Pozytywka rekomendacje grupowe

Michał Brzeziński

Jan Hapunik

18. 01. 2025

Działanie algorytmu bazowego:

- Algorytm dla każdego użytkownika znajduje piosenki z którymi miał styczność w danym przedziale czasowym

- Na podstawie jego historii interakcji z nimi, ilość wystąpień każdej piosenki dla każdego typu interakcji mnoży się przez opisane wagi i sumuje by uzyskać ocenę piosenki dla użytkownika, którą następnie się normalizuje funkcją gausa

- Po znalezieniu ich, określona ilość najlepszych piosenek jest przekazywana dalej

- Zbieramy otrzymane najlepsze piosenki każdego użytkownika do jednej listy dodając ocenę grupową, poprzez sumowanie ocen każdego z użytkowników dla danego utworu

- Wyniki są grupowane w klastry za pomocą algorytmu K-means z biblioteki sklearn

- Dla każdej grupy, na podstawie określonych atrybutów Track, znajduje się wielowymiarowy środek ciężkości grupy i szuka się określonej liczby najbliższych jemu piosenek które do niej nie należą, z piosenek które pojawiły się do tej pory w historii użytkowników, używając podobieństwa cosinusowego

- Następnie losuje się z nich odpowiednią liczbę piosenek

- Można ją wywołać komendą

  curl -X POST -H "Content-Type: application/json" -d '[lista id użytkowników]' http://localhost:8001/worse_recommend

  By otrzymać w terminalu listę track_id

Działanie modelu zaawansowanego:

- Zamiast losować piosenki jak w ostatnim kroku modelu bazowego, wybiera się je

- dla każdego użytkownika uczy się, używając danych z sesji, decision tree które przewiduje jak użytkownik zareaguje na piosenkę

- Następnie zbierane są przewidywane reakcje na wszystkie proponowane piosenki z klastrów i wybiera się najpopularniejsze opcje

- Można ją wywołać z głównego serwisu, wpisując w przeglądarkę adres http://localhost:8000, co spowoduje wyświetlenie wersji z interfejsem użytkownika. W oknie należy wpisać listę identyfikatorów użytkowników jako tablicę, np. “[user_id1, user_id2]“."

- Lub wywołując komendę

  curl -X POST -H "Content-Type: application/json" -d '[lista id użytkowników]' http://localhost:8001/

  By otrzymać w terminalu listę track_id

Dodatkowy algorytm:

- By uniknąć niespójności, zamiast od razu robić playlistę z wybranych piosenek, są one zwracane jako lista propozycji z której można wybrać piosenki do dodania do playlisty, lub je odrzucić

- Korzystajac z takiej reakcji na piosenki, uczymy osobny algorytm LinearSVC z biblioteki sklearn, by wykrywał czy piosenka zostanie dodana czy nie na podstawie poprzednio dodanych piosenek, generując na tej podstawie nowe propozycje

- Ponieważ jest to oddzielny model, testy A/B, dotyczą wersji bazowej i zaawansowanej. Do tego modelu jest oddzielny test

- by ją wywołać należy najpierw wykorzystać model zaawansowany generowania rekomendacji. z przeglądarki należy przejść na http://localhost:8000, wygenerować rekomendację, wybrać pasujące nam piosenki i kliknąć “Submit Selected Recommendations”

Testy

- Model bazowy i zaawansowany były testowane, wykorzystując dane o sesjach użytkowników w danym okienku czasu i sprawdzając czy rekomendacje pokrywają się z historią użytkowników po danym okienku czasu

  Wykorzystano do tego funkcję test_create_recommendations() umożliwiającą wywołanie algorytmu dla wielu kombinacji parametrów modelu, jedna po drugiej, zwracająca listę wartości parametrów i jakości przewidywań na koniec

  Można ją wykorzystać do testów A/B obu modelów wywołując

  curl -X POST -H "Content-Type: application/json" -d '[lista id użytkowników]' http://localhost:8001/test_recommendations

- By określić jakie atrybuty należy użyć, wykorzystano backward selection i stwierdzono że jakość modelu wzrosła po porzuceniu kolumny “acousticness” i “valence”

  Wykorzystano do tego funkcję test_features którą można wywołać w bardzo podobny sposób

  curl -X POST -H "Content-Type: application/json" -d '[lista id użytkowników]' http://localhost:8001/test features

- by przetestować działanie UpdateGroupReccomendations wykorzystano rekordy podanych użytkowników, z danego okna czasowego. “skip” było brane jako odrzucenie, “like” jako akceptację piosenki. Następnie algorytm otrzymywał rekordy po danym oknie czasowym i musiał przewidzieć czy piosenka się spodoba czy nie, zwracając listę “accuracy”dla każdego dostanego użytkownika.

  curl -X POST -H "Content-Type: application/json" -d '[lista id użytkowników]' http://localhost:8001/test_update

Podział:

- Serwis główny “app” w folderze app na porcie 8000, odpowiedzialny za zarządzanie bazą danych i prowizorycznym ui

    - folder data z zapakowanymi plikami jsonl z danymi

    - templates z wizualizacją strony w html

    - app_routes.py z endpointami związanymi z bazą danych

    - recommendation_routes.py z endpointami kontaktującymi się z mikroserwisem

- Mikroserwis “model” na porcie 8001 odpowiedzialny za generowanie i utrzymywanie playlisty dla grupy

    - init_gen.py - plik z klasą GroupReccomendations z modelem generacji playlisty

    - active_gen.py - plik z klasą UpdateGroupReccomendations opdpowiadającą za aktualizację playlisty w czasie rzecywistym

    -tests.py zawierający funkcję do testowania jakości modeli

Wyniki

- Do sprawozdania dołączono plik “log.txt” z danymi porównującymi oba modele po wywołaniu

  curl -X POST -H "Content-Type: application/json" -d '[492, 456]' http://localhost:8001/test_recommendations

- oraz “przykładowe wyniki.xlsx” zawierający przykładowe wyniki działania różnych testów, oraz wizualizację porówania modelu rozszerzonego z bazowym.

- Jak widać z danych, model zaawansowany nie wprowadził widocznej poprawy, a zabrał zauważalnie więcej czasu

- jak pokazują arkusze “A|B” i “A|B 2” skalowanie niektórych parametrów, takich jak liczba klastrów wydaje się pomagać w stabilizacji wykresu accuracy, ale skalowanie tego liniowo nie wydaje się optymalnym rozwiązaniem. Dodatkowo wydłuża to znacząco czas oczekiwania w miarę zwiększania się liczby użytkowników