O'REILLY®

Wydanie II
Aktualizacja
do modułu TensorFlow 2

Uczenie maszynowe

z użyciem Scikit-Learn 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, 2nd Edition

Tłumaczenie: Krzysztof Sawka

ISBN: 978-83-283-6002-0

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/uczem2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 6.

Drzewa decyzyjne

Drzewa decyzyjne (ang. *decision trees*), podobnie jak maszyny wektorów nośnych, 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 `DecisionTree ↪Regressor` 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()
X = iris.data[:, 2:] # Długość i szerokość płatka
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```

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

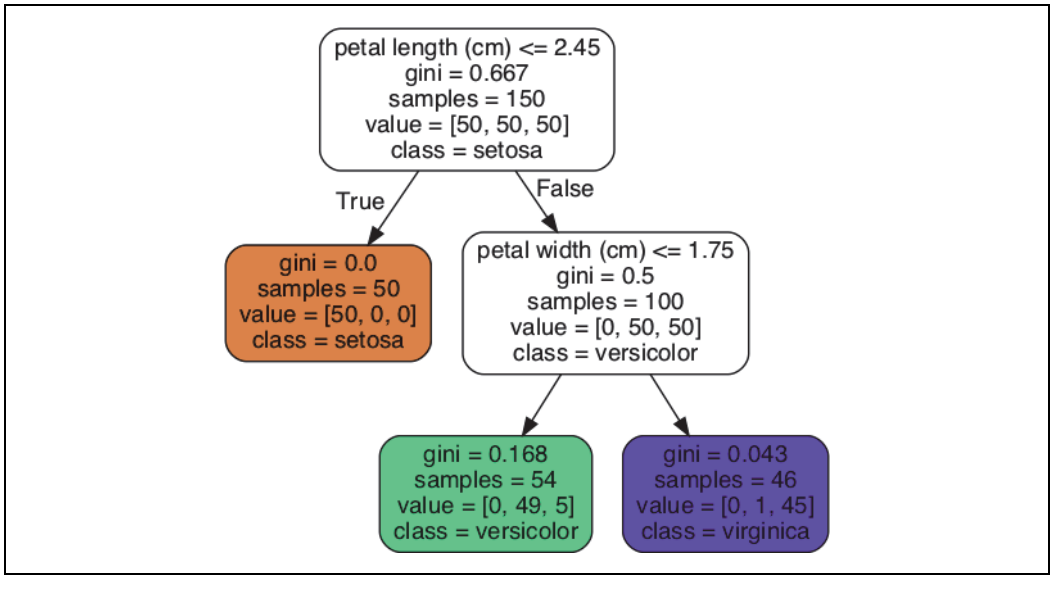
```
from sklearn.tree import export_graphviz

export_graphviz(
    tree_clf,
```

```
        out_file=image_path("iris_drzewo.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

Możemy teraz przekonwertować wynikowy plik *.dot* na różne formaty (np. *.pdf* lub *.png*) za pomocą narzędzia wiersza poleceń `dot`, dostępnego w pakiecie *graphviz*[1]. Poniższe polecenie przekształca plik *.dot* do formatu obrazu *.png*:

```
$ dot -Tpng iris_drzewo.dot -o iris_drzewo.png
```

Nasze 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. Zaczynamy od **korzenia** (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`).

[1] Graphviz jest otwartym pakietem oprogramowania służącego do wizualizowania grafów, dostępnym pod adresem *http://www.graphviz.org/*.

Załóżmy teraz, że znajdujemy kolejny kwiat kosaćca, którego długość płatka przekracza teraz 2,45 cm. Musimy teraz skierować się ku prawemu węzłowi potomnemu (wysokość 1, po prawej), który nie jest liściem, 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: np. 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** (ang. *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 **wskaźnik Giniego** (ang. *Gini impurity*) jest równy 0. Równanie 6.1 pokazuje, w jaki sposób algorytm wylicza wskaźnik Giniego G_i dla i-tego węzła. Węzeł znajdujący się na wysokości 2 po lewej stronie ma wskaźnik Giniego o wartości $1–(0{:}54)^2–(49{:}54)^2–(5{:}54)^2 \approx 0{,}168$.

Równanie 6.1. Wskaźnik Giniego

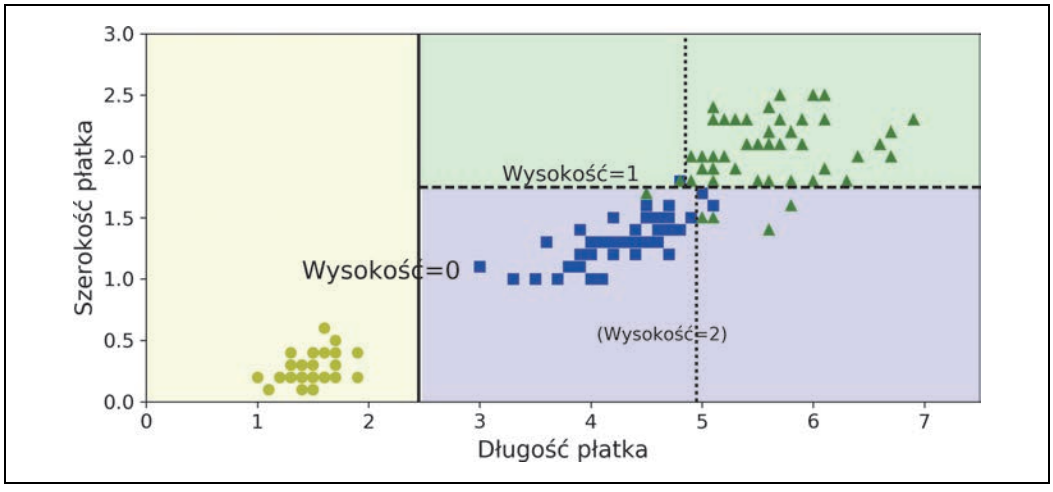
$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^{\ 2}$$

W tym równaniu:

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

Moduł Scikit-Learn wykorzystuje algorytm CART generujący wyłącznie **drzewa binarne** (ang. *binary trees*): węzły niebędące liśćmi zawsze mają 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 korzenia (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 linię kropkowaną).



Rysunek 6.2. Granice decyzyjne drzewa decyzyjnego

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 przekonamy, algorytmy losowego lasu lub 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 ciężko 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).

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]])
array([[0. , 0.90740741, 0.09259259]])
>>> 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 — np. 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:

- $G_{lewy/prawy}$ — miara zanieczyszczenia lewego/prawego podzbioru,
- $m_{lewy/prawy}$ — liczba próbek w lewym/prawym podzbiorze.

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**[2]. 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.

[2] P stanowi zbiór problemów, jakie mogą zostać rozwiązane w wielomianowym czasie, z kolei 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).

Złożoność obliczeniowa

Wyliczanie prognoz wymaga poruszania się po drzewie decyzyjnym od korzenia 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[3]. W każdym węźle wymagane jest jedynie sprawdzenie wartości jednej cechy, zatem 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 $O(n \times m\ log_2(m))$. W przypadku niewielkich zbiorów uczących (zawierających do kilku tysięcy próbek) moduł Scikit-Learn może przyśpieszyć proces uczenia za pomocą wstępnego sortowania danych (`presort=True`), jednak włączenie tej funkcji powoduje znaczne spowolnienie trenowania za pomocą dużych zestawów uczących.

Wskaźnik Giniego czy entropia?

Domyślnie używany jest wskaźnik 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[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 $-\frac{49}{54}\log_2\left(\frac{49}{54}\right)-\frac{5}{54}\log_2\left(\frac{5}{54}\right)\approx 0,$.

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 sytu-

[3] Zapis log_2 oznacza logarytm binarny (dwójkowy). Jest on równy $log_2(m) = log(m)/log(2)$.

[4] Proces zmniejszania entropii często jest określany mianem **przyrostu informacji** (ang. *information gain* — IG).

acjach, w których obydwa wskaźniki się różnią, wskaźnikowi Giniego zdarza się izolować najczęściej występującą klasę w osobnej gałęzi, natomiast entropia generuje nieco bardziej zrównoważone drzewa[5].

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, zatem cechuje je ograniczenie stopni swobody, dzięki czemu zmniejszamy ryzyko przetrenowania (ale jednocześnie zwiększamy możliwość niedotrenowania).

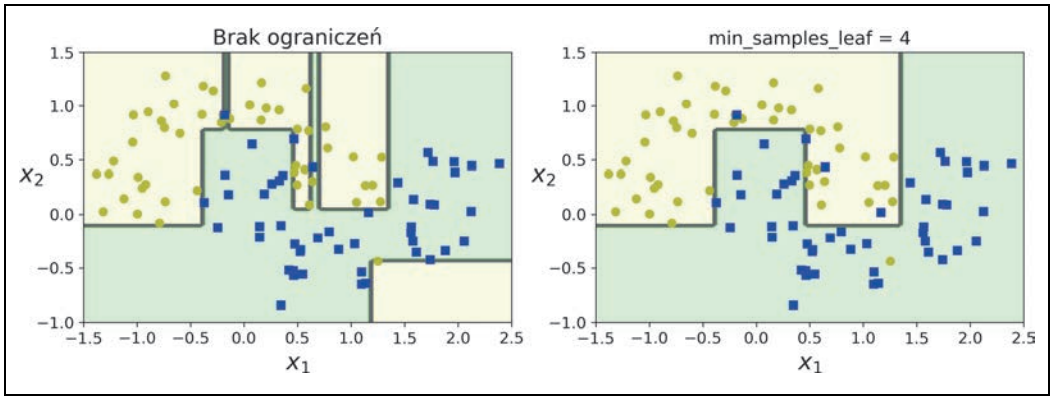
Aby uniknąć przetrenowania modelu, musimy ograniczyć swobodę algorytmu drzewa decyzyjnego podczas uczenia. Wiemy już, że ten proces nosi nazwę regularyzacji. Hiperparametry regularyzacyjne zależą od stosowanego algorytmu, zazwyczaj jednak możemy ograniczyć przynajmniej maksymalną wysokość drzewa. W module 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: `min_samples_split` (minimalna liczba próbek, jakie muszą znajdować się w węźle, zanim zostanie podzielony), `min_samples_leaf` (minimalna liczba próbek, jakie muszą znajdować się w liściu), `min_weight_fraction_leaf` (podobny do parametru `min_samples_leaf`, tu jednak wartością jest ułamek całkowitej liczby ważonych próbek), `max_leaf_nodes` (maksymalna liczba liści) oraz `max_features` (maksymalna liczba cech używanych do dzielenia poszczególnych węzłów). 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.

[5] Więcej szczegółów znajdziesz w interesującej analizie (*https://sebastianraschka.com/faq/docs/decision-tree-binary.html*), której autorem jest Sebastian Raschka.

Rysunek 6.3 przedstawia dwa drzewa decyzyjne wyuczone wobec zestawu danych sierpowatych (mieliśmy z nimi do czynienia w rozdziale 5.). Drzewo po lewej stronie zostało wytrenowane za pomocą standardowych hiperparametrów (tj. bez ograniczeń), natomiast w drzewie po prawej wyznaczyliśmy hiperparametr `min_samples_leaf=4`. Wyraźnie widać, że lewy model jest przetrenowany, natomiast prawy będzie prawdopodobnie lepiej przeprowadzał uogólnianie.



Rysunek 6.3. Regularyzacja za pomocą hiperparametru min_samples_leaf

Regresja

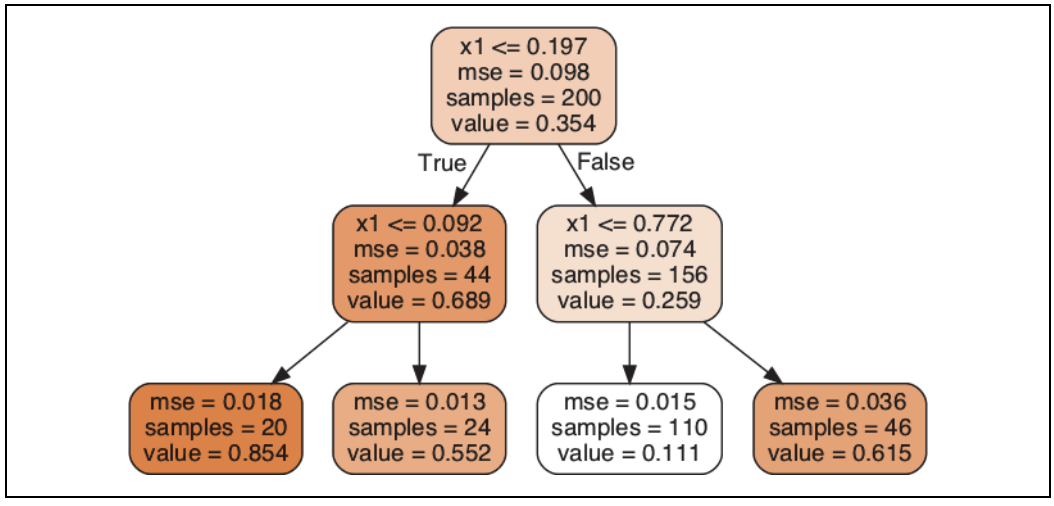
Drzewa decyzyjne mogą również wykonywać zadania regresyjne. Stwórzmy takie drzewo regresyjne za pomocą klasy `DecisionTreeRegressor`; wyuczymy je wobec zaszumionego, kwadratowego zbioru danych, a jego maksymalną wysokość ustalmy na poziomie `max_depth=2`:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2)
tree_reg.fit(X, y)
```

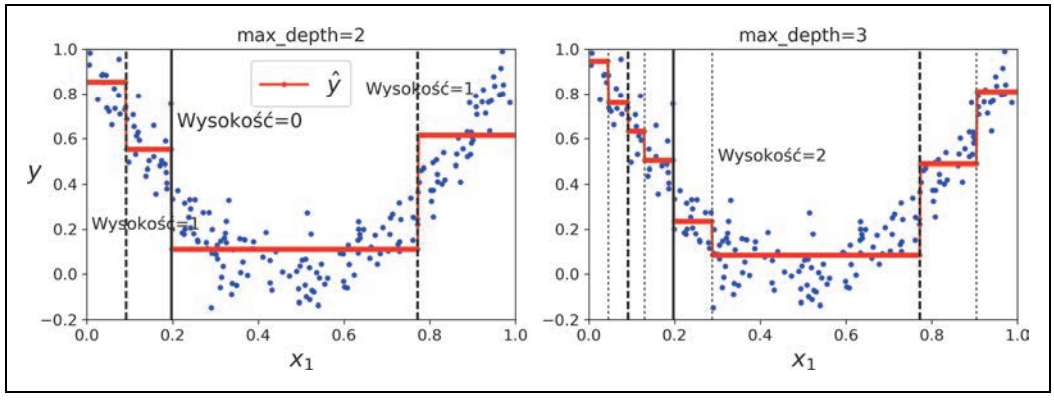
Rysunek 6.4 przedstawia wygenerowane drzewo decyzyjne.

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 $x_1 = 0{,}6$. Poruszamy się po drzewie, począwszy od korzenia, i w końcu docieramy do liścia przewidującego `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 6.4. Regresyjne drzewo decyzyjne



Rysunek 6.5. Prognozy dwóch modeli regresyjnych drzew decyzyjnych

Algorytm CART działa niemal tak samo, jak w czasie klasyfikacji, 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

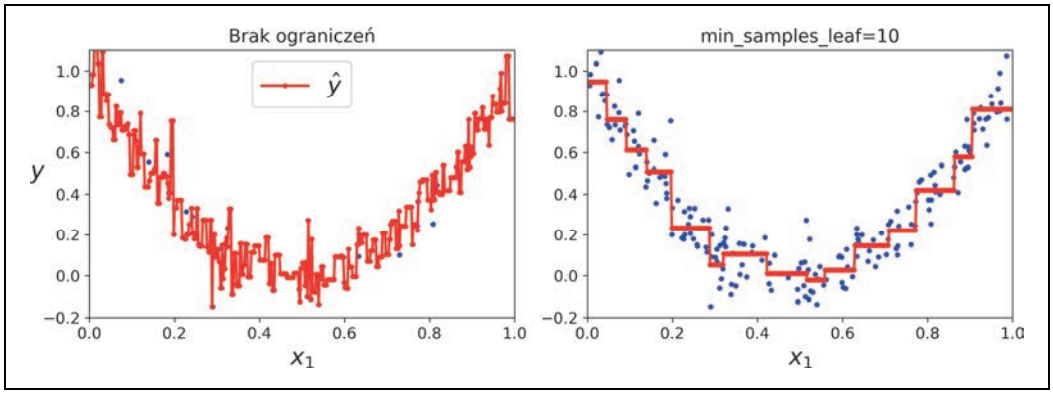
$$J\left(k, t_k\right) = \frac{m_{lewy}}{m} MSE_{lewy} + \frac{m_{prawy}}{m} MSE_{prawy}$$

gdzie:

$$MSE_{\text{węzeł}} = \sum_{i \in \text{węzeł}} \left(\hat{y}_{\text{węzeł}} - y^{(i)}\right)^2$$

$$\hat{y}_{\text{węzeł}} = \frac{1}{m_{\text{węzeł}}} \sum_{i \in \text{węzeł}} y^{(i)}$$

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.



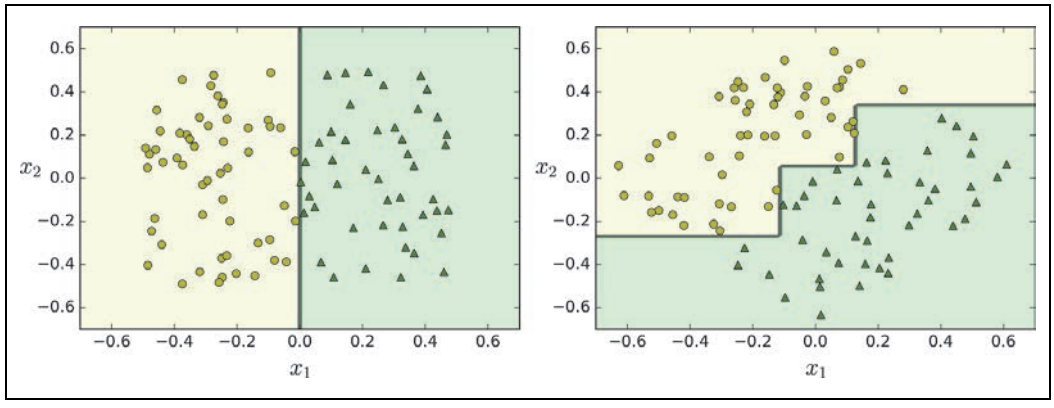
Rysunek 6.6. Regularyzacja regresyjnego drzewa decyzyjnego

Niestabilność

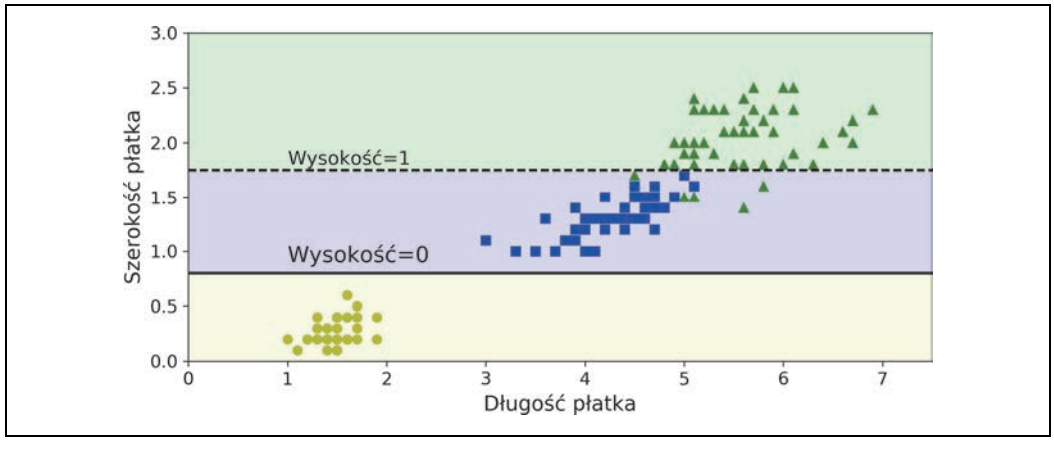
Mam nadzieję, że dostrzegasz już olbrzymi potencjał drzew decyzyjnych: są zrozumiałe i łatwe do interpretacji, przystępne, wszechstronne i wydajne. Mają jednak kilka ograniczeń. Po pierwsze, jak pewnie zdążyłaś/zdążyłeś zauważyć, algorytm drzewa decyzyjnego uwielbia ortogonalne granice decyzyjne (wszystkie podziały są przeprowadzane prostopadle do osi odciętych), przez co model ten jest wrażliwy na rotację zbioru uczącego. 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 tego, że drzewo decyzyjne zostało perfekcyjnie dopasowane do danych, istnieje duże ryzyko, że model ten nie będzie zbyt dobrze przeprowadzał uogólniania. Jednym ze sposobów ograniczenia tego problemu jest wprowadzenie algorytmu analizy głównych składowych (PCA; zob. rozdział 8.), dzięki któremu model staje się lepiej zorientowany wobec danych uczących.

W bardziej ogólnym ujęciu główny problem drzew decyzyjnych polega na ich olbrzymiej czułości na niewielką wariancję w zestawie danych uczących. Na przykład, jeśli usuniesz tylko najszerszy płatek kosaćca *Iris versicolor* ze zbioru danych *Iris* (jego płatki mają długość 4,8 cm i szerokość 1,8 cm) i wyuczysz nowe drzewo decyzyjne, możesz uzyskać model zaprezentowany na rysunku 6.8. Jak widać, różni się on znacznie od wygenerowanego wcześniej drzewa (rysunek 6.2). W rzeczywistości algorytm uczący używany przez moduł Scikit-Learn jest stochastyczny[6], zatem możesz uzyskiwać zupełnie odmienne modele wobec tego samego zestawu danych uczących (chyba że używasz hiperparametru `random_state`).

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



Rysunek 6.7. Wrażliwość na rotację zbioru uczącego



Rysunek 6.8. Wrażliwość na niuanse zbioru danych uczących

Jak się przekonamy w następnym rozdziale, losowe lasy mogą ograniczać tę niestabilność poprzez uśrednianie prognoz uzyskiwanych z wielu drzew.

Ćwiczenia

1. Jaką przybliżoną wysokość osiągnie drzewo decyzyjne uczone (bez ograniczeń) wobec zestawu danych składającego się z miliona próbek?
2. Czy wskaźnik Giniego węzła podrzędnego jest zazwyczaj większy lub mniejszy 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?
6. Jeżeli zbiór danych zawiera 100 000 próbek, to czy wyznaczenie parametru `presort=True` przyśpieszy proces uczenia?
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. Powinnaś/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: powinnaś/powinieneś uzyskać nieco większą dokładność niż w przypadku pierwotnego modelu (wyższą o 0,5% – 1,5%). Gratulacje, właśnie wytrenowałaś/wytrenowałeś swój pierwszy klasyfikator losowego lasu!

Rozwiązania tych ćwiczeń znajdziesz w dodatku A.

TensorFlow 2: źródło magii zaawansowanych technologii!

W 2006 roku świat nauki zafascynował się głębokimi sieciami neuronowymi. Wbrew wcześniejszym przekonaniom okazało się, że ich uczenie jest możliwe. Technika ta została nazwana uczeniem głębokim. Wymagała zapewnienia olbrzymiej mocy obliczeniowej i potężnych ilości danych, jednak potencjał wytrenowanych sieci głębokich był niesamowity. Kolejne lata przyniosły bujny rozwój tej technologii w wielu obszarach, co pozwoliło na tworzenie przeróżnych zaawansowanych produktów. Prace nad nowymi zastosowaniami sieci głębokich trwają. Wszystko wskazuje na to, że już wkrótce zdominują one większość dziedzin naszego życia.

To drugie wydanie bestsellerowego przewodnika po technikach uczenia maszynowego. Wystarczą minimalne umiejętności programistyczne, aby dzięki tej książce nauczyć się budowania i trenowania głębokiej sieci neuronowej. Zawarto tu minimum teorii, a proces nauki jest ułatwiony przez liczne przykłady i ćwiczenia. Wykorzystano gotowe rozwiązania i przedstawiono zasady pracy ze specjalistycznymi narzędziami, w tym z TensorFlow 2, najnowszą odsłoną modułu. W efekcie niepostrzeżenie przyswoisz niezbędny zasób pojęć i narzędzi służących do tworzenia systemów inteligentnych. Poznasz różnorodne techniki i zaczniesz samodzielnie ich używać. Po lekturze będziesz biegle posługiwać się najnowszymi technologiami sztucznej inteligencji!

W książce między innymi:

- podstawy uczenia maszynowego i rozpoczęcie pracy z TensorFlow
- techniki wykrywania obiektów, segmentacji semantycznej i mechanizmy uwagi
- interfejs Keras, narzędzia TF Transform i TF Serving
- wdrażanie modeli TensorFlow
- techniki uczenia nienadzorowanego, wykrywanie anomalii oraz biblioteka TF Agents

Aurélien Géron jest konsultantem w dziedzinie uczenia maszynowego i wykładowcą, a także założycielem i dyrektorem do spraw technicznych w kilku różnych firmach technicznych. W przeszłości studiował mikrobiologię i genetykę ewolucyjną; pracował w firmach: Google, JPMorgan i Société Générale oraz w Ministerstwie Obrony Kanady. Napisał kilka książek o programowaniu w języku C++, sieciach Wi-Fi oraz architekturze sieci internetowych. Wykładał też informatykę na politechnice we Francji.

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 978-83-283-6002-0
9 788328 360020
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 129,00 zł