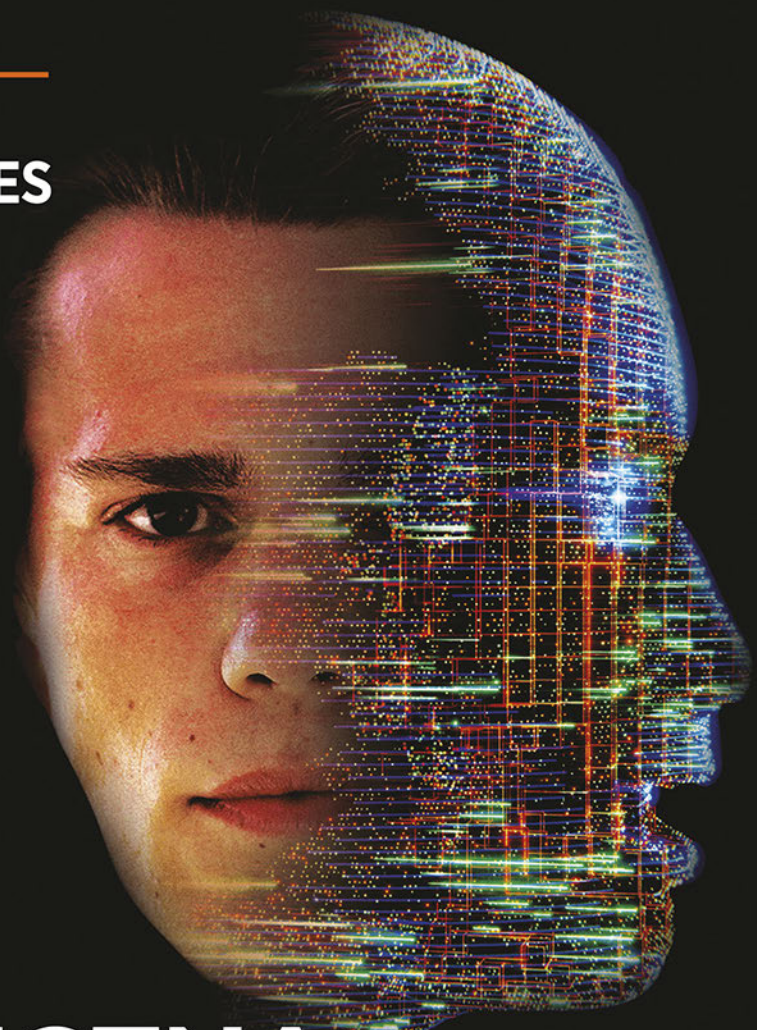
HADELIN
DE PONTEVES



SZTUCZNA INTELIGENCJA

Błyskawiczne wprowadzenie do uczenia maszynowego, uczenia ze wzmocnieniem i uczenia głębokiego

Helion

Packt>

Tytuł oryginału: AI Crash Course A fun and hands-on introduction to machine learning, reinforcement learning, deep learning, and artific

Tłumaczenie: Łukasz Wójcicki

ISBN: 978-83-283-7478-2

Helion S.A.
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:
https://ftp.helion.pl/przyklady/sztinb.zip

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

Printed in Poland.

Spis treści

5

ROZDZIAŁ

Twój pierwszy model AI — uważaj na bandytów!

W tym rozdziale poznasz swój pierwszy model AI! Stworzysz model, który rozwiąże bardzo dobrze znany problem wielorękich bandytów. Jest to klasyczny problem w sztucznej inteligencji, często spotykany w wielu rzeczywistych problemach biznesowych.

Problem wielorękiego bandyty

Wyobraź sobie, że jesteś w Las Vegas, w swoim ulubionym kasynie. Znajdujesz się w pomieszczeniu zawierającym pięć automatów do gry. W przypadku każdego z nich gra jest taka sama: stawiasz określoną sumę pieniędzy, powiedzmy 1 dolara, pociągasz za ramię, a wtedy maszyna albo odbiera Twoje pieniądze, albo zwraca dwa razy więcej pieniędzy. Pamiętasz nagrody, o których rozmawialiśmy w poprzednim rozdziale? Powiedzmy, że jeśli maszyna zabierze Twoje pieniądze, Twoja nagroda wynosi –1, a jeśli maszyna zwróci Ci dwa razy więcej pieniędzy, Twoja nagroda to +1.

Jak widać, zaczynasz już definiować środowisko sztucznej inteligencji, co, przypomnę, jest absolutnie fundamentalne podczas rozwiązywania problemu z użyciem AI. Jak dotąd sztucznej inteligencji nie ma, ale wkrótce nadejdzie. Zawsze zaczynasz od zdefiniowania środowiska.

Zdefiniowałeś nagrody; później zdefiniujesz stany (wejścia) i akcje (wyjścia). Powiedzmy, że teraz, wciąż w trakcie definiowania środowiska, w jakiś sposób wiesz, że w przypadku jednej z tych maszyn prawdopodobieństwo, że da ci premię +1, gdy pociągniesz ją za ramię, jest większe. Nie ma znaczenia, skąd znasz te informacje, ale musi to być część założeń dotyczących problemu. Zapewniam Cię, że to założenie jest zawsze naturalnie weryfikowane w realnych problemach biznesowych wspomnianych powyżej, do których można zastosować problem wielorękiego bandyty.

Twoim celem, podobnie jak w każdym środowisku sztucznej inteligencji, jest zdobycie jak najwyższej nagrody w czasie gry. Załóżmy, że zamierzasz postawić łącznie 1000 dolarów, co oznacza, że za każdym razem zamierzasz postawić 1 dolara 1000 razy, przez pociągnięcie za ramię któregokolwiek z tych pięciu automatów do gry. Pytanie brzmi:

Jaka powinna być Twoja strategia, aby po rozegraniu 1000 kolejek uzyskać maksymalną ilość pieniędzy?

Pierwszym krokiem Twojej strategii musi być ustalenie, który z pięciu automatów ma, w minimalnej liczbie gier, największą szansę na jedną nagrodę. Innymi słowy, musisz szybko znaleźć automat z najwyższym wskaźnikiem sukcesu. Następnie, gdy tylko to zrozumiesz, po prostu musisz grać dalej na tym najbardziej udanym automacie.

Znalezienie najlepszego automatu do gry nie jest trudne; jedną prostą strategią może być zagranie 100 razy na każdym z tych pięciu automatów do gry, a na koniec sprawdzenie, który z nich wyrzucił więcej pieniędzy. Statystycznie daje to dużą szansę na znalezienie najbardziej hojnego automatu.

Całe wyzwanie jest „szybkie". Najtrudniejsze jest znalezienie najlepszego automatu **w minimalnej liczbie prób**. Tutaj do gry wkracza Twój pierwszy model AI.

Model próbkowania Thompsona

Od razu zbudujesz ten model. Teraz stworzysz prostą implementację tej metody, a później zobaczysz stojącą za nią teorię. Przejdźmy do dzieła!

Jak już określiliśmy, naszym problemem jest znalezienie najlepszego automatu, który zapewnia największą szansę na wygraną, spośród wielu. Niezbyt optymalnym rozwiązaniem byłoby rozegranie 100 rund na każdym z automatów do gry i sprawdzenie, który z nich ma najwyższy współczynnik wygranych. Lepszym jest metoda o nazwie próbkowanie Thompsona.

Nie będę się zagłębiać w teorię, która za tym stoi; omówimy to później. Na razie wystarczy powiedzieć, że próbkowanie Thompsona używa funkcji rozkładu (zostanie objaśniona w dalszej części tego rozdziału), zwanej Beta, która przyjmuje dwa argumenty. Dla uproszczenia załóżmy, że im wyższy jest pierwszy argument, tym lepszy jest nasz automat, a im wyższy drugi argument, tym automat jest gorszy.

Dlatego możemy zdefiniować tę funkcję jako:

$$x = \beta(a,b)$$

gdzie:

- x — losowy wybór z naszej funkcji rozkładu Beta,
- β — nasza funkcja Beta,
- a — pierwszy argument,
- b — drugi argument.

Nie martw się, jeśli jeszcze tego do końca nie rozumiesz; przeczytasz o tym później.

Kodowanie modelu

Zacznijmy kodować nasze rozwiązanie. Cały ten kod jest również dostępny na stronie GitHuba tej książki w folderze *Chapter 05*. Przechodzimy do pierwszej sekcji kodu:

```
# Import bibliotek
import numpy as np
```

Będziesz potrzebować tylko jednej biblioteki, o nazwie NumPy. Jest to bardzo przydatna biblioteka, pomocna, gdy mamy do czynienia z wielowymiarowymi tablicami i ogólnie listami. Nadaj jej skrót `np`, który jest standardem branżowym, aby nazwa była łatwiejsza w użyciu. Teraz musimy zrozumieć coś bardzo ważnego. Tworzysz symulację, której celem jest symulacja sytuacji z życia wziętych. W rzeczywistości każdy automat daje nam szansę na wygraną, a niektóre automaty dają większą niż inne. Dlatego symulując to środowisko, musisz zrobić to samo. Należy jednak pamiętać, że nasza sztuczna inteligencja nie będzie znać tych z góry określonych współczynników wygranych.

Nie może ich po prostu przeczytać i ocenić na ich podstawie, która maszyna jest najlepsza. W tym przykładzie nazwijmy tę listę szans na wygraną jako `conversionRates`.

```
# Ustawianie współczynników konwersji i liczby próbek
conversionRates = [0.15, 0.04, 0.13, 0.11, 0.05]
N = 10000
d = len(conversionRates)
```

Tutaj masz pięć automatów do gry. Dają konkretną szansę na wygraną; na przykład automat nr 1 zapewnia 15-procentową szansę. Następnie tworzysz kilka próbek, `N`. Pamiętaj, że wykonujesz symulację, więc musisz mieć predefiniowany zestaw danych, który powie Ci, czy wygrałeś, czy nie. Wprowadzasz również zmienną, `d`, która jest długością listy współczynników konwersji — odpowiada liczbie automatów do gry. Warto używać takich krótkich nazw zmiennych, w przeciwnym razie kod byłby dłuższy i mniej czytelny.

Czy masz pomysł, co powinieneś zrobić dalej? Prowadzisz symulację, więc musisz mieć predefiniowany zestaw wygranych i przegranych dla każdego automatu oraz każdej próbki. Gorąco zachęcam, abyś spróbował to zrobić samodzielnie. Musisz mieć zestaw, który powie Ci, czy w pewnym momencie wygrałeś, czy nie, grając na określonym automacie. Odpowiedź znajduje się w następnym fragmencie kodu.

```
# Tworzenie zbioru danych
X = np.zeros((N, d))
for i in range(N):
    for j in range(d):
        if np.random.rand() < conversionRates[j]:
            X[i][j] = 1
```

W pierwszym wierszu tworzysz dwuwymiarową tablicę wypełnioną zerami o rozmiarze `N * d`. Oznacza to, że utworzyłeś tablicę z `N` (w tym przypadku `10000`) wierszami oraz `d` (w tym przypadku `5`) kolumnami. Następnie, w pętli `for`, przechodzisz przez każdy wiersz w tej dwuwymiarowej

tablicy `X`. W zagnieżdżonej pętli `for` przechodzisz przez każdą kolumnę w tym wierszu. W linii 5 powyższego fragmentu kodu dla każdego automatu (każdej kolumny) sprawdzamy, czy losowa liczba zmiennoprzecinkowa z zakresu (0,1) jest mniejsza niż współczynnik konwersji dla odpowiedniego automatu.

To tak jak gra na automacie; ponieważ istnieje szansa na uzyskanie dowolnej liczby zmiennoprzecinkowej z tego zakresu jest taka sama jak szansa na uzyskanie każdej z pozostałych, szansa na uzyskanie liczby mniejszej niż `x` (gdzie `x` także mieści się w zakresie (0,1)) jest równa `x`. Na przykład dla `d` = 0,15 w 15 przypadkach na 100 uzyskamy liczbę zmiennoprzecinkową mniejszą niż 0,15, a zatem jest 15-procentowa szansa na zwrócenie wysokiej nagrody dla automatu do gry nr 1. Innymi słowy, jeśli losowa liczba zmiennoprzecinkowa jest mniejsza, oznacza to, że wygrasz, jeśli zagrasz na tej konkretnej maszynie w tym określonym czasie.

Aby się upewnić, że rozumiesz: jeśli jedna z `N` próbek z Twojego zbioru danych `X` wygląda tak: `[0, 1, 0, 0, 1]`, wygrałbyś w tym momencie, grając na automacie nr 2 lub nr 5.

Następnie musisz utworzyć dwie tablice, które będą liczyć, ile razy przegrałeś i wygrałeś, grając na każdym automacie, na przykład:

```
# Tworzenie tablic do liczenia naszych strat i zwycięstw
nPosReward = np.zeros(d)
nNegReward = np.zeros(d)
```

Nazwij je: `nPosReward` (liczba wygranych) i `nNegReward` (liczba przegranych).

Utworzyłeś zestaw symulacji i te dwa liczniki — możesz rozpocząć kodowanie algorytmu próbkowania Thompsona. Pamiętaj, że teoria, a także inny przykład, zostaną omówione później.

Następnie zainicjalizuj pętlę `for`, która będzie iterować przez każdą próbkę w naszym zbiorze danych i wybierze najlepszy automat do gry. Początkowo utwórz tylko dwie zmienne, jedną o nazwie `selected`, która powie Ci, który automat do gry został wybrany, oraz `maxRandom`, której użyjesz, aby uzyskać najwyższy rozkład Beta na wszystkich automatach:

```
# Przeniesienie naszego najlepszego automatu do rozkładu Beta i aktualizowanie jego strat i wygranych
for i in range(N):
    selected = 0
    maxRandom = 0
```

Teraz możesz przejść do sedna próbkowania Thompsona. Weźmiesz losowe przypuszczenia z naszej dystrybucji Beta i znajdziesz najwyższą wartość na wszystkich swoich automatach.

Możesz użyć metody zaczerpniętej z NumPy o nazwie `np.random.beta(a, b)`, która zwraca to losowe przypuszczenie. Wiedząc to, spróbuj samodzielnie znaleźć najlepsze przypuszczenie i najlepszą maszynę! Jeśli nie zdasz egzaminu — nie omawialiśmy jeszcze tej teorii — pokażę Ci odpowiedź. Powodzenia!

Mam nadzieję, że spróbowałeś. Niezależnie od tego, czy to wyszło, czy nie, oto moja odpowiedź:

```
for j in range(d):
    randomBeta = np.random.beta(nPosReward[j] + 1, nNegReward[j] + 1)
```

```
        if randomBeta > maxRandom:
            maxRandom = randomBeta
            selected = j
```

Nie przegapiłeś niczego — to cały kod potrzebny do tego zadania. Tworzysz pętlę `for`, aby iterować po każdym automacie i znaleźć najlepszy. Dla każdego automatu z indeksem `j` (pamiętaj, że nadal jesteś w większej pętli `for` z indeksem `i`) bierzesz losowe losowanie o nazwie `randomBeta` z naszego rozkładu Beta i sprawdzasz, czy jest większe niż `maxRandom`.

Jeśli tak, to ponownie przypisujesz `maxRandom` tak, aby był równy `randomBeta`, i ustawiasz `selected` na wartość równą indeksowi tego nowego automatu do gry z najwyższym przypuszczeniem — `j`. Warto też wspomnieć, jakie są w tym przypadku argumenty `a` i `b` funkcji Beta; to liczba wygranych i przegranych gier na konkretnym automacie. Pamiętaj, im większy pierwszy argument, tym lepiej i tym wyższe będzie nasze przypadkowe przypuszczenie; im większy drugi argument, tym gorsze i niższe będzie nasze przypadkowe przypuszczenie.

Teraz, gdy wybrałeś najlepszy automat do gry, co Twoim zdaniem powinieneś zrobić dalej?

Musisz zaktualizować wartości `nPosReward` lub `nNegReward` w zależności od tego, czy wygrałeś, czy nie. Możemy to zrobić za pomocą tego kodu:

```
if X[i][selected] == 1:
    nPosReward[selected] += 1
else:
    nNegReward[selected] += 1
```

Powyżej możesz zobaczyć wykorzystanie utworzonej wcześniej tablicy `X`. Możesz sprawdzić, czy wygrałeś tę rundę, przez sprawdzenie, czy w odpowiednim miejscu w tablicy `X` jest `1`. Jeśli wygrasz, aktualizujesz indeks odpowiadający wybranej maszynie w `nPosReward` — dodajesz `1`. Jeśli jednak przegrasz, aktualizujesz `nNegReward` przez dodanie 1 w tym samym indeksie. Możesz wyraźnie zobaczyć, że jeśli wygrasz następnym razem, twoje losowe przypuszczenie z dystrybucji Beta dla tej maszyny będzie wyższe; a jeśli przegrasz, będzie niższe.

Ten kod już działa, chociaż warto dodać kilka wierszy kodu, aby wyświetlić automat, który Twój kod uważa za najlepszy:

```
# Pokazanie, który automat jest uważany za najlepszy
nSelected = nPosReward + nNegReward
for i in range(d):
    print("Maszyna numer " + str(i + 1) + "została wybrana " + str(nSelected[i]) +
    ↪" razy")
print("Wniosek: najlepsza maszyna to maszyna nr " + str(np.
argmax(nSelected) + 1))
```

Tutaj po prostu wyświetlasz, ile razy każdy automat został wybrany przez Twój algorytm. Aby uzyskać te liczby, możesz dodać razem listy `nPosReward` i `nNegReward`. W ostatnim wierszu pokazujesz, który automat był wybierany najwięcej razy, dzięki czemu jest uważany za najlepszy.

Teraz możesz po prostu uruchomić kod i zobaczyć wyniki:

```
Maszyna numer 1 została wybrana 7927,0 razy
Maszyna numer 2 została wybrana 82,0 razy
Maszyna numer 3 została wybrana 1622,0 razy
Maszyna numer 4 została wybrana 306,0 razy
Maszyna numer 5 została wybrana 63,0 razy
Wniosek: Najlepsza maszyna to maszyna numer 1
```

Jak widać, Twój algorytm szybko wykrył, że maszyna nr 1 jest najlepsza. Nastąpiło to po mniej więcej 2000 rundac (2000 próbek w zestawie X).

Zrozumienie modelu

Próbkowanie Thompsona jest zdecydowanie najlepszym modelem do tego rodzaju problemów; na końcu tego rozdziału zobaczysz porównanie z inną metodą. Oto jak działa. Pierwszą rzeczą, jaką robimy, szukając najlepszego automatu do gry, jest rozegranie gry po kolei na każdym z pięciu automatów. Zatem zaczynamy:

Runda 1: Gramy na automacie nr 1. Powiedzmy, że otrzymujemy nagrodę 0.

Runda 2: Gramy na automacie nr 2. Powiedzmy, że otrzymujemy nagrodę 1.

Runda 3: Gramy na automacie nr 3. Powiedzmy, że otrzymujemy nagrodę 0.

Runda 4: Gramy na automacie nr 4. Powiedzmy, że otrzymujemy nagrodę 0.

Runda 5: Gramy na automacie nr 5. Powiedzmy, że otrzymujemy nagrodę 1.

Jak myślisz, dlaczego musieliśmy to zrobić? Zrobiliśmy to tylko po to, aby zebrać początkowe informacje z każdego z automatów do gry. Te informacje będą potrzebne w przyszłych rundach.

Teraz zaczyna się robić interesująco. Co będziemy robić w rundzie 6? Na którym automacie zagramy ?

Cóż, musimy spojrzeć wstecz na to, co wydarzyło się podczas pierwszych pięciu rund. Dla każdego automatu wprowadzamy dwie nowe zmienne: jedna zlicza, ile razy automat zwrócił nagrodę 0, a druga — ile razy automat zwrócił nagrodę 1.

Oznaczmy te zmienne jako $N^0_i(n)$ oraz $N^1_i(n)$, gdzie $N^0_i(n)$ wyznacza, ile razy przez n rund automat i zwrócił nagrodę 0, a $N^1_i(n)$ wyznacza, ile razy przez n rund automat i zwrócił nagrodę 1. Te dwie zmienne są oznaczone w naszym kodzie przez `nNegReward` i `nPosReward`. Tak więc opierając się na tym, co do tej pory uzyskaliśmy w rundzie 5., podajmy przykładowe wartości tych zmiennych:

$N^0_1(1) = 1$ oznacza, że automat 1 zwrócił 1 stratę w ciągu 1 rundy.

$N^1_1(1) = 0$ oznacza, że automat 1 zwrócił 0 zwycięstw w ciągu 1 rundy.

$N^0_2(1) = 0$ oznacza, że automat 2 zwrócił 0 strat w ciągu 1 rundy.

$N^1_2(1) = 1$ oznacza, że automat 2 zwrócił 1 wygraną w ciągu 1 rundy.

$N^0{}_5(4) = 0$ oznacza, że automat 5 zwrócił 0 strat w ciągu 4 rund.

$N^1{}_5(4) = 0$ oznacza, że automat 5 zwrócił 0 wygranych w ciągu 4 rund.

$N^0{}_5(5) = 0$ oznacza, że automat 5 zwrócił 0 strat w ciągu 5 rund.

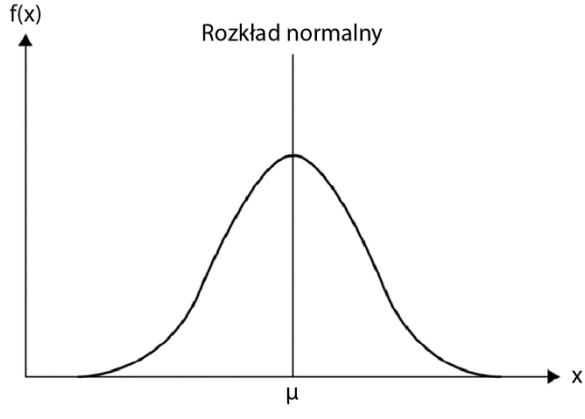
$N^1{}_5(5) = 1$ oznacza, że automat 5 zwrócił 1 wygraną w ciągu 5 rund.

W porządku, to była łatwa część. Dobra wiadomość jest taka, że utworzyliśmy wszystkie zmienne, których potrzebowaliśmy dla naszej sztucznej inteligencji. Zła wiadomość jest taka, że teraz przed nami jest najtrudniejsza część — matematyka. Jeśli uważasz, że matematyka to dobra wiadomość, podoba mi się Twój zapał, ale nie martw się, jeśli nie lubisz matematyki, nie zawiodę Cię.

Co to jest rozkład?

Następnym krokiem w naszej podróży do sztucznej inteligencji jest wprowadzenie rozkładów w matematyce. W tym celu podam prostą definicję, własnymi słowami, a nie tymi bardzo formalnymi, które można znaleźć w książkach matematycznych. Chcę mieć pewność, że wszyscy to zrozumieją. Oto ona: rozkład zmiennej to funkcja, która dla każdej wartości z możliwego zakresu wartości, jakie może przyjąć zmienna, daje prawdopodobieństwo takie, że ta zmienna jest równa tej wartości.

Zrozum naprawdę, co to jest, na przykładzie:



Rysunek 5.1. Rozkład normalny

Na poprzednim wykresie można zobaczyć przykład rozkładu. Teraz pamiętaj, że w podanej przez siebie definicji wspomniałem o dwóch miarach: „zakres wartości, jakie może przyjąć zmienna" oraz „prawdopodobieństwo, że ta zmienna jest równa tej wartości". W każdym rozkładzie na osi x masz zakres wartości, które zmienna może przyjąć, a na osi y masz prawdopodobieństwo, że zmienna jest równa każdej wartości.

Nie martw się, jeśli to nie jest jeszcze jasne. Aby rozszerzyć nasz przykład, załóżmy, że na poprzednim wykresie tą zmienną jest roczne wynagrodzenie osób w określonym kraju.

Na osi x mielibyśmy zakres rocznych pensji od płacy minimalnej do płacy maksymalnej, powiedzmy, że od 15 000 do 150 000 dolarów. Na osi y mielibyśmy prawdopodobieństwo, że dana osoba otrzyma taką pensję.

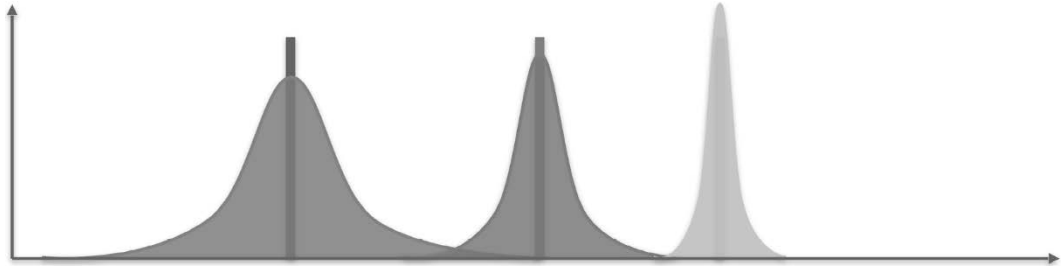
Teraz powinno to mieć więcej sensu. W przypadku niskich wynagrodzeń krzywa jest niska, czyli prawdopodobieństwo, że dana osoba zarobi około 15 000 dolarów, jest niskie.

Wówczas do środka osi x, oznaczonego jako μ, będącego średnią z wynagrodzeń, prawdopodobieństwo wzrostu wynagrodzeń rośnie. Powiedzmy, że μ jest równe 45 000 dolarów. Intuicyjnie rozumiemy, że prawdopodobieństwo, że osoba w danym kraju zarabia 45 000 dolarów rocznie, jest największe, po prostu dlatego, że większość ludzi zarabia około 45 000 dolarów rocznie. I właśnie dlatego rozkład na wykresie jest najwyższy przy tej pensji.

Im bardziej przekraczamy 45 000 dolarów rocznie, tym liczba osób, które mają taką pensję, spada, a zatem prawdopodobieństwo, że dana osoba tyle zarabia, maleje, aż wreszcie dochodzimy do rocznej pensji w wysokości 150 000 dolarów, którą dostaje bardzo niewielu, co prowadzi do prawdopodobieństwa bliskiego zeru.

W porządku, wyjaśniłem rozkład w sposób intuicyjny. Teraz musisz wiedzieć, że istnieje wiele typów rozkładów: rozkład Gaussa (który wygląda jak na poprzednim wykresie), rozkład normalny (rozkład Gaussa o średniej 0 i wariancji 1), rozkład Beta i wiele innych.

To następny krok: ***rozkłady Beta***. Rozkład Beta jest sercem sztucznej inteligencji, którą zbudowaliśmy, aby rozwiązać nasz problem wielorękiego bandyty. Oto jak wyglądają rozkłady Beta:



Rysunek 5.2. Trzy rozkłady Beta

Przejdźmy przez praktyczny przykład, aby się upewnić, że rozumiesz, jak działają rozkłady.

Wyobraź sobie, że te trzy rozkłady odpowiadają trzem różnym krajom, i ponownie przyjmijmy, że są to rozkłady wynagrodzeń w tych krajach. Który kraj ma najwyższe zarobki? Fioletowy, zielony czy żółty? Odpowiedź jest oczywista: żółty! To w tym kraju mamy dodatnie prawdopodobieństwa najwyższych zarobków (pamiętaj, pensje są na osi x, a prawdopodobieństwa na osi y).

To był tylko szybki test, aby się upewnić, że za mną nadążasz. Teraz nie musisz pamiętać dokładnej formuły rozkładu Beta, ale musisz wiedzieć, że ma on dwa parametry, oraz to, jak one wpływają na rozkład. Nie zapominaj, że już o tym wspomniałem, gdy rozwiązywaliśmy problem w praktyce; teraz wyjaśniłem to znacznie bardziej szczegółowo.

Jeśli ponownie oznaczymy te dwa parametry jako a i b, rozkład Beta będziemy mogli oznaczyć w następujący sposób:

$$y = \beta(x,a,b)$$

Możesz zapytać, co się właśnie stało — dlaczego pojawiło się x? Nie martw się, wyjaśnimy to wszystko. W powyższym wzorze y to prawdopodobieństwo, β to funkcja tylko x, x to wynagrodzenie, a a, b to dwa parametry obecne w dowolnym rozkładzie Beta. Nie musisz znać dokładnej definicji funkcji β, ale po prostu pamiętaj o kształcie jej krzywej, podanej na poprzednim wykresie.

Jednak naprawdę ważne jest, abyś teraz zrozumiał rolę parametrów a i b. To dwa punkty, które musisz znać i zwizualizować w swojej głowie:

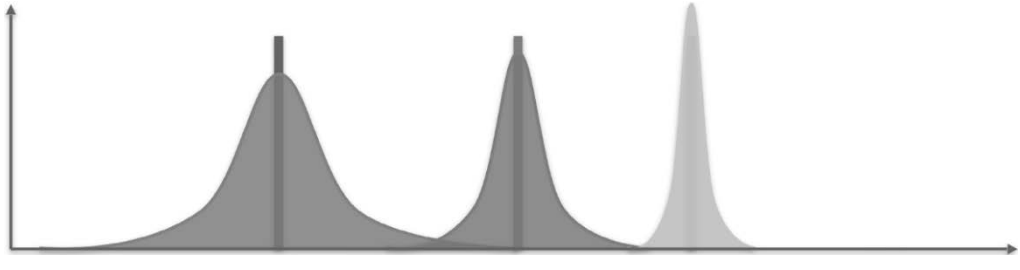
1. Biorąc pod uwagę dwa rozkłady Beta z tym samym parametrem b, ten z większym parametrem a zostanie przesunięty bardziej w prawo.
2. Biorąc pod uwagę dwa rozkłady Beta z tym samym parametrem a, ten z większym parametrem b zostanie przesunięty bardziej w lewo.

Otóż to! To wystarczy, aby intuicyjnie zrozumieć, w jaki sposób nasza sztuczna inteligencja rozwiąże problem bandytów. Innymi słowy, im większy parametr a, tym bardziej rozkład Beta przesunie się w prawo, a im większy parametr b, tym bardziej rozkład Beta przesunie się w lewo.

Poćwiczmy to! Jeśli dam Ci następujące trzy rozkłady Beta:

1. $\beta(1,5)$
2. $\beta(5,1)$
3. $\beta(3,3)$

Czy możesz mi powiedzieć, który z trzech rozkładów Beta na poniższym wykresie odpowiada powyższym przykładom?



Rysunek 5.3. Trzy rozkłady Beta

Opierając się na dwóch powyższych stwierdzeniach, $\beta(1,5)$ to wykres fioletowy, $\beta(5,1)$ to wykres żółty, a $\beta(3,3)$ — zielony. Gratulacje, jeśli dobrze zgadłeś!

Teraz jesteś gotowy, aby rozwiązać nasz problem wielorękiego bandyty. Ale pozwól, że najpierw zadam Ci pytanie, które może pomóc Ci zrozumieć magię szybciej niż wyjaśnienie zawarte w tej książce:

Jeśli zamiast wynagrodzenia oś x zawiera wskaźniki wygranej na automatach i jeśli każdy z trzech rozkładów Beta reprezentuje jeden konkretny automat do gry, który z nich wybrałbyś do postawienia 1000 dolarów?

Wybrałbyś żółty!

Oczywiście! Ten rozkład ma dodatnie prawdopodobieństwa dla najwyższych współczynników konwersji, ponieważ jest to rozkład najbardziej przesunięty w prawo.

Zostało to już omówione w poprzednim punkcie tego rozdziału; powiedziałem Ci tam, że im wyższy pierwszy parametr, tym lepszy automat. Rzeczywiście, rozkład Beta zostanie przesunięty bardziej w prawo, co oznacza, że ten automat daje większą szansę na wygraną. Dodatkowo im wyższy drugi parametr, tym gorszy jest automat i rozkład Beta zostanie przesunięty bardziej w lewo, co oznacza, że ta maszyna daje mniejsze szanse na wygraną.

A teraz kolejne pytanie, zanim rozwiążemy nasz problem z bandytami. Pamiętając, że masz pięć automatów do gry, spróbuj na nie odpowiedzieć, Otóż jeśli pięć automatów do gier jest powiązanych z następującymi pięcioma rozkładami Beta współczynników sukcesu:

$\beta(1,3)$, $\beta(1,5)$, $\beta(3,3)$, $\beta(5,3)$ i $\beta(5,1)$,

który wybierzesz, aby postawić 1000 dolarów?

Odpowiedź brzmi: $\beta(5,1)$!

Oczywiście! Ponieważ jest to ten z największym parametrem a i najniższym parametrem b, a zatem rozkładem najbardziej przesuniętym w prawo, czyli to on zapewnia dodatnie prawdopodobieństwa dla najwyższych współczynników konwersji.

Jeśli nadal za mną nadążasz, jesteś zdecydowanie gotowy, aby zrozumieć magię sztucznej inteligencji. Jeśli nie, przeczytaj ponownie ten punkt. Na końcu następnego ujawnię, co dzieje się po rundzie 5.

Walka z MABP

Od teraz przed rozpoczęciem każdej rundy będziemy przypisywać każdemu automatowi gier określony rozkład Beta. W każdej rundzie n numer automatu i ($i = 1,2,3,4,5$) będzie powiązany z następującym rozkładem Beta:

$$\beta\left(N^1_i(n)+1,\ N^0_i(n)+1\right)$$

W tym miejscu należy przypomnieć sobie, że:

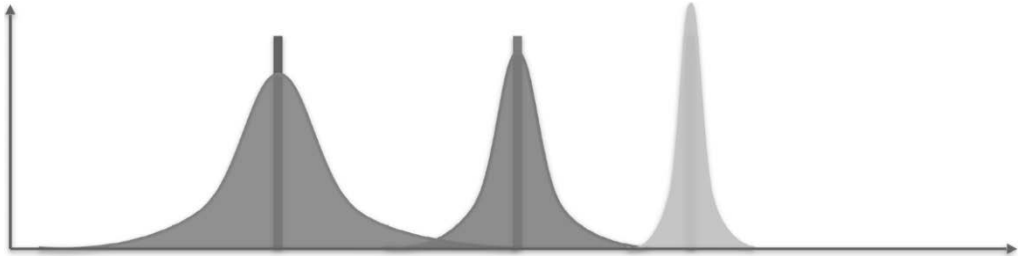
- $N^1_i(n)$ wyznacza, ile razy przez n rund automat i zwrócił nagrodę 1,
- $N^0_i(n)$ wyznacza, ile razy przez n rund automat i zwrócił nagrodę 0.

Pamiętaj, że w rozkładzie Beta $\beta(a, b)$ im wyższy parametr a, tym bardziej rozkład jest przesunięty w prawo. Im wyższy parametr b, tym bardziej rozkład jest przesunięty w lewo. Dlatego ponieważ w każdej rundzie n i dla każdego automatu parametr a wyznacza liczbę (plus 1), która mówi, ile razy przez n rund automat i zwrócił nagrodę 1, a parametr b wyznacza liczbę (plus 1), która mówi, ile razy przez n rund automat i zwrócił nagrodę 0, im częściej automat będzie zwracał 1 (sukces), tym bardziej jego rozkład będzie przesunięty w prawo, a im częściej będzie zwracał 0 (brak sukcesu), tym bardziej jego rozkład będzie przesunięty w lewo.

Gratulacje, jeśli wiesz, jakie wartości powinny przyjąć a i b. Użyliśmy ich już w praktycznym samouczku powyżej; mieliśmy dwie tablice, `nPosReward` i `nNegReward`, które odpowiadają, odpowiednio, $N^1_i(n)$ oraz $N^0_i(n)$.

Kiedy już to zrozumiesz, spróbuj wypracować strategię, zanim podam Ci rozwiązanie.

W porządku, zaraz zobaczysz magię. Przed rozgrywką w każdej rundzie będziemy pobierać losową próbkę dla każdego z pięciu rozkładów odpowiadających pięciu automatom do gry. Jeśli nie wiesz, co to znaczy, wyjaśnię. Pozwól, że pokażę Ci ponownie wykres trzech rozkładów Beta:



Rysunek 5.4. Trzy rozkłady Beta

Co miałem na myśli, mówiąc o pobieraniu losowej próbki? Po pierwsze, pamiętaj, że dla naszego problemu z bandytami na osi x mamy wskaźniki sukcesu od 0 do 1. Na przykład $x = 0{,}25$ oznacza, że maszyna zwraca 1 nagrodę (sukces) w 25% przypadków. Następnie na osi y nadal mamy prawdopodobieństwa uzyskania tych wskaźników sukcesu.

Skoncentrujmy się na jednym rozkładzie, na przykład fioletowym. Co oznaczałoby pobranie losowej próbki dla tego rozkładu? Oznaczałoby to po prostu, że losowo wybieramy wartość na osi x, gdzie rozkład jest dodatni. Wartości x, dla których prawdopodobieństwo jest największe, będą miały największą szansę na wybranie. Powiedzmy, że górna część fioletowej krzywej odpowiada wartościom $x = 0{,}2$ i $y = 0{,}35$.

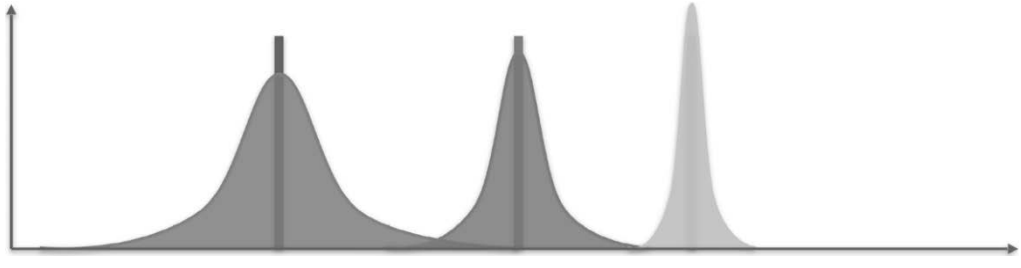
Pobranie losowej próbki dla tego fioletowego rozkładu oznacza, że będziemy mieli 35% szansy na wybranie wskaźnika sukcesu 20%. Aby to uogólnić, powiedzmy, że $y = \beta_{\text{purpurowy}}(x)$ jest funkcją związaną z fioletowym rozkładem, więc pobranie losowej próbki z tego fioletowego rozkładu oznacza, że dla każdego wskaźnika sukcesu x na osi x mamy $\beta_{\text{purpurowy}}(x)$ szans na wybranie x. To właśnie oznacza „pobranie losowej próbki dla rozkładu", a nazywane jest to również „próbkowaniem rozkładu".

Teraz, gdy to rozumiesz, zobaczmy, gdzie skończyliśmy. Powiedzieliśmy, że w każdej rundzie przed rozgrywką będziemy pobierać losową próbkę dla każdego z pięciu rozkładów, odpowiadających pięciu automatom do gry. Otrzymujemy w ten sposób pięć wartości na osi x, z których każda odpowiada każdemu z pięciu automatów. Następnie pojawia się kluczowe pytanie, które pokaże, czy masz właściwą intuicję dotyczącą strategii.

Jak sądzisz, na którym automacie najlepiej zagrać, na podstawie obserwacji tych pięciu wartości? Bardzo bym chciał, żebyś poświęcił trochę czasu i spróbował odpowiedzieć na to pytanie, ponieważ w tej chwili jesteśmy w centrum strategii (możesz również rzucić okiem na nasz wcześniej napisany kod). Odpowiedź znajduje się w następnym akapicie.

Naprawdę mam nadzieję, że spróbowałeś odpowiedzieć samodzielnie: automat, na którym zagrasz w następnej kolejności, to ten, dla którego mamy najwyższy wynik z pięciu losowych próbek. Czemu? Ponieważ najwyższy wynik odpowiada najwyższemu wskaźnikowi sukcesu, rozkład Beta powiązany z wybranym automatem ma wokół tego najwyższego wskaźnika dodatnie prawdopodobieństwo.

Ponieważ chcemy zmaksymalizować wskaźnik sukcesu maszyn, na których gramy (chcemy bowiem zarabiać), musimy wybrać automat, dla którego rozkład Beta ma dodatnie prawdopodobieństwo wokół najwyższych wskaźników sukcesu. Na poniższym wykresie jest to rozkład żółty.



Rysunek 5.5. Trzy rozkłady Beta

Teraz musimy cofnąć się o krok. Byłem w Twojej sytuacji wiele razy, kiedy uczyłem się czegoś nowego i technicznego. Czasami było to przytłaczające. W takim przypadku najlepszym posunięciem jest cofnięcie się o krok i dokładnie to teraz zrobimy, podsumowując strategię i rozumowanie, na którym się opiera.

Strategia próbkowania Thompsona w trzech krokach

Oto co zrobi sztuczna inteligencja w każdej rundzie n po rozegraniu gry dla każdego z pięciu automatów w pierwszych pięciu rundach:

1. Dla każdego automatu i (i = 1,2,3,4,5) bierzemy losową próbkę $\theta_i(n)$ z jego rozkładu Beta:

   $\theta_i(n) \sim \beta\,(N^1_i(n)+1,\ N^0_i(n)+1)$

   - $N^1_i(n)$ wyznacza, ile razy przez n rund automat i zwrócił nagrodę 1.
   - $N^0_i(n)$ wyznacza, ile razy przez n rund automat i zwrócił nagrodę 0.

2. Gramy na automacie $s(n)$, który ma najwyższą próbkę $\theta_i(n)$:

   $s(n) = argmax(\theta_i(n))_{i=1,2,3,4,5}$

3. Aktualizujemy $N^1_{s(n)}(n)$ *oraz* $N^0_{s(n)}(n)$:

   Jeśli gra na automacie $s(n)$ zwróciła nagrodę 1:

   $N^1_{s(n)}(n) := N^1_{s(n)}(n)+1$

   Jeśli gra na automacie $s(n)$ zwróciła nagrodę 0:

   $N^0_{s(n)}(n) := N^0_{s(n)}(n)+1$

Następnie powtarzamy te trzy kroki w każdej rundzie, aż wydamy 1000 dolarów. Ta strategia, zwana próbkowaniem Thompsona, jest podstawowym, ale potężnym modelem gałęzi sztucznej inteligencji zwanej uczeniem ze wzmacnianiem.

Ostateczny krok ku zrozumieniu próbkowania Thompsona

Twoja intuicja dotycząca tego, dlaczego i jak to działa, powinna podążać następującą ścieżką (staraj się o tym pamiętać lub wizualizować to na wykresie):

Każdy automat ma własny rozkład Beta. W trakcie kolejnych rund rozkład Beta automatu z najwyższym współczynnikiem konwersji będzie stopniowo przesuwany w prawo, a rozkład Beta strategii z niższymi współczynnikami konwersji będzie stopniowo przesuwany w lewo (kroki 1. i 3.). Dlatego też ze względu na krok 2. coraz częściej wybierany będzie automat o najwyższym współczynniku konwersji.

I voilà! Gratulacje — właśnie dowiedziałeś się o potężnym modelu AI, ogromnym kroku na swojej drodze. Aby zobaczyć próbkowanie Thompsona w akcji i sprawdzić, czy rzeczywiście działa, nie będziesz musiał iść do kasyna; zastosujemy tę technikę w innym rzeczywistym modelu w rozdziale 6., „AI w sprzedaży i reklamie — sprzedawaj jak Wilk z AI Street".

Na koniec pozwolę sobie zakończyć ten samouczek teoretyczny pytaniem. Wcześniej w książce mówiłem, że każda sztuczna inteligencja, którą budujemy dzisiaj, przyjmuje stan jako dane wejściowe, zwraca jako wyjście akcję do rozegrania, a po jej rozegraniu otrzymuje nagrodę (dodatnią lub ujemną). **Jakie są stany wejściowe, wykonywane działania i otrzymane nagrody w przypadku tego konkretnego problemu z bandytami?** Pomyśl o tym, zanim przeczytasz następny akapit.

Oto odpowiedź:

- Stan wejściowy to dokładna runda, którą osiągnęliśmy, w tym informacje o dwóch parametrach: $N^1_{s(n)}(n)$ oraz $N^0_{s(n)}(n)$.
- Akcją wyjściową jest pociągnięcie za ramię wybranego automatu.
- Nagroda wynosi 1 lub 0: 1, jeśli automat zwróci dwukrotność zainwestowanego dolara, i 0, jeśli stracimy dolara.

Gratulacje, jeśli odpowiedziałeś poprawnie na to pytanie i zainteresowałeś się tym pierwszym modelem AI, próbkowaniem Thompsona. I pamiętaj, że w rozdziale 6., „AI w sprzedaży i reklamie — sprzedawaj jak Wilk z AI Street", wprowadziliśmy to w życie, aby rozwiązać prawdziwy problem biznesowy.

Próbkowanie Thompsona w porównaniu ze standardowym modelem

Kiedy po raz pierwszy nauczyłem się próbkowania Thompsona, miałem jedno główne pytanie: Czy to naprawdę takie dobre? W rzeczywistości, gdybyś miał uruchomić model standardowy (pisząc „model standardowy", mam na myśli granie na każdym automacie określoną liczbę razy) i osobno próbkowanie Thompsona, mógłbyś nie zauważyć dużej różnicy; prawdopodobnie doszedłbyś do wniosku, że działają one tak samo dobrze jak inne.

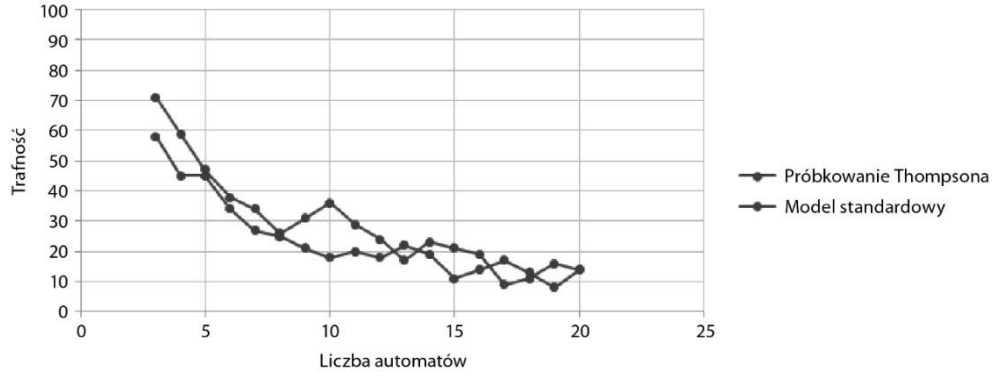
Aby sprawdzić, czy to prawda, że próbkowanie Thompsona nie jest lepsze, zaimplementowałem kod do testowania obu rozwiązań w wielu różnych scenariuszach. Zmiany obejmowały: liczbę próbek (200, 1000 lub 5000), liczbę automatów (od 3 do 20) oraz zakresy współczynników konwersji (zakresy, w których można było ustawić współczynniki konwersji: 0 – 0,1; 0 – 0,3; 0 – 0,5).

Każdy scenariusz przetestowałem 100 razy, aby obliczyć dokładność każdego modelu.

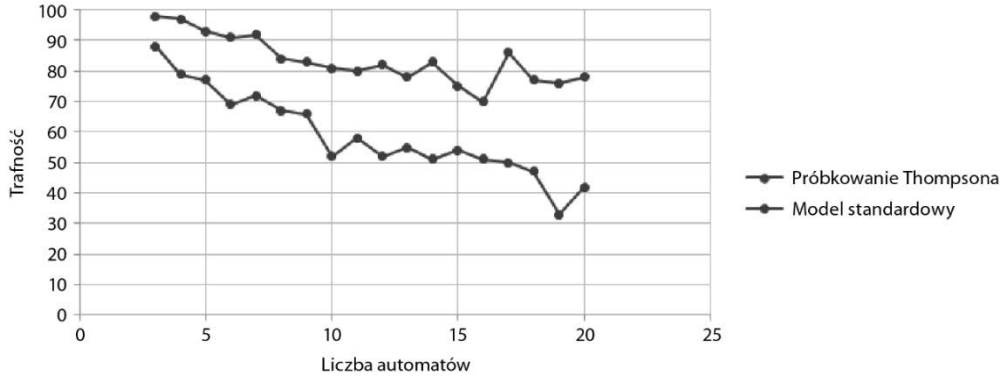
Wyniki i użyty kod znajdują się, odpowiednio, w plikach *resultsModified.xlsx* i *compare.py* w katalogu *Chapter 05* tej książki na stronie GitHuba. Tutaj możesz zobaczyć parę wykresów zaczerpniętych z pliku Excela, które pokazują trafność obu modeli (patrz rysunki 5.6 i 5.7).

Pierwszy wykres, pokazany na rysunku 5.6., ilustruje trafność obu modeli w zależności od liczby automatów do gier. Liczbę próbek ustawiono na 200, a zakresy współczynnika konwersji ustawiono na 0 – 0,1, co oznacza, że różnice między tymi współczynnikami były niewielkie. To najtrudniejsze ustawienie dla tego porównania. Ogólnie rzecz biorąc, próbkowanie Thompsona działało lepiej niż model standardowy (o 22%).

Drugi wykres, pokazany na rysunku 5.7., przedstawia trafność w najłatwiejszych warunkach. Liczbę próbek ustawiono na 5000, a zakresy współczynnika konwersji ustawiono na 0 – 0,5, co oznacza, że różnice były wyraźnie widoczne. Ogólny spadek trafności próbkowania Thompsona jest mniejszy niż spadek trafności standardowego rozwiązania. Tym razem próbkowanie Thompsona wypadło znacznie lepiej (o 41%).



Rysunek 5.6. Trafność w stosunku do liczby automatów (200 próbek)



Rysunek 5.7. Trafność w stosunku do liczby automatów (5000 próbek)

Biorąc pod uwagę wszystkie scenariusze, próbkowanie Thompsona osiągnęło średnią trafność 57%, a model standardowy osiągnął trafność 43%. Jest to znacząca różnica, biorąc pod uwagę fakt, że testowano bardzo trudne scenariusze (na przykład tylko 200 próbek, zakres 0 – 0,1 i 20 automatów do gry).

Podsumowanie

Próbkowanie Thompsona to potężna technika próbkowania, która pozwala szybko obliczyć najwyższy z wielu nieznanych współczynników konwersji. Jest zawsze stosowane w problemie wielorękiego bandyty, w klasycznym przypadku składającego się z kilku automatów do gier, z których każdy ma inny współczynnik konwersji pozytywnych wyników. Po raz pierwszy rzuciliśmy okiem na to, jak ta sztuczna inteligencja rozwiązuje ten problem lepiej i szybciej niż standardowe metody.

W następnym rozdziale wykonamy w pełni praktyczne ćwiczenie, podczas którego zobaczymy, jak wieloręki bandyta może z łatwością modelować problem biznesowy — reklamę internetową — oraz jak próbkowanie Thompsona może wnieść znaczną wartość dodaną.

Oto Twoja świetlana przyszłość w świecie AI!

Grono entuzjastów sztucznej inteligencji stale rośnie. Jest już bowiem jasne, że stanowi ona dostępną metodę zmiany świata na lepsze. Pełnymi garściami ze zdobyczy AI czerpią naukowcy, analitycy danych, przedsiębiorcy i menedżerowie, a nawet politycy i ekonomiści. Jej możliwości wydają się dziś nieograniczone — aby je wykorzystać, wystarczy zdobyć gruntowną wiedzę i dobrze zrozumieć podstawy sztucznej inteligencji. Na pierwszy rzut oka nie są to trudne zadania. Choćby ze względu na dostęp do wielu artykułów, kursów czy książek o technologiach sztucznej inteligencji. Jednak w tym nadmiarze materiałów bardzo trudno dokonać właściwego dla siebie wyboru.

To kompletny, zwięzły przewodnik po świecie sztucznej inteligencji. Znalazły się tu przejrzyście wyłożone podstawy i bardziej zaawansowane zagadnienia. Wyjaśniono, jak najlepiej zabrać się do tworzenia systemów AI wykorzystujących uczenie ze wzmacnianiem oraz głębokie uczenie. Krok po kroku pokazano, jak zrealizować pięć praktycznych projektów. To książka skierowana zarówno do studentów, jak i naukowców, menedżerów czy przedsiębiorców — dowiedzą się z niej, jak zbudować inteligentne oprogramowanie przy użyciu najlepszych i najprostszych narzędzi do programowania AI. Co ważne, aby w pełni z niej skorzystać, nie trzeba posiadać umiejętności programowania.

Dzięki tej książce:

- opanujesz kluczowe umiejętności związane z uczeniem maszynowym
- zrozumiesz Q-learning oraz głęboki Q-learning
- poznasz takie narzędzia jak TensorFlow, Keras czy PyTorch
- będziesz samodzielnie tworzyć takie projekty jak wirtualny samochód
- wykorzystasz AI do rozwiązywania rzeczywistych problemów biznesowych
- nauczysz się budować inteligentne roboty

Hadelin de Ponteves

jest współzałożycielem i dyrektorem generalnym BlueLife AI, firmy zajmującej się najnowocześniejszą sztuczną inteligencją. Jest także przedsiębiorcą internetowym i twórcą ponad 50 wysoko ocenianych e-kursów edukacyjnych na takie tematy jak uczenie maszynowe, głębokie uczenie, sztuczna inteligencja i łańcuch bloków. Z jego szkoleń korzysta ponad 700 000 subskrybentów w 204 krajach.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7478-2

Cena: 79,00 zł