O'REILLY®

Python dla DevOps

Naucz się bezlitośnie skutecznej automatyzacji

Noah Gift, Kennedy Behrman,
Alfredo Deza, Grig Gheorghiu

Helion

Tytuł oryginału: Python for DevOps

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-6830-9

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

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 14.

MLOps i inżynieria uczenia maszynowego

Jednym z najbardziej atrakcyjnych zawodów w 2020 roku jest inżynier uczenia maszynowego. Do innych „gorących" zawodów można zaliczyć inżyniera danych (ang. *data engineer*), analityka danych (ang. *data scientist*) i analityka uczenia maszynowego (ang. *machine learning scientist*). Chociaż można być specjalistą DevOps, to w gruncie rzeczy DevOps to zachowanie, a zasady DevOps można stosować do każdego projektu oprogramowania, w tym do projektów uczenia maszynowego. Przyjrzyjmy się niektórym, najważniejszym praktykom DevOps: ciągła integracja (ang. *Continuous Integration*), ciągłe dostarczanie (ang. *Continuous Delivery*), mikrousługi (ang. *microservices*), infrastruktura jako kod (IaC), monitorowanie i logowanie oraz komunikacja i współpraca. Które z nich nie mają zastosowania do uczenia maszynowego?

Im bardziej złożony projekt inżynierii oprogramowania — a uczenie maszynowe jest złożone — tym większa potrzeba stosowania zasad DevOps. Czy istnieje lepszy przykład mikrousługi niż API, które realizuje prognozy korzystając z technik uczenia maszynowego? W tym rozdziale omówimy szczegóły stosowania technik uczenia maszynowego w sposób profesjonalny i powtarzalny, przy użyciu zasad i technik DevOps.

Czym jest uczenie maszynowe?

Uczenie maszynowe jest metodą korzystania z algorytmów w celu automatycznego uczenia się na podstawie danych. Istnieją cztery główne rodzaje uczenia maszynowego: nadzorowane (ang. *supervised*), półnadzorowane (ang. *semi-supervised*), nienadzorowane (ang. *unsupervidsed*) i uczenie poprzez wzmacnianie (ang. *reinforcement*).

Nadzorowane uczenie maszynowe

W nadzorowanym uczeniu maszynowym poprawne odpowiedzi są znane i opisane. Na przykład, gdybyś chciał przewidzieć wzrost na podstawie wagi, mógłbyś zebrać przykłady wzrostu i wagi ludzi. Wzrost byłby celem, a waga byłaby cechą.

Spróbujmy przeanalizować przykład nadzorowanego uczenia maszynowego:

- Wejściowy zestaw danych (*https://oreil.ly/jzWmI*).
- 25 000 sztucznych rekordów z informacjami na temat wzrostu i wagi osób w wieku 18 lat.

Pobieranie danych (ang. ingest)

```
In[0]:
    import pandas as pd

In[7]:

    df = pd.read_csv(
        "https://raw.githubusercontent.com/noahgift/\
        regression-concepts/master/\
        height-weight-25k.csv")
    df.head()

Out[7]:

Index  Height-Inches  Weight-Pounds
0 1    65.78331       112.9925
1 2    71.51521       136.4873
2 3    69.39874       153.0269
3 4    68.21660       142.3354
4 5    67.78781       144.2971
```

Eksploracyjna analiza danych (ang. Explorative Data Analysis — EDA)

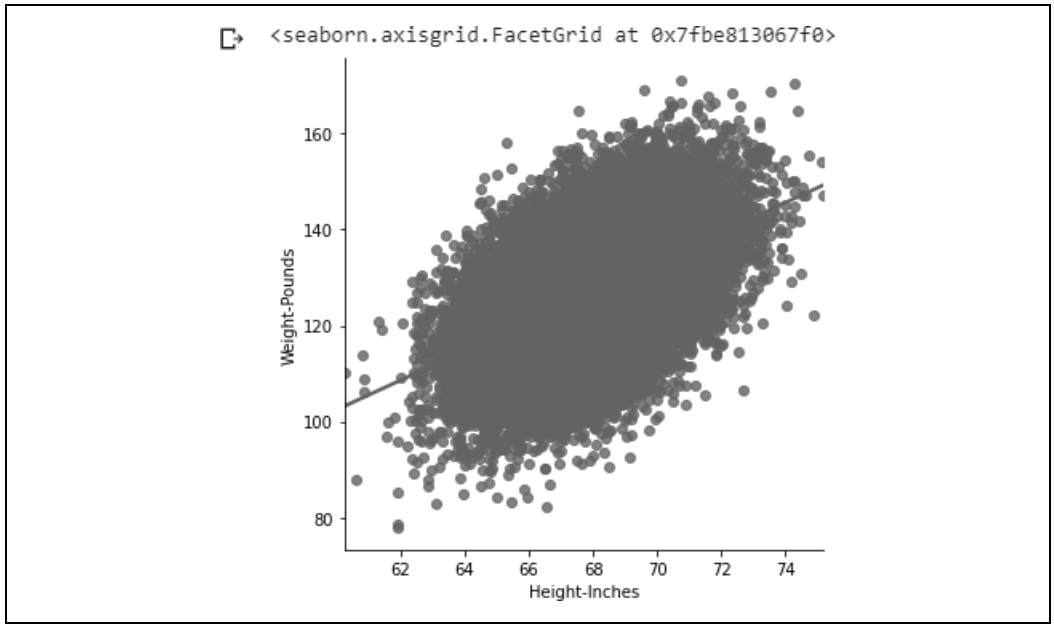
Przyjrzyjmy się danym i zobaczmy, jak je można zbadać.

Wykres punktowy. W tym przykładzie do wizualizacji zestawu danych skorzystano z `seaborn`, popularnej biblioteki do tworzenia wykresów w Pythonie. Aby ją zainstalować w notatniku, można skorzystać z polecenia `!pip install seaborn`. Wszystkie inne biblioteki wykorzystane w tym punkcie także można zainstalować za pomocą polecenia `!pip install <nazwa pakietu>`. Jeśli używasz notatnika Colab, te biblioteki już zostały zainstalowane bez Twojego udziału.

Spójrz na wykres zależności wzrostu od wagi (rysunek 14.1).

```
In[0]:
    import seaborn as sns
    import numpy as np

In[9]:
    sns.lmplot("Height-Inches", "Weight-Pounds", data=df)
```



Rysunek 14.1. Wykres lm wzrost/waga

Statystyki opisowe

Teraz możemy wygenerować kilka statystyk opisowych.

In[10]:

```
df.describe()
```

Out[10]:

```
        Index          Height-Inches   Weight-Pounds
count   25000.000000   25000.000000    25000.000000
mean    12500.500000   67.993114       127.079421
std     7217.022701    1.901679        11.660898
min     1.000000       60.278360       78.014760
25%     6250.750000    66.704397       119.308675
50%     12500.500000   67.995700       127.157750
75%     18750.250000   69.272958       134.892850
max     25000.000000   75.152800       170.924000
```

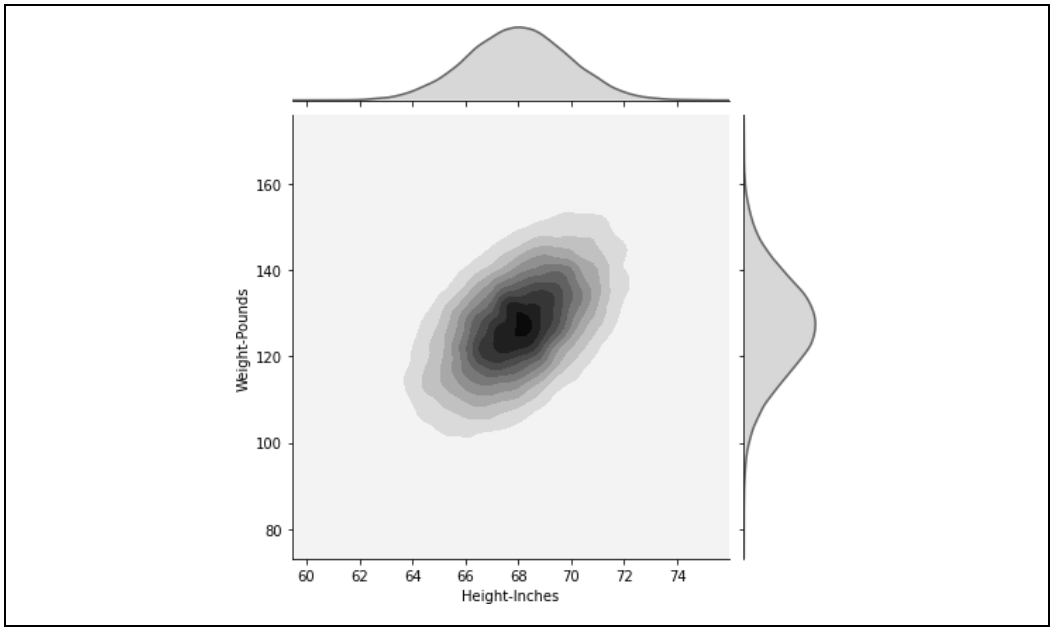
Rozkład gęstości jądra

Wykres rozkładu gęstości (rysunek 14.2) pokazuje wzajemny związek pomiędzy dwoma zmiennymi.

In[11]:

```
sns.jointplot("Height-Inches", "Weight-Pounds", data=df, kind="kde")
```

Out[11]:



Rysunek 14.2. Wykres gęstości

Modelowanie

Teraz przyjrzyjmy się modelowaniu. Modelowanie w uczeniu maszynowym zachodzi w czasie, kiedy algorytm uczy się na podstawie danych. Ogólna idea polega na wykorzystaniu poprzednich danych do przewidywania danych przyszłych.

Model regresji sklearn

Najpierw należy wyodrębnić z danych cechy (ang. *features*) i cele (ang. *targets*), a następnie podzielić zestaw danych na szkoleniowy i testowy. Pozwala to przetwarzać zestaw testowy oddzielnie, aby przetestować dokładność modelu wyszkolonego.

`In[0]:`

```
from sklearn.model_selection import train_test_split
```

Wyodrębnienie i zbadanie cechy i celu. Dobrym pomysłem jest jawne wyodrębnienie zmiennych celu i cechy i przekształcanie ich w jednej komórce. Następnie można sprawdzić ich kształt, aby uzyskać pewność, że jest to odpowiednia właściwość do zastosowania uczenia maszynowego za pomocą pakietu `sklearn`.

`In[0]:`

```
y = df['Weight-Pounds'].values # Cel
y = y.reshape(-1, 1)
X = df['Height-Inches'].values # Cechy
X = X.reshape(-1, 1)
```

`In[14]:`

```
y.shape
```

Out[14]:

```
(25000, 1)
```

Podział danych. Dane zostały podzielone w proporcji 80% / 20%.

In[15]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

Out[15]:

```
(20000, 1) (20000, 1)
(5000, 1) (5000, 1)
```

Dopasowanie modelu. W kolejnym kroku dopasowujemy model za pomocą algorytmu `Linear ↪Regression` zaimportowanego z modułu `sklearn`.

In[0]:

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
model = lm.fit(X_train, y_train)
y_predicted = lm.predict(X_test)
```

Wyświetl dokładność modelu regresji liniowej. Teraz możemy wyświetlić, jaką dokładność prognozowania ma wyszkolony model. Aby to zrobić, wyznaczamy błąd RMSE (ang. *root mean squared error* — błąd średniokwadratowy) danych prognozowanych i testowych.

In[18]:

```
from sklearn.metrics import mean_squared_error
from math import sqrt

# Błąd średniokwadratowy RMSE
rms = sqrt(mean_squared_error(y_predicted, y_test))
rms
```

Out[18]:

```
10.282608230082417
```

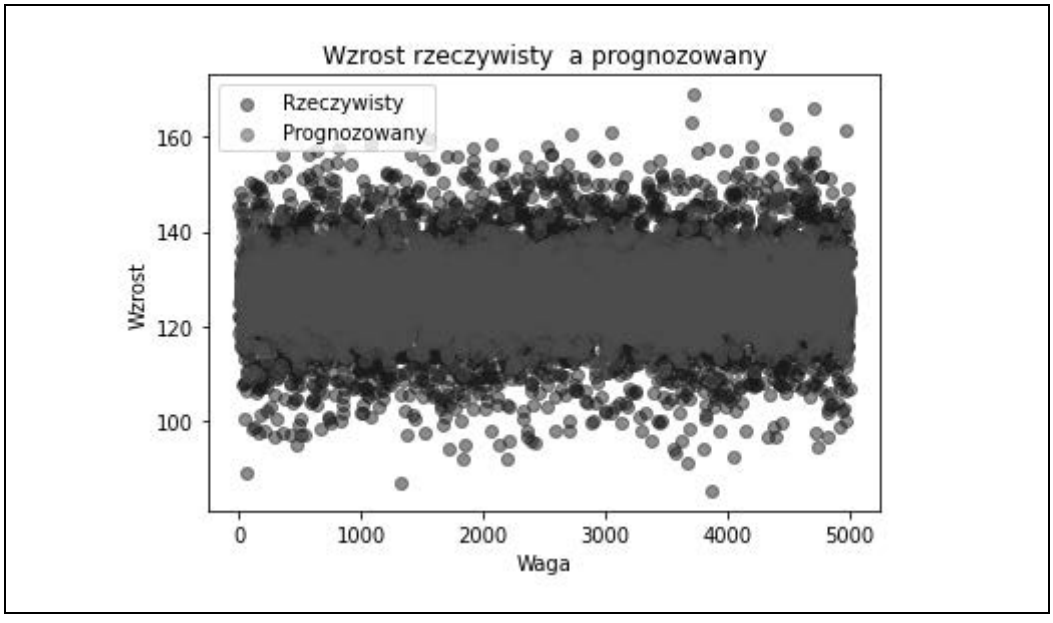
Wykreślenie prognozowanego wzrostu w porównaniu z rzeczywistym. Następnie wykreślimy prognozowany wzrost w porównaniu z rzeczywistym (rysunek 14.3), aby zobaczyć, jak dobrze ten model radzi sobie z prognozami.

In[19]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

ax.scatter(x = range(0, y_test.size), y=y_test, c = 'blue', label = 'Rzeczywisty',
↪alpha = 0.5)
ax.scatter(x = range(0, y_predicted.size), y=y_predicted, c = 'red',
    label = 'Prognozowany', alpha = 0.5)

plt.title('Wzrost rzeczywisty a prognozowany')
plt.xlabel('Waga')
plt.ylabel('Wzrost')
plt.legend()
plt.show()
```



Rysunek 14.3. Wzrost prognozowany w porównaniu z rzeczywistym

Jest to bardzo prosty, ale wartościowy przykład realistycznego przepływu pracy w celu stworzenia modelu uczenia maszynowego.

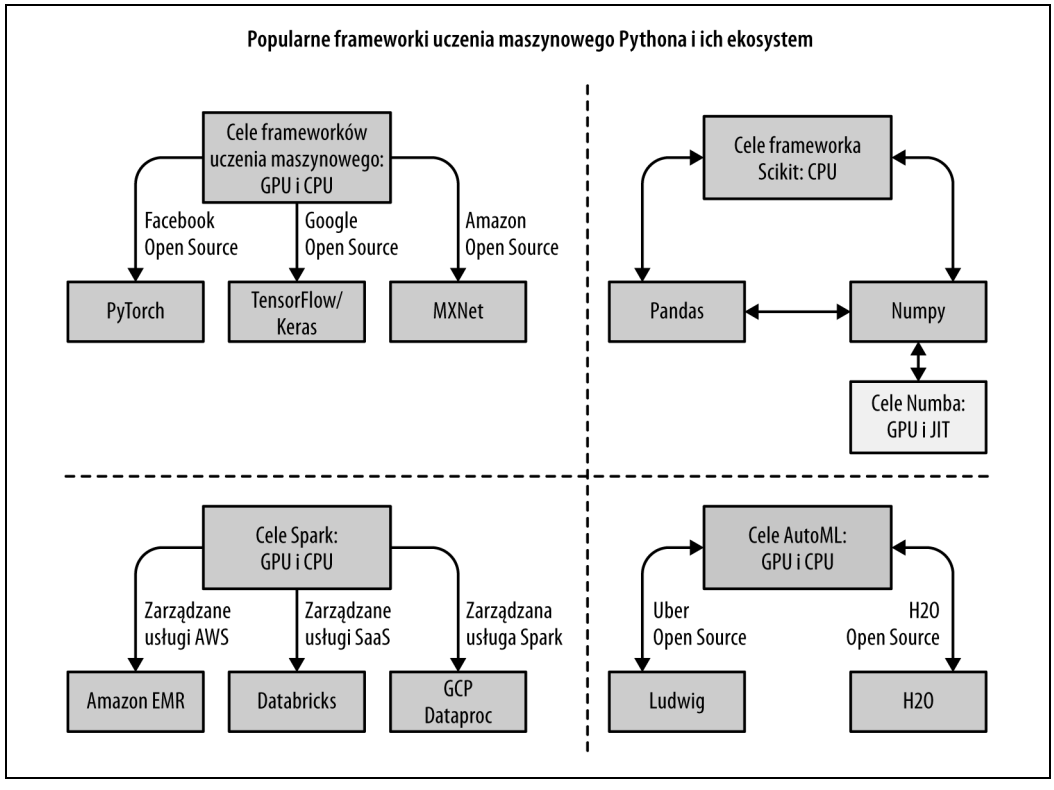
Ekosystem uczenia maszynowego w Pythonie

Rzućmy okiem na ekosystem uczenia maszynowego w Pythonie (rysunek 14.4).

W ekosystemie uczenia maszynowego Pythona można wyróżnić cztery główne obszary: uczenie głębokie, `sklearn`, AutoML i Spark. W obszarze uczenia głębokiego, najbardziej popularnymi frameworkami są (w kolejności odpowiadającej popularności): TensorFlow/Keras, PyTorch i MXNet. Google sponsoruje TensorFlow, Facebook sponsoruje PyTorch, a MXNet pochodzi z firmy Amazon. Framework MXNet opiszemy przy okazji omawiania narzędzia Amazon SageMaker. Należy zapamiętać, że te frameworki uczenia głębokiego korzystają z procesorów GPU, co sprawia, że są ponad pięćdziesięciokrotnie wydajniejsze w porównaniu z tymi, które wykorzystują układy CPU.

Ekosystem frameworka Sklearn często korzysta z bibliotek Pandas i Numpy. Framework Sklearn celowo nie używa układów GPU. Istnieje jednak projekt o nazwie Numba, który jawnie je wykorzystuje (zarówno układy NVIDIA, jak i AMD).

W obszarze AutoML dwoma najważniejszymi frameworkami są Uber z biblioteką Ludwig i H20 z biblioteką H20 AutoML. Oba pozwalają zaoszczędzić dużo czasu podczas programowania modeli uczenia maszynowego. Pozwalają również potencjalnie zoptymalizować istniejące modele uczenia maszynowego.



Rysunek 14.4. Ekosystem uczenia maszynowego Pythona

Istnieje także ekosystem Spark, który bazuje na dziedzictwie frameworka Hadoop. Spark może korzystać z układów GPU i CPU za pośrednictwem wielu różnych platform: między innymi Amazon EMR, Databricks i GCP Dataproc.

Uczenie głębokie z wykorzystaniem frameworka PyTorch

Po zdefiniowaniu ekosystemu Pythona dla uczenia maszynowego, przyjrzyjmy się przykładowi przeniesienia prostego przykładu regresji liniowej do frameworka PyTorch i spróbujmy go uruchomić na GPU CUDA. Prostym sposobem na uzyskanie dostępu do układu GPU NVIDIA jest użycie notatników Colab. Są one hostowane w usłudze Google. To notatniki zgodne z Jupyterem, które dają użytkownikowi swobodny dostęp zarówno do procesorów graficznych, jak i procesorów tensorowych (ang. *tensor processing units* — TPU). Kod z tego repozytorium (*https://oreil.ly/kQhKO*) możesz uruchomić na układach GPU.

Regresja z wykorzystaniem PyTorch

Najpierw dokonamy konwersji danych na typ `float32`.

`In[0]:`

```
# Dane treningowe
x_train = np.array(X_train, dtype=np.float32)
```

```
x_train = x_train.reshape(-1, 1)
y_train = np.array(y_train, dtype=np.float32)
y_train = y_train.reshape(-1, 1)
# Dane testowe
x_test = np.array(X_test, dtype=np.float32)
x_test = x_test.reshape(-1, 1)
y_test = np.array(y_test, dtype=np.float32)
y_test = y_test.reshape(-1, 1)
```

Należy pamiętać, że jeśli nie korzystasz z notatników Colab, być może będziesz zmuszony zainstalować pakiet PyTorch. Ponadto, jeśli używasz notatników Colab, możesz uzyskać dostęp do procesora NVIDIA GPU, aby uruchomić ten kod. Jeśli z nich nie korzystasz, musisz uruchomić ten kod na platformie, która jest wyposażona w układ GPU.

```
In[0]:
    import torch
    from torch.autograd import Variable
    class linearRegression(torch.nn.Module):
        def __init__(self, inputSize, outputSize):
            super(linearRegression, self).__init__()
            self.linear = torch.nn.Linear(inputSize, outputSize)
        def forward(self, x):
            out = self.linear(x)
            return out
```

Teraz stworzymy model z włączoną obsługą CUDA (przy założeniu, że uruchamiasz kod w notatniku Colab lub na maszynie z GPU NVIDIA).

```
In[0]:
    inputDim = 1            # pobiera zmienną 'x'
    outputDim = 1           # pobiera zmienną 'y'
    learningRate = 0.0001
    epochs = 1000
    model = linearRegression(inputDim, outputDim)
    model.cuda()

Out[0]:
    linearRegression(
        (linear): Linear(in_features=1, out_features=1, bias=True)
    )
```

Stwórz funkcję stochastycznego spadku wzdłuż gradientu (SGD) i funkcję straty.

```
In[0]:
    criterion = torch.nn.MSELoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=learningRate)
```

Teraz możesz przeszkolić model.

```
In[0]:
    for epoch in range(epochs):
        inputs = Variable(torch.from_numpy(x_train).cuda())
        labels = Variable(torch.from_numpy(y_train).cuda())
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        print(loss)
```

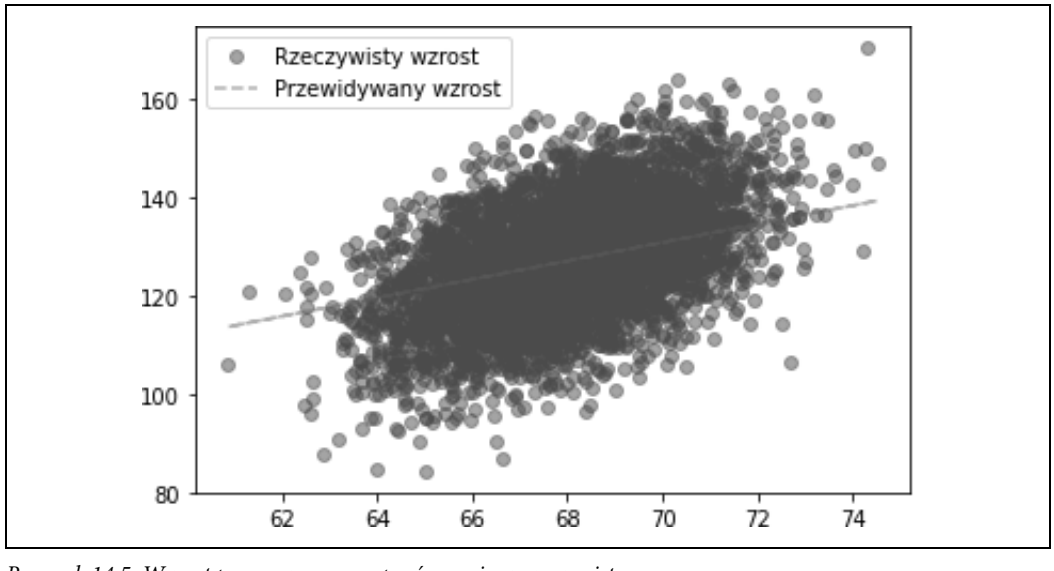
```
    # pobierz gradienty w.r.t. do parametrów
    loss.backward()
    # aktualizacja parametrów
    optimizer.step()
    print('epoch {}, loss {}'.format(epoch, loss.item()))
```

Wyjście z ponad 1000 przebiegów pominięto w celu zaoszczędzenia miejsca.

```
Out[0]:
    tensor(29221.6543, device='cuda:0', grad_fn=<MseLossBackward>)
    epoch 0, loss 29221.654296875
    tensor(266.7252, device='cuda:0', grad_fn=<MseLossBackward>)
    epoch 1, loss 266.72515869140625
    tensor(106.6842, device='cuda:0', grad_fn=<MseLossBackward>)
    epoch 2, loss 106.6842269897461
    ....output suppressed....
    epoch 998, loss 105.7930908203125
    tensor(105.7931, device='cuda:0', grad_fn=<MseLossBackward>)
    epoch 999, loss 105.7930908203125
```

Wykreślenie prognozowanego wzrostu w porównaniu z rzeczywistym. Spróbujmy teraz wykreślić prognozowany wzrost w zestawieniu z rzeczywistym (rysunek 14.5), tak jak w prostym modelu.

```
In[0]:
    with torch.no_grad():
        predicted = model(Variable(torch.from_numpy(x_test).cuda())).cpu().\
            data.numpy()
        print(predicted)
    plt.clf()
    plt.plot(x_test, y_test, 'go', label='Rzeczywisty wzrost', alpha=0.5)
    plt.plot(x_test, predicted, '--', label='Przewidywany wzrost', alpha=0.5)
    plt.legend(loc='best')
    plt.show()
```



Rysunek 14.5. Wzrost prognozowany w porównaniu z rzeczywistym

Wyświetlenie błędu RMSE. Na koniec spróbujmy wyświetlić błąd RMSE i porównajmy z poprzednimi wynikami.

In[0]:

```
# Błąd średniokwadratowy RMSE
rms = sqrt(mean_squared_error(x_test, predicted))
rms
```

Out[0]:

```
59.19054613663507
```

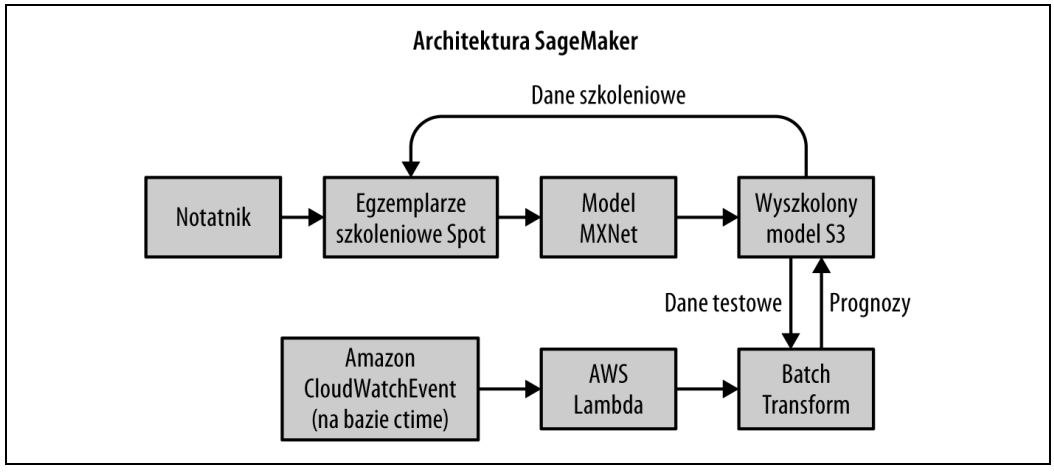
Wykorzystanie uczenia głębokiego wymagało użycia nieco więcej linii kodu, ale pojęcia są takie same, jak w przypadku zastosowania modelu sklearn. Ważnym wnioskiem z powyższego przykładu jest to, że procesory graficzne stają się integralną częścią produkcyjnych przepływów pracy. Nawet jeśli na co dzień nie korzystasz z technik uczenia głębokiego, warto zdobyć podstawową wiedzę o procesie budowania modeli uczenia maszynowego bazujących na GPU.

Platformy uczenia maszynowego w chmurze

Jednym z elementów infrastruktury uczenia maszynowego, który staje się coraz bardziej powszechny, są platformy uczenia maszynowego w chmurze. Google oferuje platformę GCP AI (rysunek 14.6).

Rysunek 14.6. Platforma GCP AI

Platforma GCP oferuje wiele wysokopoziomowych komponentów automatyzacji — od przygotowania danych do ich etykietowania. Platforma AWS oferuje Amazon SageMaker (rysunek 14.7).

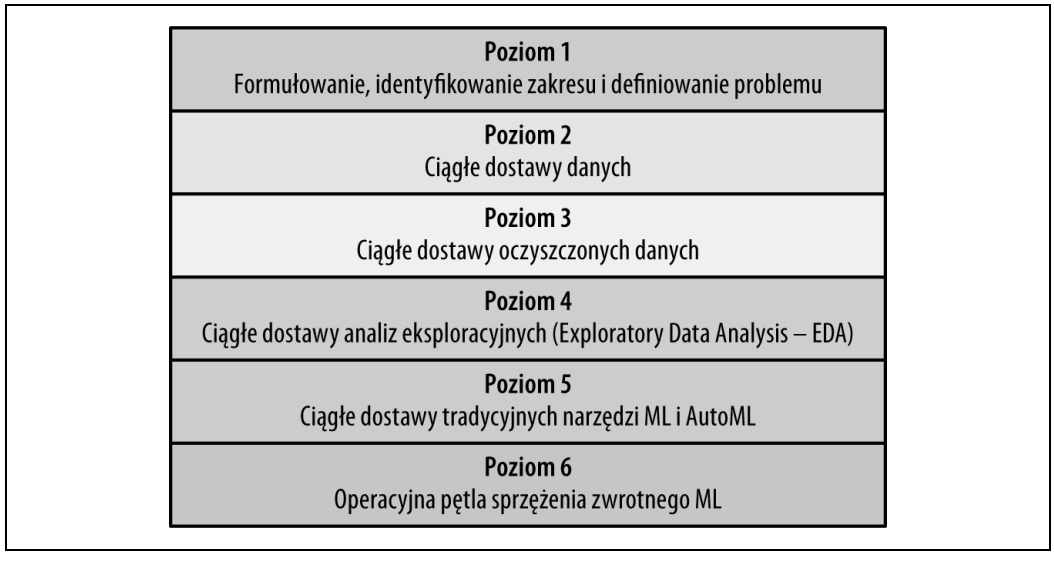


Rysunek 14.7. Amazon SageMaker

Platforma SageMaker również oferuje wiele elementów wysokiego poziomu, w tym szkolenie na egzemplarzach Spot oraz elastyczne punkty końcowe prognozowania.

Model dojrzałości uczenia maszynowego

Obecnie jednym z największych wyzwań jest uświadomienie sobie, że w firmach, które chcą korzystać z uczenia maszynowego, potrzebne są transformacyjne zmiany. Wybrane wyzwania i możliwości zaprezentowano na modelu dojrzałości uczenia maszynowego (rysunek 14.8).



Rysunek 14.8. Model dojrzałości uczenia maszynowego

Najważniejsza terminologia uczenia maszynowego

Poniżej zdefiniowano najważniejsze pojęcia związane z uczeniem maszynowym, które będą pomocne w pozostałej części rozdziału.

Uczenie maszynowe

Sposób budowania modeli matematycznych na podstawie danych przykładowych lub szkoleniowych.

Model

Produkt w aplikacji uczenia maszynowego. Prostym przykładem jest równanie liniowe — modelem jest linia prosta, która pozwala przewidzieć zależność pomiędzy współrzędną X a Y.

Cecha

Cechą jest kolumna w arkuszu kalkulacyjnym, używana jako motywacja do stworzenia modelu uczenia maszynowego. Dobrym przykładem są punkty zdobyte w poszczególnych meczach przez drużynę z ligi NBA.

Cel

Cel to kolumna w arkuszu kalkulacyjnym, którą próbujemy odgadnąć. Dobrym przykładem jest liczba meczów, które wygra w określonym sezonie drużyna z ligi NBA.

Nadzorowane uczenie maszynowe

Jest to rodzaj uczenia maszynowego, w którym prognozujemy przyszłe wartości na podstawie znanych, prawidłowych wartości historycznych. Dobrym przykładem może być przewidywanie liczby zwycięstw drużyny NBA w sezonie za pomocą cechy oznaczającej liczbę punktów na mecz.

Nienadzorowane uczenie maszynowe

Jest to rodzaj uczenia maszynowego, który korzysta z nieoznakowanych danych. Zamiast przewidywać przyszłe wartości, znajduje — za pomocą takich narzędzi, jak klasteryzacja — ukryte wzorce, które z kolei mogą być wykorzystywane jako etykiety. Dobrym przykładem może być tworzenie klastrów graczy NBA, którzy mają podobne liczby zdobytych punktów, zbiórek, blokad i asyst. Jeden z klastrów można nazwać „Wysocy, najlepsi gracze", a inny „Obrońcy, którzy zdobywają dużo punktów".

Uczenie głębokie

Jest to rodzaj uczenia maszynowego wykorzystujący sztuczne sieci neuronowe, które mogą być użyte do nadzorowanego lub nienadzorowanego uczenia maszynowego. Najbardziej popularnym frameworkiem do uczenia głębokiego jest TensorFlow firmy Google.

Scikit-learn

Jest to jeden z najbardziej popularnych frameworków uczenia maszynowego w Pythonie.

Pandas

Jedna z najbardziej popularnych bibliotek do wykonywania zadań polegających na manipulowaniu danymi i ich analizowaniu. Działa dobrze z bibliotekami scikit-learn i NumPy.

Numpy

Jedna z głównych bibliotek Pythona do wykonywania niskopoziomowych obliczeń naukowych. Zapewnia wsparcie dla dużych, wielowymiarowych tablic, zawiera ponadto obszerną kolekcję wysokopoziomowych funkcji matematycznych. Jest szeroko stosowana razem z bibliotekami scikit-learn, Pandas i TensorFlow.

Poziom 1. Formułowanie, identyfikowanie zakresu i definiowanie problemu

Przyjrzyjmy się pierwszej warstwie. Podczas implementowania technik uczenia maszynowego w firmie, ważne jest, aby zastanowić się, jakie problemy wymagają rozwiązania oraz w jaki sposób te problemy powinny być sformułowane. Jedną z najważniejszych przyczyn niepowodzenia projektów uczenia maszynowego jest fakt, że organizacje nie zadały sobie wcześniej pytania o to, jakie problemy muszą rozwiązać.

Dobrą analogią do tej sytuacji może być tworzenie aplikacji mobilnej dla sieci restauracji w San Francisco. Naiwnym podejściem mogłoby być natychmiastowe rozpoczęcie budowy natywnych aplikacji iOS i Android (przez dwa zespoły programistów). Typowy zespół pracujący nad każdą aplikacją mobilną mógłby składać się z trzech programistów pracujących w pełnym wymiarze godzin. Realizacja przedsięwzięcia wymagałaby zatem zatrudnienia sześciu programistów, z których każdy kosztuje około dwieście tysięcy dolarów rocznie. Koszt takiego projektu to około 1,2 mln dolarów. Czy aplikacja mobilna zapewni w skali roku większy przychód niż 1,2 mln dolarów? A jeśli nie, to czy istnieje tańsza alternatywa? Być może lepszym rozwiązaniem byłoby opracowanie zoptymalizowanej pod kątem wykorzystania w urządzeniach mobilnych aplikacji webowej przez już zatrudnionych w firmie programistów?

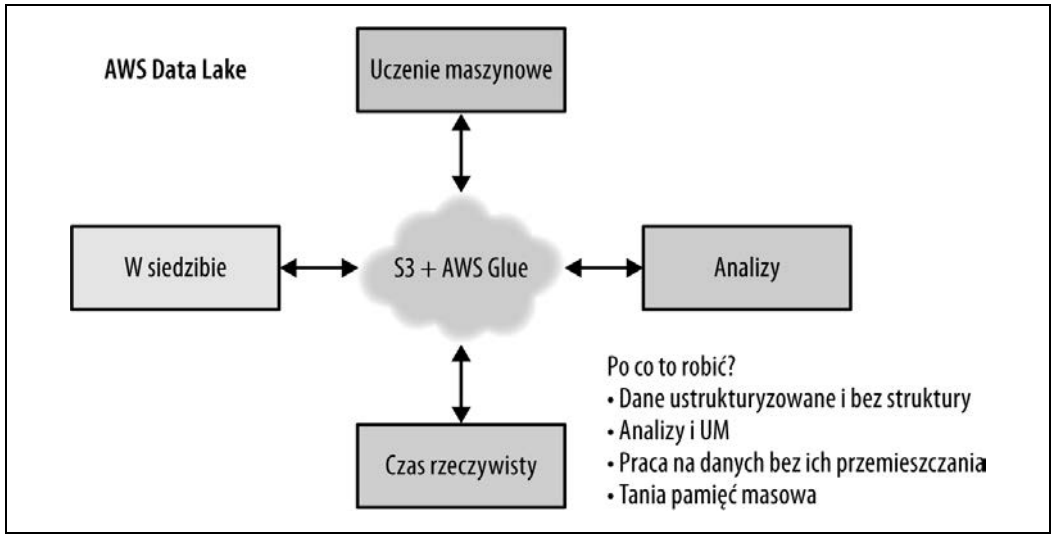
A może warto nawiązać współpracę z firmą, która specjalizuje się w dystrybucji żywności, i w całości zlecić to zadanie podmiotowi zewnętrznemu? Jakie są plusy i minusy takiego podejścia? Ten sam typ procesu myślowego może i powinien być stosowany do inicjatyw uczenia maszynowego i inżynierii danych. Na przykład, czy Twoja firma powinna zatrudniać sześciu specjalistów w dziedzinie uczenia maszynowego z tytułami doktora i z zarobkami na poziomie, powiedzmy, pięciuset tysięcy dolarów rocznie? A może istnieje jakaś alternatywa? Aby uzyskać większą szansę na sukces, wystarczy poświęcić trochę czasu na sformułowanie problemu i jego zakresu.

Poziom 2. Ciągłe dostawy danych

Jednym z fundamentów cywilizacji jest bieżąca woda. Już w 312 r. p.n.e. akwedukty rzymskie doprowadzały wodę na odległość wielu kilometrów, aby zapewnić ją zatłoczonym miastom. Bieżąca woda zapewniła infrastrukturę niezbędną dużym skupiskom ludzkim do odniesienia sukcesu. Tymczasem UNICEF szacuje, że w 2018 roku kobiety i dziewczęta na całym świecie poświęciły około dwustu milionów godzin dziennie na dostarczanie wody. Konsekwencje takiej sytuacji są znaczące — strata czasu, który można byłoby poświęcić nauce, opiece nad dziećmi, pracy lub odpoczynkowi.

Popularne jest wyrażenie, że „oprogramowanie zjada świat". Następstwem tego jest fakt, że wszystkie firmy produkujące oprogramowanie — co w przyszłości będzie oznaczało wszystkie firmy — będą musiały wdrożyć strategię korzystania z technik uczenia maszynowego i sztucznej inteligencji (ang. *artifficial intelligence* — AI). Częścią tej strategii jest poważniejsze myślenie o ciągłym dostarczaniu

danych. Podobnie jak bieżąca woda, „bieżące dane" pozwolą zaoszczędzić wiele godzin dziennie. Jednym z możliwych rozwiązań jest coś, co kryje się pod pojęciem „jeziora danych", które zilustrowano na rysunku 14.9.



Rysunek 14.9. Jezioro danych AWS

Na pierwszy rzut oka jezioro danych może wydawać się rozwiązaniem dla poszukiwań problemu albo narzędziem zbyt prostym, aby technika ta była do czegokolwiek przydatna. Przyjrzyjmy się jednak niektórym z problemów, które rozwiązuje:

- można przetwarzać dane bez ich przemieszczania,
- przechowywanie danych jest tanie,
- można w prosty sposób opracować zasady cyklu życia w celu archiwizowania danych,
- można w prosty sposób opracować zasady cyklu życia do zabezpieczania danych i ich audytu,
- systemy produkcyjne są oddzielone od przetwarzania danych,
- systemy produkcyjne mogą dysponować niemal nieskończoną przestrzenią do składowania danych oraz niemal nieskończonymi możliwościami wykonywania dyskowych operacji wejścia-wyjścia.

Alternatywą dla tej architektury jest często bałagan — odpowiednik chodzenia cztery godziny do studni i z powrotem tylko po to, by przynieść trochę wody. W architekturze jeziora danych ważnym czynnikiem jest bezpieczeństwo — podobnie, jak ważne jest bezpieczeństwo sieci wodociągowej. Dzięki centralizacji architektury przechowywania i dostarczania danych, prostsze staje się zapobieganie wykradaniu i fałszowaniu danych oraz monitorowanie takich prób. Oto kilka pytań, które trzeba sobie postawić, aby opracować system zapobiegania naruszeniom dostępu do danych:

- Czy przechowywane dane są zaszyfrowane? Jeśli tak, to kto ma klucze? Czy zdarzenia odszyfrowywania są rejestrowane i kontrolowane?

- Czy dane opuszczające sieć są rejestrowane i kontrolowane? Na przykład, czy można pozwolić na to, aby cała baza danych klientów kiedykolwiek została przeniesiona poza sieć? Dlaczego takie zdarzenie nie jest monitorowane i kontrolowane?
- Czy są przeprowadzane okresowe audyty bezpieczeństwa danych? Dlaczego nie?
- Czy przechowujesz dane osobowe w sieci? Po co?
- Masz systemy monitorowania kluczowych zdarzeń produkcyjnych. Czy monitorujesz zdarzenia związane z bezpieczeństwem danych? Dlaczego nie?

Dlaczego mielibyśmy kiedykolwiek pozwolić na to, aby dane wypłynęły poza wewnętrzną sieć? A gdyby tak zaprojektować kluczowe dane, aby były jak „kwadratowy korek", który nie może być przesyłany poza macierzystą sieć bez czegoś, co można nazwać „wybuchem jądrowym" danych? Uniemożliwienie przemieszczania danych poza środowisko wydaje się łatwym sposobem zapobiegania naruszeniom dostępu do danych. A gdyby tak sieć zewnętrzna mogła przesyłać wyłącznie pakiety, które są „okrągłymi korkami"? Taki system mógłby również być doskonałą „blokadą" dla dostawców chmury oferujących tego rodzaju bezpieczne jezioro danych.

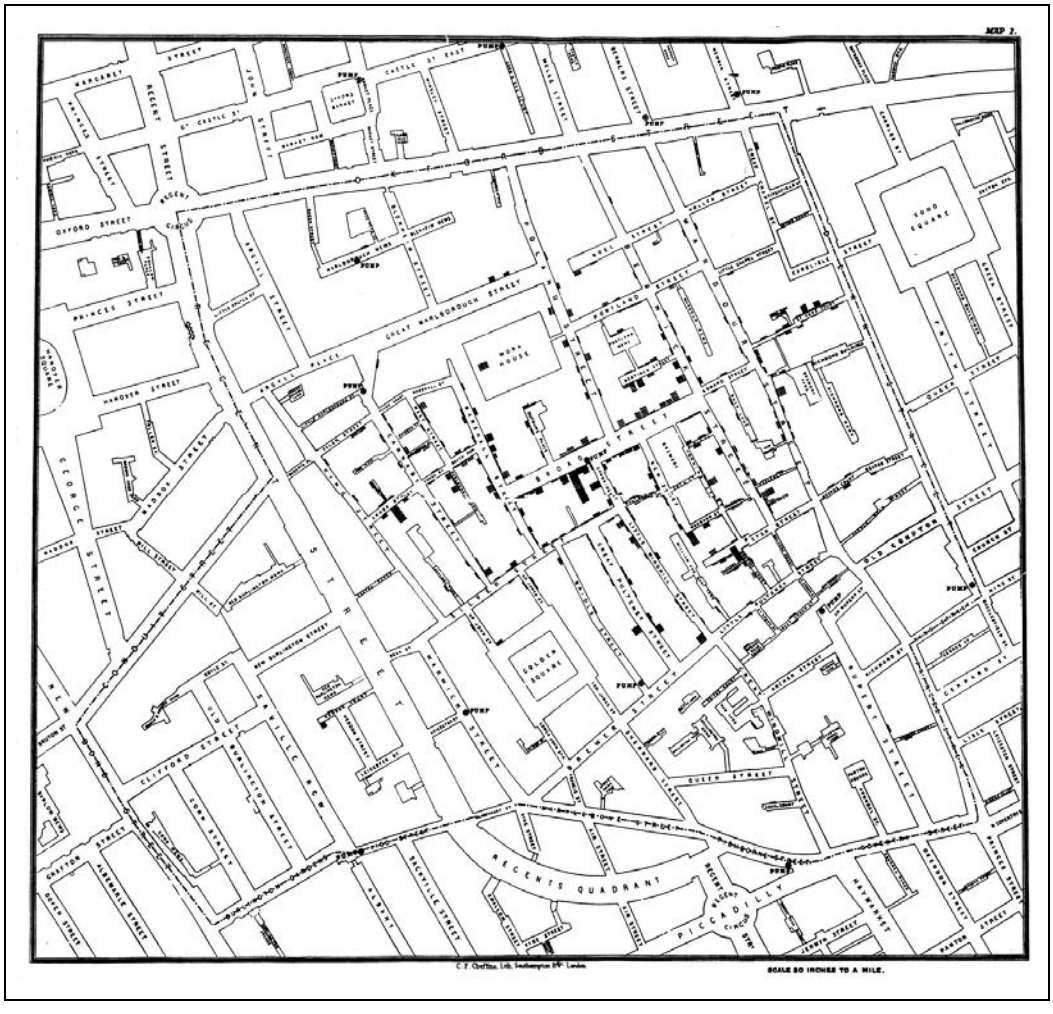
Poziom 3. Ciągłe dostawy oczyszczonych danych

Mam nadzieję, że Czytelnik jest przekonany co do słuszności koncepcji ciągłego dostarczania danych oraz zdaje sobie sprawę z tego, jak ważne ono jest dla powodzenia firmowych planów stosowania technik uczenia maszynowego. Ogromnym usprawnieniem mechanizmów ciągłego dostarczania danych jest ciągłe dostarczanie oczyszczonych danych. Po co zadawać sobie trud dostarczania danych, które są w kompletnym nieładzie? Przypomina to niedawny problem skażonej wody w mieście Flint, w stanie Michigan. Około 2014 roku, miasto Flint zmieniło źródło wody z jeziora Huron i rzeki Detroit na rzekę Flint. Nie zastosowano inhibitorów zanieczyszczeń, co spowodowało przedostanie się do instalacji wodociągowej związków ołowiu pochodzących ze starych rur. Możliwe jest również, że zmiana źródła wody spowodowała wybuch choroby legionistów, która zabiła 12 osób spośród 87, które na nią zachorowały.

Jednym z pierwszych skutecznych zastosowań inżynierii danych było przedsięwzięcie Johna Snowa z lat 1849 – 1854; udało się mu skorzystać z wizualizacji danych w celu zidentyfikowania ognisk cholery (rysunek 14.10). To doprowadziło do odkrycia przyczyny wybuchu epidemii. Okazało się, że ścieki były pompowane bezpośrednio do cieków zaopatrujących w wodę pitną!

Zwróć uwagę na następujące kwestie:

- Dlaczego dane nie są automatycznie przetwarzane w celu ich oczyszczenia?
- Czy potrafisz zwizualizować te części Twojego potoku danych, w których są „zanieczyszczenia"?
- Ile czasu w Twojej firmie poświęca się na związane z oczyszczaniem danych zadania, które w 100% można by zautomatyzować?



Rysunek 14.10. Skupiska przypadków cholery

Poziom 4. Ciągłe dostawy eksploracyjnych analiz danych

Jeśli inżynierię danych postrzegasz wyłącznie z perspektywy projektów Kaggle'a, może Ci się wydawać, że jedyny sens inżynierii danych polega na wygenerowaniu możliwie jak najdokładniejszej prognozy. Inżynieria danych i uczenie maszynowe to coś więcej niż tylko dokonywanie prognoz. Inżynieria danych jest dziedziną interdyscyplinarną, którą można postrzegać na kilka sposobów. Jedna z perspektyw to koncentracja na przyczynowości. Jakie ukryte funkcje sterują modelem? Czy potrafisz wyjaśnić, w jaki sposób model znajduje prognozy? W tym obszarze może pomóc kilka bibliotek Pythona: ELI5, SHAP i LIME. Celem działania każdej z nich jest pomoc w wyjaśnieniu, co robią modele uczenia maszynowego.

W przypadku prognoz mniej interesuje nas to, w jaki sposób została znaleziona odpowiedź, a bardziej to, czy prognoza jest dokładna. W macierzystym dla chmury świecie Big Data, takie podejście ma

swoje zalety. Do rozwiązywania pewnych problemów uczenia maszynowego — na przykład rozpoznawania obrazów z wykorzystaniem uczenia głębokiego — dobrze sprawdzają się duże ilości danych. Im więcej danych i im większa moc obliczeniowa do dyspozycji, tym większa dokładność prognozy.

Czy w systemach produkcyjnych zastosowałeś właśnie takie podejście? Dlaczego nie? Jeśli zbudujesz modele uczenia maszynowego, które nie będą wykorzystane, to po co je budować?

Czego nie wiesz? Czego możesz się dowiedzieć przyglądając się danym? W inżynierii danych często jesteśmy bardziej zainteresowani procesem niż wynikiem. Jeśli szukasz wyłącznie prognoz, to możesz nie zauważyć zupełnie innego sposobu, w jaki można postrzegać dane.

Poziom 5. Ciągłe dostarczania tradycyjnych narzędzi ML i AutoML

Zwalczanie automatyzacji jest tak stare, jak historia człowieka. Ruch luddystów była tajną organizacją angielskich robotników branży tekstylnej, którzy w latach 1811 – 1816 w ramach protestu niszczyli maszyny włókiennicze. Ostatecznie protestujący zostali ukarani, rebelia została stłumiona siłą, a branża włókiennicza wróciła na drogę rozwoju.

W historii ludzkości można zauważyć ciągły rozwój narzędzi automatyzujących zadania wykonywane niegdyś ręcznie przez ludzi. W wyniku technologicznego bezrobocia, niżej wykwalifikowani pracownicy są zwalniani, a pracownicy z lepszymi kwalifikacjami otrzymują zwiększone wynagrodzenie. Przykładem mogą być administratorzy systemów kontra specjaliści DevOps. Prawdą jest, że niektórzy administratorzy systemów stracili pracę — na przykład pracownicy wykonujący takie zadania, jak wymiana dysków twardych w centrach danych, ale powstały nowe, lepiej płatne miejsca pracy, takie jak architekci rozwiązań bazujących na chmurze.

Niczym niezwykłym są oferty pracy dla specjalistów w dziedzinie uczenia maszynowego i inżynierii danych, z roczną pensją na poziomie od trzystu tysięcy do jednego miliona dolarów. Dodatkowo zadania te często zawierają wiele składników, które w istocie polegają na stosowaniu reguł biznesowych — dostrajanie hiperparametrów, usuwanie wartości pustych i dystrybucja zadań do klastra. Prawo Automatyzatora (które sam wymyśliłem) mówi: „Jeśli uważasz, że coś mogłoby być zautomatyzowane, to w końcu to zostanie zautomatyzowane. Obecnie wiele się mówi na temat narzędzi AutoML, więc jest nieuniknione, że duża część zadań związanych z uczeniem maszynowym będzie zautomatyzowana.

W związku z tym — podobnie jak w przypadku innych przykładów automatyzacji — charakter pracy ulegnie zmianie. Niektóre stanowiska będą wymagały jeszcze większych kwalifikacji (wyobraźmy sobie osobę, która potrafi wyszkolić wiele tysięcy modeli uczenia maszynowego dziennie), a niektóre zadania (zajęcie osób, które poprawiają wartości w plikach o strukturze JSON — tzn. zajmują się dostrajaniem hiperparametrów), zostaną zautomatyzowane, ponieważ maszyny będą zdolne wykonać je znacznie lepiej

Poziom 6. Operacyjna pętla sprzężenia zwrotnego narzędzi ML

Po co rozwijać aplikacje mobilne? Przypuszczalnie po to, by z Twojej aplikacji mogli skorzystać użytkownicy urządzeń mobilnych. A co z uczeniem maszynowym? Sens uczenia maszynowego, zwłaszcza w porównaniu z inżynierią danych lub statystyką, polega na stworzeniu modelu i znalezieniu prognoz. Jeśli model nie jest używany w produkcji, to do czego on służy?

Dodatkowo wdrożenie modelu do produkcji jest okazją, aby nauczyć się więcej. Czy model potrafi dokładnie przewidzieć wynik, kiedy zostanie wdrożony w środowisku, w którym otrzyma nowe dane? Czy model wywiera oczekiwany wpływ na użytkowników: to znaczy powoduje zwiększenie zakupów lub dłuższe przebywanie w witrynie? Te cenne informacje można uzyskać tylko wtedy, gdy model zostanie właściwie wdrożony w środowisku produkcyjnym.

Innymi ważnymi zagadnieniami są skalowalność i powtarzalność. Organizacja w pełni dojrzała pod kątem technologii potrafi wdrażać oprogramowanie, w tym modele uczenia maszynowego na żądanie. Dla modeli ML wymagane jest stosowanie najlepszych praktyk DevOps: ciągłego wdrażania, mikrousług, narzędzi monitorowania oraz instrumentacji.

Prostym sposobem na doprowadzenie do takiej dojrzałości technologicznej organizacji jest zastosowanie tej samej logiki, jaką stosujemy w przypadku rozstrzygnięć co do stosowania przetwarzania w chmurze zamiast korzystania z fizycznego ośrodka przetwarzania danych. Lepiej „wynająć" wiedzę od innych i skorzystać z zasad ekonomii w przedsięwzięciach o dużej skali.

Model Sklearn Flask z wykorzystaniem systemów Kubernetes i Docker

Spróbujmy prześledzić praktyczne wdrożenie modelu bazującego na `sklearn` z wykorzystaniem Dockera i Kubernetesa.

Poniżej zamieszczono zawartość pliku *Dockerfile*. Zwróć uwagę, że serwujemy aplikację Flask. Aplikacja Flask będzie hostem dla aplikacji `sklearn`. Możemy zainstalować Hadolint, który pozwala na sprawdzenie poprawności pliku *Dockerfile*: *https://github.com/hadolint/hadolint*.

```
FROM python:3.7.3-stretch

# Katalog roboczy
WORKDIR /app

# Skopiowanie kodu źródłowego do katalogu roboczego
COPY . app.py /app/
# Instalacja pakietów wymienionych w pliku requirements.txt
# hadolint ignore=DL3013
RUN pip install --upgrade pip &&\
    pip install --trusted-host pypi.python.org -r requirements.txt

# Udostępnienie aplikacji przez port 80
EXPOSE 80

# Uruchomienie aplikacji przy uruchomieniu kontenera
CMD ["python", "app.py"]
```

Oto plik *Makefile*, który spełnia rolę centralnego punktu podczas wykonywania aplikacji:

```
setup:
    python3 -m venv ~/.python-devops
```

```
install:
    pip install --upgrade pip &&\
        pip install -r requirements.txt

test:
    #python -m pytest -vv --cov=myrepolib tests/*.py
    #python -m pytest --nbval notebook.ipynb

lint:
    hadolint Dockerfile
    pylint --disable=R,C,W1203 app.py
all: install lint test
```

Tak wygląda plik *requirements.txt*:

```
Flask==1.0.2
pandas==0.24.2
scikit-learn==0.20.3
```

A to zawartość pliku *app.py*:

```
from flask import Flask, request, jsonify
from flask.logging import create_logger
import logging

import pandas as pd
from sklearn.externals import joblib
from sklearn.preprocessing import StandardScaler

app = Flask(__name__)
LOG = create_logger(app)
LOG.setLevel(logging.INFO)

def scale(payload):
    """Skaluje ładunek danych """

    LOG.info(f"Skalowanie ładunku danych: {payload}")
    scaler = StandardScaler().fit(payload)
    scaled_adhoc_predict = scaler.transform(payload)
    return scaled_adhoc_predict

@app.route("/")
def home():
    html = "<h3>Sklearn Prediction Home</h3>"
    return html.format(format)

# TO DO: Zapisz do logu prognozowaną wartość
@app.route("/predict", methods=['POST'])
def predict():
    """Wykonuje prognozę sklearn

    input looks like:
            {
    "CHAS":{
        "0":0
    },
```

```
        "RM":{
            "0":6.575
        },
        "TAX":{
            "0":296.0
        },
        "PTRATIO":{
            "0":15.3
        },
        "B":{
            "0":396.9
        },
        "LSTAT":{
            "0":4.98
        }
        result looks like:
        { "prediction": [ 20.35373177134412 ] }
        """
        json_payload = request.json
        LOG.info(f"Dane JSON: {json_payload}")
        inference_payload = pd.DataFrame(json_payload)
        LOG.info(f"DataFrame z prognozowanymi danymi: {inference_payload}")
        scaled_payload = scale(inference_payload)
        prediction = list(clf.predict(scaled_payload))
        return jsonify({'prediction': prediction})
    if __name__ == "__main__":
        clf = joblib.load("boston_housing_prediction.joblib")
        app.run(host='0.0.0.0', port=80, debug=True)
```

Oto zawartość pliku *run_docker.sh*:

```
#!/usr/bin/env bash

# Budowanie obrazu
docker build --tag=flasksklearn .

# Wyświetlenie listy obrazów Dockera
docker image ls

# Uruchomienie aplikacji Flask
docker run -p 8000:80 flasksklearn
```

To zawartość pliku *run_kubernetes.sh*:

```
#!/usr/bin/env bash

dockerpath="noahgift/flasksklearn"

# Uruchom w kontenerze Docker Hub z wykorzystaniem systemu Kubernetes
kubectl run flaskskearlndemo\
    --generator=run-pod/v1\
    --image=$dockerpath\
    --port=80 --labels app=flaskskearlndemo

# Wyświetlenie listy strąków Kubernetesa
kubectl get pods
```

```
# Przekierowanie portu kontenera do hosta
kubectl port-forward flaskskearlndemo 8000:80
```

```
#!/usr/bin/env bash
# Znakowanie i przesłanie obrazu do rejestru Docker Hub

#Zakłada się, że obraz jest zbudowany
#docker build --tag=flasksklearn

dockerpath="noahgift/flasksklearn"
# Uwierzytelnianie i tagowanie
echo "Docker ID and Image: $dockerpath"
docker login &&\
    docker image tag flasksklearn $dockerpath


# Przesłanie obrazu
docker image push $dockerpath
```

Sklearn Flask z wykorzystaniem Kubernetesa i Dockera

Być może interesuje Cię, w jaki sposób model został stworzony, a następnie zserializowany. Cały notatnik jest dostępny pod tym adresem: *https://oreil.ly/_pHz-*.

Najpierw należy zaimportować kilka bibliotek dostarczających funkcje uczenia maszynowego:

```
import numpy
from numpy import arange
from matplotlib import pyplot
import seaborn as sns
import pandas as pd
from pandas import read_csv
from pandas import set_option
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

`In[0]:`

```
    boston_housing = "https://raw.githubusercontent.com/\
    noahgift/boston_housing_pickle/master/housing.csv"
    names = ['CRIM', 'ZN', 'INDUS', 'CHAS',
```

```
    'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
      'PTRATIO', 'B', 'LSTAT', 'MEDV']
    df = read_csv(boston_housing,
      delim_whitespace=True, names=names)
```

In[0]:

```
df.head()
```

Out[0]:

```
  CRIM    ZN   INDUS CHAS NOX   RM    AGE
0 0.00632 18.0 2.31  0    0.538 6.575 65.2
1 0.02731 0.0  7.07  0    0.469 6.421 78.9
2 0.02729 0.0  7.07  0    0.469 7.185 61.1
3 0.03237 0.0  2.18  0    0.458 6.998 45.8
4 0.06905 0.0  2.18  0    0.458 7.147 54.2

  DIS    RAD TAX   PTRATIO B      LSTAT MEDV
0 4.0900 1   296.0 15.3    396.90 4.98  24.0
1 4.9671 2   242.0 17.8    396.90 9.14  21.6
2 4.9671 2   242.0 17.8    392.83 4.03  34.7
3 6.0622 3   222.0 18.7    394.63 2.94  33.4
4 6.0622 3   222.0 18.7    396.90 5.33  36.2
```

EDA

Oto cechy modelu:

CHAS

Sztuczna zmienna dotycząca rzeki Charles (1, jeśli ścieżka obejmuje rzekę, w przeciwnym przypadku 0).

RM

Przeciętna liczba pokojów w mieszkaniu.

TAX

Pełna wartość stawki podatku od nieruchomości za wartość równą 10 000 dolarów.

PTRATIO

Liczba uczniów w stosunku do liczby nauczycieli dla miasta.

Bk

Proporcja czarnoskórej ludności w mieście w stosunku do ogółu mieszkańców.

LSTAT

Procent ludności o niższym statusie.

MEDV

Mediana wartości (w tysiącach dolarów) mieszkań zajmowanych przez właścicieli.

In[0]:

```
prices = df['MEDV']
```

```
df = df.drop(['CRIM','ZN','INDUS','NOX','AGE','DIS','RAD'], axis = 1)
features = df.drop('MEDV', axis = 1)
df.head()
```

Out[0]:

	CHAS	RM	TAX	PTRATIO	B	LSTAT	MEDV
0	0	6.575	296.0	15.3	396.90	4.98	24.0
1	0	6.421	242.0	17.8	396.90	9.14	21.6
2	0	7.185	242.0	17.8	392.83	4.03	34.7
3	0	6.998	222.0	18.7	394.63	2.94	33.4
4	0	7.147	222.0	18.7	396.90	5.33	36.2

Modelowanie

W tym miejscu w notatniku wykonywane jest modelowanie. Jedną z przydatnych strategii jest stworzenie w notatniku czterech głównych sekcji:

- Pobieranie danych.
- EDA.
- Modelowanie.
- Wnioski.

W sekcji modelowania wyodrębniamy dane z obiektu `DataFrame` i przekazujemy do modułu `sklearn` `train_test_split`, który realizuje zadania podziału danych na dane szkoleniowe i testowe.

Podział danych

In[0]:

```
# Podział testowego zestawu danych
array = df.values
X = array[:,0:6]
Y = array[:,6]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,
  test_size=validation_size, random_state=seed)
```

In[0]:

```
for sample in list(X_validation)[0:2]:
    print(f"X_validation {sample}")
```

Out[0]:

```
X_validation [ 1. 6.395 666. 20.2 391.34 13.27 ]
X_validation [ 0. 5.895 224. 20.2 394.81 10.56 ]
```

Dostrajanie skalowanego algorytmu GBM

Ten model wykorzystuje kilka zaawansowanych technik, które zastosowano w wielu udanych projektach Kaggle'a. Można do nich zaliczyć wyszukiwanie siatkowe (ang. *GridSearch*), które może pomóc znaleźć optymalne hiperparametry. Należy również zwrócić uwagę na wykonywane skalowanie danych. Większość algorytmów uczenia maszynowego, w celu utworzenia dokładnych prognoz, oczekuje pewnego rodzaju skalowania.

In[0]:

```
# Opcje testowe i ewaluacja metryk przy użyciu metody błędu średniokwadratowego
num_folds = 10
seed = 7
RMS = 'neg_mean_squared_error'
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = dict(n_estimators=numpy.array([50,100,150,200,250,300,350,400]))
model = GradientBoostingRegressor(random_state=seed)
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=RMS, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
print("%f (%f) with: %r" % (mean, stdev, param))
```

Out[0]:

```
Best: -11.830068 using {'n_estimators': 200}
-12.479635 (6.348297) with: {'n_estimators': 50}
-12.102737 (6.441597) with: {'n_estimators': 100}
-11.843649 (6.631569) with: {'n_estimators': 150}
-11.830068 (6.559724) with: {'n_estimators': 200}
-11.879805 (6.512414) with: {'n_estimators': 250}
-11.895362 (6.487726) with: {'n_estimators': 300}
-12.008611 (6.468623) with: {'n_estimators': 350}
-12.053759 (6.453899) with: {'n_estimators': 400}

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:841:
DeprecationWarning:
DeprecationWarning)
```

Dopasowywanie modelu

Ten model jest dopasowywany przy użyciu algorytmu `GradientBoostingRegressor`. Ostatnim krokiem po wyszkoleniu modelu jest jego dopasowanie i sprawdzenie, czy nie wystąpiły błędy. Do tego celu stosujemy podzbiór danych, który wcześniej został wydzielony ze zbioru głównego.

In[0]:

```
# przygotowanie modelu
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = GradientBoostingRegressor(random_state=seed, n_estimators=400)
model.fit(rescaledX, Y_train)

# przekształcenie walidacyjnego zestawu danych
rescaledValidationX = scaler.transform(X_validation)
predictions = model.predict(rescaledValidationX)
```

```
    print("Błąd średniokwadratowy: \n")
    print(mean_squared_error(Y_validation, predictions))
```

Out[0]:

```
    Błąd średniokwadratowy:
    26.326748591395717
```

Ocena

Jednym z trudniejszych aspektów uczenia maszynowego jest ocena modelu. W tym przykładzie pokazano, jak do tej samej ramki `DataFrame` można dodać prognozowaną i rzeczywistą wycenę domu. Do znalezienia różnic można wykorzystać następującą ramkę `DataFrame`:

In[0]:

```
    predictions=predictions.astype(int)
    evaluate = pd.DataFrame({
            "Rzeczywista cena domu": Y_validation,
            "Prognozowana cena domu": predictions
    })
evaluate["difference"] = evaluate["Rzeczywista ocena domu"]-evaluate["Prognozowana cena
↪domu"] evaluate.head()
```

Oto znalezione różnice.

Out[0]:

```
    Rzeczywista cena domu Prognozowana cena domu  Różnica
0   21.7                  21                       0.7
1   18.5                  19                      -0.5
2   22.2                  20                      2.2
3   20.4                  19                      1.4
4   8.8                   9                       -0.2
```

Za pomocą metody `describe` z biblioteki Pandas można obejrzeć rozkład danych.

In[0]:

```
    evaluate.describe()
```

Out[0]:

```
       Rzeczywista cena domu Prognozowana cena domu  Różnica
count  102.000000            102.000000              102.000000
mean   22.573529             22.117647               0.455882
std    9.033622              8.758921                5.154438
min    6.300000              8.000000                -34.100000
25%    17.350000             17.000000               -0.800000
50%    21.800000             20.500000               0.600000
75%    24.800000             25.000000               2.200000
max    50.000000             56.000000               22.000000
```

adhoc_predict

Spróbujmy przetestować ten model przewidywania, aby zobaczyć, jaki będzie przepływ pracy po deserializacji. Podczas tworzenia webowego interfejsu API dla modelu uczenia maszynowego warto przetestować te sekcje kodu, które interfejs API będzie wykonywać w samym notatniku.

O wiele łatwiej jest tworzyć i debugować funkcje w notatniku niż próbować tworzyć prawidłowe funkcje wewnątrz aplikacji webowej.

In[0]:

```
actual_sample = df.head(1)
actual_sample
```

Out[0]:

```
  CHAS RM     TAX    PRATIO  B      LSTAT  MEDV
0 0    6.575  296.0  15.3    396.9  4.98   24.0
```

In[0]:

```
adhoc_predict = actual_sample[["CHAS", "RM", "TAX", "PTRATIO", "B", "LSTAT"]]
 adhoc_predict.head()
```

Out[0]:

```
   CHAS  RM     TAX    PTRATIO B      LSTAT
0  0     6.575  296.0  15.3    396.9  4.98
```

Przepływ pracy JSON

To jest sekcja notatnika, która przydaje się do debugowania aplikacji Flask. Jak wspomniano wcześniej, o wiele prostsze niż w aplikacji webowej jest utworzenie kodu API wewnątrz projektu uczenia maszynowego, sprawdzenie, czy wszystko działa, a następnie przeniesienie tego kodu do skryptu. Alternatywą jest próba stworzenia poprawnego kodu w projekcie oprogramowania, które nie ma takich interaktywnych narzędzi, jakie zapewnia środowisko Jupyter.

In[0]:

```
json_payload = adhoc_predict.to_json()
json_payload
```

Out[0]:

```
{"CHAS":{"0":0},"RM":
{"0":6.575},"TAX":
{"0":296.0},"PTRATIO":
{"0":15.3},"B":{"0":396.9},"LSTAT":
{"0":4.98}}
```

Skalowanie danych wejściowych

Aby można było prognozować dane, trzeba je ponownie skalować. Ten przepływ pracy także powinien zostać wykonany w notatniku. To lepsze podejście w porównaniu z próbą przeprowadzenia go w aplikacji webowej, gdzie debugowanie jest znacznie trudniejsze. W poniższej sekcji przedstawiono kod, który rozwiązuje tę część potoku prognozowania w procesie uczenia maszynowego. Następnie można go użyć do utworzenia funkcji w aplikacji Flask.

In[0]:

```
scaler = StandardScaler().fit(adhoc_predict)
```

```
    scaled_adhoc_predict = scaler.transform(adhoc_predict)
    scaled_adhoc_predict
```

Out[0]:

```
    array([[0., 0., 0., 0., 0., 0.]])
```

In[0]:

```
    list(model.predict(scaled_adhoc_predict))
```

Out[0]:

```
    [20.35373177134412]
```

Serializacja sklearn

Teraz wyeksportujemy model.

In[0]:

```
    from sklearn.externals import joblib
```

In[0]:

```
    joblib.dump(model, 'boston_housing_prediction.joblib')
```

Out[0]:

```
    ['boston_housing_prediction.joblib']
```

In[0]:

```
    !ls -l
```

Out[0]:

```
    total 672
    -rw-r--r-- 1 root root 681425 May 5 00:35 boston_housing_prediction.joblib
    drwxr-xr-x 1 root root 4096 Apr 29 16:32 sample_data
```

Deserializacja i prognozowanie

In[0]:

```
    clf = joblib.load('boston_housing_prediction.joblib')
```

adhoc_predict z modułu Pickle

In[0]:

```
    actual_sample2 = df.head(5)
    actual_sample2
```

Out[0]:

```
  CHAS  RM     TAX    PTRATIO B       LSTAT  MEDV
0 0     6.575  296.0  15.3    396.90  4.98   24.0
1 0     6.421  242.0  17.8    396.90  9.14   21.6
2 0     7.185  242.0  17.8    392.83  4.03   34.7
3 0     6.998  222.0  18.7    394.63  2.94   33.4
4 0     7.147  222.0  18.7    396.90  5.33   36.2
```

In[0]:

```
adhoc_predict2 = actual_sample[["CHAS", "RM", "TAX", "PTRATIO", "B", "LSTAT"]]
adhoc_predict2.head()
```

Out[0]:

```
   CHAS  RM     TAX    PTRATIO  B     LSTAT
0  0     6.575  296.0  15.3     396.9 4.98
```

Skalowanie danych wejściowych

In[0]:

```
scaler = StandardScaler().fit(adhoc_predict2)
scaled_adhoc_predict2 = scaler.transform(adhoc_predict2)
scaled_adhoc_predict2
```

Out[0]:

```
array([[0., 0., 0., 0., 0., 0.]])
```

In[0]:

```
# Użycie zserializowanego modelu
list(clf.predict(scaled_adhoc_predict2))
```

Out[0]:

```
[20.35373177134412]
```

Na koniec zserializowany model jest ładowany ponownie i testowany na rzeczywistym zestawie danych.

Ćwiczenia

- Jakie są kluczowe różnice między frameworkami scikit-learn i PyTorch?
- Co to jest AutoML i dlaczego warto używać tej techniki?
- Zmień model scikit-learn w taki sposób, aby na podstawie wzrostu przewidywać wagę.
- Uruchom przykład z wykorzystaniem frameworka PyTorch w notatnikach Google Colab i przełączaj się pomiędzy środowiskami uruchomieniowymi bazującymi na CPU i GPU. Wyjaśnij różnicę w wydajności, jeśli taka istnieje.
- Czym są analizy EDA i dlaczego są tak ważne w projekcie inżynierii danych?

Zadanie związane ze studium przypadku

- Korzystając z przykładu pokazanego w tym rozdziale, przejdź do witryny Kaggle, pobierz popularny notatnik w Pythonie i przekształć go na aplikację Flask w kontenerze, serwującą prognozy. Następnie zainstaluj aplikację w środowisku chmury obliczeniowej za pośrednictwem hostowanych usług Kubernetesa, takich jak Amazon EKS.

Pytania i zadania kontrolne

- Opisz różne rodzaje frameworków i ekosystemów uczenia maszynowego.
- Uruchom i zdebuguj istniejący projekt uczenia maszynowego, zaimplementowany za pomocą frameworków scikit-learn i PyTorch.
- Umieść w kontenerze model z aplikacją Flask bazującą na scikit-learn.
- Opisz produkcyjny model dojrzałości uczenia maszynowego.

Python: tutaj ważna jest prawdziwa nowoczesność oprogramowania!

Ostatnia dekada zmieniła oblicze IT. Kluczowego znaczenia nabrały big data, a chmura i automatyzacja rozpowszechniły się wszędzie tam, gdzie mowa o efektywności. Inżynierowie muszą wykorzystywać zalety systemów linuksowych w codziennej praktyce, aby zapewnić należyty poziom automatyzacji swoich zadań. Do tych celów świetnie nadaje się Python. Język ten zdobywa coraz większe uznanie z uwagi na jego wszechstronność, jak również wydajność, przenaszalność i bezpieczeństwo kodu. Warto więc wykorzystywać Pythona do administrowania systemami Linux wraz z takimi narzędziami DevOps jak Docker, Kubernetes i Terraform.

Dzięki tej książce dowiesz się, jak sobie z tym poradzić. Znalazło się w niej krótkie wprowadzenie do Pythona oraz do automatyzacji przetwarzania tekstu i obsługi systemu plików, a także do pisania własnych narzędzi wiersza poleceń. Zaprezentowano również przydatne narzędzia linuksowe, systemy zarządzania pakietami oraz systemy budowania, monitorowania i automatycznego testowania kodu. Zagadnienia te szczególnie zainteresują specjalistów DevOps. Ponadto zawarto tu podstawowe informacje o chmurze obliczeniowej, usługach IaC i systemach Kubernetes. Omówiono zasady uczenia maszynowego i inżynierii danych z perspektywy DevOps. Przedstawiono także kompletny przewodnik po procesach budowania, wdrażania oraz operacyjnego wykorzystywania modelu uczenia maszynowego z użyciem systemów Flask, sklearn, Docker i Kubernetes.

W książce między innymi:

- wprowadzenie do Pythona
- automatyczne przetwarzanie tekstu oraz automatyzacja operacji na plikach
- automatyzacja za pomocą sprawdzonych narzędzi linuksowych
- chmura, infrastruktura jako kod, Kubernetes i tryb bezserwerowy
- uczenie maszynowe i inżynieria danych z perspektywy DevOps
- tworzenie i operacjonalizacja projektu uczenia maszynowego

Noah Gift jest wykładowcą na uniwersytetach Northwestern i Duke'a. Prowadzi zajęcia z inżynierii danych oraz informatyki.
Kennedy Behrman jest doświadczonym konsultantem specjalizującym się w tworzeniu architektury i implementacji rozwiązań w chmurze dla start-upów.
Alfredo Deza jest inżynierem oprogramowania i programistą *open source*.
Grig Gheorghiu jest kierownikiem laboratorium badawczego, architektem systemów, sieci i zabezpieczeń oraz inżynierem testów kodu.

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

ISBN 978-83-283-6830-9
9 788328 368309
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 89,00 zł