Wiem czego chcesz

Systemy rekomendacji produktów

Współcześnie trudno jest znaleźć w sieci, film, książkę, piosenkę – produkt, który będzie się nam podobał. W artykule opiszę dwa algorytmy stosowane najczęściej w serwisach internetowych, takich jak np. Amazon.com lub YouTube.com, do polecania produktów. Systemy rekomendacji pozwalają użytkownikowi znaleźć szybciej interesujący go produkt, a właścicielom serwisów osiągać większe zyski.

Dowiesz się:

- Co to jest system rekomendacji produktów?
- Jak działa system do polecania produktów?
- Jak zaimplementować system polecania produktów na podstawie podobieństwa pomiędzy użytkownikami lub produktami?

Powinieneś wiedzieć:

- Podstawy programowania w dowolnym języku.

W dzisiejszych czasach internetowe serwisy oferują nam ogromne ilości różnych produktów. Znalezienie ciekawej książki, fajnego filmu lub piosenki, która się nam spodoba może okazać się niełatwym zadaniem, na które trzeba będzie poświęcić sporo czasu. Wiele serwisów internetowych takich jak np. Amazon.com czy YouTube.com oferuje swoim użytkownikom listy polecanych produktów. W artykule tym opiszę dwa najpopularniejsze i najłatwiejsze algorytmy stosowane w systemach rekomendacji produktów. Ich implementacja oraz używanie pozwoli zadowolić użytkowników, którym będą podsuwane ciekawe produkty oraz właścicieli, którzy będą osiągać większe zyski.

Systemy rekomendacji

Systemy rekomendacji (SR), nazywane też ang. *collaborative filtering* (CF), bazując na zebranych danych historycznych potrafią zasugerować z dużym prawdopodobieństwem jakie produkty mogą podobać się zalogowanemu użytkownikowi (ang. *active user*). Bazują one na obserwacji, iż użytkownicy którzy kupili podobne produkty mają zbliżone gusta, co za tym idzie w przyszłości będą dokonywać podobnych zakupów. Algorytmy stosowane w systemach rekomendacji można podzielić na trzy grupy. Pierwsza z nich, to metody modelowe (ang. *model based*), które opierają się na budowaniu modeli użytkownika i produktu w postaci wektora wag. Wagi te opisują wzajemne zależności pomiędzy bytami występującymi w systemie np. wagi użytkownika opisują jego preferencje do określonych gatunków filmów natomiast wagi filmu opisują przynależność do rodzaju filmu. Do metod tych należą między innymi metody wykorzystujące sieci neuronowe, rozkład na wartości osobliwe (SVD) lub różne techniki klasteryzacji. Drugie podejście wykorzystywane w SR to metody pamięciowe (ang. *memory based*). Metody te przy predykcji polecanego produktu używają całej historycznej informacji zapisanej w systemie. Trzecia grupa metod, to metody hybrydowe które wykorzystują połączenie metod z modelem i metod pamięciowych. W artykule przedstawię dwa algorytmy należące do grupy drugiej. Oba algorytmy jako wynik działania zwracają listę N produktów, które będą interesować najbardziej użytkownika – takie algorytmy nazywają się ang. top-N recommendation.

Reprezentacja danych

W systemie przechowywane są dane opisujące preferencje użytkowników do produktów. Zakładamy, że

Tablica 1. *Przykładowa macierz R z oddanymi głosami – wiersze opisują użytkowników, a kolumny oznaczją produkty w systemie. Wartości w macierzy to oddane głosy.*

	P_1	P_2	P_3	P_4	P_5
U1	-	3	7	8	8
U2	2	3	-	7	9
U3	8	9	3	5	2
U4	9	-	-	3	3
U5	1	4	9	7	?

użytkownik może wyrazić swoją opinię o produkcie głosując na niego. Załóżmy również, iż głosy są liczbami naturalnymi z zakresu *[0, 10]*, gdzie *0* oznacza "nienawidzę tego produktu", a *10* - "uwielbiam ten produkt". Oddane głosy możemy przedstawić jako macierz *R* - (tablicę dwuwymiarową). Wiersze macierzy będą opisywać numer użytkownika, a kolumny będą przedstawić numer produktu. Wartość w macierzy będzie określać oddany głos. Przyjmijmy również, że macierz ma *M* wierszy (liczba użytkowników), oraz *L* kolumn (liczba produktów). Oznaczmy użytkowników indeksami *p, q* natomiast produkty jako *i, j*. Zmienna R_{pi} przedstawiać będzie notę wystawioną produktowi numer *i* przez użytkownika o indeksie *p*. W tablicy nr 1 przedstawiono przykładową macierz oddanych głosów. Dla przykładu, z tablicy nr 1 widzimy, że R_{11}='-' co znaczy, że użytkownik *U1* nie głosował na produkt P_1 i np. że $R_{34}=5$ co znaczy że użytkownik U_3 zagłosował na produkt P_4, wystawiając mu notę o wyskości *5*. Natomiast R_{55}='?' znaczy, że chcemy wyznaczyć przewidywaną notę jaką odda użytkownik U_5 na produkt P_5.

Warto zwrócić uwagę, że użytkownik nie musi zagłosować na każdy produkt w systemie. W rzeczywistych systemach rekomendacji macierz *R* jest w około 99% pusta. Trafność proponowanych produktów w dużej mierze zależy od tego, na ile produktów użytkownik zagłosował. Im więcej użytkownik oddał głosów, tym trafniejsze okażą się zaproponowane później produkty.

W artykule opiszę dwa podejścia w wyznaczaniu przewidywanej liczby głosów. Pierwsze podejście bazuje na podobieństwie pomiędzy użytkownikami (ang. *user based*). Polega ono na znalezieniu w systemie użytkowników, którzy podobnie głosowali w przeszłości. Opiera się na obserwacji, iż skoro użytkownicy głosowali podobnie w przeszłości to ich głosy w przyszłości powinny również być podobne. Drugie podejście, opiera się na wyszukaniu podobnych do siebie produktów (ang. *item based*). I tak, jak w pierwszym podejściu, zakłada, że produkty które były oceniane podobnie w przeszłości otrzymają przybliżone noty w przyszłości.

Algorytm bazujący na podobieństwie użytkowników

Do oceny podobieństwa pomiędzy dwoma użytkownikami potrzebna jest metryka odległości między nimi. Dwa najczęściej stosowane to odległość cosinusowa i korelacja Pearsona. Odległość cosinusowa zdefiniowana jest następująco:

$$d(U_p, U_q) = \frac{\sum_{i=1}^{i=L} R_{pi} \cdot R_{qi}}{\sqrt{\sum_{i=1}^{i=L} R_{pi}^2 \cdot \sum_{i=1}^{i=L} R_{qi}^2}}$$

[1]

Natomiast odległość wykorzystująca korelację Pearsona podana jest poniżej:

$$d(U_p, U_q) = \frac{\sum_{i=1}^{i=L} (R_{pi} - r_p) \cdot (R_{qi} - r_q)}{\sqrt{\sum_{i=1}^{i=L} (R_{pi} - r_p)^2 \cdot \sum_{i=1}^{i=L} (R_{qi} - r_q)^2}}$$

[2]

Odległość Pearsona jest trochę bardziej skomplikowana od odległości cosinusowej, i do jej obliczenia potrzebna jest znajomość średniej głosów oddanej przez użytkownika o indeksie *p* – r_p i średniej głosów użytkownika *q* - r_q. To, którą definicję metryki podobieństwa wybrać najlepiej sprawdzić empirycznie. Wiedząc, jak obliczyć podobieństwo pomiędzy użytkownikami, szukamy *K* najbliższych użytkowników do aktywnego użytkownika U_a (zalogowanego). Zbiór *K* najbliższych użytkowników oznaczmy jako $K(U_a)$. Następnie dla każdego produktu, na który nie zagłosował użytkownik U_a, przewidujemy ocenę na podstawie równania:

$$\hat{R}_{aj} = r_j + \frac{\sum_{q=K(U_a)} d(U_a, U_q) \cdot R_{qj}}{\sum_{q=K(U_a)} d(U_a, U_q)}$$

[3]

Gdzie r_j oznacza średnią ocenę jaką dostał produkt *j*. Wzór [3] można interpretować jako średnią ważoną głosów oddanych przez *K* sąsiadów. Następnym krokiem w algorytmie jest wybranie *N* produktów, które dostały najwyższe noty. Wskazanych *N* produktów jest wynikiem działania algorytmu. Wartość *K* najbliższych sąsiadów potrzebna w tym algorytmie wyznaczana jest eksperymentalnie. Sprawdzając różne wartości *K* wybieramy taką, dla której wartość błędu predykcji na zbiorze testowym będzie najmniejsza. Typowe wartości dla *K* zawierają się w przedziale *[30, 50]*.

Podsumowując kroki algorytmu, można zapisać następująco:

1. Wyznacz odległość między aktywnym użytkownikiem, a wszystkimi użytkownikami w systemie.
2. Wybierz *K* użytkowników najbliższych do aktywnego użytkownika.
3. Korzystając ze zbioru *K* najbliższych użytkowników wyznacz przewidywaną ocenę produktów na które nie zagłosował aktywny użytkownik.
4. Wybierz *N* produktów z najwyższymi przewidywanymi ocenami.

Algorytm bazujący na podobieństwie produktów

Algorytm bazujący na podobieństwie produktów jest bardzo podobny do wcześniejszego algorytmu, z tą różnicą, że szukane są najbardziej pdobne produkty. Odległość cosinusowa w tym podejściu zdefiniowana jest następująco:

$$d(P_i, P_j) = \frac{\sum_{q=1}^{q=M} R_{qi} \cdot R_{qj}}{\sqrt{\sum_{q=1}^{q=M} R_{qi}^2 \cdot \sum_{q=1}^{q=M} R_{qj}^2}}$$

[4]

Natomiast odległość wykorzystująca korelację Pearsona podana jest poniżej:

$$d(P_i, P_j) = \frac{\sum_{q=1}^{q=M} (R_{qi} - r_i) \cdot (R_{qj} - r_j)}{\sqrt{\sum_{q=1}^{q=M} (R_{qi} - r_i)^2 \cdot \sum_{q=1}^{q=M} (R_{qj} - r_j)^2}}$$

[5]

Zbiór *K* najbliższych produktów do produktu P_a oznaczmy jako $K(P_a)$. Przewidywana ocena produktu obliczna jest na podstawie wzoru [6].

$$\hat{R}_{qa} = r_j + \frac{\sum_{i=K(P_a)} d(P_a, P_i) \cdot R_{qi}}{\sum_{i=K(P_a)} d(P_a, P_i)}$$

[6]

Podobnie jak poprzednio po wyznaczeniu przewidywanych ocen dla produktów na które nie zagłosował zalogowany użytkownik wybierana jest lista *N* produktów o największych estymowanych ocenach.

Problemy w systemch rekomendacji

W rzeczywistych systemach rekomendacji często można spotkać się z następującymi problemami:

- duża liczba użytkowników i produktów, przez co do przechowywania ich potrzebna jest duża pojemność pamięci RAM;
- użytkownicy głosują tylko na małą część wszystkich produktów (ang. *sparsity problem*), a jakość przewidywanej oceny zależy od liczby oddanych głosów w systemie;

- problem nowego użytkownika lub produktu (ang. *cold star problem*) – wprowadzając do systemu nowy byt, nie mamy żadnej informacji o jego głosach;
- problem czarnej owcy (ang. *black sheep problem*) – użytkownik oddaje specjalnie głosy inaczej niż większość ludzi;
- problem sprawiedliwego oceniania – wiele użytkowników ma tendencje do dawania tylko minimalnej lub maksymalnej oceny.

Mając zaimplementowany jako bazowy algorytm top-N recommendation możemy dodawać różne modyfikacje do naszego systemu by móc radzić sobie z tymi problemami.

Podsumowanie

W artykule zostały przedstawione dwa podejścia wykorzystujące algorytm najbliższych sąsiadów do rekomendacji *N* produktów. Opisane metody są intuicyjne, stosunkowo proste w implementacji i dają dobre wyniki. Stosowanie systemów rekomendacji pozwala dostać dwie pieczenie na jednym ogniu – zadowolony użytkownik, ponieważ podsuwane mu są ciekawe produkty trafiające w jego gust oraz zadowolony właściciel serwisu, gdyż potencjalnie wzrasta sprzedaż oferowanych przez niego produktów.

PIOTR PŁOŃSKI
Doktorant na Politechnice Warszawskiej, Wydziału Elektroniki i Technik Informacyjnych. Autor interesuje się sztuczną inteligencją, metodami analizy danych i bioinformatyką. Ulubionym językiem programowania jest C++.
Kontakt: pplonski86@gmail.com