Helion

40 algorytmów, które powinien znać każdy programista

Nauka implementacji algorytmów w Pythonie

Packt›

Imran Ahmad

Tytuł oryginału: 40 Algorithms Every Programmer Should Know: Hone your problem-solving skills by learning different algorithms and their implementation in Python

Tłumaczenie: Katarzyna Bogusławska

ISBN: 978-83-283-7777-6

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/40algo.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/40algo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 7

Tradycyjne algorytmy uczenia nadzorowanego

W tym rozdziale skupimy się na algorytmach uczenia nadzorowanego, które stanowią jeden z najważniejszych współcześnie typów algorytmów. Cechą wyróżniającą algorytmy uczenia nadzorowanego jest wykorzystanie danych z etykietami do wytrenowania modelu. W tej książce tego typu algorytmy omawiane są w dwóch rozdziałach. W tym omówię tradycyjne algorytmy nadzorowanego uczenia maszynowego z pominięciem sieci neuronowych. Kolejny będzie poświęcony implementacji takich algorytmów za pomocą sieci neuronowych. Wynika to z faktu, że przy tak intensywnym rozwoju na tym polu sieci neuronowe stały się tematem-rzeką, który trudno streścić w jednym rozdziale.

Zatem ta sekcja otwiera część książki dotyczącą uczenia nadzorowanego. Najpierw wprowadzę podstawowe pojęcia związane z uczeniem nadzorowanym. Następnie przedstawię dwa rodzaje modeli — klasyfikatory i regresję. Aby pokazać możliwości klasyfikatorów, postawię przed Tobą wyzwanie w postaci rzeczywistego problemu do rozwiązania. Potem zaprezentuję sześć różnych algorytmów klasyfikujących pozwalających na rozwiązanie tego problemu. Dalej spojrzymy na algorytmy regresji, zaczynając od zmierzenia się z podobnym problemem, który będzie trzeba rozwiązać za pomocą regresji. Tu przedstawię trzy możliwości rozwiązania takiego wyzwania. Wreszcie porównamy wyniki, by ułatwić sobie podsumowanie pojęć zawartych w tym rozdziale.

Ogólnym celem tego rozdziału jest umożliwienie zrozumienia różnych rodzajów technik uczenia nadzorowanego i wybranie najwłaściwszego algorytmu dla danej klasy problemów.

Oto główne zagadnienia poruszane w tym rozdziale:

- nadzorowane uczenie maszynowe,
- algorytmy klasyfikacji,
- metody oceny efektywności klasyfikatorów,

- algorytmy regresji,
- metody oceny efektywności algorytmów regresji.

Powinniśmy jednak zacząć od wyjaśnienia podstawowych pojęć związanych z nadzorowanym uczeniem maszynowym.

Nadzorowane uczenie maszynowe

Uczenie maszynowe wykorzystuje podejście oparte na danych do tworzenia autonomicznych systemów wspierających podejmowanie decyzji czy to pod nadzorem człowieka, czy bez niego. Aby zbudować te autonomiczne systemy, uczenie maszynowe korzysta z grupy algorytmów i metodologii pozwalających odkryć i nazwać powtarzające się w zbiorze danych wzorce. Jedną z najbardziej popularnych i przynoszących najlepsze efekty technik w uczeniu maszynowym jest podejście nadzorowanego uczenia maszynowego. W tym podejściu algorytmowi dostarczany jest zbiór danych wejściowych nazywanych cechami i odpowiadających im danych wyjściowych, nazywanych **zmiennymi celu**. Przy użyciu pewnego zbioru danych algorytm nadzorowanego uczenia maszynowego trenuje model, który jest w stanie uchwycić złożone relacje między cechami a zmiennymi celu w postaci wzoru matematycznego. Wytrenowany model staje się potem punktem wyjścia do prognoz.

Te prognozy tworzy się poprzez wygenerowanie zmiennych celu na nieznanym uprzednio zbiorze cech za pomocą wytrenowanego modelu.

> Zdolność uczenia się na podstawie istniejących danych w uczeniu nadzorowanym przypomina umiejętność ludzkiego mózgu do nauki na podstawie doświadczenia. Ta możliwość uczenia nadzorowanego wykorzystuje jedną z cech ludzkiego mózgu i stanowi kamień milowy w wyposażaniu maszyn w zdolności podejmowania decyzji, a nawet inteligencję.

Prześledźmy, co algorytm nadzorowanego uczenia maszynowego musiałby zrobić, aby wytrenować model potrafiący oznaczyć wiadomości e-mail jako autentyczne, przydatne treści (nazywane autentykami) lub niechcianą korespondencję (nazywaną **spamem**). Po pierwsze, potrzebujemy zbioru starych przykładów, na podstawie których maszyna będzie mogła nauczyć się, jaki rodzaj treści klasyfikować jako spam. Ta nauka oparta na analizie treści jest złożonym procesem korzystającym z jednego z algorytmów nadzorowanego uczenia maszynowego. Do przykładów takich algorytmów należą drzewa decyzyjne i naiwny klasyfikator bayesowski, które będą omawiane w dalszej części tego rozdziału.

Żargon nadzorowanego uczenia maszynowego

Zanim zajmiemy się szczegółami algorytmów nadzorowanego uczenia maszynowego, warto zdefiniować kilka podstawowych terminów z tej dziedziny.

Pojęcie	Wyjaśnienie
Zmienna celu	Zmienna celu to zmienna, której wartość ma przewidzieć nasz model. W modelu nadzorowanego uczenia maszynowego może być tylko jedna zmienna celu.
Etykieta	Jeśli zmienna celu, którą chcemy przewidzieć, jest cechą jakościową, nazywamy ją etykietą.
Cechy	Zbiór danych wejściowych używany do przewidzenia zmiennej celu to cechy.
Inżynieria cech	Przetwarzanie cech w celu przygotowania ich na potrzeby wybranego algorytmu uczenia maszynowego to inżynieria cech.
Wektor cech	Wszystkie cechy zebrane razem w jedną strukturę przed dostarczeniem ich w charakterze danych wejściowych do algorytmu nazywamy wektorem cech.
Dane historyczne	Dane z przeszłości, które są wykorzystywane do sformułowania związków między zmienną celu a cechami, to dane historyczne. Historyczne dane obejmują przykłady.
Zbiór treningowy/testowy	Dane historyczne dzieli się na dwie części — większy zbiór nazywany treningowym, a mniejszy nazywany testowym.
Model	Matematyczne sformułowanie schematu, który najlepiej ujmuje związek między zmienną celu a cechami.
Trenowanie	Tworzenie modelu na podstawie zbioru treningowego.
Testowanie	Ocena jakości wytrenowanego modelu za pomocą zbioru testowego.
Przewidywanie	Zastosowanie modelu do stworzenia prognozy wartości zmiennej celu.

Wytrenowany model nadzorowanego uczenia maszynowego jest w stanie dokonywać prognoz za pomocą szacowania wartości zmiennej celu na podstawie cech.

W tym rozdziale przy omówieniu technik uczenia maszynowego będę posługiwał się też charakterystyczną notacją, którą definiuję poniżej:

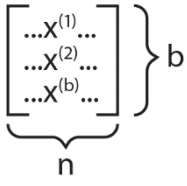
Zmienna	Znaczenie
y	Rzeczywista etykieta
$\acute{y}$	Przewidziana etykieta
d	Łączna liczba próbek
b	Liczba próbek treningowych
c	Liczba próbek testowych

Spójrzmy teraz, jak tą terminologią posługiwać się w praktyce.

Jak wspominałem, wektor cech to struktura danych przechowująca wszystkie cechy.

Jeśli liczba cech wynosi n, a liczba próbek treningowych to b, to *X_train* przedstawia wektor cech treningowych. Każda próbka to wiersz w wektorze cech.

Dla zbioru treningowego wektor cech reprezentowany jest przez *X_train*. Jeśli w zbiorze treningowym jest b próbek, to wektor *X_train* będzie miał b wierszy. Jeśli każda próbka ma n zmiennych, wektor *X_train* będzie miał n kolumn. Zatem rozmiary zbioru treningowego będą wynosiły $n \times b$, co widać na poniższym diagramie:



Przyjmijmy, że istnieje b próbek treningowych i c próbek testowych. Konkretną próbkę treningową będzie wtedy przedstawiało wyrażenie (X, y).

Używamy indeksu górnego, by określić, która próbka treningowa jest którą w zbiorze treningowym.

Zatem nasz zbiór danych z etykietami przedstawia $D = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(d)}, y^{(d)})\}$.

Podzielimy go następnie na dwie części — D_{train} i D_{test}.

W tej sytuacji zbiór treningowy można zapisać jako $D_{train} = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(b)}, y^{(b)})\}$.

Celem trenowania modelu jest sytuacja, w której dla i-tej próbki danych w zbiorze treningowym przewidziana wartość zmiennej celu była jak najbliższa rzeczywistej wartości w próbce. Innymi słowy, $\acute{y}^{(i)} \approx y^{(i)}$ dla $1 \leq i \leq b$.

Zatem zbiór testowy można przedstawić następująco: $D_{test} = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(c)}, y^{(c)})\}$.

Wartości zmiennych celu przedstawia wektor Y:

$$Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\}$$

Warunki konieczne

Nadzorowane uczenie maszynowe opiera się na zdolności algorytmu do wytrenowania modelu na podstawie próbek. Aby tego typu algorytm działał poprawnie i efektywnie, muszą zostać spełnione pewne warunki konieczne, mianowicie:

- **Próbek musi być wystarczająco dużo.** Algorytmy nadzorowanego uczenia maszynowego potrzebują odpowiedniej liczby próbek, by wytrenować model.
- **W danych historycznych muszą istnieć schematy.** Próbki użyte do trenowania modelu muszą zawierać jakieś wzorce czy schematy. Prawdopodobieństwo

wystąpienia zdarzenia, jakie nas interesuje, powinno zależeć od kombinacji wzorców, tendencji i zdarzeń. Bez nich mamy do czynienia z danymi losowymi, na podstawie których nie wytrenujemy żadnego modelu.

- **Założenia muszą być poprawne.** Gdy trenujemy model nadzorowanego uczenia maszynowego, oczekujemy, że założenia, jakie mają zastosowanie do próbek obecnie, będą aktualne także w przyszłości. Przyjrzyjmy się przykładowi. Jeśli chcemy wytrenować model, który będzie potrafił przewidzieć dla agencji rządowej prawdopodobieństwo, czy dany wniosek wizowy zostanie zaakceptowany czy odrzucony, zakładamy, że dysponujemy prawem i regulacjami, które nie zmienią się w czasie, gdy model będzie używany do dokonywania przewidywań. Jeśli w tym czasie w życie wejdzie nowa legislacja, model będzie trzeba wytrenować ponownie, by obejmował on nowe informacje.

Rozróżnienie między klasyfikatorami a regresorami

W modelu uczenia maszynowego zmienna celu może być zmienną jakościową lub ciągłą. Rodzaj zmiennej celu będzie określał, jakiego typu modelu potrzebujemy. Ogólnie wyróżniamy dwa główne typy modeli uczenia maszynowego:

- **Klasyfikatory.** Jeśli zmienna celu jest zmienną jakościową, to model uczenia maszynowego, jaki zostanie użyty do jej przewidzenia, będzie klasyfikatorem. Klasyfikatory mogą być użyte do odpowiedzenia na poniższe pytania biznesowe:
  - Czy ten ponadstandardowy przyrost tkanki jest złośliwym guzem?
  - W oparciu o obecne warunki pogodowe, czy jutro będzie padać?
  - W oparciu o profil konkretnego kandydata, czy jego wniosek o kredyt hipoteczny powinien zostać zaakceptowany?
- **Regresory.** Jeśli zmienna celu jest zmienną ciągłą, będziemy trenować regresor. Dzięki nim można odpowiedzieć na następujące pytania:
  - Biorąc pod uwagę obecne warunki pogodowe, jak dużo deszczu spadnie jutro?
  - Jaka będzie cena domu o konkretnej charakterystyce?

Każdemu z rodzajów modeli poświęcę osobną sekcję.

Algorytmy klasyfikujące

Jeśli zmienna celu jest zmienną jakościową w nadzorowanym uczeniu maszynowym, model, który taką zmienną skategoryzuje, nazywamy klasyfikatorem.

- Zmienną celu nazywamy **etykietą**.
- Dane historyczne nazywamy **danymi etykietowanymi** (danymi z etykietami).
- Dane produkcyjne, których etykietę należy przewidzieć, nazywane są **danymi nieetykietowanymi** (danymi bez etykiet).

Zdolność dokładnego przypisania etykiet nieetykietowanym danym za pomocą wytrenowanego modelu jest prawdziwą siłą algorytmów klasyfikujących. Klasyfikatory przewidują etykiety dla nieetykietowanych danych, by odpowiedzieć na konkretne pytania biznesowe.

Zanim zagłębimy się w szczegóły algorytmów klasyfikujących, poświęćmy chwilę na przedstawienie problemów biznesowych, jakie będziemy starali się rozwiązać za pomocą klasyfikatorów. Następnie skorzystamy z sześciu różnych algorytmów, by rozwiązać to samo zadanie, dzięki czemu będziemy mogli porównać ich metody, podejście i efektywność.

Wyzwanie dla klasyfikatorów

Przedstawię jeden problem, który będziemy rozwiązywać za pomocą sześciu różnych algorytmów klasyfikujących. W dalszej części rozdziału nazywam ten problem wyzwaniem dla klasyfikatorów. Użycie sześciu algorytmów do rozwiązania jednego problemu będzie dla nas ważne z dwóch względów:

- Wszystkie dane wejściowe muszą zostać przetworzone i zgromadzone w złożonej strukturze danych, jaką jest wektor cech. Użycie tego samego wektora pozwala nam uniknąć powtarzania etapu przygotowania danych dla sześciu algorytmów.
- Będziemy w stanie porównać efektywność różnych algorytmów, ponieważ używamy tego samego wektora cech jako danych wejściowych.

Wyzwanie dla klasyfikatorów polega na przewidzeniu prawdopodobieństwa, że dany klient dokona zakupów w sklepie. W handlu jednym z czynników mogących zwiększyć sprzedaż jest lepsze zrozumienie zachowania klientów. Można to osiągnąć poprzez analizę schematów wykrytych w danych historycznych. Zacznijmy więc od definicji problemu.

Definicja problemu

Czy na podstawie danych historycznych możemy wytrenować klasyfikator binarny, który w oparciu o profil klienta będzie potrafił przewidzieć, czy dany użytkownik kupi pewien produkt?

Po pierwsze, zbadajmy zbiór etykietowanych danych historycznych, który jest dostępny do rozwiązania tego problemu:

$$x \in \Re^b, y \in \{0,1\}$$

Dla danej próbki, jeśli $y = 1$, nazwiemy taką klasyfikację pozytywną, a gdy $y = 0$ — negatywną.

Chociaż poziom pozytywnej i negatywnej klasyfikacji można wybrać arbitralnie, dobrą praktyką jest definiowanie klasyfikacji pozytywnej jako interesującego nas zdarzenia. Jeśli chcemy oznaczać podejrzane transakcje dla banku, klasyfikacją pozytywną (tj. $y = 1$) powinna być transakcja podejrzana, a nie na odwrót.

Przyjrzyjmy się też:

- rzeczywistej etykiecie, oznaczanej jako y,
- przewidywanej etykiecie, oznaczanej jako $\acute{y}$.

Pamiętaj, że w naszym wyzwaniu dla klasyfikatorów rzeczywistą wartość etykiety dla każdej próbki przedstawia y. Jeśli w przypadku pewnej próbki ktoś kupił dany produkt, mówimy, że $y = 1$. Wartości przewidywane reprezentowane są przez $\acute{y}$. Wektor cech danych wejściowych — x — ma rozmiar 4. Chcemy określić, jakie jest prawdopodobieństwo, że użytkownik ponownie dokona zakupu przy konkretnych danych wejściowych.

Zatem chcemy określić prawdopodobieństwo $y = 1$ przy pewnych wartościach wektora cech x. Matematycznie przedstawimy to następująco:

$\acute{y} = P(y = 1|x)$, gdzie $x \in \Re^{n_x}$.

Sprawdźmy teraz, jak przetworzyć i zebrać różne dane wejściowe w wektor cech x. Metoda zbierania różnych części zbioru x za pomocą kolejki przetwarzania jest szczegółowo omówiona w kolejnej sekcji.

Inżynieria cech w przetwarzaniu potokowym

Przygotowanie danych do wybranego algorytmu uczenia maszynowego nazywamy inżynierią cech i stanowi ono kluczowy etap cyklu życia uczenia maszynowego. Zachodzi on w różnych fazach. Wieloetapowy proces służący do przetwarzania danych nazywamy **przetwarzaniem potokowym** (*data pipeline*). Sprawienie, że tego typu przetwarzanie będzie się opierało na ustandaryzowanych krokach wszędzie, gdzie to możliwe, umożliwi jego powtórne wykorzystanie i zmniejszy wysiłek niezbędny do wytrenowania modelu. Dzięki użyciu dobrze przetestowanych modułów oprogramowania rośnie także jakość kodu.

Spójrzmy, jak zaprojektować dające się wielokrotnie używać przetwarzanie potokowe w przypadku wyzwania dla klasyfikatorów. Jak wspominałem, przygotujemy dane raz i będziemy używać ich dla wszystkich klasyfikatorów.

Importowanie danych

Dane historyczne dla tego zadania przechowywane są w pliku o nazwie dataset w formacie CSV. Skorzystamy z funkcji `pd.read_csv` z biblioteki *pandas*, by zaimportować dane jako ramkę danych.

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

Wybór cech

Proces wybierania cech, które mają znaczenie w kontekście rozwiązywanego przez nas problemu, nazywamy wyborem cech. Jest to kluczowa część inżynierii cech.

Gdy uda się nam zaimportować plik, usuwamy z niego kolumnę `User ID`, która wskazuje konkretnych użytkowników i nie powinna służyć do trenowania modelu:

```
dataset = dataset.drop(columns=['User ID'])
```

Spójrzmy teraz na zbiór danych:

```
dataset.head(5)
```

Wygląda on następująco:

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

Zastanówmy się, jak możemy dalej przetworzyć dane wejściowe.

Kod 1 z n

Wiele algorytmów uczenia maszynowego wymaga, by wszystkie cechy były wartościami ciągłymi. Oznacza to, że jeśli niektóre z wartości cech to zmienne jakościowe, musimy znaleźć sposób, by zamienić je na zmienne ciągłe. Kodowanie 1 z *n* to jeden z najbardziej wydajnych sposobów dokonania tej transformacji. W tym konkretnym zadaniu jedyną zmienną jakościową jest `Gender` (płeć). Zamieńmy ją na zmienną ciągłą za pomocą kodowania 1 z *n*:

```
enc = sklearn.preprocessing.OneHotEncoder()
enc.fit(dataset.iloc[:,[0]])
onehotlabels = enc.transform(dataset.iloc[:,[0]]).toarray()
genders = pd.DataFrame({'Female': onehotlabels[:, 0], 'Male': onehotlabels[:, 1]})
result = pd.concat([genders,dataset.iloc[:,1:]], axis=1, sort=False)
result.head(5)
```

Po dokonaniu konwersji ponownie spójrzmy na zbiór danych:

	Female	Male	Age	EstimatedSalary	Purchased
0	0.0	1.0	19	19000	0
1	0.0	1.0	35	20000	0
2	1.0	0.0	26	43000	0
3	1.0	0.0	27	57000	0
4	0.0	1.0	19	76000	0

Zauważ, że aby doprowadzić do konwersji zmiennej ze zmiennej jakościowej na ciągłą, kodowanie 1 z *n* podzieliło kolumnę `Gender` na dwie — `Male` (mężczyzna) i `Female` (kobieta).

Wskazanie cech i etykiet

Określmy teraz cechy i etykiety. W tej książce będę oznaczał etykietę zmienną `y`, a zmienną `X` — zbiór cech:

```
y=result['Purchased']
X=result.drop(columns=['Purchased'])
```

`X` reprezentuje wektor cech i zawiera wszystkie zmienne wejściowe, których potrzebujemy do wytrenowania modelu.

Podzielenie danych na zbiór testowy i treningowy

Powinniśmy teraz podzielić dostępne dane na zbiór testowy i treningowy. Za pomocą funkcji `train_test_split` z modułu `sklearn.model_selection` przeznaczymy 25% danych na testy i 75% na trenowanie modelu:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
↪random_state = 0)
```

W ten sposób powstały następujące cztery struktury danych:

- `X_train`. Struktura danych przechowująca cechy danych treningowych.
- `X_test`. Struktura danych przechowująca cechy danych testowych.
- `y_train`. Wektor przechowujący wartości etykiet w danych treningowych.
- `y_test`. Wektor przechowujący wartości etykiet w danych testowych.

Skalowanie cech

W przypadkach wielu algorytmów uczenia maszynowego dobrą praktyką jest skalowanie zmiennych do przedziału 0 – 1. Nazywamy to też **normalizacją cech**. Skorzystajmy w tym celu z transformacji skalującej:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Po znormalizowaniu danych są one gotowe do użycia jako dane wejściowe różnych algorytmów, które przedstawię w kolejnych sekcjach.

Ocena klasyfikatorów

Gdy wytrenujemy już model, musimy ocenić jego efektywność. W tym celu posłużymy się następującym procesem:

1. Podzielimy dane na dwie części — zbiór treningowy i zbiór testowy. Zbioru testowego użyjemy do oceny wytrenowanego modelu.

2. Użyjemy cech w zbiorze testowym do wygenerowania etykiet w każdym wierszu. Będzie to stanowiło zbiór przewidywanych etykiet.
3. Porównamy zbiór przewidywanych etykiet z rzeczywistymi etykietami w celu oceny modelu.

O ile nie próbujemy rozwiązać trywialnego problemu, zawsze pojawią się jakieś błędy klasyfikacji w ocenie modelu. To, jak zinterpretujemy te błędy klasyfikacji w celu określenia jakości modelu, zależy od wskaźników efektywności, jakie wybierzemy.

Gdy mamy już przewidywane i rzeczywiste etykiety, zbiór wskaźników efektywności pozwoli nam ocenić modele. Najwłaściwszy wskaźnik, na którym powinniśmy opierać ocenę modelu, będzie zależał od wymagań problemu biznesowego, jaki staramy się rozwiązać, oraz charakterystyki zbioru treningowego.

Tablica pomyłek

Z tablicy pomyłek korzystamy w celu podsumowania wyników oceny klasyfikatora. Tablica pomyłek dla klasyfikatora binarnego będzie wyglądała następująco:

		Klasa przewidywana
		Prawdziwie pozytywna (TP)	Fałszywie negatywna (FN)
Klasa rzeczywista		Fałszywie pozytywna (FP)	Prawdziwie negatywna (TN)

Jeśli etykieta klasyfikatora, jaki trenujemy, ma tylko dwie kategorie, nazywamy ją klasyfikacją binarną. Pierwszym, niezwykle ważnym zastosowaniem nadzorowanego uczenia maszynowego jeszcze z czasów pierwszej wojny światowej — dokładniej: zastosowaniem klasyfikacji binarnej — było rozróżnianie pomiędzy samolotami a ptakami.

Klasyfikację można podzielić na cztery kategorie:

- ***True Positives*** **(TP)** — pozytywna klasyfikacja, która została dokonana słusznie.
- ***True Negatives*** **(TN)** — negatywna klasyfikacja, która została dokonana słusznie.
- ***False Positives*** **(FP)** — pozytywna klasyfikacja, która została dokonana niesłusznie.
- ***False Negatives*** **(FN)** — negatywna klasyfikacja, która została dokonana niesłusznie.

Sprawdźmy, jak możemy skorzystać z tych czterech kategorii, by stworzyć różne wskaźniki efektywności.

Wskaźniki efektywności

Wskaźniki efektywności wykorzystuje się do liczbowego ujęcia efektywności wytrenowanego modelu. Zdefiniujmy na tej podstawie poniższe cztery wskaźniki:

Wskaźnik	Wzór
Dokładność	$\frac{TP + TN}{TP + TN + FP + FN}$
Czułość	$\frac{TP}{TP + TN} = \frac{\text{poprawnie zaklasyfikowane}}{\text{poprawnie zaklasyfikowane} + \text{niezaklasyfikowane}}$
Precyzja	$\frac{TP}{TP + FP} = \frac{\text{poprawnie zaklasyfikowane}}{\text{poprawnie zaklasyfikowane} + \text{niepoprawnie zaklasyfikowane}}$
Wskaźnik F1	$2\left(\frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}}\right)$

Dokładność to proporcja poprawnych klasyfikacji pośród wszystkich przewidywań. Podczas jej wyliczania nie rozróżniamy wyników prawdziwie pozytywnych i prawdziwie negatywnych. Ocena modelu na podstawie jego dokładności jest prosta, ale nie zawsze się sprawdza.

Zastanówmy się nad sytuacjami, w których potrzebujemy więcej niż dokładności do ujęcia efektywności modelu. Jedną z nich jest ta, w której korzystamy z modelu do przewidzenia wystąpienia rzadkiego zjawiska, np.:

- modelu do przewidywania oszustw w bazie danych transakcji banku,
- modelu do przewidywania prawdopodobieństwa mechanicznej usterki części silnika w samolocie.

W obu tych przypadkach staramy się przewidzieć wydarzenie nieczęsto zachodzące. Dwa inne wskaźniki stają się wtedy ważniejsze od dokładności — to czułość i precyzja. Przyjrzyjmy się im.

- **Czułość.** Obliczamy tu liczbę trafień. W kontekście pierwszego przykładu interesuje nas proporcja oszustw skutecznie wskazanych przez model spośród wszystkich oszustw. Załóżmy, że w zbiorze testowym był milion transakcji, 100 z nich było oszustwami, a model wykrył 78 z nich. W tej sytuacji czułość modelu wynosiłaby 78/100.
- **Precyzja.** Tym razem interesuje nas to, ile transakcji wskazanych przez model, rzeczywiście było oszustwami. Zamiast skupiać się na oszustwach, których model nie wykrył, chcemy wiedzieć, jak precyzyjnie model wskazuje oszustwa.

Zwróć uwagę, że wskaźnik *F1* łączy czułość i precyzję. Jeśli model uzyskał idealne noty pod względem precyzji i czułości, również jego wskaźnik *F1* będzie idealny. Wysoki wskaźnik *F1* oznacza, że wytrenowaliśmy model o wysokiej jakości, który cechuje się wysoką czułością i precyzją.

Nadmierne dopasowanie

Jeśli model uczenia maszynowego sprawdza się świetnie w środowisku deweloperskim, ale jego efektywność znacznie maleje w środowisku produkcyjnym, możemy sądzić, że jego wadą jest nadmierne dopasowanie. Oznacza to, że wytrenowany model zbyt ściśle trzyma się zbioru treningowego. Stanowi to wskazówkę, że w zasadach stworzonych przez model jest zbyt wiele szczegółów. Najlepiej problem ten ujmuje kompromis pomiędzy obciążeniem a wariancją. Zdefiniujmy sobie każde z tych pojęć.

Obciążenie

Każdy model uczenia maszynowego jest wytrenowany w oparciu o pewne założenia. Ogólnie założenia te są przybliżeniami pewnych rzeczywistych zjawisk. Upraszczają one rzeczywiste relacje między cechami a ich charakterystyką i sprawiają, że model łatwiej wytrenować. Więcej założeń oznacza jednak większe obciążenie. Zatem podczas trenowania modelu więcej uproszczonych założeń = większe obciążenie, a więcej realistycznych założeń lepiej oddających rzeczywiste zjawiska = mniejsze obciążenie.

> W regresji liniowej nieliniowość cech jest zupełnie pomijana, a zmienne przybliżane są jako zmienne liniowe. Powoduje to, że modele regresji liniowej są z definicji podatne na wysokie obciążenie.

Wariancja

Wariancja wskazuje, jak dokładnie model szacuje zmienną celu, jeśli do jego wytrenowania użyje się innego zbioru danych. Określa on, jak dobrą generalizacją istniejącego schematu jest dany wzór matematyczny.

Szczegółowe, nadmiernie dopasowane reguły oparte na szczegółowych scenariuszach i sytuacjach = wysoka wariancja. Reguły uogólnione i dające się zastosować do różnych scenariuszy i sytuacji = niska wariancja.

> Naszym celem w uczeniu maszynowym jest wytrenowanie takiego modelu, który wykazuje niskie obciążenie i niską wariancję. Osiągnięcie tego celu nie zawsze jest łatwe i spędza sen z powiek specjalistom data science.

Kompromis między obciążeniem a wariancją

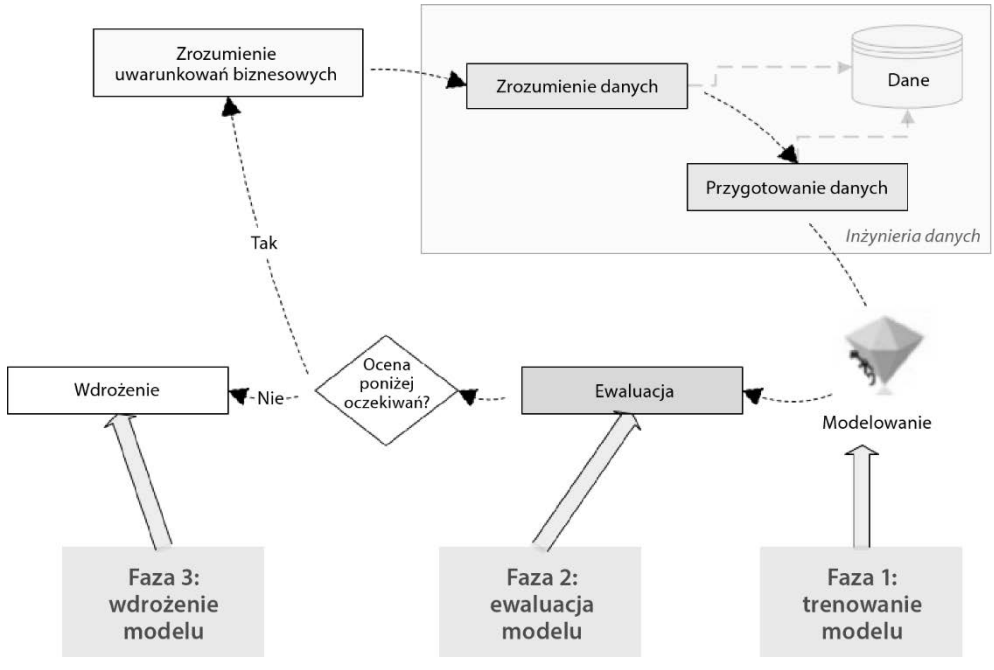
Gdy trenujemy konkretny model uczenia maszynowego, trudno nam zadecydować, jaki poziom generalizacji reguł będzie właściwy do stworzenia ram modelu. Starania o wskazanie tego poziomu określa się jako kompromis między obciążeniem a wariancją.

> Zwróć uwagę, że więcej uproszczonych założeń = więcej generalizacji = niska wariancja.

Granice tego kompromisu będą wyznaczane przez wybrany algorytm, charakterystykę danych i liczne hiperparametry. Osiągnięcie kompromisu między obciążeniem a wariancją w oparciu o wymagania konkretnego problemu, jaki rozwiązujesz, jest niezwykle ważne.

Określenie faz klasyfikacji

Gdy dane etykietowane są już gotowe, tworzenie klasyfikatorów obejmuje trenowanie, ewaluację i wdrożenie. Te trzy fazy implementacji klasyfikatorów przedstawia proces CRISP-DM (*Cross-Industry Standard Process for Data Mining*), przedstawiony na poniższym diagramie (cykl życia CRISP-DM był szczegółowo omawiany w rozdziale 6., „Algorytmy nienadzorowanego uczenia maszynowego").



W pierwszych dwóch fazach implementacji klasyfikatorów — fazie testowania i trenowania — korzystamy z danych etykietowanych. Dane etykietowane są dzielone na dwie części: większą — zbiór treningowy, i mniejszą — zbiór testowy. W tym celu używana jest technika losowego próbkowania, by upewnić się, że w obu częściach występują spójne schematy. Zwróć uwagę, że jak widać na powyższym diagramie, najpierw następuje faza trenowania, w której model uczy się na podstawie danych treningowych. Gdy etap ten się zakończy, gotowy model podlega ocenie za pomocą danych testowych. Do wyliczenia jego efektywności używane są różne wskaźniki. Gdy model zbierze wysokie noty, wkraczamy w fazę wdrożenia, gdzie model jest dostarczany na środowisko produkcyjne i używany do rozwiązywania rzeczywistych problemów poprzez przewidywanie etykiet danych nieetykietowanych.

Przejdźmy teraz do kilku algorytmów klasyfikujących.

W kolejnych sekcjach zajmiemy się:

- drzewem decyzyjnym,
- algorytmem wzmacniania gradientowego,
- lasem losowym,
- algorytmem regresji logistycznej,
- maszyną wektorów nośnych,
- naiwnym klasyfikatorem bayesowskim.

Zaczniemy od algorytmu drzewa decyzyjnego.

Algorytm drzewa decyzyjnego

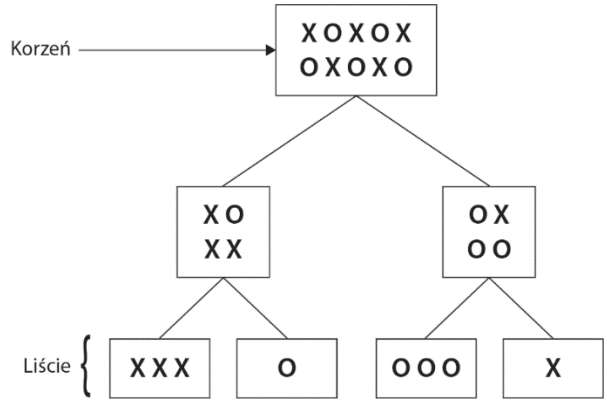
Drzewo decyzyjne opiera się na podejściu rekurencyjnego podziału („dziel i rządź"), generującego zestaw reguł, na podstawie których można przewidzieć etykietę. Drzewo takie zaczyna się od korzenia i dzieli się dalej na wiele gałęzi. W każdym węźle wewnętrznym wykonywany jest test dotyczący wartości pewnego atrybutu, a z węzła tego odchodzi na kolejny poziom tyle gałęzi, ile jest możliwych wyników takiego testu. Drzewo decyzyjne kończy się liśćmi, które zawierają decyzje. Proces kończy się, gdy dalszy podział nie poprawia już rezultatu.

Algorytm klasyfikujący drzewa decyzyjnego

Cechą wyróżniającą klasyfikację za pomocą drzewa decyzyjnego jest fakt, że w wyniku jego działania powstaje zrozumiała dla człowieka hierarchia reguł wykorzystanych w czasie uruchomienia do przewidzenia etykiety. Algorytm ten ma rekurencyjną naturę. Stworzenie wspomnianej hierarchii reguł obejmuje:

1. **Znalezienie najważniejszej cechy.** Spośród wszystkich cech algorytm wybiera tę, która pozwala najdokładniej odróżnić od siebie próbki danych pod kątem etykiety w zbiorze danych. Obliczenia te opierają się na wskaźnikach takich jak współczynnik wzmocnienia informacji czy miara nieczystości Giniego.
2. **Podział.** Na podstawie wykrytej najważniejszej cechy algorytm tworzy kryterium podziału zbioru treningowego na dwie gałęzie:
   - próbki danych spełniające kryterium,
   - próbki danych niespełniające kryterium.
3. **Poszukiwanie liści.** Jeśli powstała w ten sposób gałąź zawiera głównie etykiety jednej klasy, kończy się ona, tworząc liść.
4. **Sprawdzenie warunku wyjścia i powtórzenie.** Jeśli nie jest spełniony warunek wyjścia, algorytm wróci do pierwszego kroku i wykona kolejną iterację. W przeciwnym wypadku model uznaje się za wytrenowany, a każdy węzeł na najniższym poziomie w wynikowym drzewie decyzyjnym uznawany jest za liść. Warunek wyjścia może być choćby tak prosty jak zadeklarowana liczba iteracji, ale można też użyć warunku domyślnego — algorytm kończy działanie, gdy osiąga pewien poziom homogeniczności dla każdego liścia.

Algorytm drzewa decyzyjnego można wyjaśnić także za pomocą poniższego diagramu:



Na powyższym diagramie korzeń zawiera kilka kółek i krzyżyków. Algorytm tworzy kryterium, na podstawie którego próbuje oddzielić kółka od krzyżyków. Na każdym kroku drzewo decyzyjne dokonuje podziału danych, które mają stawać się coraz bardziej jednolite od poziomu 1 w górę. Idealny klasyfikator miałby liście, które zawierałyby tylko kółka lub tylko krzyżyki. Wytrenowanie idealnego klasyfikatora jest jednak trudne ze względu na nieuniknioną losowość zestawu treningowego.

Wykorzystanie algorytmu klasyfikacji drzewa decyzyjnego do wyzwania dla klasyfikatorów

Skorzystajmy teraz z algorytmu klasyfikacji drzewa decyzyjnego do rozwiązania problemu, jaki wcześniej zdefiniowaliśmy, tj. przewidzenia, czy klient zakupi produkt:

1. Po pierwsze, musimy stworzyć instancję algorytmu drzewa decyzyjnego i wytrenować model na podstawie zbioru treningowego, który przygotowaliśmy wcześniej dla wszystkich naszych klasyfikatorów:

```
classifier = sklearn.tree.DecisionTreeClassifier(criterion = 'entropy',
↪random_state = 100, max_depth=2)
classifier.fit(X_train, y_train)
```

2. Teraz możemy użyć wytrenowanego modelu do przewidzenia etykiety testowej części naszych etykietowanych danych. Stwórzmy też tablicę pomyłek, która podsumuje efektywność naszego modelu:

```
import sklearn.metrics as metrics
y_pred = classifier.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

Po wyświetleniu wartości `cm` na ekranie zobaczymy:

```
Out[22]: array([[64,  4],
                [ 2, 30]])
```

3. Powinniśmy obliczyć wartość wskaźników jakości algorytmu, tj. *accuracy* (dokładność), *recall* (czułość) i *precision* (precyzja) dla stworzonego klasyfikatora:

```
accuracy = metrics.accuracy_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
precision = metrics.precision_score(y_test,y_pred)
print(accuracy,recall,precision)
```

4. Uruchomienie powyższego kodu da nam wynik w postaci:

```
0.94 0.9375 0.8823529411764706
```

Miary efektywności pozwolą nam porównać różne techniki trenowania modeli.

Wady i zalety klasyfikatorów opartych na drzewach decyzyjnych

W tej sekcji przeanalizuję mocne i słabe strony korzystania z algorytmów klasyfikujących opartych na drzewach decyzyjnych.

Zalety

Drzewa decyzyjne mają następujące zalety:

- Reguły modelu stworzone podczas wykonania algorytmu drzewa decyzyjnego są zrozumiałe dla ludzi. Tego rodzaju modele nazywamy białoskrzynkowymi. Ich użycie jest nieodzowne, gdy wymagana jest przejrzystość i możliwość prześledzenia szczegółów podejmowania decyzji w modelu. Taka transparentność jest kluczowa w aplikacjach walczących ze stronniczością czy chroniących zagrożone społeczności. Przykładowo w sektorze publicznym i ubezpieczeniowym stosowanie modeli białoskrzynkowych jest najczęściej niezbędnym wymaganiem.
- Klasyfikatory oparte na drzewach decyzyjnych zostały zaprojektowane do pozyskiwania informacji z przestrzeni dyskretnej, co oznacza, że większość cech to zmienne jakościowe. Gdy rzeczywiście mamy do czynienia ze zbiorem danych o takiej charakterystyce, tego rodzaju model będzie dobrym wyborem.

Wady

Drzewa decyzyjne mają następujące wady:

- Jeśli drzewo stworzone przez tego typu klasyfikator za bardzo się rozrośnie, jego reguły obejmują zbyt wiele szczegółów, skutkując nadmiernym dopasowaniem modelu. Gdy korzystamy z algorytmów opartych na drzewach decyzyjnych, musimy zdawać sobie sprawę z tej podatności i w miarę potrzeb ograniczać drzewo, by uniknąć jej negatywnych konsekwencji.
- Niemożliwość uchwycenia nieliniowych relacji w regułach, jakie generują tego rodzaju modele.

Przypadki użycia

W tej sekcji podam kilka przypadków użycia algorytmów drzew decyzyjnych.

Klasyfikacja rekordów

Klasyfikatory oparte na drzewach decyzyjnych mogą być użyte do kategoryzowania próbek danych, tak jak:

- **We wnioskach kredytowych.** Trenujemy klasyfikator binarny mający określić, czy kredytobiorca będzie terminowo spłacał swoje zobowiązania.
- **W segmentacji klientów.** Dzielimy klientów na klientów premium, klientów klasy średniej i klientów poniżej średniej, by dopasować strategie marketingowe do każdej grupy.
- **W diagnozach lekarskich.** Trenujemy klasyfikator oznaczający łagodne i złośliwe guzy.
- **W analizie skuteczności leczenia.** Trenujemy klasyfikator wskazujący pacjentów, którzy dobrze zareagowali na daną kurację.

Wybór cech

Algorytm klasyfikujący drzewa decyzyjnego wybiera mały podzbiór cech, na podstawie którego tworzy reguły. Wynik tego wyboru może posłużyć Ci także do wykorzystania przy innym algorytmie uczenia maszynowego, jeśli w Twoim zbiorze danych jest wiele cech.

Metody zespolone

Metody zespolone w uczeniu maszynowym to praktyka tworzenia kilku niewiele różniących się od siebie modeli na podstawie różnych parametrów i łączenia ich w zagregowany model. Aby efektywnie tworzyć takie modele, musimy najpierw poznać kryterium agregacji. Przyjrzyjmy się więc algorytmom zespolonym.

Implementacja wzmacniania gradientowego

Algorytm **wzmacniania gradientowego** (*XGBoost*) został stworzony w 2014 roku w oparciu o zasady procesu, który dał mu nazwę. Stał się on jednym z najbardziej popularnych klasyfikatorów zespolonych. Generuje on wiele powiązanych ze sobą drzew i wykorzystuje metodę gradientu prostego do zminimalizowania błędu resztkowego. Sprawia to, że jest to doskonały wybór dla infrastruktury rozproszonej, takiej jak Apache Spark, czy chmur obliczeniowych jak Google Cloud lub Amazon Web Services (AWS).

Spójrzmy, jak zaimplementować algorytm wzmacniania gradientowego:

1. Po pierwsze, stworzymy instancję klasyfikatora `XGBClassifier` i wytrenujemy model przy użyciu zbioru treningowego:

```
In [20]: from xgboost import XGBClassifier
         classifier = XGBClassifier()
         classifier.fit(X_train, y_train)

Out[20]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, gamma=0,
                       learning_rate=0.1, max_delta_step=0, max_depth=3,
                       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                       nthread=None, objective='binary:logistic', random_state=0,
                       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                       silent=None, subsample=1, verbosity=1)
```

2. Następnie wygenerujemy przewidywania na podstawie nowo wytrenowanego modelu:

```
y_pred = classifier.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

Uzyskamy w ten sposób następujące dane wyjściowe:

```
Out[21]: array([[64,  4],
                [ 3, 29]])
```

3. Wreszcie, wyliczymy efektywność modelu:

```
accuracy= metrics.accuracy_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
precision = metrics.precision_score(y_test,y_pred)
print(accuracy,recall,precision)
```

Dane wyjściowe będą wyglądały tak:

```
0.93 0.90625 0.8787878787878788
```

Przejdźmy teraz do algorytmu lasu losowego.

Lasy losowe

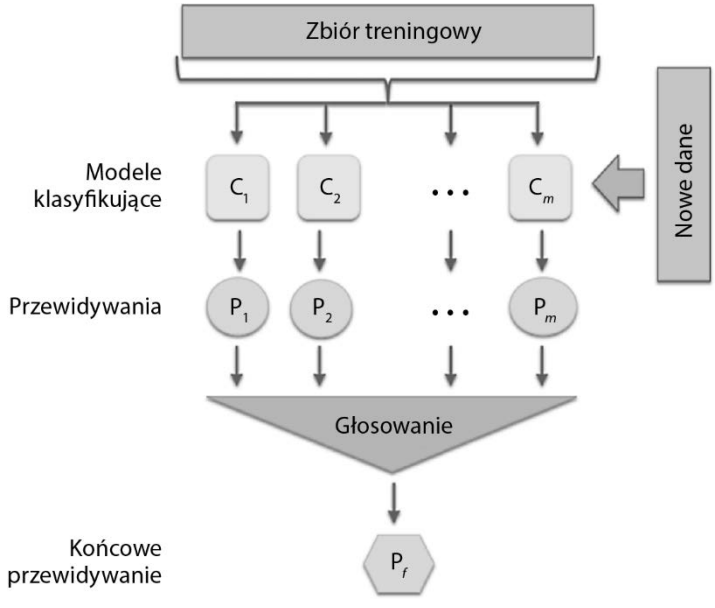
Las losowe to rodzaj metody zespolonej łączącej kilka drzew decyzyjnych w celu zmniejszenia zarówno obciążenia, jak i wariancji.

Wytrenowanie algorytmu lasu losowego

Podczas trenowania algorytm bierze N próbek ze zbioru treningowego i tworzy m podzbiorów ze wszystkich danych. Podzbiory te tworzy się przez losowy dobór wierszy i kolumn w danych wejściowych. Algorytm buduje m niezależnych drzew decyzyjnych. Te drzewa klasyfikacji oznaczane są jako C_1 do C_m.

Wykorzystanie lasu losowego do dokonywania przewidywań

Gdy model zostanie wytrenowany, można go użyć do etykietowania nowych danych. Każde pojedyncze drzewo wygeneruje etykietę. Końcowa etykieta jest wynikiem głosowania nad poszczególnymi przewidywaniami, co widać poniżej:



Zwróć uwagę, że na powyższym diagramie wytrenowano m drzew, które przedstawiono jako $C_1 - C_m$, tj. `Trees = {C1, ..., Cm}`.

Każde z tych drzew generuje predykcję, a wszystkie one są przechowywane w zbiorze:

Pojedyncze predykcje $= P = \{P_1, \ldots, P_m\}$

Ostateczne przewidywanie przedstawia się jako P_f. Określa się je na podstawie większości pojedynczych predykcji. Można skorzystać z funkcji `mode` w celu znalezienia wartości większości decyzji (funkcja ta da nam dominantę, czyli wartość powtarzającą się najczęściej, a więc stanowiącą większość). Pojedyncze przewidywania i przewidywania końcowe są ze sobą związane:

$P_f = mode(P)$

Różnica między lasem losowym a wzmocnieniem gradientowym

Każde z drzew wygenerowanych w wyniku działania algorytmu lasu losowego jest zupełnie niezależne od pozostałych. Nie zna ono żadnych szczegółów innych drzew. Cecha ta odróżnia las losowy od innych podobnych technik, np. wzmacniania gradientowego.

Wykorzystanie algorytmu lasu losowego do wyzwania dla klasyfikatorów

Stwórzmy instancję algorytmu lasu losowego i wykorzystajmy ją do wytrenowania modelu na podstawie danych treningowych.

Będziemy korzystali z dwóch kluczowych hiperparametrów:

- `n_estimators`,
- `max_depth`.

Hiperparametr `n_estimators` kontroluje liczbę pojedynczych drzew decyzyjnych stworzonych w wyniku działania algorytmu, a `max_depth` odpowiada za to, jak głębokie może być każde z nich.

Drzewo decyzyjne może dzielić się tak długo, aż będzie miało po jednym węźle dla każdej próbki danych w zbiorze treningowym. Wskazując wartość `max_depth`, ograniczamy liczbę podziałów. Odpowiada to za złożoność modelu oraz stopień dopasowania do danych treningowych. Jeśli spojrzymy na poniższe dane wyjściowe, wartość `n_estimators` reprezentuje szerokość lasu losowego, a `max_depth` — głębokość tego modelu:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, max_depth = 4,criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
Out[9]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                               max_depth=4, max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=10,
                               n_jobs=None, oob_score=False, random_state=0, verbose=0,
                               warm_start=False)
```

Po wytrenowaniu modelu możemy użyć go do przewidywań:

```
y_pred = classifier.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

Uzyskamy w ten sposób następujące dane wyjściowe:

```
Out[10]: array([[64,  4],
                [ 3, 29]])
```

Pora zmierzyć, jak dobry jest nasz model:

```
accuracy= metrics.accuracy_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
precision = metrics.precision_score(y_test,y_pred)
print(accuracy,recall,precision)
```

Tym razem dane wyjściowe będą takie:

```
0.93 0.90625 0.8787878787878788
```

Kolejnym algorytmem na naszej liście będzie regresja logistyczna.

Regresja logistyczna

Regresja logistyczna to algorytm klasyfikujący wykorzystywany do klasyfikacji binarnej. Posługuje się on funkcją logistyczną do opisania relacji między danymi wejściowymi a zmienną celu. Jest to jedna z najprostszych technik klasyfikacji, która jest stosowana do modelowania zależnej zmiennej binarnej.

Założenia

Regresja logistyczna zakłada, że:

- w zbiorze treningowym nie brakuje żadnych wartości,
- etykieta to binarna zmienna jakościowa,
- etykieta jest wartością porządkową — zmienną jakościową z uporządkowanymi wartościami,
- wszystkie cechy lub zmienne wejściowe są od siebie niezależne.

Określanie relacji

W przypadku regresji liniowej wartości przewidywane oblicza się za pomocą następującego wzoru:

$$\acute{y} = \sigma(\omega X + j)$$

Załóżmy, że $z = \omega X + j$.

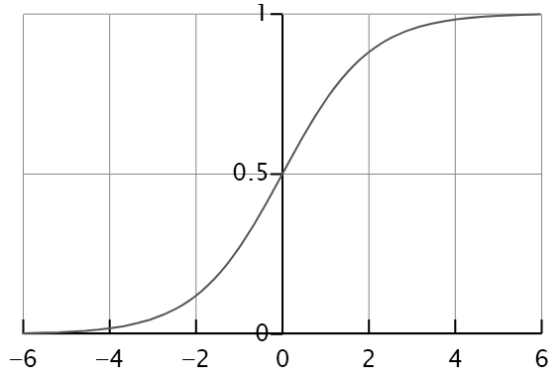
Zatem:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Powyższą relację można graficznie przedstawić w ten sposób (zobacz rysunek na następnej stronie).

Zauważ, że jeśli z jest duże, to $\sigma(z)$ będzie równe 1. Jeśli z jest małe lub ma dużą wartość ujemną, $\sigma(z)$ będzie równe 0. Zatem celem regresji logistycznej jest znalezienie poprawnych wartości w oraz j.

> Regresja logistyczna bierze swoją nazwę od funkcji, na której się opiera, tj. funkcji logistycznej.



Funkcje straty i kosztu

Funkcja straty definiuje, jak chcemy liczyć błędy w konkretnej próbce danych w zestawie treningowym. Funkcja kosztu określa z kolei, jak chcemy minimalizować liczbę błędów w całym zbiorze treningowym. Zatem funkcji straty używamy na próbce w zbiorze, a funkcji kosztu — w odniesieniu do całkowitego narzutu, który wskazuje łączne odchylenie wartości przewidzianych od wartości rzeczywistych. Zależy on od doboru wartości `w` i `j`.

Funkcji straty używa się w regresji logistycznej następująco:

$$Loss(\acute{y}^{(i)}, y^{(i)}) = -(y^{(i)} log\ \acute{y}^{(i)} + (1 - y^{(i)})\ log\ (1 - \acute{y}^{(i)})$$

Zwróć uwagę, że gdy $y^{(i)} = 1$, $Loss(\acute{y}^{(i)}, y^{(i)}) = -log\ \acute{y}^{(i)}$. Minimalizowanie wartości funkcji straty będzie skutkowało wysoką wartością $\acute{y}^{(i)}$. Ponieważ jest to funkcja logistyczna, maksymalną wartością będzie *1*.

Jeśli $y^{(i)} = 0$, $Loss(\acute{y}^{(i)}, y^{(i)}) = -log\ (1 - \acute{y}^{(i)})$. Minimalizowanie wartości funkcji straty będzie skutkowało najniższą możliwą wartością $\acute{y}^{(i)}$, czyli *0*.

Funkcja kosztu w regresji logistycznej wygląda następująco:

$$Koszt(w, b) = \frac{1}{b}\sum Strata\left(\acute{y}^{(i)}, y^{(i)}\right)$$

Kiedy używać regresji logistycznej

Regresja logistyczna sprawdza się znakomicie w przypadku klasyfikatorów binarnych. Nie będzie jednak działać dobrze, gdy zbiór danych jest bardzo duży, a jakość danych — mizerna. Algorytm ten jest w stanie wychwycić relacje, które nie są szczególnie złożone. Chociaż zwykle jego efektywność nie jest oszałamiająca, stanowi on dobry punkt wyjściowy.

Wykorzystanie algorytmu regresji logistycznej do wyzwania dla klasyfikatorów

W tej sekcji przekonamy się, jak wykorzystać regresję logistyczną do rozwiązania naszego problemu:

1. Zacznijmy od stworzenia instancji modelu regresji logistycznej i wytrenowania go na podstawie zbioru treningowego:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

2. Dokonajmy predykcji wartości danych testowych i stwórzmy tablicę pomyłek:

```
y_pred = classifier.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

   Da to poniższy rezultat:

```
Out[11]: array([[65,  3],
                [ 6, 26]])
```

3. Rzut oka na wskaźniki efektywności:

```
accuracy= metrics.accuracy_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
precision = metrics.precision_score(y_test,y_pred)
print(accuracy,recall,precision)
```

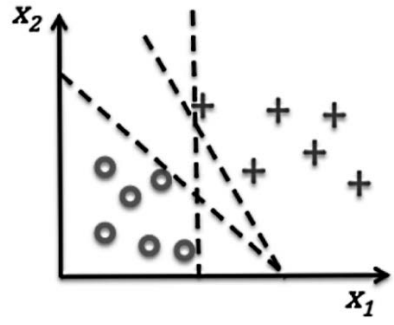
4. Po uruchomieniu powyższego kodu uzyskamy następujące dane wyjściowe:

```
0.91 0.8125 0.896551724137931
```

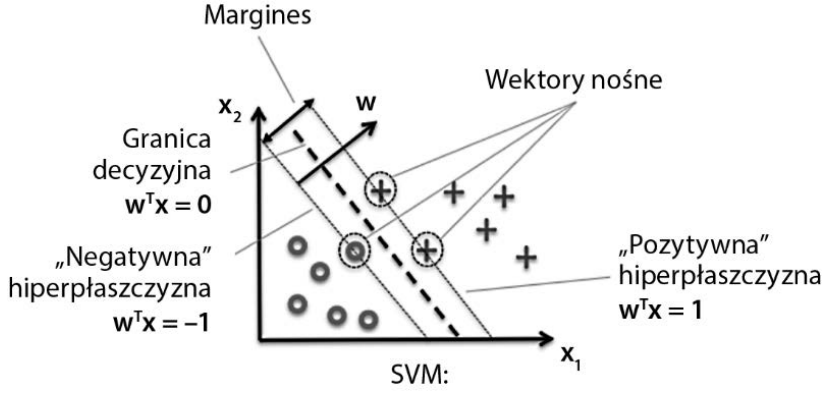
Kolejnym omawianym algorytmem będzie maszyna wektorów nośnych.

Maszyna wektorów nośnych

Poświęcę teraz trochę miejsca algorytmowi **maszyny wektorów nośnych** (*SVM*). Jest to klasyfikator, który stara się znaleźć optymalną hiperpłaszczyznę rozdzielającą dwie klasy z maksymalnym marginesem. Celem optymalizacji maszyny wektorów nośnych jest właśnie zwiększanie tego marginesu, który definiuje się jako odległość między rozdzielającą hiperpłaszczyzną (granicą decyzyjną) a próbkami treningowymi, znajdującymi się najbliżej niej (tzw. wektorami nośnymi). Zacznijmy od bardzo prostego przykładu o zaledwie dwóch wymiarach, `X1` i `X2`. Chcemy wyrysować linię oddzielającą kółka od krzyżyków. Widać to na poniższym diagramie:



Wyznaczyliśmy dwie linie. Obie z nich doskonale oddzielają kółka od krzyżyków. Musi jednak istnieć optymalna linia, czy też granica decyzyjna, która daje nam największe szanse na prawidłową klasyfikację większości dodatkowych próbek. Rozsądnym wyborem może być linia biegnąca równo pośrodku pomiędzy tymi klasami, dająca nieco zapasu każdej z klas, jak widać to poniżej:



Spójrzmy teraz, jak możemy użyć maszyny wektorów nośnych do wytrenowania klasyfikatora na nasz użytek.

Wykorzystanie maszyny wektorów nośnych do wyzwania dla klasyfikatorów:

1. Pierwszym krokiem będzie stworzenie instancji klasyfikatora `SVM` i skorzystanie ze zbioru treningowego etykietowanych danych, by go wytrenować. Hiperparametr `kernel` określa rodzaj transformacji, jaka jest stosowana na danych wejściowych w celu umożliwienia ich liniowego podziału:

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

2. Skoro mamy już wytrenowany model, wygenerujmy przewidywania i spójrzmy na tablicę pomyłek:

```
y_pred = classifier.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

3. Tak będzie wyglądał wynik na tym etapie:

```
Out[9]: array([[66,  2],
               [ 9, 23]])
```

4. Pora teraz na sprawdzenie wskaźników efektywności:

```
accuracy= metrics.accuracy_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
precision = metrics.precision_score(y_test,y_pred)
print(accuracy,recall,precision)
```

Po uruchomieniu powyższego kodu uzyskamy następujące dane wyjściowe:

```
0.89 0.71875 0.92
```

Naiwny klasyfikator bayesowski

Oparty na teorii prawdopodobieństwa, naiwny klasyfikator bayesowski to jeden z najprostszych algorytmów klasyfikacji. Jeśli właściwie się go używa, może dawać dokładne przewidywania. Nazwa tego klasyfikatora ma swoje źródło w jego dwóch charakterystycznych cechach:

- Opiera się on na naiwnym założeniu, że pomiędzy cechami a zmiennymi wejściowymi nie ma zależności.
- Opiera się on na twierdzeniu Bayesa.

Algorytm stara się zaklasyfikować próbki na podstawie oceny prawdopodobieństwa poprzedniego atrybutu czy klasy, zakładając przy tym całkowitą niezależność zmiennych niezależnych.

Mamy tu do czynienia z trzema typami zdarzeń:

- **Niezależnymi**. Tego typu zdarzenia nie wpływają na prawdopodobieństwo wystąpienia innego zdarzenia (np. otrzymanie darmowej wejściówki na konferencję technologiczną i restrukturyzacja w Twojej firmie).
- **Zależnymi**. Tego typu zdarzenia mają wpływ na prawdopodobieństwo wystąpienia innego zdarzenia, ponieważ w jakiś sposób się one łączą (np. prawdopodobieństwo przybycia na czas na konferencję może być determinowane przez strajk kierowców komunikacji miejskiej czy opóźnione kursy autobusów).
- **Wzajemnie wykluczającymi się**. Tego typu wydarzenia nie mogą zajść w tym samym czasie (np. prawdopodobieństwo wyrzucenia zarówno trójki, jak i szóstki na jednej kości wynosi 0 — te dwa rezultaty wzajemnie się wykluczają).

Twierdzenie Bayesa

Twierdzenie Bayesa jest używane do wyliczenia warunkowego prawdopodobieństwa między dwoma niezależnymi zdarzeniami, `A` i `B`. Prawdopodobieństwo wystąpienia każdego z nich oznacza się odpowiednio jako `P(A)` i `P(B)`. Prawdopodobieństwo warunkowe natomiast określa się jako `P(B|A)`, co przedstawia warunkowe prawdopodobieństwo, że nastąpi zdarzenie `B`, jeśli zajdzie zdarzenie `A`:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Wyliczanie prawdopodobieństwa

Naiwny klasyfikator bayesowski opiera się na podstawach teorii prawdopodobieństwa. Prawdopodobieństwo, że nastąpi pojedyncze zdarzenie, wylicza się, biorąc pod uwagę liczbę wystąpień zdarzenia i dzieląc ją przez łączną liczbę procesów, które mogły do niego doprowadzić. Przykładowo centrum obsługi klientów obsługuje ponad 100 telefonów dziennie. Chcesz się dowiedzieć, jakie jest prawdopodobieństwo, że telefon zostanie odebrany w czasie krótszym niż 3 minuty, w oparciu o dane dotyczące poprzednich połączeń. Jeśli w przypadku 50 wcześniejszych połączeń udało się to 27 razy, prawdopodobieństwo odebrania telefonu w ciągu mniej niż 3 minut wynosi:

P(obsłużenie telefonu w czasie poniżej 3 minut) = 27/50 = 0,54 (54%)

100 telefonów uda się odebrać w ciągu mniej niż 3 minut w przeszło połowie przypadków, wnioskując na podstawie 50 poprzednich przypadków.

Reguły mnożenia dla koniunkcji zdarzeń

Aby wyliczyć prawdopodobieństwo wystąpienia dwóch lub więcej zdarzeń jednocześnie, trzeba ustalić, czy są to zdarzenia zależne, czy niezależne. Jeśli są one niezależne, stosuje się zwykłą regułę mnożenia:

P(zdarzenie1 i zdarzenie 2) = P(zdarzenie1) · P(zdarzenie2)

Przykładowo żeby policzyć prawdopodobieństwo otrzymania darmowej wejściówki na konferencję technologiczną i restrukturyzacji w Twoim miejscu pracy, użylibyśmy prostej zasady mnożenia. Zdarzenia te są od siebie niezależne, jedno nie wpływa na szanse wystąpienia drugiego.

Jeśli prawdopodobieństwo otrzymania darmowej wejściówki na konferencję technologiczną wynosi 31%, a prawdopodobieństwo restrukturyzacji w Twoim miejscu pracy wynosi 82%, to prawdopodobieństwo wystąpienia ich obu razem obliczymy według poniższego wzoru:

P(wejściówka i restrukturyzacja) = P(wejściówka) · P(restrukturyzacja) = 0,31 · 0,82 = 0,2542 (25%)

Ogólne zasady mnożenia

Jeśli dwa zdarzenia (lub więcej) są zależne, stosuje się ogólną zasadę mnożenia. Wzór ten ma zastosowanie do obu przypadków — zdarzeń zależnych i niezależnych:

$$P(zdarzenie1 \text{ i } zdarzenie2) = P(zdarzenie1) \cdot P(zdarzenie2 \mid zdarzenie1)$$

Pamiętaj, że $P(zdarzenie2 \mid zdarzenie1)$ odnosi się do warunkowego prawdopodobieństwa *zdarzenia2*, jeśli zajdzie *zdarzenie1*. Wzór ten bierze pod uwagę zależność między zdarzeniami. Jeśli zdarzenia są niezależne, prawdopodobieństwo warunkowe nie ma znaczenia, ponieważ jedno zdarzenie nie wpływa na szanse zajścia innego, a $P(zdarzenie1) \cdot P(zdarzenie2 \mid zdarzenie1)$ jest po prostu równe $P(zdarzenie2)$. W takiej sytuacji wzór ten staje się prostą zasadą mnożenia.

Zasady dodawania dla alternatywy zdarzeń

Aby obliczyć prawdopodobieństwo wystąpienia jednego lub drugiego zdarzenia (spośród zdarzeń wzajemnie wykluczających się), stosuje się poniższą prostą zasadę dodawania:

$$P(zdarzenie1 \text{ lub } zdarzenie2) = P(zdarzenie1) + P(zdarzenie2)$$

Jakie jest na przykład prawdopodobieństwo wyrzucenia szóstki lub trójki kością? Aby odpowiedzieć na to pytanie, należy pamiętać, że oba te rezultaty nie mogą zaistnieć jednocześnie. Prawdopodobieństwo wyrzucenia szóstki wynosi 1/6 i takie samo jest prawdopodobieństwo wyrzucenia trójki:

$$P(6 \text{ lub } 3) = 1/6 + 1/6 = 0{,}33\ (33\%)$$

Jeśli zdarzenia wzajemnie się nie wykluczają i mogą zajść równocześnie, będziesz musiał skorzystać z poniższego wzoru, którego można używać uniwersalnie, również w przypadku zdarzeń wzajemnie się wykluczających:

$$P(zdarzenie1 \text{ lub } zdarzenie2) = P(zdarzenie1) + P(zdarzenie2) - P(zdarzenie\ 1 \text{ i } zdarzenie2)$$

Wykorzystanie naiwnego klasyfikatora bayesowskiego do wyzwania dla klasyfikatorów

Pora wykorzystać ten klasyfikator w naszym wyzwaniu:

1. Zaczniemy od zaimportowania funkcji `GaussianNB()` i skorzystania jej do trenowania modelu:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

2. Wykorzystamy wytrenowany model do przewidzenia rezultatów. Będziemy więc przewidywać etykiety dla zbioru testowego `X_test`:

```
# przewidywanie etykiet w zbiorze testowym
y_pred = classifier.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

3. Przygotujmy też tablicę pomyłek:

```
Out[10]: array([[66,  2],
                [ 6, 26]])
```

4. Spójrzmy także na wskaźniki efektywności, by poznać jakość wytrenowanego modelu:

```
accuracy= metrics.accuracy_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
precision = metrics.precision_score(y_test,y_pred)
print(accuracy,recall,precision)
```

Uzyskamy w ten sposób poniższy rezultat:

```
0.92 0.8125 0.9285714285714286
```

Zwycięzcą wśród algorytmów klasyfikacji jest…

Raz jeszcze przeanalizujmy wskaźniki efektywności poszczególnych omówionych tu algorytmów. Podsumowuje je tabela poniżej:

Algorytm	Dokładność	Czułość	Precyzja
Drzewo decyzyjne	0,94	0,93	0,88
Wzmacnianie gradientowe	0,93	0,90	0,87
Las losowy	0,93	0,90	0,87
Regresja logistyczna	0,91	0,81	0,89
Maszyna wektorów nośnych	0,89	0,71	0,92
Naiwny klasyfikator bayesowski	0,92	0,81	0,92

Patrząc na powyższe zestawienie, możemy wywnioskować, że klasyfikator drzewa decyzyjnego wypada najlepiej pod względem dokładności i czułości. Jeśli zależy nam na precyzji, równą uwagę powinniśmy poświęcić maszynie wektorów nośnych i naiwnemu klasyfikatorowi bayesowskiemu — każdy z nich się pod tym względem sprawdzi.

Algorytmy regresji

Modele nadzorowanego uczenia maszynowego wykorzystują któryś z algorytmów regresji, jeśli zmienna celu jest zmienną ciągłą. W takim przypadku model uczenia maszynowego będzie regresorem.

W tej sekcji zaprezentuję różne algorytmy, które mogą zostać wykorzystane do trenowania regresyjnego modelu uczenia maszynowego. Zanim jednak skupię się na ich szczegółach, chcę zdefiniować postawione przed nimi wyzwanie, które posłuży nam za test ich efektywności, zdolności i skuteczności.

Wyzwanie dla regresji

W przypadku regresji wybrałem to samo podejście, które prezentowałem przy klasyfikatorach — zacznę od przedstawienia problemu, jaki będę się starał rozwiązać przy użyciu kilku algorytmów. Ten wspólny dla wszystkich problem nazwę tym razem wyzwaniem dla regresji. Następnie skorzystam z trzech różnych algorytmów w celu rozwiązania zadania. Użycie trzech algorytmów do rozwiązania jednego problemu będzie dla nas ważne z dwóch względów:

- Możemy przygotować dane raz i wykorzystać gotowe zbiory we wszystkich trzech algorytmach regresji.
- Będziemy w stanie porównać efektywność różnych algorytmów, ponieważ używamy ich do rozwiązania dokładnie tego samego problemu.

Zdefiniujmy jednak stojące przed nami zadanie.

Definicja problemu

Przewidywanie odległości, jaką auto jest w stanie przejechać na jednym baku, to ważna kwestia. Wydajne auto służy środowisku i nie rujnuje portfela właściciela. Taki przebieg na jednym tankowaniu można oszacować na podstawie mocy silnika i charakterystyki auta. Postawmy sobie zatem za cel wytrenowanie modelu, który będzie w stanie przewidzieć liczbę przejechanych przez samochód mil na jednym galonie paliwa w oparciu o cechy auta.

Przeanalizujmy historyczne dane, które posłużą do wytrenowania regresorów.

Dane historyczne

Spójrzmy na poniższe cechy zbioru danych historycznych:

Nazwa	Rodzaj	Opis
`NAME`	Jakościowa	Identyfikuje konkretne auto
`CYLINDERS`	Ciągła	Liczba cylindrów (między 4 a 8)
`DISPLACEMENT`	Ciągła	Objętość skokowa w calach sześciennych

Nazwa	Rodzaj	Opis
HORSEPOWER	Ciągła	Liczba koni mechanicznych
ACCELERATION	Ciągła	Czas potrzebny na przyspieszenie od 0 do 100 km (w sekundach)

Zmienną celu w tym zadaniu jest zmienna ciągła, mpg, która wskazuje, ile mil dane auto pokona na galonie paliwa.

Zacznijmy od zaprojektowania strumieniowego procesu przetwarzania danych dla tego problemu.

Inżynieria cech w strumieniowym przetwarzaniu danych

Zastanówmy się, jak zaprojektować dające się wielokrotnie stosować przetwarzanie strumieniowe dla wyzwania regresyjnego. Jak wspominałem, przygotujemy dane raz i będziemy używać tych samych zbiorów we wszystkich algorytmach. Powinniśmy wykonać następujące kroki:

1. Powinniśmy zacząć od zaimportowania zbioru danych:

```
dataset = pd.read_csv('auto.csv')
```

2. Zerknijmy, co się w nim kryje:

```
dataset.head(5)
```

   Oto, jak wygląda nasz zbiór danych:

	NAME	CYLINDERS	DISPLACEMENT	HORSEPOWER	WEIGHT	ACCELERATION	MPG
0	chevrolet chevelle malibu	8	307.0	130	3504	12.0	18.0
1	buick skylark 320	8	350.0	165	3693	11.5	15.0
2	plymouth satellite	8	318.0	150	3436	11.0	18.0
3	amc rebel sst	8	304.0	150	3433	12.0	16.0
4	ford torino	8	302.0	140	3449	10.5	17.0

3. Przejdźmy do wyboru cech. Usuńmy kolumnę NAME, ponieważ jest ona jedynie identyfikatorem aut. Kolumny służące do identyfikacji wierszy w naszym zbiorze danych nie mają znaczenia dla modelu treningowego, więc należy się ich pozbyć:

```
dataset=dataset.drop(columns=['NAME'])
```

4. Przekonwertujmy wszystkie zmienne wejściowe i przypiszmy brakujące wartości:

```
dataset= dataset.apply(pd.to_numeric, errors='coerce')
dataset.fillna(0, inplace=True)
```

   Imputacja poprawia jakość danych i przygotowuje je do użycia w trenowaniu modelu. Przejdźmy więc do tego ostatniego etapu.

5. Podzielimy dane na zbiór treningowy i testowy:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
↪random_state = 0)
```

W ten sposób powstały następujące cztery struktury danych:

- `X_train`. Struktura danych przechowująca cechy danych treningowych.
- `X_test`. Struktura danych przechowująca cechy danych testowych.
- `y_train`. Wektor przechowujący wartości etykiet w danych treningowych.
- `y_test`. Wektor przechowujący wartości etykiet w danych testowych.

Teraz jesteśmy gotowi, by skorzystać z tych danych w trzech algorytmach regresji i porównać ich efektywność.

Regresja liniowa

Ze wszystkich technik nadzorowanego uczenia maszynowego regresja liniowa jest najłatwiejsza do zrozumienia. Zaczniemy od prostej regresji liniowej, a następnie rozszerzymy to pojęcie do regresji wielomianowej.

Prosta regresja liniowa

W swojej najprostszej formie regresja liniowa określa relację między pojedynczą, zależną, ciągłą zmienną i inną pojedynczą, niezależną, ciągłą zmienną. Prosta regresja pokazuje, do jakiego stopnia zmiany w zmiennej zależnej (przedstawianej na osi y) mogą być tłumaczone przez zmienną objaśniającą (przedstawianą na osi x). Można sformułować to następująco:

$$y = (X)\omega + \alpha$$

Wzór ten można objaśnić zgodnie z poniższymi definicjami:

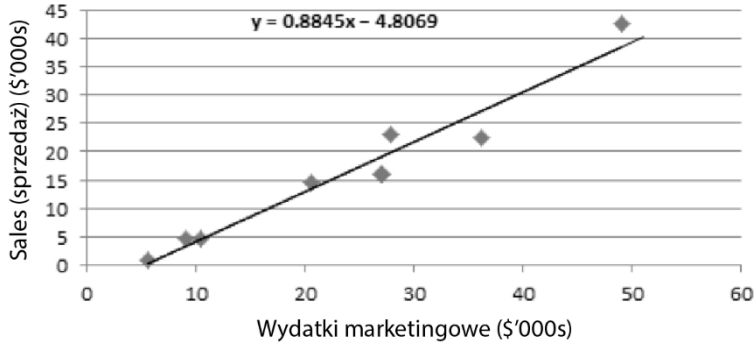
- y to zmienna zależna;
- X to zmienna niezależna;
- ω to nachylenie, które pokazuje, o ile podnosi się linia na wykresie z każdym przyrostem X;
- α to punkt przecięcia prostej z osią współrzędnych, który wskazuje wartość y, gdy $X = 0$.

Do przykładów relacji między pojedynczą, zależną, ciągłą zmienną i inną pojedynczą, niezależną, ciągłą zmienną należą:

- Waga danej osoby i liczba spożywanych przez nią kalorii.
- Cena nieruchomości i jej powierzchnia w metrach kwadratowych.
- Wilgotność powietrza i prawdopodobieństwo opadów.

W przypadku regresji liniowej zarówno dane wejściowe (niezależne), jak i wyjściowe (zależne) muszą być numeryczne. Najlepszą relację znajduje się, ograniczając sumę kwadratów pionowej odległości każdego punktu od linii poprowadzonej przez wszystkie punkty. Zakłada się, że między predykatorem a zmienną celu zachodzi relacja liniowa, np. im więcej pieniędzy zainwestuje się w rozwój i badania, tym wyższa będzie sprzedaż.

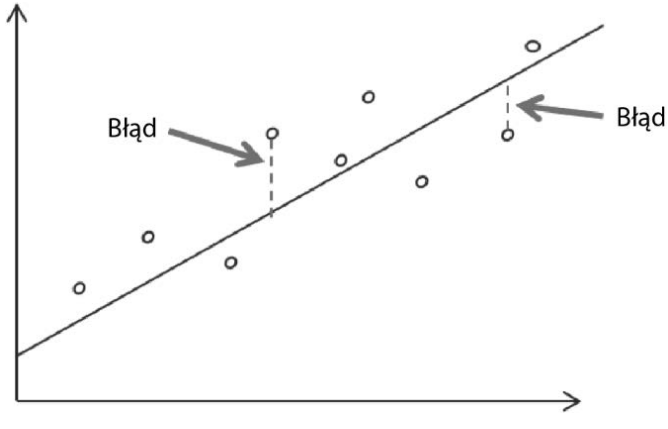
Przyjrzymy się konkretnemu przykładowi. Spróbujmy sformalizować związek między wydatkami na marketing a sprzedażą danego produktu. Okazuje się, że są one bezpośrednio ze sobą związane. Wydatki na marketing i sprzedaż zaznaczono na poniższym dwuwymiarowym wykresie w postaci niebieskich punktów. Relację tę można oszacować, rysując prostą, tak jak na poniższym wykresie:



Gdy zaznaczymy prostą, możemy zauważyć matematyczny związek między wydatkami a sprzedażą.

Ewaluacja regresorów

Prosta, jaką narysowaliśmy, jest przybliżeniem relacji między zmienną zależną i niezależną. Nawet najbardziej precyzyjne przybliżenie będzie wykazywało pewne odchylenie od rzeczywistych wartości, co widać na poniższym wykresie:



Standardowym sposobem mierzenia efektywności modeli regresji liniowej jest korzystanie z pierwiastka błędu średniokwadratowego (*RMSE*). Matematycznie oblicza on odchylenie standardowe błędów popełnionych przez wytrenowany model. Dla pewnego przypadku w zbiorze treningowym funkcja straty wyliczana jest w następujący sposób:

$$Loss(\acute{y}^{(i)}, y^{(i)}) = 1/2(\acute{y}^{(i)} - y^{(i)})^2$$

Prowadzi to do następującej funkcji kosztu, minimalizującej stratę we wszystkich próbkach zbioru treningowego:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\acute{y}^{(i)} - y^{i}\right)^2}$$

Spróbujmy zrozumieć pierwiastek błędu średniokwadratowego. Jeśli RMSE wynosi 50 dolarów w przykładowym modelu, który przewiduje cenę pewnego produktu, oznacza to, że ok. 68,2% przewidywań nie będzie różniło się od rzeczywistej ceny produktu o więcej niż 50 dolarów (tj. α). Oznacza to także, że 95% przewidywań nie będzie się różniło od rzeczywistej ceny produktu o więcej niż 100 dolarów (tj. 2α). Wreszcie, 99,7% przewidywań nie będzie się różniło od rzeczywistej ceny produktu o więcej niż 150 dolarów.

Regresja wielomianowa

Rzeczywistość współczesnej analizy jest taka, że w większości przypadków mamy więcej niż jedną zmienną niezależną. Regresja wielomianowa jest rozszerzeniem prostej regresji liniowej. Główna różnica polega natomiast na tym, że w tej pierwszej mamy do czynienia z dodatkowymi beta czynnikami dodatkowych zmiennych predykatora. Kiedy trenujemy model, powinniśmy szukać takich beta czynników, które minimalizują błędy równania liniowego. Spróbujmy matematycznie ująć związek między zmienną zależną a zbiorem zmiennych niezależnych (cech).

Podobnie jak w przypadku prostego równania liniowego zależna zmienna y jest określana jako suma wartości w punkcie przecięcia osi układu współrzędnych i β czynników pomnożonych przez wartość x dla każdej z cech i:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_i x_i + \varepsilon$$

W powyższym wzorze ε odpowiada błędowi i wskazuje, że przewidywania nie będą doskonałe.

Czynniki β sprawiają, że każda cecha może mieć inny szacowany wpływ na wartość y, ponieważ y zmieni się o wartość β_i przy każdym wzroście x_i o jednostkę. Co więcej, punkt przecięcia osi układu współrzędnych (α) pokazuje oczekiwaną wartość y, gdy wszystkie zmienne niezależne są równe *0*.

Zauważ, że wszystkie zmienne w powyższym równaniu można przedstawić jako wektory. Zmienna celu i predykator są wektorami, podobnie jak czynniki regresji β i błędy ε.

Wykorzystanie algorytmu regresji liniowej do wyzwania dla regresji

Wytrenujmy model przy użyciu zbioru treningowego:

1. Zacznijmy od zaimportowania pakietu do obsługi regresji liniowej:

```
from sklearn.linear_model import LinearRegression
```

2. Stwórzmy instancję modelu regresji liniowej i wytrenujmy go przy użyciu danych treningowych:

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

3. Możemy zabrać się już za przewidywanie wyników w zbiorze testowym:

```
y_pred = regressor.predict(X_test)
from sklearn.metrics import mean_squared_error
from math import sqrt
sqrt(mean_squared_error(y_test, y_pred))
```

4. Dane wyjściowe wygenerowane przez powyższy kod będą wyglądały następująco:

```
Out[10]: 4.36214129677179
```

Jak wspominałem wcześniej, RMSE to standardowe odchylenie błędu. Wskazuje ono, że 68,2% przewidywań nie będzie różniło się od rzeczywistej wartości zmiennej celowej o więcej niż 4,36.

Kiedy używa się regresji liniowej?

Z regresji liniowej korzysta się do rozwiązywania wielu rzeczywistych problemów, między innymi:

- przewidywania sprzedaży;
- ustalania optymalnej ceny produktów;
- wyliczania związku przyczynowego między zdarzeniem a odpowiedzią, np. w testach klinicznych leków, testach inżynierii bezpieczeństwa czy badaniach marketingowych;
- wskazywania wzorców służących do przewidywania przyszłych zachowań opartych na znanych kryteriach — np. przewidywania liczby zgłoszonych szkód ubezpieczeniowych, zniszczeń w wyniku katastrof naturalnych, wyników wyborów czy wskaźnika przestępczości.

Wady regresji liniowej

Regresja liniowa ma jednak pewne wady, m.in.:

- sprawdza się tylko w zbiorach danych numerycznych,
- dane jakościowe muszą przejść wstępną obróbkę,
- nie radzi sobie dobrze z brakującymi danymi,
- opiera się na założeniach dotyczących danych.

Algorytm drzewa regresji

Algorytm drzewa regresji przypomina algorytm drzewa klasyfikacji z tą różnicą, że zmienna celu jest w pierwszym przypadku zmienną ciągłą, a nie jakościową.

Wykorzystanie drzewa regresji do wyzwania dla regresji

W tej sekcji przekonamy się, jak zastosować algorytm drzewa regresji do wyzwania, jakie sobie postawiliśmy:

1. Zaczniemy od wytrenowania modelu za pomocą algorytmu drzewa regresji:

```
In [43]: from sklearn.tree import DecisionTreeRegressor
         regressor = DecisionTreeRegressor(max_depth=3)
         regressor.fit(X_train, y_train)

Out[43]: DecisionTreeRegressor(criterion='mse', max_depth=4, max_features=None,
                               max_leaf_nodes=None, min_impurity_decrease=0.0,
                               min_impurity_split=None, min_samples_leaf=1,
                               min_samples_split=2, min_weight_fraction_leaf=0.0,
                               presort=False, random_state=None, splitter='best')
```

2. Po jego wytrenowaniu możemy zająć się przewidywaniem wartości:

```
y_pred = regressor.predict(X_test)
```

3. Następnie musimy wyliczyć wartość błędu średniokwadratowego do oceny efektywności modelu:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
sqrt(mean_squared_error(y_test, y_pred))
```

Uzyskamy następujący wynik:

```
Out[45]: 5.2771702288377
```

Regresyjny algorytm wzmocnienia gradientowego

Przejdźmy do regresyjnego algorytmu wzmocnienia gradientowego. Korzysta on z metody zespolonej opierającej się na szeregu drzew decyzyjnych, by trafniej ująć wzorce w danych.

Wykorzystanie algorytmu wzmocnienia gradientowego do wyzwania dla regresji

W tej sekcji przekonamy się, jak zastosować algorytm wzmocnienia gradientowego do wyzwania, jakie sobie postawiliśmy:

1. Zaczniemy od wytrenowania modelu za pomocą algorytmu wzmocnienia gradientowego (zobacz rysunek na następnej stronie).

2. Po jego wytrenowaniu możemy się zająć przewidywaniem wartości:

```
y_pred = regressor.predict(X_test)
```

3. Następnie musimy wyliczyć wartość błędu średniokwadratowego do oceny efektywności modelu:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
sqrt(mean_squared_error(y_test, y_pred))
```

4. Uzyskamy następujący wynik:

```
Out[7]: 4.034836373089085
```

```
In [5]: from sklearn import ensemble
        params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
                  'learning_rate': 0.01, 'loss': 'ls'}
        regressor = ensemble.GradientBoostingRegressor(**params)

        regressor.fit(X_train, y_train)
```

```
Out[5]: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                                  learning_rate=0.01, loss='ls', max_depth=4,
                                  max_features=None, max_leaf_nodes=None,
                                  min_impurity_decrease=0.0, min_impurity_split=None,
                                  min_samples_leaf=1, min_samples_split=2,
                                  min_weight_fraction_leaf=0.0, n_estimators=500,
                                  n_iter_no_change=None, presort='auto',
                                  random_state=None, subsample=1.0, tol=0.0001,
                                  validation_fraction=0.1, verbose=0, warm_start=False)
```

Zwycięzcą wśród algorytmów regresji jest…

Przeanalizujmy efektywność trzech algorytmów regresji, których używaliśmy z tym samym zbiorem danych i w dokładnie tych samych zastosowaniach:

Algorytm	Błąd średniokwadratowy
Regresja liniowa	4,36214129677179
Drzewo regresji	5,2771702288377
Regresyjne wzmacnianie gradientowe	4,034836373089085

Na podstawie tego podsumowania efektywności algorytmów regresji wyraźnie widać, że najlepiej wypada regresyjne wzmacnianie gradientowe, ponieważ jego wskaźnik RSME jest najniższy. Na drugim miejscu jest prosta regresja liniowa, natomiast drzewo regresji w tym przypadku wypada najsłabiej.

Praktyczny przykład, jak przewidywać pogodę

Spójrzmy, jak wykorzystać omawiane w tym rozdziale zagadnienia do przewidywania pogody. Załóżmy, że chcemy dowiedzieć się, czy jutro będzie padać, w oparciu o dane pogodowe dla danego miasta z ostatniego roku.

Dane dostępne do wytrenowania tego modelu znajdują się w pliku CSV o nazwie `weather.csv`.

1. Zaimportujmy dane jako ramkę danych modułu `pandas`:

```
import numpy as np
import pandas as pd
df = pd.read_csv("weather.csv")
```

2. Sprawdźmy, jak wyglądają kolumny tej ramki danych:

```
In [63]: df.columns
Out[63]: Index(['Date', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
       'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
      dtype='object')
```

3. Następnie warto rzucić okiem na dane z 13 pierwszych kolumn w kilku pierwszych wierszach w pliku `weather.csv`:

```
In [124]: df.iloc[:,0:12].head()
```

Out[124]:

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm
0	2007-11-01	8.0	24.3	0.0	3.4	6.3	7	30.0	12	7	6.0	20
1	2007-11-02	14.0	26.9	3.6	4.4	9.7	1	39.0	0	13	4.0	17
2	2007-11-03	13.7	23.4	3.6	5.8	3.3	7	85.0	3	5	6.0	6
3	2007-11-04	13.3	15.5	39.8	7.2	9.1	7	54.0	14	13	30.0	24
4	2007-11-05	7.6	16.1	2.8	5.6	10.6	10	50.0	10	2	20.0	28

4. A teraz spójrzmy na ostatnich 10 kolumn tego samego pliku:

```
In [127]: df.iloc[:,12:25].head()
```

Out[127]:

	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow
0	68	29	1019.7	1015.0	7	7	14.4	23.6	0	3.6	1
1	80	36	1012.4	1008.4	5	3	17.5	25.7	1	3.6	1
2	82	69	1009.5	1007.2	8	7	15.4	20.2	1	39.8	1
3	62	56	1005.5	1007.0	2	7	13.5	14.1	1	2.8	1
4	68	49	1018.3	1018.5	7	7	11.1	15.4	1	0.0	0

5. Niech `x` oznacza cechy w danych wejściowych. Usuniemy kolumnę `Date` z listy cech, ponieważ nie jest ona przydatna w kontekście naszych prognoz. Usuniemy także etykietę `RainTomorrow`:

```
x = df.drop(['Date','RainTomorrow'],axis=1)
```

6. Niech `y` oznacza etykietę:

```
y = df['RainTomorrow']
```

7. Podzielimy teraz dane na zbiór testowy i treningowy za pomocą funkcji `train_test_split`:

```
from sklearn.model_selection import train_test_split
train_x , train_y ,test_x , test_y = train_test_split(x,y ,
test_size = 0.2,random_state = 2)
```

8. Ponieważ etykieta jest zmienną binarną, trenujemy klasyfikator. Dobrym wyborem wydaje się regresja logistyczna. Stwórzmy najpierw instancję modelu regresji logistycznej:

```
model = LogisticRegression()
```

9. Możemy teraz użyć `train_x` i `test_x` do wytrenowania modelu:

```
model.fit(train_x , test_x)
```

10. Gdy model jest już wytrenowany, skorzystamy z niego do przewidywań:

```
predict = model.predict(train_y)
```

11. Zweryfikujmy też dokładność modelu:

```
In [89]: predict = model.predict(train_y)

In [90]: from sklearn.metrics import accuracy_score

In [91]: accuracy_score(predict , test_y)

Out[91]: 0.9696969696969697
```

Taki klasyfikator binarny może być używany do przewidzenia prawdopodobieństwa wystąpienia deszczu następnego dnia.

Podsumowanie

Rozdział ten zaczął się od wprowadzenia podstaw nadzorowanego uczenia maszynowego. Następnie szczegółowo omówiłem różne algorytmy klasyfikacji. Potem zająłem się sposobami ewaluacji efektywności klasyfikatorów oraz kilkoma algorytmami regresyjnymi. Poświęciłem też nieco miejsca metodom oceny efektywności algorytmów, których używaliśmy.

Tematem kolejnego rozdziału będą sieci neuronowe i algorytmy uczenia głębokiego. Przyjrzymy się metodom trenowania takich sieci oraz narzędziom i modelom oceny i wdrażania sieci neuronowych.

Oto algorytm: poznaj, zaimplementuj, zastosuj!

Wiedza o algorytmach jest niezbędna każdemu, kto rozwiązuje problemy programistyczne. Algorytmy są również ważne w teorii i praktyce obliczeń. Każdy programista powinien znać możliwie szeroki ich zakres. Powinien też umieć z nich korzystać przy rozwiązywaniu rzeczywistych problemów, w tym przy projektowaniu algorytmów, ich modyfikacji i implementacji. Niezależnie od tego, czy zajmujesz się sztuczną inteligencją, zabezpieczaniem systemów informatycznych lub inżynierią danych, musisz dobrze zrozumieć, czym właściwie są i jak działają algorytmy.

Ta książka jest praktycznym wprowadzeniem do algorytmów i ich zastosowania. Znalazły się w niej podstawowe informacje i pojęcia dotyczące algorytmów, ich działania, a także ograniczeń, jakim podlegają. Opisano też techniki ich projektowania z uwzględnieniem wymagań dotyczących struktur danych. Zaprezentowano klasyczne algorytmy sortowania i wyszukiwania, algorytmy grafowe, jak również wiele zagadnień związanych ze sztuczną inteligencją: algorytmy uczenia maszynowego, sieci neuronowych i przetwarzania języka naturalnego. Ważną częścią publikacji są rozdziały poświęcone przetwarzaniu danych i kryptografii oraz algorytmom powiązanym z tymi zagadnieniami. Wartościowym podsumowaniem prezentowanych treści jest omówienie technik pracy z problemami NP-trudnymi.

W tej książce między innymi:

- struktury danych i algorytmy w bibliotekach Pythona
- algorytm grafowy służący do wykrywania oszustw w procesie analizy sieciowej
- przewidywanie pogody przy użyciu algorytmów uczenia nadzorowanego
- rozpoznawanie obrazu za pomocą syjamskich sieci neuronowych
- tworzenie systemu rekomendacji filmów
- szyfrowanie symetryczne i asymetryczne podczas wdrażania modelu uczenia maszynowego

Imran Ahmad jest certyfikowanym instruktorem Google z wieloletnim doświadczeniem. Wykłada Pythona, uczenie maszynowe i głębokie, algorytmikę oraz zagadnienia *big data*. Przez ostatnie lata pracował w rządowym laboratorium Kanady nad projektem z zakresu uczenia maszynowego. Obecnie zajmuje się algorytmami używającymi GPU do optymalnego trenowania złożonych modeli uczenia maszynowego.

ISBN 978-83-283-7777-6
9 788328 377776
Cena: 69,00 zł

Packt>