Data Science - Przegląd Zagadnień

Zawartość

Ta książka jest, w zamyśle, pomocą dla osób zainteresowanych tematyką zaawansowanej analizy danych (ang. *data science* lub, w jednej z polskich interpretacji: *danologią*), a także studentów uczestniczących w zajęciach w ramach studiów podyplomowych **Data Science na Wrocławskim Uniwersytecie Ekonomicznym**.

Dostępnych jest bardzo wiele materiałów w języku angielskim dotyczących tej tematyki. Ze względu na ich powszechność - obejmują znacznie szersze spektrum tematów, często tworzą je także renomowane ośrodki naukowe takie jak MIT czy Stanford. Nierzadko jednak, zwłaszcza osoby początkujące, wolą rozpoczynać przygodę i poznawać nowe koncepcje *data science* w rodzimym języku. Mnogość zagadnień z pograniczna algebry liniowej, statystyki i probabilistyki, programowania, wizualizacji i analizy biznesowej bywa na tyle przytłaczająca, że nie zawsze osoby zainteresowane chcą równocześnie mierzyć się z nauką terminologii w języku angielskim, oraz zakresem merytorycznym.

To opracowanie jest odpowiedzią na przedstawione zapotrzebowanie. Forma interaktywnego Jupyter Booka dostępnego w sieci, pozwoli szybko wprowadzać zmiany i modyfikacje, zwłaszcza w odniesieniu do bibliotek i narzędzi, zmieniających się w błyskawicznym tempie.

Udanej lektury :)

Wstęp

Celem niniejszego opracowania jest stworzenie uporządkowanie i zaprezentowanie najważniejszych koncepcji z zakresu uczenia maszynowego i zaawansowanej analizy danych (ang. *data science*).

Z założenia, materiał ma być przeglądowy i przystępny dla osób, które nie miały wcześniej styczności z tematyką uczenia maszynowego, a także dla tych, którzy chcą uporządkować wiedzę rozproszoną w różnych źródłach.

W opracowaniu skupiono się głównie na praktycznych aspektach uczenia maszynowego, a omawiane zagadnienia poparto przykładami w języku Python. Gdzie to tylko możliwe - podano odsyłacze do materiałów źródłowych/przewodników/dokumentacji/tutoriali, które warto przeczytać, aby lepiej zrozumieć dany aspekt. Oczywiście, pojawią się też sekcje poświęcone np. teorii uczenia maszynowego czy szeregów czasowych - mają one na celu kompleksowe i całościowe przedstawienie danego tematu. Czytelnik lub Czytelniczka odnajdzie tu też szereg formalnych definicji, które starają się usystematyzować pojęcia powszechnie znane i stosowane w praktyce, ale niekoniecznie wyłożone w ustrukturyzowany sposób (np. czym różni się model od algorytmu uczenia maszynowego? Jak zdefiniować ograniczenia poznawcze modelu?).

Podręcznik ma charakter otwarty i płynny, co oznacza, że prace nad nim wciąż trwają, a ich efekty będą systematycznie udostępniane na niniejszej stronie.

Dalsze plany i zakres prac / „early access”

Niniejsze opracowanie jest w fazie rozwoju. W miarę postępów w pracach nad skryptem, będziemy dodawać nowe rozdziały, poprawiać błędy, uzupełniać treści. Docelowo, planowane jest pokrycie następujących tematów:

1. **Data science** - czym jest, z czego się składa, jakie są narzędzia, jakie są etapy procesu analizy danych.
2. **Praca z danymi** - rozdział poświęcony obróbce i przetwarzaniu danych. W szczególności:
    1. Typy i struktury danych - ustruktyzowane i nieustrukturyzowane, numeryczne, itd.
    2. Typowe operacje na danych - filtrowanie, grupowanie, łączenie, itd.
    3. Analiza poprawności i kompletności danych.
    4. Potoki przetwarzania (ang. pipelines) - automatyzacja procesu przetwarzania danych.
    5. Wizualizacja danych.
3. **Wprowadzenie do uczenia maszynowego** - rozdział omawiający niezbędne minimum teorii i notacji potrzebnej do zrozumienia tej dziedziny. W szczególności:
    1. Podstawowe pojęcia - model, cechy, etykiety, funkcja kosztu, itd.
    2. Elementy teorii COLT - elementy teorii uczenia obliczeniowego i generalizacji.
    3. Taksonomia modeli uczenia maszynowego - modele liniowe, drzewiaste, sieci neuronowe, itd.
4. **Projektowanie eksperymentów** - w jaki sposób projektować i przeprowadzać eksperymenty związane z uczeniem maszynowym. W szczególności:
    1. Podział danych na zbiory treningowy, walidacyjny i testowy.
    2. Ocena modeli - metryki jakości, walidacja krzyżowa, itd.
    3. Optymalizacja hiperparametrów.
    4. Analiza wyników eksperymentów.
5. **Modele uczenia maszynowego** - rozdział poświęcony praktycznemu zastosowaniu modeli uczenia maszynowego. W szczególności:
    1. Regresja - modele liniowe, drzewiaste, sieci neuronowe.
    2. Klasyfikacja - modele liniowe, drzewiaste, sieci neuronowe.
    3. Grupowanie - k-means, DBSCAN, itd.
    4. Klasyfikacja wieloklasowa.
    5. Regresja wielowymiarowa.
    6. Modele sekwencyjne - sieci rekurencyjne, sieci splotowe.
    7. Modele generatywne - sieci GAN, VAE.
6. **Budowanie projektów data science** - w jaki sposób konstruować projekty DS. W szczególności:
    1. Metoda CRISP-DM zarządzania projektami.
    2. Szablony projektów DS i rozwiązań technicznych.

7. **Narzędzia i zaplecze techniczne Data Science** - czyli o technikach, narzędziach i rozwiązaniach, które warto znać, aby płynnie poruszać się po projektach DS. W szczególności:
    1. Środowiska pracy - Visual Studio Code, Jupyter, Google Colab, Lightning Studio, etc.
    2. Git, jako podstawowe narzędzie kontroli wersji.
    3. Przydatne biblioteki w Pythonie - czyli które trzeba znać obowiązkowo, które ułatwiają pracę, automatyzują zadania, itd.
    4. Docker o Kubernetes - narzędzia do konteneryzacji aplikacji.
    5. Kedro, DVC, MlFlow - narzędzia do zarządzania eksperymentami, danymi i modelami.
    6. MLOPS - szersze spojzrenie na to, jak zarządzać cyklem życia modeli uczenia maszynowego.

Opracowanie całego tego materiału zajmie oczywiście wiele czasu. Rozdziały, podrozdziały i sekcje będą udostępniane w miarę ich pojawiania się

Autorzy

Autorami niniejszego opracowania są:

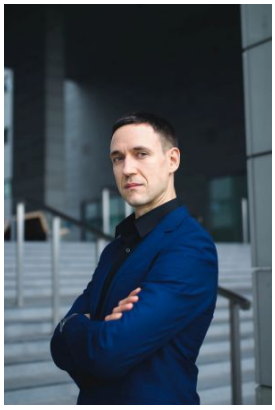
prof. UEW, dr. hab. Helena Dudycz

Ekspertka w dziedzinie Informatyki Ekonomicznej i systemów informatycznych w zarządzaniu. Specjalizuje się w zaawansowanej analizie danych, wizualizacji informacji i wiedzy ekonomicznej oraz użyteczności interfejsów człowiek-komputer. **Kierownik Katedry Technologii Informacyjnych na Wydziale Zarządzania Uniwersytetu Ekonomicznego we Wrocławiu**, gdzie od lat kształci kolejne pokolenia specjalistów Data Science. Pełniła również funkcję prodziekana ds. studenckich. Zaangażowana w liczne projekty badawczo-rozwojowe, współpracując z firmami w zakresie efektywności przedsięwzięć informatycznych.

dr Ryszard Zygała

Główny pomysłodawca i inspirator powstania niniejszego projektu. Analityk danych z ponad trzydziestoletnim doświadczeniem, praktyk systemów Buisness Intelligence, od ponad dziesięciu lat wykorzystujący narzędzia z ekosystemu Pythona w swoich badaniach. **Kierownik studiów podyplmowych Data Science** na Uniwersytecie Ekonomicznym we Wrocławiu oraz, do niedawna, kierownik i pomysłodawca międzynarodowego projektu Data Science Summer School.

dr Filip Wójcik



Analityk danych, zajmujący się problemami uczenia maszynowego od 2010 roku. W tym czasie pracował dla międzynarodowych firm, takich jak NOKIA, Credit Suisse i Capgemini jako konsultant ds. uczenia maszynowego w różnych obszarach biznesowych, kierownik naukowy lub lider techniczny. Łączy zaangażowanie komercyjne z pracą akademicką - związany z Uniwersytetem Ekonomicznym we Wrocławiu, gdzie specjalizujące się w optymalizacji procesów podejmowania decyzji za pomocą algorytmów uczenia maszynowego, w szczególności głębokich sieci neuronowych. Aktywny wykładowca, chętnie dzielący się swoją wiedzą i doświadczeniem ze studentami na różnych poziomach. W niniejszym projekcie nadzoruje stronę inżynieryjną, wdrożenie i redakcję całości. Strona internetowa: filip-wojcik.com

Współpraca

Ten projekt jest tworzony w formacie *open-source* i udostępniany za darmo dla każdej osoby zainteresowane tematyką DS. Będziemy wdzięczni za wszelkie uwagi, poprawki, sugestie, które pomogą nam ulepszyć skrypt.

Na tym jednak nie koniec.

Jeśli chcesz wziąć udział w tworzeniu tego skryptu, zapraszamy do współpracy. Możesz pomóc w różny sposób:

1. **Poprawiając błędy** - jeśli zauważysz jakieś nieścisłości, błędy merytoryczne, literówki, itp., daj nam znać. Możesz to zrobić poprzez zgłoszenie błędu (ang.*issue*) na GitHubie, albo poprzez zgłoszenie *pull request* z poprawkami.
2. **Dodając nowe treści** - jeśli masz pomysł na dodanie nowego rozdziału, sekcji, przykładu, itp., również daj nam znać. Możesz to zrobić poprzez zgłoszenie *issue* na GitHubie, albo poprzez zgłoszenie *pull request* z nowymi treściami.
3. **Poprawiając styl** - jeśli masz doświadczenie w redagowaniu tekstów, możesz pomóc nam poprawić styl, zwiększyć czytelność, itp. Wszelkie sugestie mile widziane.

Dodawanie treści do projektu

Niniejsze opracowanie jest tworzone z pomocą biblioteki Jupyter Book i umieszczony na GithubPages. Dodawanie nowych treści / nanoszenie poprawek itd. jest możliwe poprzez zgłoszenie *pull request* na GitHubie.

Prosilibyśmy o trzymanie się kilku zasad, które ułatwią nam wspólną pracę:

1. **Dodanie zależności za pomocą biblioteki Poetry** - każdy, kto pracował z Pythonem wie, jak ciężko jest zarządzać w nim zależnościami bibliotek. W tym projekcie zdecydowaliśmy się na **Poetry**, które stara się w sposób spójny utrzymywać wersje i powiązane biblioteki. Jeśli dodajesz nowy notebook lub fragmenty wykonywalnego kodu - upewnij się, że dodajesz odpowiednie zależności do pliku `pyproject.toml`. Więcej o Poetry i jego zastosowaniach: https://python-poetry.org/.

2. **Mieszanie treści tekstowych i notebooków** - to opracowanie zapewne mogłoby być książką wydaną tradycyjnie, jednak zdecydowaliśmy się na format JuypterBook, aby dołączyć kod, treści interaktywne, obrazy itd. Jeśli to możliwe - dodając nowe fragmenty umieść w nich zarówno tekst, jak i kod, aby zilustrować opisywane koncepcje.
3. **Równania, dowody, definicje, obrazki** - chcemy, żeby to opracowanie pozwalało na dynamiczną nawigację między elementami. Kluczowe pojęcia powinny być dodawane do słownika (ang. *glossay* w JupyterBook), równania/dowody/obrazy - powinny być dodawane w formie umożliwiającej ich indeksowanie i budowanie odsyłaczy. Więcej o tym w dokumentacji JupyterBook.
4. **Cytowania** - wszystkie treści będące odwołaniem do literatury naukowej lub istniejących opracowań powinny być cytowane, a odpowiednie materiały źródłowe - dodane do pliku `references.bib`. Więcej o tym w dokumentacji JupyterBook.

Będziemy bardzo wdzięczni za wszelką pomoc przy realizacji projektu!

Jak wprowadzać zmiany do projektu?

Projekt jest otwarty na współpracę i każde wsparcie jest mile widziane. W celu zapewnienia płynnej i zorganizowanej pracy nad podręcznikiem, prosimy o stosowanie się do poniższych zasad:

1. Forkowanie repozytorium (*fork*)

Aby rozpocząć współpracę, należy najpierw wykonać *fork* repozytorium na GitHubie. Oznacza to utworzenie własnej kopii projektu na Twoim koncie GitHub, gdzie możesz wprowadzać zmiany bez wpływu na główną wersję. Po zalogowaniu się na swoje konto GitHub, przejdź do strony repozytorium i kliknij przycisk „Fork" w prawym górnym rogu.

Tutorial:

- Fork a Repo - GitHub Docs

2. Tworzenie nowej gałęzi (*branch*)

Przed wprowadzeniem jakichkolwiek zmian w swoim *forku*, zalecamy utworzenie nowej gałęzi (*branch*), która będzie zawierała Twoje zmiany. Dzięki temu Twoje modyfikacje będą odseparowane od głównej gałęzi (*main*) i łatwiej będzie je przetestować i przejrzeć. Możesz nazwać swoją gałąź odpowiednio do rodzaju zmian, np. `poprawki-stylistyczne` lub `nowa-sekcja-uczenie-maszynowe`.

Tutorial:

- Creating and Deleting Branches - GitHub Docs

3. Wprowadzanie zmian i tworzenie commitów (*commit*)

Wprowadź zmiany, które uważasz za konieczne, na nowo utworzonej gałęzi. Po zakończeniu pracy, zapisz zmiany w repozytorium lokalnym za pomocą commitów. Każdy *commit* powinien być opisany w sposób zwięzły, ale informatywny, np. „Dodano nową sekcję o regresji liniowej" lub „Poprawiono literówki w rozdziale 3".

Tutorial:

- About commits - GitHub Docs

4. Zgłoszenie Pull Request (*pull request*)

Gdy Twoje zmiany są gotowe do włączenia do głównego repozytorium, zgłoś *pull request* z gałęzi, na której pracowałeś/aś. *Pull request* to prośba o włączenie Twoich zmian do głównej gałęzi (*main*) projektu. Przed wysłaniem *pull requestu* upewnij

się, że Twój kod jest zgodny ze standardami projektu i został przetestowany. W opisie *pull requestu* jasno określ, jakie zmiany zostały wprowadzone i dlaczego.

Tutorial:

- About Pull Requests - GitHub Docs

5. Wyznaczenie recenzenta (*reviewer*)

Po zgłoszeniu *pull requestu*, prosimy o wyznaczenie recenzenta (*reviewer*), który sprawdzi Twoje zmiany. Recenzentem może być jeden z członków zespołu projektowego lub inna osoba, która zna się na danym temacie. Recenzent dokona przeglądu zmian, zgłosi ewentualne uwagi i zatwierdzi (lub odrzuci) zmiany. Dopiero po akceptacji przez recenzenta Twoje zmiany zostaną połączone z główną gałęzią.

Tutorial:

- Requesting a Pull Request Review - GitHub Docs

6. Dokumentacja i zasoby

Zachęcamy do zapoznania się z dokumentacją Jupyter Book oraz z narzędziami używanymi w projekcie, takimi jak Poetry. Dzięki temu Twoje zmiany będą w pełni zgodne ze standardami projektu i łatwiejsze do zintegrowania.

Będziemy bardzo wdzięczni za wszelką pomoc i wkład w rozwój tego projektu! Razem możemy stworzyć wartościowy materiał edukacyjny dostępny dla wszystkich.

Notacja i oznaczenia [1]

W dyskusjach dotyczących *data science* i uczenia maszynowego, bardzo ważne jest ujednolicenie notacji i stosowanych oznaczeń. Nierzadko zdarza się, że różne źródła naukowe, czy też praktycy, stosują różne symbole dla tych samych pojęć. W celu uniknięcia nieporozumień, warto zdefiniować zestaw oznaczeń, które będą stosowane w dalszej części tego skryptu.

Język polski vs język angielski

Dominacja języka angielskiego w (szeroko rozumianej) branży IT jest faktem, z którym trudno się spierać. Od lat toczą się debaty między zwolennikami tłumaczenia wszelkich terminów na język polski, nawet jeśli te nie przyjęły się jeszcze w codziennym użyciu, a entuzjastami pozostawiania oryginalnej nomenklatury. Nie inaczej jest w przypadku głównego tematu tego podręcznika.

Sama nazwa *data science* bywa tłumaczona na różne sposoby – najpopularniejszy to zapewne *danologia*, a bardziej opisowy wariant to *zaawansowana analiza danych*. Każdy z tych terminów ma swoje plusy i minusy, ale nie będziemy tu próbować rozstrzygać tej odwiecznej batalii.

Niniejszy podręcznik ma przede wszystkim służyć jako praktyczny przewodnik. Jego autorzy nie są wprawdzie ekspertami językowymi, ale za to mają doświadczenie w pracy w sektorze komercyjnym, często w międzynarodowych firmach. Nie popieramy bezmyślnego stosowania językowych kalek ani humorystycznie karykaturalnego “korpo-lengłydżu” („fokusujemy się na targecie, czekając na approvale tasków na tym ekałncie”), ale równie niekorzystne, naszym zdaniem, jest używanie polskich neologizmów, które mają zerową praktyczną użyteczność i rozpoznawalność.

Dlatego w tym podręczniku podajemy polskie terminy, o ile istnieją w powszechnie uznanej wersji, zawsze obok ich oryginalnych odpowiedników po angielsku. Jeśli polski odpowiednik nie istnieje, jest mało znany lub nie ma zgody co do jego stosowania – używamy jedynie oryginalnej nazwy, zapisanej kursywą.

Nasze założenie jest proste: Czytelnik lub Czytelniczka tego materiału najczęściej będą mieli do czynienia z terminologią anglojęzyczną – czy to przy przeglądaniu dokumentacji bibliotek, czy też współpracując z zagranicznymi partnerami. Osoba

rozwiązująca problem w Pythonie z pomocą modułu `scikit-learn` raczej nie będzie używać terminu „przeszukiwanie siatki parametrów”, tylko *grid search*. Dlatego też taki zapis będziemy stosować: przeszukiwanie siatki parametrów (ang. *grid search*).

Zachęcamy więc do odkrywania zagadnień **zaawansowanej analizy danych**, czyli *data science* – w duchu praktycznym, ale bez nadmiernego zapętlenia się w językowe rozważania.

Skalary i wektory

- Skalar oznaczamy małą literą zwykłą, np. x lub wielką, niepogrubioną (zwł. w przypadku wymiarowości macierzy lub liczności elementów zbioru).
- Wektor oznaczamy małą literą zwykłą z pogrubieniem, np. $\mathbf{x}$.
- Macierze oznaczamy dużą literą z pogrubieniem, np. $\mathbf{X}$.
- Wymiarowość macierzy lub wektorów będzie oznaczana słowem dim np. $\dim(\mathbf{X}) : N \times K$ lub w notacji przynależności do domeny np. $\mathbf{X} \in \mathbb{R}^{N \times K}$.
- Krotki / n-tki (ang. *tuple*) oznaczamy symbolami zgodnymi z ich zawartością (np. macierze, wektory, skalary), w okrągłym nawiasie: $(x_1, x_2, \ldots, x_n)$.
- Zbiory oznaczamy wielkimi literami z frakturą/ręcznym pismem np. $\mathcal{X}$.
- Klasyczne zbiory / domeny takie jak liczny naturalne i rzeczywiste oznaczane są podwójnymi, wielkimi literami: $\mathbb{N}$, $\mathbb{R}$.

Operacje na macierzach

Niejednokrotnie będzie stosowany zapis charakterystyczny dla bibliotek i narzędzie w Pythonie, związanych z indeksowaniem macierzy:

- $\mathbf{X}^T$ - transpozycja macierzy $\mathbf{X}$.
- $\mathbf{X}^{-1}$ - odwrotność macierzy $\mathbf{X}$.
- a_i - i-ty element wektora $\mathbf{a}$.
- $\mathbf{X}_{i,j}$ - element macierzy $\mathbf{X}$ w i-tym wierszu i j-tej kolumnie.
- $\mathbf{X}_{i,:}$ - i-ty wiersz macierzy $\mathbf{X}$. Taki zapis bezpośrednio koresponduje z notacją w Pythonie, w bibliotece `NumPy`.
- $\mathbf{X}_{:,j}$ - j-ta kolumna macierzy $\mathbf{X}$. Taki zapis bezpośrednio koresponduje z notacją w Pythonie, w bibliotece `NumPy`.
- $\mathbf{X}_{i:j,k:l}$ - wycinek macierzy $\mathbf{X}$ od wiersza i do j i od kolumny k do l.

Funkcje

- Funkcje oznaczamy małą literą zwykłą, np. f.
- $f : \mathbb{A}^N \to \mathbb{B}^M$ - funkcja f o dziedzinie $\mathbb{A}^N$ i przeciwdziedzinie $\mathbb{B}^M$.
- $f(x)$ - wartość funkcji f w punkcie x.
- $f \circ g$ - złożenie funkcji f i g.
- $f(\mathbb{x}, \theta)$ - funkcja f wektora $\mathbb{x}$ i parametrów θ.

---

[1] Autor sekcji: dr Filip Wójcik

Wprowadzenie do Uczenia Maszynowego [1]

Uczenie maszynowe (ML, *ang* Machine learning) jest bardzo szeroką dziedziną, łączącą w sobie elementy:

1. statystyki,
2. matematyki,
3. informatyki,

4. a nawet psychologii.

Co więcej, w ostatnich latach wykształciły się w jej ramach pod-dziedziny poświęcone między innymi:

1. Uczeniu głębokiemu i sieciom neuronowym,
2. Wielkim modelom językowym (ang. LLM),
3. Uczeniu ze wzmocnieniem (ang. RL) za pomocą symulatorów i (bardzo szeroko rozumianym) gier (np. giełdowych, optymalizacyjnych).
4. Uczeniu grafowemu i analizom sieci (w tym społecznościowych).

Płynne poruszanie się po zagadnieniach ML wymaga znajomości między innymi:

1. Podstaw algebry liniowej -w zakresie wymaganym do zrozumienia działania modeli liniowych, sieci neuronowych, operacji na macierzach.
2. Podstaw statystyki i probabilistyki - w zakresie zrozumienia próbkowania, kształtowania eksperymentów, oceny jakości modeli.
3. Podstaw teorii uczenia obliczeniowego COLT - w zakresie zrozumienia generalizacji, złożoności modeli, zbieżności algorytmów.
4. Znajomości specyfiki poszczególnych rodzin modeli, sposobów szkolenia, dobrych praktyk, itd.

Częstym problemem dla osób rozpoczynających przygodę w tej dziedzinie jest z jednej strony ogromna ilość dostępnych materiałów, z drugiej kwestia wybiórczości podejścia. Wiele osób, które chcą zapoznać się np. z grafowymi sieciami neuronowymi napotyka na kaskadę problemów. Najpierw należałoby zacząć od przypomnienia sobie, czym są grafy i z czego się składają. Grafowe sieci neuronowe korzystają z całego aparatu matematycznego i mechaniki sieci klasycznych - też należałoby je znać. Na grafach wykonuje się typowo zadania takie jak: klasyfikacja lub regresja dla wierzchołków, klasyfikacja lub regresja dla połączeń, klasyfikacja lub regresja dla całych grafów. Jeśli ktos nie zna zagadnień klasyfikacji lub regresji - trzeba doczytać czym są, jakich metryk się używa.Materiały o metrykach z kolei odwołują się do pojęć statystycznych, próbkowania, estymacji, walidacji krzyżowej itd.W efekcie, osoba, która planowała nauczyć się czegoś o sieciach neuronowych , musi najpierw zapoznać się z całym bagażem pokrewnych zagadnień.

Kolejną kwestią jest kompletność wiedzy. Wyrywkowe zapoznanie się ze specyfiką poszczególnych modeli, bez zrozumienia ich teoretycznych podstaw, może prowadzić do sytuacji, w której osoba, która zna np. sieci neuronowe lub drzewa decyzyjne, ale potrafi je stosować jedynie w prostych, standardowych przypadkach, bez zapewnienia rzetelnej statystycznej ewaluacji błędów, mocy modelu, etc.

W tym opracowaniu postanowiliśmy zacząć od podstaw, budując na tym fundamencie kolejne piętra budynku. I tak, Czytelnik lub Czytelniczka będzie się zapoznawać kolejno:

1. Z podstawowymi pojęciami i terminami, charakterystycznymi dla uczenia maszynowego.
2. Z podstawami teorii COLT, czyli teorii uczenia obliczeniowego.
3. Z taksonomią zadań realizowanych przez różne modele.
4. Z podstawowymi zagadnieniami z zakresu błędów, ograniczeń modeli oraz projektowaniem eksperymentów dla nich.
5. Z przekrojem przez najważniejsze rodziny modeli uczenia maszynowego.
6. (i dopiero potem) Ze szczegółami dotyczącymi po kolei każdej z wielkich rodzin modeli uczenia maszynowego, w tym:
    1. Klasyczne modele drzew decyzyjnych.
    2. Modele minimalnoodległościowe (KNN, etc.).
    3. Modele typu „ensemble” czyli zespoły połączonych estymatorów.
    4. Podstawy sieci neuronowych.
    5. Podstawowe modele dla szeregów czasowych.

Mamy nadzieję, że taka prezentacja zagadnień, nawet w formie hasłowej i skróconej, ułatwi nawigowanie w skomplikowanym świecie analizy danych :)

---

[1] Autor sekcji: dr Filip Wójcik.

Kluczowe pojęcia [1]

Ten podrozdział poświęcono kluczowym pojęciom z zakresu uczenia maszynowego. Obejmuje on definicje i wyjaśnienia terminów, które są niezbędne do zrozumienia dalszych zagadnień.

Uczenie się

Dla zrozumienia koncepcji uczenia maszynowego, warto zacząć od podstaw - od zdefiniowania samego zjawiska nauki. Aspekt ten jest i był przedmiotem badań w ramach różnych dziedzin - od psychologii po informatykę. Poniższe zestawienie przedstawia kilka wybranych definicji z rozmaitych źródeł.

Herbert Simon | **Paweł Cichosz** | **Tom Mitchell**

Uczenie się oznacza takie zmiany adaptacyjne w systemie, iż jest on w stanie lepiej wykonać w przyszłości takie same zadania lub zadania należące do tej samej kategorii. [Sim83]

W powyższych definicjach na pierwszy plan wysuwają się kluczowe pojęcia, takie jak dostosowywanie działania, gromadzenie doświadczenia, czy wreszcie poprawa jakości działania. Mając na uwadze formę przytoczonych definicji, dopuszczalne jest zdefiniowanie, na nasze potrzeby, uczenia w sposób następujący:

Uczenie się

jest to proces będący ciągiem (sekwencją) zachodzących zmian, którego głównymi składowymi są:

1. możliwość gromadzenia doświadczenia w miarę wykonywania zadań;
2. posiadanie przez system wewnętrznego stanu wiedzy i doświadczenia, aktualizowanego w miarę powtarzania zadanych czynności;
3. funkcja lub procedura oceniająca jakość wykonywanych zdań;
4. wewnętrzny mechanizm reagowania na pozytywny/negatywny sygnał zwrotny, opisujący jakość wykonanego zadania. [Woj21]

Indukcja i dedukcja

Uczenie indukcyjne i dedukcyjne to dwa najważniejsze paradygmaty prowadzenia rozumowań.

Uczenie dedukcyjne

Pojęcie to odnosi się do klasy systemów uczących się na bazie zestawu założeń i koncepcji, co do których zachodzi założenie (możliwe do dowiedzenia), że są poprawne. Tego typu systemy stosowane są zwykle do przyspieszania i wspomagania działania innych narzędzi, poprzez uproszczenie procesu wnioskowania. [tł. wł.] [SW17]

Dedukcja Sherlocka Holmesa

Kto czytał lub zna historie o Sherlocku Holmesie, ten na pewno będzie kojarzyć rozumowanie dedukcyjne właśnie ze słynnym detektywem.

Uczenie indukcyjne

Rozumowanie indukcyjne jest odwróceniem dedukcji - zachodzi, gdy obserwowane dane uznamy za konkluzje, będące następstwami nieznanych faktów. Algorytm uczy się na ich podstawie, próbując odnaleźć lub skonstruować mechanizmy wyjaśniające zastany stan rzeczy [Woj21]. Innymi słowy - model stara się odnaleźć uogólnione reguły, na podstawie zaobserwowanych, przykładowych danych [tł. wł.] [SW17]

Podobieństwa i różnice pomiędzy tymi podejściami możemy podsumować w tabeli poniżej:

	Uczenie dedukcyjne	Uczenie indukcyjne
Cel	Wykorzystywanie teorii i wiedzy początkowej do formułowania konkluzji.	Tłumaczenie zjawisk poprzez generalizację przykładów
Uzyskanie predykcji	Wnioskowanie od przesłanek i założeń do konkluzji, przy założeniu, że przesłanki i reguły wnioskowania są prawdziwe.	• Poszukiwanie ogólnych reguł na podstawie przykładów • Analiza statystyczna i ilościowa zjawisk.
Słabości	Wymóg zdefiniowania przesłanek a priori oraz posiadania wyczerpującego opisu mechanizmów	• Obarczone błędami wynikającymi z przykładów. • Wymaga (relatywnie) dużych ilości danych.
Typowe zastosowania	Objaśnianie niewielkich zbiorów danych, za pomocą ugruntowanej teorii	Objaśnianie zbiorów danych, gdy nie ma jasno sformułowanej teorii i opisanych mechanizmów wnioskowania.

Jak łatwo się domyślić - w otaczającym nas świecie rzadko kiedy mamy podane na tacy teorie i wyjaśnienia zjawisk, których doświadczamy.Z tego względu, uczenie indukcyjne jest częściej stosowane w praktyce, ze względu na możliwość wykorzystania danych do wygenerowania modelu.

Tu kryje się jednak pułapka: wnioskowanie indukcyjne jest tylko tak dobre, jak dane, na których się opiera. W związku z tym, kluczowe jest, aby dane były reprezentatywne i niezafałszowane.

GIGO

Warto zapamiętać zasadę, którą można streścić w skrócie jako GIGO (Garbage In, Garbage Out) - jeśli dane wejściowe są złe, to wyniki będą złe.

Porównywanie indukcji i dedukcji czasem ociera się o dyskusję filozoficzną. Kto nam bowiem zagwarantuje, że prawa fizyki albo matematyka są nienaruszalnymi aksjomatami: w końcu też je zaobserwowano, albo wynaleziono eksperymentalnie. Warto jednak pamiętać, że w praktyce, zazwyczaj mamy do czynienia z pewnymi ustalonymi prawami, które przyjmujemy jako punkt wyjścia.

Ćwiczenie - dedukcja vs indukcja

Zastanów się, jakie z poniższych zdań są przykładami dedukcji, a które indukcji:

- **Widzę gotującą się wodę w garnku. Mogę opisać i uzasadnić proces wrzenia wody.**
- **Wszystkie ptaki, które widziałem w moim mieście mają skrzydła. Wnioskuję, że wszystkie ptaki mają skrzydła.**
- **Ludzie, którzy jedzą dużo owoców, są zdrowsi. Wnioskuję, że jedzenie owoców jest zdrowe.**
- **Klient otrzymał kredyt zgodnie z procedurą. Należy sprawdzić, czy spełnione zostały wszystkie warunki.**
- **Ludzie chorują z powodu nowego wirusa. Należy ustalić u kogo występują powikłania.**

Dane wykorzystywane w uczeniu

Uczenie maszynowe nie istnieje bez zbiorów danych. Formalnie, możemy sobie zdefiniować zbiór danych i jego składowe w następujący sposób:

Dziedzina

dziedziną nazywamy oznaczany przez $\mathcal{X}$ zbiór obiektów, których dotyczyć ma wiedza nabywana przez model. Mogą to być przedmioty, osoby, wydarzenia, sytuacje, etc. [Cic07], [Lea97]

Przykłady

każdy obiekt, element dziedziny $x \in \mathcal{X}$ nazywamy przykładem. W praktyce, obiekty te są opisywane za pomocą swojego zbioru cech, najczęściej wyrażanych w postaci macierzy lub wektora. [Cic07], [Lea97], [Woj21]

Próbka ucząca

jest zbiorem par $(\mathbf{x}, y)$, gdzie $\mathbf{x} \in \mathcal{X}$ to wektor cech przykładu, a y to etykieta (numeryczna lub dyskretna). W przypadku uczenia nadzorowanego, etykieta y jest zazwyczaj wartością, którą model ma przewidzieć. W przypadku uczenia nienadzorowanego, próbka ucząca składa się z samych wektorów cech $\mathbf{x}$. Formalnie, w postaci zbioru, możemy ją zapisać jak w równaniu (1).

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n) \mid \forall \mathbf{x} \in \mathcal{X}, y \in \mathcal{Y}\} \tag{1}$$

W postaci macierzowej ma postać, jak w równaniu (2).

$$D = (\mathbf{X}, \mathbf{y}), \mathbf{X} \in \mathbb{R}^{n \times m}, \mathbf{y} \in \mathbb{R}^n \tag{2}$$

gdzie, n jest ilością próbek, m jest ilością cech, $\mathcal{X}$ to przestrzeń cech, a $\mathcal{Y}$ to przestrzeń etykiet. [Woj21]

Wszystko to brzmi w teorii dużo bardziej skomplikowanie niż jest w rzeczywistości. W przypadku danych tabelarycznych, próbka ucząca się strukturą danych przypominającą tabelę w bazie, albo arkusz Excela, w którym:

1. Każdy wiersz jest obiektem (np. osobą / klientem).
2. Każda kolumna reprezentuje cechę obiektu / atrybut (np. wiek, płeć, dochód).
3. Jedna z kolumn jest etykietą, którą model ma przewidzieć (np. czy klient kupi produkt).

Spójrzmy na poniższy przykład: mamy tabelę tabelę z filmami, opisanymi za pomocą pewnych cech. Ostatnia kolumna - „ocena Filmweb.pl" - podaje średnią ocen z portalu.

Tabela 1 Filmy

Tytuł	Gatunek	Rok produkcji	Ocena Filmweb.pl
Matrix	SF / cyberpunk	1999	7.6
Pulp Fiction	Kryminał	1994	8.3
Titanic	Dramat	1997	7.3
Ghost in the shell	SF / cyberpunk	1995	7.9

Przekładając to na formalną notację zdefiniowaną równaniem (2) i zakładając przez chwilę, że zamienilibyśmy etykiety tekstowe np. `SF/cyberpunk` na liczby naturalne, otrzymamy:

$$D = (\mathbf{X}, \mathbf{y}), \mathbf{X} \in \mathbb{N}^{4 \times 3}, \mathbf{y} \in \mathbb{N}^4$$

mając cztery wiersze z filmami o trzech atrybutach (tytuł, gatunek, rok produkcji), oraz wektor 4 etykiet (ocen).

Algorytmy i modele uczenia maszynowego

Bardzo ważne jest wprowadzenie rozróżnienia pomiędzy algorytmami i modelami uczenia maszynowego. O dziwo, jest to pytanie, które dość często pojawia się na rozmowach kwalifikacyjnych, ale zagadnienie jest warte zbadania również z innych powodów.

Algorytm uczenia maszynowego

> pod nazwą algorytm uczenia maszynowego (lub skrótowo: algorytm bądź procedura) funkcjonować będzie określona procedura, będąca instrukcją wykonania kolejnych kroków, mających na celu nauczenie systemu komputerowego rozpoznawania wzorców, w ramach postawionego zadania klasyfikacji, regresji, grupowania czy odkrywania zależności. Procedura ta ma postać zapisu (symbolicznego, słownego, matematycznego lub hybrydowego, w postaci pseudokodu) kolejnych czynności. Algorytm jest zatem niezależny od kontekstu, w jakim się go wykonuje. [Woj21]

Model uczenia maszynowego

> pod nazwą model uczenia maszynowego (lub skrótowo: model) funkcjonować będzie instancja (w rozumieniu informatyki: konkretny egzemplarz) algorytmu, zastosowany do zadanego problemu, w celu utworzenia uproszczonego opisu wycinka rzeczywistości. Proces ten nosi nazwę szkolenia, uczenia lub trenowania27 i odbywa się na podstawie dostępnych danych, mając najczęściej postać procesu indukcyjnego. Jest to zatem algorytm o ukształtowanym i ustalonym wewnętrznym stanie – konkretyzacja abstrakcyjnie i symbolicznie zdefiniowanej procedury. [Woj21]

> **Model uczenia maszynowego - jeszcze krótsza definicja**
>
> W tym kontekście można podsumować model uczenia maszynowego jeszcze prościej: jest to uproszczony opis pewnego wycinka rzeczywistości, skonstruowany przez model, na bazie dostępnych danych.

Niejednokrotnie możemy się spotkać z formalną definicją modelu, jako funkcji, która przyjmuje na wejściu dane oraz parametry/wewnętrzny stan i zwraca predykcję. Możemy to zapisać w postaci równania (3):

$$\hat{y} = f(\mathbf{X}, \theta) \tag{3}$$

Parametry modelu są przedmiotem szkolenia, zwanego także treningiem lub strojeniem/tuningiem.

---

[1] Autor sekcji: dr Filip Wójcik.

Elementy teorii uczenia się [1]

W niniejszym rozdziale przedstawione zostaną najważniejsze zagadnienia teoretyczne powiązane z teorią COLT - ang. *Computational Learning Theory*. Jest to dziedzina zajmująca się badaniem algorytmów uczenia maszynowego, ich złożoności obliczeniowej oraz ich zdolności do generalizacji. W ramach tego rozdziału omówione zostaną kluczowe elementy takie jak przestrzeń hipotez, pojecie docelowe i generalizacja.

Pojęcie docelowe i hipotezy

W przypadku uczenia maszynowego, tworząc i szkoląc poszczególne modele, poruszamy się w obszarze dużej niepewności. Warto sobie zadać następujące pytania:

> **Pytania do procesu uczenia się**
>
> 1. Skąd mamy gwarancję, że dane, na których uczymy model, są reprezentatywne dla całej populacji?
> 2. Skąd mamy gwarancję, że etykiety (y) przypisane do danych są poprawne i reprezentatywne dla całej populacji?
> 3. Skąd wiemy, czy i jaki proces odpowiada w rzeczywistości za przypisywanie etykiet do danych?

Wbrew pozorom, są to bardzo istotne uwagi. Nierzadko zdarza się, że nie wiemy, czy istnieje **jakakolwiek** zasada, przypisująca etykiety do danych. W takim przypadku, model, który uczymy, może być jedynie zbiorem przypadkowych reguł, które nie mają żadnego związku z rzeczywistością.

Wyobraźmy sobie przez chwilę **przestrzeń wszystkich możliwych cech obiektów**, oznaczaną jako $\mathcal{X}$ - jest to wyczerpująca lista wszelkich kombinacji cech, jakie tylko mogą wystąpić w naszych danych. W przypadku filmów, mogą to być takie cechy jak: gatunek, rok produkcji, reżyser, aktorzy, budżet, długość filmu, itd.

Obiekty w tej przestrzeni mają przypisane pewne etykiety - oznaczane jako $\mathcal{Y}$. W przypadku filmów, mogą to być takie etykiety jak: ocena Filmweb.pl, ocena IMDB, liczba nominacji do Oscara, itd.

Pojęcie docelowe

Zakładamy, że istnieje pewne wyjaśnienie - proces, funkcja, zjawisko (naturalne lub sztuczne), które przypisuje etykiety do obiektów. Tę funkcję nazywamy **funkcją docelową albo pojęciem docelowym** (ang. *target function*), oznaczaną jako $c(x) : \mathcal{X} \rightarrow \mathcal{Y}$. [Cic07] [Woj21]

Funkcja docelowa

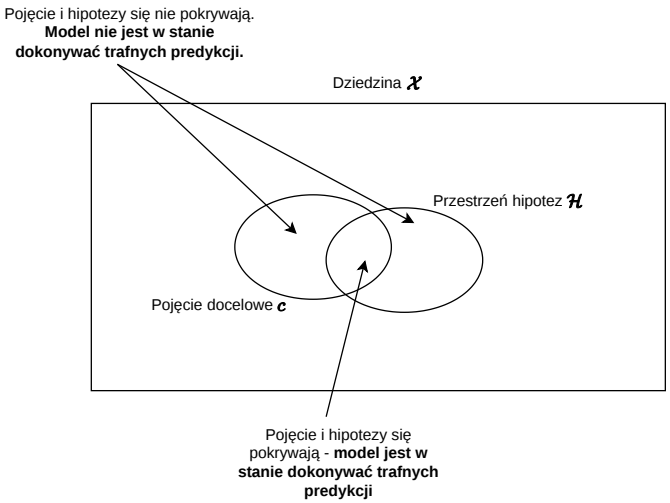
Niektórzy autorzy (np. [AMMIL12]) posługują się terminem **funkcja docelowa** zamiast **pojęcie docelowe**. Funkcja docelowa to funkcja, która przypisuje etykiety do obiektów z zastrzeżeniem, że jest ona **zaszumiona** - może zawierać błędy, wynikające np. ze złych pomiarów, niedokładności zapisów, niedoprecyzowania pojęć, etc. W kontekście uczenia maszynowego, jest to funkcja, którą staramy się odnaleźć. Funkcję docelową oznacza się jako $f(x) : \mathcal{X} \rightarrow \mathcal{Y}$ [AMMIL12]

Przestrzeń hipotez

Jest to zbiór wszystkich możliwych do wyrażenia przez określony model potencjalnych wyjaśnień danych, mających na celu odnalezienie Pojęcie docelowe. [Cic07] [Woj21]

Algorytm uczenia maszynowego, którego używamy, żeby wyszkolić Model uczenia maszynowego stara się stworzyć **uproszczony opis danych, które ma dostępne** w taki sposób, aby jak najlepiej odwzorować **pojęcie docelowe**. W tym kontekście, mówimy o **hipotezach**: każdy algorytm może tworzyć/reprezentować różne **hipotezy**, starając się wyjaśnić zależności w danych.

Pytanie brzmi: **czy przestrzeń hipotez generowana przez algorytm zawiera w sobie pojęcie docelowe?**



Rys. 1 Ilustracja przestrzeni hipotez. Adaptacja za: [Lea97]

Zewnętrzny, duży kwadrat przedstawia całą dziedzinę - przestrzeń wszystkich możliwych wartości atrybutów. Pojęcie docelowe, czyli funkcja, którą chcemy znaleźć, pokrywa część tego obszaru (owal z lewej). Model, tworzony przez określony algorytm może tworzyć różne hipotezy, które są reprezentowane przez owal po prawej stronie. Jak widać - czasem zdarza się, że przestrzeń hipotez nie pokrywa się z pojęciem docelowym: wówczas model „nie trafia" z odpowiedziami. W innych wypadkach - przestrzeń jego hipotez pokrywa się z pojęciem docelowym i wówczas otrzymujemy prawidłowe odpowiedzi.

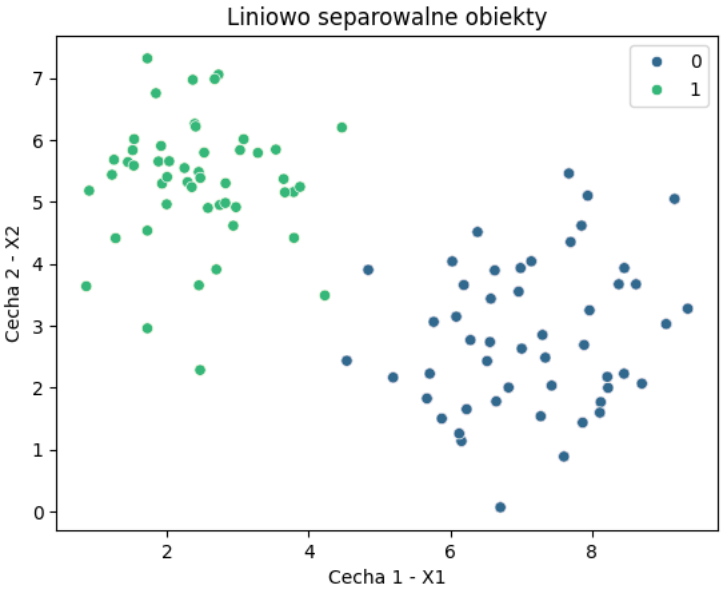
Przykład - przestrzeń hipotez zawiera pojęcie docelowe

Przyjrzyjmy się zbiorowi danych, prezentowanemu poniżej - każda kropka oznacza obiekt (np. klienta), opisany dwiema cechami numerycznymi (X1, X2 - np. wiek i waga). Każdy obiekt ma przypisaną etykietę (np. czy klient kupił produkt czy nie) - w tym przypadku odpowiadającą kolorowi zielonemu lub niebieskiemu.

Na pierwszy rzut oka wydaje się, że:

1. **Pojęcie docelowe istnieje** - obiekty po prawej stronie są „niebieskie” a po lewej „zielone”. Istnieje zatem jakiś proces, który przypisuje etykiety do danych i jest to proces sensowny.
2. **Przestrzeń hipotez** - w tym przypadku, przestrzeń hipotez jest bardzo prosta: wystarczy narysować linię, która oddzieli obiekty zielone od niebieskich. Wówczas, model, który stworzy taką hipotezę, będzie w stanie poprawnie klasyfikować nowe obiekty.

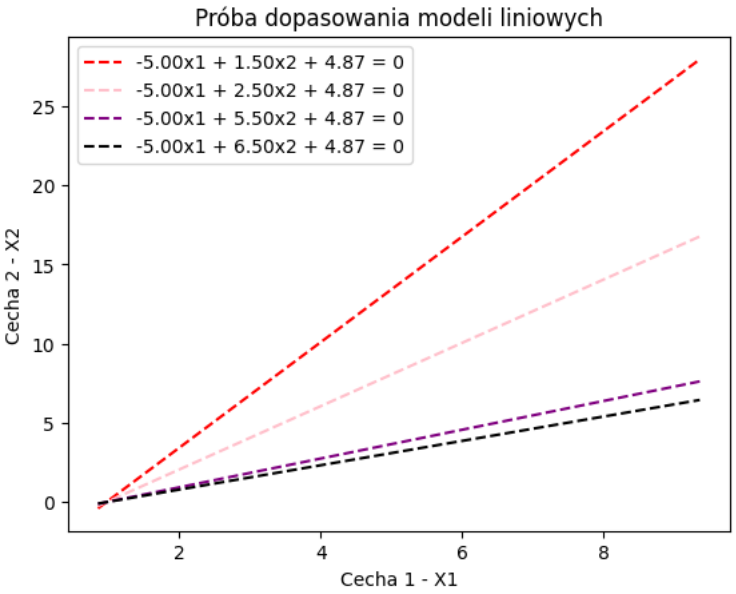
▶ Show code cell source



Możemy teraz wyznaczyć kilka linii oddzielających od siebie obie strefy. Każda taka linia (formalnie: **model liniowy**) jest **hipotezą** - czyli próbą wyjaśnienia zależności w danych.

Ewidentnie jedne są lepsze od innych.

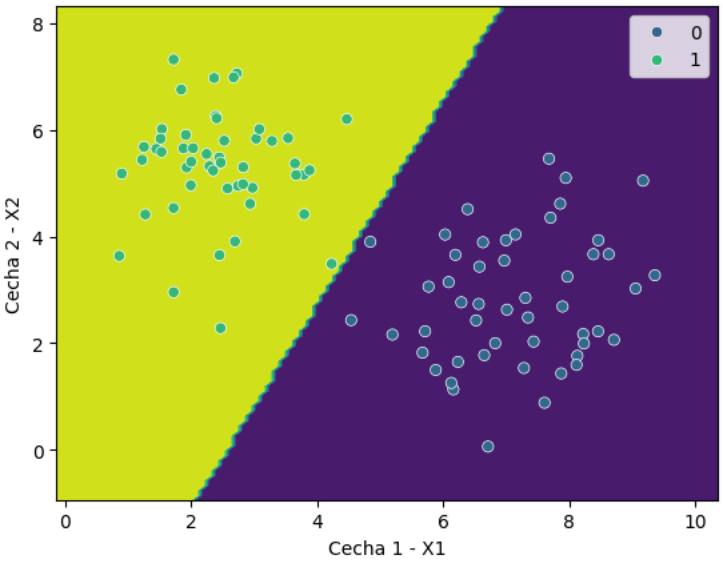
▶ Show code cell source



Jeśli pozwolimy modelowi liniowemu nauczyć się własności danych - znajdzie on w swojej **przestrzeni hipotez** najlepsze rozwiązanie, które oddzieli punkty od siebie.

Możemy to przedstawić jak na obrazku poniżej, gdzie model tworzy **płaszczyznę decyzyjną** - linię, która oddziela obiekty zielone od niebieskich.

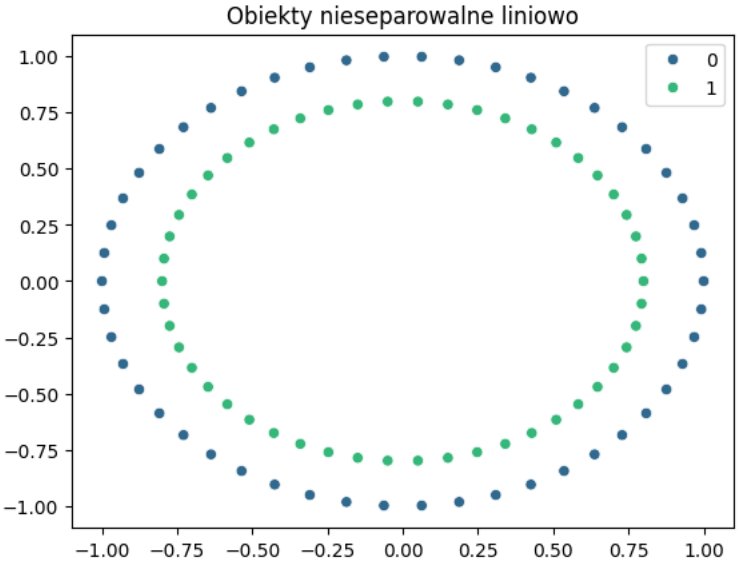
▶ Show code cell source



Przykład - pojęcie docelowe poza przestrzenią hipotez

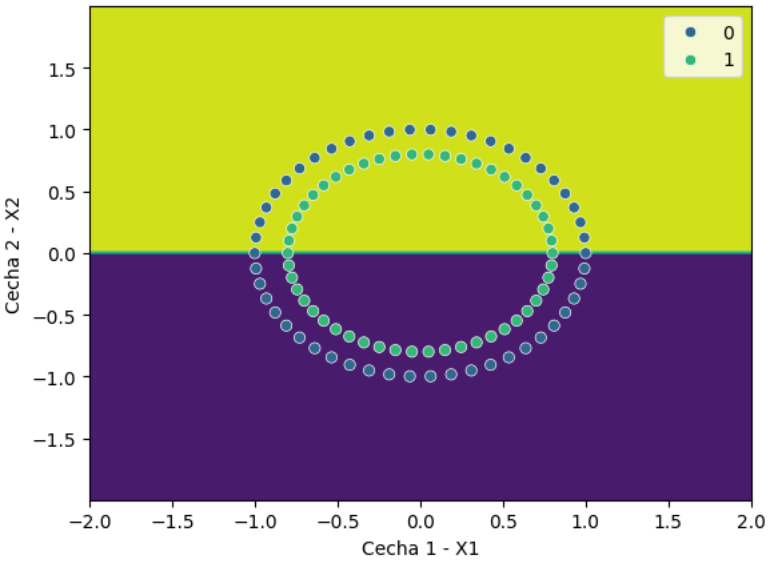
Rozpatrzmy kolejny przykład - bardzo niekorzystny dla naszego algorytmu liniowego. Także i tym razem mamy zbiór danych, opisany dwiema cechami numerycznymi (X1, X2) i etykietami (zielony, niebieski).Poszczególne etykiety (klasy) układają się w okrąg - co sugeruje, że nie istnieje żadna prosta, która oddzieliłaby obiekty jednej klasy od drugiej.

▶ Show code cell source



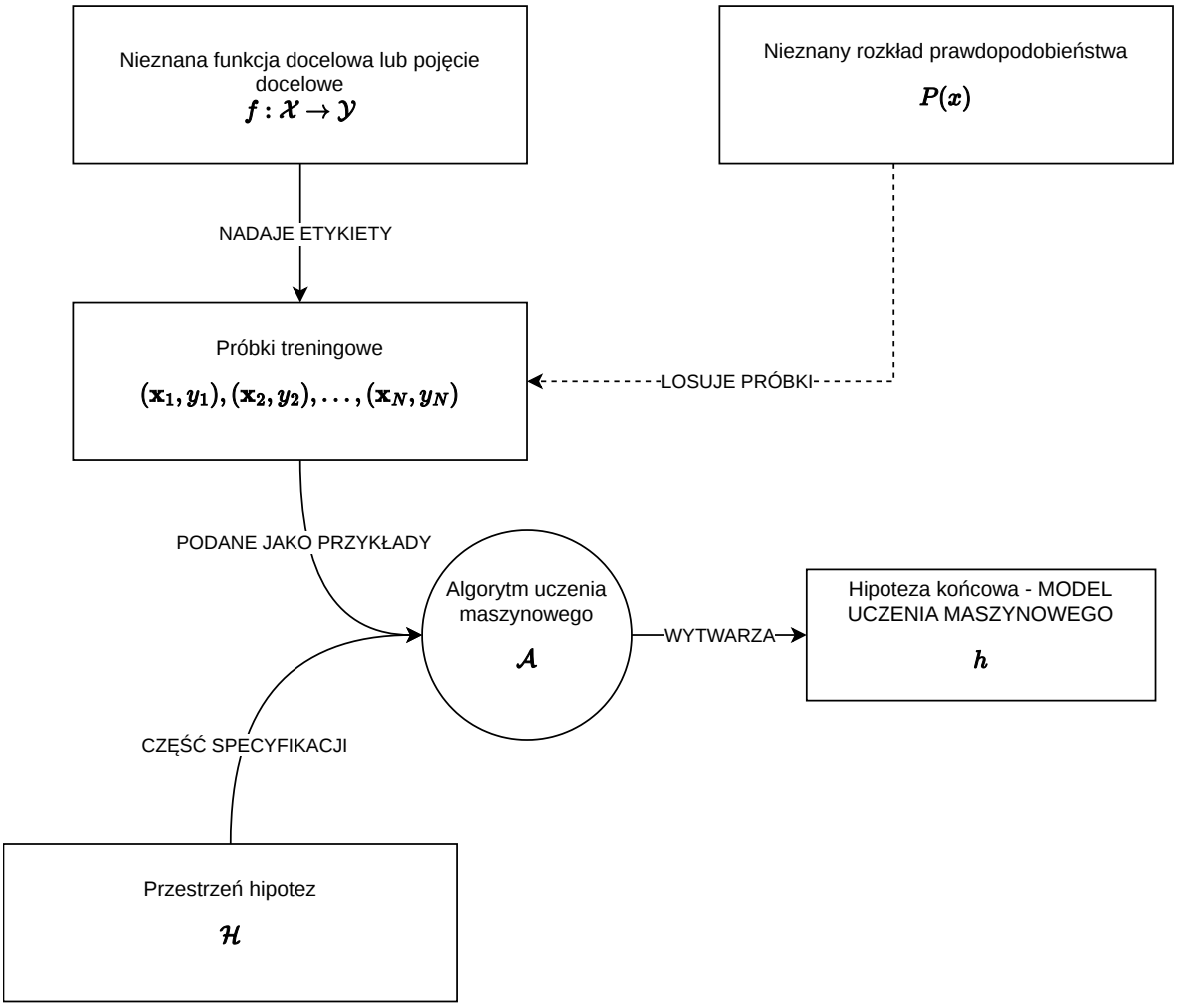
Choćbyśmy dopasowali nie wiadomo jak dobry model liniowy nie ma on najmniejszych szans sensownie oddzielić od siebie tych obiektów. W najlepszym wypadku, połowa obiektów każdej klasy znajdzie się po niewłaściwej stronie.

▶ Show code cell source



Elementy składowe procesu uczenia się

Mając na uwadze powyższe, możemy wskazać na diagramie najważniejsze elementy, składające się na proces uczenia się.



Rys. 2 Składowe procesu uczenia się. Adaptacja za: [MRTB12]

Co dokładnie mamy na tym diagramie? Wszystkie elementy wymienione do tej pory. Po kolei:

1. Nieznana nam funkcja prawdopodobieństwa $P(x)$ odpowiada za losowanie przykładów. Innymi słowy: model uczy się na próbce całej populacji. Próbki są wybierane, a o tym, które trafią nam w ręce decyduje właśnie rozkład $P(x)$.
2. Nieznana funkcja docelowa lub pojęcie docelowe nadaje etykiety do obiektów. W przypadku klasyfikacji, jest to funkcja, która przypisuje obiektom klasy. To właśnie tej funkcji staramy się nauczyć, albo przybliżyć. Przykłady przytaczaliśmy już wcześniej: `co decyduje o tym, że jedne osoby mają powikłania po chorobie C a inne nie?`, `co decyduje o wzroście/spadku cen akcji?`, `co decyduje o tym, że klient kupi produkt?`.
3. Zbiór uczący się składa się z par: $\mathbf{x}_i, y_i$, czyli zestawu macierzy atrybutów i etykiet.
4. Wybieramy algorytm uczenia maszynowego, który posiada pewną przestrzeń hipotez $\mathcal{H}$ - jest to jego zdolność do wyrażania różnych zależności w danych. Przykład: model liniowy omówiony wcześniej - przestrzenią hipotez są wszystkie możliwe do narysowania proste linie.
5. Model uczy się, starając się jak najlepiej dopasować do posiadanych etykiet przykładów. Na końcu otrzymujemy hipotezę docelową h. **Mamy nadzieję**, że dobrze przybliża ona pojęcie docelowe.

Tym co steruje procesem uczenia się i oceną jego jakości jest **błąd generalizacji**:

Błąd generalizacji

Mając daną hipotezę $h \in \mathcal{H}$, oraz pojęcie docelowe $c \in \mathcal{C}$ oraz rozkład danych D z którego pochodzą przykłady uczące, błąd generalizacji jest średnim prawdopodobieństwem, że hipoteza h będzie błędna na losowym przykładzie (x, y) zgodnie z rozkładem D. Formalnie:

$$R(h) = P_{(x \sim D)}\left[h(x) \neq c(x)\right] = \mathbb{E}_{x \sim D}\left[\mathbb{I}_{h(x) \neq c(x)}\right]$$

gdzie $\mathbb{I}$ jest funkcją indykatorową (wskaźnikową), która przyjmuje wartość 1, gdy warunek wewnątrz jest spełniony, a 0 w przeciwnym wypadku. [MRTB12]

Z powyższych, dość mocno teoretycznych (jak dotąd) rozważań, wysuwa się jeden, bardzo istotny dla wszystkich praktyków uczenia maszynowego wniosek:

Uczenie maszynowe jako probabilistyczne poszukiwanie hipotez

W tym ujęciu, uczenia maszynowego widać, jak mocno niepewny i stochastyczny jest to proces. Niepewność pojawia się w kilku miejscach:

- **Po pierwsze** - przykłady uczące są wybierane losowo z populacji, co oznacza, że nie mamy pewności, czy są one reprezentatywne.
- **Po drugie** - etykiety przypisane do danych mogą być błędne, niepełne, lub nieodpowiednie.
- **Po trzecie** - przestrzeń hipotez, w której poruszamy się, może nie zawierać w sobie pojęcia docelowego.
- **Po czwarte** - model, który tworzymy na bazie algorytmu **PRZESZUKUJE** przestrzeń swoich hipotez próbując się dopasować do posiadanych danych uczących.

Warto mieć te elementy na uwadze, gdy klienci, kontrahenci lub osoby chcące otrzymać od nas wyniki, pytają o **pewność** naszych modeli. Warto wtedy zwrócić uwagę na to, że **nie ma pewności** - są jedynie **prawdopodobieństwa**.

Obciążenie indukcyjne

Podczas wyboru najlepszego algorytmu, który posłuży nam do zbudowania modelu, naturalnym jest, że staramy sie wybrać taki, który ma jak najmniejszy błąd generalizacji. Warto w ramach tego procesu rozważyć mocne i słabe strony każdego z algorytmów, które bierzemy pod uwagę. Jak widzieliśmy we wcześniejszych przykładach, może się zdarzyć tak, że próbujemy zastosować narzędzie absolutnie nieprzystosowane do naszego problemu - np. model liniowy do rozdzielenia obiektów w kształcie okręgu.

Istnieje kilka pojęć, które posłużą nam do zdefiniowania tak rozumianych ograniczeń i możliwości algorytmów uczenia maszynowego.

Obciążenie indukcyjne

Zbiór założeń i ograniczeń algorytmu, sprawiający, że stawiane przez niego hipotezy różnią się od pojęcia docelowego lub skłaniające do preferowania jednej hipotezy nad inną, nazywane są zbiorczo obciążeniem indukcyjnym. [Cic07] [Woj21]

Innymi słowy - Obciążenie indukcyjne mówi nam o wszystkich czynnikach, które wpływają na to, dlaczego dany algorytm lub model, podejmuje takie decyzje, jakie podejmuje.

Obciążenie indukcyjne występuje w dwóch najczęstszych postaciach: reprezentacji i preferencji.

Obciążenie reprezentacji

Informuje o tym, jak ograniczona jest przestrzeń hipotez, które ma do dyspozycji określony algorytm. Innymi słowy – definiuje zawężenie funkcji wyjaśniających, których system może używać do wyjaśniania zjawisk. Ten rodzaj obciążania nie daje się w łatwy sposób wyeliminować - jest bowiem wpisany w samą strukturę algorytmu. [Cic07] [Woj21]

Obciążenie preferencji

związane jest z heurystykami wyboru najlepszej hipotezy spośród generowanych (proponowanych) przez algorytm. Funkcje te mogą zostać zaprojektowane z użyciem takich założeń lub aparatu matematycznego, iż prowadzić będą w sposób systematyczny i powtarzalny do preferowania określonych wyborów ponad innymi. [Cic07] [Woj21]

Brzmi skomplikowanie? Rozważmy kilka przykładów.

Ćwiczenie - obciążenie indykcyjne

Zastanów się nad następującymi przykładami ograniczeń posiadanych przez algorytmy i modele. Które z nich dotyczą **preferowania** pewnych wyników, a które wpisane są w samą **strukturę** algorytmu/modelu?

- **Model liniowy nie jest w stanie oddzielić danych w postaci dwóch okręgów.**
- **Model K-najbliższych sąsiadów nadaje najwyższą wagę atrybutom wyrażonym w największych jednostkach.**
- **W przypadku silnej nierównowagi klas (np. 95% zdrowych osób, 5% chorych) model zawsze przewiduje klasę większościową.**
- **Sieć neuronowa typu Multi-Layer-Perceptron nie może przetwarzać danych w postaci grafowej.**

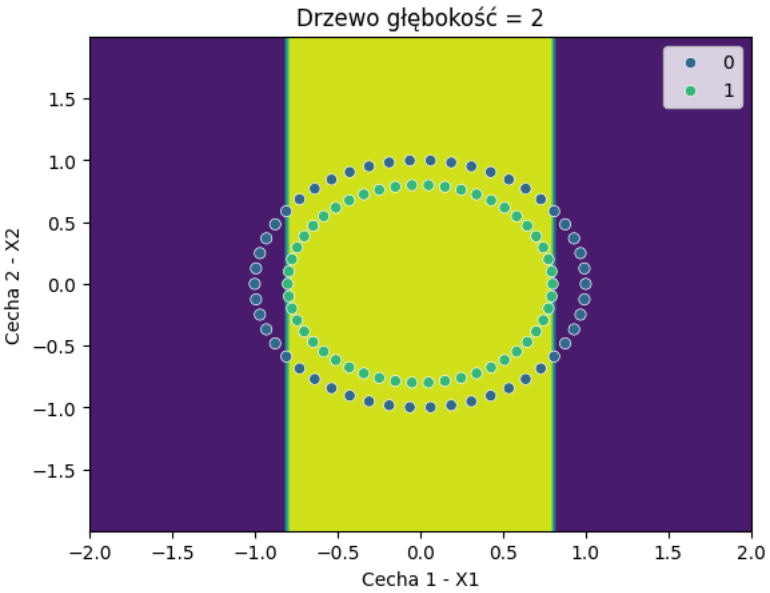
Przykład obciążenia indukcyjnego - obciążenia reprezentacji

Ograniczenie reprezentacji przez model liniowy, w przypadku danych układających się w kształt okręgów widzieliśmy już wcześniej. Zobaczmy teraz, jak poradzi sobie np. **drzewo decyzyjne** dla tego samego zbioru, w zależności od swojej konfiguracji - głębokości.

▶ Show code cell source

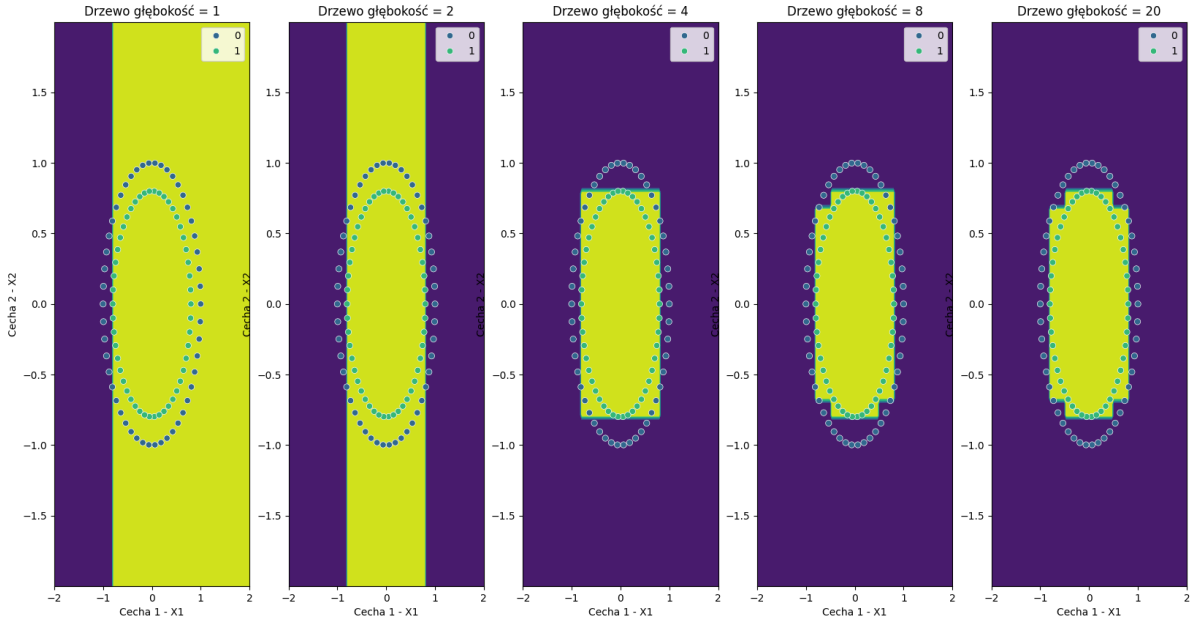
Proste drzewo o głębokości 2 jest niewiele lepsze od modelu liniowego.

▶ Show code cell source



W miarę zwiększania głębokości drzewa, widzimy, że stopniowo coraz lepiej radzi sobie z podziałem danych.

▶ Show code cell source

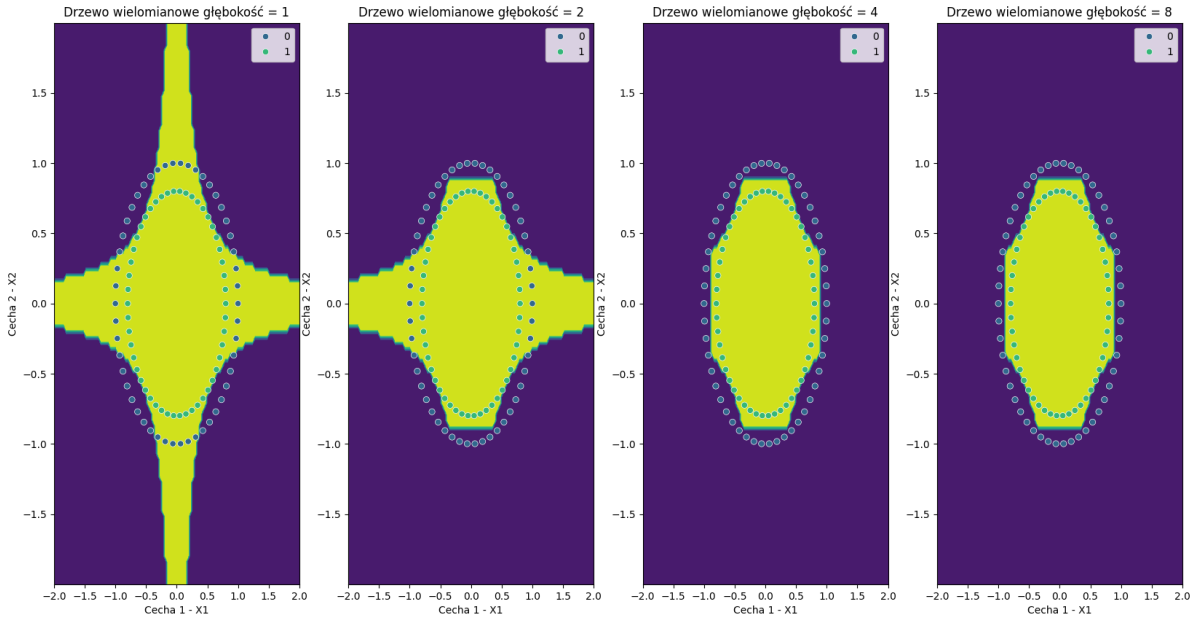


Drzewo o głębokości 8, co prawda, nie jest w stanie idealnie oddzielić obiektów, ale już znacznie lepiej niż model liniowy.

Zwiększenie głębokości drzewa do 20 nijak nie poprawia jednak trafności - zachodzi tu więc niewielkie obciążenie reprezentacji.

Zobaczmy jednak, co się stanie, jeśli wyszkolimy model, który otrzyma bardziej złożoną przestrzeń cech - cechy X1 i X2, podniesione do kwadratu i kolejnych potęg, a także przekształcone wielomianowo.

▶ Show code cell source



Widać, że w tym przypadku trafność znacznie się poprawiła już przy głębokości drzewa = 4. Tym samym znieśliśmy ograniczenie reprezentacji drzewa, poprzez **wzbogacenie danych źródłowych**. Jest to częsta strategia działania - nierzadko wystarczy zmienić format oryginalnych danych lub je przekształcić, by otrzymać lepsze wyniki.

PAC-nauczalność

Tak silna obecność niepewności i elementów probabilistycznych doprowadziła do sformułowania pojęcia PAC-nauczalności (ang. *Probably - Approximately - Correct learning*). Formalna definicja przedstawia się następująco:

PAC-nauczalność

Klasa pojęć $\mathcal{C}$ jest PAC-nauczalna, jeśli istnieje algorytm $\mathcal{A}$ i funkcja wielomianowa $poly(\cdot,\cdot,\cdot,\cdot)$ taka, że dla dowolnego $\epsilon > 0$ oraz $\delta > 0$ dla wszystkich rozkładów prawdopodobieństwa D dla zbioru obiektów $\mathcal{X}$ i dla każdego pojecia docelowego $c \in \mathcal{C}$ zachodzi następująca zależność dla próbki o wielkości $
$m \geq poly\left(\frac{1}{\epsilon}, \frac{1}{\delta}, \text{rozmiar}(\mathcal{X}), \text{złożoność}(\mathcal{C})\right)$:

$$P_{S \sim D^m}\left[R(h_S) \leq \epsilon\right] \geq 1 - \delta$$

gdzie h_S to hipoteza wygenerowana przez algorytm $\mathcal{A}$ na podstawie próbki uczącej S o rozmiarze m, a $R(h_S)$ to błąd generalizacji hipotezy h_S. [MRTB12]

Efektywna PAC-nauczalność

Jeśli algorytm $\mathcal{A}$ działa w czasie $poly(\cdot,\cdot,\cdot,\cdot)$, to klasa pojęć $\mathcal{C}$ jest efektywnie PAC-nauczalna. [MRTB12]

Powyższe definicje mogą się początkowo wydawać skomplikowane, ale ich znaczenie jest *de facto* dość proste.

PAC-nauczalność, czyli co?

PAC-nauczalność oznacza, że dla dowolnego pojęcia docelowego c z klasy $\mathcal{C}$, istnieje algorytm $\mathcal{A}$, który potrafi nauczyć się tego pojęcia, a prawdopodobieństwo, że popełni błąd mniejszy niż ϵ wynosi co najmniej $1 - \delta$. W praktyce oznacza to, że mamy pewność, że nasz model jest w stanie nauczyć się pojęcia docelowego z pewnym prawdopodobieństwem. Kluczowe staje się zwrócenie uwagi na znaczenie poszczególnych słów ze skrótu PAC. Zrobimy to w kolejności zgodnej z intuicyjnym zrozumieniem:

1. **Approximately** (w przybliżeniu) - model nie musi być idealny, ale musi być blisko prawdy. O tym, jaki błąd jest dopuszczalny mówi parametr ϵ. Na przykład, jeśli zakładamy $\epsilon = 0.05$, to oznacza, że model może popełnić błąd w 5% przypadków.
2. **Probably** (prawdopodobnie) - mówi o tym, jakie jest prawdopodobieństwo, że model popełni błąd mniejszy niż ϵ. O tym decyduje parametr δ. Na przykład, jeśli zakładamy $\delta = 0.01, 1 - \delta = 0.99$, to oznacza, że mamy 99% pewności, że nasz model popełni błąd nie większy niż ϵ (w tym przypadku: 5%).
3. **Correct** (poprawny) - model musi być poprawny, czyli musi być w stanie nauczyć się pojęcia docelowego, a zatem minimalizuje błąd generalizacji.

Jak widzimy, **najlepsze**, co możemy dostać w przypadku PAC-nauczalności, to założenie, że model **nie popełni błędu większego niż założony ϵ, nie częściej niż z prawdopodobieństwem δ**.

Takie ujęcie uczenia maszynowego jest dość daleko od pragnienia posiadania zawsze idealnych, albo prawie idealnych modeli, które działają poprawnie. W praktyce, mamy do czynienia z nieredukowalnym ryzykiem i niepewnością.

Dodatkowym aspektem, obecnym w przestawionej definicji jest **wielkość próbki uczącej m** oraz, na razie niezdefiniowana, **funkcja wielomianowa** decydująca o czasie wykonywania algorytmu.

---

[1] Autor sekcji: dr Filip Wójcik

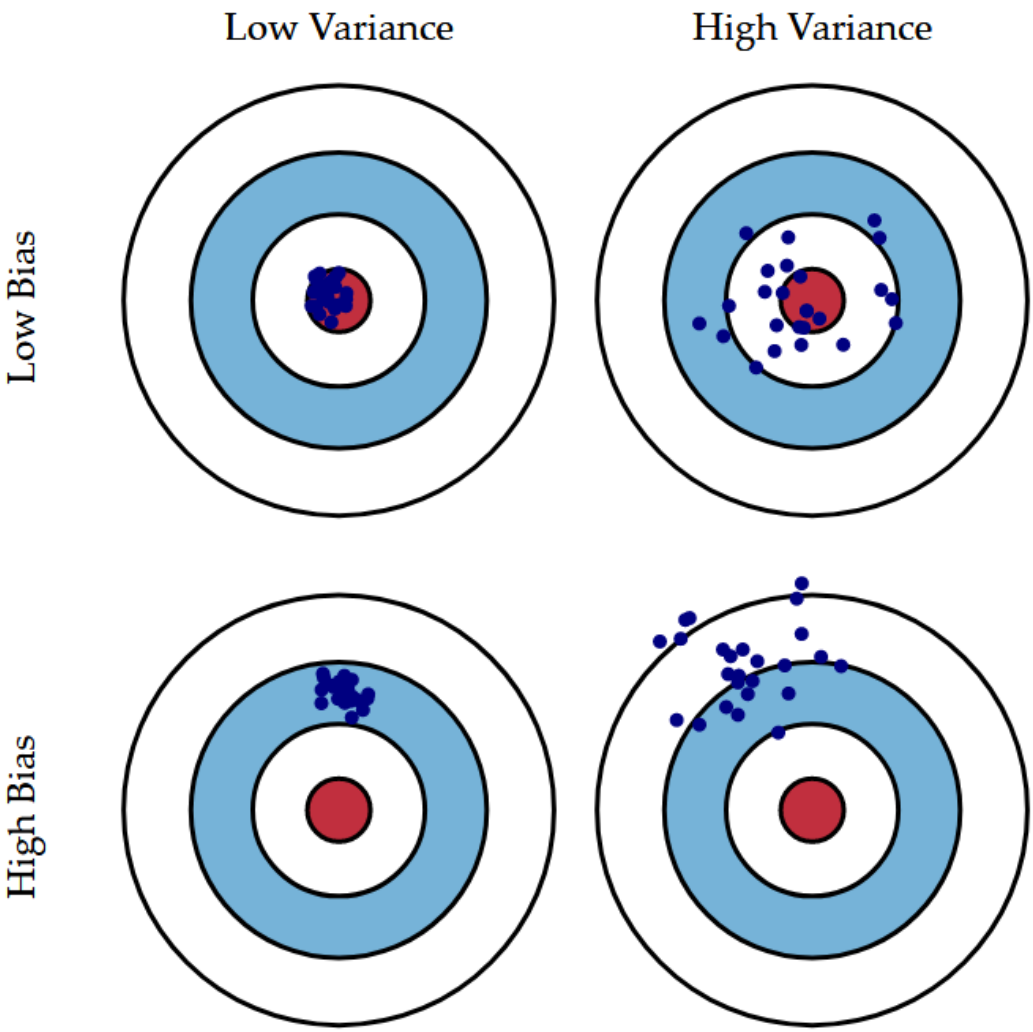
Błąd obciążenia i wariancji - kompromis [1]

Niniejszy podrozdział przedstawia kolejny, bardzo istotny aspekt uczenia maszynowego, jakim jest kompromis pomiędzy błędem obciążenia i wariancji. Jest to jedna z najczęściej przywoływanych zależności w uczeniu maszynowym, która ma kluczowe znaczenie w procesie budowy modeli predykcyjnych.

Kompromis między obciążeniem a wariancją

W kontekście uczenia maszynowego, mówimy o **kompromisie między obciążeniem a wariancją** (ang. *bias-variance tradeoff*). Jest to jedno z kluczowych pojęć, które pozwala zrozumieć, jakie są dwa najczęściej pojawiające się błędy, popełniane przez modele.

Zacznijmy jednak od obrazu, który mówi więcej niż tysiąc słów. W kontekście kompromisu między obciążeniem a wariancją bardzo często przywołuje się analogię z tarczami strzelniczymi - taką, jak pokazana poniżej.



Rys. 3 Obciążenie i wariancja - targe strzelnicze. Źródło: Devopedia - bias variance tradeoff

Na rysunku widzimy cztery hipotetyczne sytuacje:

1. **Low bias + low variance / niskie obciążenie + niska wariancja** - wszystkie trafienia układają się blisko centrum i nie są zbyt rozproszone. Oznacza to, że model jest w stanie dobrze przewidywać wartości, a jego trafienia są stabilne.
2. **Low bias + high variance / niskie obciążenie + wysoka wariancja** - trafienia są rozproszone, ale układają się blisko centrum. Oznacza to, że model jest w stanie średnio i co do zasady dobrze przewidywać wyniki, ale jego trafienia są niestabilne.
3. **High bias + low variance / wysokie obciążenie + niska wariancja** - trafienia są skupione, ale daleko od centrum. Oznacza to, że model ma problem z trafnym przewidywaniem w sposób systematyczny, ale jego trafienia są stabilne.

4. **High bias + high variance / wysokie obciążenie + wysoka wariancja** - najgorsza możliwa sytuacja. Trafienia są rozproszone i daleko od centrum. Oznacza to, że model ma problem z trafnym przewidywaniem w sposób systematyczny, a jego trafienia są niestabilne.

Każdy strzał do tarczy strzelniczej możemy traktować jako pojedyncze przewidywanie modelu. W praktyce, chcemy, aby nasz model był w stanie przewidywać wyniki w sposób **stabilny** i **trafny**, co niestety nie zawsze jest możliwe.

> **Czynniki sprzyjające błędowi obciążenia i wariancji**
>
> W praktyce obserwujemy często następującą zależność:
>
> 1. **Im większa złożoność modelu - tym większa wariancja i mniejsze obciążenie**. Wynika to z faktu, że bardziej skomplikowany model (np. sieć neuronowa) jest w stanie reprezentować niezwykle złożone zależności w danych, ale jednocześnie jest bardziej podatny na zmiany w danych uczących i może to doprowadzić do niestabilnych predykcji.
> 2. **Im mniejsza złożoność modelu - tym mniejsza wariancja i większe obciążenie**. Prostsze modele (np. regresja liniowa) są mniej podatne na zmiany w danych uczących, ale jednocześnie mają ograniczoną zdolność do reprezentowania złożonych zależności w danych.

Mając na uwadze powyższe elementy, możemy teraz zdefiniować **obciążenie**, **wariancję** i **kompromis między nimi**.

Błąd obciążenia modelu / Prediction bias

Obciążenie modelu to systematyczny błąd, który wynika z uproszczeń, jakie wprowadzamy w procesie uczenia. W praktyce oznacza to, że jego predykcja mijają się z wartością docelową. [AMMIL12] [Dom12]

Błąd wariancji modelu / Prediction variance

Wariancja modelu to miara tego, jak bardzo różnią się predykcje modelu, w zależności od zmian w zbiorze uczącym. Wysoka wariancja oznacza, że model często zmienia swoje decyzje, w zależności od niewielkich wahań w danych. [AMMIL12] [Dom12].

Kompromis między obciążeniem a wariancją / Bias-variance tradeoff

Kompromis między obciążeniem a wariancją to pojęcie, które mówi o tym, że w praktyce gdy zwiększamy złożoność modelu - redukując jego błąd obciążenia, zwiększamy jednocześnie wariancję. Zmniejszając złożoność modelu - zwiększamy błąd obciążenia, ale jednocześnie zmniejszamy wariancję. Bardzo rzadko możliwe jest jednoczesne zmniejszenie obydwu rodzajów błędu. [AMMIL12] [Dom12] [SW17]

Zanim przejdziemy do praktycznych przykładów, zdefiniujemy powyższe elementy w postaci formalnej.

Mając dany konkretny zestaw danych $\mathbf{X}_i$ lub skrótowo $\mathbf{X}$,oraz docelowych etykiet/kategorii (czyli nasze Pojęcie docelowe, którego szukamy) $\mathbf{y} = f(\mathbf{X})$, zbiór wielu takich zestawów: $\mathcal{D} = \{(\mathbf{X}_1, \mathbf{y}_1), (\mathbf{X}_2, , \mathbf{y}_2), \ldots, (\mathbf{X}_k, \mathbf{y}_k)\}$, model uczenia maszynowego h, dokonujący predykcji na tym zbiorze $\hat{y} = h(\mathbf{X})$, możemy zdefiniować błąd modelu jako sumę błędu obciążenia, błędu wariancji i błędu nieredukowalnego w kontekście.

Zaczynamy od definicji naszej funkcji kosztu - posłużymy się błędem średniokwadratowym:

$$S = (y - \hat{y})^2$$

Posługując się wzorem skróconego mnożenia $(a+b)^2 = a^2 + 2ab + b^2$ oraz przekształceniem algebraicznym polegającym na dodaniu i odjęciu tej samej wartości - w tym przypadku wartości oczekiwanej predykcji (czyli średniej predykcji) dla wszystkich zbiorów danych, możemy zapisać:

$$\begin{aligned} S &= (y - \hat{y})^2 \\ &= (y - E_{\mathcal{D}}[\hat{y}] + E_{\mathcal{D}}[\hat{y}] - \hat{y})^2 \\ &= (y - E_{\mathcal{D}}[\hat{y}])^2 + 2(y - E_{\mathcal{D}}[\hat{y}])(E_{\mathcal{D}}[\hat{y}] - \hat{y}) + (E_{\mathcal{D}}[\hat{y}] - \hat{y})^2 \end{aligned} \tag{6}$$

W równaniu (6) powyżej traktujemy człony

1. $y - E_{\mathcal{D}}[\hat{y}]$ jako *a* we wzorze skróconego mnożenia
2. $E_{\mathcal{D}}[\hat{y}] - \hat{y}$ jako *b* we wzorze skróconego mnożenia i potem korzystamy z własności wzoru skróconego mnożenia.

Ponieważ uczenie modelu jest procesem stochastycznym, musimy uśrednić ten wynik dla wielu różnych zbiorów danych. Do tego służy operator **wartości oczekiwanej** $E_{\mathcal{D}}[\cdot]$. Zanim to zrobimy, zwróćmy uwagę, że:

1. y nie jest zależne od zbioru danych - to deterministyczna funkcja, Pojęcie docelowe, które chcemy przewidzieć - ono jest raz na zawsze ustalone. Tym samym $E_{\mathcal{D}}[y] = y$
2. $E_{\mathcal{D}}[\hat{y}]$ to średnia predykcja modelu na wielu różnych zbiorach danych. To jest nasze **obciążenie** - czyli błąd, który wynika z uproszczeń, jakie wprowadzamy w procesie uczenia. Ta część nie będzie stałą, ale będzie silnie zależeć od specyfiki zbioru danych.
3. Z ogólnej własności operatora własności oczekiwanej wiemy, że: $E[E[x]] = E[x]$.
4. Z definicji wariancji, wiemy, że jest ona definiowana jako: $Var(x) = E[(x - E[x])^2]$

Zobaczmy teraz, jak możemy zdefiniować błąd obciążenia i wariancji w kontekście powyższego równania.

$$
\begin{aligned}
E_{\mathcal{D}}[S] &= E_{\mathcal{D}}[(y - \hat{y})^2] \\
&= E_{\mathcal{D}}\left[(y - E_{\mathcal{D}}[\hat{y}])^2 + 2(y - E_{\mathcal{D}}[\hat{y}])(E_{\mathcal{D}}[\hat{y}] - \hat{y}) + (E_{\mathcal{D}}[\hat{y}] - \hat{y})^2\right] \\
&= E_{\mathcal{D}}[(y - E_{\mathcal{D}}[\hat{y}])^2] + E_{\mathcal{D}}[2(y - E_{\mathcal{D}}[\hat{y}])(E_{\mathcal{D}}[\hat{y}] - \hat{y})] + E_{\mathcal{D}}[(E_{\mathcal{D}}[\hat{y}] - \hat{y})^2] \\
&= (y - E_{\mathcal{D}}[\hat{y}])^2 + E_{\mathcal{D}}[(E_{\mathcal{D}}[\hat{y}] - \hat{y})^2] \\
&= [\text{błąd obciążenia}^2] + \text{wariancja}
\end{aligned}
\tag{5}
$$

W równaniu (5) dużo się dzieje, ale (wbrew pozorom) są to dość podstawowe przekształcenia, wynikające z definicji podstawowych terminów probabilistyki i statystyki.

- **Bias** (czyli nasz **błąd obciążenia**) uzyskuje definicję:

$$\text{błąd obciążenia} = (y - E_{\mathcal{D}}[\hat{y}])^2$$

, możemy zatem o nim myśleć, jako o **średnim błędzie kwadratowym pomiędzy pojeciem docelowym, a wartością oczekiwaną (np. średnią) predykcji różnych modeli dla tego samego zbioru danych**.

- **Wariancja** uzyskuje definicję

$$\text{wariancja} = E_{\mathcal{D}}[(E_{\mathcal{D}}[\hat{y}] - \hat{y})^2]$$

, możemy zatem o niej myśleć, jako o **średnim błędzie kwadratowym pomiędzy średnią predykcją różnych modeli dla tego samego zbioru danych, a predykcją konkretnego modelu**.

Znikający wyraz

Uważna Czytelniczka lub Czytelnik stwierdzi, że w (5) w przedostatniej linijce znika wyraz $E_{\mathcal{D}}[2(y - E_{\mathcal{D}}[\hat{y}])(E_{\mathcal{D}}[\hat{y}] - \hat{y})]$. Dlaczego tak się dzieje?

Otóż po zastosowaniu operatora wartości oczekiwanej, ten wyraz upraszcza się i skraca do zera. Warto to sprawdzić samemu (słynne: „pozostawiamy to jako ćwiczenie…"). Zachęcam do sprawdzenia samodzielnie tej zależności. W razie czego, poniżej znajduje się jej wyprowadzenie.

Wyprowadzenie odpowiedzi

Praktyczne zastosowanie

Czy da się to jakoś zastosować w praktyce, albo zbadać za pomocą narzędzi? Owszem. W praktyce, możemy zastosować **walidację krzyżową** (ang. *cross-validation*), aby zbadać, jak zmieniają się błąd obciążenia i wariancja w zależności od

złożoności modelu.

Ponadto, mamy gotowe biblioteki, które pomagają nam wyznaczać obciążenie i wariancję poszczególnych modeli dla danego zbioru danych / problemu. w ten sposób, możemy wybrać alternatywę, która stanowi rozsądny kompromis.

Przykład

W przykładzie poniżej zbadamy rozkład obciążenia i wariancji kilku modeli dla zbioru danych California housing - zawierającego informacje o cenach nieruchomości w Kalifornii, w zależności od wielu różnych czynników.

```
import pandas as pd
from tqdm.autonotebook import tqdm
from time import time
from mlxtend.evaluate import bias_variance_decomp
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression

california = fetch_california_housing()
X = pd.DataFrame(california.data, columns=california.feature_names)
y = california.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X.shape)

X.head(3)
```

```
(20640, 8)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24

Nasz zbiór danych składa się z 20640 obserwacji i 8 zmiennych objaśniających. Naszym celem jest przewidzenie ceny nieruchomości na podstawie tych zmiennych.

Wykorzystamy pakiet MlXtend oraz zawartą w nim funkcję `bias_variance_decomp`, która wykonuje następujące działania:

Algorithm 1 (Dekompozycja błędów)

Wejścia

1. Zbiór danych treningowych $\mathbf{X}$ wraz z wartościami docelowymi $\mathbf{y}$,
2. Zbiór danych testowych $\mathbf{X}_{test}$ wraz z wartościami docelowymi $\mathbf{y}_{test}$,
3. Liczba iteracji próbkowania k,
4. Funkcja kosztu L,
5. Model h

Wyjście Wyliczenie błędu obciążenia, wariancji i średniej oczekiwanej funkcji kosztu.

1. Dla każdej iteracji i od 1 do k:
    1. Wybierz losowo ze zwracaniem (metodą `bootstrap`) próbkę ze zbioru treningowego $\mathbf{X}', \mathbf{y}'$ z $\mathbf{X}, \mathbf{y}$.
    2. Wytrenuj model h na zbiorze $\mathbf{X}', \mathbf{y}'$.
    3. Dokonaj predykcji na zbiorze testowym.
    4. Zapisz różnice pomiędzy predykcją a wartością docelową dla każdej obserwacji.
2. Za pomocą bias-variance-derivation oblicz błąd obciążenia, wariancję i błąd nieredukowalny.

Sprawdźmy zachowanie kilku modeli:

1. **Prosta regresja liniowa** - oczekujemy najmniejszej wariancji i dużego obciążenia,
2. **Pojedyncze drzewo decyzyjne dla regresji** - oczekujemy średniej wariancji i średniego obciążenia.
3. **Las losowy** - oczekujemy małego obciążenia i małej wariancji.

```
model_results = []

models = [
    ('Linear Regression', LinearRegression()),
    ('Decision Tree', DecisionTreeRegressor(max_depth=10)),
    ('Random Forest', RandomForestRegressor(n_estimators=25, max_depth=10, n_jobs=-1))
]

for name, model in tqdm(models):
    train_start = time()
    avg_expected_loss, avg_bias, avg_var = bias_variance_decomp(
        model, X_train.values, y_train, X_test.values, y_test,
        loss='mse',
        num_rounds=100,
        random_seed=123)
    train_end = time()
    trianing_time = train_end - train_start
    model_results.append([name, avg_expected_loss, avg_bias, avg_var, trianing_time])

model_results = pd.DataFrame(model_results, columns=['Model', 'Expected Loss', 'Bias', 'Variance', 'Trai
model_results.sort_values('Expected Loss', ascending=True)
```

	Model	Expected Loss	Bias	Variance	Training Time
2	Random Forest	0.316762	0.294988	0.021775	26.608934
1	Decision Tree	0.456935	0.294924	0.162011	6.042207
0	Linear Regression	0.556295	0.555354	0.000942	0.391138

Widzimy, że nasze obserwacje się potwierdziły:

1. **Regresja liniowa** - ma największe obciążenie, ale najmniejszą, niemal zerową wariancję.
2. **Drzewo decyzyjne** - ma średnie obciążenie i wariancję.
3. **Las losowy** - ma najmniejsze obciążenie i wariancję.

Oczywiście wszystko to jest funkcją złożoności modelu - spójrzmy na czas szkolenia.

---

[1] Autor sekcji: dr Filip Wójcik

Wprowadzenie do zadań i metryk w uczeniu maszynowym [1]

w poprzednich rozdziałach omówiliśmy sobie podstawowe zagadnienia związane z teorią uczenia się - błędami generalizacji i dopasowania. Brakuje nam w tej układance jednego elementu - odpowiedzi na pytanie: **jak oceniać jakość modeli uczenia maszynowego?**. Względem jakich kryteriów? Metryki oceny są uzależnione od **zadania, jakie dany model ma realizować**.

W niniejszym rozdziale przedstawione zostaną podstawowe pojęcia związane z funkcjami kosztu i straty, omówiony zostanie podział zadań w uczeniu maszynowym oraz zaprezentowane zostaną najważniejsze metryki oceny modeli. Omówimy sobie:

1. Funkcje kosztu i straty - Funkcje kosztu i straty stanowią fundament w ocenie jakości modeli uczenia maszynowego. Są to funkcje matematyczne, które kwantyfikują różnicę pomiędzy przewidywaniami modelu a rzeczywistymi wartościami. Celem nauki modelu jest minimalizacja tej różnicy, co prowadzi do poprawy jego dokładności.
2. Podział zadań w uczeniu maszynowym Zadania w uczeniu maszynowym można podzielić na kilka głównych kategorii, z których najważniejsze to:
    - **Klasyfikacja**: Przypisywanie elementów do określonych klas.
    - **Regresja**: Przewidywanie wartości ciągłych.

- **Grupowanie (klasteryzacja)**: Grupowanie podobnych elementów w zbiory.
- **Ranking**: Ustalanie kolejności elementów, szczególnie w systemach rekomendacyjnych.

3. Klasyfikację i jej metryki.
4. Regresję i jej metryki.
5. Grupowanie i jego metryki.
6. Ranking i jego metryki.

> **Inne zadania, inne metryki**
>
> Warto zauważyć, że niektóre specyficzne dziedziny uczenia maszynowego, takie jak analiza grafów czy analiza obrazów, mają swoje unikalne metryki oceny. Ten rozdział jednak skupia się na klasycznych, ogólnych metrykach stosowanych w różnych zadaniach uczenia maszynowego, zapewniając solidne podstawy do dalszego zgłębiania bardziej zaawansowanych zagadnień.

Funkcje kosztu i straty

W uczeniu maszynowym funkcje kosztu i straty odgrywają kluczową rolę w ocenie i optymalizacji modeli. Stanowią one formalne narzędzie do kwantyfikacji różnic między przewidywaniami modelu a rzeczywistymi wartościami, co pozwala na iteracyjne doskonalenie modeli poprzez minimalizację tych różnic.

> **Brak jednej definicji**
>
> Nie ma konsensusu odnośnie definicji tych terminów wśród autorów i praktyków uczenia maszynowego. W praktyce często używa się ich zamiennie, choć niektórzy wyróżniają między nimi różnice. W niniejszym rozdziale przedstawimy obie definicje oraz ich wspólne rozumienie, jako alternatywny punkt widzenia.

Funkcja straty

(ang. *loss function*) - oznaczana abstrakcyjnie jako $\mathcal{L}(\hat{y}, y)$ mierzy różnice między predykcja modelu $\hat{y} = h(\mathbf{x}_i; \theta)$ (gdzie h jest modelem/hipotezą uczenia maszynowego przewidującą wynik dla przykładu uczącego $\mathbf{x}$) a rzeczywistą wartością oczekiwaną dla danego przykładu uczącego.

Funkcja kosztu

(ang. *cost function*) - oznaczana abstrakcyjnie jako $\mathcal{J}(\theta)$ mierzy różnice między predykcjami modelu a rzeczywistymi wartościami dla **całego zbioru danych uczących**. Jest to agregacja funkcji straty dla wszystkich przykładów uczących. Agregację można przeprowadzić za pomocą średniej, mediany lub podobnych miar. Funkcja kosztu nierzadko zawiera również komponent regularyzujący złożoność modelu (np. złożoność wag) uzyskując postać:
$J(\theta) = \text{AGG}\left\{\mathcal{L}(\hat{y}_i, y_i), \forall i \in \mathcal{D}\right\} + \Omega(\theta)$

Definicje zamieszczone poniżej odnajdziemy w zbliżonej postaci m. in. w [MN07], [HTFF09], [GBC16]. Co ciekawe, prof. Andrew Ng używa rozróżnia ang. *cost* i *loss function* w swoich kursach prowadzonych na platformie Coursera, **ale** w publikowanych materiałach Stanfordu do kursu CS229 posługuje się **czasami określeniem **cost** a czasem **cost/loss** function* dodając słowne określenie, czy mówi o funkcji dotyczącej wszystkich przykładów, czy całego zbioru danych.

Przykładowo:

1. W rozdziale 8 [MN07], „Generalization” czytamy:

> (…) we typically learn a model $h(\theta)$ by minimizing a loss/cost function $J(\theta)$, which encourages $h(\theta)$ to fit the data. E.g., E.g., when the loss function is the least square loss (aka mean squared error), we have $
> $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - h_\theta(x^{(i)}))^2$$. This loss function for training purposes is oftentimes referred to as the training loss/error/cost.

2. W rozdziale 7.1. [MN07] „Deep learning” czytamy:

> For simplicity, we start with the case where the output is a real number, that is, $y^{(i)} \in \mathbb{R}$ and thus the model $h(\theta\theta)$ also outputs a real number (...). We define the least square cost function for the i-th example $(x^{(i)}, y^{(i)})$ as: $ $J^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2, and de finethemean - squarecostfunctionforthedatasetas$:J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J^{(i)}(\theta)$.

Z kolei Ian Goodfellow, uznany praktyk i naukowiec (do niedawna zaangażowany w DeepMind) w swojej książce „Deep Learning Book" [GBC16] w ogóle nie rozróżnia pojęć *loss / cost function* wprost, stosując je zamiennie i dodając, w definicji, czy opisuje funkcję dla pojedynczego przykładu czy dla całego zbioru danych.

Funkcje kosztu i straty w praktyce

Nie chodzi nam tutaj o teoretyczne spory i rozważania. Ważna lekcja do zapamiętania ogranicza się do tych kilku punktów:

1. Możemy odróżniać funkcje opisujące błędy dla **pojedynczego przykładu**;
2. Wiele takich funkcji można **zagregować** (np. za pomocą średniej) i uzyskać funkcję kosztu dla całego zbioru danych, dodając np. karę za złożoność całego modelu;
3. Jeśli ktoś próbuje nam zarzucić, że źle używamy tych pojęć, to warto zwrócić uwagę, że nie ma zgody co do ich rozdziału wśród ekspertów.

Klasyfikacja

Klasyfikacja

Klasyfikacja jest jednym z kluczowych zadań w uczeniu maszynowym, polegającym na przypisaniu obiektów do określonych kategorii (klas) na podstawie ich cech. Formalnie, klasyfikator jest funkcją $h : \mathcal{X} \to \mathcal{C},\ \text{gdzie}\ \mathcal{C} = \{c_1, c_2, \ldots, c_k\}$, która przypisuje obiektom $x \in \mathcal{X}$ etykiety z zadanego, skończonego zbioru. Możemy wyróżnić dwa podstawowe typy klasyfikacji: klasyfikację binarną, gdzie mamy do czynienia z dwiema klasami, oraz klasyfikację wieloklasową, gdzie obiekt może należeć do jednej z wielu klas. W języku polskim określa się czasem zadnie klasyfikacji, mianem `taksonomii`. [Fla12], [HTFF09], [MN07]

---

[1] Autor sekcji: dr Filip Wójcik.

Wprowadzenie do narzędzi Data Science

Jeszcze kilka(naście) lat temu, osoby zajmujące się *data science* poruszały się, co do zasady, w obrębie najważniejszych bibliotek wybranego języka - R albo Pythona. W miarę upływu czasu oraz rozrastania się społeczności praktyków, dostępnych narzędzi, a także upowszechniania się praktyk związanych z zarządzaniem projektem DS, pojawiło się wiele nowych rozwiązań, które w zasadzie trzeba znać i umieć wykorzystać, aby skutecznie poruszać się w profesjonalnych projektach DS.

W ramach tej sekcji zostaną przedstawione najistotniejsze spośród nich. Lista oczywiście nie jest wyczerpująca, oraz będzie się zmieniać z upływem czasu: nie ma miesiąca, by nie pojawiło się nowe narzędzie, które zasługuje na uwagę.

Przegląd bibliotek [1]

Python jest jednym z najczęściej wybieranych języków programowania w zastosowaniach analitycznych. Swoją pozycję zawdzięcza między innymi:

1. Relatywnej łatwości nauki;
2. Dużej ilości dostępnych bibliotek;
3. Aktywnej społeczności;
4. Wykorzystaniu przez różne grupy zawodowe: od testerów, administratorów, po programistów i analityków danych.

Nic więc dziwnego, że jest uznawany za wiodące rozwiązanie w dziedzinie Data Science.

Istnieje cała masa bibliotek przeznaczonych do rozmaitych zastosowań, nierzadko dublujących się lub będących „nakładkami" na inne narzędzia. W tej sekcji zostaną przedstawione najważniejsze, naszym zdaniem, biblioteki, które warto znać i umieć wykorzystać w codziennej pracy.

Dla ułatwienia zadania podzieliliśmy je na kilka kategorii:

1. Biblioteki do analizy danych;
2. Biblioteki do operacji matematycznych i statystycznych;
3. Biblioteki do wizualizacji danych;
4. Biblioteki do uczenia maszynowego;
5. Biblioteki do analizy szeregów czasowych;
6. Biblioteki do automatyzacji i optymalizacji pracy;
7. Inne.

Lista ta oczywiście nie jest w pełni kompletna i nie obejmuje m. in. narzędzi z zakresu np. MLOps, czyli zarządzania cyklem życia modeli uczenia maszynowego oraz całych projektów. Dodatkowo, niemal każda pod-dziedzina uczenia maszynowego posiada własne zestawy specyficznych bibliotek, których są dziesiątki. Przykładowo:

1. Dla analizy grafów i grafowych sieci neuronowych:
    1. Pytorch Geometric
    2. NetworkX
    3. Deep Graph Library DGL
2. Dla analizy języka naturalnego i przetwarzania tekstu:
    1. SpaCy
    2. NLTK
    3. Gensim
    4. Transformers od Hugginface - cały zestaw powiązanych narzędzi dla modeli LLM i szkolenia modeli językowych.
3. Dla analizy obrazów:
    1. OpenCV
    2. Scikit-image

i tak dalej. Pracując z konkretną pod-dziedziną musimy zapoznać się z konkretnymi narzędziami w niej używanymi.

Lista zamieszczona w tej sekcji zawiera tylko absolutne minimum przydatne każdemu, rozpoczynającemu swoją przygodę w świecie DS.

Biblioteki do pracy z danymi

Praca z danymi jest kluczowym elementem Data Science. Poniżej przedstawiamy dwie popularne biblioteki do manipulacji i analizy danych: `pandas` oraz `polars`.

Pandas

Pandas to jedna z najbardziej wszechstronnych bibliotek Pythona, używana do manipulacji i analizy danych. Oferuje elastyczne struktury danych, takie jak Series (jednowymiarowe tablice) i DataFrame (dwuwymiarowe tablice/tabele bazodanowe/arkusze Excel - ogólniej: dane ustrukturyzowane).

Pandas jest nieodzownym elementem w zestawie narzędzi Data Science.

Zastosowania Pandas

Pandas jest używany do:

1. Ładowania, czyszczenia i przekształcania danych.
2. Analizy danych, takich jak grupowanie, agregacja, filtrowanie.
3. Pracy z danymi z plików CSV, Excel, baz danych, oraz formatów takich jak JSON.

Dodatkowe materiały

Przydatne linki - Pandas

1. Pandas - przewodnik i oficjalna dokumentacja
2. Książka - Data analytics with Python pokrywająca dużą część funkcjonalności biblioteki Pandas.

Przykład 1: Proste wczytywanie danych z pliku CSV

```
import pandas as pd

# Wczytywanie danych z pliku CSV
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv"
titanic_data = pd.read_csv(url)

# Przegląd pierwszych kilku wierszy
titanic_data.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	emb
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Sou
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Sou
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Sou
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Sou

```
# Generowanie podsumowania danych
print(titanic_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    object
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    object
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
None
```

W powyższym przykładzie załadowano dane dotyczące pasażerów Titanica z pliku CSV dostępnego online, a następnie wyświetlono pierwsze kilka wierszy oraz podstawowe informacje o zbiorze danych, takie jak liczba wierszy i kolumn oraz typy danych.

Przykład 2: Filtrowanie i grupowanie

```
# Filtrowanie danych - tylko kobiety, które przeżyły
surviving_females = titanic_data[(titanic_data['sex'] == 'female') & (titanic_data['survived'] == 1)]

print("Kobiety, które przetrwały katastrofę")
surviving_females.head()
```

```
Kobiety, które przetrwały katastrofę
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	en
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	S
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	S
8	1	3	female	27.0	0	2	11.1333	S	Third	woman	False	NaN	S
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False	NaN	

```
# Grupowanie danych - średni wiek i cena biletu dla różnych klas
average_stats = titanic_data.groupby('class').agg({'age': 'mean', 'fare': 'mean'})

print("\nŚredni wiek i cena biletu dla różnych klas:\n", average_stats)
```

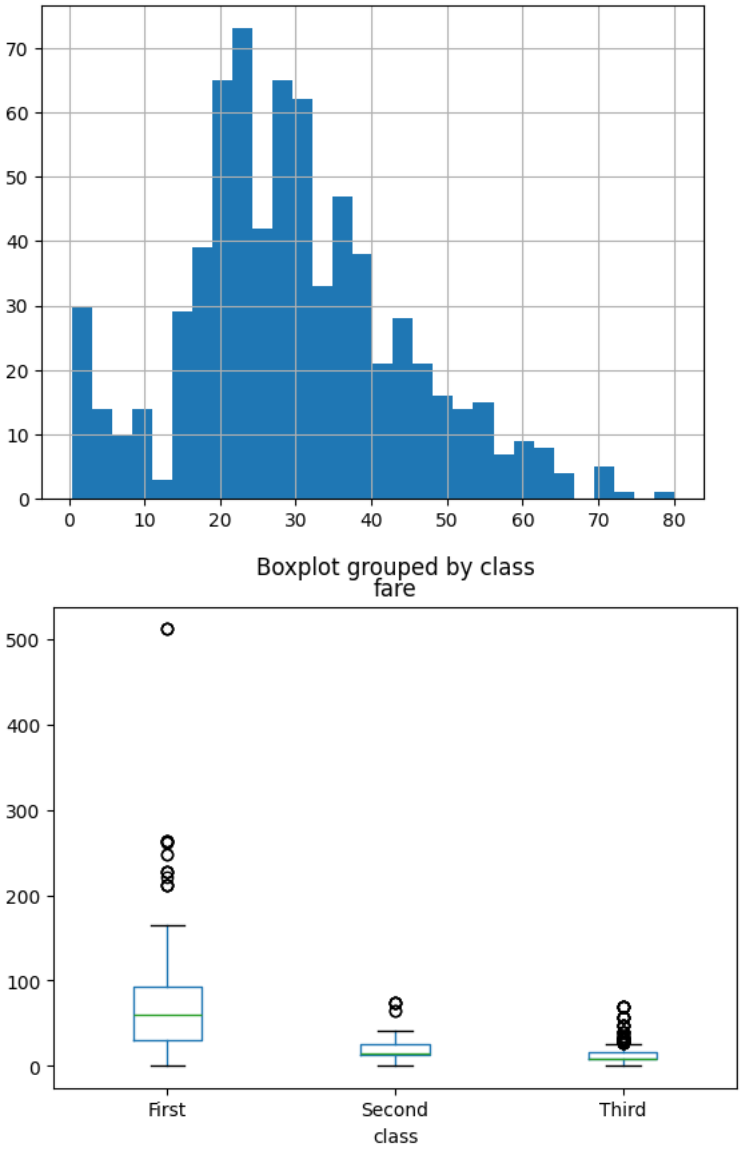
```
Średni wiek i cena biletu dla różnych klas:
              age       fare
class
First   38.233441  84.154687
Second  29.877630  20.662183
Third   25.140620  13.675550
```

W tym przykładzie przefiltrowano dane, aby uzyskać informacje tylko o kobietach, które przeżyły katastrofę, a także dokonano grupowania danych według klasy pasażerów, obliczając średni wiek oraz średnią cenę biletu w każdej klasie.

Przykład 3: Wizualizacja danych przy pomocy Pandasa

```
# Histogram wieku pasażerów
titanic_data['age'].hist(bins=30)

# Wykres pudełkowy (boxplot) dla cen biletów w różnych klasach
titanic_data.boxplot(column='fare', by='class', grid=False)
```

```
<Axes: title={'center': 'fare'}, xlabel='class'>
```



Pandas oferuje również proste metody wizualizacji danych. Powyżej wygenerowano histogram wieku pasażerów oraz wykres pudełkowy (boxplot) dla cen biletów w różnych klasach, co pozwala na szybkie zwizualizowanie rozkładu danych.

Pandas posiada uproszczone funkcje do wizualizacji danych, które delegują wykonanie określonych funkcji do pakietów jak np. `matplotlib`.

Polars

Polars to nowoczesna, szybka i wielowątkowa alternatywa dla Pandas, napisana w języku Rust. Jest zoptymalizowana pod kątem wydajności, zwłaszcza przy pracy z dużymi zbiorami danych. Polars oferuje podobne struktury danych jak Pandas, w tym DataFrame, ale dzięki swojej architekturze może działać szybciej i efektywniej na dużych danych.

Zastosowania Polars

Polars jest używany do:

1. Szybkiej analizy dużych zbiorów danych.
2. Przetwarzania danych na wielu rdzeniach procesora (wielowątkowość).
3. Wykonywania operacji podobnych do Pandas, ale z lepszą wydajnością.

Dodatkowe materiały

Przydatne linki - Polars

1. Dokumentacja i przewodnik Polars
2. Tutorial Polars od Real Python

Przykład 1: Wczytaywanie i analiza dancyh

```
import polars as pl

# Wczytywanie danych z pliku CSV
titanic_data = pl.read_csv(url)

# Przegląd pierwszych kilku wierszy
titanic_data.head()
```

shape: (5, 15)

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	em
i64	i64	str	f64	i64	i64	f64	str	str	str	bool	str	
0	3	"male"	22.0	1	0	7.25	"S"	"Third"	"man"	true	null	"So
1	1	"female"	38.0	1	0	71.2833	"C"	"First"	"woman"	false	"C"	"
1	3	"female"	26.0	0	0	7.925	"S"	"Third"	"woman"	false	null	"So
1	1	"female"	35.0	1	0	53.1	"S"	"First"	"woman"	false	"C"	"So
0	3	"male"	35.0	0	0	8.05	"S"	"Third"	"man"	true	null	"So

```
# Informacje o danych
print(titanic_data.describe())
```

```
shape: (9, 16)
┌────────────┬──────────┬──────────┬────────┬───┬──────┬─────────────┬───────┬──────────┐
│ statistic  ┆ survived ┆ pclass   ┆ sex    ┆ … ┆ deck ┆ embark_town ┆ alive ┆ alone    │
│ ---        ┆ ---      ┆ ---      ┆ ---    ┆   ┆ ---  ┆ ---         ┆ ---   ┆ ---      │
│ str        ┆ f64      ┆ f64      ┆ str    ┆   ┆ str  ┆ str         ┆ str   ┆ f64      │
╞════════════╪══════════╪══════════╪════════╪═══╪══════╪═════════════╪═══════╪══════════╡
│ count      ┆ 891.0    ┆ 891.0    ┆ 891    ┆ … ┆ 203  ┆ 889         ┆ 891   ┆ 891.0    │
│ null_count ┆ 0.0      ┆ 0.0      ┆ 0      ┆ … ┆ 688  ┆ 2           ┆ 0     ┆ 0.0      │
│ mean       ┆ 0.383838 ┆ 2.308642 ┆ null   ┆ … ┆ null ┆ null        ┆ null  ┆ 0.602694 │
│ std        ┆ 0.486592 ┆ 0.836071 ┆ null   ┆ … ┆ null ┆ null        ┆ null  ┆ null     │
│ min        ┆ 0.0      ┆ 1.0      ┆ female ┆ … ┆ A    ┆ Cherbourg   ┆ no    ┆ 0.0      │
│ 25%        ┆ 0.0      ┆ 2.0      ┆ null   ┆ … ┆ null ┆ null        ┆ null  ┆ null     │
│ 50%        ┆ 0.0      ┆ 3.0      ┆ null   ┆ … ┆ null ┆ null        ┆ null  ┆ null     │
│ 75%        ┆ 1.0      ┆ 3.0      ┆ null   ┆ … ┆ null ┆ null        ┆ null  ┆ null     │
│ max        ┆ 1.0      ┆ 3.0      ┆ male   ┆ … ┆ G    ┆ Southampton ┆ yes   ┆ 1.0      │
└────────────┴──────────┴──────────┴────────┴───┴──────┴─────────────┴───────┴──────────┘
```

W tym przykładzie załadowano dane o pasażerach Titanica za pomocą Polars i wyświetlono pierwsze kilka wierszy oraz podstawowe statystyki opisowe dla każdej kolumny. Operacje te będą bardzo szybkie w Polarsie, nawet przy dużych zbiorach danych.

Pozornie składnia Polarsa wydaje sie bardzo podobna do Pandasa, ale różni się w szczegółach, upodabniając go nieco do np. `Sparka`, używanego dla zastosowań Big Data.

Przykład 2: Filtrowanie i agregacja dnaych

```
# Filtrowanie danych - tylko mężczyźni powyżej 30 lat
filtered_data = titanic_data.filter((pl.col('sex') == 'male') & (pl.col('age') > 30))

filtered_data
```

shape: (202, 15)

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	emba
i64	i64	str	f64	i64	i64	f64	str	str	str	bool	str	
0	3	"male"	35.0	0	0	8.05	"S"	"Third"	"man"	true	null	"South
0	1	"male"	54.0	0	0	51.8625	"S"	"First"	"man"	true	"E"	"South
0	3	"male"	39.0	1	5	31.275	"S"	"Third"	"man"	true	null	"South
0	2	"male"	35.0	0	0	26.0	"S"	"Second"	"man"	true	null	"South
1	2	"male"	34.0	0	0	13.0	"S"	"Second"	"man"	true	"D"	"South
…	…	…	…	…	…	…	…	…	…	…	…	
0	1	"male"	31.0	0	0	50.4958	"S"	"First"	"man"	true	"A"	"South
0	1	"male"	33.0	0	0	5.0	"S"	"First"	"man"	true	"B"	"South
0	3	"male"	47.0	0	0	9.0	"S"	"Third"	"man"	true	null	"South
0	3	"male"	33.0	0	0	7.8958	"S"	"Third"	"man"	true	null	"South
0	3	"male"	32.0	0	0	7.75	"Q"	"Third"	"man"	true	null	"Que

```
# Grupowanie danych - suma biletów dla różnych klas
fare_sum = titanic_data.group_by('class').agg(pl.sum('fare').alias('total_fare'))
print("\nSuma biletów dla różnych klas:\n", fare_sum)
```

```
Suma biletów dla różnych klas:
 shape: (3, 2)
┌────────┬────────────┐
│ class  ┆ total_fare │
│ ---    ┆ ---        │
│ str    ┆ f64        │
╞════════╪════════════╡
│ Third  ┆ 6714.6951  │
│ First  ┆ 18177.4125 │
│ Second ┆ 3801.8417  │
└────────┴────────────┘
```

Podobnie jak w Pandas, Polars pozwala na filtrowanie i grupowanie danych. W powyższym kodzie przefiltrowano dane, aby wyświetlić tylko mężczyzn powyżej 30 roku życia, a następnie pogrupowano dane według klasy pasażerów, sumując ceny biletów.

Widzimy jednak, że zachodzą spore różnice w składni. Polars wykorzystuje funkcyjne podejście do wywoływania operacji, podobnie jak `Apache Spark`.

Biblioteki do operacji matematycznych i statystycznych

NumPy

NumPy jest fundamentalną biblioteką Pythona do obliczeń numerycznych. Oferuje wsparcie dla efektywnego przechowywania i manipulacji dużymi tablicami oraz macierzami wielowymiarowymi. Posiada również bogaty zbiór funkcji matematycznych umożliwiających wykonywanie operacji na tych danych, takich jak wszelkiego rodzaju arytmetyka, przekształcenia algebraiczne, badanie funkcji, i inne.

Znajomość NumPy, zwłaszcza w zakresie macierzy jest **fundamentalną** umiejętnością dla działań w zakresie uczenia maszynowego.

Zastosowania NumPy

NumPy jest często używany do:

1. Tworzenia i manipulowania macierzami: dodawanie, mnożenie, transpozycja, itd.
2. Wszelkich innych działań z zakresu algebry liniowej, jak np. rozwiazywanie układów równań.

Dodatkowe materiały

Przydatne linki - NumPy

1. Oficjalny przewodnik Numpy
2. Seria przewodników po funkcjonalnościach, o różnych poziomach zaawansowania

Przykład 1: Tworzenie macierzy i podstawowe operacje na nich

```
import numpy as np

# Tworzenie macierzy 3x3
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Transpozycja macierzy
A_transposed = A.T

# Mnożenie macierzy
B = np.array([[9, 8, 7],
              [6, 5, 4],
              [3, 2, 1]])
result = np.dot(A, B)

print("Macierz A:\n", A)
print("Macierz A po transpozycji:\n", A_transposed)
print("Wynik mnożenia A i B:\n", result)
```

```
Macierz A:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Macierz A po transpozycji:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Wynik mnożenia A i B:
 [[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
```

W powyższym kodzie utworzono macierz A o wymiarach 3x3, a następnie dokonano jej transpozycji. Następnie wykonano operację mnożenia macierzy A z macierzą B za pomocą funkcji np.dot.

Przykład 2: Rozwiązywanie układów równań liniowych

W poniższym przykładzie wykorzystamy numpy do rozwiazania układu równań liniowych:

$$\begin{cases} x + 2y & = 4 \\ 3x - 5y & = 1 \end{cases}$$

```python
X = np.array([
    [1., 2.],
    [3, -5]
])
y = np.array([4, 1])

coef = np.linalg.solve(X, y)
print(f"Rozwiązanie układu to: [x,y] = {coef}")
```

```
Rozwiązanie układu to: [x,y] = [2. 1.]
```

SciPy

SciPy to rozbudowana biblioteka dla zaawansowanych obliczeń naukowych. Oferuje funkcje do optymalizacji procesów, rozwiązywania równań różniczkowych, statystyki, przetwarzania sygnałów i wielu innych zadań. W tej sekcji skupimy się na module `scipy.stats`, który zawiera zestaw narzędzi statystycznych, w tym rozkłady prawdopodobieństwa, testy statystyczne, oraz funkcje estymacji.

Zastosowania SciPy

SciPy jest używany do:

1. Przeprowadzania testów statystycznych (np. testu t-Studenta, testu chi-kwadrat, etc).
2. Estymacji parametrów rozkładów.
3. Analizy regresji i korelacji.

Dodatkowe materiały

Przydatne linki - SciPy

1. Przewodnik po module Scipy Stats
2. Kompleksowy kurs Scipy+Numpy

Przykład 1: test T-studenta dla dwóch próbek

```
from scipy import stats

# Przykładowe dane: dwie próbki
sample1 = [5, 7, 8, 9, 10]
sample2 = [6, 9, 7, 12, 11]

# Test t-Studenta
t_stat, p_value = stats.ttest_ind(sample1, sample2)

print("t-statystyka:", t_stat)
print("p-wartość:", p_value)
```

```
t-statystyka: -0.8401680504168061
p-wartość: 0.4252097205513896
```

W powyższym kodzie przeprowadzono test t-Studenta dla dwóch niezależnych próbek danych. Wynikiem jest t-statystyka oraz p-wartość, która mówi nam o tym, czy możemy odrzucić hipotezę zerową o równości średnich w populacji.

Przykad 2: Estymacja parametrów rozkładu

```
# Przykładowe dane - normalnei nie wiemy, z jakiego rozkładu by pochodziły :)
data = np.random.normal(loc=10, scale=3, size=1000)

# Estymacja parametrów "nieznanego" rozkładu na podstawie danych
mu, std = stats.norm.fit(data)

print("Estymowana średnia (mu):", mu)
print("Estymowana odchylenie standardowe (std):", std)
```

```
Estymowana średnia (mu): 10.13091461552006
Estymowana odchylenie standardowe (std): 3.023561041396359
```

W tym przykładzie przeprowadzono estymację parametrów rozkładu normalnego (średniej oraz odchylenia standardowego) na podstawie wygenerowanej próbki danych.

Biblioteki do wizualizacji danych

Wizualizacja danych to kluczowy element pracy z danymi, umożliwiający efektywne komunikowanie wyników analizy, odkrywanie wzorców i trendów, a także weryfikację hipotez. W tej sekcji omówimy trzy popularne biblioteki do wizualizacji danych: `matplotlib`, `seaborn` oraz `plotly`.

Matplotlib

Matplotlib to podstawowa biblioteka do tworzenia wykresów w Pythonie. Jest niezwykle elastyczna i umożliwia tworzenie szerokiej gamy obrazów od prostych wykresów liniowych po skomplikowane wizualizacje 3D. Matplotlib jest często używany jako fundament dla innych bibliotek takich jak Seaborn. Oferuje kolosalną ilość funkcjonalności, których poznanie wymaga sporo czasu i wysiłku. W praktyce, zwykle uczymy się tych elementów, które najbardziej nam się przydadzą w prowadzonych analizach.

Zastosowania Matplolib

Matplotlib jest używany do:

1. Tworzenia wykresów liniowych, słupkowych, kołowych, histogramów, i wielu innych.
2. Dostosowywania elementów wykresu, takich jak osie, legendy, etykiety, kolory.
3. Tworzenia interaktywnych wykresów i animacji.

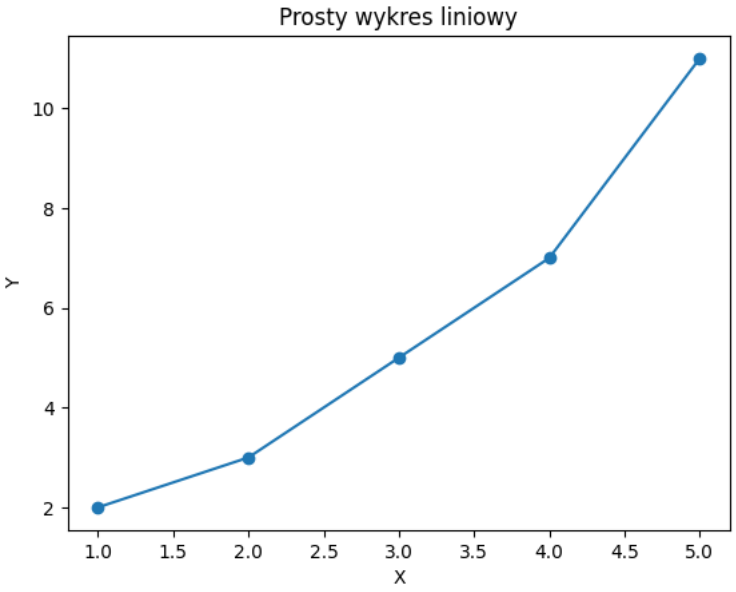
Dodatkowe materiały

Przydatne linki - Matplotlib

1. Oficjalny przewodnik po funkcjonalnościach
2. Przegląd podstawowych funkcji od DataCamp

Przykład 1: Prosty wykres liniowy

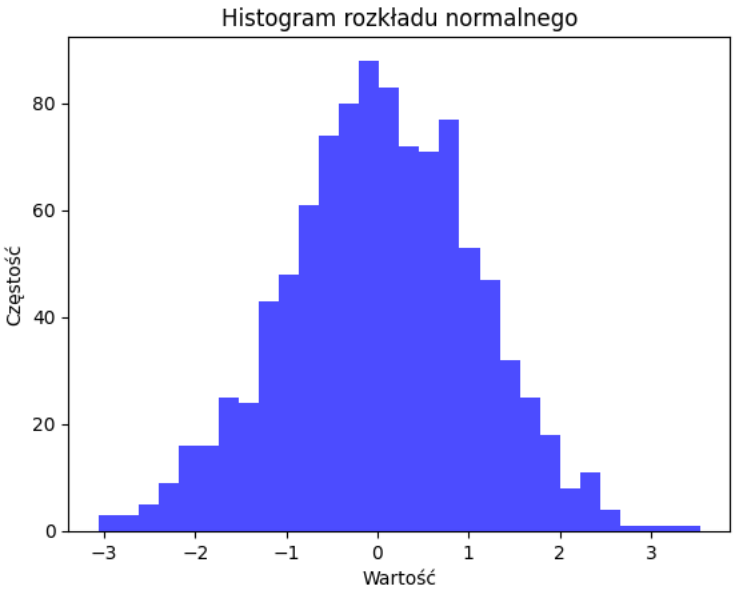
```
import matplotlib.pyplot as plt

# Przykładowe dane
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Tworzenie wykresu liniowego
plt.plot(x, y, marker='o')

# Dodawanie tytułu i etykiet osi
plt.title('Prosty wykres liniowy')
plt.xlabel('X')
plt.ylabel('Y')

# Wyświetlanie wykresu
plt.show()
```

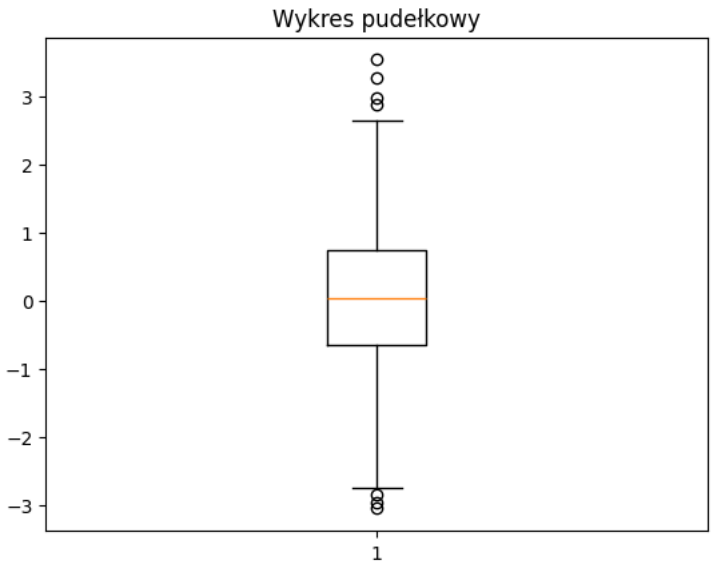


W powyższym przykładzie utworzono prosty wykres liniowy z pięcioma punktami. Dodano również tytuł wykresu oraz etykiety osi X i Y. Wykres został wygenerowany za pomocą funkcji plt.plot i wyświetlony z użyciem plt.show.

Przykład 2: Tworzenie histogramu i wykresu pudełkowego

```
# Generowanie losowych danych
data = np.random.normal(0, 1, 1000)

# Tworzenie histogramu
plt.hist(data, bins=30, alpha=0.7, color='blue')
plt.title('Histogram rozkładu normalnego')
plt.xlabel('Wartość')
plt.ylabel('Częstość')
plt.show()

# Tworzenie wykresu pudełkowego
plt.boxplot(data)
plt.title('Wykres pudełkowy')
plt.show()
```





W powyższym kodzie najpierw wygenerowano losowy zbiór danych o rozkładzie normalnym. Następnie utworzono histogram dla tych danych, a także wykres pudełkowy, który pomaga wizualizować rozkład oraz identyfikować wartości odstające.

Seaborn

Seaborn to biblioteka zbudowana na bazie Matplotlib, która upraszcza proces tworzenia atrakcyjnych wizualnie wykresów. Seaborn oferuje domyślne style i palety kolorów, które poprawiają estetykę wykresów, a także dodatkowe funkcje ułatwiające wizualizację złożonych relacji między zmiennymi, jak wykresy rozrzutu z liniami regresji, wykresy skrzynkowe, czy wykresy gęstości.

Zastosowania Seaborn

Seaborn jest używany do:

1. Tworzenia bardziej zaawansowanych wykresów, jak wykresy korelacji, heatmapy.
2. Wizualizacji rozkładów danych, takich jak wykresy gęstości, violin ploty.
3. Łatwego łączenia danych kategorycznych z ilościowymi na wykresach.

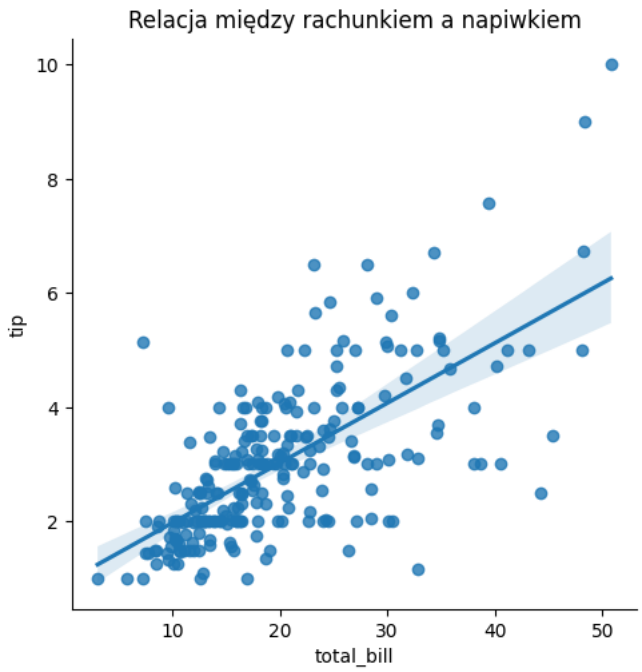
Dodatkowe materiały

Przydatne linki - Seaborn

1. Oficjalny przewodnik po bibliotece
2. Kurs Seaborn od DataCamp

Przykład 1: Tworzenie wykresu rozrzutu z linią regresji

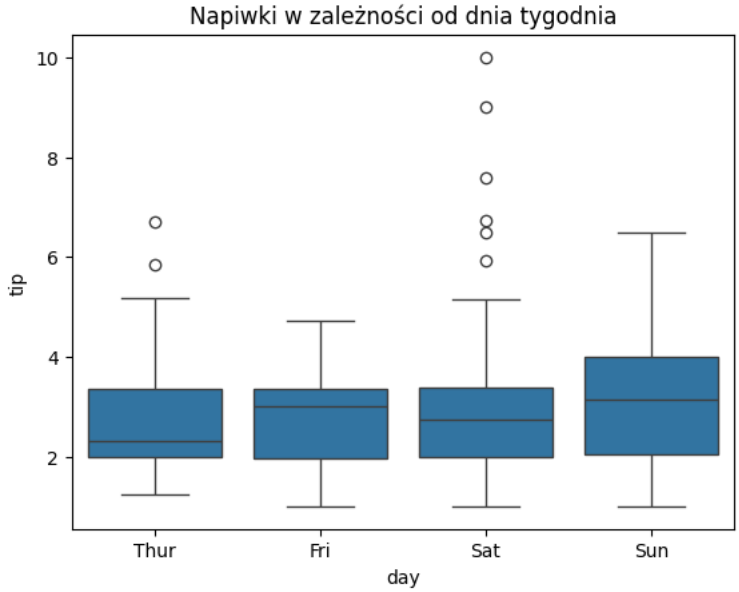
```
import seaborn as sns

# Wczytywanie danych z seaborn
tips = sns.load_dataset('tips')

# Tworzenie wykresu rozrzutu z linią regresji
sns.lmplot(x='total_bill', y='tip', data=tips)

# Dodawanie tytułu wykresu
plt.title('Relacja między rachunkiem a napiwkiem')
plt.show()
```

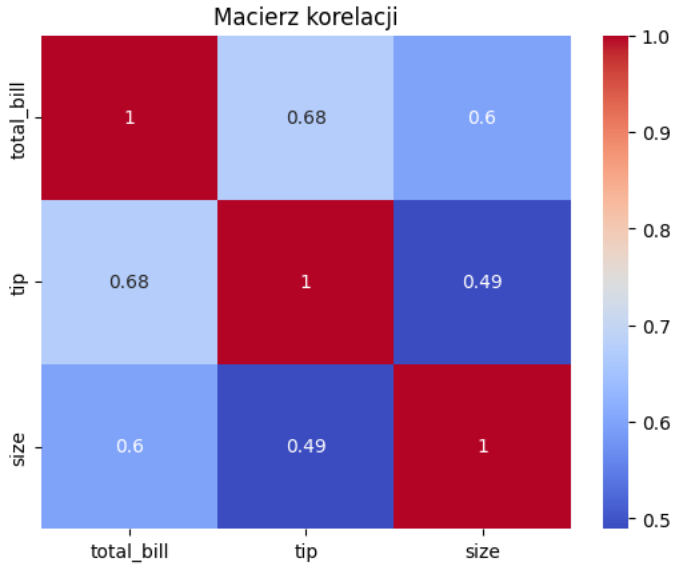


W powyższym przykładzie użyto wbudowanego w Seaborn zbioru danych tips i utworzono wykres rozrzutu, przedstawiający relację między wysokością rachunku a napiwkiem, z nałożoną linią regresji. Wykres jest automatycznie dostosowany do estetyki Seaborn.

Przykład 2: tworzenie wykresu pudełkowego (boxplot) i heatmapy

```
# Wykres skrzynkowy dla napiwków w zależności od dnia tygodnia
sns.boxplot(x='day', y='tip', data=tips)
plt.title('Napiwki w zależności od dnia tygodnia')
plt.show()

# Tworzenie macierzy korelacji
corr = tips[['total_bill', 'tip', 'size']].corr()

# Heatmapa korelacji
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Macierz korelacji')
plt.show()
```

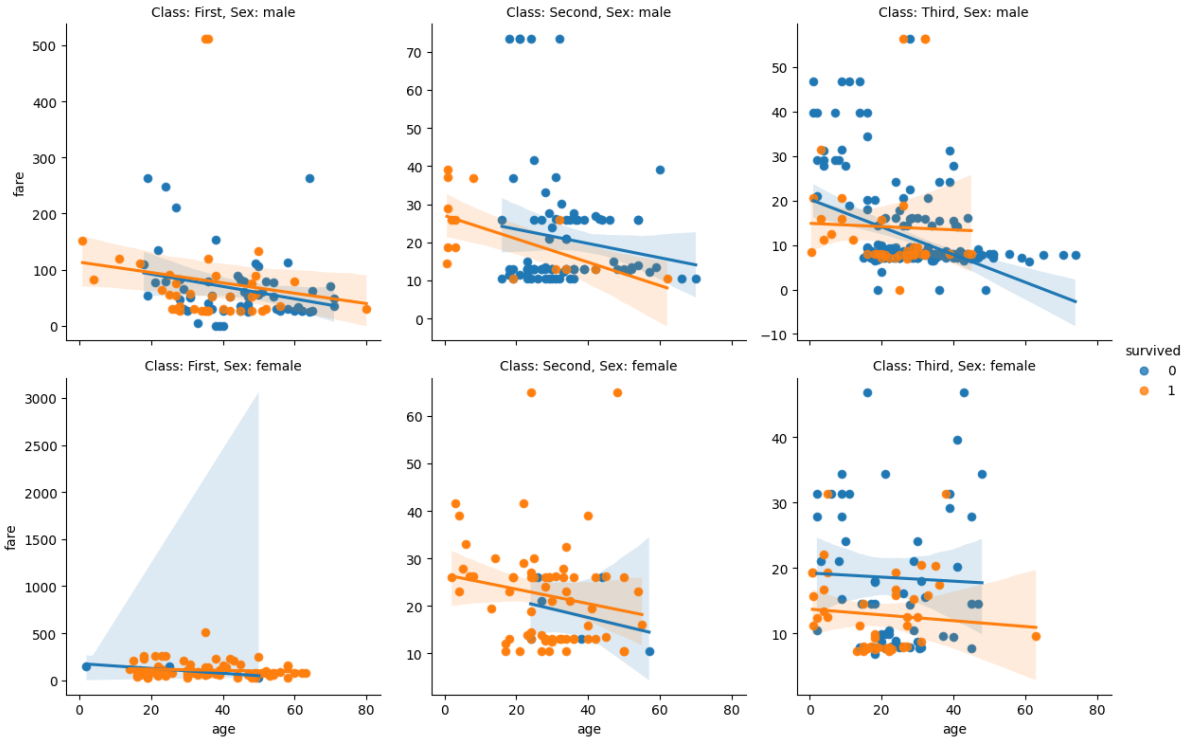




Najpierw stworzono wykres skrzynkowy, który porównuje rozkład napiwków w zależności od dnia tygodnia. Następnie, na podstawie macierzy korelacji obliczonej dla danych z tips, wygenerowano heatmapę z zaznaczonymi wartościami korelacji pomiędzy różnymi zmiennymi.

Przykład 3: Tworzenie zaawansowanych wykresów wielofacetowych

```
import seaborn as sns

# Load the Titanic dataset
titanic_data = sns.load_dataset('titanic')

# Create a FacetGrid with multiple facets
g = sns.FacetGrid(titanic_data, col='class', row='sex', hue='survived', height=4, sharey=False)

# Add a scatter plot to each facet
g.map(sns.scatterplot, 'age', 'fare')

# Add a regression line to each facet
g.map(sns.regplot, 'age', 'fare')

# Set titles for each facet
g.set_titles('Class: {col_name}, Sex: {row_name}')

# Add a legend
g.add_legend()

# Show the plot
plt.show()
```



W przykładzie powyżej wykonano tzw. wykres wielofasetowy z użyciem Seaborn, dzieląc zbiór danych na różne kategorie i przedstawiając na kilku planszach. Jednocześnie nałożono także linie regresji.

Plotly

Plotly to biblioteka do tworzenia interaktywnych wykresów, która jest idealna do publikowania wyników analizy danych online. W przeciwieństwie do Matplotlib czy Seaborn, Plotly pozwala na tworzenie interaktywnych wizualizacji, które można łatwo integrować z aplikacjami internetowymi. Plotly obsługuje szeroki zakres wykresów, od podstawowych po zaawansowane, takie jak wykresy 3D, mapy geograficzne, wykresy sankey'ego i wiele innych.

Co ważne, Plotly można operować na kilku poziomach złożoności:

1. `Plotly Express` - wysokopoziomowy interfejs do tworzenia wykresów, który pozwala na szybkie generowanie wykresów z minimalną ilością kodu.
2. `Plotly Graph Objects` - niskopoziomowy interfejs, który umożliwia pełną kontrolę nad wyglądem i zachowaniem wykresu.

Zastosowania Plotly

Plotly jest używany do:

1. Tworzenia interaktywnych wykresów, które można łatwo publikować online.
2. Tworzenia zaawansowanych wykresów, takich jak wykresy 3D, mapy geograficzne, wykresy sankey'ego.
3. Integracji z aplikacjami internetowymi.

Dodatkowe materiały

Przydatne linki - Plotly

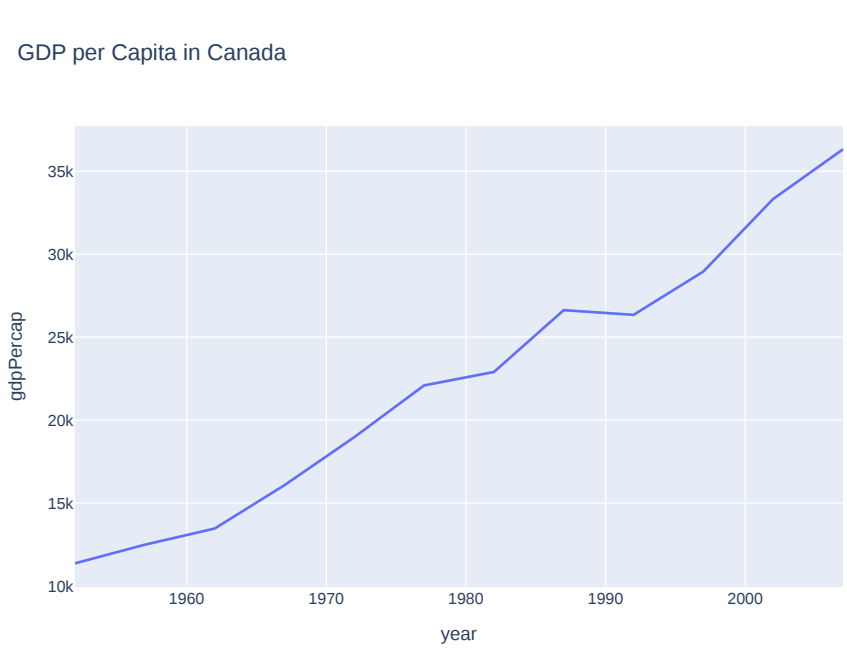
1. Oficjalny przewodnik po funkcjach Plotly
2. Kurs Plotly Express od DataCamp

Renderowanie wykresów w notebookach i książce

Wykresy tworzone w Plotly są pod spodem konwertowane do HTML. Z tego względu nie zawsze prawidłowo renderują się w postaci Jupyter Notebooka oraz w materiałach takich, jak ta książka. Warto zatem uruchomić kod w swoim środowisku, aby zobaczyć wykresy w pełnej krasie - np. w Google Colab albo lokalnie.

Przykład 1: Tworzenie interaktywnego wykresu liniowego

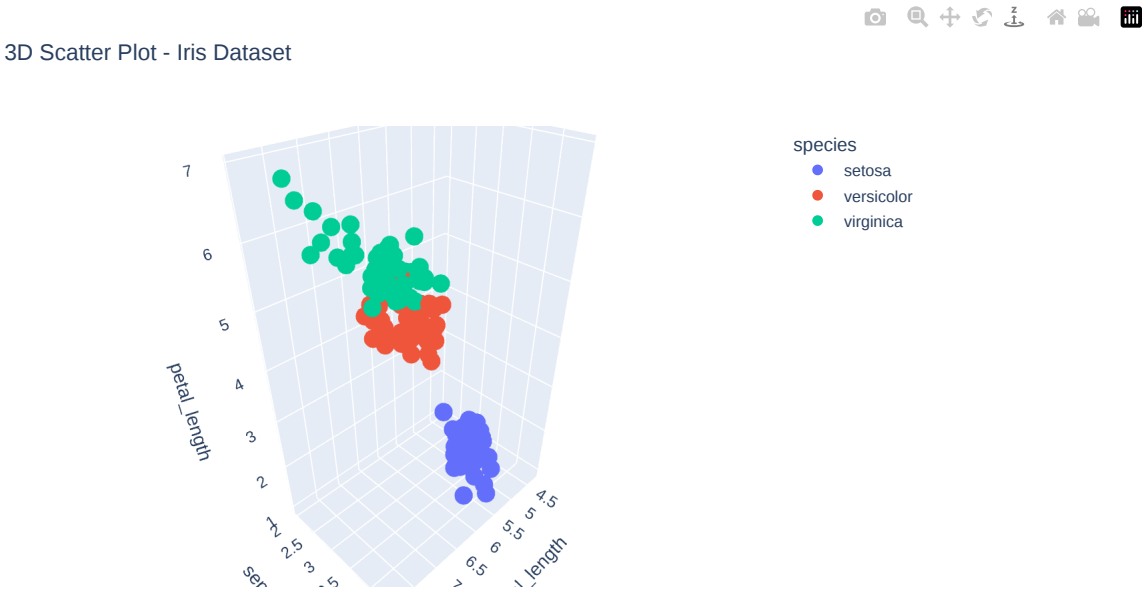
```
import plotly.express as px

# Przykładowe dane
df = px.data.gapminder().query("country=='Canada'")

# Tworzenie interaktywnego wykresu liniowego
fig = px.line(df, x='year', y='gdpPercap', title='GDP per Capita in Canada')

# Wyświetlanie wykresu
fig.show()
```



W tym przykładzie użyto wbudowanego zbioru danych gapminder i utworzono interaktywny wykres liniowy przedstawiający wzrost PKB na mieszkańca w Kanadzie na przestrzeni lat. Wykres można eksplorować, np. przybliżając wybrane fragmenty lub klikając na punkty danych.

Przykład 2: Tworzenie wykresu 3D

```
# Przykładowe dane
df = px.data.iris()

# Tworzenie wykresu 3D
fig = px.scatter_3d(df, x='sepal_length', y='sepal_width', z='petal_length',
                    color='species', title='3D Scatter Plot - Iris Dataset')

# Wyświetlanie wykresu
fig.show()
```



W tym przykładzie stworzono trójwymiarowy wykres rozrzutu dla zbioru danych iris, który jest często używany do testowania algorytmów uczenia maszynowego. Kolory na wykresie reprezentują różne gatunki irysów, a wykres jest interaktywny, co umożliwia obracanie i przybliżanie przestrzeni 3D.

Biblioteki uczenia maszynowego

W poniższej sekcji omówimy wyłącznie **najważniejsze** biblioteki przeznaczone do uczenia maszynowego. Warto zaznaczyć, że w tej dziedzinie istnieje wiele innych narzędzi, które są używane w zależności od konkretnego problemu, takich jak np. `XGBoost`, `LightGBM`, `CatBoost` do modeli gradient boosting, czy `TensorFlow`, `PyTorch`, `Keras` do sieci neuronowych. Nie sposób tutaj wymienić i przedstawić ich wszystkich.

Co więcej - nawet w obrębie omówionych tutaj bibliotek jak `scikit-learn` znajduja się setki (jeśli nie tysiące) różnych algorytmów, które można wykorzystać do rozwiązania konkretnego problemu. Warto zatem zapoznać się z oficjalną dokumentacją, aby dowiedzieć się, jakie funkcjonalności są dostępne.

W tym opracowaniu chcemy przedstawić tylko przykłady zastosowania, oraz podstawowe funkcjonalności, które są dostępne w wybranych bibliotekach, wskazując punkt startowy do dalszej eksploracji narzędzi.

Scikit-learn

Scikit-learn to jedna z najbardziej wszechstronnych i najczęściej używanych bibliotek do uczenia maszynowego w Pythonie. Oferuje narzędzia do przetwarzania danych, uczenia nadzorowanego i nienadzorowanego, a także do oceny i wyboru modeli. Scikit-learn jest idealny do szybkiego prototypowania modeli oraz do realizacji klasycznych zadań ML, takich jak klasyfikacja, regresja, klasteryzacja czy redukcja wymiarów.

Ważne jest, że Scikit-learn udosstępnia **spójne API**, czyli interfejs programistyczny, który jest taki sam dla wszystkich algorytmów. Dzięki temu, nauka i stosowanie różnych modeli jest znacznie ułatwione!

Zastosowania Scikit-learn

Scikit-learn jest używany do:

1. Wszelkich zadań z zakresu klasycznego uczenia maszynowego, (ang. *classic ML*), czyli klasyfikacji, regresji, klasteryzacji, redukcji wymiarów.
2. Walidacji krzyżowej i projektowania eksperymentów.
3. Wyliczania metryk i sprawdzania jakości modeli.

Dodatkowe materiały

Przydatne linki - Scikit-learn

Oficjalny przewodnik i podręcznik Scikit-learn - to jest **nieocenione źródło wiedzy**, podzielone na rozdziały i sekcje. Można je traktować jak pełnoprawny podręcznik do uczenia maszynowego.

Przykład 1: Klasyfikacja za pomocą SVM i przegląd typowego API

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Wczytywanie wbudowanego zbioru danych (Iris)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Podział na zbiór treningowy i testowy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Tworzenie modelu SVM i trenowanie
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Przewidywanie na zbiorze testowym
y_pred = model.predict(X_test)

# Ocena modelu
accuracy = accuracy_score(y_test, y_pred)
print(f"Dokładność modelu SVM: {accuracy:.2f}")
```

```
Dokładność modelu SVM: 1.00
```

W powyższym przykładzie użyto Scikit-learn do klasyfikacji gatunków irysów za pomocą maszyny wektorów nośnych (SVM) z liniowym kernelem. Model został przetrenowany na zbiorze treningowym, a następnie oceniony na zbiorze testowym, osiągając określoną dokładność.

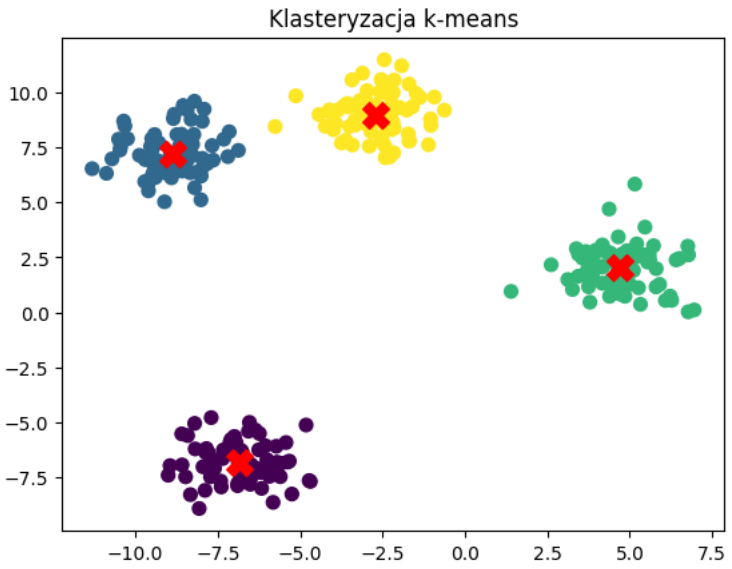
Charakterystyczna jest obecność następujących elementów API:

1. `fit` - trenowanie modelu na danych treningowych.
2. `predict` - predykcja na danych testowych.
3. Zastosowanie metryki, w tym przypadku `accuracy_score`, poprzez jej wywołanie z sygnaturą: `METRYKA(y_true, y_pred)`.

Taki sposób działania będzie wspólny dla niemal wszystkich modeli w Scikit-learn.

Przykład 2: Klasteryzacja z pomocą algorytmu K-średnich

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Wygenerowanie losowych danych 2D
X, _ = datasets.make_blobs(n_samples=300, centers=4, random_state=42)

# Klasteryzacja przy użyciu k-means
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Wizualizacja wyników
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X')
plt.title('Klasteryzacja k-means')
plt.show()
```



Powyższy kod pokazuje, jak użyć k-means do klasteryzacji danych 2D. Po wygenerowaniu losowych danych, zastosowano algorytm k-means, który podzielił dane na cztery klastry. Wynik został zwizualizowany na wykresie z zaznaczonymi centroidami klastrów.

PyTorch + Pytorch Lightning

PyTorch to jedna z najpopularniejszych bibliotek do uczenia głębokiego (ang. *deep learning*) oraz innych zadań związanych z ML. PyTorch został opracowany przez MEta AI Research i jest znany ze swojej elastyczności i łatwości użycia, szczególnie w badaniach naukowych. Umożliwia definiowanie, trenowanie i testowanie sieci neuronowych, wspiera również dynamiczne przetwarzanie grafów obliczeniowych, co jest jego dużą zaletą w porównaniu do innych frameworków.

`PyTorch Lightning` to wysokopoziomowy framework zbudowany na bazie PyTorch, który ma na celu uproszczenie procesu trenowania sieci neuronowych. PyTorch jest bardzo elastyczny, ale ta elastyczność może prowadzić do skomplikowanego kodu, szczególnie gdy projekt staje się bardziej złożony. PyTorch Lightning pomaga w rozdzieleniu kodu związanego z logiką modelu od kodu związanego z infrastrukturą, taką jak trenowanie, walidacja, logowanie i zarządzanie eksperymentami. Dzięki temu kod staje się bardziej czytelny, łatwiejszy do debugowania i skalowania.

Zastosowania PyTorch

PyTorch jest używany do:

1. Tworzenia i trenowania sieci neuronowych.
2. Badań naukowych i tworzenia innowacyjnych modeli sieci neuronowych.

W czym pomaga PyTorch Lightning:

1. **Struktura kodu**: Uporządkowanie kodu poprzez oddzielenie logiki modelu od logiki trenowania.
2. **Skalowanie**: Umożliwia łatwe przejście od trenowania na pojedynczym GPU do trenowania na klastrach wielo-GPU czy TPU bez konieczności zmiany kodu.
3. **Zarządzanie eksperymentami**: Automatyczne logowanie, punkty zapisu (ang. *checkpointing*), oraz wsparcie dla integracji z narzędziami do monitorowania eksperymentów (np. TensorBoard).
4. **Uproszczona walidacja i testowanie**: PyTorch Lightning ułatwia implementację pętli walidacyjnych i testowych, co jest szczególnie przydatne w większych projektach.

Dodatkowe materiały

Przydatne linki - PyTorch

1. Oficjalna dokumentacja PyTorch
2. Deep Learning Fundamendtals - kurs - **fantastyczny** (i darmowy) kurs poświęcony bibliotece Pytorch oraz Pytorch Lightning, prowadzony przez dr Sebastiana Raschkę, autora wielu narzędzi i publikacji z zakresu ML. Fantastyczne źródło wiedzy.

Przykład 1: Prosta sieć neuronowa do klasyfikacji danych tabelarycznych

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch.utils.data import DataLoader, TensorDataset
from typing import Tuple

# Wczytywanie i przygotowanie danych
data = load_wine()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.3, random_state=4

# Normalizacja danych
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Konwersja do tensora
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

# Tworzenie DataLoaderów
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Definiowanie modelu MLP
class MLP(nn.Module):
    def __init__(self, dims: Tuple[int, ...]):
        super().__init__()
        layers = []
        for dim_in, dim_out in zip(dims, dims[1:]):
            layer = nn.Linear(dim_in, dim_out)
            layers.append(layer)
            layers.append(nn.ReLU())
        self.net = nn.Sequential(*layers[:-1])

    def forward(self, x):
        out = self.net(x)
        return out

# Inicjalizacja modelu, funkcji kosztu i optymalizatora
torch.manual_seed(42)
model = MLP([X_train.shape[1], 32, 16, len(data.target_names)])
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Trenowanie modelu
for epoch in range(20):
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

# Ocena modelu
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Dokładność na zbiorze testowym: {100 * correct / total:.2f}%")
```

```
Epoch 1, Loss: 1.0892
Epoch 2, Loss: 1.0654
Epoch 3, Loss: 1.0424
Epoch 4, Loss: 1.0159
Epoch 5, Loss: 0.9890
Epoch 6, Loss: 0.9614
Epoch 7, Loss: 0.9311
Epoch 8, Loss: 0.8964
Epoch 9, Loss: 0.8612
Epoch 10, Loss: 0.8256
Epoch 11, Loss: 0.7891
Epoch 12, Loss: 0.7513
Epoch 13, Loss: 0.7106
Epoch 14, Loss: 0.6666
Epoch 15, Loss: 0.6277
Epoch 16, Loss: 0.5841
Epoch 17, Loss: 0.5435
Epoch 18, Loss: 0.5027
Epoch 19, Loss: 0.4608
Epoch 20, Loss: 0.4207
Dokładność na zbiorze testowym: 92.59%
```

W powyższym przykładzie zdefiniowano model MLP, który składa się z dwóch ukrytych warstw (z 32 i 16 neuronami). Dane wejściowe zostały przeskalowane przy użyciu standardowej normalizacji. Model został przetrenowany przez 20 epok, a następnie oceniony na zbiorze testowym.

Przykład 2: MLP do regresji na danych tabelarycznych

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch.utils.data import DataLoader, TensorDataset
from typing import Tuple

# Wczytywanie i przygotowanie danych
data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.3, random_state=

# Normalizacja danych
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Konwersja do tensora
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

# Tworzenie DataLoaderów
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Definiowanie modelu MLP do regresji
class MLPRegressor(nn.Module):
    def __init__(self, dims: Tuple[int, ...]):
        super().__init__()
        layers = []
        for dim_in, dim_out in zip(dims, dims[1:]):
            layer = nn.Linear(dim_in, dim_out)
            layers.append(layer)
            layers.append(nn.ReLU())
        self.net = nn.Sequential(*layers[:-1])

    def forward(self, x):
        out = self.net(x)
        return out

# Inicjalizacja modelu, funkcji kosztu i optymalizatora
torch.manual_seed(42)
model = MLPRegressor([X_train.shape[1], 32, 16, 1])
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Trenowanie modelu
for epoch in range(25):
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

# Ocena modelu
model.eval()
with torch.no_grad():
    test_predictions = model(X_test)
    mse = criterion(test_predictions, y_test)
    print(f"MSE na zbiorze testowym: {mse.item():.4f}")
```

```
Epoch 1, Loss: 1.4307
```

```
Epoch 2, Loss: 0.4944
```

```
Epoch 3, Loss: 0.4216
```

```
Epoch 4, Loss: 0.3952
```

```
Epoch 5, Loss: 0.3777
```

```
Epoch 6, Loss: 0.3673
```

```
Epoch 7, Loss: 0.3571
```

```
Epoch 8, Loss: 0.3505
```

```
Epoch 9, Loss: 0.3416
```

```
Epoch 10, Loss: 0.3340
```

```
Epoch 11, Loss: 0.3266
```

```
Epoch 12, Loss: 0.3242
```

```
Epoch 13, Loss: 0.3173
```

```
Epoch 14, Loss: 0.3122
```

```
Epoch 15, Loss: 0.3087
```

```
Epoch 16, Loss: 0.3120
```

```
Epoch 17, Loss: 0.3042
```

```
Epoch 18, Loss: 0.3020
```

```
Epoch 19, Loss: 0.2986
```

```
Epoch 20, Loss: 0.2961
```

```
Epoch 21, Loss: 0.2983
```

```
Epoch 22, Loss: 0.2934
```

```
Epoch 23, Loss: 0.2921
```

```
Epoch 24, Loss: 0.2904
```

```
Epoch 25, Loss: 0.2893
MSE na zbiorze testowym: 0.2992
```

W tym przykładzie zdefiniowano model MLP do zadania regresji. Model składa się z dwóch ukrytych warstw (32 i 16 neuronów) i jednej warstwy wyjściowej z jednym neuronem (do regresji). Model został przetrenowany przez 25 epok, a następnie oceniony na zbiorze testowym za pomocą miary błędu średniokwadratowego (MSE).

Przykład 3: Implementacja prostej sieci neuronowej w Pytorch Lightning

```
import torch
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from typing import Tuple
import pytorch_lightning as pl

# Wczytywanie i przygotowanie danych
data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.3, random_state=4

# Normalizacja danych
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Konwersja do tensora
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

# Tworzenie DataLoaderów
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Definiowanie modelu MLP za pomocą PyTorch Lightning
class IrisMLP(pl.LightningModule):
    def __init__(self, dims: Tuple[int, ...]):
        super().__init__()
        layers = []
        for dim_in, dim_out in zip(dims, dims[1:]):
            layer = torch.nn.Linear(dim_in, dim_out)
            layers.append(layer)
            layers.append(torch.nn.ReLU())
        self.network = torch.nn.Sequential(*layers[:-1])

    def forward(self, x):
        out = self.network(x)
        return out

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.forward(x)
        loss = F.cross_entropy(y_hat, y)
        self.log('train_loss', loss)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.forward(x)
        val_loss = F.cross_entropy(y_hat, y)
        self.log('val_loss', val_loss)
        return val_loss

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=0.001)

# Trenowanie modelu za pomocą PyTorch Lightning
model = IrisMLP([X_train.shape[1], 32, 16, len(data.target_names)])
trainer = pl.Trainer(max_epochs=20)
trainer.fit(model, train_loader, test_loader)
```

```
GPU available: False, used: False
```

```
TPU available: False, using: 0 TPU cores
```

```
HPU available: False, using: 0 HPUs
```

```
/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/pytorch_lightning/trainer/connectors/log
```

```
  | Name    | Type       | Params | Mode 
-----------------------------------------------
0 | network | Sequential | 739    | train
-----------------------------------------------
739       Trainable params
0         Non-trainable params
739       Total params
0.003     Total estimated model params size (MB)
6         Modules in train mode
0         Modules in eval mode
```

```
/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/pytorch_lightning/trainer/connectors/dat
/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/pytorch_lightning/trainer/connectors/dat
/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/pytorch_lightning/loops/fit_loop.py:298:
```

```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

W powyższym przykładzie:

1. Model: Zdefiniowano model MLP będący modułem Pytorch Lighting obsługującym predefiniowane funkcje takie, jak `forward`, `training_step`, `validation_step`, `configure_optimizers`.
2. Dane: Dane z zestawu Iris zostały przeskalowane, a następnie skonwertowane na tensory PyTorch.
3. Trening: Model został przetrenowany przez 20 epok. PyTorch Lightning automatycznie zarządza procesem trenowania, w tym logowaniem strat/kosztów (train_loss, val_loss) i optymalizacją modelu.

MLXtend

Mlxtend (Machine Learning Extensions) to biblioteka, która rozszerza możliwości Scikit-learn, oferując dodatkowe narzędzia, takie jak metody wyboru cech, łączenie klasyfikatorów, różnorodne algorytmy do klasyfikacji i regresji, a także funkcje do wizualizacji wyników. Mlxtend jest idealny dla osób, które chcą szybko przetestować różne techniki ML, które nie są wbudowane w Scikit-learn.

MLXtend pomaga też przy analizowaniu możliwości poszczególnych klas model, oraz ich własności. W tym podręczniku w rozdziale poświęconym obciążeniom indukcyjnym modeli ML, użyliśmy właśnie `MLXtend`.

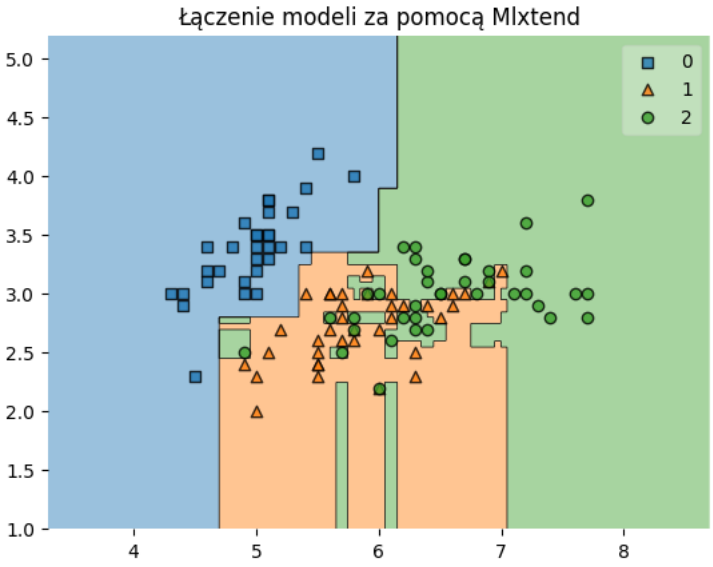
> **Zastosowania MLXtend**
>
> Mlxtend jest używany do:
>
> 1. Wyboru cech i redukcji wymiarów.
> 2. Łączenia klasyfikatorów i regresorów.
> 3. Wizualizacji wyników i analizy modeli.

Dodatkowe materiały

> **Przydatne linki - MLXtend**
>
> Oficjalna dokumentacja MLXtend - dokumentacja jest też dodatkowo bardzo dobrym podręcznikiem.

Przykład 1: Łączenie modeli klasyfikacji

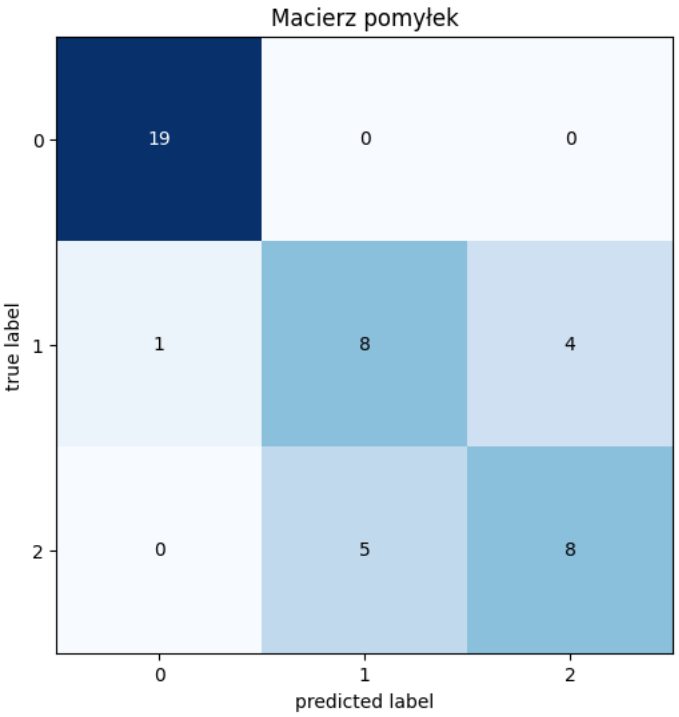
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from mlxtend.classifier import EnsembleVoteClassifier
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

# Wczytywanie danych
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X[:, :2], y, test_size=0.3, random_state=42)

# Definiowanie modeli bazowych
clf1 = RandomForestClassifier(random_state=1)
clf2 = GradientBoostingClassifier(random_state=1)

# Łączenie modeli
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2], voting='soft')

# Trenowanie i wizualizacja wyników

eclf.fit(X_train, y_train)
plot_decision_regions(X_train, y_train, clf=eclf),
plt.title('Łączenie modeli za pomocą Mlxtend')
plt.show()
```



W powyższym przykładzie wykorzystano Mlxtend do połączenia dwóch : lasu losowego i gradient boosting. Modele zostały połączone w celu stworzenia bardziej stabilnego i dokładnego modelu końcowego, a wynik został zwizualizowany za pomocą funkcji `plot_decision_regions`, ukazującą regiony działania/podejmowania decyzji przez modele.

Przykład 2: Wizualizacja macierzy pomyłek

```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

# Przewidywanie na zbiorze testowym
y_pred = eclf.predict(X_test)

# Tworzenie macierzy pomyłek
cm = confusion_matrix(y_test, y_pred)

# Wizualizacja macierzy pomyłek
fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(6, 6))
plt.title('Macierz pomyłek')
plt.show()
```



Kod ten pokazuje, jak za pomocą Mlxtend można łatwo wizualizować macierz pomyłek. Macierz ta przedstawia liczbę poprawnych i błędnych klasyfikacji dokonanych przez model, co jest kluczowym narzędziem do oceny jego wydajności.

Optuna

Optuna to biblioteka do automatycznej optymalizacji hiperparametrów, która pomaga w znalezieniu najlepszych ustawień parametrów modeli ML poprzez próbkowanie ich z różnych rozkładów. Optuna jest bardzo elastyczna i oferuje zaawansowane techniki optymalizacji, takie jak próbkowanie TPE (ang. *Tree-structured Parzen Estimator*) i optymalizacja oparta na modelu bayesowskim. Jest to szczególnie przydatne w automatyzacji procesu strojenia modeli, co jest jednym z bardziej czasochłonnych aspektów pracy nad ML.

Zastosowania Optuny

Optuna jest używana do:

1. Automatycznego strojenia hiperparametrów modeli ML.
2. Optymalizacji modeli ML w celu uzyskania najlepszych wyników.

Dodatkowe materiały

> **Przydatne linki - Optuna**
>
> Oficjalny przewodnik Optuny

Przykład: Optymalizacja parmetrów klasyfikatora

```
import optuna
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

optuna.logging.set_verbosity(optuna.logging.WARNING)

# Wczytywanie danych
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Funkcja celu
def objective(trial):
    C = trial.suggest_float('C', 1e-4, 1e2, log=True)
    gamma = trial.suggest_float('gamma', 1e-4, 1e1, log=True)
    model = SVC(C=C, gamma=gamma)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return accuracy_score(y_test, y_pred)

# Tworzenie study i optymalizacja
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

print("Najlepsze parametry:", study.best_params)
print("Najlepsza dokładność:", study.best_value)
```

```
Najlepsze parametry: {'C': 1.2637585051006384, 'gamma': 0.07568787616877622}
Najlepsza dokładność: 1.0
```

Biblioteki do pracy z szeregami czasowymi

Analiza szeregów czasowych jest jedna z kluczowych dziedzin w Data Science, szczególnie w kontekście prognozowania danych finansowych, czy ekonomicznych. Python oferuje różnorodne narzędzia do pracy z szeregami czasowymi. W tej sekcji omówimy dwie biblioteki: `statsmodels` oraz `statsforecast` z Nixtlaverse.

Zwłaszcza ta ostatnia rodzina/grupa bibliotek zgrupowana pod wspólną nazwą jest bardzo interesująca - oferuje m. in. moduły prognozowania opartego o uczenie maszynowe (`mlforecast`), oraz sieci neuronowe (`neuralforecast`) w tym samym ekosystemie.

Statsmodels

Statsmodels to wszechstronna biblioteka Pythona, która oferuje zaawansowane narzędzia do modelowania statystycznego. Biblioteka ta jest szczególnie użyteczna w analizie szeregów czasowych, oferując modele takie jak ARIMA, SARIMAX, modele stacjonarne oraz narzędzia do diagnostyki i oceny modeli.

Statsmodels - kłopoty z czytelnością

Wiele osób zauważa, że dokumentacja tego narzędzia pozostawia wiele do życzenia, podobnie jak czytelność i przejrzystość kodu. Jest to co prawda kwestia gustu, ale rzeczywiście, z punktu widzenia inżynierii oprogramowania, dobrych praktyk i przejrzystości kodu - `statsmodels` ma wiele do poprawy.

Ciężko jest znaleźć interesujące nas tematy w dokumentacji, a także zrozumieć, jakie są dostępne funkcjonalności. Warto zatem korzystać z dodatkowych materiałów, takich jak kursy online, czy przewodniki, które pomogą w zrozumieniu tej biblioteki. Niestety - nierzadko dokumentacja nijak się ma do tego, co aktualnie jest zaimplementowane w określonej wersji :/

Zastosowania Statsmodels

Statsmodels jest używany do:

1. Modelowania szeregów czasowych, w tym ARIMA, SARIMAX.
2. Diagnostyki modeli szeregów czasowych.
3. Analizy statystycznej i modelowania danych.

Dodatkowe materiały

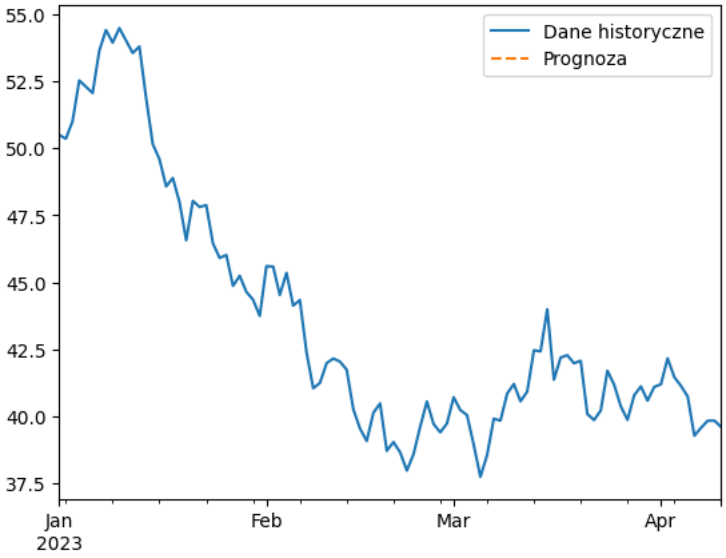
Przydatne linki - Statsmodels

Oficjalny przewodnik po funkcjach Statsmodels

Przykład 1: Modelowanie ARIMA dla danych szeregów czasowych

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from matplotlib import pyplot as plt

# Przykładowe dane: symulacja szeregu czasowego
np.random.seed(42)
data = np.cumsum(np.random.randn(100)) + 50
dates = pd.date_range('2023-01-01', periods=100, freq='D')
series = pd.Series(data, index=dates)

# Tworzenie i dopasowanie modelu ARIMA
model = sm.tsa.ARIMA(series, order=(1, 1, 1))
results = model.fit()

# Podsumowanie wyników modelu
print(results.summary())

# Prognozowanie przyszłych wartości
forecast = results.forecast(steps=10)
print("Prognoza na 10 dni:", forecast)

# Wizualizacja
series.plot(label='Dane historyczne')
plt.plot(forecast, label='Prognoza', linestyle='--')
plt.legend()
plt.show()
```

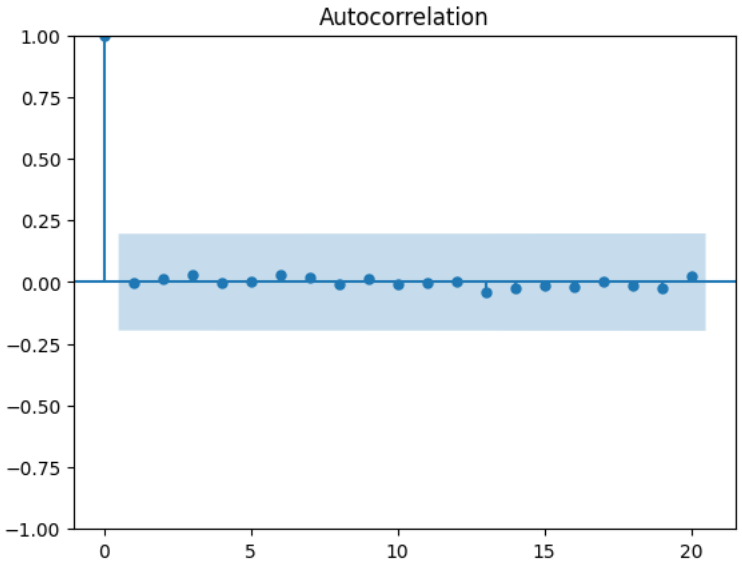
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  100
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -131.425
Date:                Sat, 24 Aug 2024   AIC                            268.850
Time:                        17:13:30   BIC                            276.635
Sample:                    01-01-2023   HQIC                           272.000
                         - 04-10-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3996      5.649      0.071      0.944     -10.672      11.471
ma.L1         -0.4156      5.591     -0.074      0.941     -11.374      10.543
sigma2         0.8329      0.123      6.780      0.000       0.592       1.074
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 0.53
Prob(Q):                              0.95   Prob(JB):                         0.77
Heteroskedasticity (H):               0.92   Skew:                            -0.16
Prob(H) (two-sided):                  0.81   Kurtosis:                         2.82
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
Prognoza na 10 dni: 2023-04-11    39.618833
2023-04-12    39.620226
2023-04-13    39.620782
2023-04-14    39.621004
2023-04-15    39.621093
2023-04-16    39.621129
2023-04-17    39.621143
2023-04-18    39.621148
2023-04-19    39.621151
2023-04-20    39.621152
Freq: D, Name: predicted_mean, dtype: float64
```

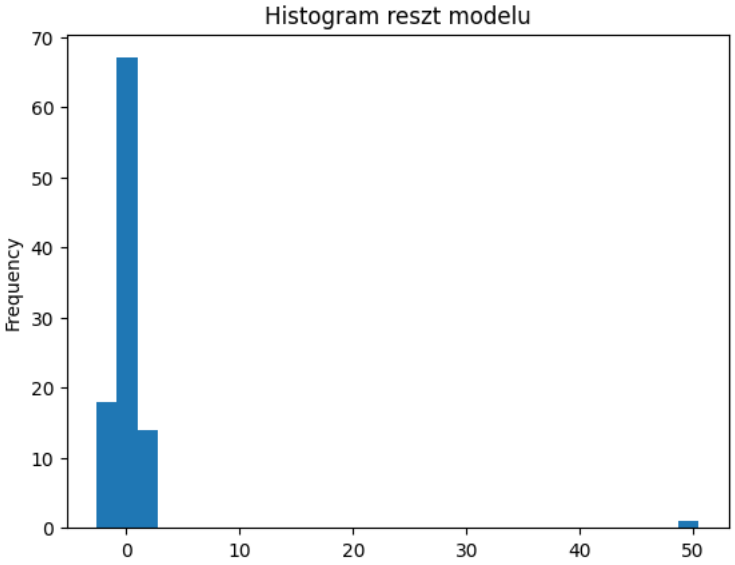


W tym przykładzie użyto Statsmodels do modelowania symulowanego szeregu czasowego za pomocą modelu ARIMA (ang. *autoregressive integrated moving average*). Model został dopasowany do danych, a następnie wykorzystany do prognozowania przyszłych wartości. Wyniki prognozy zostały zwizualizowane na wykresie, pokazując przewidywane wartości w stosunku do danych historycznych.

Przykład 2: Diagnostyka i wizualizacja reszt modelu ARIMA

```
# Diagnostyka reszt modelu
residuals = results.resid

# Autokorelacja reszt
sm.graphics.tsa.plot_acf(residuals)
plt.show()

# Histogram reszt
residuals.plot(kind='hist', bins=30)
plt.title('Histogram reszt modelu')
plt.show()
```





Powyższy kod pokazuje, jak przeprowadzić diagnostykę reszt modelu ARIMA. Analiza autokorelacji reszt pomaga ocenić, czy model dobrze dopasowuje się do danych, podczas gdy histogram reszt pozwala sprawdzić ich rozkład, co jest istotne dla oceny jakości modelu.

Statsforecast (Nixtlaverse)

Statsforecast to nowoczesna biblioteka zaprojektowana do szybkiego i skalowalnego prognozowania szeregów czasowych. Należy do ekosystemu Nixtlaverse, który skupia narzędzia optymalizujące prognozowanie za pomocą różnych technik - od klasycznej analizy szeregów czasowych po metody ML. Statsforecast oferuje implementacje popularnych modeli, takich jak ETS, ARIMA, i inne, zoptymalizowane pod kątem szybkości i wydajności. Dzięki wykorzystaniu wielowątkowości i innych technik optymalizacji, Statsforecast pozwala na szybkie prognozowanie setek tysięcy szeregów czasowych jednocześnie.

Co więcej - w porównaniu do Statsforecast - dokumentacja i przewodniki Nixlaverse są znakomicie i przejrzyście napisane!!! Razem z pokrewnymi bibliotekami `mlforecast` i `neural` tworzą spójny ekosystem.

Zastosowania Statsforecast

Statsforecast jest używany do:

1. Prognozowania szeregów czasowych za pomocą modeli ARIMA, ETS, i innych.
2. Szybkiego i skalowalnego prognozowania wielu szeregów czasowych.

Dodatkowe materiały

Przydatne linki - Statsforecast

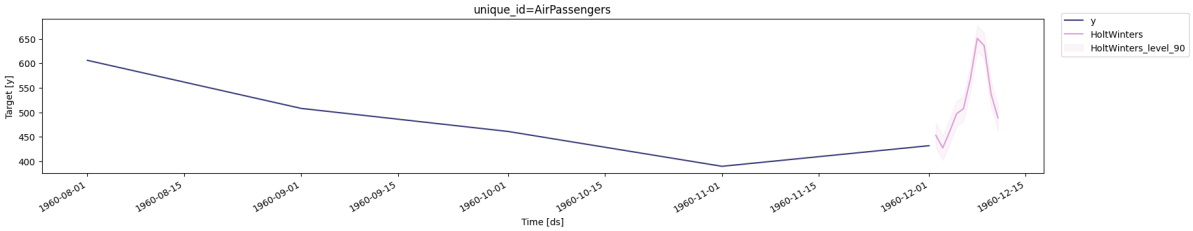
Oficjalny przewodnik po funkcjach Statsforecast

Przykład 1: Szybkie prognozowanie szeregu czasowego za pomocą metody Holta-Wintersa

```
from statsforecast import StatsForecast
from statsforecast.models import HoltWinters
import pandas as pd

# Przykładowe dane: air passengers
df = pd.read_csv('https://datasets-nixtla.s3.amazonaws.com/air-passengers.csv', parse_dates=['ds'])

# Tworzenie modelu Holta-Wintersa i prognozowanie
sf = StatsForecast(models=[HoltWinters(season_length=12)], freq='D')

# Uczenie modelu na podstawie historycznych danych
sf.fit(df=df)

# Prognozowanie przyszłych wartości
forecast_df = sf.predict(h=10, level=[90])

# Wyświetlanie prognozy
print(forecast_df)

# Wizualizacja prognozy
sf.plot(df.tail(5), forecast_df, level=[90])
```

```
/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/statsforecast/core.py:492: FutureWarning

In a future version the predictions will have the id as a column. You can set the `NIXTLA_ID_AS_COL` envi

/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/statsforecast/core.py:1447: FutureWarnir

Passing the ids as the index is deprecated. Please provide them as a column instead.
```

unique_id	ds	HoltWinters	HoltWinters-lo-90	HoltWinters-hi-90
AirPassengers	1960-12-02	453.088745	428.115234	478.062256
AirPassengers	1960-12-03	427.476532	402.366119	452.586975
AirPassengers	1960-12-04	460.631775	435.381927	485.881622
AirPassengers	1960-12-05	497.036804	471.645081	522.428528
AirPassengers	1960-12-06	507.507538	481.971436	533.043640
AirPassengers	1960-12-07	567.922485	542.239502	593.605408
AirPassengers	1960-12-08	650.842590	625.010376	676.674866
AirPassengers	1960-12-09	635.878601	609.894531	661.862610
AirPassengers	1960-12-10	537.471191	511.332947	563.609497
AirPassengers	1960-12-11	488.687286	462.392334	514.982239



W powyższym przykładzie użyto Statsforecast do prognozowania szeregów czasowych przy użyciu modelu Holta-Wintersa. Wyniki prognozy zostały wyświetlone i zwizualizowane, pokazując prognozowane wartości na kolejne 10 dni.

Przykład 2: Szkolenie wielu modeli jednocześnie

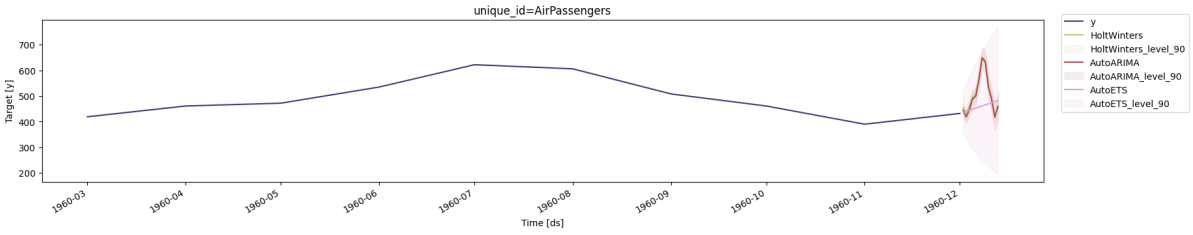
```python
from statsforecast import StatsForecast
from statsforecast.models import HoltWinters, AutoARIMA, AutoETS
import pandas as pd

# Przykładowe dane: air passengers
df = pd.read_csv('https://datasets-nixtla.s3.amazonaws.com/air-passengers.csv', parse_dates=['ds'])

# Szkolenie kilku modeli jednocześnie
sf = StatsForecast(models=[
    HoltWinters(season_length=12),
    AutoARIMA(season_length=12),
    AutoETS()], freq='D')

# Uczenie modelu na podstawie historycznych danych
sf.fit(df=df)

# Prognozowanie przyszłych wartości
forecast_df = sf.predict(h=12, level=[90])

# Wyświetlanie prognozy
print(forecast_df)

# Wizualizacja prognozy
sf.plot(df.tail(10), forecast_df, level=[90])
```

```
/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/statsforecast/core.py:492: FutureWarning

In a future version the predictions will have the id as a column. You can set the `NIXTLA_ID_AS_COL` envi

/home/fwojcik/anaconda3/envs/dsbook/lib/python3.11/site-packages/statsforecast/core.py:1447: FutureWarnir

Passing the ids as the index is deprecated. Please provide them as a column instead.
```

	ds	HoltWinters	HoltWinters-lo-90	HoltWinters-hi-90	\
unique_id					
AirPassengers	1960-12-02	453.088745	428.115234	478.062256	
AirPassengers	1960-12-03	427.476532	402.366119	452.586975	
AirPassengers	1960-12-04	460.631775	435.381927	485.881622	
AirPassengers	1960-12-05	497.036804	471.645081	522.428528	
AirPassengers	1960-12-06	507.507538	481.971436	533.043640	
AirPassengers	1960-12-07	567.922485	542.239502	593.605408	
AirPassengers	1960-12-08	650.842590	625.010376	676.674866	
AirPassengers	1960-12-09	635.878601	609.894531	661.862610	
AirPassengers	1960-12-10	537.471191	511.332947	563.609497	
AirPassengers	1960-12-11	488.687286	462.392334	514.982239	
AirPassengers	1960-12-12	420.511292	394.057220	446.965363	
AirPassengers	1960-12-13	463.407196	436.791565	490.022827	

	AutoARIMA	AutoARIMA-lo-90	AutoARIMA-hi-90	AutoETS	\
unique_id					
AirPassengers	444.300049	424.971436	463.628693	436.156677	
AirPassengers	418.210022	394.616974	441.803070	440.317139	
AirPassengers	446.237030	418.134003	474.340057	444.477600	
AirPassengers	488.228943	456.491608	519.966248	448.638092	
AirPassengers	499.231354	464.168854	534.293884	452.798553	
AirPassengers	562.230652	524.150452	600.310852	456.959015	
AirPassengers	649.230835	608.349976	690.111694	461.119476	
AirPassengers	633.230774	589.730652	676.730896	465.279938	
AirPassengers	535.230774	489.260010	581.201599	469.440399	
AirPassengers	488.230804	439.915619	536.545959	473.600891	
AirPassengers	417.230804	366.679810	467.781799	477.761353	
AirPassengers	459.230804	406.538788	511.922821	481.921814	

	AutoETS-lo-90	AutoETS-hi-90
unique_id		
AirPassengers	359.774292	512.539062
AirPassengers	331.475494	549.158813
AirPassengers	310.160889	578.794312
AirPassengers	292.366394	604.909790
AirPassengers	276.760742	628.836365
AirPassengers	262.666473	651.251587
AirPassengers	249.683929	672.555054
AirPassengers	237.554657	693.005249
AirPassengers	226.100342	712.780457
AirPassengers	215.192047	732.009705
AirPassengers	204.733047	750.789612
AirPassengers	194.648727	769.194885



[1] Autor sekcji: dr Filip Wójcik

Informacje

Strony w tej części zawierają dodatkowe informacje, takie jak bibliotgrafia i indeks pojeć.

Indeks

JupyterBook na bieżąco tworzy indeks pojęć, które są zdefiniowane w tekście. Poniżej znajdziesz listę wszystkich pojęć, które zostały zdefiniowane w tym opracowaniu.

Indeks

Bibliografia

[AMMIL12] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*. Volume 4. AMLBook New York, 2012.

[Cic07] Paweł Cichosz. *Systemy uczące się*. Wydawnictwa Naukowo-Techniczne, 2007.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[Fla12] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge university press, 2012.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. book in preparation for mit press. *URL¡ http://www. deeplearningbook. org*, 2016.

[HTFF09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Volume 2. Springer, 2009.

[Lea97] Machine Learning. Tom mitchell. *Publisher: McGraw Hill*, 1997.

[MN07] Tengyu Ma and Andrew Ng. Cs229 lecture notes. In *CS229 Lecture notes*. 2007. URL: https://api.semanticscholar.org/CorpusID:628573.

[MRTB12] M Mohri, A Rostamizadeh, A Talwalkar, and F Bach. *Foundations of Machine Learning*. MIT Press, 2012. ISBN 9780262018258.

[SW17] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning and data mining*. Springer Publishing Company, Incorporated, 2017.

[Sim83] Herbert A Simon. Why should machines learn? In *Machine learning*, pages 25–37. Elsevier, 1983.

[Woj21] Filip Wójcik. *Supporting decision processes in the organization with the use of associative analysis algorithms*. PhD thesis, Wroclaw University of Economics and Buisiness, Wrocław, Poland, November 2021. Available at https://wir.ue.wroc.pl/info/phd/UEWR3483636aa91d4d2d81133ea64251c351/.