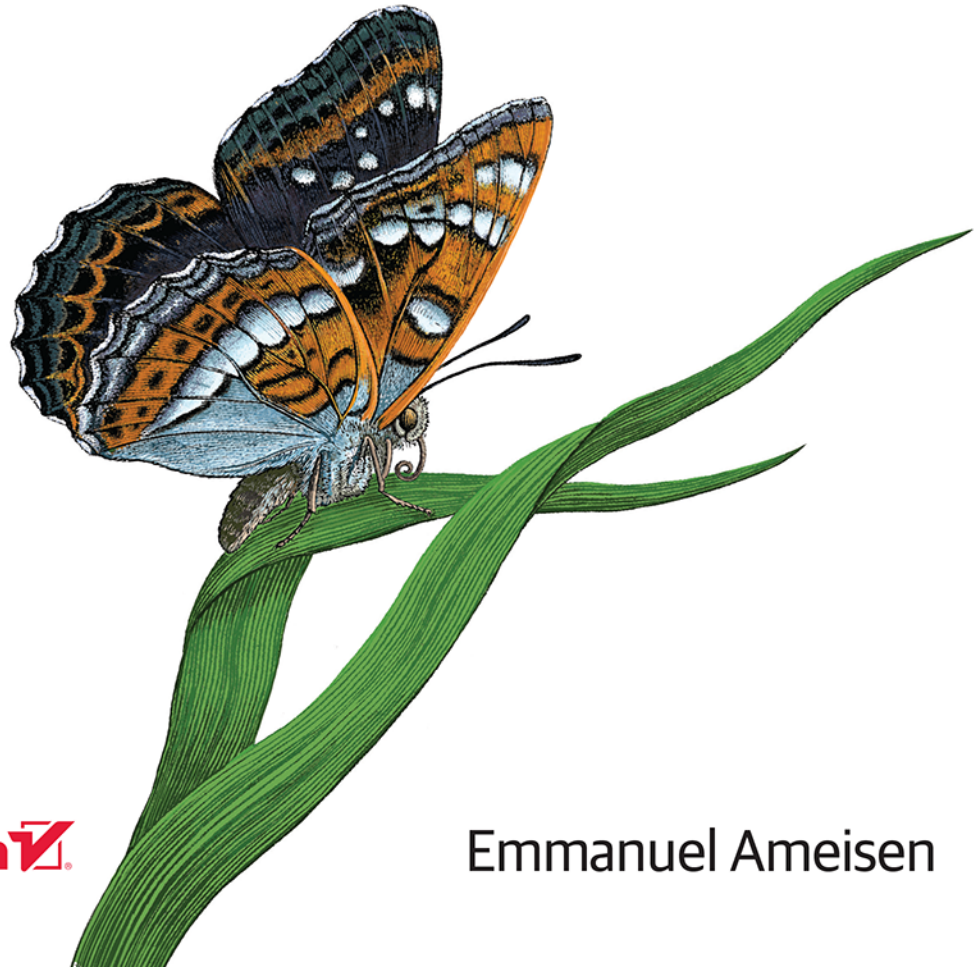
O'REILLY®

Uczenie maszynowe w aplikacjach

Projektowanie, budowa i wdrażanie



Helion

Emmanuel Ameisen

Tytuł oryginału: Building Machine Learning Powered Applications: Going from Idea to Product

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-7186-6

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

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

Printed in Poland.

Spis treści

Część II. Budowanie procesu

Część III. Iterowanie modeli

Część IV. Wdrażanie i monitorowanie modeli

ROZDZIAŁ 2.

Opracowanie planu

W poprzednim rozdziale opisałem, jak oceniać, czy niezbędne jest stosowanie ML, w którym miejscu można je najwłaściwiej wykorzystać i jak przejść od celu do odpowiedniego podejścia ML. Ten rozdział jest poświęcony wskaźnikom kontrolującym ML i postępy prac, jak również porównywaniu różnych implementacji. W dalszej części zajmiemy się metodami tworzenia modelu odniesienia i planowaniem iteracji modelujących.

Wielokrotnie miałem do czynienia z projektami, które już na początku były skazane na niepowodzenie w wyniku niedopasowania wskaźników produktu do wskaźników modelu. Częściej produkt jest nieudany z powodu nieprzydatnego, choć dobrego modelu, niż wskutek trudności modelowania. Dlatego postanowiłem poświęcić osobny rozdział wskaźnikom i planowaniu.

Przedstawię wskazówki, jak przygotować skuteczny plan uwzględniający istniejące zasoby i ograniczenia, który powinien ułatwić realizację każdego projektu ML.

Zacznijmy od szczegółowego zdefiniowania wskaźników wydajnościowych.

Mierzenie postępów

Pierwszy model ML powinien być najprostszym modelem spełniającym wymagania produktu, ponieważ generowanie i analizowanie wyników zapewnia najszybsze postępy w ML. W poprzednim rozdziale opisałem trzy potencjalne, wymienione niżej dla przypomnienia, podejścia zwiększające złożoność edytora ML.

Odniesienie; projektowanie heurystyk opartych na wiedzy dziedzinowej

> Na początku można samodzielnie zdefiniować reguły, bazując na dotychczasowej wiedzy o tym, co składa się na dobrze napisaną treść. Sprawdzimy, czy reguły te pozwolą odróżnić dobrze napisany tekst od złego.

Prosty model; klasyfikowanie dobrego i złego tekstu oraz wykorzystanie klasyfikatora do przygotowywania rekomendacji

> Następnie przetrenujemy prosty model rozróżniający dobre i złe pytania. Jeżeli będzie działał dobrze, sprawdzimy, jakie znalazł cechy, które z dużym prawdopodobieństwem określają dobre pytania. Cechy te wykorzystamy jako zalecenia.

Złożony model; trenowanie pełnego modelu przekształcającego zły tekst w dobry

Jest to najbardziej skomplikowane podejście, zarówno pod względem modelu, jak i danych. Jeżeli jednak istnieje możliwość zebrania danych treningowych, jak również zbudowania i przetrenowania skomplikowanego modelu, można w ten sposób bezpośrednio spełnić wymagania produktu.

Powyższe podejścia różnią się od siebie i mogą ewoluować, w miarę jak będziemy dowiadywać się coraz więcej o prototypach. Jednak pracując nad ML, należy zdefiniować zbiór ogólnych wskaźników umożliwiających mierzenie postępów w procesie modelowania.

Nie zawsze ML jest potrzebne

Zapewne zauważyłeś, że w podejściu z odniesieniem, ML w ogólne nie jest wykorzystywane. Jak pisałem w rozdziale 1., niektóre cechy nie wymagają ML. Ważna jest również świadomość, że nawet cechy bazujące na ML mogą w pierwszej wersji po prostu wykorzystywać heurystykę. W takim przypadku może się okazać, że ML w ogóle nie jest potrzebne.

Budowanie heurystyki jest również często szybszym sposobem niż budowanie cech. Gdy cechy są zbudowane i stosowane, uzyskuje się lepszy obraz potrzeb użytkownika. Dzięki temu można określić, czy ML jest potrzebne i wybrać podejście modelujące.

W większości przypadków start bez ML jest najszybszą drogą do zbudowania produktu opartego na ML.

Opiszę cztery kategorie wydajności, które mają duży wpływ na przydatność każdego produktu ML: wskaźniki biznesowe, wskaźniki modelu, aktualność i szybkość. Jasno zdefiniowane wskaźniki pozwolą nam dokładnie mierzyć wdrożenie każdej iteracji.

Wydajność biznesowa

Pisałem już, jak ważne jest jasne określenie na początku przeznaczenia produktu lub cech. Jeżeli cel jest jasno określony, należy zdefiniować wskaźniki sukcesu. Trzeba je oddzielić od wskaźników modelu. Powinny one odzwierciedlać wyłącznie sukces produktu. Wskaźniki produktowe mogą być proste, na przykład liczba użytkowników, dla których dana cecha jest atrakcyjna, lub bardziej zaawansowane, jak współczynnik klikalności (ang. *click-through rate* — CTR) prezentowanych rekomendacji.

Wskaźniki produktowe są jedynymi, które mają znaczenie, ponieważ reprezentują przeznaczenie produktu lub funkcji. Wszystkie inne wskaźniki powinny służyć jako narzędzia do poprawiania wskaźników produktowych. Wskaźniki te nie muszą być unikatowe. Większość projektów skupia się na poprawianiu jednego wskaźnika produktowego, wpływającego jednak na wiele innych wskaźników, również nadrzędnych, które nie powinny spadać poniżej określonego poziomu. Na przykład projekt ML może się skupiać na zwiększaniu wskaźnika CTR i utrzymaniu innych wskaźników, takich jak średnia długość sesji użytkownika.

W przypadku edytora ML wybierzemy wskaźniki określające przydatność rekomendacji. Może to być na przykład odsetek sugestii wykorzystywanych przez użytkowników. Aby można było wyliczyć tego rodzaju wskaźnik, interfejs edytora powinien wykrywać przypadki przyjęcia sugestii przez użytkownika. Może w tym celu na przykład wyświetlać sugestie powyżej wypełnianego pola, aby użytkownik mógł je klikać.

Jak już wiesz, w każdym produkcie można zastosować kilka potencjalnych podejść ML. Aby mierzyć ich efektywność, trzeba śledzić wydajność modelu.

Wydajność modelu

W przypadku większości produktów internetowych, wskaźnikiem determinującym skuteczność modelu jest stosunek liczby użytkowników wykorzystujących jego wyniki do liczby wszystkich użytkowników, którzy mogliby z tych wyników korzystać. Na przykład wydajność systemu rekomendacyjnego jest często mierzona jako liczba użytkowników, którzy kliknęli polecany produkt (w rozdziale 8. opisane są potencjalne pułapki tego podejścia).

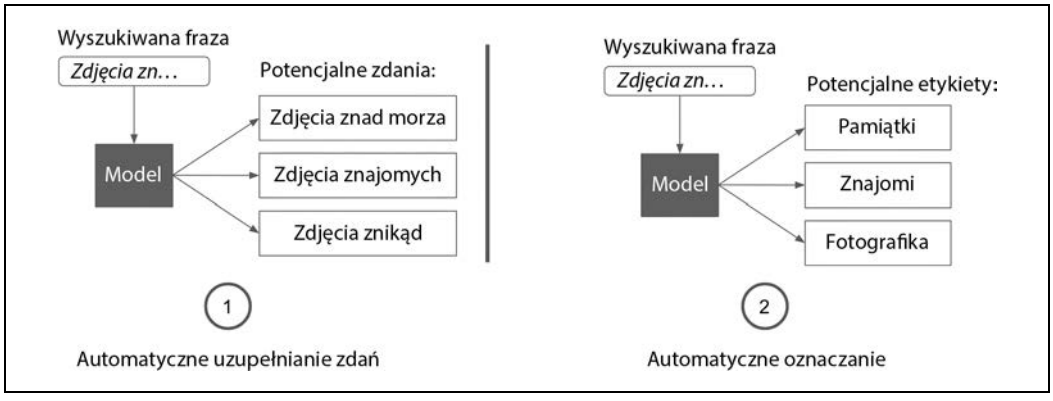
Jeżeli produkt jest jeszcze na etapie tworzenia i nie jest wdrożony, nie ma możliwości wyznaczenia jego wskaźników użycia. Aby ocenić postępy, trzeba zdefiniować osobne wskaźniki sukcesu, zwane **wskaźnikami offline** lub **wskaźnikami modelu**. Dobry wskaźnik offline powinien ewoluować bez konieczności udostępniania modelu użytkownikom i musi być jak najściślej skorelowany ze wskaźnikami i przeznaczeniem produktu.

W różnych podejściach stosuje się różne wskaźniki. Zmieniając podejścia, można o wiele łatwiej osiągnąć poziom wydajności modelowania wystarczający do osiągnięcia celów produktowych.

Załóżmy, że chcemy oferować przydatne sugestie użytkownikom wpisującym wyszukiwane frazy na stronie sklepu internetowego. Wskaźnikiem sukcesu tego rodzaju funkcjonalności może być CTR określający, jak często użytkownicy klikają sugestie.

Aby móc przygotowywać sugestie, należy zbudować model, który będzie odgadywał słowa wpisywane przez użytkownika i prezentował sugerowane uzupełnienia zdań. Wydajność takiego modelu można mierzyć, wyliczając precyzję słów, czyli częstość prognozowania właściwych słów. Aby model mógł zwiększyć wskaźnik CTR produktu, musi mieć wyjątkowo wysoką precyzję, ponieważ jedno błędnie prognozowane słowo może zniweczyć przydatność sugestii. Podejście to jest pokazane po lewej stronie rysunku 2.1.

Inne podejście polega na trenowaniu modelu, który klasyfikuje dane wprowadzane przez użytkownika według kategorii umieszczonych w katalogu i sugeruje najbardziej prawdopodobne kategorie. Wydajność modelu można mierzyć jako dokładność względem liczby kategorii, a nie wszystkich słów w języku. Ponieważ liczba kategorii w katalogu jest znacznie mniejsza od liczby słów w języku, optymalizacja wskaźników powinna być znacznie łatwiejsza. Ponadto model musiałby poprawnie prognozować tylko jedną kategorię, aby wygenerować kliknięcie. W przypadku takiego modelu znacznie łatwiej jest zwiększyć wskaźnik CTR produktu. Prawa strona rysunku 2.1. pokazuje, jak takie podejście wygląda w praktyce.



Rysunek 2.1. Niewielka modyfikacja produktu może znacznie ułatwić modelowanie

Jak widać, wprowadzenie niewielkich zmian w interakcji pomiędzy modelem a produktem pozwala zastosować prostsze podejście modelujące i uzyskać bardziej wiarygodne wyniki. Poniżej opisanych jest kilka innych przykładów modyfikacji aplikacji, mogących ułatwić zadanie modelowania:

- *Zmiana interfejsu tak, aby można było pomijać wyniki modelu, których poziom ufności jest mniejszy od określonego progu.* Na przykład model automatycznie uzupełniający wpisywane przez użytkownika zdania może działać dobrze tylko w przypadku określonego zbioru zdań. Można zaimplementować algorytm tak, aby pokazywał tylko te sugestie, których poziom ufności jest wyższy niż 90%.
- *Prezentowanie kilku dodatkowych predykcji lub heurystyk oprócz najlepszych predykcji modelu.* Na przykład większość stron internetowych wyświetla więcej niż jedną rekomendację przygotowaną przez model. Wyświetlając pięć proponowanych elementów zamiast tylko jednego, można zwiększyć prawdopodobieństwo, że sugestie będą przydatne dla użytkowników, nawet gdy model będzie taki sam.
- *Informowanie użytkowników, że model jest wciąż na etapie eksperymentów i umożliwienie im wyrażenia opinii.* Witryny automatycznie wykrywające, że użyty w nich język nie jest językiem ojczystym użytkownika, często zawierają przycisk, za pomocą którego użytkownik może wyrazić opinię, czy tłumaczenie było dokładne i przydatne.

Nawet gdy podejście modelujące jest odpowiednie dla problemu, warto zdefiniować kilka dodatkowych wskaźników, lepiej skorelowanych z wydajnością modelu.

Pracowałem kiedyś z badaczem danych, który budował model generujący kod HTML strony na podstawie odręcznego szkicu (patrz artykuł *Automated Front-End Development Using Deep Learning*, *https://oreil.ly/SdYQj*). Optymalizacja modelu polegała na porównywaniu prognozowanych i poprawnych tokenów HTML z wykorzystaniem straty entropii krzyżowej. Przeznaczeniem produktu było jednak generowanie kodu HTML strony, która wyglądałaby jak naszkicowana, niezależnie od kolejności tokenów.

Entropia krzyżowa nie uwzględnia wyrównania. Jeżeli model generuje poprawną sekwencję HTML z wyjątkiem jednego tokenu na początku, wszystkie pozostałe są przesuwane o jeden w porównaniu z celem. Tego rodzaju wynik cechuje się bardzo wysoką stratą, mimo że jest niemal idealny.

Oznacza to, że próbując ocenić przydatność modelu, należy patrzeć nie tylko na wskaźniki optymalizacyjne. W opisanym przykładzie wskaźnik BLEU (*https://oreil.ly/8s9JE*) jest lepszą miarą podobieństwa generowanego kodu HTML do idealnego wyniku.

Poza tym, projektując produkt, należy przyjąć rozsądne założenia dotyczące wydajności modelu. Jeżeli model musi być idealny, aby był użyteczny, bardzo prawdopodobne jest, że będzie dostarczał niedokładnych, a nawet niebezpiecznych wyników.

Jeżeli na przykład model ma na podstawie zdjęcia pastylki określać jej rodzaj i dawkę, jaka może być jego najgorsza dokładność, aby wciąż był użyteczny? Jeżeli trudno jest spełnić wymóg dokładności, wykorzystując bieżące metody, czy można przeprojektować produkt tak, aby użytkownicy byli dobrze obsługiwani, bez narażania ich na ryzyko potencjalnie błędnych prognoz?

W naszym przypadku model, który zamierzamy zbudować, ma dostarczać wskazówek dotyczących wpisywanego tekstu. Większość modeli ML ma pewne dane wejściowe, w których się wyróżniają, i pewne dane wejściowe, z którymi będą się zmagać. Z perspektywy produktu, jeżeli nie można w nim nic poprawić, trzeba mieć pewność, że nie będzie szkodził. Trzeba ograniczyć czas generowania wyniku, który jest gorszy niż dane wejściowe. Jak można to wyrazić za pomocą wskaźników modelu?

Załóżmy, że budujemy model klasyfikujący, który ma prognozować na podstawie liczby polubień, czy pytanie jest dobre. Dokładność klasyfikatora można zdefiniować jako proporcję pomiędzy liczbą pytań faktycznie dobrych a liczbą prognozowanych jako dobre. Z drugiej strony, nożе to być stosunek liczby pytań prognozowanych jako dobre do wszystkich dobrych pytań w zbiorze danych.

Jeżeli zawsze chcemy uzyskiwać odpowiednią podpowiedź, musimy określić priorytet dokładności modelu. Jeżeli model o wysokiej **precyzji** sklasyfikuje pytanie jako dobre (i będzie je rekomendował), z dużym prawdopodobieństwem będzie ono faktycznie dobre. Wysoka precyzja modelu oznacza, że zwykle jego rekomendacje są poprawne. Aby dowiedzieć się więcej, dlaczego precyzyjne modele przydają się przy rekomendowaniu wpisywanego tekstu, zajrzyj do podrozdziału „Chris Harland: eksperymenty wysyłkowe” w rozdziale 8.

Tego rodzaju wskaźniki wylicza się na podstawie wyników generowanych przez model na podstawie reprezentatywnego zbioru weryfikacyjnego. W szczegóły tego zagadnienia zagłębimy się w podrozdziale „Ocena modelu: nie tylko dokładność” w rozdziale 5. Na razie przyjmijmy, że zbiór weryfikacyjny są to dane, które nie są wykorzystywane do trenowania modelu, tylko do oceniania jego wydajności przetwarzania nieznanych wcześniej danych.

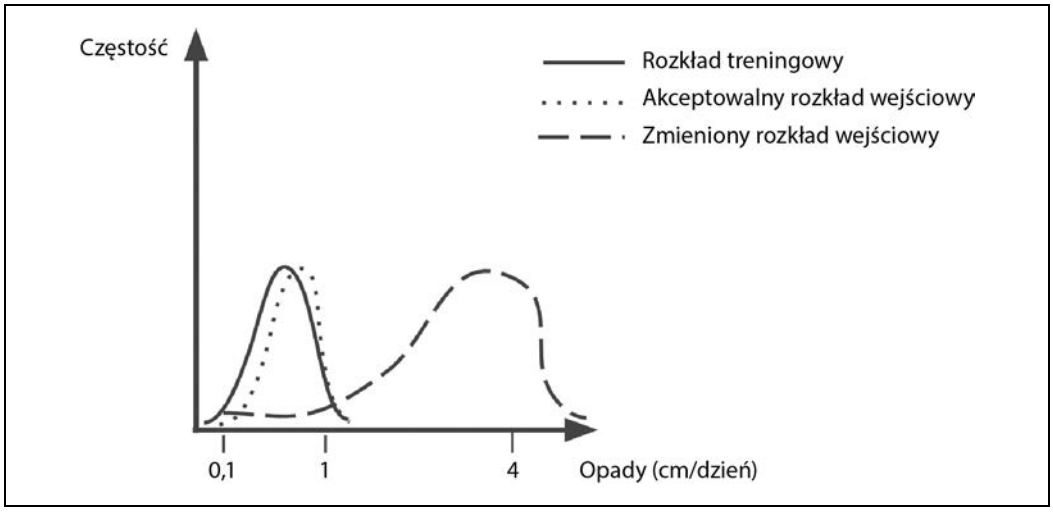
Początkowa wydajność modelu jest ważna, podobnie jak jego użyteczność przy zmieniających się zachowaniach użytkowników. Model przetrenowany na określonych danych będzie dobrze przetwarzał podobne dane, ale jak się dowiedzieć, czy trzeba zaktualizować zbiór danych?

Aktualność i przesunięcie rozkładu danych

Siła modeli nadzorowanych bierze się z korelacji pomiędzy cechami wejściowymi a prognozowanymi danymi docelowymi. Oznacza to, że w większości przypadków, aby model działał dobrze, trzeba go trenować na danych podobnych do wejściowych. Model, który ma prognozować wiek użytkownika na postawie jego zdjęcia, trenowany tylko na zdjęciach mężczyzn, nie będzie działał dobrze w przypadku zdjęć kobiet. Jednak nawet jeżeli model jest trenowany na odpowiednim zbiorze danych,

źródłem wielu problemów jest ich dystrybucja, która zmienia się w miarę upływu czasu. W takim wypadku zazwyczaj model też trzeba *zmieniać*, aby utrzymać jego dotychczasową wydajność.

Załóżmy, że zauważyliśmy wpływ deszczu na ruch uliczny i budujemy model prognozujący warunki ruchu na podstawie ilości opadów w ubiegłym tygodniu. Jeżeli model zostanie zbudowany w październiku na podstawie danych z ostatnich trzech miesięcy, będzie prawdopodobnie trenowany na dziennych opadach rzędu centymetra. Rysunek 2.2. przedstawia przykładowy wygląd takiej dystrybucji. Gdy nadchodzi zima, średnie opady zbliżają się do poziomu 3 centymetrów. Są to wartości większe niż użyte do trenowania modelu, co ilustruje rysunek 2.2. Jeżeli model nie zostanie przetrenowany na aktualnych danych, nie będzie działał dobrze.



Rysunek 2.2. Zmiana rozkładu danych

Ogólnie model działa dobrze na nieznanych mu wcześniej danych, jeżeli są one podobne do użytych w treningu.

Nie wszystkie modele mają takie same wymagania dotyczące aktualności danych. W przypadku usługi tłumaczącej treści napisane w starożytnych językach można się spodziewać, że przetwarzane dane będą stałe. Natomiast tworząc wyszukiwarki internetowe, należy przyjąć, że będą one musiały szybko ewoluować w miarę zmieniających się zwyczajów użytkowników.

W zależności od problemu biznesowego należy ocenić, jak trudno będzie utrzymać *aktualność* modelu. Jak często trzeba będzie go trenować i ile to będzie za każdym razem kosztować?

Wyobraźmy sobie, że w przypadku edytora ML częstość zmian definicji dobrze sformułowanej wypowiedzi jest dość niska, na przykład jednoroczna. Wymagania dotyczące aktualności danych mogą się jednak zmienić, jeżeli zmieni się dziedzina docelowa. Sposoby zadawania poprawnych pytań z matematyki będą się zmieniały wolniej niż w przypadku trendów muzycznych. Jeżeli oszacujemy, że model trzeba trenować co roku, równie często będą potrzebne aktualne dane.

Nasz prosty model odniesienia może się uczyć również na niesparowanych danych, co ułatwia proces ich zbierania (trzeba po prostu co roku wyszukiwać nowe pytania). Złożony model wymaga sparowa-

nych danych. Oznacza to, że co roku trzeba wyszukiwać przykłady tych samych dobrych i złych zdań. Spełnienie wymagań dotyczących aktualności danych w przypadku modelu oczekującego sparowanych danych będzie znacznie trudniejsze, ponieważ pozyskanie aktualnego zbioru będzie bardziej czasochłonne.

W przypadku większości aplikacji ich popularność może złagodzić wymagania dotyczące gromadzenia danych. Jeśli usługa formułowania pytań zyska na popularności, można dodać przycisk umożliwiający użytkownikom ocenianie jakości wyników. Następnie można zebrać dane wprowadzone wcześniej przez użytkowników wraz z prognozami modelu i ocenami oraz wykorzystać je jako zbiór treningowy.

Jednak aby aplikacja była popularna, musi być użyteczna. Często wymaga to terminowego reagowania na żądania użytkowników. Szybkość, z jaką model może dostarczać prognozy, jest zatem ważnym czynnikiem, który należy wziąć pod uwagę.

Szybkość

Idealny model powinien prognozować szybko. Ułatwia to użytkownikom posługiwanie się nim, jak również pozwala oferować model wielu użytkownikom jednocześnie. Jak zatem szybki powinien być model? W niektórych zastosowaniach, na przykład tłumaczeniu krótkich zdań, użytkownicy oczekują natychmiastowych odpowiedzi. W innych, takich jak diagnozy medyczne, pacjent może poczekać 24 godziny, aby dostać jak najdokładniejsze wyniki.

W naszym przypadku rozważymy dwie możliwości dostarczania sugestii: poprzez pole, w którym użytkownik wpisuje tekst, klika przycisk *Wyślij* i uzyskuje wyniki, lub poprzez dynamiczne aktualizowanie treści podczas wpisywania kolejnych liter. Preferowane jest drugie rozwiązanie, ponieważ narzędzie będzie wtedy bardziej interaktywne. Jednak model musiałby wtedy działać znacznie szybciej.

Można sobie wyobrazić, że użytkownik po kliknięciu przycisku *Wyślij* będzie czekał kilka sekund na wynik, ale aby model działał sprawnie w miarę wpisywania testu przez użytkownika, musiałby odpowiadać w ciągu ułamka sekundy. Najlepsze modele przetwarzają dane długo, dlatego podczas iterowania modeli trzeba pamiętać o tym wymaganiu. Każdy model powinien przetwarzać przykładowe dane w ciągu maksymalnie dwóch sekund.

Im bardziej złożony model, tym dłuższy jest czas wnioskowania. Różnice są znaczne nawet w dziedzinach, w których poszczególne punkty danych są relatywnie małe, na przykład w przetwarzaniu języka naturalnego (w przeciwieństwie do takich zadań, jak analiza obrazu na żywo). W przypadku danych tekstowych, wykorzystanych w przykładzie opisanym w tej książce, model LSTM (ang. *Long Short--Term Memory* — długa pamięć krótkoterminowa) jest mniej więcej trzy razy wolniejszy od modelu losowego lasu (pierwszy odpowiada w ciągu ok. 22 ms, a drugi w ciągu 7 ms). Dla pojedynczych punktów danych różnice są niewielkie, ale szybko się kumulują, gdy trzeba przetwarzać za jednym razem dziesiątki tysięcy przykładów.

W skomplikowanych aplikacjach, w których jedno wnioskowanie jest powiązane z wieloma zapytaniami sieciowymi, czas działania modelu może być krótki w porównaniu z czasochłonnością innych operacji. W takich sytuacjach szybkość samego modelu staje się mniejszym problemem.

W zależności od problemu, należy rozważyć inne kwestie, takie jak ograniczenia sprzętowe, czas programowania i łatwość utrzymania aplikacji. Przed wybraniem modelu ważne jest zrozumienie potrzeb, aby można było podjąć świadomą decyzję.

Po określeniu wyzwań i związanych z nimi wskaźników należy rozpocząć przygotowywanie planu. Wymaga to oszacowania przyszłych wyzwań. W następnym podrozdziale opisane są sposoby wykorzystania dotychczasowej pracy i badania zbioru danych, umożliwiające podejmowanie decyzji, co budować dalej.

Szacowanie zakresu i wyzwań

Jak już wiesz, wydajność ML często opisuje się za pomocą wskaźników modelu. Wskaźniki te są przydatne, ale należy je stosować w celu poprawiania wskaźników zdefiniowanego produktu, reprezentujących bieżące zadania przeznaczone do wykonania. Podczas iterowania procesu należy pamiętać o wskaźnikach produktu i starać się je poprawiać.

Opisane do tej pory narzędzia pozwalają określić, czy projekt w ogóle jest wart realizacji oraz jak dobrze sobie z nią obecnie radzimy. Kolejnym logicznym krokiem jest nakreślenie planu, aby można było oszacować zakres i czasochłonność projektu oraz przewidzieć potencjalne przeszkody.

Osiągnięcie sukcesu w dziedzinie ML zazwyczaj wymaga zrozumienia kontekstu zadania, pozyskania dobrego zbioru danych i zbudowania odpowiedniego modelu.

Powyższe zagadnienia są opisane w kolejnych podrozdziałach.

Wykorzystanie doświadczenia w danej dziedzinie

Najprostszym modelem, od którego można zacząć, jest heurystyka. Jest to dobra, niepisana zasada, oparta na znajomości problemu i danych. Najlepszym sposobem opracowania heurystyki jest sprawdzenie, co obecnie robią eksperci. Większość praktycznych aplikacji nie jest wymyślana od podstaw. Jak inni obecnie rozwiązują problem, który sam starasz się rozwiązać?

Innym sposobem opracowania heurystyki jest przejrzenie danych. W jaki sposób, bazując na posiadanym zbiorze danych, można zrealizować zadanie, gdyby trzeba było to zrobić ręcznie?

Aby określić dobrą heurystykę, polecam wykorzystać doświadczenia ekspertów w danej dziedzinie lub zapoznanie się z danymi. Obie kwestie są opisane dokładniej w kolejnych podrozdziałach.

Doświadczenia ekspertów

W przypadku wielu domen, które chcielibyśmy automatyzować, wykorzystanie doświadczeń ekspertów może zaoszczędzić dziesiątki godzin pracy. Na przykład budując system prognozowania prac serwisowych w fabryce, warto skontaktować się z menedżerem i dowiedzieć się, jakie rozsądne założenia należy przyjąć. Może to obejmować określenie częstotliwości wykonywania prac serwisowych, zdefiniowania sygnałów, że urządzenie będzie wkrótce wymagało serwisowania, oraz poznania prawnych wymagań dotyczących utrzymania.

Oczywiście są sytuacje, w których znalezienie eksperta w danej dziedzinie jest trudne, ponieważ dane są poufne (na przykład dotyczące prognozowania wykorzystania unikatowej funkcjonalności strony internetowej). W takich przypadkach można jednak znaleźć specjalistów, którzy mieli do czynienia z podobnymi problemami, i wykorzystać ich doświadczenia.

W ten sposób można się dowiadywać o przydatnych funkcjonalnościach, które można wykorzystać, określić pułapki, których należy unikać, i co najważniejsze, nie wynajdywać na nowo koła, co wielu naukowcom przysporzyło złej sławy.

Badanie danych

Zarówno Monica Rogati w podrozdziale „Monica Rogati: jak wybierać projekty ML i określać ich priorytety", jak i Robert Munro w „Robert Munro: jak wyszukiwać, oznaczać i wykorzystywać dane?" stwierdzili, że kluczowe znaczenie przed rozpoczęciem modelowania ma przejrzenie danych.

Badania eksploracyjne (ang. *exploratory data analysis* — EDA) jest to proces wizualizowania i badania zbioru danych, często oparty na intuicji i znajomości danego problemu biznesowego. Badania te są krytyczną częścią każdego produktu wykorzystującego dane. Oprócz tego bardzo ważne jest oznaczenie przykładów w sposób, w jaki prawdopodobnie będzie to robił model. Dzięki temu można zweryfikować założenia i upewnić się, że wybrany model właściwie wykorzystuje zbiór danych.

Proces EDA pozwala zrozumieć trendy w danych, a samodzielne ich oznaczenie zmusza do zbudowania zestawu heurystyk rozwiązujących problem. Po wykonaniu obu opisanych kroków powinieneś mieć lepsze wyobrażenie, jaki model będzie optymalny, jak również jakie dodatkowe dane powinieneś zebrać i jakie będą wymagane strategie ich oznaczania.

Następnym logicznym krokiem jest sprawdzenie, czy ktoś już wcześniej mierzył się z podobnym problemem.

Podążanie za wielkimi

Czy inni rozwiązywali podobne problemy? Jeżeli tak, to najlepszym startem będzie poznanie i odtworzenie uzyskanych przez nich wyników. Warto poszukać publicznych implementacji wykorzystujących podobne modele czy dane lub jedno i drugie.

Najlepiej byłoby znaleźć otwarty kod źródłowy i dostępny zbiór danych. Jednak nie zawsze jest to łatwe, szczególnie w przypadku bardzo specyficznych produktów. Niemniej jednak najszybszym sposobem uruchomienia projektu ML jest odtworzenie istniejących wyników i oparcie się na nich.

W dziedzinie tak różnorodnej jak ML bardzo ważne jest wykorzystanie wyników uzyskanych przez ekspertów.

Jeżeli zamierzasz wykorzystać otwarty kod źródłowy lub dane, sprawdź, czy możesz to zrobić. Większość repozytoriów i zbiorów danych zawiera licencje określające warunki użytkowania. Oprócz tego powinieneś podać źródło, z którego korzystasz, najlepiej z odniesieniem do oryginalnej pracy.

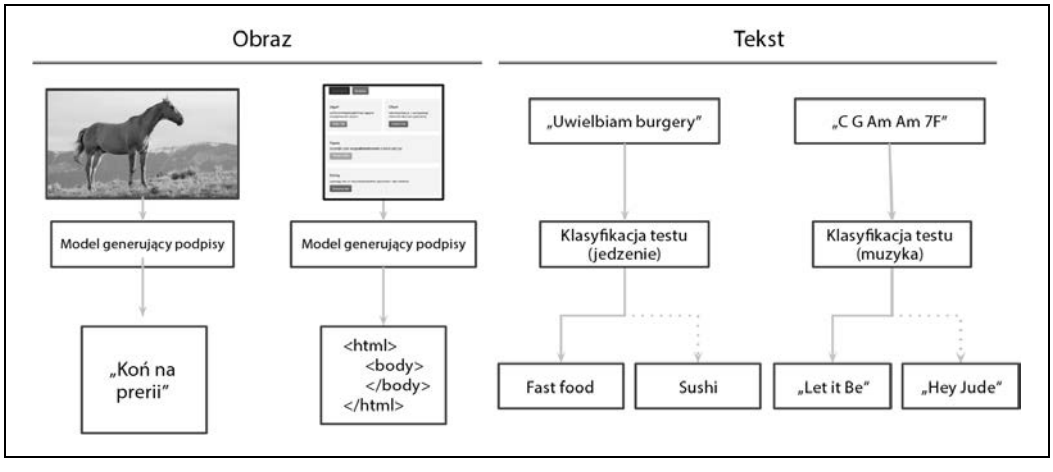
Dobrym pomysłem, przed zaangażowaniem znacznych zasobów w projekt, jest przeprowadzenie wiarygodnych testów. Na przykład przed zainwestowaniem czasu i pieniędzy w oznaczenie danych warto się przekonać, czy można zbudować model, który będzie się uczył na podstawie takich danych.

Jaki jest więc skuteczny sposób, aby zacząć? Podobnie jak większość tematów, które opisuję w tej książce, można wyróżnić dwa główne zadania: przejrzenie danych i kodu.

Otwarte dane

Znalezienie zbioru danych odpowiadającego potrzebom nie zawsze jest możliwe, ale może to być zbiór na tyle podobny, że będzie go można wykorzystać. Co oznacza w tym kontekście słowo „podobny"? Pomocne jest tu potraktowanie modelu ML jako powiązania pomiędzy danymi wejściowymi i wyjściowymi. Podobny zbiór danych oznacza wtedy podobne dane wejściowe i typy wyjściowe (choć nie zawsze domeny).

Często modele wykorzystujące podobne dane wejściowe i wyjściowe można stosować w całkowicie odmiennych kontekstach. Po lewej stronie rysunku 2.3. pokazane są dwa modele prognozujące tekst na podstawie obrazu. Jeden z nich jest wykorzystywany do opisywania zdjęć, a drugi do generowania kodu HTML strony na podstawie zrzutu ekranu. Analogicznie, po prawej stronie rysunku znajduje się model prognozujący nazwę posiłku na podstawie jego opisu, oraz model prognozujący rodzaj muzyki na podstawie transkrypcji.



Rysunek 2.3. Różne modele wykorzystujące podobne dane wejściowe i wyjściowe

Załóżmy, że chcemy zbudować model prognozujący poczytność artykułów prasowych, ale nie możemy znaleźć zbioru artykułów i powiązanych z nimi liczb wyświetleń. W takiej sytuacji możemy pobrać publicznie dostępne statystyki ruchu w serwisie Wikipedia (*https://oreil.ly/PdwgN*) i przetrenować na nich nasz model. Jeżeli wydajność będzie zadowalająca, rozsądne będzie założenie, że dla danego zbioru wyświetleń artykułów model będzie działał dobrze. Wyszukanie podobnego zbioru danych może pomóc zweryfikować zastosowane podejście i wiarygodnie uzasadnić wykorzystanie zasobów w celu pozyskania danych.

Powyższa metoda sprawdza się również w pracy nad zastrzeżonymi danymi. Często zbiór danych, potrzebny do prognozowania, nie jest łatwo dostępny. Dane mogą nie być nawet zbierane. W takich sytuacjach zbudowanie modelu działającego dobrze na podobnym zbiorze jest najlepszym sposobem przekonania udziałowców projektu do zbudowania procesu zbierania nowych danych lub uproszczenia dostępu do istniejących.

W przypadku publicznie dostępnych danych nowe źródła i kolekcje pojawiają się regularnie. Poniżej wymienionych jest kilka z nich, moim zdaniem przydatnych:

- Internet Archive (*https://oreil.ly/tIjl9*), serwis zawierający różne zbiory danych, w tym strony WWW, filmy i książki.
- Reddit, podkatalog *r/datasets* (*http://reddit.com/r/datasets*), serwis przeznaczony do współdzielenia zbiorów danych.
- Kaggle Datasets (*https://www.kaggle.com/datasets*), serwis oferujący duży wybór danych z różnych dziedzin.
- UCI Machine Learning Repository (*https://oreil.ly/BXLA5*), ogromne archiwum danych ML.
- Google Dataset Search (*https://oreil.ly/Gpv8S*), duży indeks różnego rodzaju zbiorów danych.
- Common Crawl (*https://commoncrawl.org*), serwis przeglądający sieć WWW, archiwizujący i publicznie udostępniający dane.
- Wikipedia (*https://oreil.ly/kXGiz*), doskonała i nieustanie powiększająca się lista badawczych zbiorów danych ML.

W większości przypadków przynajmniej jedno z powyższych źródeł zawiera zbiór danych wystarczająco podobnych do tych, które są potrzebne.

Trenowanie modelu na przybliżonym zbiorze danych pozwala szybko tworzyć prototypy i weryfikować wyniki. W niektórych przypadkach można nawet na takich danych przetrenować model i przenieść niektóre wskaźniki wydajnościowe na docelowy zbiór (więcej na ten temat dowiesz się w rozdziale 4.).

Gdy już wiadomo, od jakiego zbioru danych można zacząć, należy zwrócić uwagę na modele. Istnieje pokusa, aby po prostu zacząć budować własny proces od podstaw, ale często lepiej jest przynajmniej zapoznać się z tym, co już zrobili inni.

Otwarty kod

Poszukując istniejącego kodu, można osiągnąć dwa wysokopoziomowe cele: dowiedzieć się, z jakimi wyzwaniami mierzyli się inni podczas podobnego modelowania oraz jakich problemów może przysporzyć posiadany zbiór danych. Dlatego zalecam przyjrzenie się zarówno procesom umożliwiającym osiągnięcie założonego celu, jak i kodowi przetwarzającemu wybrany zbiór danych. Pierwszym krokiem do znalezienia przykładu powinno być samodzielne odtworzenie wyników.

Spotkałem wielu badaczy danych, którzy próbowali wykorzystać znaleziony kod ML tylko po to, aby przekonać się, że trenując model, nie można osiągnąć poziomu dokładności podawanego przez autorów. Ponieważ do nowatorskich rozwiązań nie zawsze jest dołączana dobra dokumentacja i poprawnie działający kod, zazwyczaj trudno jest odtworzyć wyniki. Dlatego zawsze trzeba je weryfikować.

Podobnie jak w przypadku danych, dobrą praktyką jest wyodrębnienie problemu dotyczącego podobnych typów wejściowych i wyjściowych, a następnie poszukanie kodów rozwiązujących podobne problemy.

Na przykład Tony Beltramelli, autor artykułu *pix2code: Generating Code from a Graphical User Interface Screenshot* (*https://oreil.ly/rTQyD*), próbując generować kod HTML strony na podstawie zrzutu ekranu, odkrył, że jego problem można sprowadzić do tłumaczenia obrazu na tekst. Wykorzystał istniejącą architekturę i dobre praktyki stosowane w dojrzalszej dziedzinie i również generował teksty na podstawie obrazów, czyli podpisy do nich. Dzięki temu uzyskał doskonałe wyniki w całkowicie nowej dziedzinie i wykorzystał efekty lat pracy innych osób nad podobną aplikacją.

Po przejrzeniu danych i kodu można iść dalej. Ideałem byłoby uzyskanie w tym procesie wskazówek, które pomogłyby rozpocząć pracę i uzyskać dokładniejszy obraz problemu. Podsumujmy sytuacje, w których można się znaleźć, szukając efektów dotychczasowej pracy innych osób.

Zebranie wszystkiego razem

Jak już wspomniałem, wykorzystując istniejący otwarty kod i dane, można przyspieszyć implementację. W najgorszym przypadku, gdy żaden z istniejących modeli nie działa dobrze z otwartym zbiorem danych, wiadomo przynajmniej, że projekt wymaga znacznej pracy nad modelowaniem i zebrania danych.

Jeżeli uda się znaleźć model rozwiązujący podobny problem, jak również przetrenować go na oryginalnych danych, wystarczy dostosować go do własnej domeny. W tym celu zalecam wykonanie następujących kroków:

1. Znalezienie podobnego, otwartego modelu, w idealnym przypadku wraz z danymi, na którym był trenowany, a następnie samodzielne odtworzenie wyników treningu.
2. Po odtworzeniu wyników — wyszukanie zbioru danych bardziej odpowiedniego do danego przypadku, a następnie przetrenowanie na nim modelu.
3. Po zintegrowaniu zbioru danych z trenowanym kodem można ocenić jakość modelu na podstawie zdefiniowanych wskaźników, a następnie rozpocząć iteracje.

W części II opiszę pułapki kryjące się w każdym z powyższych kroków, a także sposoby ich unikania. Na razie wróćmy do naszego przykładu i przejrzyjmy opisany proces.

Planowanie edytora ML

Zbadajmy popularne porady dotyczące pisania tekstów i poszukajmy potencjalnych zbiorów danych i kodów dla edytora ML.

Początkowy plan edytora

Należy zacząć od implementacji heurystyki opartej na ogólnych zasadach pisania tekstów. Zasady te można znaleźć, przeszukując istniejące poradniki dotyczące pisania i edytowania tekstów, na przykład wymienione w rozdziale 1. w podrozdziale „Najprostsze podejście: wykonanie algorytmu”.

Idealny zbiór danych powinien zawierać pytania i oceny ich jakości. Najpierw należy szybko znaleźć podobny zbiór danych, który można łatwiej pozyskać. Bazując na wydajności tego zbioru, można w razie potrzeby rozszerzyć i pogłębić poszukiwania.

Dobrymi przykładami tekstów i przypisanych im wskaźników jakościowych są wpisy w serwisach społecznościowych i na forach. Ponieważ wskaźniki te są głównie po to, aby faworyzować przydatne treści, w większości są to polubienia lub głosy za.

Popularnym serwisem zawierającym pytania i odpowiedzi społeczności użytkowników jest Stack Exchange (*https://stackexchange.com*). Jego całkowicie zanonimizowane zrzuty są dostępne w jednym ze wspomnianych wcześniej źródeł — serwisie Internet Archive (*https://oreil.ly/NR6iQ*). Jest to doskonały zbiór danych, od którego można zacząć.

Początkowy model można zbudować, wykorzystując pytania z serwisu Stack Exchange i próbując prognozować na podstawie ich treści liczbę głosów poparcia. Przy okazji można przejrzeć zbiór danych, spróbować znaleźć wzorce i go oznaczyć.

Model, który zamierzamy zbudować, będzie miał za zadanie dokładnie oceniać jakość tekstu, a następnie przedstawiać rekomendacje. Istnieje wiele otwartych modeli klasyfikujących teksty. Warto zapoznać się z popularnym podręcznikiem do biblioteki scikit-learn napisanej w języku Python (*https://oreil.ly/y6Qdp*).

Gdy działający klasyfikator będzie gotowy, w rozdziale 7. dowiesz się, jak można go wykorzystać do przygotowywania rekomendacji.

Teraz, mając potencjalny początkowy zbiór danych, zajmijmy się modelami i zadecydujmy, od czego zacząć.

Zawsze należy zaczynać od prostego modelu

Z niniejszego rozdziału płynie ważny wniosek, że celem zbudowania początkowego modelu i zbioru danych jest uzyskanie wartościowych wyników, na podstawie których będzie można prowadzić dalsze modelowanie i gromadzenie danych w kierunku bardziej użytecznego produktu.

Rozpoczynając od prostego modelu i wyodrębniając z pytań w serwisie Stack Overflow cechy decydujące o ich przydatności, można szybko określić wydajność modelu i iterować go.

Odwrotne podejście, polegające na zbudowaniu idealnego modelu od podstaw, nie sprawdza się w praktyce. Jest tak dlatego, że ML jest procesem iteracyjnym, w którym postępy robi się najszybciej, sprawdzając, kiedy model działa źle. Im wcześniej model zawiedzie, tym większe postępy można zrobić. Znacznie dokładniej tym procesem iteracyjnym zajmiemy się w części III.

Należy również pamiętać o mankamentach każdego podejścia. Na przykład ocena pytania zależy nie tylko od jakości jego sformułowania, ale od wielu innych czynników. Bardzo duże znaczenie ma też kontekst wpisu, społeczność, czas publikacji, popularność serwisu i wiele innych szczegółów, które początkowy model pominie. Aby uwzględnić te czynniki, należy ograniczyć zbiór danych do określonych społeczności. Pierwszy model będzie pomijał wszystkie metadane wpisu, ale będzie można rozważyć ich uwzględnienie, jeżeli okaże się to konieczne.

Taki model wykorzystuje tak zwaną *słabą etykietę*, która jest tylko nieznacznie skorelowana z żądanym wynikiem. Analizując wydajność modelu, można określić, czy dana etykieta zawiera wystarczające informacje, aby była użyteczna.

Mamy punkt początkowy i możemy podjąć decyzję, co robić dalej. Robienie regularnych postępów w ML często jest trudne z powodu nieprzewidywalności modelowania. Nie wiadomo zawczasu, w jakim stopniu zastosowane podejście będzie skuteczne. Dlatego poniżej przedstawiam kilka rad, dzięki którym można robić systematyczne postępy.

Sposób na systematyczne postępy — prosty początek

Warto podkreślić, że największe wyzwanie w ML jest takie samo, jak w programowaniu: oparcie się pokusie tworzenia elementów, które nie są jeszcze potrzebne. Wiele projektów ML upada, ponieważ opiera się na początkowym pozyskiwaniu danych i budowaniu planu, który nie jest regularnie oceniany i aktualizowany. Ze względu na stochastyczny charakter ML niezwykle trudno jest przewidzieć, jak daleko zaprowadzi nas dany zbiór danych lub model.

Z tego powodu *bardzo ważne* jest, aby rozpoczynać od najprostszego modelu spełniającego wymagania, zbudować kompletny prototyp wykorzystujący ten model i ocenić jego wydajność nie tylko pod kątem wskaźników optymalizacyjnych, lecz także przeznaczenia produktu.

Prosty początkowy proces

W ogromniej większości przypadków, aby podjąć decyzję, co robić dalej, najlepiej jest przyjrzeć się wydajności prostego modelu przetwarzającego początkowy zbiór danych. Podejście to należy powtarzać w każdym opisanym niżej kroku, wprowadzając niewielkie, przyrostowe, łatwe do zweryfikowania ulepszenia. Nie należy próbować budować od razu idealnego modelu.

Należy więc zbudować proces, który będzie przetwarzał dane wejściowe i generował wyniki. W większości problemów ML trzeba rozważyć dwa osobne procesy.

Trening

Aby model dostarczał dokładnych prognoz, należy go przede wszystkim przetrenować.

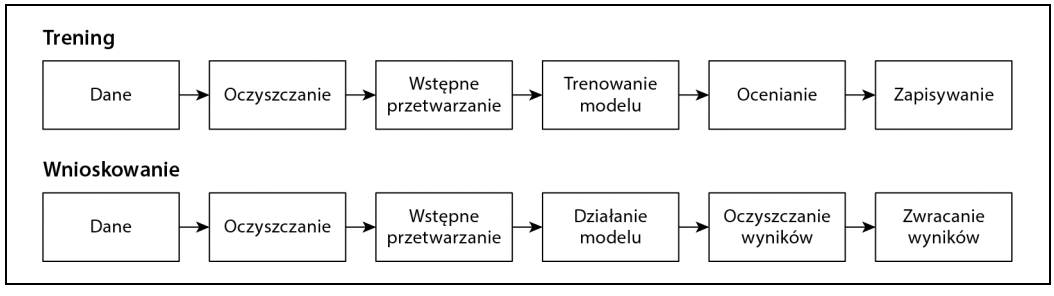
W procesie treningu pozyskiwane są wszystkie dane treningowe opatrzone etykietami (czasami zbiór może być tak duży, że pojedynczy komputer nie będzie w stanie go pomieścić) i przekazywane do modelu. Następnie model jest trenowany do momentu, aż osiągnie satysfakcjonującą wydajność. Najczęściej w tym procesie jest trenowanych kilka modeli, których wydajność jest porównywana na podstawie weryfikacyjnego zbioru danych.

Wnioskowanie

Wnioskowanie jest procesem produkcyjnym, w którym przetrenowany model jest udostępniany użytkownikom.

Ogólnie rzecz ujmując, proces wnioskowania rozpoczyna się od przyjęcia danych wejściowych i ich wstępnego przetworzenia. Faza wstępnego przetwarzania zazwyczaj dzieli się na kilka etapów. Najczęściej jest to oczyszczanie danych, ich weryfikacja, wyodrębnianie cech wymaganych przez model i formatowanie do postaci numerycznej. W przypadku bardziej złożonych systemów, proces często obejmuje również pozyskiwanie dodatkowych informacji wymaganych przez model, na przykład zapisanych w bazie danych cech użytkowników. Następnie do modelu są przekazywane przykładowe dane, przetwarzane są dane wyjściowe i zwracane uzyskane wyniki.

Rysunek 2.4. przedstawia diagram typowego procesu wnioskowania i treningu. W idealnym przypadku etapy oczyszczania i wstępnego przetwarzania danych są takie same w procesach treningu i wnioskowania. Daje to gwarancję, że przetrenowany model podczas wnioskowania otrzymuje dane o takim samym formacie i charakterystyce.



Rysunek 2.4. Uzupełniające się procesy trenowania modelu i wnioskowania

Procesy dla różnych modeli buduje się z uwzględnieniem różnych problemów, ale wysokopoziomowa infrastruktura pozostaje niezmienna. Dlatego warto tworzyć całościowe procesy treningowe i wnioskujące, aby można było szybko oszacować wpływ słabych punktów, o których wspomniała Monica Rogati w rozdziale 1. w podrozdziale „Jak wybierać projekty ML i określać ich priorytety — Monica Rogati".

Wysokopoziomowa infrastruktura większości procesów jest podobna, jednak z powodu różnic w strukturach zbiorów danych same funkcje często nie mają ze sobą nic wspólnego. Zilustruję to na przykładzie procesu dla edytora.

Proces dla edytora ML

Oba procesy edytora, treningowy i wnioskujący, zbudujemy, wykorzystując popularny w dziedzinie ML język Python. Zadaniem prototypu będzie zbudowanie całościowego procesu bez nadmiernego przywiązywania wagi do jego jakości.

Tak jak w każdej pracy, która wymaga czasu, możemy i *będziemy* przeglądać poszczególne części procesu i je usprawniać. Napiszemy standardowy proces treningowy, szeroko stosowany w różnych problemach ML, realizujący kilka funkcji:

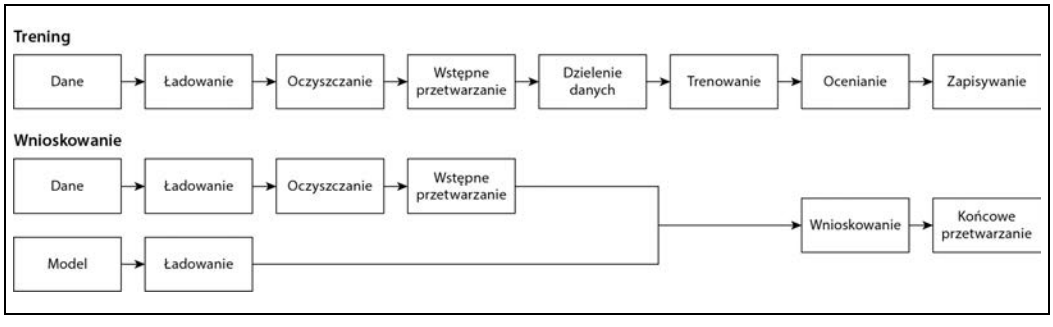
- Ładowanie rekordów danych.
- Oczyszczanie danych z niepełnych rekordów i uzupełnianie brakujących wartości w razie potrzeby.
- Wstępne przetwarzanie i formatowanie danych w sposób zrozumiały dla modelu.

- Usuwanie części danych, które nie będą wykorzystywane do trenowania modelu, tylko do weryfikowania wyników (zbiór weryfikacyjny).
- Trenowanie modelu na zadanym zbiorze danych, przygotowanie przetrenowanego modelu i statystyk podsumowujących.

W procesie wnioskowania wykorzystamy kilka funkcji z procesu treningowego, a także napiszemy kilka niestandardowych. W idealnym przypadku powinny to być funkcje realizujące następujące operacje:

- Ładowanie przetrenowanego modelu i umieszczanie do w pamięci (aby szybciej dostarczał wyniki).
- Wstępne przetwarzanie danych (jak w procesie treningowym).
- Zbieranie wszelkich potrzebnych zewnętrznych informacji.
- Umieszczanie pojedynczych przykładów w modelu (funkcja wnioskująca).
- Końcowe przetwarzanie danych w celu oczyszczenia wyników przed przekazaniem ich użytkownikowi.

Najłatwiej jest zwizualizować proces w postaci diagramu przepływów, jak na rysunku 2.5.



Rysunek 2.5. Procesy edytora

Dodatkowo napiszemy kilka funkcji analitycznych i badawczych, ułatwiających diagnozowanie problemów, na przykład:

- Funkcję wizualizującą przykłady danych najlepiej i najgorzej przetwarzanych przez model.
- Funkcje badające dane.
- Funkcję badającą wyniki modelu.

Wiele procesów zawiera etapy weryfikacji danych wejściowych i sprawdzania danych końcowych. Tego rodzaju operacje ułatwiają diagnostykę, o czym przekonasz się w rozdziale 10., i gwarantują jakość aplikacji, ponieważ wychwytują złe wyniki i zapobiegają pokazywaniu ich użytkownikowi.

Pamiętaj, że jeżeli jest stosowane ML, wyniki modelu uzyskane na podstawie nieznanych mu wcześniej danych często są nieprzewidywalne, a przez to niesatysfakcjonujące. Dlatego ważna jest świadomość, że modele nie zawsze działają dobrze. Systemy należy projektować tak, aby radziły sobie z tego rodzaju błędami.

Podsumowanie

Dowiedziałeś się, jak definiuje się najważniejsze wskaźniki umożliwiające porównywanie całkowicie różnych modeli i określanie związanych z nimi kompromisów. Poznałeś zasoby i metody przyspieszające budowanie kilku pierwszych procesów. Wiesz już, co jest potrzebne do zbudowania każdego procesu, aby uzyskać pierwsze wyniki.

Teraz powinieneś mieć pomysł sformułowany w postaci problemu, sposobu mierzenia postępów i wstępnego planu. Czas zająć się implementacją.

W części II dowiesz się, jak zbudować pierwszy proces oraz jak zbadać i zwizualizować początkowy zbiór danych.

Dobry pomysł — to zaledwie początek. Najważniejsze dzieje się później!

Uczenie maszynowe jest coraz popularniejsze. Stosuje się je w systemach wsparcia, systemach rekomendacyjnych, tłumaczeniach tekstów i wielu innych aplikacjach. Jednak podczas tworzenia tego rodzaju produktów inżynierowie napotykają bardzo poważne problemy. Jeśli ich nie rozwiążą, nawet obiecujący projekt może upaść. Trudność polega na tym, że zastosowanie uczenia maszynowego w konkretnej, użytkowej aplikacji jest złożonym zadaniem. Konieczne są wybór właściwej implementacji danej funkcjonalności, analiza błędów modelu, rozwiązanie problemów z czystością danych, a także weryfikacja wyników gwarantująca odpowiednią jakość produktu.

To książka przeznaczona dla programistów i menedżerów, którzy wśród rodzących się idei uczenia maszynowego wciąż poszukują rozwiązań dla swojego biznesu. Autor omawia krok po kroku proces tworzenia i wdrażania aplikacji opartej na uczeniu maszynowym, a praktyczne koncepcje przedstawia za pomocą przykładowych kodów, rysunków i wywiadów z liderami w tej dziedzinie. Podpowiada, jak planować aplikację i oceniać jej jakość. Wyjaśnia także, jak budować skuteczny model, i demonstruje metody jego systematycznego usprawniania, aż do momentu osiągnięcia celu. W końcowej części opisuje strategie wdrażania i monitorowania modelu. W odróżnieniu od innych pozycji poświęconych uczeniu maszynowemu ten przewodnik skupia się przede wszystkim na definiowaniu problemów, diagnozowaniu modeli i ich wdrażaniu.

Dzięki tej książce:

- łatwiej określisz, do czego produkt ma służyć
- trafnie zdefiniujesz problem uczenia maszynowego
- szybko zbudujesz kompletny proces i pozyskasz początkowy zbiór danych
- zbudujesz, wytrenujesz i zoptymalizujesz model
- wdrożysz model w środowisku produkcyjnym
- przyjmiesz najlepszą metodę monitorowania pracy modelu

Emmanuel Ameisen jest inżynierem uczenia maszynowego w Stripe. Wcześniej implementował i wdrażał rozwiązania do analiz predykcyjnych i uczenia maszynowego w Local Motion oraz Zipcar. Prowadził program sztucznej inteligencji w Insight Data Science. Zrealizował ponad sto projektów wykorzystujących uczenie maszynowe.

ISBN 978-83-283-7186-6