Projekt 1

Aleksander Milach

26/04/2020

Opis problemu, cel działania programu

Otrzymaliśmy plik w formacie csv, zawierający trzy interesujące nas kolumny z danymi. Pierwsza kolumna zawiera jednoznaczne numery odpowiadające użytkownikom serwisu internetowego umożliwiającego ocenianie obejrzanych filmów, druga numery obejrzanych przez nich filmów, a trzecia ocenę filmu. Każdy wiersz pliku zawierał ocenę określonego w drugiej kolumnie filmu według użytkownika wymienionego w pierwszej kolumnie.

Dane z pliku rozdzieliliśmy do dwóch plików w taki sposób, żeby pierwszy z nich zawierał około 90% ocen każdego z użytkowników, a drugi pozostałe dane. Pierwszy plik będziemy nazywali plikiem treningowym, a drugi plikiem testowym.

Dla wygody obserwacje z pliku treningowego umieścimy w macierzy Z w taki sposób, że i-temu użytkownikowi przyporządkujemy i-ty wiersz, zaś j-temu filmowi przyporządkujemy j-tą kolumnę, wówczas ocena i-tego użytkownika dla j-tego filmu wynosi $Z[i, j]$, o ile jest nam znana, czyli jest w pliku treningowym. W ten sposób umieścimy wszystkie dane treningowe naszej macierzy. Dane testowe umieścimy według tej samej reguły w macierz V. Z tego powodu obie macierz będą takiego samego wymiaru $n \times d$, gdzie n to liczba użytkowników, a d to liczba filmów.

Celem programu będzie jak najlepsze oszacowanie ocen filmów z pliku testowego, korzystając z danych z pliku treningowego. Nasze oszacowanie będzie tym lepsze, im mniej oceny zaprognozowane przez program dla par złożonych z numeru użytkownika i numeru filmu z pliku testowego będą różniły się od rzeczywistych ocen (zapisanych w pliku), które zapisane są w pliku testowym. Sposobem na zmierzenie jakości oszacowania jest obliczenie RMSE, które jest określone następująco

$$RMSE = \sqrt{\frac{1}{|\mathbb{V}|} \sum_{(u,m)\in\mathbb{V}} (Z'[u, m] - V[u, m])^2}.$$

W macierzy Z, która zawiera wszystkie nasze dane i którą będziemy przekształcali celem oszacowania nieznanych nam jej wartości, wiele elementów pozostaje nieokreślonych. Może wynikać to z tego, że dany użytkownik mógł nie ocenić danego filmu lub jego ocena znajduje się w zbiorze testowym i właśnie ją chcemy odgadnąć. Z tej przyczyny należy uzupełnić nieznane elementy macierzy Z; reguła, według której będziemy dopełniali macierz Z będzie miała duży wpływ na jakość oszacowań podawanych przez program, do tego problemu wrócimy przy jego omawianiu.

Macierz Z będziemy przekształcać zgodnie z algorytmami poznanymi na wykładzie, to jest NMF, SVD1 i SVD2, korzystając z ich implementacji w Pythonie w pakiecie sklearn. Poniżej pokrótce omówię wyniki, jakie uzyskałem, korzystając z każdego z algorytmów; w pliku recom_system_291008.py umieściłem najlepszą pod względem RMSE wersję każdego z wymienionych algorytmów.

Omówienie wyników

NMF

W przypadku uzupełniania macierzy Z zerami otrzymałem RMSE równe około 8.36. Okazało się, że nie jest to wybitnie korzystny sposób uzupełnienia, ponieważ w momencie gdy uzupełniłem ją średnimi z ocen filmów, otrzymałem wynik 1.25, który jest znacznie mniejszy, natomiast dopełnienie średnimi użytkowników pozwoliło mi uzyskać jeszcze lepsze RMSE na poziomie 0.81. Manipulując zmienną n_components w funkcji NMF() dla

macierzy Z uzupełnionej zerami, najmniejsze (czyli zapisane powyżej) RMSE otrzymałem dla n_components równego 6, dla mniejszych argumentów RMSE powoli spadało, natomiast dla wiekszych rosło coraz szybciej. Inna sytuacja miała miejsca kiedy sprawdzałem wpływ tego argumentu dla macierzy uzupełnionej średnimi - w obu przypadkach RMSE najpierw powoli malało, a potem dla n_components > 20 stabilizowało się; były to bardzo małe różnice - odpowiednio wynik dla n_compoments $= 1$ różnił się od najlepszego o 0.03 (gdy liczyłem średnie dla filmów) i o 0.06 (gdy liczyłem średnie użytkowników).

SVD1

W przypadku, kiedy uzupełniałem macierz Z zerami, otrzymałem RMSE równe około 8.29. Ten wynik jest najmniejszym jaki uzyskałem, sprawdzając dla różnych wartości zmiennej n_components w funkcji TruncatedSVD(). RMSE zachowywało się analogicznie jak dla algorytmu NMF na macierzy uzupełnionej zerami, ponownie najlepszy wynik osiągałem dla n_components $= 6$.

SVD2

Tak jak wcześniej najmniejsze RMSE otrzymywałem dla n_components $= 6$. Kolejne iteracje algorytmu zmniejszają RMSE - ten spadek jest coraz wolniejszy, dla ograniczenia różnicy w RMSE na poziomie większym od 0.01 algorytm zatrzymuje się po około 50 iteracjach. Dla tego algorytmu otrzymywałem RMSE wynoszące około 1.92, w porównaniu do wywołań, w których macierz Z uzupełniona jest zerami jest to o wiele lepszy wynik, jednak wciąż gorszy od wersji NMF dla uzupełniania średnimi. W tym przypadku program podaje wynik zauważalnie później niż dla dwóch pozostałych algorytmów, z tego powodu, że jak wspomniałem wyżej trzeba kilkadziesiąt razy wykonać SVD1.

Tabela wyników najlepszych wersji poszczególnych algorytmów dla kilku wylosowań zbiorów treningowych i testowych

	1	2	3	4	5	6	7	8	9
NMF	0.842	0.826	0.832	0.820	0.816	0.802	0.821	0.839	0.811
SVD1	8.298	8.319	8.216	8.255	8.308	8.306	8.261	8.269	8.355
SVD2	1.994	1.941	1.900	1.936	1.926	1.979	1.933	1.990	1.929

Poza plikiem z właściwym programem recom_system_291008.py załączam do wglądu plik create_train_test_291008.py, przy którego pomocy losowałem swoje zbiory treningowe i testowe. Nie edytowałem go tak, żeby korzystał z argparse, bo dzielić na zbiór treningowy i testowy będzie Pan swoim kodem. Komentarze dotyczące samego kodu i jego poszczególnych elementów umieściłem w plikach .py w odpowiednich miejscach programów.