O'REILLY®

Wydanie III

Uczenie maszynowe z użyciem Scikit-Learn, Keras i TensorFlow

powered by

jupyter™

Helion

Aurélien Géron

Tytuł oryginału: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 3rd Edition

Tłumaczenie: Krzysztof Sawka

ISBN: 978-83-8322-423-7

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

Spis treści

ROZDZIAŁ 6.

Drzewa decyzyjne

Drzewa decyzyjne (ang. *decision trees*) stanowią wszechstronne algorytmy uczenia maszynowego, służące zarówno do zadań klasyfikacji, jak i regresji, a nawet do operacji wielowyjściowych. Uzyskujemy za ich pomocą potężne modele zdolne do uczenia się wobec złożonych zbiorów danych. Na przykład w rozdziale 2. wyuczyliśmy model `DecisionTreeRegressor` wobec zbioru danych *California Housing* i uzyskaliśmy doskonałe wyniki (w rzeczywistości wręcz przetrenowaliśmy ten model).

Drzewa decyzyjne są również elementami składowymi losowych lasów (zob. rozdział 7.), czyli obecnie jednych z najlepszych algorytmów uczenia maszynowego.

W tym rozdziale zaczniemy od omówienia procesu uczenia drzew decyzyjnych, wyliczania prognoz i wizualizowania wyników. Następnie zajmiemy się algorytmem uczącym CART dostępnym w module Scikit-Learn, a także nauczymy się regularyzować drzewa i wykorzystywać je w zadaniach regresji. Na koniec przyjrzymy się niektórym ograniczeniom drzew decyzyjnych.

Uczenie i wizualizowanie drzewa decyzyjnego

Aby zrozumieć koncepcję drzew decyzyjnych, stwórzmy jedno i zobaczmy, w jaki sposób wylicza prognozy. Za pomocą poniższego kodu wyuczymy model `DecisionTreeClassifier` wobec zbioru danych *Iris* (zob. rozdział 4.):

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)", "petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

Możemy zwizualizować wyuczone drzewo decyzyjne, używając najpierw funkcji `export_graphviz()`, aby stworzyć plik definicji grafu, nazwany *iris_tree.dot*:

```
from sklearn.tree import export_graphviz

export_graphviz(
    tree_clf,
    out_file="iris_tree.dot",
    feature_names=["petal length (cm)", "petal width (cm)"],
```

```
    class_names=iris.target_names,
    rounded=True,
    filled=True
)
```

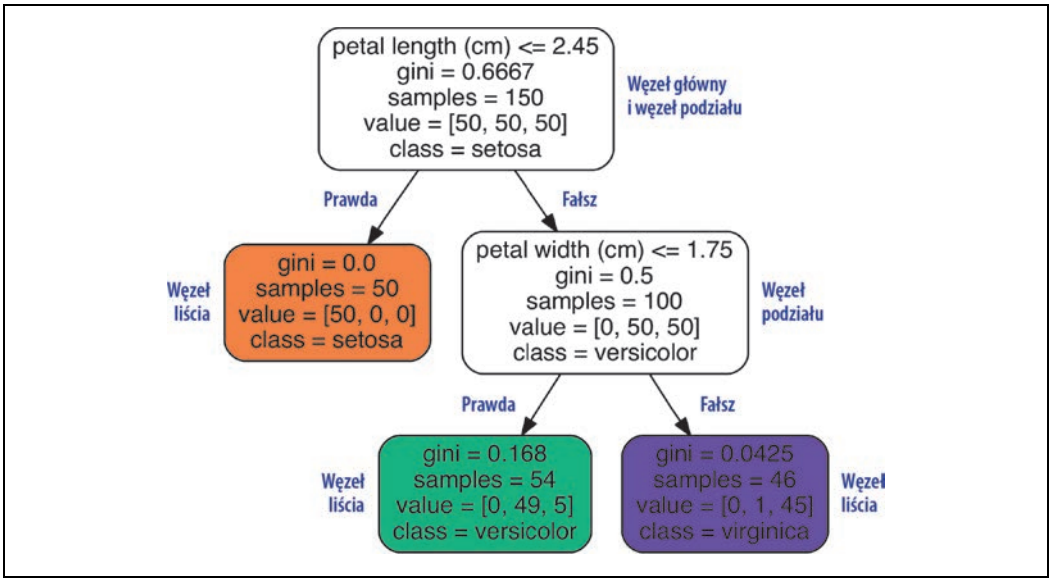
Możesz następnie użyć metody `graphviz.Source.from_file()`, aby załadować i wyświetlić plik w notatniku Jupyter:

```
from graphviz import Source

Source.from_file("iris_tree.dot")
```

Graphviz (*https://graphviz.org/*) to objęty licencją otwartego oprogramowania pakiet do wizualizacji grafów. Zawiera on również narzędzie wiersza polecenia `dot` służące do konwertowania plików *.dot* na różne formaty, takie jak *.pdf* lub *.png*.

Twoje pierwsze drzewo decyzyjne zostało pokazane na rysunku 6.1.



Rysunek 6.1. Drzewo decyzyjne wyuczone na zbiorze danych Iris

Wyliczanie prognoz

Zastanówmy się, w jaki sposób drzewo widoczne na rysunku 6.1 przewiduje wyniki. Załóżmy, że znaleźliśmy kwiat kosaćca i chcemy go sklasyfikować na podstawie jego płatków. Zaczynamy od **węzła głównego** (ang. *root node*; wysokość 0 — węzeł na samej górze grafu): zadajemy w tym węźle pytanie, czy długość płatka jest mniejsza niż 2,45 cm. Jeśli jest, przechodzimy do lewego węzła potomnego (wysokość 1, po lewej). W takim przypadku docieramy do **liścia** (ang. *leaf node*; nie wychodzą z niego kolejne węzły potomne), zatem nie zadajemy tu już więcej pytań. Wystarczy teraz spojrzeć na przewidywaną klasę w tym węźle — drzewo decyzyjne przewiduje, że mamy do czynienia z gatunkiem *Iris setosa* (`class=setosa`).

Załóżmy teraz, że znajdujemy kolejny kwiat kosaćca, którego długość płatka przekracza teraz 2,45 cm. Ponownie zaczynasz od węzła głównego, ale tym razem kierujesz się ku prawemu węzłowi potomnemu (wysokość 1, po prawej). Nie jest to węzeł liścia, lecz **węzeł podziału** (ang. *split node*), dlatego zostaje postawione kolejne pytanie: czy szerokość płatka jest mniejsza niż 1,75 cm? Jeśli tak, to prawdopodobnie znaleziony kwiat należy do gatunku *Iris versicolor* (wysokość 2, po lewej). W przeciwnym razie możemy mieć do czynienia z gatunkiem *Iris virginica* (wysokość 2, po prawej). To naprawdę jest takie proste.

Jedną z licznych zalet drzew decyzyjnych jest fakt, że prawie nie wymagają one przygotowywania danych. W istocie nie jest potrzebne skalowanie ani środkowanie cech.

Atrybut `sample` węzła zlicza liczbę wyznaczonych do niego próbek uczących. Na przykład 100 próbek ma długość płatka przekraczającą 2,45 cm (wysokość 1, po prawej), spośród których 54 mają szerokość płatka nieprzekraczającą 1,75 cm (wysokość 2, po lewej). Dzięki atrybutowi `value` dowiadujemy się, jak wiele przykładów uczących z każdej klasy przynależy do danego węzła: na przykład węzeł znajdujący się na dole po prawej stronie zawiera 0 próbek *Iris setosa*, 1 *Iris versicolor* i 45 *Iris virginica*. Z kolei atrybut `gini` stanowi miarę **zanieczyszczenia Giniego** (ang. *Gini impurity*) węzła: węzeł jest „czysty” (`gini=0`), jeżeli wszystkie znajdujące się w nim próbki uczące należą do tej samej klasy. Przykładem jest węzeł na wysokości 1 po lewej stronie, ponieważ zawiera on tylko przykłady uczące *Iris setosa*; jego zanieczyszczenie Giniego jest równe 0. Równanie 6.1 pokazuje, w jaki sposób algorytm wylicza zanieczyszczenie Giniego G_i dla i-tego węzła. Węzeł znajdujący się na wysokości 2 po lewej stronie ma zanieczyszczenie Giniego o wartości $1–(0{:}54)^2–(49{:}54)^2–(5{:}54)^2 \approx 0{,}168$.

Równanie 6.1. Zanieczyszczenie Giniego

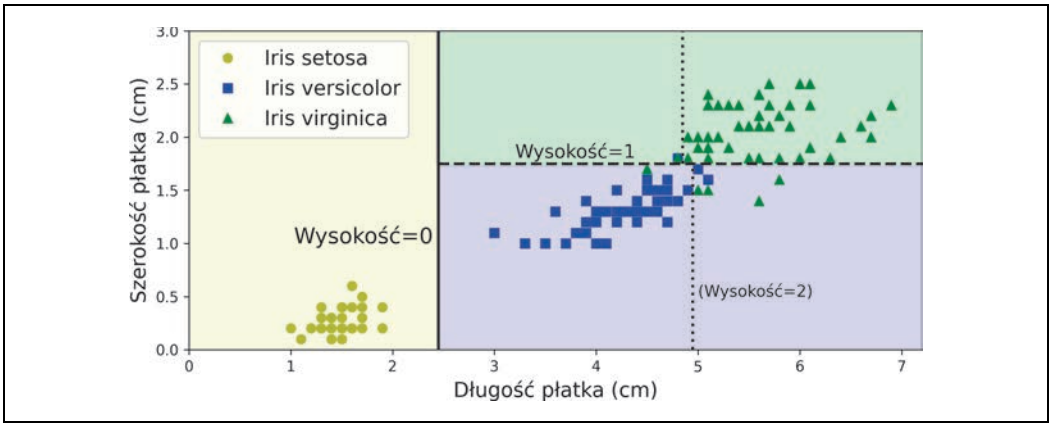
$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

W tym równaniu:

- G_i — zanieczyszczenie Giniego w i-tym węźle.
- $p_{i,k}$ — współczynnik występowania klas k wśród próbek uczących w i-tym węźle.

Moduł Scikit-Learn wykorzystuje algorytm CART generujący wyłącznie **drzewa binarne** (ang. *binary trees*), czyli drzewa, których węzły podziału mają zawsze dokładnie dwoje potomków (tj. odpowiedziami na pytania są „tak” albo „nie”). Jednak inne algorytmy, takie jak ID3, mogą tworzyć drzewa decyzyjne, w których węzły mogą mieć większą liczbę potomków.

Na rysunku 6.2 widzimy granice decyzyjne drzewa decyzyjnego. Pogrubiona linia pionowa przedstawia granicę decyzyjną węzła głównego (wysokość 0): długość płatka = 2,45 cm. Obszar po lewej stronie jest czysty (występuje tu wyłącznie klasa `Iris setosa`), dlatego nie da się go bardziej podzielić. Jednakże obszar po prawej stronie pozostaje zanieczyszczony, zatem prawy węzeł na wysokości 1 rozdziela się na węzły potomne przy szerokości płatka = 1,75 cm (linia kreskowana). Wartość parametru `max_depth` wynosi 2, dlatego na takiej wysokości zatrzymuje się drzewo decyzyjne. Jeśli wyznaczymy wartość 3 parametru `max_depth`, to dwa węzły na wysokości 2 wprowadziłyby kolejną granicę decyzyjną (reprezentowaną przez dwie linie kropkowane).



Rysunek 6.2. Granice decyzyjne drzewa decyzyjnego

Struktura drzewa, zawierająca wszystkie informacje ukazane na rysunku 6.1, jest dostępna za pomocą atrybutu `tree_` klasyfikatora. Aby uzyskać więcej szczegółów, wpisz `help(tree_clf.tree_)`, natomiast w towarzyszącym książce notatniku Jupyter (znajdziesz przykład pod adresem *https://homl.info/colab3* lub *https://ftp.helion.pl/przyklady/uczem3.zip*).

Interpretacja modelu: „czarne skrzynki" i „białe skrzynki"

Algorytm drzew decyzyjnych jest bardzo intuicyjny i możemy go z łatwością interpretować. Tego typu modele są często nazywane **modelami „białej skrzynki"** (ang. *white box models*). Jak się przekonasz, algorytmy losowego lasu i sieci neuronowe należą do zgoła odmiennej kategorii, **modeli „czarnej skrzynki"** (ang. *black box models*). Ich przewidywania są znakomite i bez trudu możemy sprawdzić, jakie zostały przeprowadzone obliczenia do ich uzyskania; nie zmienia to faktu, że nieraz trudno wytłumaczyć prostymi słowami, skąd się te predykcje wzięły. Na przykład jeśli sieć neuronowa stwierdzi, że na danym zdjęciu znajduje się taka to a taka osoba, ciężko stwierdzić, jakie czynniki wpłynęły na ten rezultat: czy model rozpoznał daną osobę po oczach? Po ustach? Nosie? Butach? A może po kanapie, na której ta osoba siedzi? Z drugiej strony drzewo decyzyjne zawiera szereg przejrzystych i dobrze zdefiniowanych reguł klasyfikacji, za pomocą których w razie potrzeby można nawet ręcznie wyliczać prognozy (np. w przypadku klasyfikowania kwiatów). Dziedzina **wytłumaczalnego uczenia maszynowego** (ang. *interpretable ML*) dąży do tworzenia systemów uczenia maszynowego zdolnych do tłumaczenia podejmowanych decyzji w sposób zrozumiały dla człowieka. Jest to ważne w wielu domenach, może na przykład sprawiać, że system nie będzie podejmował niesprawiedliwych decyzji.

Szacowanie prawdopodobieństw przynależności do klas

Drzewo decyzyjne może również szacować prawdopodobieństwo przynależności danej próbki do określonej klasy *k*. Najpierw jest wyszukiwany liść, w którym dana próbka się znajduje, po czym zostaje zwrócony odsetek przykładów uczących w tym węźle należących do klasy *k*. Załóżmy na przykład, że znaleźliśmy kwiat, którego płatki mają 5 cm długości i 1,5 cm szerokości. Próbka symbolizująca

ten kwiat znajduje się w lewym liściu na wysokości 2 drzewa, zatem algorytm wyliczy następujące prawdopodobieństwa: 0% przynależności do gatunku *Iris setosa* (0/54), 90,7% dla *Iris versicolor* (49/54) i 9,3% dla *Iris virginica* (5/54). Zostanie dla tego kwiatu przewidziana klasa `Iris versicolor` (klasa 1.), ponieważ uzyskała ona największe prawdopodobieństwo. Sprawdźmy:

```
>>> tree_clf.predict_proba([[5, 1.5]]).round(3)
array([[0. , 0.907, 0.093]])
>>> tree_clf.predict([[5, 1.5]])
array([1])
```

Doskonale! Zwróć uwagę, że szacowane prawdopodobieństwa będą identyczne w dowolnym obszarze prawego dolnego prostokąta widocznym na rysunku 6.2 — na przykład dla płatków o długości 6 cm i szerokości 1,5 cm (nawet jeśli jest dla nas oczywiste, że w tym przypadku płatki te należałyby najprawdopodobniej do gatunku *Iris virginica*).

Algorytm uczący CART

Moduł Scikit-Learn wykorzystuje algorytm **drzew klasyfikacyjnych i regresyjnych** (ang. *classification and regression tree* — CART) do uczenia drzew decyzyjnych (ich „wzrostu"). Algorytm rozdziela najpierw dane uczące na dwa podzbiory przy użyciu pojedynczej cechy k i progu t_k (np. „długość płatka ≤ 2,45 cm"). W jaki sposób są dobierane wartości k i t_k? Wyszukiwana jest para parametrów (k, t_k) generująca najczystsze podzbiory, ważone pod względem rozmiaru. Funkcja kosztu, jaką algorytm stara się minimalizować, została przedstawiona w równaniu 6.2.

Równanie 6.2. Funkcja kosztu algorytmu CART używana w zadaniach klasyfikacji

$$J(k, t_k) = \frac{m_{lewy}}{m} G_{lewy} + \frac{m_{prawy}}{m} G_{prawy}$$

gdzie $\begin{cases} G_{\frac{lewy}{prawy}} & \text{mierzy zanieczyszczenie lewego/prawego podzbioru} \\ m_{lewy/prawy} & \text{stanowi liczbę próbek w lewym/prawym podzbiorze} \end{cases}$

Gdy algorytm CART rozdzieli zestaw danych uczących na dwa podzbiory, są one dalej dzielone na tej samej zasadzie aż do osiągnięcia maksymalnej wysokości (zdefiniowanej za pomocą hiperparametru `max_depth`) lub jeśli nie uda się określić takiego podziału, który zmniejszałby zanieczyszczenie. Kilka innych hiperparametrów (omówimy je niebawem) określa dodatkowe warunki zatrzymania budowy drzewa: `min_samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf` i `max_leaf_nodes`.

Jak widać, CART jest **algorytmem zachłannym** (ang. *greedy algorithm*): zachłannie poszukuje optymalnego podziału na najniższym poziomie, a następnie na każdym kolejnym powtarza tę czynność. Nie sprawdza on, czy dany podział będzie prowadził kilka poziomów wyżej do najmniejszego możliwego zanieczyszczenia. Algorytmy zachłanne często dają dobre wyniki, nie muszą być one jednak optymalne.

Niestety szukanie optymalnego drzewa jest klasyfikowane jako **problem NP-zupełny**[1]. Należy poświęcić $O(\exp(m))$ czasu na jego rozwiązanie, przez co problem staje się trudny nawet dla małych zbiorów uczących. Dlatego musi nam wystarczyć poszukiwanie „w miarę dobrego" rozwiązania podczas trenowania drzew decyzyjnych.

Złożoność obliczeniowa

Wyliczanie prognoz wymaga poruszania się po drzewie decyzyjnym od węzła głównego aż do liścia. Generalnie drzewa decyzyjne są w miarę zrównoważone, zatem poruszanie się po drzewie decyzyjnym wymaga jedynie odwiedzenia mniej więcej $O(log_2(m))$ węzłów, gdzie zapis log_2 oznacza **logarytm binarny** (**dwójkowy**) (ang. *binary logarithm*) z m, równy $log_2(m) = log(m)/log(2)$. W każdym węźle wymagane jest jedynie sprawdzenie wartości jednej cechy, tak więc całkowita złożoność prognoz to $O(log_2(m))$, niezależnie od liczby cech. Wyliczanie przewidywań jest zatem bardzo szybkie nawet w przypadku bardzo dużych zbiorów uczących.

Algorytm uczący porównuje wszystkie cechy (mniej, jeśli wyznaczymy wartość parametru `max_features`) ze wszystkimi próbkami znajdującymi się w danym węźle. Porównanie wszystkich cech we wszystkich przykładach w każdym węźle skutkuje złożonością uczenia na poziomie $O(n \times m\, log_2(m))$.

Wskaźnik Giniego czy entropia?

Domyślnie klasa `DecisionTreeClassifier` wykorzystuje wskaźnik zanieczyszczenia Giniego, ale możemy wybrać również **entropię** jako miarę zanieczyszczenia, wprowadzając wartość `entropy` dla hiperparametru `criterion`. Pojęcie entropii wywodzi się z termodynamiki, gdzie służy do opisu miary nieuporządkowania cząsteczek: gdy cząsteczki są nieruchome i uporządkowane w przestrzeni, to wartość entropii wynosi 0. Koncepcja entropii wkradła się również w różne inne dziedziny naukowe, w tym również do **teorii informacji** Shannona, gdzie służy do pomiaru średniej zawartości informacji w wiadomości, o czym przekonaliśmy się w rozdziale 4. Entropia wynosi 0, gdy wszystkie wiadomości są takie same. W świecie uczenia maszynowego za pomocą entropii często mierzy się zanieczyszczenie: entropia zbioru jest równa zeru, gdy mieszczą się w nim wyłącznie próbki należące do jednej klasy. Równanie 6.3 ukazuje definicję entropii i-tego węzła. Przykładowo entropia lewego węzła na wysokości 2 (rysunek 6.1) wynosi: $-\frac{49}{54}\log_2\left(\frac{49}{54}\right)-\frac{5}{54}\log_2\left(\frac{5}{54}\right)\approx 0,445$.

[1] P stanowi zbiór problemów, jakie mogą zostać rozwiązane w wielomianowym czasie (tj. wielomianowym od rozmiaru zestawu danych). Zbiór NP zawiera problemy, których rozwiązania mogą zostać zweryfikowane w wielomianowym czasie. Problem NP-trudny to każdy problem, do którego problem NP może zostać zredukowany w czasie wielomianowym. Do problemu NP-zupełnego zaliczają się problemy NP i NP-trudne. Jednym z głównych pytań natury matematycznej, niemających do tej pory odpowiedzi, pozostaje, czy P = NP. Jeżeli P ≠ NP (co jest bardzo prawdopodobne), to nigdy nie zostanie odkryty wielomianowy algorytm dla dowolnego problemu NP-zupełnego (może zmieni się to w przypadku komputerów kwantowych).

Równanie 6.3. Entropia

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2(p_{i,k})$$

Powinniśmy zatem korzystać ze wskaźnika Giniego czy z entropii? Prawdę mówiąc, w większości przypadków nie ma to większego znaczenia, gdyż uzyskujemy za ich pomocą podobne drzewa. Wskaźnik Giniego jest obliczany nieco szybciej, więc stanowi dobrą wartość domyślną. Jednak w sytuacjach, w których obydwa wskaźniki się różnią, wskaźnikowi Giniego zdarza się izolować najczęściej występującą klasę w osobnej gałęzi, natomiast entropia generuje nieco bardziej zrównoważone drzewa[2].

Hiperparametry regularyzacyjne

Algorytm drzew decyzyjnych nie przyjmuje niemal żadnych założeń dotyczących danych uczących (w przeciwieństwie na przykład do modeli liniowych, które zakładają, że operują na danych liniowych). Jeżeli nie nałożymy żadnych ograniczeń, struktura drzewa samoistnie dostosuje się do danych uczących i zrobi to prawie idealnie, niemal z pewnością ulegając przetrenowaniu. Jest to tak zwany **model nieparametryczny** (ang. *nonparametric model*) — nie dlatego, że nie zawiera parametrów (często ma ich znaczną liczbę), lecz dlatego, że liczba tych parametrów nie jest ustalana przed rozpoczęciem trenowania, zatem struktura modelu jest w stanie ściśle dopasować się do danych. Z drugiej strony mamy do czynienia z **modelami parametrycznymi** (ang. *parametric models*; reprezentowanymi m.in. przez model liniowy), w których występuje ustalona liczba parametrów, cechuje je więc ograniczenie stopni swobody, dzięki czemu zmniejszamy ryzyko przetrenowania (ale jednocześnie zwiększamy możliwość niedotrenowania).

Aby uniknąć przetrenowania modelu, musimy ograniczyć swobodę algorytmu drzewa decyzyjnego podczas uczenia. Wiemy już, że ten proces nosi nazwę regularyzacji. Hiperparametry regularyzacyjne zależą od stosowanego algorytmu, zazwyczaj jednak możemy ograniczyć przynajmniej maksymalną wysokość drzewa. W module Scikit-Learn odpowiada za to hiperparametr `max_depth`. Jego wartość domyślna, `None`, powoduje tworzenie drzew o nieograniczonej wysokości. Podanie wartości liczbowej w hiperparametrze `max_depth` spowoduje regularyzację modelu i zmniejszenie ryzyka przetrenowania.

Klasa `DecisionTreeClassifier` zawiera także kilka innych parametrów ograniczających kształt drzewa decyzyjnego:

`max_features`
: Maksymalna liczba cech używanych do dzielenia w każdym węźle.

`max_leaf_nodes`
: Maksymalna liczba liści.

`min_samples_split`
: Minimalna liczba próbek, jakie muszą się znajdować w węźle, aby został podzielony.

[2] Więcej szczegółów znajdziesz w interesującej analizie (*https://homl.info/19*), której autorem jest Sebastian Raschka.

`min_samples_leaf`

Minimalna liczba próbek, jakie muszą się znajdować w liściu.

`min_weight_fraction_leaf`

Taki sam jak parametr `min_samples_leaf`, tu jednak wartością jest ułamek całkowitej liczby ważonych próbek.

Zwiększanie wartości hiperparametrów `min_*` lub zmniejszanie `max_*` powoduje regularyzację modelu.

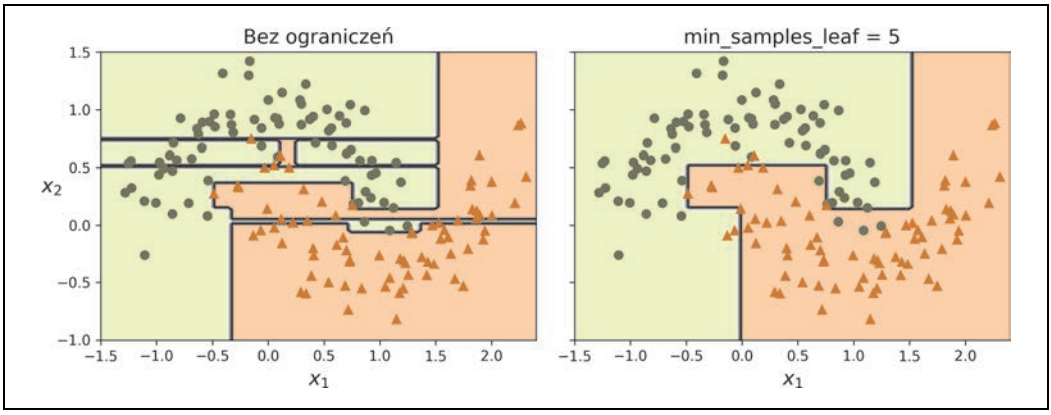
Inne algorytmy najpierw trenują model drzewa decyzyjnego bez żadnych ograniczeń, a następnie **przycinają** (ang. *prune*; usuwają) niepotrzebne liście. Węzeł zawierający same liście jest uznawany za niepotrzebny, jeśli zapewniana przez niego redukcja zanieczyszczenia okazuje się **nieistotna statystycznie**. Standardowe testy statystyczne, takie jak test chi kwadrat (test χ^2), służą do oszacowania prawdopodobieństwa, że zmniejszenie zanieczyszczenia stanowi wyłącznie wynik przypadku (jest to tzw. **hipoteza zerowa**). Jeżeli to prawdopodobieństwo, zwane **p-wartością**, przekroczy pewien określony próg (zazwyczaj 5% — definiujemy go za pomocą hiperparametru), to węzeł jest uznawany za niepotrzebny, a jego liście zostają usunięte. Proces przycinania trwa, dopóki nie zostaną usunięte wszystkie niepotrzebne węzły.

Przetestujmy regularyzację na zestawie danych sierpowatych, znanym nam już z rozdziału 5. Wytrenujemy jedno drzewo decyzyjne bez regularyzacji, a drugie z parametrem `min_samples_leaf=5`. Kod znajduje się poniżej; rysunek 6.3 ukazuje granice decyzyjne każdego drzewa:

```
from sklearn.datasets import make_moons

X_moons, y_moons = make_moons(n_samples=150, noise=0.2, random_state=42)

tree_clf1 = DecisionTreeClassifier(random_state=42)
tree_clf2 = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
tree_clf1.fit(X_moons, y_moons)
tree_clf2.fit(X_moons, y_moons)
```



Rysunek 6.3. Granice decyzyjne nieregularyzowanego (po lewej) i regularyzowanego (po prawej) drzewa

Widoczny po lewej stronie nieregularyzowany model jest wyraźnie przetrenowany, regularyzowany model z prawego wykresu prawdopodobnie będzie sobie lepiej radził z uogólnianiem. Możemy to sprawdzić, oceniając obydwa drzewa na zbiorze testowym wygenerowanym za pomocą innego ziarna losowości:

```
>>> X_moons_test, y_moons_test = make_moons(n_samples=1000, noise=0.2,
...                                         random_state=43)
...
>>> tree_clf1.score(X_moons_test, y_moons_test)
0.898
>>> tree_clf2.score(X_moons_test, y_moons_test)
0.92
```

Rzeczywiście, drugie drzewo uzyskuje większą dokładność na zbiorze testowym.

Regresja

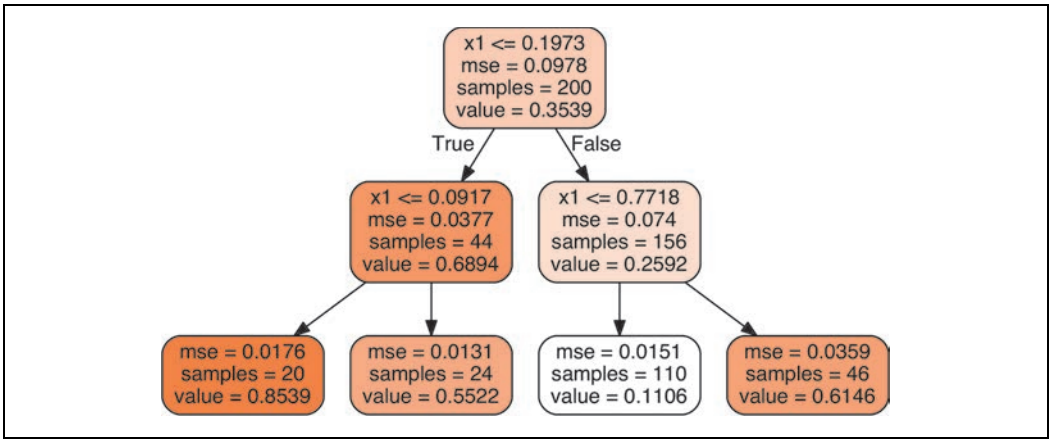
Drzewa decyzyjne mogą również wykonywać zadania regresyjne. Stwórzmy takie drzewo regresyjne za pomocą klasy `DecisionTreeRegressor`; wyuczymy je wobec zaszumionego, kwadratowego zbioru danych, a jego maksymalną wysokość ustalmy na poziomie `max_depth=2`:

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

np.random.seed(42)
X_quad = np.random.rand(200, 1) - 0.5  # pojedyncza losowa cecha wejściowa
y_quad = X_quad ** 2 + 0.025 * np.random.randn(200, 1)

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X_quad, y_quad)
```

Rysunek 6.4 przedstawia wygenerowane drzewo decyzyjne.

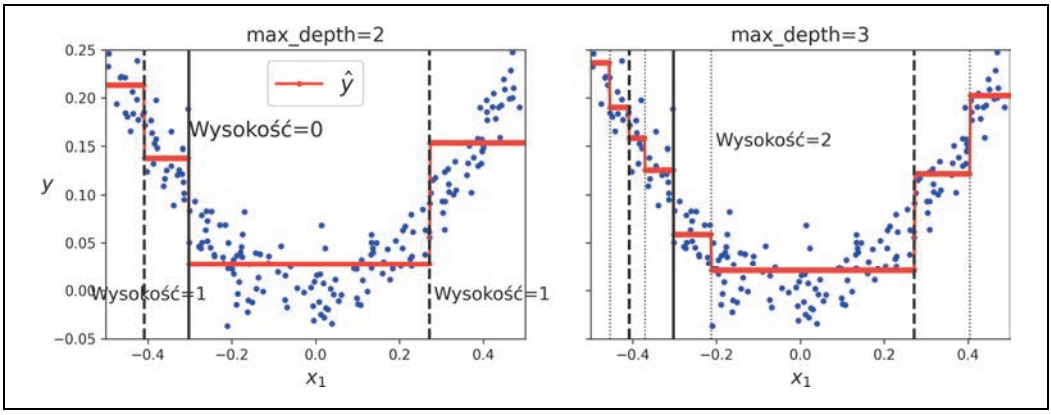


Rysunek 6.4. Regresyjne drzewo decyzyjne

Model ten bardzo przypomina stworzone przez nas wcześniej drzewo klasyfikacyjne. Główna różnica polega na tym, że w każdym węźle jest prognozowana nie klasa, lecz wartość. Załóżmy przykładowo, że chcemy wyliczyć prognozę dla nowej próbki $x_1 = 0{,}2$. Węzeł główny pyta, czy $x_1 \leq 0{,}197$.

Otrzymaliśmy inną wartość, więc algorytm przechodzi do prawego węzła potomnego, gdzie jest sprawdzane, czy $x_1 \leq 0{,}772$. Tak jest w istocie, zostaje zatem wybrany lewy węzeł potomny. Jest to liść przewidujący `value=0.111`. Predykcja ta stanowi średnią wartość docelową 110 próbek uczących powiązanych z tym liściem, a w wyniku tej prognozy otrzymujemy wartość błędu MSE wynoszącą 0,015 dla tych 110 próbek.

Przewidywania tego modelu zostały zaprezentowane po lewej stronie na rysunku 6.5. Jeżeli wyznaczymy hiperparametr `max_depth=3`, prognozy będą wyglądały tak jak na prawym wykresie. Zwróć uwagę, że prognozowana wartość dla każdego obszaru stanowi zawsze średnią wartość docelową próbek znajdujących się na tym obszarze. Algorytm rozdziela każdy obszar w taki sposób, żeby jak najwięcej próbek uczących znajdowało się możliwie blisko tej przewidywanej wartości.



Rysunek 6.5. Prognozy dwóch modeli regresyjnych drzew decyzyjnych

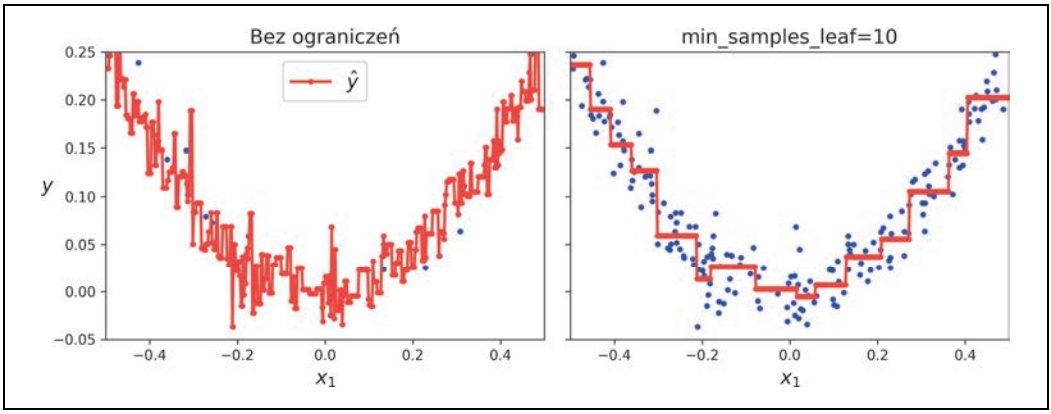
Algorytm CART działa niemal identycznie jak we wcześniejszym opisie, teraz jednak stara się rozdzielać zbiór danych uczących w sposób minimalizujący błąd MSE (a nie zanieczyszczenie). Równanie 6.4 zawiera funkcję kosztu minimalizowaną przez ten algorytm.

Równanie 6.4. Funkcja kosztu CART stosowana w regresji

$$J\left(k, t_k\right) = \frac{m_{lewy}}{m} MSE_{lewy} + \frac{m_{prawy}}{m} MSE_{prawy}$$

$$\text{gdzie} \begin{cases} MSE_{\text{węzeł}} = \sum_{i \in \text{węzeł}} \left(\hat{y}_{\text{węzeł}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{węzeł}} = \frac{1}{m_{\text{węzeł}}} \sum_{i \in \text{węzeł}} y^{(i)} \end{cases}$$

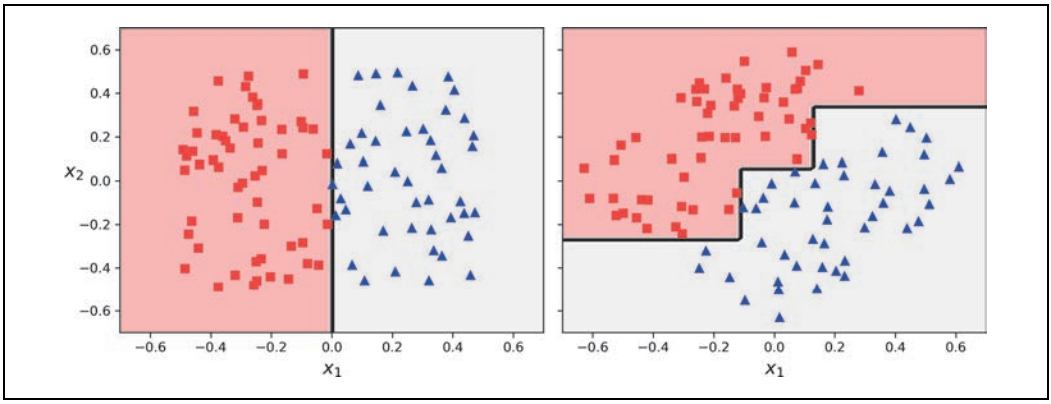
Podobnie jak w przypadku zadań klasyfikacji, drzewa decyzyjne są podatne na przetrenowanie w modelach regresji. Jeśli nie będziemy stosować żadnej formy regularyzacji (np. korzystając wyłącznie z domyślnych wartości hiperparametrów), uzyskamy prognozy widoczne na lewym wykresie rysunku 6.6. Jak widać, model jest znacznie przetrenowany. Wystarczy jednak, że wyznaczymy hiperparametr `min_samples_leaf=10`, aby uzyskać znacznie lepszy model, widoczny na prawym wykresie.



Rysunek 6.6. Przewidywania nieregularyzowanego (po lewej) i regularyzowanego (po prawej) drzewa regresyjnego

Wrażliwość na orientację osi

Mam nadzieję, że dostrzegasz już olbrzymi potencjał drzew decyzyjnych: są względnie łatwe do zrozumienia i interpretacji, przystępne, wszechstronne i wydajne. Mają jednak kilka ograniczeń. Przede wszystkim, jak pewnie zdążyłeś zauważyć, algorytm drzewa decyzyjnego uwielbia ortogonalne granice decyzyjne (wszystkie podziały są przeprowadzane prostopadle do osi odciętych), przez co model ten jest wrażliwy na orientację danych. Na przykład rysunek 6.7 przedstawia prosty, liniowo rozdzielny zestaw danych: na lewym wykresie próbki są z łatwością rozdzielane, natomiast na prawym wykresie te same dane zostały obrócone o 45° i granica decyzyjna jest tu niepotrzebnie skomplikowana. Pomimo że drzewo decyzyjne zostało perfekcyjnie dopasowane do danych, istnieje duże ryzyko, że model ten nie będzie zbyt dobrze przeprowadzał uogólniania.



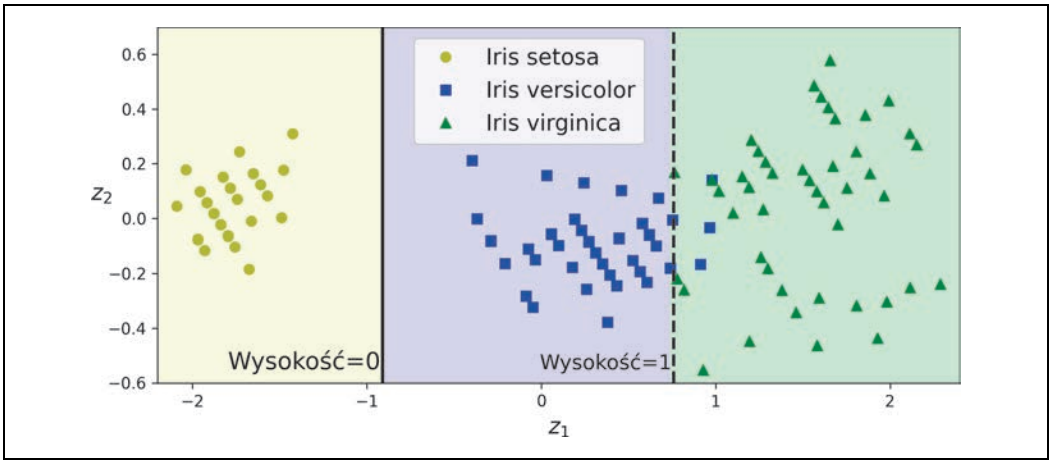
Rysunek 6.7. Wrażliwość na rotację zbioru uczącego

Jednym ze sposobów ograniczenia tego problemu jest skalowanie danych, a następnie wprowadzenie algorytmu analizy głównych składowych (PCA). W rozdziale 8. zajmiemy się PCA, na razie jednak musisz tylko wiedzieć, że rozwiązanie to obraca dane w sposób zmniejszający korelację między cechami, co często (nie zawsze) jest korzystne dla drzew decyzyjnych.

Stwórzmy mały potok skalujący dane i obracający je za pomocą PCA, a następnie wytrenujmy na nich model `DecisionTreeClassifier`. Rysunek 6.8 prezentuje granice decyzyjne tego drzewa: jak widać, dzięki rotacji możliwe jest całkiem skuteczne dopasowanie zestawu danych wyłącznie za pomocą jednej cechy, z_1, stanowiącej funkcję liniową pierwotnej długości i szerokości płatka. Kod znajduje się poniżej:

```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pca_pipeline = make_pipeline(StandardScaler(), PCA())
X_iris_rotated = pca_pipeline.fit_transform(X_iris)
tree_clf_pca = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_pca.fit(X_iris_rotated, y_iris)
```



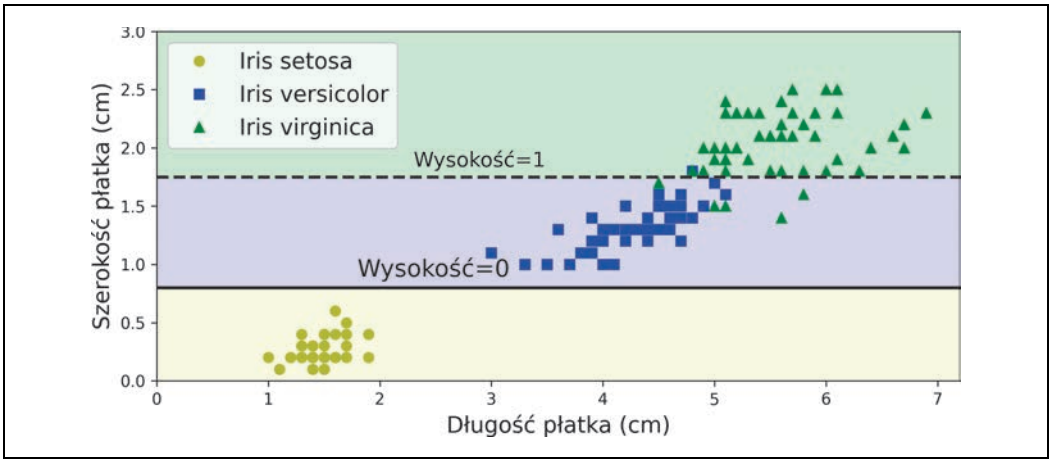
Rysunek 6.8. Granice decyzyjne drzewa na przeskalowanym i obróconym metodą PCA zestawie danych Iris

Drzewa decyzyjne mają znaczną wariancję

W bardziej ogólnym ujęciu główny problem drzew decyzyjnych polega na ich całkiem dużej wariancji: małe zmiany w hiperparametrach lub danych mogą generować bardzo różne modele. W istocie skoro algorytm uczący używany przez moduł Scikit-Learn jest stochastyczny[3] (losowo dobiera zestaw cech do oceniania każdego węzła), to nawet powtórne wytrenowanie identycznego drzewa decyzyjnego na dokładnie takich samych danych może wygenerować całkiem odmienny model, co widać na rysunku 6.9 (chyba że ustawisz hiperparametr `random_state`). Jak widać, różni się on znacznie od poprzedniego drzewa decyzyjnego (z rysunku 6.2).

Na szczęście dzięki uśrednianiu prognoz uzyskiwanych z wielu drzew jest możliwe znaczne redukowanie wariancji. Taki *zespół* drzew nazywany jest **lasem losowym** i stanowi on jeden z najpotężniejszych współczesnych rodzajów modeli, o czym przekonasz się w następnym rozdziale.

[3] Losowo dobiera zestaw cech do oceniania każdego węzła.



Rysunek 6.9. Powtórne uczenie tego samego modelu na tych samych danych może prowadzić do zupełnie odmiennego modelu

Ćwiczenia

1. Jaką przybliżoną wysokość osiągnie drzewo decyzyjne uczone (bez ograniczeń) wobec zestawu danych składającego się z miliona próbek?
2. Czy zanieczyszczenie Giniego węzła podrzędnego jest zazwyczaj większe, czy mniejsze od tego wskaźnika w węźle nadrzędnym? Czy jest on *zazwyczaj*, czy też *zawsze* mniejszy/większy?
3. Czy jeżeli drzewo decyzyjne ulega przetrenowaniu, warto próbować zmniejszyć wartość hiperparametru `max_depth`?
4. Czy dobrym pomysłem jest próba skalowania cech wejściowych, jeżeli drzewo decyzyjne wykazuje oznaki niedotrenowania?
5. Jeżeli wyuczenie drzewa decyzyjnego wobec zbioru danych składającego się z miliona próbek zajmuje godzinę, w przybliżeniu jak wiele czasu należy poświęcić na wytrenowanie drzewa przy użyciu zestawu składającego się z 10 milionów przykładów? Podpowiedź: weź pod uwagę złożoność obliczeniową algorytmów z rodziny CART.
6. Jeżeli wytrenowanie drzewa decyzyjnego na określonym zestawie danych zajmuje jedną godzinę, to jak długo w przybliżeniu zajmie ten proces po podwojeniu liczby cech?
7. Wytrenuj i dostrój model drzewa decyzyjnego wobec danych sierpowatych, korzystając z następujących kroków:
   a. Stwórz zbiór danych za pomocą funkcji `make_moons(n_samples=10000, noise=0.4)`.
   b. Rozdziel uzyskany zestaw danych na podzbiory uczący i testowy przy użyciu metody `train_test_split()`.
   c. Wykorzystaj przeszukiwanie siatki wraz ze sprawdzianem krzyżowym (przyda się klasa `GridSearchCV`), aby znaleźć dobre wartości hiperparametrów dla klasy `DecisionTreeClassifier`. Podpowiedź: wypróbuj różne wartości hiperparametru `max_leaf_nodes`.

d. Wytrenuj ten model wobec pełnego zbioru uczącego, korzystając z uzyskanych wartości hiperparametrów, a następnie sprawdź wydajność modelu wobec zestawu testowego. Powinieneś uzyskać wyniki rzędu 85 – 87%.

8. Posadź las za pomocą następujących kroków:

   a. Korzystając z poprzedniego ćwiczenia, wygeneruj 1000 podzbiorów zestawu uczącego, każdy zawierający 100 losowo dobranych próbek. Podpowiedź: możesz w tym celu skorzystać z klasy `ShuffleSplit`.

   b. Wytrenuj po jednym drzewie decyzyjnym dla każdego podzbioru, korzystając z najlepszych wartości hiperparametrów odkrytych w poprzednim ćwiczeniu. Oceń wydajność tego tysiąca drzew decyzyjnych na zestawie testowym. Drzewa te zostały wyuczone przy użyciu mniejszych zbiorów danych, dlatego prawdopodobnie będą miały gorszą dokładność od pierwotnego drzewa decyzyjnego, oscylującą w granicach 80%.

   c. Czas na odrobinę magii. Dla każdej próbki zbioru testowego wygeneruj prognozy wyliczane przez wszystkie 1000 drzew i zachowaj jedynie najczęściej powtarzający się wynik (możesz użyć do tego metody `mode()`, stanowiącej część modułu SciPy). Uzyskujesz w ten sposób **prognozy metodą głosowania większościowego** dla zbioru testowego.

   d. Oceń te przewidywania wobec zbioru testowego: powinieneś uzyskać nieco większą dokładność niż w przypadku pierwotnego modelu (wyższą o 0,5% – 1,5%). Gratulacje, właśnie wytrenowałeś swój pierwszy klasyfikator losowego lasu!

Rozwiązania tych ćwiczeń znajdziesz na końcu notatnika Jupyter zawierającego kod tego rozdziału, dostępnego pod adresem *https://ftp.helion.pl/przyklady/uczem3.zip* (po polsku) lub *https://homl.info/colab3* (po angielsku).

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

Z

Twórz i trenuj nowoczesne sieci neuronowe!

Znajdziesz tu rozsądne, intuicyjne objaśnienia, a także mnóstwo praktycznych porad!

François Chollet, twórca interfejsu Keras

Głębokie sieci neuronowe mają niesamowity potencjał. Osiągnięcia ostatnich lat nadały procesom uczenia głębokiego zupełnie nową jakość. Obecnie nawet programiści niezaznajomieni z tą technologią mogą korzystać z prostych i niezwykle skutecznych narzędzi, pozwalających na sprawne implementowanie programów uczących się z danych.

To trzecie wydanie bestsellerowego przewodnika po uczeniu maszynowym. Książka jest adresowana do osób, które chcą wejść w świat uczenia maszynowego — przy czym wystarczą do tego minimalne umiejętności programistyczne. Zawarto tu minimum teorii, a proces nauki ułatwiają liczne przykłady i ćwiczenia. Dzięki temu przyswoisz niezbędne pojęcia i nauczysz się korzystać z gotowych platform produkcyjnych Pythona: Scikit-Learn, Keras i TensorFlow. W tym wydaniu pokazano różnorodne techniki, od prostej regresji liniowej aż po głębokie sieci neuronowe. Szybko nauczysz się tworzyć działające systemy inteligentne!

W książce między innymi:

- korzystanie ze Scikit-Learn, z TensorFlow i Keras
- modele: maszyny wektorów nośnych, drzewa decyzyjne, lasy losowe i metody zespołowe
- uczenie nienadzorowane: redukcja wymiarowości, analiza skupień, wykrywanie anomalii
- sieci neuronowe: sieci splotowe, rekurencyjne, modele dyfuzyjne i transformatory
- trenowanie i implementacje sieci neuronowych

Aurélien Géron jest konsultantem i wykładowcą. Studiował mikrobiologię i genetykę ewolucyjną, pracował w Google, JP Morgan i Société Générale, a także w Ministerstwie Obrony Kanady. Napisał kilka książek o programowaniu w języku C++, sieciach Wi-Fi i architekturze sieci internetowych. Uczył też informatyki na francuskiej politechnice.

To znakomite wprowadzenie do teoretycznych i praktycznych rozważań na temat rozwiązywania problemów za pomocą sieci neuronowych!

Pete Warden, mobile lead projektu TensorFlow

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-8322-423-7
9 788383 224237
Cena: 179,00 zł