Microsoft

Wprowadzenie do uczenia maszynowego

według Esposito

Professional

Dino Esposito
Francesco Esposito

Microsoft

Dino Esposito
Francesco Esposito

Wprowadzenie do uczenia maszynowego

Przekład Joanna Zatorska

APN Promise, Warszawa 2020

Wprowadzenie do uczenia maszynowego

Authorized translation from the English language edition, entitled: Introducing Machine Learning, ISBN: 978-0-13-556566-7, by Dino Esposito and Francesco Esposito, published by Pearson Education, Inc, publishing as Microsoft Press, a Division of Microsoft Corporation.

Polish language edition published by APN PROMISE S.A., Copyright © 2020

Autoryzowany przekład z wydania w języku angielskim, zatytułowanego: Introducing Machine Learning, ISBN: 978-0-13-556566-7, by Dino Esposito and Francesco Esposito, opublikowanego przez Pearson Education, Inc, publikującego jako Microsoft Press, oddział Microsoft Corporation.

APN PROMISE SA, ul. Domaniewska 44a, 02-672 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

ISBN: 978-83-7541-422-6 (druk), 978-83-7541-423-3 (ebook)

Przekład: Joanna Zatorska
Redakcja: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Dedykacje

Dla Micheli i jej marzeń

Dla moich ukochanych, którym naprawdę musiałem zadedykować książkę

Spis treści

Część V Finalne uwagi

Podziękowania

Pisanie książki wspólnie z własnym synem jest szczególnym doświadczeniem, nawet jeśli jest to jedna z kolejnych napisanych książek. W tej książce ująłem w słowa (mam nadzieję jasne) myśli Francesca, jego wizję i głębokie, w dużej mierze niewyjaśnione zrozumienie uczenia maszynowego. Podczas pisania wiele się nauczyłem i mam nadzieję, że tyle samo nauczą się czytelnicy.

Zdołałem się tyle nauczyć głównie dzięki dwóm osobom.

Nie pierwszy raz pisałem pod technicznym nadzorem Cesara De la Torre Llorente i zawsze było to wspaniałe doświadczenie. Uwielbiam jego pragmatyzm i dokładność w opracowywaniu, jeszcze przed projektowaniem, produktów programistycznych. Obecnie jest on głównym menedżerem produktu do spraw platformy .NET w firmie Microsoft i odpowiada za rozwój ML.NET. Ta książka nie dotyczy tylko platformy ML.NET, ale dzięki pomocy Cesara udało się dokładnie opisać podejście do uczenia maszynowego w .NET.

Ze źródłami energii odnawialnej związany jest mało znany aspekt: do ich wykorzystania potrzebne jest inteligentne oprogramowanie. Jest ono kluczowe, przynajmniej na poziomie funkcjonalnym, ponieważ zapewnia dokładne prognozy dotyczące poziomu produkcji, przerw w dostawie, awarii i cen. Nie sądzę, aby na planecie żyło wiele osób, które miałyby dziesięcioletnie doświadczenie w dziedzinie, którą dość niedawno nazwano sztuczną inteligencją. Tiago Santos był naszym przewodnikiem w losowym lesie uczenia maszynowego oraz w rzeczywistym świecie sztucznej inteligencji. „AI jest tylko oprogramowaniem" stało się naszym wspólnym motto.

Jeszcze jedna zmiana w mojej ścieżce kariery (od systemu Windows do tworzenia aplikacji internetowych i od architektury oprogramowania do uczenia maszynowego) okazała się możliwa dzięki dwóm kolejnym osobom, które stale pobudzają moją kreatywność. Giorgio Garcia-Agreda z Crionet spełnił moje marzenia fanatyka tenisowego, posunąwszy się do odśpiewania „Easy like Sunday morning" przed tenisowymi grubymi rybami. Simone Massaro z BaxEnergy odkrył nowe fascynujące miejsce, w którym mogę dać upust swojej odnawialnej energii myśliciela, czasem nawet w obecności najwyższego kierownictwa.

Każda książka powstaje w zespole. Mamy przyjemność przedstawić nazwiska ludzi, którzy pracowali nad wydaniem tego tytułu: Loretta Yates, redaktor zamawiająca; Charvi Arora, redaktor prowadząca i Tonya Simpson, redaktor projektu.

– Dino

Szkołę średnią ukończyłem rok wcześniej i chciałem tylko zdobyć fundusze na praktykę w roli zawodowego inwestora. Jednak miałem niewłaściwych rodziców i to podejście nie zadziałało. Zapytałem zatem ojca skąd wziąć pieniądze. „To twój problem", odpowiedział. „Mogę cię tylko nauczyć tego, co sam wiem". Nauczył mnie więc, jak solidnie wykonywać swoje zadania, ale zapomniał nauczyć, jak wykonywać je źle. W efekcie w swoich programach popełniam stale te same błędy. Na tym etapie, dysponując pewną kwotą, opierałem się pomysłowi studiowania, gdy pewnego gorącego letniego popołudnia, mój ojciec powiedział: „Bądź ze sobą szczery. Jeśli nie chcesz już ćwiczyć swojego mózgu, zrezygnuj ze studiów". Skutek był taki, że miesiąc później wróciłem na uczelnię z zupełnie innym nastawieniem. Uwielbiam matematykę i nie wyobrażam sobie życia bez niej.

Wtedy spotkałem Gianfranco – przyjaciela, partnera biznesowego, ojca i dziadka. Jest on prawdziwym zawodowym inwestorem i również on nauczył mnie, jak robić coś dobrze, ale zapomniał nauczyć, jak robić coś źle. W efekcie nadal popełniam te same błędy w kwestiach finansowych.

W szkole, w pracy, na giełdzie uczę się i próbuję różnych rzeczy, które czasem się sprawdzają, a czasem nie. Zawsze gdy coś się nie uda, mam okazję się czegoś nauczyć. Jest to zasada kija i marchewki: sedno uczenia się u ludzi i maszyn. Ta książka powstała dzięki mojej obsesji na punkcie matematycznego rygoru oraz obsesji mojego ojca na punkcie klarowności. Podczas pisania staliśmy nad sobą ze swoimi kijami, aby się upewnić, że podczas czytania będziemy mieć marchewki w zasięgu ręki.

Ta książka jest dla Ciebie Mamo, ponieważ kochasz mnie niezależnie od tego, czy osiągam sukcesy, odnoszę porażki, czy napotykam na inne przeszkody. Jest również dla Ciebie Maicol, ponieważ kochałabyś mnie jeszcze bardziej, gdybym przestał hałasować w niedzielne poranki.

Ta książka jest też dla Ciebie Alessandro, ponieważ przypominasz mi, kiedy należy przestać i dla Ciebie Antonino, ponieważ przypominasz mi czasy, gdy byłem zbyt przemądrzały, aby być miłym. Jest też dla Ciebie Saro, ponieważ zawsze mam gdzie spędzić Boże Narodzenie. Także dla Ciebie, Giorgio, ponieważ przed tobą zawsze będę młokosem. Książka ta jest również dla Was, Babciu Concetto i Dziadku Salvatore, za kiełbaski oraz dla Ciebie Babciu Ledo, za energię, którą dorównujesz młodzieży.

Ta książka jest też dla Ciebie Tiago, ponieważ spotkaliśmy się do tej pory tylko jeden raz, ale na tym spotkaniu dowiedziałem się, jak wiele mogę się jeszcze od Ciebie nauczyć.

Ta książka jest też dla wszystkich tych, których nie mogłem tu wymienić, włącznie z moimi obecnymi i przyszłymi ukochanymi. Relacje z nimi zasługują na osobną książkę! Jest również dla mnie, abym mógł zrozumieć kim chcę zostać.

– Francesco

O autorach

DINO ESPOSITO

Gdy spoglądam w przeszłość, mogę naliczyć ponad 20 książek i ponad 1000 artykułów, jakie napisałem w ciągu 25-letniej kariery. Przez 22 lata, co miesiąc pisałem felieton „Cutting Edge" magazynu *MSDN Magazine*. Panuje powszechne przekonanie, że te książki i artykuły pomogły w rozwoju zawodowym tysiącom programistów .NET i ASP.NET oraz architektom oprogramowania na całym świecie.

Gdy udało mi się uciec z koszmarnego projektu w COBOL-u, w 1992 roku zacząłem pracę jako programista C i od tamtej pory byłem świadkiem rozwoju technologii MFC i ATL, COM i DCOM, początków .NET, wzlotu i upadku Silverlight, a także burzliwych losów różnych wzorców architektonicznych. W 1995 roku kierowałem zespołem pięciu marzycieli wdrażających rozwiązania, które dziś moglibyśmy porównać z Google Photos i Shuttershock – były to aplikacje biurkowe, służące do przetwarzania zdjęć przechowywanych w wirtualnym miejscu, którego nikt nie nazywał jeszcze chmurą. Począwszy od 2003 roku piszę książki dla wydawnictwa Microsoft Press o technologii ASP.NET. Napisałem też bestsellerową książkę *Microsoft .NET: Architecting Applications for the Enterprise*. Utworzyłem kilka cieszących się powodzeniem kursów w platformie Pluralsight, dotyczących architektury .NET, ASP.NET MVC UI i ostatnio ML.NET. Jako architekt większości działających za kulisami aplikacji, które umożliwiają przeprowadzanie turniejów tenisa zawodowego, przez ostatnie dwa lata koncentruję się na odnawialnej energii, Internecie rzeczy i sztucznej inteligencji, pracując na stanowisku stratega do spraw cyfrowych w BaxEnergy.

Można się ze mną skontaktować poprzez stronę https://youbiquitous.net lub twitter.com/despos, a także nawiązać kontakt poprzez serwis LinkedIn.

FRANCESCO ESPOSITO

Gdy pojawił się Windows Phone miałem około 12 lat i zdecydowanie pragnąłem dostać w swoje ręce jedno z tych urządzeń. Mogłem poprosić Tatę lub Mamę, aby mi je kupili, ale nie wiedziałem jak na to zareagują. Jako zwykły nastolatek miałem zerowe szanse, że ktoś mi je kupi. Wymyśliłem więc, że dość dobrze znam się na językach programowania i postarałem się zrobić tak dobre wrażenie na pewnych pracownikach firmy Microsoft, że otrzymałem urządzenie do testowania.

Windows Phone był tylko początkiem; kolejna była moja szalona pasja do systemu iOS, a od niedawna do języka C#.

Obecny etap mojego życia zaczął się wraz z ukończeniem szkoły średniej, rok wcześniej niż się tego spodziewałem. Tak przy okazji, wyczynem takim może się pochwalić tylko 0,006 procent uczniów we Włoszech. Czułem się jak młody bóg i zapisałem na studia matematyczne. Oblałem pierwszy egzamin, co wstrząsnęło mną tak bardzo, że za karę zacząłem dniami i nocami programować w ASP.NET. Założyłem małą firmę programistyczną Youbiquitous i zacząłem zarabiać własne pieniądze. W 2017 roku moja miłość do matematyki dała o sobie znać i sprawiła, że wróciłem na studia i zainteresowałem się inwestycjami finansowymi oraz uczeniem maszynowym.

Ta książka jest więc naturalną konsekwencją końca mojego dzieciństwa. Chciałem oddać coś swojemu ojcu i ułatwić mu zrozumienie matematycznych podstaw sieci neuronowych i algorytmów. Tak przy okazji wspomnę, że marzę o rozwinięciu superteorii inteligencji, która w matematyczny sposób wyjaśniałaby, dlaczego współczesna sztuczna inteligencja działa i jak się może rozwinąć.

Można się ze mną skontaktować za pośrednictwem strony *https://youbiquitous.net*.

Wstęp

Potrzebujemy ludzi, którzy potrafią śnić o rzeczach,
których nigdy nie było i pytać, dlaczego nie.

– John F. Kennedy
przemówienie w parlamencie irlandzkim, czerwiec 1963

Mamy dziś dwa spojrzenia na sztuczną inteligencję, które się wzajemnie nie wykluczają. Pierwsze spojrzenie jest dyktowane przez większość przedstawicieli mediów; drugie przez społeczność IT. W obydwu obozach znajdziemy prawdziwych ekspertów i prawdziwych mędrców.

Spojrzenie lansowane przez media koncentruje się na wpływie, jaki sztuczna inteligencja, w znanej i jeszcze nieznanej postaci, może mieć na nasze życie w jakiejś niezbadanej przyszłości. Spojrzenie dominujące w społeczności IT (do której należą programiści i analitycy danych) prezentuje uczenie maszynowe jako podstawę nowej generacji usług programistycznych, które są po prostu inteligentniejsze niż współczesne.

Pomiędzy masami ludzi, do których docierają media, a znacznie mniejszą społecznością IT znajduje się przestrzeń kontrolowana przez chmurowych gigantów. To oni prowadzą badania i codziennie przyczyniają się do rozwoju wiedzy, publikując nowe usługi dla każdego, kto jest zainteresowany dodaniem inteligentnej warstwy do nowych i istniejących aplikacji.

U podstaw piramidy sztucznej inteligencji zasiadają menedżerowie i zarząd. Z jednej strony są oni skłonni do wykorzystania w przedsięwzięciach biznesowych tych oszałamiających usług, o których dowiedzieli się z doniesień technologicznych, aby uzyskać przewagę nad konkurencją. Z drugiej strony stoją w obliczu astronomicznych kosztów projektów, które zapoczątkowali w najlepszej wierze.

- Sztuczna inteligencja nie jest magiczną różdżką.
- Sztuczna inteligencja nie jest usługą, za którą można płacić na podstawie użycia. Co gorsza, nie można jej zaliczyć ani do kapitału, ani do wydatków operacyjnych.
- Sztuczna inteligencja jest po prostu oprogramowaniem.

Każda decyzja biznesowa dotycząca sztucznej inteligencji będzie lepsza, jeśli zostanie podjęta z punktu widzenia oprogramowania. Należy przy tym ustalić wymagania, znaleźć solidnego partnera, ustalić budżet, pracować, zacząć ponownie, zgodnie z metodologią zwinną.

Czy rzeczywiście jest to takie proste?

Chociaż sztuczna inteligencja jest związana z tworzeniem oprogramowania, proces ten różni się od tworzenia sklepu internetowego czy platformy do rezerwacji.

- Nie zabierajmy się za projekty sztucznej inteligencji, jeśli nie mamy wyraźnej wizji problemu, który trzeba rozwiązać, jego kontekstu i wynikających z nich wniosków.
- Nie zabierajmy się za ambitne i śmiałe projekty, w których będziemy tylko podążać za przykładem najbliższego konkurenta.
- Nie zabierajmy się za takie projekty, jeśli nie jesteśmy przygotowani na stratę sporej ilości pieniędzy.

Rozwiązujmy na raz tylko jeden problem, zbudujmy zespół specjalistów z różnych dziedzin i zadbajmy o pełny dostęp do danych.

Kto powinien przeczytać tę książkę?

Przygotowując się do napisania tej książki, otrzymaliśmy mnóstwo informacji zwrotnych o strukturze i kilkakrotnie się do nich zastosowaliśmy. Co najmniej trzykrotnie znacząco przebudowaliśmy spis treści. Trudność polegała na tym, że chcieliśmy napisać unikatową i innowacyjną książkę. Chcieliśmy przedstawić ideę uczenia maszynowego i tworzenia oprogramowania w nieco innym świetle, niż w jakim jest postrzegana obecnie. Mamy nadzieję, że nasza wizja uczenia maszynowego ziści się w niedalekiej przyszłości!

Zauważyliśmy, że uczenie maszynowe wciśnięto w sztywne ramy analizy danych. Traktuje się je jako artefakt przekazywany programistom, którzy mają go wbudować w usługę internetową lub aplikację biurkową. Jest to podejście wodospadu – ni mniej, ni więcej. Gdzie się podziała cała zwinność, którą tak szczycą się firmy i przedsiębiorstwa? Zwinne uczenie maszynowe polega na współpracy analityków danych i programistów, przy wsparciu ze strony analityków biznesowych i ekspertów z danej dziedziny. Do zespołu mogą też dołączyć interesariusze – pracownicy z działu IT, DevOps lub innego – którzy mogą ułatwić dostęp do danych i ich przetwarzania. To jest zwinna praca zespołowa – ni mniej, ni więcej.

Zauważyliśmy potrzebę (biznesową) łączenia umiejętności – analitycy danych powinni znać zasady programowania, a programiści poznać tajniki analizy danych. Ta przeznaczona dla początkujących książka sprawdzi się w obydwu przypadkach. Przemówi do programistów pokazując ML.NET w działaniu (z pominięciem i z wykorzystaniem Pythona) i analizując mechanizmy algorytmów uczenia maszynowego. Przemówi też do analityków danych, którzy muszą lepiej poznać aspekty programistyczne.

Ta książka jest doskonałym podręcznikiem dla programistów, chcącym dodać analizę danych i uczenie maszynowe do kolekcji swoich umiejętności. Świetnie sprawdzi się też w przypadku analityków danych, chcących poznać lepiej tajniki oprogramowania. Jednak przedstawiciele obydwu tych zawodów muszą poznać więcej aspektów wzajemnej pracy.

Takie jest założenie tej książki. Zakwalifikowaliśmy ją do kategorii „dla początkujących", ponieważ obejmuje dość szeroki zakres zagadnień, ale bez zagłębiania się w nie. Zawiera przykłady napisane w .NET, ponieważ uważamy, że chociaż ekosystem Pythona jest bogaty i cieszy się powodzeniem, nie ma powodu, aby nie rozejrzeć się za platformami, umożliwiającymi zbliżenie uczenia maszynowego do technik tworzenia aplikacji, usług i mikrousług programistycznych. To przecież one stanową ostateczne miejsce działania każdego potoku uczenia (włącznie z TensorFlow, PyTorch, czy dostosowanym do potrzeb kodem Pythona).

Kto nie powinien czytać tej książki?

Ta książka dla początkujących została opracowana z myślą o zapewnieniu szerokiego, klarownego i dokładnego omówienia uczenia maszynowego, z uwzględnieniem przykładów wykorzystujących platformę ML.NET. Jeśli ktoś szuka mnóstwa przykładów w języku Python, nie znajdzie ich w tej książce. Jeśli ktoś szuka przykładów praktycznych, które można skopiować i wkleić do swojego kodu w Pythonie lub w ML.NET, ta książka na pewno się nie sprawdzi. Jeśli ktoś szuka wyjaśnienia matematycznych zawiłości związanych z algorytmami lub objaśnień implementacji algorytmów, ta książka może nie spełnić tych oczekiwań. (Uwzględniliśmy pewne aspekty matematyczne, ale bardzo powierzchownie).

Organizacja tej książki

Tę książkę podzieliliśmy na pięć części. Część I, „Podstawy uczenia maszynowego" zawiera krótkie omówienie podstaw sztucznej inteligencji, inteligentnego oprogramowania oraz podstawowe kroki każdego projektu uczenia maszynowego w kompleksowych rozwiązaniach. Część II, „Uczenie maszynowe w .NET" skupia się na bibliotece ML.NET i omawia jej główne aspekty, takie jak zadania przetwarzania danych, trenowanie i ocenę w kontekście typowych problemów, takich jak regresja i klasyfikacja. Część III, „Podstawy uczenia płytkiego", zawiera matematyczne szczegóły dotyczące rodzin algorytmów, powszechnie wykorzystywanych do rozwiązywania rzeczywistych problemów. Uwzględniliśmy algorytmy regresyjne, drzewa decyzyjne, metody grupowe, klasyfikatory bayesowskie, maszyny wektorów nośnych, algorytm k-średnich, algorytmy gradientowe. Część IV, „Podstawy uczenia głębokiego", jest poświęcona

sieciom neuronowym, które sprawdzą się, gdy wszystkie poprzednie algorytmy zawiodą. Ostatnia część V, „Finalne uwagi" dotyczy ogólnej biznesowej wizji sztucznej inteligencji, ze szczególnym uwzględnieniem uczenia maszynowego. Zawiera skrótowe omówienie usług przetwarzania danych i obliczeniowych dostępnych w chmurze, szczególnie w platformie Azure.

Przykładowy kod

Wszystkie przykłady kodu zawarte w tej książce, wraz z poprawkami i rozszerzeniami są dostępne pod adresem *MicrosoftPressStore.com/IntroMachineLearning/downloads*.

Errata i wsparcie dotyczące książki

Dołożyliśmy wszelkich starań, aby zapewnić dokładność tej książki i dołączonych do niej materiałów dodatkowych. Aktualizacje – w formie listy zgłoszonych błędów i poprawek – są dostępne pod adresem:

MicrosoftPressStore.com/IntroMachineLearning/errata

Jeśli zauważysz błąd, którego nie ma na tej liście, zgłoś go korzystając z tej samej strony.

Dodatkowe informacje i wsparcie są dostępne pod adresem

http://www.MicrosoftPressStore.com/Support.

Należy zauważyć, że wsparcie dla oprogramowania i sprzętu oferowanego przez firmę Microsoft nie jest dostępne za pośrednictwem wymienionych adresów. Pomoc dotyczącą oprogramowania lub sprzętu firmy Microsoft można uzyskać pod adresem *http://support.microsoft.com*.

CZĘŚĆ I

Podstawy uczenia maszynowego

ROZDZIAŁ 1

Jak uczą się ludzie

Komputery mogą teoretycznie emulować ludzką inteligencję i ją przewyższyć.

– Stephen Hawking, 2014

Współczesna beletrystyka obfituje w opowieści o superkomputerach zdolnych do przekształcenia dowolnych danych do formatu przyswajalnego przez ludzi. Powszechnie znanym przykładem jest HAL 9000 – komputer sterujący statkiem kosmicznym Discovery w filmie *2001: Odyseja kosmiczna* (1968). Innym sławnym przedstawicielem tej kategorii jest JARVIS (Just A Rather Very Intelligent System), czyli komputer, który, podobnie jak współczesne Alexa i Cortana, pełni rolę asystenta domowego Tony'ego Starka w komiksach Marvel Comics i nakręconych na ich podstawie filmach. Jeszcze jednym przykładem, chociaż znanym mniej szerokiej publiczności niż HAL 9000 i JARVIS, jest Max – superkomputer obsługiwany przez Hirama Yeagera w opowieściach dotyczących organizacji NUMA (1983) autorstwa Clive'a Cusslera.

Wszystkie te maszyny są przykładami niesłychanie zaawansowanych systemów komputerowych, zdolnych do przetwarzania każdego rodzaju informacji, niezależnie od sposobu ich przechowywania, formatu i struktury. Zwykle bohaterowie takich książek i filmów muszą jedynie „wczytać dane do komputera", niezależnie od tego, czy są to dokumenty papierowe, pliki cyfrowe lub nośniki multimedialne. Komputery autonomicznie rozpoznają ich zawartość, uczą się na ich podstawie i zwracają ludziom wyniki w języku naturalnym.

W czasach, gdy autorzy wspomnianych dzieł wymyślali superkomputery, koncepcje te należały do świata science-fiction. Obecnie jednak niektóre implementacje systemów oprogramowania umożliwiają uzyskanie zbliżonych efektów, chociaż wymagają pewnej pomocy ze strony ludzi. Systemy komputerowe mogą sterować samolotami lub przeprowadzać wyrafinowane operacje chirurgiczne. Jednak już za kilka lat prawdopodobnie komputery będą zupełnie autonomicznie sterować samochodami bez pomocy kierowców. To będzie prawdziwa zabawa!

Ta książka jest poświęcona algorytmom, magazynowaniu, procesom i platformom oprogramowania (.NET), za pomocą których programiści mogą po prostu wczytać dane do komputerów i uzyskać od nich użyteczne informacje zwrotne, a być może również

rozwiązania. W tej książce nie znajdziemy teorii o futurystycznej formie sztucznej inteligencji. Nie poruszam także technologicznych aspektów pewnych wstępnie opracowanych usług kognitywistycznych.

Ta książka jest o tym, jak wykorzystać niektóre dostępne już narzędzia programistyczne do bardziej inteligentnego przetwarzania danych.

Ku myślącym maszynom

Ludzie zawsze marzyli o innych sztucznych bytach, zdolnych do rozumowania i myślenia na wyższym niż nasz poziomie. Literatura jest pełna takich fantastycznych bohaterów, którzy pojawili się już w starożytnych greckich tragediach Ajschylosa i Eurypidesa. W dziełach tych urządzenia (Rzymianie nazwali je później mianem *deus ex machina*) wykorzystywano czasem do symulowania boskiej interwencji. Były to byty zdolne do rozwiązywania konfliktów w dramacie, którego zawiłości były zbyt trudne dla ludzi.

Pragnienie zbudowania maszyny rozwiązującej nasze problemy jest zatem cechą ludzkiej natury. Na falach tego pragnienia wielu filozofów podejmowało próby sformułowania teoretycznych podstaw mechanizmów ludzkiego myślenia.

Świt rozumowania mechanicznego

Pierwsze wzmianki znajdziemy w dziełach Euklidesa (trzeci wiek przed naszą erą), który logicznie wyjaśnił wszystkie teorie geometryczne na podstawie pięciu podstawowych aksjomatów. Geometria euklidesowa jest doskonałym przykładem rozumowania formalnego. Później, w siedemnastym wieku Gottfried Leibniz i inni wybitni myśliciele doszli do wniosku, że ludzkie myślenie można usystematyzować poprzez zbiór reguł algebraicznych i założyli istnienie ogólnego języka, który mógłby sprowadzić ludzkie rozumowanie do zwykłych obliczeń maszyny mechanicznej.

Leibniz w swojej pracy inspirował się w dużym stopniu dokonaniami innego wybitnego uczonego Raymonda Lulla, który zapoczątkował badania nad współczesną sztuczną inteligencją już w trzynastym wieku i stał się inspiracją dla dalszego rozwoju logiki matematycznej przez George'a Boole'a, a na początku dwudziestego wieku przez Davida Hilberta i Bertranda Russella.

Na uwagę zasługują prace Hilberta, który w 1900 roku zdefiniował kilka problemów matematycznych, które miały na celu znalezienie odpowiedzi na pytanie „Czy wszystkie twierdzenia matematyczne można wyrazić i przetwarzać za pośrednictwem zbioru dobrze zdefiniowanych reguł?". Hilbert chciał znaleźć sposób na sformalizowanie wnioskowania matematycznego w podobny sposób, jak Euklides w swoich czasach. Celem Hilberta było znalezienie zbioru aksjomatów, na podstawie których mógłby ustalić wszystkie twierdzenia matematyczne.

Twierdzenia Gödla o niezupełności

W 1931 roku Kurt Gödel przedstawił kilka twierdzeń z logiki matematycznej, które zostały zinterpretowane jako negatywna odpowiedź na fundamentalne pytanie Hilberta. Szczególnie dwa z nich stały się podstawą poniższego stwierdzenia:

> *W dowolnym systemie formalnym, który umożliwia modelowanie arytmetyki liczb naturalnych, istnieje co najmniej jedno nierozstrzygalne stwierdzenie, potwierdzone doświadczalnie, którego nie da się udowodnić lub mu zaprzeczyć korzystając z aksjomatów systemu.*

Ponadto, Gödel udowodnił, że nawet jeśli ktoś aksjomatycznie przypisze wartość prawda lub fałsz do nierozstrzygalnego stwierdzenia, wszelkie dalsze rozumowanie będzie ostatecznie prowadzić do następnego nierozstrzygalnego stwierdzenia.

Dlaczego twierdzenia Gödla są kluczowe w rozumowaniu formalnym, a co za tym idzie, w sztucznej inteligencji?

Po pierwsze, twierdzenie Gödla o niezupełności wyznacza granicę, której logika matematyczna nie może przekroczyć: istnieją rzeczy, których nikt nie zdoła udowodnić, stosując rozumowanie formalne. Jednak z drugiej strony twierdzenia Gödla pokazują, że w granicach spójnego systemu formalnego każde rozumowanie można zawsze wyrazić za pomocą zbioru formalnych reguł transformacji, a następnie w pewien sposób zmechanizować.

Ten drugi aspekt jest niesłychanie istotny w sztucznej inteligencji, ponieważ ustanawia teoretyczne podstawy mechanicznego, komputerowego rozumowania.

Formalizacja maszyn liczących

Chociaż prace Hilberta i Gödla są czysto teoretyczne, dały początek trzem równoległym i niezależnym ścieżkom badawczym, które w ciągu kilku lat doprowadziły do wysnucia jednakowych wniosków. Miało to miejsce około połowy lat 30-tych XX wieku.

- W 1933 roku Gödel sformułował koncepcję *ogólnych funkcji rekurencyjnych*. Ta logiczna funkcja obliczeniowa przyjmuje skończoną tablicę liczb naturalnych i zwraca liczbę naturalną.
- Później, w roku 1936, Alonzo Church zdefiniował system formalny nazwany *rachunkiem lambda*, który wyrażał podobne obliczenia na liczbach naturalnych.
- Niemal w tym samym czasie, całkowicie niezależnie Alan Turing opracował model teoretyczny maszyny obliczeniowej (*maszyny Turinga*), służącej do wykonywania obliczeń na symbolach zapisanych na skończonej taśmie.

Te trzy klasy funkcji obliczeniowych ujednoliciła teza Churcha-Turinga, dowodząc, że funkcja jest obliczalna w rachunku lambda wtedy i tylko wtedy, gdy jest rozwiązywalna przez maszynę Turinga, wtedy i tylko wtedy, gdy jest ogólną funkcją rekurencyjną. Z tezy Churcha-Turinga wynika zatem, że możliwe jest zbudowanie urządzenia mechanicznego zdolnego do odtworzenia każdego akceptowalnego procesu wnioskowania matematycznego poprzez manipulację symbolami.

Począwszy od końca lat 30-tych XX wieku, powyższa teza była punktem wyjścia do dalszych rozważań o możliwości utworzenia rzeczywiście myślących maszyn.

Świetnie, jak jednak sformalizować ludzkie myślenie?

Formalizacja ludzkiego myślenia

Historia zna wiele wspaniałych przykładów maszyn obliczeniowych zbudowanych lub tylko opracowanych przez wszechstronnie uzdolnionych ludzi i naukowców. Jak już wspomniano, jedną z nich zaprojektował Leibnitz w siedemnastym wieku, a w dziewiętnastym wieku Charles Babbage opracował bardziej szczegółowe teoretyczne zasady działania innej.

Przodkami nowoczesnych komputerów były maszyny szyfrujące i maszyny służące do łamania szyfrów, wykorzystywane podczas II wojny światowej. Jedną z nich była Enigma, jej odpowiednik służący do łamania szyfrów o nazwie Bombe (zbudowany z dużą pomocą Alana Turinga), maszyna Lorenza wykorzystywana przez armię niemiecką oraz wielka maszyna brytyjska o nazwie Colossus, która ostatecznie złamała kod maszyny Lorenza. ENIAC jest kolejną maszyną, zbudowaną w Stanach Zjednoczonych pod koniec II wojny światowej.

Podczas budowy wspomnianych maszyn wykorzystano teoretyczne podstawy teorii Churcha-Turinga. Nad rozwojem komputera ENIAC czuwał John von Neumann, kolejny słynny naukowiec zasłużony w dziedzinie informatyki. W istocie nieprzypadkowo Alan Turing i John von Neumann są uznawani za ojców dzisiejszej *sztucznej inteligencji*.

Wyobraźmy sobie teraz, że jesteśmy jednym z tych świetnych naukowców. Są lata 50-te XX wieku i sądzimy, że możemy zbudować maszyny przetwarzające wszystko, co da się zapisać poprzez spójną gramatykę symboli, a nie tylko obliczające liczby na podstawie innych liczb. Prawdopodobnie czulibyśmy się jak bogowie; prawdopodobnie widzielibyśmy oczami wyobraźni maszynę, która w niedalekiej przyszłości będzie się zachowywać podobnie do ludzi. Prawdopodobnie zadalibyśmy sobie wtedy kluczowe pytanie: *Czy maszyny mogą myśleć*?

Alan Turing opracował test sprawdzający, czy maszyny mogą myśleć. Wyobraził sobie rozmowę człowieka z maszyną prowadzoną za pośrednictwem dalokopisu i nadzorowaną przez ludzkiego sędziego. Gdyby maszyna mogła odpowiadać na pytania i przekonać sędziego, że jest człowiekiem, wówczas należałoby stwierdzić, że maszyna myśli.

Przez lata wielu podważało skuteczność testu Turinga. Szczególnie John Searle (emerytowany profesor wydziału Philosophy of Mind and Language w Berkeley) zauważył, że każdy, kogo wyposażymy w odpowiedni słownik i instrukcje napisane w jego języku, prawdopodobnie zdoła udzielić odpowiedzi, na przykład po chińsku, które zostaną zaakceptowane przez chińskiego sędziego. Czy oznacza to, że udzielający odpowiedzi (maszyna lub człowiek) rozumie język chiński?

Uwagę Searle'go o maszynach myślących można sprowadzić do spostrzeżenia, że maszyny mogą jedynie przetwarzać symbole zgodnie z regułami, jednak nie osiągną świadomości, poznania i percepcji na ludzkim poziomie, nie wspominając o ich zdolnościach językowych. Searle uważał, że język jest czymś więcej niż zwykłą manipulacją symbolami, a wspomniane „więcej" definiuje ludzkie myślenie. Komputery mogą tylko wykonywać obliczenia, ale czynią to tak uważnie i szybko, że w niektórych zadaniach przewyższają ludzi.

Zgodnie z poglądami Searle'go, od obecnych systemów uczenia maszynowego oczekuje się działania w wysoce kontrolowanych scenariuszach, według reguł biznesowych i z wykorzystaniem wzorców danych. Stawia się przed nimi wyzwania przewidywania problemów i zdarzeń. Przykłady obejmują systemy prognozujące awarie sprzętu lub wykrywające oszustwa finansowe. Te wszystkie nowoczesne systemy mogą się doskonale sprawdzać w swoich dziedzinach, lecz ludzkie wyobrażenie o „myśleniu" wymaga (znacznie) większych mocy obliczeniowych.

Narodziny sztucznej inteligencji jako dyscypliny

Sztuczna inteligencja (ang. *Artificial Intelligence* – AI) została oficjalnie opracowana w 1956 roku, gdy John McCarthy zorganizował sześciotygodniowe warsztaty naukowe w Dartmouth College w New Hampshire. Zaprosił kilkunastu kolegów z różnych dziedzin naukowych, zajmujących się matematyką, inżynierią, neurologią i psychologią.

Celem warsztatów było przeprowadzenie burzy mózgów dotyczącej idei maszyn myślących. McCarthy ukuł nową nazwę *sztuczna inteligencja*, ze względu na jej neutralność, a także aby odseparować i zunifikować dwa nurty badań akademickich, które postrzegał jako jedność.

W rzeczywistości w tamtych czasach abstrakcyjny temat maszyn myślących był rozważany w dwóch aspektach badawczych – teorii automatów, bezpośrednio wywodzącej się z pracy Churcha i Turinga, oraz cybernetyki, pochodzącej bezpośrednio od teorii Babbage'a, która zyskała fizyczną reprezentację w postaci urządzeń opracowanych przez von Neumanna.

Ostatecznym celem warsztatów było ustanowienie podstaw do opracowania sztucznego mózgu. Jest to również ostateczny cel nowoczesnej sztucznej inteligencji.

Ważne W 1943 roku McCulloch i Pitts zaproponowali model obliczeniowy zainspirowany znaną strukturą neuronu mózgowego. Ich przełomowa praca stanowi podstawy nowoczesnych sieci neuronowych. Istotne są dwa fakty. Po pierwsze, w czasie gdy McCulloch i Pitts zaproponowali swój model, nie istniały fizyczne komputery. Architektura von Neumanna, na podstawie której opierają się wszystkie nowoczesne komputery, nadal znajdowała się na wczesnym etapie rozwoju. Innymi słowy, fascynujące jest, że mogliśmy mieć komputery neuronowe już od początku istnienia informatyki. Po drugie, po pomyślnym wprowadzeniu pierwszych fizycznych komputerów w latach 60-tych XX wieku, badania nad sieciami neuronowymi niemal całkowicie wygasły i zostały wznowione pod koniec lat 80-tych XX wieku. W ciągu dekady nastąpiło kolejne spowolnienie, przyćmione przez narodziny Internetu. Kilka lat temu nastąpiło odrodzenie, głównie dzięki chmurze obliczeniowej i innym czynnikom, takim jak społeczna potrzeba utrzymywania ciągłego kontaktu.

Biologia uczenia się

Po pewnych modyfikacjach słownictwa znalezionego w różnych słownikach, szeroko rozpowszechniona definicja *inteligencji* wygląda następująco:

Zdolność do zdobywania wiedzy i przekształcania jej w umiejętność.

Między słowami tej definicji można się dopatrzyć innej warstwy znaczeniowej, którą warto wydobyć na powierzchnię. Inteligencja polega też na zdolności do:

- Formowania osądów i opinii na podstawie zdobytej wiedzy;
- Działania na tej podstawie;
- Reagowania na nieznane zdarzenia.

W skrócie inteligencja łączy w sobie możliwości poznawcze, włącznie z percepcją, pamięcią, językiem i rozumowaniem, a także wykorzystuje określone podejście językowe do wyodrębniania, przekształcania i przechowywania informacji.

Czym właściwie jest inteligentne oprogramowanie?

W tym podrozdziale omawiamy biologię ludzkiego myślenia. Jednak naszym ostatecznym celem jest eksploracja zbioru technik, umożliwiających takie uczenie maszyn, aby oprogramowanie wykonywało określone zadania w coraz bardziej inteligentny sposób.

ROZDZIAŁ 6

Sposób .NET

Sztuczna inteligencja byłaby ostateczną wersją Google. Dokładnie rozumiałaby nasze potrzeby i je zaspokajała. Jest to jednak pieśń dalekiej przyszłości.

– Larry Page
współzałożyciel Google

W pierwszej części książki przeanalizowaliśmy ogólne kroki, jakie należy podjąć w celu zbudowania rozwiązania uczenia maszynowego. Każde rozwiązanie obejmuje gotowy do wdrożenia model, powstały w wyniku treningu algorytmu na wystarczająco dużym zbiorze danych. Proces wyboru i dostosowania zbioru danych jest zdecydowanie najbardziej czasochłonnym etapem, który ma zarazem największy wpływ na ostateczne rozwiązanie. Szacuje się, że przygotowanie danych w rzeczywistym scenariuszu może pochłonąć do 80 procent budżetu całego projektu. W istocie przygotowanie danych wiąże się z faktycznymi problemami, wizją biznesową, a także dostępnością danych. Zauważmy, że w pewnych przypadkach firmy chcące skorzystać z uczenia maszynowego nie dysponują nawet infrastrukturą potrzebną do zebrania danych potrzebnych w projekcie. Jednak gdy tylko dane są dostępne, czy to w surowym formacie, takim jak pliki CSV, czy w postaci relacyjnej bazy danych, potok uczenia maszynowego może wymagać kroku inżynierii cech, w którym manipuluje się zbiorem danych, aby skuteczniej osiągnąć określony cel uczenia maszynowego.

Następnie wybiera się algorytm, po czym następuje faza treningu.

Faza treningu jest czasochłonna, ale zajmuje kilka godzin, najwyżej dni – a zatem jest krótka w porównaniu z czasem, jaki należy poświęcić na zbieranie danych, jeśli nie są jeszcze dostępne. Ogólnie rzecz biorąc, trening polega na znalezieniu zbioru wartości, które umożliwią skuteczne działanie wybranego algorytmu w rozważanym scenariuszu. Wartości określa się na podstawie zawartości treningowego zbioru danych. Następnie, jeśli wykryte wartości umożliwiają uzyskanie akceptowalnych wyników, praca jest prawie skończona. Model jest zapisany w pliku .zip i można się zastanowić nad najbardziej odpowiednim sposobem udostępnienia go w aplikacjach klienckich. Jednak uzyskiwanie akceptowalnych wyników jest procesem cyklicznym. Jeśli otrzymamy nowe dane lub takie, które nie pasują dokładnie do danych z rzeczywistego

świata, musimy ponownie wytrenować model. Ostatecznie cały cykl tworzenia rozwiązania uczenia maszynowego przypomina cykl DevOps.

Jednak ostatecznie klienci potrzebują gotowego rozwiązania, z zaimplementowanymi możliwościami inteligentnego przewidywania i klasyfikacji, do czego obecnie najlepiej nadaje się uczenie maszynowe. Oznacza to nie tylko budowanie i trenowanie modelu, ale także udostępnianie go w postaci usługi wchodzącej w skład procesu lub działającej osobno. W kontekście architektury aplikacji rozwiązanie uczenia maszynowego jest po prostu usługą uruchamianą poprzez logikę biznesową w celu uzyskania wyników dla warstwy prezentacji i infrastruktury.

Większość obecnych rozwiązań uczenia maszynowego utworzono w ekosystemie Pythona. Jednak wynika to z wygody, a nie ze względu na przewagę technologiczną tego środowiska. W następnym rozdziale poznamy platformę ML.NET, umożliwiającą tworzenie projektów uczenia maszynowego na sposób .NET.

Dlaczego (nie) Python?

Powiedzmy to od razu: nie ma biznesowych powodów, które uniemożliwiałyby implementację uczenia maszynowego w platformie .NET. Ponadto, w platformie .NET nic nie blokuje możliwości przetwarzania zbiorów danych i pisania algorytmów treningowych rozwiązujących problemy biznesowe.

Jednak gdy przychodzi do implementacji uczenia maszynowego, na czoło wysuwa się Python wraz ze swoim imponującym ekosystemem specjalnych narzędzi i bibliotek.

Dlaczego Python jest tak popularny w uczeniu maszynowym?

Python, R, i C++ dominują w uczeniu maszynowym, jednak trudno znaleźć rzeczywistą przyczynę takiego stanu rzeczy. Jednak niezależnie od powodów rozpowszechnienia platform Python, R i C/C++ nie ma technicznych przeciwskazań, uniemożliwiających użycie platformy .NET i związanych z nią języków (głównie C# i F#) do trenowania modeli uczenia maszynowego.

Rzut oka na Pythona

Python jest interpretowanym i zorientowanym obiektowo językiem programowania, utworzonym przez Guido van Rossuma pod koniec lat 80-tych XX wieku w Centrum Wiskunde & Informatica – CWI (centrum matematyki i informatyki) w Amsterdamie.

Opublikowany po raz pierwszy w 1991 roku język ten przeszedł wiele ulepszeń, nie tracąc przy tym orientacji na minimalizację składni i czytelność. Wizja Pythona

jako języka programowania to niewielki silnik dla głównych aspektów języka i wielka biblioteka standardowa oraz łatwo rozszerzalny interpreter.

Mówiąc o Pythonie warto podkreślić, że bycie językiem interpretowanym niekoniecznie jest oznaką kiepskiej wydajności. Jednak gdy wydajność ma szczególne znaczenie, kod Pythona musi zostać przetłumaczony na kod C, skompilowany na poczekaniu lub rozszerzony o moduły napisane w języku C. Jednak pisanie programów w języku C jest znacznie bardziej niskopoziomowe i kosztowne niż programowanie w języku .NET lub Pythonie.

Uwaga Python nie jest w pełni wielowątkowy, tak jak wszystkie języki platformy .NET. Nawet jeśli teoretycznie można korzystać z wątków, to Global Interpreter Lock (GIL) gwarantuje, że w danym czasie kod Pythona może działać tylko w jednym wątku. GIL zapobiega użyciu więcej niż jednego rdzenia CPU (lub osobnych jednostek CPU) w celu uruchamiania wielu równoległych wątków. Dotyczy to tylko czystego kodu Pythona, ponieważ rozszerzenia w języku C zwykle wyłączają GIL, umożliwiając działanie kodu w wielu wątkach.

Doskonałe narzędzie dla naukowców

Zważywszy na powszechne wykorzystanie Pythona w uczeniu maszynowym, ośmielamy się stwierdzić, że język ten zawdzięcza swój sukces głównie *interpretowanej* naturze (brak konieczności kompilacji) i *interaktywności* (można wpisywać instrukcje bezpośrednio w wierszu poleceń), ze składnią, która jest bardziej opisowa, niż lepiej strukturyzowane i dobrze uformowane składnie języków Java i C#, nie wspominając o trudniejszych językach, takich jak C i C++.

Zrodzony w środowisku naukowym, Python stał się de facto standardowym językiem programowania wykorzystywanym przez naukowców do praktykowania, eksploracji i eksperymentowania z liczbami. W pewnym sensie zajął miejsce, które w latach 60-tych i 70-tych XX wieku zajmował Fortran.

Początkowo korzystanie z języka Python w nowej, gorącej dziedzinie naukowej, takiej jak uczenie maszynowe, było naturalnym wyborem i z czasem – zważywszy na naturalną rozszerzalność języka – doprowadziło do powstania ogromnego ekosystemu specjalnych bibliotek i narzędzi. To z kolei wzmocniło przekonanie, że Python jest najlepszą opcją do tworzenia modeli obliczeniowych.

Obecnie, większość analityków danych uważa, że Python jest wygodny w użyciu w projektach uczenia maszynowego, co prawdopodobnie wynika z połączenia prostoty języka, dostępnych narzędzi i wielu przykładów.

Jednak zapytajmy ponownie, dlaczego korzystać z Pythona, a nie z Javy lub bardziej nowoczesnego języka, takiego jak C#?

Złożony jest lepszy niż skomplikowany

Dlaczego? Ponieważ analitycy danych rzadko są programistami, chociaż zwykle umieją skutecznie korzystać z języków programowania. Analitycy danych zwykle postrzegają bardziej formalnie eleganckie i zwięzłe języki, takie jak C# i Java za niepotrzebnie skomplikowane, nie wspominając o obciążeniu wynikającym z dodatkowych kroków, takich jak kompilowanie, budowanie i wdrażanie.

Zen Pythona – zbiór przejrzystych zasad podsumowujących główną filozofię języka – opiera się na poniższych punktach:

- Prosty jest lepszy niż złożony.
- Złożony jest lepszy niż skomplikowany.
- Płaski jest lepszy niż zagnieżdżony.
- Czytelność ma znaczenie.
- Jeśli łatwo wyjaśnić implementację, może być ona dobrym pomysłem.

Pełny zestaw zasad jest dostępny pod adresem www.python.org/dev/peps/pep-0020.

Mając to jednak na uwadze, z punktu widzenia czystej funkcjonalności nie ma powodu pomijać języka C# i platformy .NET jako godnej uwagi alternatywy do budowania i trenowania modeli uczenia maszynowego.

Gdyby tylko udało się osiągnąć w C# i .NET ten sam poziom doskonałości na polu uczenia maszynowego, jaki jest obecnie możliwy w Pythonie! Ostatecznie jest to tylko kwestia narzędzi i ekosystemu.

Taksonomia bibliotek Pythona wykorzystywanych w uczeniu maszynowym

Ekosystem narzędzi i bibliotek dostępnych w Pythonie można podzielić na pięć głównych obszarów: manipulacja danymi, wizualizacja danych, obliczenia numeryczne, trenowanie modelu i sieci neuronowe. Prawdopodobnie nie jest to wyczerpująca lista, ponieważ wiele innych istniejących bibliotek wykonuje bardziej specyficzne zadania i koncentruje się na pewnych specyficznych obszarach uczenia maszynowego, takich jak przetwarzanie języka naturalnego i rozpoznawanie obrazów.

W języku Python kroki służące do budowania potoku uczenia maszynowego są zwykle wykonywane w obrębie *notatnika*. Notatnik jest dokumentem tworzonym w specyficznym środowisku interaktywnym, sieciowym lub lokalnym, nazywanym Jupyter Notebook (patrz *https://jupyter.org*). Każdy notatnik zawiera kombinację wykonywalnego kodu Pythona, sformatowanego tekstu, tabel danych, a także wykresy i obrazy, poprzez które budujemy i udostępniamy historię tworzenia projektu.

W notatniku wykonuje się takie zadania, jak manipulacja danymi, tworzenie wykresów i trenowanie. Mamy przy tym do dyspozycji wiele predefiniowanych i przetestowanych w praktyce bibliotek.

Manipulowanie i analiza danych

Pandas (*https://pandas.pydata.org*) jest biblioteką skoncentrowaną wokół obiektu *DataFrame*, za pomocą którego programiści mogą wczytywać i manipulować danymi tabelarycznymi znajdującymi się w pamięci. Do obiektu tego można zaimportować dane z plików CSV oraz z baz danych SQL; dostępne są podstawowe metody przetwarzania danych, takie jak wyszukiwanie warunkowe, filtrowanie, indeksowanie i sortowanie, krojenie i grupowanie, a także operacje na kolumnach, takie jak dodawanie, usuwanie i zmiana nazwy. Obiekt *DataFrame* ma wbudowaną możliwość elastycznej zmiany kształtu i przestawiania danych, a także scalania ze sobą wielu ramek danych. Dobrze się sprawdza także w przetwarzaniu serii czasowych.

Biblioteka Pandas doskonale nadaje się do przygotowania danych, a dzięki integracji z interaktywnymi notatnikami umożliwia testowanie różnych konfiguracji i grupowania danych w locie.

Biblioteka ta nie jest też pozbawiona wad. Pandas nie obsługuje strumieni danych, co oznacza, że jeśli mamy ogromny zbiór danych (rzędu wielu gigabajtów), prawdopodobnie nie będziemy mogli zmieścić wszystkiego w pamięci naszego komputera treningowego.

Ważne W dalszej części tego rozdziału zapoznamy się z biblioteką ML.NET. Biblioteka ta zawiera interfejs *IDataView*, który wspiera streaming danych. Jeśli dysponujemy np. plikiem z danymi o rozmiarze 1 TB, możemy z niego skorzystać na dowolnym komputerze, ponieważ proces treningowy odczyta (w rzeczywistości skorzysta ze strumienia) dane w miarę potrzeb, bez konieczności wczytywania wszystkiego do pamięci.

Wizualizacja danych

Matplotlib (*https://matplotlib.org*) jest biblioteką pomocniczą, która chociaż nie jest bezpośrednio związana z żadnym typowym zadaniem z potoku uczenia maszynowego, bardzo przydaje się do wizualnej reprezentacji danych w wielu fazach przygotowania danych lub miar otrzymywanych po ocenie wytrenowanych modeli.

Ogólnie rzecz biorąc, jest to jedynie biblioteka do wizualizacji danych, utworzona z myślą o kodzie w języku Python. Zawiera silnik renderowania 2D i obsługuje popularne typy wykresów, takie jak histogramy, wykresy kołowe i kolumnowe. Wykresy można w pełni dostosowywać do swoich potrzeb, modyfikując style linii, właściwości czcionki, osie, legendy, itp.

Obliczenia numeryczne

Jako język w dużej mierze wykorzystywany w środowiskach naukowych, Python zawiera wiele rozszerzeń, zaprojektowanych do celów obliczeń numerycznych. Do popularnych bibliotek z tego obszaru, różniących się nieco możliwościami, należą NumPy i SciPy.

NumPy (www.numpy.org) koncentruje się na operacjach na tablicach i oferuje funkcje służące do tworzenia, przetwarzania i przekształcania tablic jedno- i wielowymiarowych. Biblioteka ta obsługuje też algebrę liniową, transformacje Fouriera i operacje na liczbach losowych.

SciPy (*https://scipy.org*) rozszerza NumPy o wielomiany, operacje wejścia-wyjścia na plikach, przetwarzanie obrazów i sygnałów oraz bardziej zaawansowane funkcje, takie jak integracja, interpolacja, optymalizacja i statystyka.

W obszarze obliczeń naukowych warto wspomnieć też o innej bibliotece Pythona, czyli o Theano (*http://deeplearning.net/software/theano*). Theano bardzo wydajnie wykonuje wyrażenia matematyczne oparte na tablicach wielowymiarowych, wykorzystując możliwości GPU. Wykonuje też różniczkowanie symboliczne funkcji jednej lub wielu zmiennych.

Trenowanie modelu

Biblioteka scikit-learn (*https://scikit-learn.org*), oryginalnie zaprojektowana z myślą o eksploracji danych, koncentruje się głównie na trenowaniu modelu. Zapewnia implementacje popularnych algorytmów służących do regresji, klasyfikacji i grupowania, a także metody wstępnego przygotowania danych, takie jak redukcja wymiarów, ekstrakcja cech i normalizacja.

Większość analityków danych wykorzystuje pakiet scikit, zawierający bibliotekę scikit-learn z algorytmami, a także SciPy i NumPy do obliczeń. Natomiast w samym uczeniu maszynowym popularna jest także biblioteka Pandas, ze względu na swoje możliwości rekombinacji danych i eksploracji ich zawartości, ułatwiające wybór najodpowiedniejszego algorytmu. Warto też wspomnieć, że biblioteka scikit-learn udostępnia metody selekcji modelu oraz wbudowane narzędzia służące do oceny wydajności trenowanych modeli na podstawie określonych miar.

W skrócie, scikit-learn jest podstawą uczenia płytkiego z wykorzystaniem języka Python.

Sieci neuronowe

Płytkie uczenie jest obszarem uczenia maszynowego, który obejmuje szeroki zakres podstawowych problemów, takich jak regresja i klasyfikacja. Poza królestwem uczenia płytkiego rozciąga się świat uczenia głębokiego i sieci neuronowych. Do budowania sieci neuronowych w Pythonie dostępne są bardziej wyspecjalizowane biblioteki.

TensorFlow (www.tensorflow.org) jest prawdopodobnie najpopularniejszą biblioteką do trenowania głębokich sieci neuronowych. Wchodzi w skład kompleksowej platformy, którą można zaprogramować na wielu poziomach. Sieci neuronowe można tworzyć za pomocą wysokopoziomowego API Keras lub ręcznie zbudować pożądaną topologię, definiując w kodzie poszczególne kroki, niestandardowe warstwy i pętle treningowe. Ogólnie rzecz biorąc, TensorFlow jest kompleksową platformą uczenia maszynowego, umożliwiającą trenowanie i wdrażanie. Zauważmy jednak, że nie można postrzegać biblioteki TensorFlow jako element jedynie ekosystemu Pythona. TensorFlow jest natywną biblioteką utworzoną w języku C, która oferuje powiązania dla Pythona i innych języków, włącznie z .NET poprzez biblioteki TensorFlow.NET i ML.NET.

Keras (*https://keras.io*) jest prawdopodobnie najprostszym sposobem na wizytę w oszałamiającym świecie uczenia głębokiego z użyciem Pythona. Oferuje bardzo prosty interfejs programistyczny, który przydaje się w szybkim prototypowaniu. Jak wspomniano, Keras można wykorzystać z poziomu platformy TensorFlow.

Jeszcze jedną opcją dostępną w języku Python jest biblioteka PyTorch, dostępna pod adresem *https://pytorch.org*. PyTorch jest adaptacją istniejącej biblioteki C, wyspecjalizowanej w przetwarzaniu języka naturalnego oraz rozpoznawaniu obrazów. Spośród wspomnianych opcji sieci neuronowych najlepiej zacząć od biblioteki Keras, z której należy korzystać zawsze, gdy tylko umożliwia osiągnięcie naszego celu. PyTorch i TensorFlow również umożliwiają budowanie wyrafinowanych sieci neuronowych, lecz stosują przy tym inne podejście. TensorFlow wymaga zdefiniowania całej topologii sieci przed rozpoczęciem trenowania, natomiast PyTorch stosuje bardziej zwinne podejście i umożliwia wprowadzanie zmian w grafie w bardziej dynamiczny sposób. Różnicę między nimi oddaje porównanie „metodyki wodospadu z metodykami zwinnymi". PyTorch jest młodszą biblioteką i nie może się poszczycić tak wielką społecznością, jak TensorFlow.

Kompleksowe rozwiązania wykorzystujące modele Pythona

Za pomocą Pythona można z łatwością zbudować i wytrenować model uczenia maszynowego. Model jest plikiem binarnym, który należy wczytać do aplikacji klienckiej i w pewien sposób wywołać. Najczęściej aplikacją kliencką dla modelu utworzonego w języku Python jest program napisany w języku Java lub .NET. Wytrenowany model można wykorzystać poza Pythonem na trzy najważniejsze sposoby, z których żaden nie jest idealny:

- Hostowanie wytrenowanego modelu w usłudze i udostępnienie jego logiki poprzez interfejs API typu REST lub gRPC.

- Pozwolenie aplikacji klienckiej na import wytrenowanego modelu w postaci zserializowanego pliku i interakcje z nim poprzez interfejs programistyczny udostępniony przez infrastrukturę, na której opiera się model. Jest to możliwe tylko pod warunkiem, że wspomniana infrastruktura umożliwia powiązanie z językiem, w którym napisano aplikację kliencką.
- Udostępnienie wytrenowanego modelu poprzez nowy uniwersalny format ONNX. W tym przypadku aplikacja kliencka może wykorzystać wrapper do wczytania plików binarnych ONNX. Zauważmy jednak, że zwykle ONNX nie obsługuje pewnych aspektów oryginalnych platform.

Najczęściej hostuje się model w środowisku umożliwiającym jego udostępnienie poprzez API typu REST lub gRPC. W ten sposób działa TensorFlow ze swoją architekturą TensorFlow Serving, a także scikit-learn i platforma Flask. Należy zauważyć, że usługa HTTP wpływa na architekturę aplikacji (efekt ten jest poważniejszy ze względu na utworzenie modelu w Pythonie); w każdym razie wprowadza opóźnienie w systemie. Jest to jedna z przyczyn, dla których firmy interesują się modelami ML.NET, ponieważ mogą je uruchamiać bezpośrednio w ramach istniejących aplikacji .NET.

Jak wspomniano, TensorFlow udostępnia bezpośrednie powiązania z wieloma językami, dzięki czemu programiści aplikacji klienckiej mogą skorzystać z modelu bez konieczności używania internetowego interfejsu API. Użycie bezpośredniego interfejsu API dla konkretnego języka może się wydawać szybszym sposobem użycia wytrenowanego modelu. Jednak należy przeanalizować kilka aspektów:

- Bezpośrednie użycie API może uniemożliwić wykorzystanie zalet akceleracji sprzętowej i rozproszenia sieci. Jeśli interfejs API jest hostowany lokalnie, w rzeczywistości cały przeznaczony dla naszych celów sprzęt (np. GPU) zależy od nas. Z tego względu, jeśli chcemy wywoływać model z wysoką częstotliwością w czasie rzeczywistym, powinniśmy rozważyć użycie jako hosta chmury ad hoc z akceleracją sprzętową. Zauważmy też, że GPU przynosi korzyści tylko w uczeniu głębokim, czyli przy używaniu bibliotek TensorFlow lub PyTorch. W przypadku zwykłych algorytmów uczenia maszynowego (np. scikit-learn) GPU nie da żadnych dodatkowych korzyści.
- Wytrenowany model nie zawsze oferuje powiązanie z wybranym przez nas językiem (lub platformą). Przykładowo TensorFlow natywnie wspiera języki Python i C, a także C++, Go, Swift i Java. Natomiast scikit-learn można tylko wykorzystać w języku Python lub za pośrednictwem usługi HTTP.

Zauważmy, że z biblioteki TensorFlow można też korzystać w aplikacji .NET. Jest to możliwe na niskim poziomie poprzez zewnętrzną bibliotekę TensorFlow.NET, która obejmuje cały niskopoziomowy interfejs API TensorFlow w taki sam sposób, jak powiązania z innymi językami (np. Java). Inny sposób polega na użyciu najnowszej platformy ML.NET, którą niebawem poznamy. Co ciekawe, platforma ML.NET

umożliwia bezpośrednie wywołania modelu TensorFlow oraz importowanie wytrenowanego modelu poprzez format ONNX.

Wywołanie biblioteki Pythona lub C++ w kodzie .NET nie jest niemożliwe pod względem technicznym. Jednak wywołanie określonej biblioteki, takiej jak wytrenowany model uczenia maszynowego, jest zwykle trudniejsze niż wywołanie pewnej zwykłej klasy Pythona lub C++. W rzeczywistości uczenie maszynowe nie jest odosobnione i należy je opakować w kontekście całościowego rozwiązania biznesowego – a istnieje wiele rozwiązań biznesowych opartych na stosie technologicznym .NET.

Jakie są zatem natywne możliwości uczenia maszynowego dostępne w .NET? Przedstawiamy ML.NET.

Wstęp do ML.NET

Udostępniona wiosną 2019 platforma ML.NET jest darmową, wieloplatformową i otwartą (open source) biblioteką zaprojektowaną z myślą o budowaniu i trenowaniu modeli uczenia maszynowego i hostowania ich w aplikacjach .NET Core i .NET Framework, a także w bibliotekach.NET. Strona poświęcona tej technologii jest dostępna pod adresem *https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet*.

Chociaż dość młoda i nadal rozwijana, platforma ML.NET ma na celu demokratyzację uczenia maszynowego dla programistów. Podejmuje próbę jego uproszczenia do poziomów, które są wystarczająco zrozumiałe dla programistów. Analitycy danych nie są jej główną grupą docelową, co oznacza, że ML.NET nie uwzględnia wszystkich podejść stosowanych w analizie danych.

Najciekawszym aspektem ML.NET jest udostępnienie pragmatycznej platformy programistycznej, skupionej wokół idei predefiniowanych *zadań uczenia*. Biblioteka ta zawiera narzędzia, dzięki którym nawet osoby stawiające pierwsze kroki w uczeniu maszynowym mogą dość łatwo przeprowadzić typowe scenariusze uczenia maszynowego, takie jak analiza sentymentu, wykrywanie oszustw oraz przewidywanie cen, zupełnie jakby to było zwykłe programowanie. Liczba wbudowanych zadań prawdopodobnie będzie stopniowo rosnąć.

W porównaniu do opisanych wcześniej filarów ekosystemu Pythona, ML.NET można przede wszystkim uznać za odpowiednik biblioteki budowania modeli *scikit-learn*. Platforma zawiera jednak pewne podstawowe narzędzia do przygotowania danych i analizy, które są dostępne w bibliotekach Pandas lub NumPy. Warto jednak zauważyć, że cała biblioteka ML.NET opiera się na niesamowitych możliwościach platform .NET Core i .NET Framework.

Aby skorzystać z platformy ML.NET, należy zainstalować pakiet ML.NET i zacząć tworzyć modele w czystym języku C#, używając dowolnego edytora w systemie Windows, Mac lub Linux. Dostępne jest też specjalizowane rozszerzenie – Model Builder – dla programu Visual Studio 2019.