do kwadratu, jest większe na jego wierzchołkach, gdzie przynajmniej jeden parametr (w przypadku dwuwymiarowym dokładnie jeden) jest równy 0. Generalnie, jeżeli mamy wektorową funkcję wypukłą *f(x)* (w rozdziale 5., „Algorytm EM i jego zastosowania", definiujemy pojęcie wypukłości), możemy napisać:

$$g(\overline{x}) = f(\overline{x}) + \left\|\overline{x}\right\|_p$$

Ponieważ każda norma L_p jest wypukła, to również suma funkcji wypukłych, *g(x)*, daje funkcję wypukłą. Człon regularyzacji jest zawsze nieujemny, zatem minimum odpowiada normie wektora zerowego. Podczas minimalizowania funkcji *g(x)* musimy również brać pod uwagę wpływ gradientu normy w okrągłym obszarze, którego środkiem jest środek układu współrzędnych, nie występują tu jednak pochodne cząstkowe. Wraz ze wzrostem wartości p norma staje się coraz bardziej wygładzona w okolicach środka układu współrzędnych, a pochodne cząstkowe dążą do zera dla $|x_i| \rightarrow 0$.

Z drugiej strony dla $p=1$ (wyłączając normę L_0 i wszystkie normy $p \in [0,1]$, gwarantujące jeszcze większą rzadkość, ale niewypukłe) pochodne cząstkowe przyjmują zawsze wartość +1 lub –1, zgodnie ze znakiem x_i ($x_i \neq 0$). Łatwiej więc normie L_1 redukować najmniejsze składowe do zera, ponieważ wpływ minimalizacji (np. metodą gradientu prostego) jest niezależny od x_i, podczas gdy norma L_2 zmniejsza jej szybkość w miarę zbliżania się do środka układu współrzędnych. Jest to luźne wyjaśnienie rzadkości osiąganej za pomocą normy L_1. W rzeczywistości musimy również uwzględniać człon *f(x)* ograniczający wartość minimum globalnego, być może jednak koncepcja ta stała się już zrozumiała dla Czytelnika. Jej formalny opis znajdziemy w następującej pozycji: *Optimization for Machine Learning*, Sra S. (red.), Nowozin S., Wright S. J., The MIT Press.

Regularyzacja typu Lasso jest szczególnie przydatna w sytuacjach wymagających przedstawiania danych w postaci rzadkiej. Moglibyśmy chcieć na przykład wyszukiwać wektory cech odpowiadające grupie rysunków. Mielibyśmy do czynienia z dużym zestawem cech, ale tylko mały ich podzbiór opisywałby poszczególne rysunki, dlatego wprowadzenie regularyzacji typu Lasso powoduje, że wszystkie najmniejsze współczynniki zostaną wyzerowane, co pozwala uniknąć występowania cech drugorzędnych. Innym potencjalnym zastosowaniem tej metody jest ukryta analiza semantyczna, w której naszym celem jest opisywanie tekstu należącego do korpusu przy użyciu ograniczonej liczby tematów. Wszystkie te metody możemy umieścić w rodzinie technik zwanych **kodowaniem rzadkim** (ang. *sparse coding*), której celem jest zredukowanie wymiarowości zestawu danych (również w zadaniach nieliniowych) poprzez wydobywanie najistotniejszych elementów i stosowanie różnych sposobów uzyskiwania rzadkości.

Regularyzacja typu ElasticNet

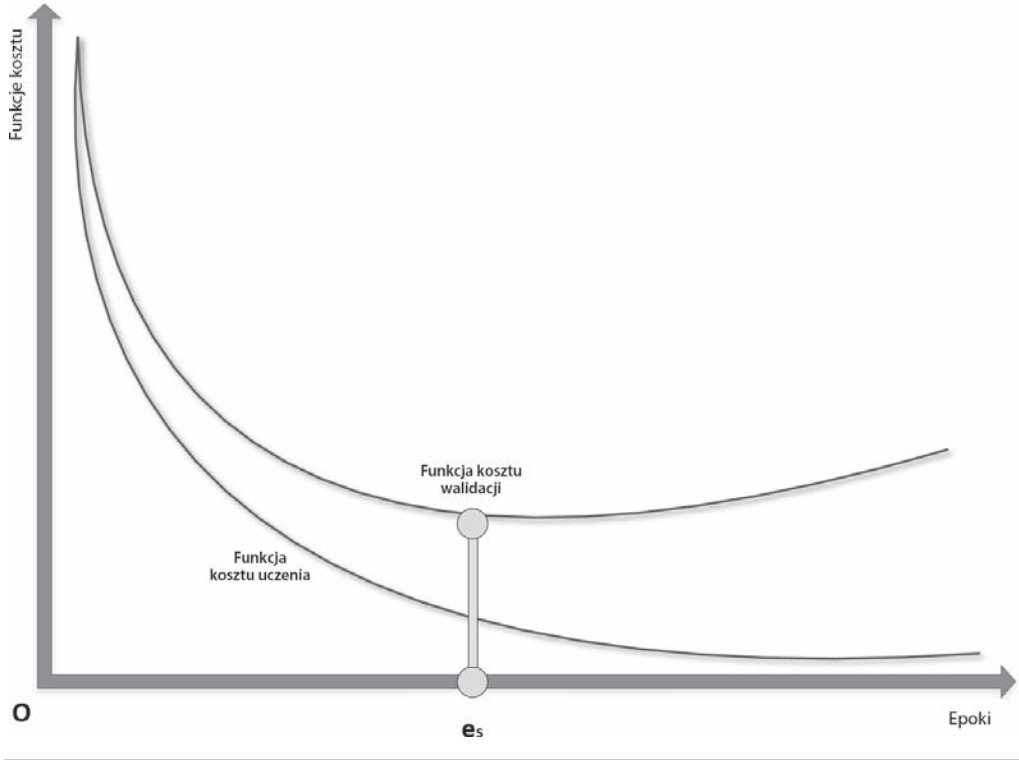
W wielu rzeczywistych sytuacjach warto stosować jednocześnie zarówno regularyzację grzbietową, jak i typu Lasso w celu wymuszenia kurczenia wag i rzadkości globalnej. Możliwość tę daje nam algorytm **ElasticNet** (zwany także **siatką elastyczną**). Regularyzacja ta jest definiowana w następujący sposób:

$$L_R\left(X,Y;\overline{\theta}\right)=L\left(X,Y;\overline{\theta}\right)+\lambda_1\left\|\overline{\theta}\right\|_2^2+\lambda_2\left\|\overline{\theta}\right\|_1$$

Siła każdego typu regularyzacji jest kontrolowana za pomocą parametrów λ_1 i λ_2. Metoda ElasticNet pozwala uzyskiwać znakomite rezultaty w każdej sytuacji wymagającej unikania efektów przetrenowania przy jednoczesnym wprowadzaniu rzadkości. Wszystkie omawiane typy regularyzacji wykorzystamy podczas omawiania niektórych architektur uczenia głębokiego.

Wczesne zatrzymywanie

Metoda **wczesnego zatrzymywania** (ang. *early stopping*) nie jest typową techniką regularyzacji, ale często uznajemy ją za **ostatnią deskę ratunku** w sytuacjach, gdy zawodzą pozostałe techniki zapobiegania przetrenowaniu oraz maksymalizowania dokładności walidacji. W wielu przypadkach (przede wszystkim w zadaniach uczenia głębokiego) możemy zaobserwować typowe zachowanie procesu uczenia w odniesieniu do funkcji kosztu, zarówno dla zbioru uczącego, jak i walidacyjnego (rysunek 1.17).



Rysunek 1.17. Przykład zastosowania wczesnego zatrzymywania przed początkiem wznoszenia się krzywej u-kształtnej

W trakcie pierwszych epok wartości kosztu maleją, może się jednak zdarzyć, że po przekroczeniu **progowej** epoki e_s koszt walidacji zacznie rosnąć. Jeżeli będziemy kontynuować proces uczenia, przetrenujemy model względem zbioru uczącego i zwiększymy wariancję. Z tego powodu możemy przedwcześnie zatrzymać proces trenowania, jeżeli nie mamy do dyspozycji innych metod. Aby tego dokonać, musimy zapisać ostatni wektor parametrów przed rozpoczęciem nowego przebiegu i jeżeli dokładność modelu zaczyna się pogarszać, przerwać proces i wczytać najnowszy zbiór parametrów. Jak już wspomnieliśmy, procedury tej nie należy nigdy uznawać za najlepsze rozwiązanie, ponieważ lepsze rezultaty możemy osiągnąć przy użyciu innego modelu lub lepszego zestawu danych. Technika wczesnego zatrzymywania nie daje możliwości weryfikowania alternatywnych rozwiązań, dlatego należy ją stosować na ostatnim etapie procesu, nigdy zaś na jego początku. Wiele środowisk uczenia głębokiego (np. biblioteka Keras) zawiera funkcje pomocnicze implementujące wywołania wczesnego zatrzymywania; zawsze jednak należy się upewnić, że został zapisany najnowszy wektor parametrów w epoce poprzedzającej próg e_s. W takiej sytuacji warto byłoby powtórzyć proces uczenia i zatrzymać go tuż przed epoką e_s (epoką, w której uzyskaliśmy minimalną wartość dla zbioru walidacyjnego).

Podsumowanie

W niniejszym rozdziale przeanalizowaliśmy podstawowe pojęcia spotykane niemal we wszystkich modelach uczenia maszynowego. Pierwszą część poświęciliśmy procesowi generowania danych jako uogólnieniu skończonego zestawu danych. Poznaliśmy najpowszechniejsze strategie rozdzielania skończonego zestawu danych na podzbiór uczący i walidacyjny, a także poznaliśmy koncepcję sprawdzianu krzyżowego oraz jego najważniejszych odmian jako jednego z najlepszych sposobów unikania ograniczeń statycznego podziału danych.

W części drugiej przyjrzeliśmy się najważniejszym własnościom estymatora: pojemności, obciążeniu i wariancji. Przytoczyliśmy teorię Vapnika-Chervonenkisa, stanowiącą matematyczną formalizację pojęcia pojemności potencjalnej, a także przeanalizowaliśmy wpływ dużej wariancji i dużego obciążenia. W szczególności naszą uwagę skupiły zjawiska przetrenowania i niedotrenowania, ustalające związek pomiędzy wariancją a obciążeniem.

Trzecia część stanowiła wprowadzenie do funkcji straty i kosztu, najpierw jako wskaźników ryzyka oczekiwanego, a następnie ukazaliśmy pewne sytuacje występujące w trakcie rozwiązywania problemu optymalizacji. Przyjrzeliśmy się również najpopularniejszym funkcjom kosztu i opisaliśmy ich główne właściwości. Ostatnia część rozdziału to opis regularyzacji — procesu umożliwiającego unikanie skutków przetrenowania.

W rozdziale 2., „Wprowadzenie do uczenia półnadzorowanego”, omówimy techniki uczenia półnadzorowanego, gdzie skoncentrujemy naszą uwagę na pojęciach uczenia transdukcyjnego i indukcyjnego.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

Ś

T

U

W

Z

Uczenie maszynowe — już dziś zaimplementuj rozwiązania przyszłości!

Imponujący rozwój standardowych algorytmów przy ciągłej obniżce cen sprzętu i udostępnianiu coraz to szybszych komponentów przyczynił się do zrewolucjonizowania wielu gałęzi przemysłu. Obecnie uczenie maszynowe pozwala automatyzować procesy, które do niedawna musiały być zarządzane przez człowieka. Zadania, które jeszcze dekadę temu stanowiły nieprzekraczalną przeszkodę, dziś są wykonywane przez zwykły komputer osobisty. W efekcie dzięki technologii oraz dostępnym wysokopoziomowym otwartym platformom każdy, kto zainteresuje się uczeniem maszynowym, może projektować i wdrażać niezwykle potężne modele.

Celem tej książki jest przybliżenie profesjonalistom tajników złożonych algorytmów uczenia maszynowego i zasad ich stosowania w praktyce. Poza praktycznymi informacjami dotyczącymi działania algorytmów i ich wdrożeń znalazły się tu również niezbędne podstawy teoretyczne. Opisano klasyczne modele uczenia nadzorowanego, nienadzorowanego i półnadzorowanego. Wskazano, w jakich sytuacjach okazują się one najbardziej przydatne. Zaprezentowano techniki wydobywania danych za pomocą modeli bayesowskich, algorytmu MCMC, a także dzięki stosowaniu ukrytych modeli Markowa. Omówiono zestaw przydatnych do uczenia maszynowego narzędzi, takich jak biblioteki: scikit-learn, Keras i TensorFlow.

Najciekawsze zagadnienia:

- najważniejsze koncepcje teoretyczne uczenia maszynowego
- modelowanie probabilistyczne i uczenie hebbowskie
- zaawansowane koncepcje modeli neuronowych
- modele generatywne, takie jak splotowe sieci GAN i sieci Wassersteina
- głębokie sieci przekonań
- zaawansowane algorytmy: TD(λ), aktor-krytyk, SARSA i Q-uczenie

Giuseppe Bonaccorso — od wielu lat prowadzi projekty dotyczące sztucznej inteligencji. W kręgu jego głównych zainteresowań znajdują się takie techniki jak uczenie maszynowe, uczenie głębokie, uczenie przez wzmacnianie, a także praca z wielkimi zbiorami danych, systemy adaptacyjne inspirowane układami biologicznymi, kryptowaluty i programowanie neurolingwistyczne.

ISBN 978-83-283-5245-2
9 788328 352452
Cena: 89,00 zł

Packt