Inżynieria Uczenia Maszynowego

Zadanie dla firmy “eSzoppping”

1. Autorzy

Jakub Gałat 300209
Bartłomiej Olber 300237

2. Treść zadania

Niektórzy klienci nie mogą zdecydować się na zakup oglądając produkt na stronie – ale pewnie gdybyśmy wiedzieli, które to są osoby, to odpowiednia zniżka skłoniłaby ich do zakupu.

3. Opis implementacji:

Zgodnie z założeniami etapu drugiego od początku planowaliśmy strukturę naszej aplikacji tak, żeby była w stanie służyć klientowi jako aplikacja mikroserwisowa. W tym celu przyjęliśmy, że będzie ona serwisem webowym wystawiającym REST API. Plik main.py odpowiada za połączenie wszystkich modułów wchodzących w skład projektu w aplikację poprzez endpointy:

- “/predictA”, odpowiada za serwowanie predykcji modelem A
- “/predictB”, odpowiada za serwowanie predykcji modelem B
- “/bought”, odpowiada za rejestrowanie zakupu produktu przez klienta
- “/viewed”, odpowiada za rejestrowanie wyświetlenia produktu przez klienta

Kod źródłowy modułów znajduje się w folderze src. Architektura aplikacji została podzielona na: kontroller (plik main.py), serwisy (folder services) i repozytoria (folder repositories). Dodatkowo utworzone zostały klasy reprezentujące obiekty bazodanowe oraz modele serwujące predykcje.

Biblioteki:

- Keras, TensorFlow (bardziej zaawansowany model B)
- xgboost (model A)
- Flask (serwer REST)
- sqlite3 (baza danych)
- pandas, numpy (standardowe narzędzia do danych)
- optparse (do obsługi arg wejścia)
- pathlib (do lepszego posługiwania się ścieżkami)
- sklearn (narzędzie wspierające testowanie)

4. Opis trenowania i testowania

Modele trenujemy dzieląc posortowane chronologicznie udostępnione dane na zbiór danych, które będą początkowo w bazie danych, zbiór trenujący, zbiór walidacyjny i zbiór testowy. Na początku trenujemy model na danych trenujących, które są danymi występującymi chronologicznie przed danymi testowymi i walidacyjnymi ale już po dodaniu początkowych do bazy. Dodatkowo dzielimy zbiory treningowy i walidacyjny na 10 chunków, tak aby podczas trenowania modele mogły trenować się na większej porcji danych niż pojedyncze przypadki, dzięki czemu będzie to efektywniejsze. Po przetrenowaniu modelu na danym chunku aktualizujemy bazę danych o występujące w tej porcji danych treningowych przypadki.
Następnie możemy przetestować nasz model. Chcemy aby dla wszystkich wydarzeń zakupu nasz model przewidział zniżkę taką samą przy jakiej klient kupił albo niższą. Dla oddzielnego zbioru testowego sprawdzamy, czy przewidywania naszego modelu dla przypadków zakupu pokrywają się z rzeczywistością. W tym przypadku po każdym wydarzeniu dla, którego mamy przewidzieć zniżkę, aktualizujemy bazę danych. Testów dokonujemy lokalnie przy pomocy uruchomienia skryptu pythona, a nie serwera RESTowego.

5. Opis budowy modelu B

Projektując ten model staraliśmy się użyć podejścia bardziej zaawansowanego niż drzewa decyzyjne, jak w przypadku xgboostClassifier, a mianowicie sieci neuronowej. Na początku rozważaliśmy przez chwile zastosowanie sieci, która będzie miała tyle klas wyjściowych ile jest rodzajów zniżki, lecz uznaliśmy, że ciężko będzie ją zmotywować do wyboru najniższej zniżki. Dlatego zastosowaliśmy warstwę output, która ma jedną daną wejściową oraz funkcją aktywacji jest funkcja softmax. Zdecydowaliśmy się na trzy warstwy ukryte (50, 30, 20), używające ReLU oraz inicjalizacji he_uniform, która powinna wspomóc regularyzację. Przed każdą warstwą ukrytą dodajemy etap normalizacji oraz dropout, który również powinien zapobiegać przeuczaniu. Jako optymalizator stosujemy Nadam (clipnorm=1 w celu redukcji eksplodujących gradientów) oraz BinaryCrossentropy jako funkcję straty.

Zarówno model A jak i model B mają takie same dane wejściowe:

```
discount,

buyingProductFrequency,

buyingFrequency,

userBuyingProductFrequency,

buyingWithBiggerDiscountFrequency,

buyingWithLowerDiscountFrequency,

buyingWithDiscountFrequency,

buyingWithoutDiscountFrequency,

buyingThisProductWithBiggerDiscountFrequency,

buyingThisProductWithLowerDiscountFrequency,

buyingThisProductWithDiscountFrequency,

buyingThisProductWithoutDiscountFrequency,

userBuyingThisProductWithBiggerDiscountFrequency,

userBuyingThisProductWithLowerDiscountFrequency,

userBuyingThisProductWithDiscountFrequency,

userBuyingThisProductWithoutDiscountFrequency,

differenceBiggerLower,

differenceDiscountWithout,

differenceThisProductBiggerLower,

userDifferenceThisProductBiggerLower
```

6. Porównanie modeli

Modele porównujemy na podstawie metryk *accuracy* oraz *precision*:

ModelA: *accuracy* - 0.61, *precision* - 0.89

ModelB: *accuracy* - 0.59, *precision* - 1

Według nas model B jest lepszy ponieważ nie oferuje niepotrzebnie zniżek.

7. Opis użytkowy

Aplikację można używać w dwóch trybach:

- poprzez wywołanie “python main.py“
- poprzez wywołanie “flask run”

Opcja pierwsza umożliwia lokalne testowanie i trenowanie modeli.

W celu trenowania i odświeżenia bazy danych podajemy -t<numerModelu> (1 modelA, 2 modelB).

W celu testowania modelu podajemy -e<numerModelu>.

Opcja druga umożliwia wystawienie serwera RESTowego.

Do danych serwowanych przez endpointy można dostać się poprzez requesty HTTP POST, które zawierają argumenty **user_id** i **product_id**, będące odpowiednio id użytkownika, dla którego ma zostać zaserwowana predykcja, oraz id produktu, któremu nasz serwis ma przewidzieć zniżkę.

Klient korzystający z naszego serwisu powinien również uaktualniać bazę danych zawierającą historię zakupów klienta poprzez korzystanie z endpointów “/bought” oraz “/viewed”. Informować o zdarzeniach powinien poprzez requesty HTTP POST, zawierające dla zakupu: **user_id**, **product_id**, **discount**, będące odpowiednio id użytkownika, który dokonał zakupu, id produktu, który zakupił, oraz zniżka, która była mu wtedy oferowana, oraz dla wyświetlenia: **user_id**, **product_id**, będące odpowiednio id użytkownika, który wyświetlił produkt, i id produktu, który wyświetlił.

8. Przykład działania

Klient chce otrzymać predykcje jaką zniżkę powinien wyświetlić na stronie sklepu “eSzoppping”.

Wysyła zapytanie HTTP POST na endpoint “/predictA”, podając dane klienta oraz produktu, który dany klient wyświetla.

Serwer zwraca wysokość zniżki, która powinna zostać zaproponowana.

Przykład:

POST na “/predictA”

```
POST http://localhost:5000/predictA?user_id=130&product_id=1234

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 23
Server: Werkzeug/2.0.1 Python/3.6.9
Date: Tue, 25 May 2021 16:24:51 GMT

<p> A prediction: 0</p>

Response code: 200 (OK); Time: 75ms; Content length: 23 bytes
```

POST na “/predictB”

```
POST http://localhost:5000/predictB?user_id=130&product_id=1234

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 23
Server: Werkzeug/2.0.1 Python/3.6.9
Date: Tue, 25 May 2021 16:23:09 GMT

<p> B prediction: 0</p>

Response code: 200 (OK); Time: 244ms; Content length: 23 bytes
```