

Zabawa
ze Sztuczną
Inteligencją

Krzysztof Wołk



KRZYSZTOF WOŁK

Zabawa ze Sztuczną Inteligencją

Wersja Demonstracyjna



Wydawnictwo Psychoskok
Konin 2018

Krzysztof Wołk
„Zabawa ze sztuczną inteligencją”

Copyright © by Krzysztof Wołk, 2018
Copyright © by Wydawnictwo Psychoskok Sp. z o.o. 2018

Wszelkie prawa zastrzeżone.
Żadna część niniejszej publikacji nie
może być reprodukowana, powielana i udostępniana w
jakiegokolwiek formie bez pisemnej zgody wydawcy.

Projekt okładki: Krzysztof Wołk
Korekta: Bogusław Jusiak
Skład epub, mobi, pdf:
Adam Brychcy,
Kamil Skitek

Krzysztof Wołk - www.wolk.pl

ISBN: 978-83-8119-372-6

Wydawnictwo Psychoskok Sp. z o.o.
ul. Spółdzielców 3, pok. 325, 62-510 Konin
tel. (63) 242 02 02, kom. 695-943-706
<http://www.psychoskok.pl/>
<http://wydawnictwo.psychoskok.pl/>
e-mail: wydawnictwo@psychoskok.pl

Spis treści

- [1. Wstęp](#)
 - [1.1. Czym jest uczenie maszynowe?](#)
 - [1.1.1. Dwa typy uczenia maszynowego](#)
 - [1.1.2. No więc napiszmy wreszcie ten program!](#)
 - [1.2. Czy nauczanie maszynowe to czarna skrzynka?](#)
 - [1.3. Gdzie znaleźć dodatkowe informacje o nauczaniu maszynowym:](#)
 - [1.3.1. Uczenie maszynowe za darmo dla każdego! Google Colab](#)
 - [1.3.1.1. Konfiguracja Google Colab do pracy z Google Drive i Fast.ai](#)
- [2. Generatywne użycie uczenia maszynowego](#)
 - [2.1. Dokonywanie trafnych zgadywań](#)
 - [2.1.1. Nadanie sieci neuronowej funkcji pamięci](#)
 - [2.1.2. Do czego nadaje się tylko jedna litera?](#)
 - [2.1.3. Tworzenie całego opowiadania](#)
 - [2.1.4. Tworzenie Mario bez konieczności jego tworzenia](#)
 - [2.1.5. Nauczanie naszego Modelu](#)
 - [2.2. Zabawki vs. realne zastosowania](#)
- [3. Rozpoznawanie obiektów](#)
 - [3.1. Rozpoznawanie obiektów z użyciem głębokiego uczenia maszynowego.](#)
 - [3.1.1. Ograniczone pole widzenia:](#)
 - [3.1.2. Prymitywny Pomysł #1: Wyszukiwanie za pomocą przesuwającego się okna](#)
 - [3.1.3. Prymitywny pomysł #2: Zwiększenie ilości danych wejściowych oraz zastosowanie głębokiej sieci neuronowej.](#)
 - [3.1.4. Rozwiązaniem jest konwolucja.](#)
 - [3.1.4.1. Zasada działania konwolucji](#)
 - [3.1.4.2. Tworzenie odpowiedniej sieci](#)
 - [3.1.4.3. Tworzenie aplikacji rozpoznającej ptaki:](#)
 - [3.1.5. Testowanie naszej sieci neuronowej](#)
 - [3.2. Klasyfikacja – rozróżnianie od siebie zawartości obrazków \(Klasyfikacja obrazu za pomocą Sieci Konwolucyjnych\)](#)
 - [3.2.1. Wybór wskaźnika uczenia się](#)
 - [3.2.2. Poprawa naszego modelu](#)
 - [3.2.3. Tuning i różnicowanie szybkości uczenia różnicowego](#)
 - [3.2.4. Analizowanie wyników](#)
 - [3.2.5. Wyjaśnienie kodu](#)

- [3.2.6. Analizowanie wyników: utrata i dokładność](#)
- [3.2.7. Klasyfikator światowej klasy](#)
- [3.2.8. Zaczniemy od poprawnego przygotowania danych treningowych](#)
- [3.2.9. Interaktywne trenowanie modelu 10 klas](#)
- [3.2.10. Trening \(i\) modelu](#)
- [3.2.11. Predykcja na zestawie walidacyjnym](#)
- [3.3. Analiza wyników – tablica pomyłek](#)
- [4. Współczesne rozpoznawanie rysów twarzy przy użyciu głębokiego uczenia maszynowego](#)
- [4.1. Posługiwanie się Uczeniem Maszynowym w rozwiązaniu bardzo skomplikowanego problemu](#)
- [4.2. Najbardziej wiarygodne sposoby otrzymania wymiarów twarzy.](#)
- [5. Tłumaczenie maszynowe](#)
- [5.1. Myślenie w prawdopodobieństwie – tłumaczenie statystyczne](#)
- [5.2. Rekurencyjne sieci neuronowe w tłumaczeniu](#)
- [5.3. Zaczniemy nasze tłumaczenie!](#)
- [5.4. Tworzenie własnego sekwencyjnego systemu tłumaczeniowego:](#)
- [5.5. Własny tłumacz neuronowy na Google Colab](#)
- [5.5.1. Tłumaczenie typu Sequence 2 Sequence](#)
- [5.5.1.1. Model Sequence 2 Sequence](#)
- [5.5.1.2. Mechanizm Atencji](#)
- [5.5.1.3. Wymagania](#)
- [5.5.2. Ładowanie plików danych](#)
- [5.5.3. Budowanie modeli](#)
- [5.5.3.1. Dekoder Atencji - Interpretacja modelu Bahdanau](#)
- [5.5.3.2. Trenowanie](#)
- [5.5.3.3. Ewaluacja sieci](#)
- [5.6. Ćwiczenia](#)
- [6. Jak stworzyć moduł rozpoznawania głosu z użyciem głębokiego nauczania maszynowego \(ASR\).](#)
- [6.1. Przekształcenie dźwięków w bity.](#)
- [6.2. Rozpoznawanie liter z krótkich fragmentów dźwięku](#)
- [6.3. Czy mogę stworzyć mój własny system rozpoznawania mowy?](#)
- [7. Generowanie treści i zasobów graficznych przez AI](#)
- [7.1. Cele Modeli Generatywnych](#)
- [7.3. Jak działają DCGANs](#)
- [7.4. Zastosowanie w grach wideo](#)
- [7.5. Generowanie sztuki](#)
- [7.5.1. Generowanie sztuki za pomocą RNN i PyTorch w oparciu o litery.](#)

[7.5.2. Budowanie modelu](#)

[7.5.3. Ćwiczenia](#)

[7.5.4. Inferencja w generatorze RNN przy użyciu GPU lub CPU](#)

[8. Hackowanie z AI – czyli jak oszukać sieć](#)

[8.1. Sieci neuronowe jako strażnicy ochrony](#)

[8.2. Co możemy zrobić z tak zahakowanym zdjęciem?](#)

[9. Inferencja \(wnioskowanie\)](#)

1. Wstęp

Czy słyszałeś kiedyś ludzi rozmawiających o uczeniu maszynowym lub widziałeś reklamę kolejnego inteligentnego urządzenia (np. iPhone X, Huawei P20, Mate 10, LG ThinQ, LG V30, telewizory LG), ale masz tylko blade pojęcie o znaczeniu tego terminu i zastanawiasz się, jak coś, co wymagało dopiero superkomputerów, nagle działa w elektronice użytkowej? Czy męczą Cię potakiwanie głową i udawanie, że wiesz o co chodzi i co ten termin oznacza? Czas to zmienić!

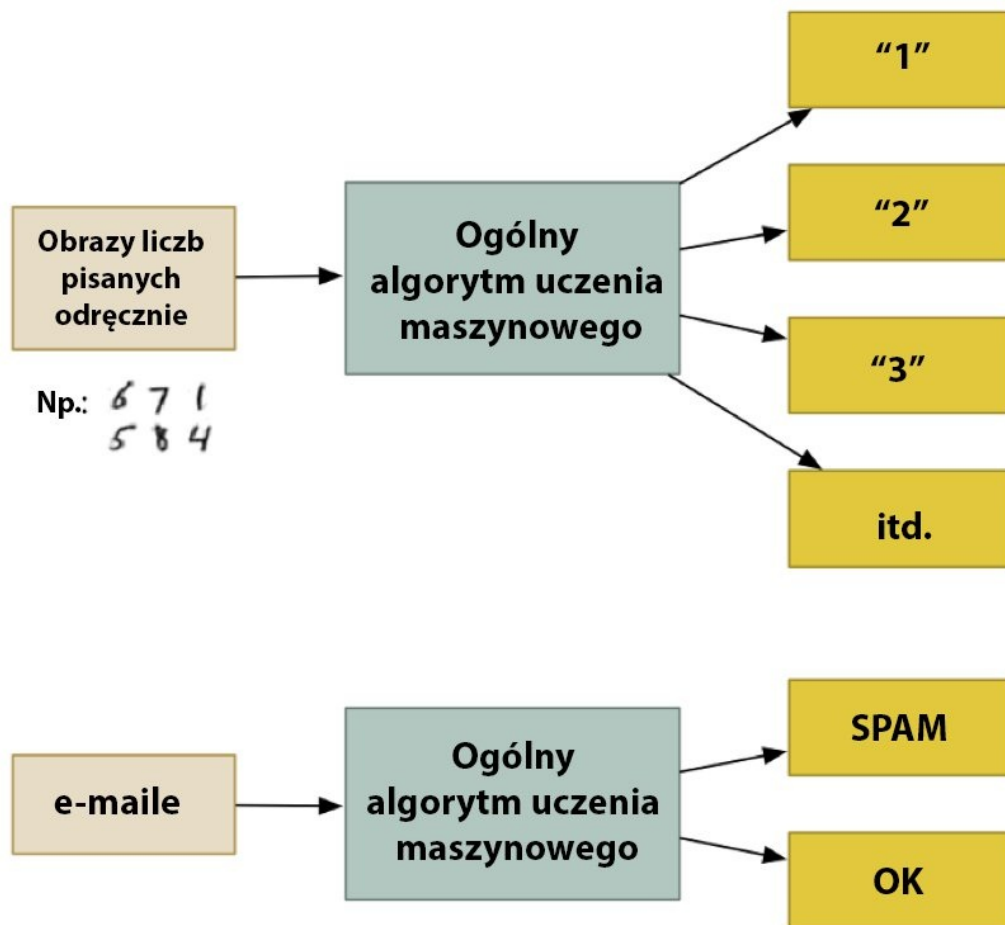
Ten podręcznik napisany jest dla każdego, kogo interesuje uczenie maszynowe, ale nie wie, od czego zacząć. Podejrzewam, że wiele osób po przeczytaniu artykułu na stronie Wikipedii poddało się z poczuciem frustracji, życząc sobie, aby ktoś dostarczyłby wyjaśnienia na wyższym poziomie. Ten poradnik spełnia więc taką funkcję.

Moim celem jest stworzenie w pełni dostępnej informacji dla każdego, stąd używam różnych uogólnień z pominięciem wielu szczegółów. Mam nadzieję, że nikt nie będzie mi brał tego za złe. Jeśli takie ukazanie tego zagadnienia zainteresuje kogoś uczeniem maszynowym, to uważam, że osiągnąłem mój cel.

1.1. Czym jest uczenie maszynowe?

Uczenie maszynowe bazuje na koncepcji istniejących algorytmów podstawowych, które ukazują ciekawą charakterystykę danego zbioru danych, ale bez konieczności pisania kodu komputerowego specyficznego dla zadanego zadania. W miejsce pisania kodu, dostarczasz danych wejściowych od algorytmu podstawowego i tworzysz swoją własną logikę, która wykorzystuje te dane.

Jako przykładem możemy posłużyć się algorytmem klasyfikacji. Grupuje on dane w różne grupy. Bez zmiany ani jednej linijki kodu ten sam algorytm klasyfikacji może zostać użyty do rozpoznawania ręcznie napisanych cyfr czy selekcji wiadomości e-mailowych do folderu ze spamem lub nie. Wystarczy dostarczyć innych danych uczących, a algorytm dostarczy inną logikę grupowania danego zbioru danych.



Ten algorytm nauczania maszynowego jest formą czarnej skrzynki (algorytmu typu black box), która może być używana dla różnych problemów klasyfikacji.

Powyższy algorytm nauczania maszynowego jest formą czarnej skrzynki z możliwym wielokrotnym wykorzystaniem do różnorodnych zadań klasyfikacji danych. "Uczenie maszynowe" czy "Nauczanie maszynowe" jest terminem bardzo szerokim i obejmuje wiele różnorodnych algorytmów podstawowych o takich możliwościach.

1.1.1. Dwa typy uczenia maszynowego

Algorytmy uczenia maszynowego rozpatrujemy w dwóch kategoriach: metody nauczania z nadzorem i bez nadzoru. Jest między nimi prosta, ale bardzo istotna różnica.

A) Nauczanie z nadzorem

Przypuśćmy, że jesteś agentem nieruchomości. Twój biznes rośnie, więc zatrudniasz grupę stażystów do pomocy. Ale masz jeden problem, bo dla Ciebie wystarczy rzucić okiem i masz dobre pojęcie, ile dana nieruchomość kosztuje, tymczasem Twoi pomocnicy z powodu braku doświadczenia nie potrafią dokonać trafnej wyceny nieruchomości.

Decydujesz się więc napisać program, który pomoże stażystom w wycenie wartości domu w Twojej okolicy (a i tobie wyjechać wcześniej na wakacje). W tym programie będziesz brał pod uwagę ceny podobnych nieruchomości oraz cechy takie jak: wielkość czy dzielnicę, w której się ten dom znajduje.

Aby tego dokonać dokonujesz zbioru danych przez trzy miesiące rejestrując wszystkie informacje o sprzedaży domów w swoim mieście. Przy każdej sprzedaży notujesz liczbę pokoi, metraż, opis dzielnicy oraz co najważniejsze cenę, za którą nieruchomość została sprzedana:

Sypialnie	Metry kwadratowe	Sąsiedztwo	Cena sprzedaży
3	185.80	Miasto	1 000 000 zł
2	74.32	Duże miasto	1 200 000 zł
2	78.96	Miasto	550 000 zł
1	51.09	Miasto	300 000 zł
4	185.80	Wieś	550 000 zł

To są dane uczące do "treningu"/uczenia

Zebrane dane będą służyć do napisania programu, który przybliży szacunkowo ceny innych nieruchomości w Twojej dzielnicy.

Sypialnie	Metry kwadratowe	Sąsiedztwo	Cena sprzedaży
3	185.80	Miasto	???

Dane te będą użyte do uczenia przewidywania ceny innych domów. To nauczanie nazywa się uczeniem z nadzorem.

Polega ono na tym, że znasz odpowiedź końcową: cenę. Program pozwolił Ci wpisanie znanych danych domów razem z ceną i poznałeś logikę wycen domów przy zebranych danych wejściowych.

Do zbudowania programu musisz wpisać dane uczące do algorytmu uczenia maszynowego. Algorytm ten próbuje oszacować jakie operacje matematyczne muszą być wykonane, aby otrzymać poprawne wyniki. Przypomina to trochę test matematyczny z wyczyszczonymi symbolami arytmetycznymi w działaniach, do których posiadasz klucz odpowiedzi:

Quiz Matematyczny #1 - Klucz odpowiedzi

1) 2 4 5 = 3

2) 5 2 8 = 2

3) 2 2 1 = 3

4) 4 2 2 = 6

5) 6 2 2 = 10

6) 3 1 1 = 2

7) 5 3 4 = 11

8) 1 8 1 = 7

Oj, nie! Cwany uczeń wymazał symbole z nauczycielskiego klucza z odpowiedziami!

Czy można więc zgadnąć jakie typy problemów matematycznych były na teście? Domyślasz się, że powinienesz "coś zrobić" z liczbami po lewej stronie, aby uzyskać odpowiedź po prawej stronie.

W nauczaniu z nadzorem, pozwalasz więc komputerowi odgadnąć te relacje. Jak tylko dowiesz się jakie działania matematyczne były potrzebne do rozwiązania tego specyficznego rodzaju zadań, możesz rozwiązywać podobne zadania!

B) Nauczanie bez nadzoru

Powróćmy do oryginalnego przykładu z agentem nieruchomości. Pomyślmy co byłoby, gdyby nie znał on ceny końcowej każdego sprzedanego domu? Jedyne dane, którymi dysponujesz jako agent są metraż, położenie itd. owe algorytmy mogą dokonać na prawdę interesujących operacji. To nazywamy uczeniem maszynowym bez nadzoru.

Sypialnie	Metry kwadratowe	Sąsiedztwo
3	185.80	Miasto
2	74.32	Duże miasto
2	78.96	Miasto
1	51.09	Miasto
4	185.80	Wieś

Uczenie maszynowe umożliwia dokonanie ciekawych rzeczy nawet, gdy nie będziesz próbował znaleźć nieznaney liczby - w tym przypadku ceny.

Przypomina to sytuację, gdzie dostajesz od kogoś zapisane na kartce papieru liczby i wręczając je mówi: "Nie mam pojęcia co

te liczby oznaczają, ale może Ty mi pomożesz zgadnąć, czy jest tutaj jakiś wzorzec, albo zbiór danych. Powodzenia!"

Co możesz zrobić z takimi danymi? Na początku możesz mieć algorytm, który automatycznie rozszyfruje z Twoich danych różne segmenty rynku. Może dowiesz się, że kupujący domy w okolicach miejscowego uniwersytetu wybierają małe domy z większą ilością pokoi a z kolei mieszkańcy przedmieść preferują domy z trzema sypialniami o dużym metrażu (czyli co najmniej 4 pokojowe)? Znajomość różnych segmentów rynku reprezentowanych przez różnych klientów pomoże ukierunkować Twoje działania marketingowe.

Innym ciekawym użyciem algorytmu byłaby automatyczne wykrywanie domów o niezwykłych cechach, które szczególnie odstają od reszty. Być może dowiesz się, że te wyjątkowe domy to ogromne rezydencje. Możesz więc skoncentrować się na nich wysyłając swoich najlepszych pomocników, którzy będą w stanie uzyskać wyższe dochody?

Poniżej do końca tego artykułu skupimy się na uczeniu maszynowym z nadzorcą i bynajmniej nie dlatego, że uczenie nienadzorowane jest w jakimś stopniu mniej użyteczne czy interesujące. Wręcz przeciwnie. Znaczenie tego rodzaju uczenia maszynowego rośnie wraz z doskonaleniem algorytmów, ponieważ umożliwia one przetwarzanie danych bez konieczności oznaczania danych z prawidłowymi wynikami.

Na marginesie: Mimo istnienia wielu innych typów algorytmów nauczania maszynowego omówione do tej pory powinny nam na początek wystarczyć.

To wspaniale, ale czy przewidywanie ceny sprzedawanego

domu na prawdę uważa się za "uczenie" (maszynowe przyp. tłum)?

Człowiek posiada mózg, który w zetknięciu się z większością sytuacji jest w stanie przystosować się do niej bez konieczności posiadania dokładnych instrukcji.

Sprzedając nieruchomości przez jakiś czas człowiek wyrobi sobie instynktownie przeczucie i będzie miał "nosa" co do odpowiedniej ceny na danym rynku, najlepszej strategii marketingowej, rodzaj klienta, którego taki dom zainteresuje itd. Celem badań nad sztuczną inteligencją (Artificial Intelligence) jest możliwość replikacji takiego zachowania.

Na bieżącą chwilę istniejące obecnie algorytmy nauczania maszynowego nie są jeszcze na tyle dobre, aby być tak wielostronne jak ludzki mózg. Działają one dobrze skupiając się na bardzo dobrze na ograniczonym, skupionym zadaniu. Czyli w naszym przypadku może zamiast definiowania "nauczanie" będzie raczej rozwiązanie równań, aby rozwiązać dane zadanie opierające się na przykładach danych".

Niestety "Maszynowe rozpoznawanie jakim równaniem rozwiązuje się określone zadanie opierając się na pewnych przykładowych danych i wynikach" to raczej niefortunna nazwa stąd w skrócie określono to "nauczaniem maszynowym" czy "uczeniem maszynowym".

Oczywiście, gdy będziesz to czytał za 50 lat i algorytmy do silnej sztucznej inteligencji będą już opracowane, to ten cały poradnik wyda się raczej przestarzały. Przyszły człowieku, przestaniesz go z pewnością czytać i zwrócisz się do swojego służącego- robota, aby zrobił ci kanapkę.

1.1.2. No więc napiszmy wreszcie ten program!

No więc jak napisałbyś program, aby szacować wycenę nieruchomości tak jak w naszych powyższych przykładach?

Postaraj się pomyśleć zanim zaczniesz czytać dalej.

Jeśli nie wiesz nic o uczeniu maszynowym, prawdopodobnie próbujesz napisać pewne proste reguły, pomagające w oszacowaniu ceny domu:

```
def oszacuj_cene_domu(liczba_sypialni, m2, okolica):

    cena = 0
    # W mojej okolicy, średnia cena domu to 200PLN na m2
    cena_na_m2 = 200
    if okolica == "hipsterska":
        # ale w niektórych rejonach cena jest trochę większa
        cena_na_m2 = 400
    elif neighborhood == "familoki":
        # a w niektórych cena jest mniejsza
        cena_na_m2 = 100
    # rozpoczynamy z ceną bazową i szacujemy w oparciu
    i wielkość w m2
    cena = cena_na_m2 * m2
    # teraz zmieniamy cenę w zależności od ilości pokoi
```

```
if liczba_sypialni == 0:
    # apartamenty typu studio są naprawdę tanie
    cena = cena — 20000
else:
    # miejsca z większą ilością sypialni są z reguły
    # bardziej wartościowe
    cena = cena + (liczba_sypialni * 1000)
return cena
```

Jeśli będziesz pracował z tym programem przez wiele godzin, możesz stworzyć coś, co mniej więcej działa. Ale Twój program nigdy nie będzie idealny i ciągle będzie miał trudności z nadążeniem za ciągle zmieniającymi się cenami na rynku.

Czy nie byłoby lepiej, gdyby komputer umiał dla Ciebie wymyślić jak wykorzystać tę funkcję? Kogo obchodzi dokładnie działanie funkcji, jeśli wykona ona swoje zadanie, czyli oszacuje poprawny wynik?

```
def oszacuj_cene_domu(liczba_sypialni, m2, okolica):

    cena = <komputerze, wykonaj całą matematykę za mnie>
    return cena
```

Jednym sposobem myślenia o tym zadaniu jest porównanie ceny domu do pysznej potrawy jednogarnkowej, gdzie sypialnie, metraż oraz dzielnica to przepyszne składniki. Jeśli jesteś w stanie oszacować jak mocno każdy składnik wpływa na

końcową cenę, może istnieją współczynniki proporcji miary poszczególnych składników, których to wymieszanie zaowocuje określoną ceną końcową?

To zredukowałoby oryginalną funkcję (dla tych, którzy szaleją ze if-ami i else-ami w swoich programach) do czegoś tak prostego:

```
def oszacuj_cene_domu(liczba_sypialni, m2, okolica):
```

```
    cena = 0
```

```
    # mała część tego
```

```
    cena += liczba_sypialni * .841231951398213
```

```
    # większa część tego
```

```
    cena += m2 * 1231.1231231
```

```
    # może trochę tego
```

```
    cena += okolica * 2.3242341421
```

```
    # i szczypta soli na koniec
```

```
    cena += 201.23432095
```

```
    return cena
```

Zwróć uwagę na pogrubione magiczne numery — .841231951398213, 1231.1231231, 2.3242341421 i 201.23432095. To są nasze wagi. Jeśli byśmy odszyfrowali idealne wagi, które działają dla każdej nieruchomości, nasza funkcja mogłaby przewidywać ceny domów!

Trywialny sposób, żeby znaleźć wagi to wykonać taką procedurę:

Krok 1:

Zacznij ze wszystkimi wagami ustawionymi na 1.0:

```
def oszacuj_cene_domu(liczba_sypialni, m2, okolica):
```

```
    cena = 0
```

```
    # mała część tego
```

```
    cena += liczba_sypialni * 1.0
```

```
    # większa część tego
```

```
    cena += m2 * 1.0
```

```
    # może trochę tego
```

```
    cena += okolica * 1.0
```

```
    # i szczypta soli na koniec
```

```
    cena += 1.0
```

```
    return cena
```

Krok 2:

Wykonaj funkcję dla każdego domu, dla którego posiadasz dane i zobacz jak bardzo funkcja myli się w zgadywaniu właściwej ceny:

Sypialnie	Metry kwadratowe	Sąsiedztwo	Cena sprzedaży	Przewidywania
3	185.80	Miasto	1 000 000 zł	800 000 zł
2	74.32	Duże miasto	1 200 000 zł	1 400 000 zł
2	78.96	Miasto	550 000 zł	520 000 zł
1	51.09	Miasto	300 000 zł	360 000 zł
4	185.80	Wieś	550 000 zł	470 000 zł

Użyj funkcji do oszacowania ceny każdego z domów.

Na przykład, jeśli pierwszy został sprzedany za 250000 PLN, ale Twoja funkcja zgadła 178 000 PLN, wtedy pomyłka wynosi 72 000 PLN dla tego jednego domu.

Teraz wysumuj kwadrat odchylenia od właściwej ceny dla każdego z wprowadzonych przez Ciebie domów. Powiedzmy, że miałeś 500 sprzedaży domów w swoim zbiorze danych i kwadrat odchylenia od ceny wzorcowej dla każdego z domów wynosił w sumie 82 123 373 PLN. To jest wskaźnik jak "nieprawidłowo" Twoja funkcja szacuje ceny nieruchomości.

Teraz weźmy końcową sumę wszystkich sprzedaży i podzielmy ją przez 500, aby uzyskać średnią tego jaki błąd został popełniony dla indywidualnego domu. Nazwijmy to średnim błędem ceny naszej funkcji.

Jeśli potrafisz zmniejszyć ten średni błąd do 0 modyfikując wagi, Twoja funkcja będzie idealna. To by znaczyło, że w każdym przypadku, Twoja funkcja idealnie szacuje cenę domu w oparciu o dane wejściowe. W takim razie, osiągnęliśmy nasz cel—obniżyliśmy średni błąd tak jak to tylko możliwe próbując różnych wag.

Krok 3:

Powtarzaj krok 2 w kółko wybierając każdą możliwą

kombinację wag. Jeśli jakakolwiek kombinacja wag przybliży średni błąd do 0—staje się ona kolejną bieżącą kombinacją wag. Kiedy znajdziesz kombinację wag, które działają, rozwiązałeś problem!

Czas na prawdziwe wyzwanie

To proste prawda? Pomyślmy czego do tej pory dokonaliśmy. Wzięliśmy pewne dane, wpisałeś je w trzy proste etapy, w efekcie których otrzymałeś funkcję, która oszacuje cenę domu w Twoich okolicach. Strzeż się Zillow (popularna strona internetowa o nieruchomościach w Stanach Zjednoczonych, szacująca ceny domów przyp. tłum).

Oto kilka faktów, które naprawdę Ciebie zadziwią:

Badania w wielu dziedzinach (lingwistyka/tłumaczenia) przez ostatnie 40 lat pokazały, że te podstawowe algorytmy uczenia maszynowego są w stanie "wymieszać naszą potrawę" (to pojęcie właśnie wymyśliłem) wykonując swoje zadania lepiej od ludzi tam, gdzie ludzie próbują odnaleźć jawne i bezpośrednie zasady samodzielnie. Ten "bezmyślny" sposób nauczania maszynowego pokonuje w tym zadaniu ludzi i ekspertów.

Funkcja końcowa jest naprawdę bezmyślna. Nie jest ona nawet w stanie wiedzieć co to są "metry kwadratowe" ani czym jest liczba sypialni. Jediną wiedzą owej funkcji jest to, że trzeba wszystko wymieszać, aby otrzymać poprawną odpowiedź.

Jest bardzo prawdopodobne, że nie będziesz miał pojęcia, dlaczego konkretna kombinacja wag w danej funkcji działa poprawnie. No ale właśnie napisałeś funkcję, której tak naprawdę nie rozumiesz, możesz jednak pokazać, że działa ona całkiem poprawnie.

Wyobraź sobie, że zamiast selekcjonować parametry takie jak

"m2", albo "liczba_sypialni", Twoja funkcja pobierać będzie to zbiór liczb. Powiedzmy, że każda liczba reprezentuje jasność pojedynczego piksela w zdjęciu zrobionym aparatem umieszczonym na dachu Twojego samochodu. Teraz powiedzmy, że zamiast wytworzyć odpowiedź na pytanie o "cenę", funkcja będzie szacować wielkość nazwaną "ilość_stopni_do_skrętu_kierownicą". Właśnie stworzyliśmy funkcję, która samodzielnie potrafi sterować naszym samochodem!

Trochę to zwariowane, nieprawdaż?

O co chodzi z tym "każdą możliwą kombinację wag" w kroku 3.?

Tak naprawdę nie jesteś w stanie spróbować każdej kombinacji i wszystkich możliwych wag, aby znaleźć tą, która będzie pracować najlepiej. To by dosłownie trwało wiecznie, bo liczby mogą być wypróbowywane w nieskończoność.

Aby tego uniknąć, matematycy wynaleźli wiele sprytnych sposobów, aby szybko znaleźć dobre wartości tych wag bez potrzeby próbowania zbyt wielu. Oto jeden ze sposobów:

Najpierw, napisz proste równanie, które reprezentuje powyższy krok 2.:

$$Koszt = \frac{\sum_{i=1}^{500} (Przewidywania(i) - PrawdziwaOdpowiedz(i))^2}{500 \cdot 2}$$

To jest Twoja funkcja kosztu.

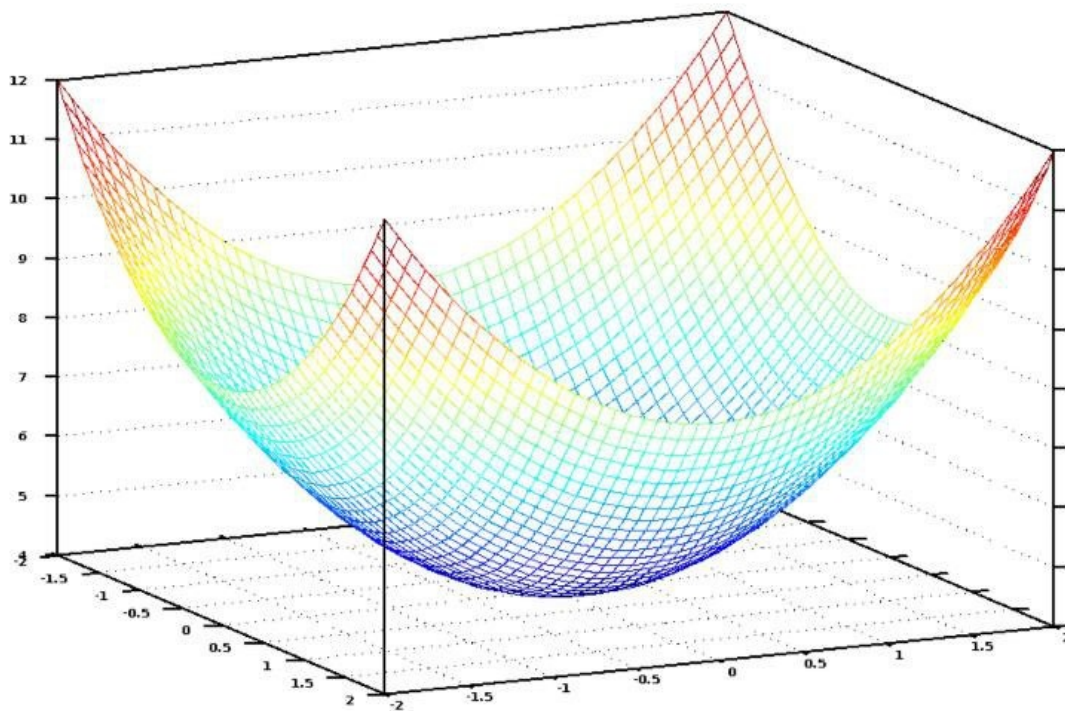
Teraz przepiszmy dokładnie to samo równanie, ale używając żargonu matematycznego uczenia maszynowego (na razie możesz to zignorować):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

θ reprezentuje Twoje bieżące wagi. $J(\theta)$ oznacza 'koszt dla Twoich aktualnych wag'.

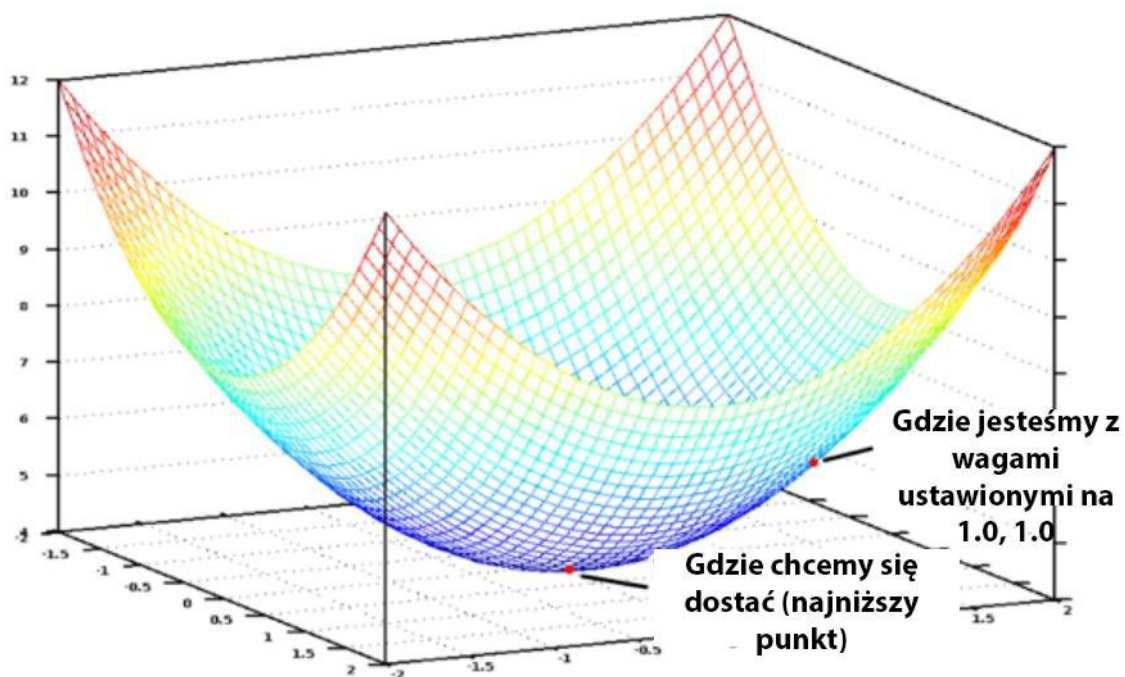
To równanie reprezentuje jak złe jest nasze szacowanie dla aktualnego zbioru wag.

Jeśli narysujemy tę funkcję kosztu dla wszystkich możliwych wartości wag dla liczba_sypialni i m2, dostalibyśmy wykres wyglądający mniej więcej tak:



Ten wykres wyglądałby jak miska. Oś pionowa oznaczona jest jako funkcja kosztu.

W tym wykresie, najniższy punkt zaznaczony na niebiesko jest tym, dla którego nasz koszt jest najmniejszy—dlatego nasza funkcja myli się tu najmniej. Najwyższy punkt jest tam, gdzie mylimy się najbardziej. W takim razie, jeśli możemy znaleźć wagi, które przenoszą nas w punkt najniższy, mamy naszą odpowiedź!



W takim razie musimy jedynie lekko zmodyfikować nasze wagi tak, abyśmy "szli w kierunku doliny" na tym wykresie, czyli w kierunku niższych wartości. Jeśli będziemy robić małe zmiany wag, które zawsze przenoszą nas niżej, dojdziemy tam bez potrzeby wykonywania zbyt wielu kroków.

Jeśli uczyłeś się rachunku różniczkowego, wiesz, że pochodna z funkcji mówi o tym jak jest ona nachylona w danym punkcie.

Parafrazując, mówi nam ona, w którym kierunku iść, aby

obniżyć wartość funkcji dla danego punktu z naszego wykresu. I tą wiedzę możemy użyć, aby zmierzać w kierunku naszej doliny.

Dlatego, jeśli policzymy pochodną cząstkową naszej funkcji kosztu względem każdej z naszych wag, wtedy możemy odjąć tą wartość od każdej wagi. W ten sposób wykonamy spacer o jeden krok bliżej w kierunku doliny, czyli minimum naszej funkcji. Wykonując to kilka razy w końcu osiągniemy dno, a tam będziemy mieli kombinację najlepszych wag dla naszego problemu. (Jeśli to dla Ciebie nie brzmi jeszcze zbyt zrozumiałe nie przejmuj się i czytaj dalej).

To jest podsumowanie jednego ze sposobów znalezienia najlepszej kombinacji wag dla Twojej funkcji, metoda ta nazywa się metodą gradientów prostych. Nie obawiaj się w to zagłębiać w tym kierunku, jeśli interesują Ciebie szczegóły.

Mimo, że gdy używasz bibliotek do nauczania maszynowego, aby rozwiązać prawdziwy problem wszystko to zostanie za Ciebie zrobione warto jest wiedzieć co dzieje się "za kulisami".

Co jeszcze kryją za sobą metody nauczania maszynowego a co ominęliśmy?

Trójstopniowy algorytm opisany wyżej jest nazywany analizą wieloczynnikową regresji liniowej. Szacowane jest równanie liniowe, które dopasowywane jest tak aby prosta przechodziła przez wszystkie punkty z danymi. Następnie używasz tego równania, aby odgadnąć ceny sprzedaży domów, których nigdy nie widziałeś przedtem opierając się na tym gdzie dom ten pojawi się na Twojej prostej. To równanie stanowi potężny instrument i możesz rozwiązać "prawdziwe" problemy z jej użyciem.

Niestety, podejście jakie pokazałem może być użyteczne w prostych zagadnieniach, ale nie będzie dobrze działać we wszystkich problemach. Powodem jest to, że ceny to skomplikowane zależności i nie zawsze ułożą się na linii ciągłej.

Na szczęście jest wiele sposobów, aby poradzić sobie z bardziej skomplikowanymi problemami. Istnieje wiele innych algorytmów nauczania maszynowego, które potrafią sobie poradzić z danymi nieliniowymi (np. sieci neuronowe lub metoda wektorów nośnych z tzw. kernelami). Istnieją też sposoby na użycie regresji liniowej w sprytny sposób tak aby przybliżyć bardziej skomplikowane funkcje do naszych danych. We wszystkich przypadkach, nadal celem jest znalezienie najlepszego zestawu wag.

Również, zignorowałem tu problem przeuczenia sieci. To dość proste, aby ukończyć algorytm z wagami, które zawsze pracują poprawnie dla nieruchomości z Twojego zbioru danych uczących, ale nigdy nie pracują poprawnie dla żadnych nowych domów, które nie były w oryginalnym zbiorze uczącym. Są sposoby na to aby temu zapobiec (regularyzacja i użycie testów walidacyjnych). Umiejętność radzenia sobie z tym problemem jest kluczowe w sprawnym posługiwaniu się narzędziami nauczania maszynowego.

Parafrazując, pomimo że podstawowa idea jest bardzo prosta, zabiera to trochę czasu i doświadczenia, aby zastosować nauczanie maszynowe i uzyskać jakiegokolwiek użyteczne rezultaty. To umiejętność, której każdy programista może się nauczyć!

*Dziękujemy za skorzystanie z oferty naszego wydawnictwa
i życzymy miło spędzonych chwil przy kolejnych naszych publikacjach.*

Wydawnictwo Psychoskok



Koniec Wersji Demonstracyjnej