Technologia i rozwiązania

Zaawansowane uczenie maszynowe z językiem Python

John Hearty

Helion

[PACKT] PUBLISHING open source community experience distilled

Tytuł oryginału: Advanced Machine Learning with Python

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-3607-0

Wydawnictwo HELION
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/zaaucz.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Nienadzorowane uczenie maszynowe

W tym rozdziale opiszę zagadnienia związane ze stosowaniem technik nienadzorowanego uczenia maszynowego w celu identyfikacji wzorców i struktur w zbiorach danych.

Techniki uczenia nienadzorowanego to doskonałe narzędzia służące do eksploracji danych. Pozwalają na wykrycie wzorców i struktur zawartych w zbiorach danych — informacje te mogą okazać się przydatne same w sobie lub pomóc w dalszej analizie zbioru. Warto dysponować zestawem konkretnych narzędzi nienadzorowanego uczenia maszynowego, które pozwolą na wyciągnięcie przydatnych informacji z nieznanych lub złożonych zbiorów danych.

Na początku opiszę **analizę głównych składowych** (ang. *Principal Component Analysis*, PCA) — podstawową technikę przetwarzania danych stosowaną w celu zredukowania liczby wymiarów i zakresu. Później zajmę się **grupowaniem metodą *k*-średnich** — popularną i w miarę prostą techniką uczenia nienadzorowanego. Na koniec opiszę zagadnienia związane z **sieciami Kohonena** (ang. *Self-Organizing Map*, SOM) — metodą topologicznego grupowania, która pozwala na przedstawienie skomplikowanych zbiorów danych za pomocą dwóch wymiarów.

W trakcie lektury tego rozdziału dowiesz się, jak efektywnie zastosować wymienione wcześniej techniki podczas pracy z wielowymiarowymi zbiorami danych. Praktyczne zastosowanie każdego z algorytmów zostanie pokazane na podstawie zbioru obrazów ręcznie pisanych cyfr *UCI Handwritten Digits*. Poza zapoznaniem się z opisem zastosowań wspomnianych technik poznasz je od strony praktycznej, a także uzyskasz odpowiedzi na metodologiczne pytania dotyczące między innymi kalibracji, walidacji i oceny wydajności poszczególnych technik. Reasumując, poruszę kolejno następujące tematy:

- analiza głównych składowych,
- grupowanie metodą *k*-średnich,
- sieci Kohonena.

Analiza głównych składowych (PCA)

Wydajna praca z wielowymiarowymi zbiorami danych wymaga opanowania zestawu technik pozwalających na redukcję liczby wymiarów do poziomu, przy którym dane da się analizować. Redukcja liczby wymiarów pozwala na przedstawienie wielowymiarowych danych na dwuwymiarowych wykresach, ujęcie najważniejszych informacji zawartych w danych za pomocą jak najmniejszej liczby cech, a także identyfikację współliniowych komponentów modelu.

> Dla przypomnienia — współliniowość w kontekście uczenia maszynowego oznacza występowanie w modelu cech, pomiędzy którymi jest zależność liniowa. To chyba oczywiste, że obecność takich cech utrudnia analizę danych — cechy zależne od siebie mogą wprowadzić analityka w błąd (cech zależnych od siebie nie można traktować tak, jakby każda z nich dostarczała tak samo ważnej informacji), a dodatkowo mogą wskazywać na występowanie lokalnych wartości minimalnych tam, gdzie w rzeczywistości ich nie ma.

Prawdopodobnie obecnie najczęściej stosowaną metodą redukcji wymiarów jest analiza głównych składowych. W dalszej części książki będziesz wielokrotnie korzystać z tej metody, a więc warto, abyś dobrze się z nią zapoznał, zrozumiał teoretyczne podstawy jej działania, a także napisał kod implementujący ją w języku Python.

Podstawy analizy głównych składowych

Analiza głównych składowych jest wszechstronną techniką dekompozycji umożliwiającą podział wielowymiarowych zbiorów danych na zestaw niezależnych komponentów. Analiza odpowiedniej liczby takich komponentów może pozwolić na opisanie wyjaśnienia wszystkich wymiarów zbioru danych. Ponadto komponenty te tworzą skrócony opis zbioru danych. Analiza głównych składowych ma wiele zastosowań, a jej uniwersalność sprawia, że warto poświęcić czas na opanowanie związanych z nią zagadnień.

> Zwróć uwagę na to, że zmniejszenie liczby zmiennych wchodzących w skład zbioru danych prawie zawsze wiąże się z utratą części informacji zawartych w zbiorze wejściowym. Jeżeli liczba komponentów jest wystarczająca, to utrata ta jest minimalna, ale w przypadku zbiorów charakteryzujących się małą liczbą głównych składowych utworzonych na podstawie zbiorów o bardzo dużej liczbie wymiarów może dojść do znacznej utraty informacji. Stosując technikę analizy głównych składowych, musisz zawsze myśleć o tym, ile komponentów jest niezbędnych do efektywnego modelowania przetwarzanego zbioru danych.

Metoda analizy głównych składowych identyfikuje kolejne osie zbioru danych o najwyższej wariancji (główne składowe). Operacja ta jest wykonywana w następujących krokach:

1. Identyfikacja środkowego punktu zbioru danych.
2. Obliczenie macierzy kowariancji danych.
3. Obliczenie wektorów własnych macierzy kowariancji.

4. Ortonormalizacja wektorów własnych.
5. Obliczenie proporcji wariancji przedstawianej przez każdy wektor własny.

Czas zdefiniować niektóre ważne pojęcia:

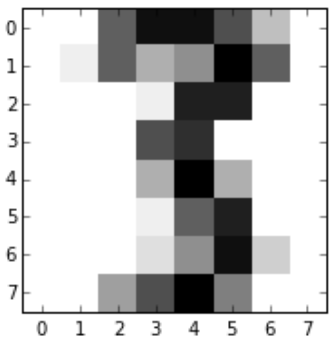
- **Kowariancja** jest efektywną wariancją wielu wymiarów — wariancją pomiędzy przynajmniej dwiema zmiennymi. Pojedyncza wartość może opisywać wariancję jednego wymiaru lub jednej zmiennej, ale w celu przedstawienia kowariancji pomiędzy dwiema zmiennymi niezbędne jest użycie macierzy 2×2, a w celu przedstawienia kowariancji pomiędzy trzema zmiennymi należy użyć macierzy 3×3 itd. W związku z tym pierwszym krokiem algorytmu analizy głównych składowych jest obliczenie macierzy kowariancji.
- **Wektor własny** to wektor, który jest charakterystyczny dla zbioru danych i transformacji liniowej. Wektor ten ma taki sam kierunek przed wykonaniem transformacji, jak po jej wykonaniu. Aby zrozumieć działanie tego wektora, wyobraź sobie, że rozciągasz pomiędzy swoimi dłońmi gumową opaskę. Załóżmy, że rozciągasz ją do momentu napięcia. Wektorem własnym jest wektor, którego kierunek był taki sam przed rozciągnięciem opaski, jak w trakcie jej rozciągania. W tym przypadku wektor ten będzie przechodził bezpośrednio przez środek gumowej opaski pomiędzy Twoimi dłońmi.
- **Ortogonalizacja** jest procesem polegającym na poszukiwaniu dwóch wektorów, które są ortogonalne (ustawione względem siebie pod kątem prostym). W przypadku *n*-wymiarowej przestrzeni danych proces ortogonalizacji polega na przetwarzaniu grupy wektorów i generuje zestaw wektorów ortogonalnych.
- **Ortonormalizacja** jest procesem ortogonalizacji, który normalizuje również iloczyn.
- **Wartość własna** (z grubsza odpowiada długości wektora własnego) jest używana do obliczenia proporcji wariancji reprezentowanej przez każdy wektor własny. Operacja ta polega na podzieleniu wartości własnej poszczególnych wektorów własnych przez sumę wartości własnych wszystkich wektorów własnych.

Podsumowując, macierz kowariancji jest używana w celu obliczenia wektorów własnych. Proces ortonormalizacji jest przeprowadzany w celu uzyskania znormalizowanych wektorów ortogonalnych na podstawie wektorów własnych. Wektor własny o najwyższej wartości własnej jest pierwszą główną składową. Kolejne składowe charakteryzują się niższymi wartościami własnymi. To w ten sposób algorytm analizy głównych składowych pomaga w przetworzeniu wejściowego zbioru danych na zbiór o mniejszej liczbie wymiarów.

Stosowanie algorytmu analizy głównych składowych

Poznałeś już działanie algorytmu analizy głównych składowych na wysokim poziomie. Teraz przeskoczę bezpośrednio do zastosowania go w analizie jednego z najciekawszych zbiorów danych — zbioru obrazów ręcznie zapisanych cyfr — `digits`, który może zostać pobrany razem z pakietem *scikit-learn*.

Zbiór ten składa się z 1797 przykładów cyfr zapisanych ręcznie przez 44 różne osoby. Nacisk długopisu osoby zapisującej cyfry został przedstawiony na siatce 8×8. W wyniku takiej operacji kwantyzacji otrzymano np. następującą mapę:



Mapy te mogą zostać zamienione na wektory cech o długości równej 64, które można stosować w roli danych wejściowych analizatora. Gdy widzi się zbiór danych wejściowych o 64 cechach, od razu chce się zastosować technikę redukcji liczby zmiennych, taką jak algorytm analizy głównych składowych. Tak duża liczba cech uniemożliwia analizę zbioru danych za pomocą wizualnych technik eksploracji danych!

Poniższy kod przetworzy zbiór danych reprezentujących napisane ręcznie cyfry (`digits`) za pomocą metody analizy głównych składowych:

```
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.cm as cm

digits = load_digits()
data = digits.data

n_samples, n_features = data.shape
n_digits = len(np.unique(digits.target))
labels = digits.target
```

Przedstawiony kod wykonuje kilka operacji:

1. Najpierw ładuje niezbędne biblioteki: bibliotekę *numpy*, komponenty pakietu *scikit-learn* (w tym sam zbiór danych `digits`, funkcję analizy głównych składowych i funkcję skalowania danych), a także bibliotekę *matplotlib* służącą do tworzenia wykresów.

2. Następnie rozpoczyna przygotowanie zbioru danych `digits`. Operacja ta składa się z kilku następujących po sobie czynności:
   - Operacja ładowania zbioru danych jest przeprowadzana przed utworzeniem zmiennych pomocniczych.
   - Tworzona jest zmienna `data`, z której kod będzie korzystać później. Ponadto tworzona jest zmienna `digits`, która definiuje liczbę cyfr w docelowym wektorze `target` (`n_digits` = 10, ponieważ będziemy przetwarzać cyfry z zakresu od 0 do 9). Utworzenie tej zmiennej ułatwi proces dalszej analizy.
   - Zapisany zostaje wektor `target`, który później będzie używany jako źródło etykiet.
   - Cały ten proces tworzenia zmiennych ma na celu uproszczenie dalszej analizy.
3. Po przygotowaniu zbioru danych inicjowany jest algorytm analizy głównych składowych:

```
pca = PCA(n_components=10)
data_r = pca.fit(data).transform(data)

print('współczynnik wyjaśnionych wariancji (10 pierwszych składowych): %s' %str
➥(pca.explained_variance_ratio_))
print('suma wyjaśnionych wariancji (10 pierwszych składowych): %s' %str
➥(sum(pca.explained_variance_ratio_)))
```

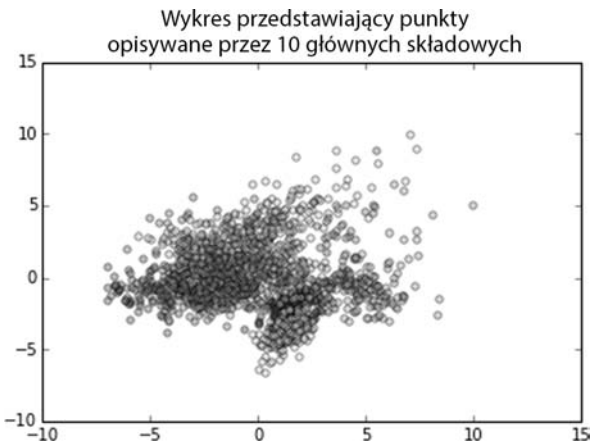
4. Powyższy kod wyświetla wariancję opisaną każdą z 10 pierwszych składowych.

W analizowanym przypadku 10 głównych składowych opisuje 0,589 wariancji całego zbioru danych. Nie jest to najgorszy wynik, biorąc pod uwagę to, że zeszliśmy z 64 do 10 składowych, ale wyraźnie widoczna jest potencjalna stratność algorytmu analizy głównych składowych. Teraz trzeba stwierdzić, czy ta redukcja sprawi, że dalsza analiza będzie prostsza. Czy wariancje pozostałych komponentów przeszkadzały w klasyfikacji?

Dysponujemy obiektem `data_r` zawierającym dane wyjściowe wygenerowane przez algorytm `pca` przetwarzający zbiór danych `digits`. Spróbujmy przedstawić te dane w sposób graficzny. W tym celu należy utworzyć wektor kolorów klasy `colors`, a następnie wygenerować wykres punktowy z pokolorowanymi klasami:

```
x = np.arange(10)
ys = [i+x+(i*x)**2 for i in range(10)]

plt.figure()
colors = cm.rainbow(np.linspace(0, 1, len(ys)))
for c, i, target_name in zip(colors, [1,2,3,4,5,6,7,8,9,10], labels):
    plt.scatter(data_r[labels == i, 0], data_r[labels == i, 1], c=c, alpha = 0.4)
    plt.legend()
    plt.title('Wykres przedstawiający punkty \n'
    'opisywane przez 10 głównych składowych')
    plt.show()
```

Wygenerowany zostanie następujący wykres:



Na podstawie tego rysunku można dojść do wniosku, że klasy dwóch pierwszych głównych składowych są do pewnego stopnia rozłączne, ale przeprowadzenie wysoce dokładnej klasyfikacji zbioru danych może być dość trudne. Klasy wydają się pogrupowane, a więc prawdopodobnie analiza skupień może dać w miarę sensowne rezultaty. Analiza głównych składowych umożliwiła lepsze poznanie struktury zbioru danych, co znacznie ułatwi dalszą analizę.

Skorzystajmy ze zdobytej wiedzy i przeprowadźmy analizę skupień za pomocą algorytmu grupowania metodą *k*-średnich.

Wprowadzenie grupowania metodą *k*-średnich

W poprzednim podrozdziale dowiedziałeś się, że algorytmy nienadzorowanego uczenia maszynowego są używane do określania struktury złożonych zbiorów danych. Większość tego typu algorytmów funkcjonuje bez potrzeby ręcznego wprowadzania danych wejściowych i działa bez treningowego zbioru danych (zbioru pogrupowanych elementów, na podstawie których algorytm jest w stanie określić zasady klasyfikacji). W związku z tym algorytmy nienadzorowane to efektywne narzędzia dostarczające informacji o strukturze i zawartości nowych lub nieznanych zbiorów danych. Pozwalają one analitykowi na szybkie rozpoznanie przetwarzanego zbioru.

Grupowanie — wprowadzenie

Grupowanie jest prawdopodobnie archetypową techniką uczenia nienadzorowanego. Są ku temu liczne przesłanki.

Poświęcono wiele czasu na optymalizację algorytmów grupowania — implementacje tych algorytmów są dostępne w większości języków (w tym w Pythonie), z których korzystają analitycy.

Algorytmy grupowania są zwykle bardzo szybkie — ich implementacje są wydajne i pozwalają na zrównoleglenie obliczeń. Umożliwia to dość proste przeprowadzanie wielu operacji grupowania nawet na dużych zbiorach danych. Skalowalne implementacje algorytmów grupowania pozwalają na zrównoleglenie obliczeń umożliwiające przetwarzanie terabajtowych zbiorów danych.

Algorytmy grupowania są zwykle łatwe do zrozumienia — zasady ich działania da się bez kłopotu wytłumaczyć.

Najpopularniejszym algorytmem grupowania jest metoda *k*-średnich. Algorytm ten tworzy *k* klastrów, zaczynając od losowego rozrzucenia *k* punktów w przestrzeni zbioru danych. Każdy z tych punktów jest środkiem (średnim elementem) grupy. Następnie algorytm przeprowadza proces iteracyjny:

- Każdy punkt jest przypisywany do najbliższego klastra (odległość pomiędzy punktem a środkami klastrów jest obliczana jako suma kwadratów).
- Środek (centroid) każdej z grup jest uznawany za nową średnią, co sprawia, że odległości punktów od środka grupy ulęgają zmianie.

Po przeprowadzeniu odpowiedniej liczby iteracji centroidy znajdą się w położeniach minimalizujących metrykę, która jest zwykle definiowana jako najmniejsza suma odległości pomiędzy środkiem grupy a przyporządkowanymi do niej obserwacjami. Po zminimalizowaniu tych odległości nie są one przypisywane do kolejnych grup podczas kolejnych iteracji. Algorytm przerywa pracę, ponieważ znalazł rozwiązanie.

Rozpoczynamy grupowanie

Poznałeś zasadę działania algorytmu grupowania, a więc możesz przystąpić do jego uruchomienia. Sprawdź, co algorytm ten zrobi z Twoim zbiorem danych:

```
rom time import time
import numpy as np
import matplotlib.pyplot as plt

np.random.seed()

digits = load_digist()
data = scale(digits.data)

n_samples, n_features = data.shape
n_digits = len(np.unique(digits.target))
labels = digits.target

sample_size = 300
```

```
print("n_digits: %d, \t n_samples: %d, \t n_features: %d"
      % (n_digits, n_samples, n_features))

print(73 * '_')
print('% 9s' % 'inic.''         czas   inercja   hom.   zup.    traf.
↪ARI    AMI     sylwetka')

def bench_k_means(estimator, name, data):
    t0 = time()
    estimator.fit(data)
    print('% 9s %.2fs %i %.3f %.3f %.3f %.3f %.3f %.3f'
          % (name, (time() - t0), estimator.inertia_,
             metrics.homogeneity_score(labels, estimator.labels_),
             metrics.completeness_score(labels, estimator.labels_),
             metrics.v_measure_score(labels, estimator.labels_),
             metrics.adjusted_rand_score(labels, estimator.labels_),
             metrics.silhouette_score(data, estimator.labels_,
                                      metric='euclidean',
                                      sample_size=sample_size)))
```

Jedną z najważniejszych różnic pomiędzy tym kodem a przedstawionym wcześniej kodem analizującym główne składowe jest to, że kod przedstawiony w tej sekcji rozpoczyna się od zastosowania funkcji `scale` na zbiorze danych `digits`. Funkcja ta skaluje wartości zbioru danych tak, aby znajdowały się w zakresie od 0 do 1. Skalowanie danych jest bardzo ważnym procesem pozwalającym na uniknięcie problemów wynikających z nieproporcjonalnych zakresów wartości przyjmowanych przez wartości opisujące różne cechy. Konieczność przeprowadzenia skalowania, a także jego rodzaj i zakres wyjściowy zależą od rodzaju i natury analizowanych danych. Jeżeli rozkład danych wykazuje elementy odstające lub rozciągnięcie danych w dużym zakresie, to właściwe będzie wtedy zastosowanie skalowania logarytmicznego. Niezależnie od tego, czy zostanie to zrobione za pomocą wizualizacji i technik analizy eksploracyjnej, czy za pomocą statystyk podsumowujących, decyzje dotyczące skalowania są ściśle związane z analizowanymi danymi i stosowanymi technikami ich analizy. Dokładniejsze omówienie zagadnień związanych ze skalowaniem znajdziesz w rozdziale 7. „Selekcja cech — część II".

Na szczęście pakiet *scikit-learn* korzysta domyślnie z algorytmu *k-means++*, który jest lepszy od standardowego algorytmu *k*-średnich, jeżeli chodzi o szybkość i umiejętność unikania słabego grupowania.

Jest to możliwe, ponieważ algorytm ten korzysta z procedury inicjowania, która pozwala na znalezienie centroidów grup będących przybliżeniami minimalnych wariancji klas.

Zapewne zauważyłeś, że w zaprezentowanym kodzie obliczane są miary wydajności pozwalające ocenić skuteczność podziału dokonanego za pomocą metody *k*-średnich. Pomiar wydajności algorytmu grupującego za pomocą jednej wartości wyrażonej procentowo nie jest zbyt praktyczny pomimo tego, że takie rozwiązanie stosowane jest w przypadku innych algorytmów. Z definicji sukces algorytmu grupującego polega na osiągnięciu dającego się zinterpretować logicznego

podziału na grupy, czyli sukces ten jest kompromisem pomiędzy czynnikami takimi jak separacja klas, podobieństwo obserwacji zakwalifikowanych do tych samych klas i różnice pomiędzy elementami przyporządkowanymi do różnych klas.

Homogeniczność (jednorodność) jest prostą miarą przyjmującą wartości z zakresu od 0 do 1, która określa przyporządkowanie do grupy tylko elementów danej klasy. Wynik równy 1 świadczy o tym, że wszystkie grupy zawierają obserwacje z pojedynczych klas. Uzupełnieniem tego parametru jest **zupełność**, która przyjmuje wartości z tego samego zakresu co homogeniczność, ale określa stopień przypisania elementów danej klasy do tej samej grupy. W związku z tym zupełność równa 1 i homogeniczność równa 1 wskazują na idealne pogrupowanie.

Trafność (**miara *V***) jest średnią harmoniczną homogeniczności i zupełności, czyli czymś analogicznym do **miary *F*** klasyfikacji binarnej. Trafność przyjmuje wartość z zakresu od 0 do 1, która pozwala na monitorowanie homogeniczności i zupełności.

Skorygowany indeks Randa (indeks **ARI**) jest miarą podobieństwa biorącą pod uwagę zgodność zestawu przypisań. W przypadku grupowania indeks ten mierzy zgodność prawdziwych przypisanych wcześniej etykiet obserwacji z etykietami przewidzianymi przez algorytm grupujący. Indeks Randa mierzy podobieństwo etykiet w skali od 0 do 1, gdzie jedynka oznacza idealne przewidywanie etykiet.

Głównym problemem tych, a także innych, podobnych do nich miar wydajności, takich jak kryterium informacyjne Akaikego, jest to, że wymagają one rozumienia danych — część obserwacji lub wszystkie obserwacje muszą być oznaczone za pomocą etykiet. Jeżeli takie etykiety nie istnieją i nie można ich wygenerować, to miary te się nie sprawdzą. To w praktyce bardzo duża wada, ponieważ większość zbiorów danych nie posiada etykiet, a ich tworzenie może być czasochłonne.

Jednym ze sposobów na określenie wydajności procesu grupowania za pomocą metody *k*-średnich przy danych bez etykiet jest skorzystanie ze **współczynnika sylwetki**. Miara ta określa trafność zdefiniowania grup wewnątrz modelu. Współczynnik sylwetki danego zbioru danych jest średnią współczynnika każdej próbki. Oblicza się go za pomocą następującego wzoru:

$$s = \frac{b - a}{\max(a, b)}$$

We wzorze tym:

- a jest średnią odległością pomiędzy próbką a wszystkimi pozostałymi punktami tego samego klastra,
- b jest średnią odległością pomiędzy próbką a wszystkimi innymi punktami kolejnego najbliższego klastra.

Współczynnik ten przyjmuje wartości od –1 do 1, przy czym –1 oznacza nieprawidłowe grupowanie, a 1 oznacza bardzo zwarte grupowanie. Wartości oscylujące w okolicy 0 oznaczają nakładanie się klastrów. Współczynnik ten może być użyty do określenia poprawności podziału, który zdefiniowaliśmy wcześniej.

W przypadku zbioru danych `digits` możemy określić wszystkie opisane parametry wydajności. W tym celu należy uzupełnić zaprezentowany wcześniej przykład — zainicjować funkcję `bench_k_means` na zbiorze danych `digits`:

```
bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_
init=10), name="k-means++", data=data)
print(79 * '_')
```

Teraz kod wygeneruje następujące dane wyjściowe (kod korzysta z losowanego ziarna, a więc uzyskane przez Ciebie wartości mogą odbiegać od poniższych):

n_digits: 10, n_samples: 1797, n_features: 64								
inic.	czas	inercja	hom.	zup.	traf.	ARI	AMI	sylwetka
k-means++	0.25s	69517	0.596	0.643	0.619	0.465	0.592	0.123

Przyjrzyjmy się otrzymanym wartościom.

Wartość współczynnika sylwetki (`0,123`) jest dość niska, ale to chyba nic dziwnego, gdy weźmie się pod uwagę to, że zbiór danych zawierający zapisane ręcznie cyfry charakteryzuje się zaszumieniem i zachodzeniem na siebie elementów. Inne uzyskane wartości również nie są zachwycające. Trafność (`0,619`) wydaje się osiągać dość rozsądną wartość, ale parametr homogeniczności jest dość słaby, co sugeruje, że centroidy grup nie zostały dobrane idealnie. Ponadto współczynnik ARI równy `0,465` także nie jest dobrym wynikiem.

> Umieśćmy tę teorię w praktycznym kontekście. Najgorszy możliwy przypadek, czyli grupowanie losowe, da w najlepszym razie wynik o dokładności 10%. W takiej sytuacji wszystkie współczynniki byłyby odpowiednio niskie. Co prawda uzyskane przez nas wartości współczynników są o wiele lepsze, ale wciąż dalekie od najlepszych rozwiązań. Podczas lektury rozdziału 4., „Konwolucyjne sieci neuronowe", przekonasz się, że konwolucyjne sieci neuronowe uzyskują bardzo niskie wartości błędów klasyfikacji zbioru danych cyfr pisanych ręcznie. Tak wysokiego poziomu dokładności najprawdopodobniej nie osiągniemy za pomocą tradycyjnego grupowania metodą *k*-średnich.

Tak czy inaczej, warto przyjąć założenie, że można osiągnąć lepszy wynik.

Podejmijmy jeszcze jedną próbę analizy. Tym razem dodajmy dodatkowy etap przetwarzania danych — zastosujmy opisaną wcześniej metodę analizy głównych składowych w celu redukcji liczby wymiarów wejściowego zbioru danych. Kod wykonujący tę operację jest bardzo prosty:

```
pca = PCA(n_components=n_digits).fit(data)
bench_k_means(KMeans(init=pca.components_, n_clusters=10),
name="PCA",
data=data)
```

Zaprezentowany kod przetwarza zbiór danych `digits` przy użyciu metody `PCA`. Generuje on na wyjściu tyle głównych składowych, ile klas ma dany zbiór (w tym przypadku klasami są cyfry).

Przed wykonaniem kolejnych operacji warto przeanalizować dane wyjściowe metody PCA, ponieważ mała ilość głównych składowych może oznaczać występowanie współliniowości cech lub innych problemów wymagających przyjrzenia się danym.

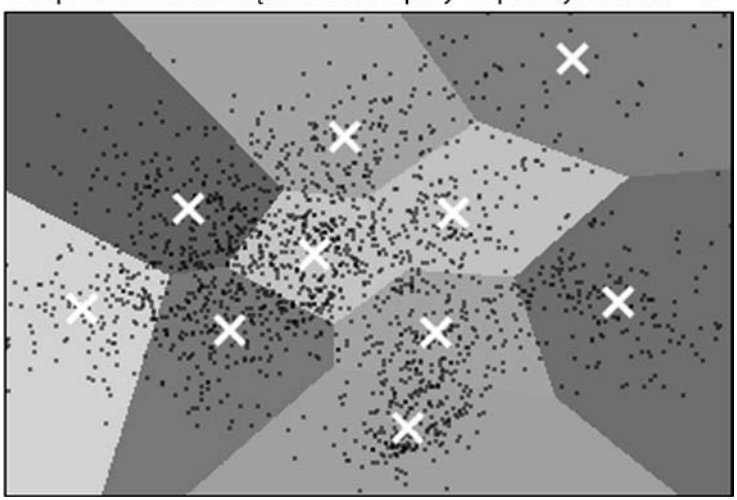
Tym razem wynik grupowania jest wyraźnie lepszy:

n_digits: 10, n_samples: 1797, n_features: 64							
inic.	czas	inercja	hom.	zup.	traf.	ARI	sylwetka
PCA	0.2s	71820	0.673	0.715	0.693	0.567	0.123

Trafność i współczynnik ARI wzrosły o około 0,08 punkta — trafność osiągnęła dość sensowną wartość (0,693). Współczynnik sylwetki podziału nie uległ znacznej zmianie. Jeśli weźmie się pod uwagę złożoność analizowanego zbioru danych, nakładanie się na siebie jego klas i to, że wykonaliśmy tylko operację prostego dodania kodu, osiągnięte wyniki można uznać za dobre.

Jeżeli przyjrzysz się zbiorowi danych digits przedstawionemu na wykresie z naniesionymi grupami, to zauważysz uformowanie pewnych znaczących klastrów, ale przekonasz się również o tym, że wykrywanie znaków na podstawie wektorów cech może okazać się trudnym zadaniem:

Grupowanie metodą *k*-średnich przy współczynniku $k = 10$



Dostrajanie konfiguracji klastrów

W poprzedniej sekcji dowiedziałeś się, jak korzystać z metody *k*-średnich, przeanalizowałeś jej kod, zobaczyłeś graficzną interpretację efektów pracy tego algorytmu i poznałeś współczynniki pozwalające na ocenę uzyskanych wyników. W tej sekcji opiszę rzeczy, na które trzeba zwrócić dodatkową uwagę podczas stosowania metody *k*-średnich do rozwiązywania prawdziwych problemów.

Kolejnym ważnym zagadnieniem jest wybór właściwej wartości parametru *k*. Rozpoczęcie grupowania metodą *k*-średnich przy wybraniu niewłaściwej wartości *k* nie będzie niczym szkodliwym, ale na początku możesz nie wiedzieć, w jakim zakresie wartości parametru *k* należy eksperymentować (na ile grup trzeba podzielić zbiór danych).

Zaprezentowany wcześniej kod można zmodyfikować tak, aby dokonywał kilku operacji grupowania dla różnych wartości parametru *k*, a następnie porównać parametry podziału, ale niestety nie powiedzą one, który z podziałów dobrze odzwierciedla strukturę danych. Wraz ze wzrostem wartości parametru *k* może dojść do spadku współczynnika sylwetki podziału i zmniejszenia ilości niewyjaśnionych wariancji, bez utworzenia grup poprawiających rozumienie danych. Do ekstremalnego przypadku doszłoby, gdyby parametr *k* był równy *o*, gdzie *o* byłoby liczbą obserwacji w każdej próbce — każdy punkt znajdowałby się wtedy we własnym klastrze, wartość współczynnika sylwetki podziału byłaby niska, ale uzyskany wynik operacji grupowania nie pozwalałby na poznanie istoty danych. W praktyce istnieje wiele mniej ekstremalnych przypadków nadmiernego dopasowania, do którego może dojść w wyniku przyjęcia zbyt dużej wartości parametru *k*.

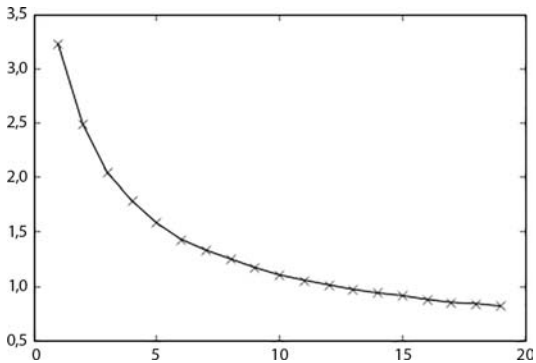
Aby uniknąć tego typu problemów, warto skorzystać z technik pomocniczych pozwalających na sensowne dobranie wartości parametru *k*. W przypadku naszego zbioru danych możemy skorzystać z **metody łokcia** — bardzo prostej techniki polegającej na przedstawieniu na wykresie procentowej ilości wyjaśnionych wariancji przy różnych wartościach parametru *k*. Wykres ten wygląda zwykle jak ugięte ramię.

Poniższy kod generuje taki wykres dla zbioru zredukowanego metodą analizy głównych składowych:

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

digits = load_digits()
data = scale(digits.data)

n_samples, n_features = data.shape
n_digits = len(np.unique(digits.target))
labels = digits.target

K = range(1,20)
explainedvariance= []
for k in K:
    reduced_data = PCA(n_components=2).fit_transform(data)
    kmeans = KMeans(init = 'k-means++', n_clusters = k, n_init = k)
    kmeans.fit(reduced_data)
    explainedvariance.append(sum(np.min(cdist(reduced_data,
    kmeans.cluster_centers_, 'euclidean'), axis =
    1))/data.shape[0])
    plt.plot(K, meandistortions, 'bx-')
```

```
plt.show()
```

Powyższa aplikacja metody łokcia korzysta z danych zredukowanych za pomocą metody analizy głównych składowych (poprzedni fragment kodu) i wykonuje test wyjaśnionej wariancji (a konkretnie test wariancji wewnątrzgrupowej). Zaprezentowany kod generuje miarę nieokreślonej wariancji dla każdej wartości *k* znajdującej się w podanym zakresie. Pracujemy nad zbiorem `digits` (wiemy, że ma on 10), a więc sprawdzamy zakres (`range`) od `1` do `20`:



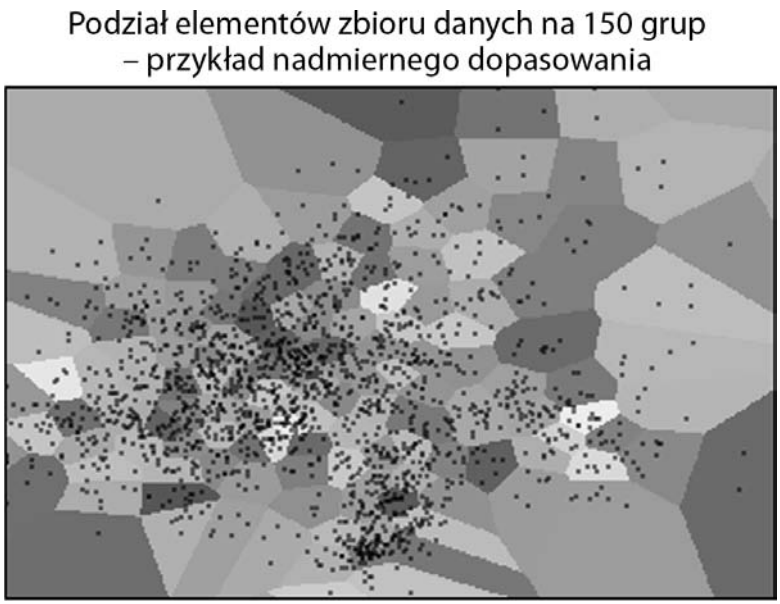
Korzystanie z metody łokcia pozwala na wybranie wartości parametru *k*, która maksymalizuje wyjaśnioną wariancję przy minimalizacji parametru *K*, to jest wartości parametru *k* znajdującej się na wygięciu „łokcia". Minimalny wzrost wyjaśnionej wariancji przy wyższych wartościach parametru *k* wiąże się ze wzrostem ryzyka nadmiernego dopasowania.

Wykresy łokciowe mogą odbiegać mniej lub bardziej od kształtu łokcia, a punkt wygięcia wykresu może być czasami trudny do zidentyfikowania. Zaprezentowany przykład pokazuje bardziej stopniową zmianę wartości od tej, którą można zaobserwować w przypadku innych zbiorów danych. Wiemy, że zbiór składa się z dziesięciu klas, ale warto zauważyć, że z wykresu wynika, że punkt wygięcia łokcia znajduje się w okolicy pięciu klas (dalszy wzrost parametru *k* wiąże się z niższą poprawą uzyskanych wyników). Bierze się to w dużej mierze z nakładania się klas. Zjawisko to można było zaobserwować na poprzednim wykresie. Zbiór składa się z dziesięciu klas, ale identyfikacja więcej niż pięciu jest wyraźnie trudna.

Wiedząc o tym, warto wziąć pod uwagę to, że metoda łokcia nie jest obiektywną zasadą. Należy ją traktować w sposób poglądowy. Zastosowanie metody analizy głównych składowych jako procesu przygotowującego dane do sprawniejszego grupowania również zwykle wygładza wykres — doprowadza do powstania bardziej smukłej krzywej.

Poza skorzystaniem z metody łokcia czasami warto przyjrzeć się samym grupom, tak jak robiliśmy to wcześniej podczas korzystania z metody analizy głównych składowych w celu zredukowania wielowymiarowości zbioru danych. Poprzez tworzenie wykresów przedstawiających zbiór danych i projektu przypisania elementów zbioru danych do poszczególnych grup czasami bardzo łatwo można wykryć to, że implementacja metody *k*-średnich dopasowała się do lokalnego minimum lub uzyskała nadmierne dopasowanie. Poniższy wykres przedstawia ekstremalne nadmierne

dopasowanie uzyskane przez zaprezentowany wcześniej algorytm grupowania metodą *k*-średnich przetwarzający zbiór danych `digits`. Algorytm ten miał podzielić zbiór na 150 grup. Niektóre grupy zawierają tylko jedną obserwację — taki podział nie pozwala również na logiczny opis pozostałych próbek:



Bardzo szybko można wygenerować i zinterpretować wykres łokcia lub przypisania do grup, ale warto pamiętać o tym, że są to techniki heurystyczne. Jeżeli zbiór danych zawiera określoną liczbę klas, nie możemy być pewni, że metoda heurystyczna doprowadzi do dających się uogólnić wyników.

Kolejną wadą jest to, że analiza wykresu jest czymś, co trzeba zrobić ręcznie, a to sprawia, że nie nadaje się do zastosowania w środowiskach produkcyjnych i nie można jej zautomatyzować. Lepiej jest korzystać z rozwiązania opartego na kodzie, które można zautomatyzować. Przykładem takiego rozwiązania jest ***v*-krotna walidacja krzyżowa**.

Walidacja krzyżowa jest prosta do przeprowadzenia — wystarczy podzielić zbiór danych na *v* części. Jedna z części jest odkładana na bok — ma w przyszłości pełnić funkcję testowego zbioru danych. Model jest trenowany na zbiorze treningowym składającym się ze wszystkich części zbioru danych poza zbiorem testowym. Spróbujmy to zrobić ze zbiorem danych `digits`:

```
import numpy as np
from sklearn import cross_validation
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.preprocessing import scale

digits = load_digits()
```

```
data = scale(digits.data)

n_samples, n_features = data.shape
n_digits = len(np.unique(digits.target))
labels = digits.target

kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=n_digits)
cv = cross_validation.ShuffleSplit(n_samples, n_iter = 10, test_size = 0.4,
random_state = 0)
scores = cross_validation.cross_val_score(kmeans, data,
↪labels, cv = cv, scoring = 'adjusted_rand_score')
print(scores)
print(sum(scores)/cv.n_iter)
```

Powyższy kod wykonuje znane Ci operacje ładowania danych, a także przygotowania i inicjowania algorytmu grupowania metodą *k*-średnich. Następnie definiuje zmienną `cv` zawierającą parametry walidacji krzyżowej, czyli między innymi liczbę iteracji (`n_iter`) i ilość danych, które powinny być użyte podczas każdej iteracji. W tym przypadku 60% danych pełni funkcję zbioru treningowego, a 40% — zbioru testowego.

Następnie stosowany jest model *k*-średnich, funkcja oceniająca krzyżowo wynik grupowania korzysta z parametrów `cv`, po czym wyświetla wyniki przypisane zmiennej `scores`. Przyjrzyjmy się tym wynikom:

```
[ 0.39276606 0.49571292 0.43933243 0.53573558 0.42459285
  0.55686854 0.4573401  0.49876358 0.50281585 0.4689295 ]
0.4772857426
```

W nawiasie kwadratowym umieszczono skorygowane indeksy Randa walidacji krzyżowej grupowania przeprowadzonego za pomocą algorytmu *k-means++* dla 10 kolejnych krotności walidacji. Wartości wahają się od 0,4 do 0,55. Wcześniejsza wartość skorygowanego indeksu Randa dla algorytmu *k-means++* bez analizy głównych składowych mieściła się w tym przedziale (wynosiła 0,465). Utworzyliśmy kod, który możemy dołączyć do kodu przeprowadzającego analizę w celu zautomatyzowania sprawdzania jakości grupowania.

Zgodnie z tym, co pisałem wcześniej, wybór miary sukcesu grupowania zależy od posiadanych przez Ciebie informacji. W większości przypadków w pracy z prawdziwymi danymi nie będziesz dysponował etykietami definiującymi przynależność obserwacji do grup i będziesz musiał korzystać z opisanych wcześniej parametrów takich jak współczynnik sylwetki podziału.

> Czasami zastosowanie zarówno walidacji krzyżowej, jak i wizualizacji nie musi dać sensownych rezultatów. Dotyczy to szczególnie nieznanych zbiorów danych, w przypadku których podział, który chcesz zweryfikować, wypada gorzej od innych podziałów (innych wartości parametru *k*) lub szumu.
>
> Podobnie jak w przypadku pozostałych algorytmów opisanych w tej książce, zachodzi konieczność zrozumienia zbioru danych, z którym pracujesz. Bez odpowiedniej wiedzy o zbiorze danych możliwe jest przeprowadzenie technicznie poprawnej i rygorystycznej analizy prowadzącej do nieprawidłowych wniosków. W rozdziale 6., „Rozpoznawanie języka naturalnego i selekcja cech”,

opiszę w sposób bardziej szczegółowy zasady i techniki badania i przygotowywania zbiorów danych o nieznanej naturze.

Sieci Kohonena

Sieci Kohonena (ang. *Self-Organizing Map*, SOM) to technika służąca do generowania topologicznych reprezentacji danych w przestrzeni charakteryzującej się zredukowaną liczbą wymiarów. Jest to jedna z wielu technik służących do redukcji liczby wymiarów. Inną, bardziej znaną techniką tego typu jest analiza głównych składowych. Sieci Kohonena oferują jednakże większe możliwości redukcji wielowymiarowości, a także wizualnej reprezentacji danych.

Sieci Kohonena — wprowadzenie

Algorytm sieci Kohonena polega na przeprowadzaniu iteracji wielu prostych operacji. W przypadku przetwarzania niewielkich danych działa podobnie do grupowania metodą *k*-średnich (za chwilę przedstawię to w praktyce). Dla danych o dużej skali sieci Kohonena są doskonałym narzędziem do poznawania ich topologii.

Sieć Kohonena (ma zwykle kształt kwadratowy lub sześcienny) tworzą węzły zawierające wektor wagi charakteryzujący się taką samą liczbą węzłów jak wejściowy zbiór danych. Węzły mogą być inicjowane w sposób losowy, ale w celu przyspieszenia procesu trenowania stosuje się inicjowanie przybliżone do rozkładu zbioru danych.

Algorytm dokonuje kolejnych iteracji, a funkcję danych wejściowych pełnią kolejne obserwacje. Podczas iteracji wykonywane są następujące operacje:

- Identyfikowany jest najlepszy węzeł w bieżącej konfiguracji — **jednostka najlepszego dopasowania** (ang. *Best Matching Unit*, BMU). Jednostka ta jest określana w wyniku pomiaru odległości euklidesowej w przestrzeni danych wszystkich wektorów wag.
- Jednostka najlepszego dopasowania jest przesuwana w kierunku wektora wejściowego.
- Korygowane są również sąsiednie węzły. Zwykle korekta ich położenia jest mniejsza — zależy ona od funkcji otoczenia. Stosuje się różne funkcje otoczenia. W tym rozdziale będziemy korzystać z funkcji Gaussa.

Proces ten jest wykonywany przez tyle iteracji, ile trzeba do zbiegnięcia się sieci (osiągnięcia punktu, w którym nowe dane wejściowe nie tworzą możliwości minimalizacji strat). O ile to możliwe, w procesie tym stosuje się próbkowanie.

Węzeł sieci Kohonena różni się nieco od węzła sieci neuronowej. Zwykle posiada on wektor wag długości równej liczbie wymiarów wejściowego zbioru danych, co umożliwia zachowanie

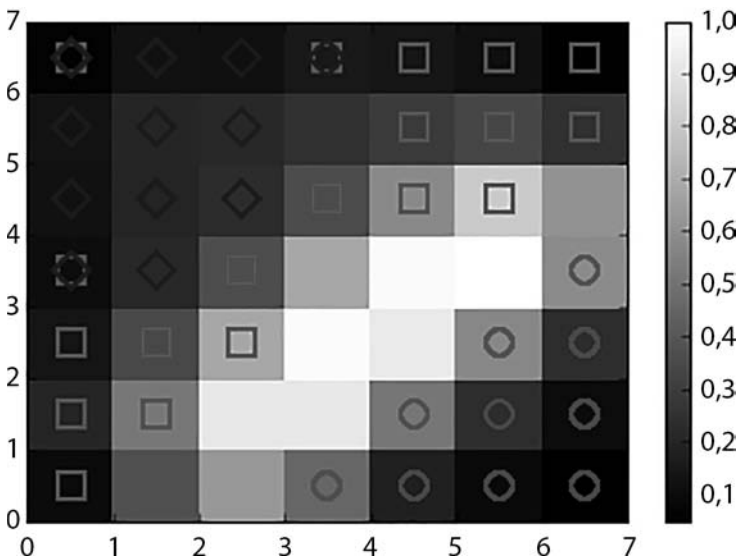
topologii wejściowego zbioru danych, a także jej wizualizację za pomocą mapowania na płaszczyznę o mniejszej liczbie wymiarów.

Kod implementacji klasy `SOM` (sieci Kohonena) znajdziesz w archiwum pobranym ze strony wydawnictwa Helion, w pliku *som.py*. Na razie spróbujmy zastosować algorytm sieci Kohonena w znanym kontekście.

Korzystanie z sieci Kohonena

Algorytm sieci Kohonena ma charakter iteracyjny i jest oparty na porównywaniu wektorów za pomocą odległości euklidesowej.

Odwzorowanie to ma zwykle formę dość czytelnej dwuwymiarowej siatki. Sieci Kohonena dość wyraźnie odwzorują popularny treningowy zbiór danych *Iris*:



Na diagramie oddzielono od siebie poszczególne klasy i przedstawiono je w sposób przestrzenny. Kolor tła odzwierciedla zagęszczenie grupowania. Niebieska i zielona klasa minimalnie się nakładają. W przypadku zbioru danych *Iris* sieci Kohonena osiągnęły zbieżność (rozwiązanie) po 100 iteracjach. Poprawa po przeprowadzeniu 1000 iteracji była niewielka. W przypadku bardziej złożonych zbiorów danych zawierających klasy, które są trudniejsze do odseparowania, algorytm sieci Kohonena może wymagać wykonania dziesiątek tysięcy iteracji.

To dziwne, ale pakiety Pythona takie jak *scikit-learn* nie zawierają implementacji algorytmu tworzącego sieci Kohonena. W związku z tym musimy korzystać z własnej implementacji.

Kod algorytmu tworzącego sieci Kohonena, z którego korzystam, możesz znaleźć w dołączonych do książki skryptach. Przyjrzyjmy się skryptowi korzystającemu z tego algorytmu:

```
import numpy as np
from sklearn.datasets import load_digits
from som import Som

from pylab import plot,axis,show,pcolor,colorbar,bone

digits = load_digits()
data = digits.data
labels = digits.target
```

Jak na razie załadowaliśmy zbiór danych `digits` i zidentyfikowaliśmy etykiety (`labels`), które będziemy traktować jako oddzielny zbiór danych. Operacja ta pozwoli nam na obserwację oddzielania od siebie klas przez algorytm sieci Kohonena i przypisywania ich do obiektu mapy (`map`):

```
som = Som(16,16,64,sigma=1.0,learning_rate=0.5)
som.random_weights_init(data)
print("Inicjowanie sieci Kohonena.")
som.train_random(data,10000)
print("\n. Przetwarzanie zakończone.")

bone()
pcolor(som.distance_map().T)
colorbar()
```

Teraz skorzystaliśmy z klasy `Som` umieszczonej w drugim pliku (*som.py*) pobranym z serwera FTP wydawnictwa Helion. Klasa ta zawiera metody wymagane do zaimplementowania opisanego wcześniej algorytmu sieci Kohonena. Do funkcji w roli argumentów przekazujemy wymiary mapy (po przeanalizowaniu różnych opcji doszedłem do wniosku, że w przypadku analizowanego zbioru najlepiej jest zacząć od siatki o wymiarach 16×16 — będzie ona miała wystarczająco dużo miejsca, aby zmieściły się w niej wszystkie grupy, a także część przestrzeni grup, które na siebie zachodzą) oraz wymiarowość danych wejściowych (argument ten określa długość wektora wag węzłów sieci Kohonena). Ponadto dostarczane są wartości parametrów *sigma* i współczynnika nauki (ang. *learning rate*).

Parametr *sigma* definiuje w tym przypadku rozkład funkcji sąsiedztwa. Zgodnie z tym, co pisałem wcześniej, korzystamy z gaussowskiej funkcji sąsiedztwa. Wartość parametru *sigma* powinna być dobrana do wielkości siatki. W przypadku siatki 8×8 parametr *sigma* przyjmuje zwykle wartość równą 1,0. W przypadku siatki 16×16 przyjmiemy parametr *sigma* równy 1,3. Jeśli zostanie wybrana zbyt niska wartość parametru *sigma*, wartości będą miały tendencję do tworzenia grup w okolicy środka siatki. Wybranie zbyt wysokiej wartości *sigma* oznacza, że na środku siatki pozostanie kilka dużych pustych miejsc.

Współczynnik nauki określa początkowe tempo uczenia się sieci Kohonena. Wraz z kolejnymi iteracjami mapy parametr ten będzie zmieniał swoją wartość zgodnie z następującą funkcją:

współczynnik nauki(t) = współczynnik nauki/(1 + t/(0,5 · t)),

w której *t* jest indeksem iteracji.

Przedstawiony skrypt inicjuje następnie algorytm sieci Kohonena losowymi wartościami wag.

> Podobnie jak to miało miejsce w przypadku grupowania metodą *k*-średnich, zastosowana metoda inicjowania jest wolniejsza od inicjowania opartego na przybliżonym rozkładzie danych. Proces przetwarzania wstępnego podobny do tego, który zastosowano w przypadku algorytmu *k-means++*, przyspieszyłby działanie algorytmu sieci Kohonena, ale jak na razie nie korzystamy z tego rozwiązania, ponieważ sieci Kohonena działają wystarczająco wydajnie podczas przetwarzania zbioru danych `digits`.

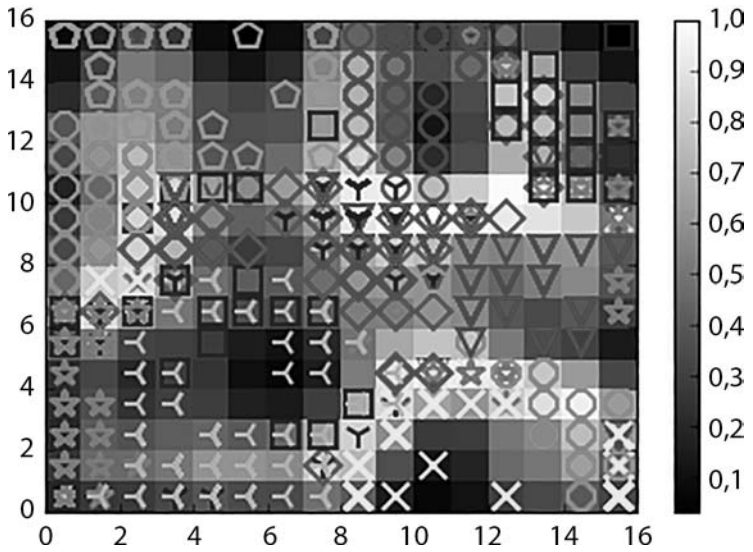
Następnie musimy określić etykiety i kolory poszczególnych klas, tak aby można je było później odróżnić podczas analizy wykresu wygenerowanego przez sieci Kohonena. W dalszej kolejności należy przeprowadzić iterację przez każdy punkt danych.

Podczas każdej iteracji zgodnie z wynikami obliczeń przeprowadzonych przez algorytm sieci Kohonena tworzony jest znacznik jednostki najlepszego dopasowania właściwy dla danej klasy.

Skrypt po zakończeniu iteracji algorytmu sieci Kohonena utworzy **macierz *u*** — pokolorowaną macierz względnego zagęszczenia obserwacji. Będzie ona miała formę monochromatycznego wykresu:

```
labels[labels == '0'] = 0
labels[labels == '1'] = 1
labels[labels == '2'] = 2
labels[labels == '3'] = 3
labels[labels == '4'] = 4
labels[labels == '5'] = 5
labels[labels == '6'] = 6
labels[labels == '7'] = 7
labels[labels == '8'] = 8
labels[labels == '9'] = 9

markers = ['o', 'v', '1', '3', '8', 's', 'p', 'x', 'D', '*']
colors = ["r", "g", "b", "y", "c", (0,0.1,0.8), (1,0.5,0), (1,1,0.3),
"m", (0.4,0.6,0)]
for cnt,xx in enumerate(data):
    w = som.winner(xx)
    plot(w[0]+.5,w[1]+.5,markers[labels[cnt]],
    markerfacecolor='None', markeredgecolor=colors[labels[cnt]],
    markersize=12, markeredgewidth=2)
    axis([0,som.weights.shape[0],0,som.weights.shape[1]])
    show()
```

Przedstawiony kod generuje wykres podobny do tego:



Kod tworzy wykres sieci Kohonena mającej 16×16 węzłów. Jak widać, mapa dość dobrze rozdziela klasy, tworząc wyróżnialne topologicznie obszary mapy. Niektóre klasy (głównie cyfry 5 w niebieskozielonych okręgach i 9 w zielonych gwiazdach) zostały umieszczone w wielu miejscach przestrzeni sieci Kohonena. Jednakże większość klas jest umieszczona tylko w jednym miejscu, a więc sieć Kohonena zadziałała dość efektywnie. Z macierzy ujednoliconych odległości wynika, że obszary o dużym zagęszczeniu punktów zawierają obserwacje należące do wielu klas. To nic nowego — zaobserwowaliśmy to już w przypadku wykresów wygenerowanych w wyniku grupowania za pomocą metody *k*-średnich i metody analizy głównych składowych.

Dalsza lektura

Na stronie *http://setosa.io/ev/principal-component-analysis/* znajdziesz fantastyczne interaktywne, wizualne wyjaśnienie metody analizy głównych składowych, stworzone przez Victora Powella i Lewisa Lehe. Stronę tę powinni odwiedzić czytelnicy, którzy nie znali wcześniej podstawowych pojęć związanych z analizą głównych składowych, i ci, którzy czują, że nie zrozumieli ich w pełni.

Na stronie *http://arxiv.org/abs/1404.1100* znajdziesz dłuższe wyjaśnienie metody analizy głównych składowych, które jest przedstawione z punktu widzenia matematyki i porusza zagadnienia związane z transformacjami macierzy. Autorem tego artykułu jest pracownik firmy Google — Jonathon Shlens.

Jeżeli szukasz dobrego przykładu związanego z artykułem Jonathona Shlensa, zajrzyj na stronę *http://sebastianraschka.com/Articles/2015_pca_in_3_steps.html* — Sebastian Raschka opracował kod Pythona przedstawiający możliwości analizy głównych składowych na przykładzie zbioru danych *Iris*.

Więcej informacji na temat argumentów obsługiwanych przez klasę `PCA` znajdziesz w dokumentacji pakietu *sklearn* znajdującej się na stronie *http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html*.

David Robinson prowadzi fantastyczny blog, na którym można znaleźć eksperckie wyjaśnienia związane z algorytmem *k*-średnich, w tym szczegółową analizę czynników, z którymi algorytm ten sobie nie radzi, i alternatywne rozwiązania, które pozwalają na obejście sytuacji sprawiających problemy: *http://varianceexplained.org/r/kmeans-free-lunch/*.

Na stronie *https://bl.ocks.org/rpgove/0060ff3b656618e9136b* znajdziesz artykuł dotyczący metody łokcia, napisany przez Roberta Gove'a.

W dokumentacji pakietu *sklearn* zamieszczono również informacje o algorytmach uczenia nienadzorowanego, w tym metody *k*-średnich: *http://scikit-learn.org/stable/tutorial/statistical_inference/unsupervised_learning.html*.

Większość materiałów dotyczących sieci Kohonena jest albo przestarzała, albo dotyczy zagadnień wysokiego poziomu, albo jest napisana językiem formalnym. Dobrą alternatywą dla przedstawionego przeze mnie opisu są notatki opracowane przez Johna Bullinarię: *http://www.cs.bham.ac.uk/~jxb/NN/l16.pdf*.

Czytelnikom, którzy chcą lepiej zrozumieć matematykę leżącą u podstaw sieci Kohonena, polecam lekturę książki dotyczącej tejże sieci, która to pozycja została napisana przez samego Tuevo Kohonena (warto zacząć od wydania z 2012 roku).

Na stronie *https://onlinecourses.science.psu.edu/stat501/node/344* znajdziesz wyjaśnienie pojęcia współliniowości mnogiej, do którego odwoływałem się w tym rozdziale.

Podsumowanie

W tym rozdziale przedstawiłem trzy techniki uczenia maszynowego i opisałem ich zastosowania we wstępnym przetwarzaniu danych i redukcji ilości wymiarów. Techniki te pozwalają uzyskać wiele przydatnych informacji dotyczących nieznanych zbiorów danych.

Zacząłem od analizy głównych składowych — popularnej techniki pozwalającej na redukcję liczby wymiarów, zrozumienie zbioru danych, a także graficzne przedstawianie zbiorów wielowymiarowych. Później zająłem się grupowaniem danych za pomocą metody *k*-średnich, opisałem sposoby poprawy wyników generowanych przez tę metodę, a także techniki oceny tych wyników

(metodę łokcia, walidację krzyżową). Okazało się, że grupowanie zbioru `digits` bezpośrednio metodą *k*-średnich nie dało oczekiwanych rezultatów. Było to spowodowane nakładaniem się klas, które zostało wcześniej wykryte podczas analizy głównych składowych. Problem ten rozwiązaliśmy, stosując analizę głównych składowych jako proces wstępny, który pozwolił na uzyskanie lepszych wyników grupowania.

Na koniec opisałem algorytm sieci Kohonena, który umożliwił lepsze rozdzielenie klas zbioru danych `digits` niż metoda analizy głównych składowych.

Poznałeś już podstawy technik nienadzorowanego uczenia maszynowego i metodykę procesu analizy danych. Czas, abyś zaczął korzystać z bardziej wydajnych algorytmów nienadzorowanego uczenia maszynowego.

Skorowidz

A

B

C

D

K

L

Ł

M

N

O

P

R

S

T

U

W

Z

Zaawansowane uczenie maszynowe z językiem Python

Uczenie maszynowe przyczyniło się do powstania wielu innowacyjnych technologii. Pojazdy autonomiczne, mechanizmy rozpoznawania obrazów, badania genetyczne, a także dynamiczne dostosowywanie prezentowanych treści do preferencji odbiorcy to tylko niektóre przykłady. Możliwości związane z rozwojem tych technik sprawiają, że analityka danych i zaawansowane uczenie maszynowe stają się wyjątkowo cenną wiedzą. Dotyczy to szczególnie nowatorskich technik analizy danych, takich jak głębokie uczenie, algorytmy częściowo nadzorowane i metody zespołowe.

Niniejsza książka jest przystępnie napisanym podręcznikiem, dzięki któremu poznasz niektóre zaawansowane techniki uczenia maszynowego. Szczególną uwagę poświęcono tu algorytmom uczenia maszynowego: zostały dokładnie wyjaśnione, opisano ich zastosowanie oraz topologię, metody uczenia i miary wydajności. Każdy rozdział uzupełniono o wykaz źródeł, pomocny w dalszym zgłębianiu tematu. Dodatkowo przedstawiono wiele cennych wskazówek dotyczących specyfiki pracy analityka danych. Do prezentacji przykładów posłużył język Python z uwagi na jego wszechstronność, elastyczność, prostotę oraz możliwość stosowania do specjalistycznych zadań.

Zaawansowane uczenie maszynowe — poznaj algorytmy przyszłości!

W książce:

- identyfikacja struktur i wzorców w zbiorach danych
- stosowanie sieci neuronowych
- praca z językiem naturalnym
- modele zespołowe i poprawa ich elastyczności
- narzędzia uczenia maszynowego w Pythonie

John Hearty jest autorytetem w dziedzinie analityki danych i inżynierii infrastruktury. Przez pewien czas zajmował się modelowaniem zachowań gracza i infrastrukturą dużych zbiorów danych w Microsofcie. Do jego ważniejszych projektów należą modelowanie umiejętności gracza w grach asymetrycznych i modele segmentacji graczy mające na celu zindywidualizowanie rozgrywki. Obecnie jest niezależnym ekspertem, szczególnie cenionym przez zespoły zajmujące się eksploracją danych. W wolnym czasie tworzy modele uczenia maszynowego w Pythonie.

[PACKT] open source PUBLISHING community experience distilled

Helion

księgarnia internetowa
http://helion.pl

zamówienia telefoniczne
0 801 339900
0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-3607-0

cena: 57,00 zł