IUM 2021Z – Dokumentacja końcowa

Zadanie 2, dane 2 – system rekomendacyjny – etap 2

Autorzy:
Bartosz Świrta
Radosław Radziukiewicz

Celem niniejszego projektu było przeprowadzenie projektu, który miał zapoznać nas z procesem wytwarzania i uruchamiania modeli uczenia maszynowego.

Kontekst projektu

W ramach projektu wcielamy się w rolę analityka pracującego w firmie „eSzoppping" –sklepu internetowego z elektroniką i grami komputerowymi. Praca na tym stanowisku nie jest łatwa –zadanie dostajemy w formie enigmatycznego opisu i to do nas należy doprecyzowanie szczegółów tak, aby dało się je zrealizować. To oczywiście wymaga zrozumienia problemu, przeanalizowania danych, czasami negocjacji z szefostwem. Poza tym, oprócz przeanalizowania zagadnienia i wytrenowania modeli, musimy przygotować je do wdrożenia produkcyjnego –zakładając, że w przyszłości będą pojawiać się kolejne ich wersje, z którymi będziemy eksperymentować.

Treść zadania

"Są osoby, które wchodzą na naszą stronę i nie mogą się zdecydować, którym produktom przyjrzeć się nieco lepiej. Może dało by się im coś polecić?"

1. Zawartość repozytorium

- „data" – zawiera ostateczną wersję danych uzyskaną od prowadzącego projekt. Dane te służą bezpośrednio do budowy i testowania modeli
- „models" – zawiera wytrenowane dwa modele (w postaci plików .json), oraz skrypty .py do ich budowy i testowania
- „notebooks" – zawiera notatniki jupyter, które były używane w procesie analizy danych i projektowania modeli. Zawiera też pod folder „data", który zawiera wszystkie wersje danych jakie zostały nam dostarczone
- „preprocessors" – zawiera skrypt .py, który dokonuje obróbki danych używanych do budowy modeli
- „service" – zawiera implementację mikro-serwisu jako skrypt .py, pozwalający na uruchomienie modeli i przeprowadzenie testów A/B. Folder zawiera również skrypt .py loggera, który jest używany przy zapisywaniu logów predykcji.

2. Modele

W celu rozwiązania problemu rekomendacji wyróżnia się trzy główne typy modeli:

1. Popularity based
2. Content based
3. Collaborative filtering

Najprostszymi w koncepcji i implementacji są modele bazujące na popularności. Ze względu na to, że posługują się one jedynie pewną zagregowaną informacją o zbiorze, był to naturalny wybór na model bazowy.

Wybór architektury modelu docelowego był bardziej skomplikowany. Na początku rozważaliśmy użycie modeli opartych na zawartości, ale szybko zorientowaliśmy się, że w otrzymanych danych ciężko upatrywać rozbudowanych metadanych produktów, które są kluczowe dla modeli tej klasy. Sprawiło to, że jako model docelowy zaimplementowaliśmy model oparty na filtrowaniu kolaboracyjnym.

1. Implementacja modelu bazowego

Model oparty na popularności. Jest to model niespersonalizowany, dla każdego użytkownika zwracana jest taka sama rekomendacja. Dokonuje on oceny wszystkich produktów w sklepie zgodnie z przyjętą funkcją oceny oraz następnie sortuje wyniki malejąco względem oceny.

Funkcja oceny została oparta na funkcji używanej przez serwis IMDB i ma postać:

$$score = \frac{P}{P + Pmin} * R + \frac{Pmin}{P + Pmin} * Ravg$$

gdzie:
P – popularność produktu – liczona jako suma interakcji w których uczestniczył
Pmin – minimalna popularność produktu wymagana by pojawił się on w rankingu – ustalona na 0.8 percentyl
R – ocena użytkowników danego produktu, który jest aktualnie oceniany przez funkcję
Ravg – średnia ocena wszystkich produktów

Proces budowy modelu przebiega następująco:

1. Preprocessing – rzutowanie kategorii, pozbycie się zbędnych kolumn.
2. Wyznaczenie argumentów *P*, *Pmin*, *R* i *Ravg* dla funkcji oceny.
3. Ocena zbioru.

Wejście modelu: n/a
Wyjście modelu: posortowana malejąco lista najlepiej ocenionych produktów

Ostateczna implementacja na potrzeby serwisu znajduje się w pliku "./models/*basic.py"* – plik został zaopatrzony w stosowne komentarze ułatwiające zapoznanie się z kodem

Zapisany model znajduje się w pliku *„./models/basic/recommendations.json"*

2. Implementacja modelu docelowego

Model pamięciowy wykorzystujący filtrowanie kolaboracyjne. Ideą działania modelu jest podział użytkowników na k grup „podobnych" użytkowników. Rekomendacją będzie pokazanie użytkownikowi n najpopularniejszych produktów z obecnie przeglądanej kategorii w grupie, do której należy.

Model korzysta z macierzy interakcji – wiersze stanowią użytkownicy, kolumny produkty a komórki to liczba interakcji użytkownik-produkt zeskalowane z użyciem skalowania min-max. Tak utworzona macierz jest redukowana z użyciem metody *TruncatedSVD* z pakietu *sklearn*. Macierz powstała w wyniku redukcji ma wymiarowość 200x10 (pierwotnie 200x319) i reprezentuje ona użytkowników o zestawie 10 cech.

Taką macierz poddajemy grupowaniu na 8 grup z użyciem algorytmu *KMeans* z pakietu *sklearn*. W efekcie uzyskujemy 8 grup podobnych do siebie użytkowników. Dla każdej z grup znajdujemy najbardziej popularne produkty w każdej z pięciu głównych kategorii.

Jako model, zapisywany jest przydział użytkowników do grup i top produkty w każdej z nich. Użytkownik przeglądając produkt z kategorii A jako rekomendację dostanie najpopularniejsze produkty w danej kategorii z grupy, do której należy.

Wejście modelu: id użytkownika, obecnie przeglądana kategoria
Wyjście modelu: posortowana malejąco lista polecanych produktów

Ostateczna implementacja na potrzeby serwisu znajduje się w pliku *"./models/advanced.py"* – plik zaopatrzony w stosowne komentarze ułatwiające zapoznanie się z kodem

Zapisany model znajduje się w plikach:

- *„./models/advanced/group_recommendations.json"* – top produkty
- *„./models/advanced/user_to_group.json"* – podział użytkowników

3. Porównanie wyników

Oba modele budowane są na identycznym zbiorze treningowych. Zbiory tworzone są zgodnie z poniższym schematem.

Ze wszystkich sesji wybierz te, które mają więcej niż k akcji (np. k=8).

1. Z wybranych sesji wybierz BATCH_SIZE kolejnych akcji (następujących bezpośrednio po sobie).
2. Wybrane akcje dodaj do zbioru testowego.
3. Utwórz zbiór treningowy jako sessions - test_set

Modele są następnie tworzone na podstawie zbioru treningowego i oceniane na zbiorze testowym zgodnie z poniższym schematem:

1. Posortuj zbiór testowy względem kolumny *session_id.*
2. Dla każdej serii ciągłych akcji:
   2.1. Dla każdej akcji ***t***, wygeneruj predykcję
   2.2. Sprawdź, czy predykcja zawiera produkt, z którym użytkownik wszedł w interakcję w akcji ***t+1***
   2.3. Jeżeli tak to podnieś o jeden licznik poprawnych predykcji
3. Wyznacz finalną skuteczność modelu jako iloraz $\frac{liczba\ trafnych\ predykcji}{liczba\ wszystkich\ predykcji}$

W wyniku zastosowania powyższego algorytmu uzyskujemy następujące zbiory:

a. Zbiór treningowy o rozmiarze 102193 próbek.
b. Zbiór testowy o rozmiarze 13857 próbek.

W celu przeprowadzenia zautomatyzowanego test, należy odpalić skrypt "test.py", znajdujący się w katalog "models".

Skrypt ten wykonuje wyżej opisane akcje. Ponadto, zaopatrzony jest on w stosowne komentarze, ułatwiające śledzenie wykonywanych operacji.

Jako ostateczny wynik przeprowadzonych testów uzyskaliśmy następujące wyniki:

a. Model bazowy (popularnościowy) - dokładność na poziomie 40%.
b. Model docelowy (filtrowanie zespołowe) - dokładność na poziomie 74%.

Poniżej, znajduje się wynik uruchomienia skryptu testowego:

```
Beginning to form test set...
Finished creation of test set...
Beginning to form train set...
Finished creation of train set...
Beginning to build models...
Built basic recommender without errors...
Built advanced recommender without errors...
Basic recommender achieved 39.66 accuracy.
Advanced recommender achieved 73.60 accuracy.
```

Warto zaznaczyć, iż testy zostały przez nas wykonane wielokrotnie. Za każdym razem, uzyskane wyniki były do siebie bardzo zbliżone (różnice w dziesiątych częściach procenta).

4. Mikro-serwis

W celach umożliwienia korzystania z wytworzonych przez nas modeli, stworzony został specjalny mikro-serwis. Wykorzystuje on pakiety "*flask*" oraz "*flask-restful*". W celach uruchomienia serwisu, należy uruchomić skrypt "*service.py*", znajdujący się w folderze "service".

Serwis udostępnia jeden "end-point", o ścieżce "/". W celach dokonania predykcji, niezbędne jest wykonanie zapytania HTTP GET, wraz z następującymi parametrami:

- user_id – identyfikator użytkownika, dla którego generowana jest predykcja.
- category_path – kategoria produktu aktualnie oglądanego przez użytkownika.
- model - informacja z jakiego modelu chce się skorzystać (dozwolone wartości to "basic" oraz "advanced").

Odpowiedzią mikro-serwisu będzie plik .json. Zawierać on będzie wartości:

- user_id – znaczenie takie jak w zapytaniu.
- date – godzina otrzymania zgłoszenia
- id – identyfikator zgłoszenia
- model – znaczenia takie jak w zapytaniu.
- recommendations – tablica rekomendowanych produktów.

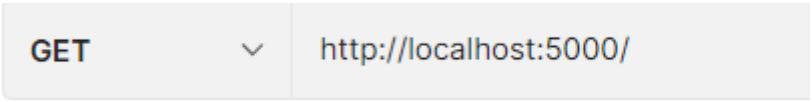
W przypadku błędnego zapytania, zwrócona wiadomość będzie posiadać kod 400 oraz zawierać będzie specjalne pole "message" informujące o przyczynie błędu.

Każde zgłoszenie do serwisu, zapisywane jest przez klasę *Logger* do pliku "*logs/logs.txt*". Zachowanie to zdecydowanie ułatwi nadzorowanie działania modelu.

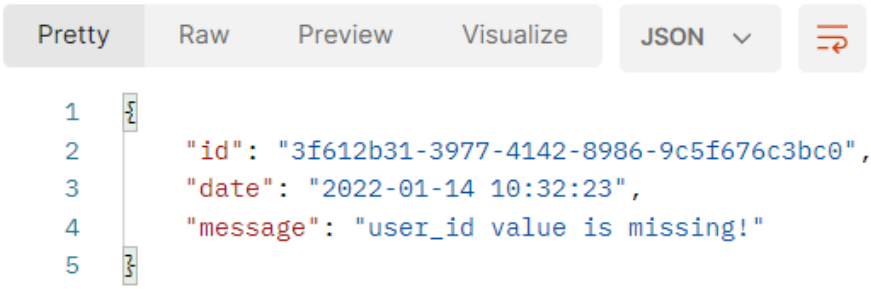
Występujące w zgłoszeniu pole "*model*" umożliwia przeprowadzanie testów A/B w dogodny sposób. W serwisie korzystającym z systemu rekomendacji wystarczy umieścić plik konfiguracyjny, informujący o tym, jakie modele będą stosowane dla jakich użytkowników.

5. Przykłady działania

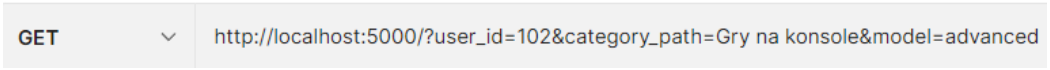
Przykładowe niepoprawne zgłoszenie do serwisu



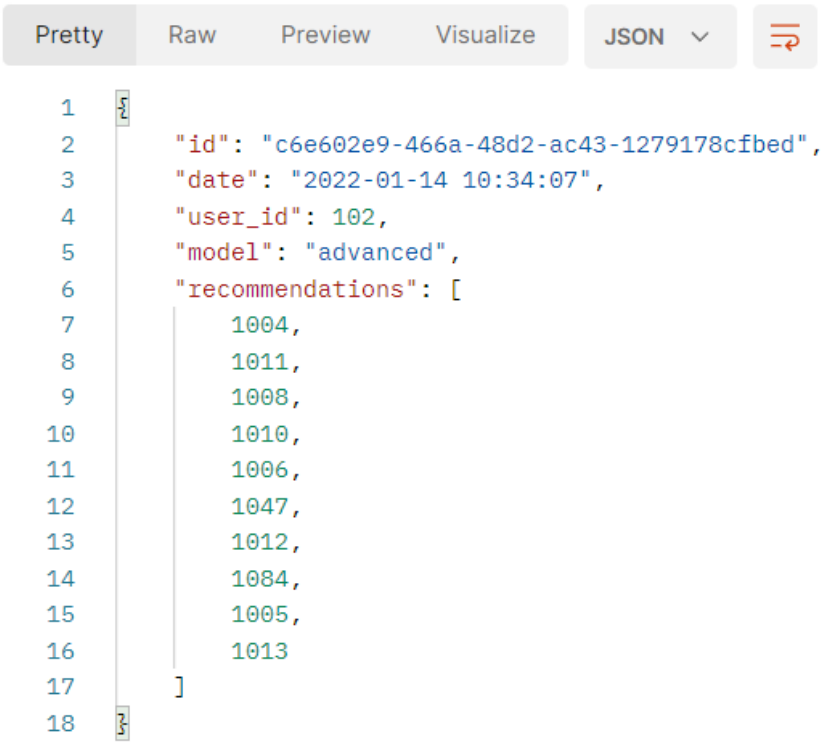
Odpowiedź



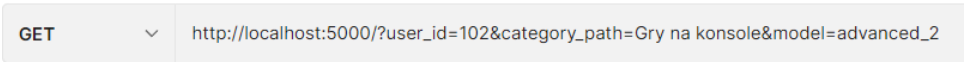
Przykładowe poprawne zgłoszenie do serwisu



Odpowiedź



Przykładowe niepoprawne zgłoszenie:



Odpowiedź