Predykcja dokonania zakupu

- Mateusz Szczepanowski
- Albert Ściseł

Budowa modelu – przygotowanie danych

Opis tensora który chcemy uzyskać

Pierwszą część stanowi tensor złożony z danych numerycznych:

Łączny spędzony w sesji czas	Zniżka oferowana w sesji	Płeć użytkownika	Ilość obejrzanych produktów w sesji	Suma cen obejrzanych produktów	Najtańszy obejrzany produkt	Najdroższy obejrzany produkt

Warto zaznaczyć, iż każda z wartości jest typu *float*, natomiast łączny spędzony czas w sesji wyrażony jest w sekundach.

Następnie wektor złożony z danych kategorycznych:

Miasto użytkownika	Dzień tygodnia, w którym sesja się rozpoczęła	Miesiąc, w którym sesja się odbyła	Odwiedzone kategorie główne

Pierwsze trzy części są zapisane jako *one hot vector*, natomiast odwiedzone kategorie są wektorem, gdzie jedynki są w miejscu odwiedzonych kategorii, więc może być ich więcej niż jedna.

Otrzymujemy też tensor wyniku sesji, czyli zakupu lub braku zakupu produktu, jest on tensorem jednostkowym zawierającym liczbę z przedziału [0;1]. Następnie przy pomocy thresholdu ustalamy czy dana wartość przyjmie wartość 1 lub 0 (zakup lub brak zakupu).

Opis kroków prowadzących do otrzymania pożądanej postaci danych

1. W pierwszym kroku dane o sesjach są ładowane z pliku *sessions.jsonl* za pomocą funkcji *get_data()* do postaci DataFrame z biblioteki pandas.
2. Wartości z kolumny ‘timestamp’ są konwertowane na typ *datetime* z biblioteki pandas, pozwoli nam to na łatwiejsze wyznaczanie późniejszych zależności.

Następnie korzystamy z funkcji *session_to_tensor(),* która ma za zadanie pojedynczą sesję użytkownika zamienić w tensor nadający się do użycia przy późniejszej nauce oraz ewaluacji modeli.

3. Załadowanie informacji o produktach z pliku *products.jsonl* oraz użytkownikach z pliku *users.jsonl*.

4. Złączenie danych z sesji z danymi o produktach na podstawie obejrzanych przez użytkownika w sesji ‘product_id’.
5. Zamiana wartości NaN w kolumnie ‘purchase_id’ na 0 oraz tych nie zawierających NaN na 1 i w następnym kroku otrzymanie informacji o wyniku sesji zapisanej w postaci jednostkowego tensora.
6. Zmiana całej ścieżki do obejrzanego produktu na kategorię główną, w której się znajduje.
7. Usunięcie zbędnych z naszego punktu widzenia kolumn: ‘event_type’ oraz ‘product_name’.
8. Złączenie danych z sesji z interesującymi nas danymi o użytkowniku, które uzyskaliśmy na podstawie analizy danych: miasta ‘city’ oraz płci ‘sex’. Złączenie na podstawie ‘user_id’, a następnie usunięcie tej kolumny.
9. Na postawie kolumny ‘timestamp’ obliczenie czasu trwania sesji przedstawionego w sekundach, zapisanego w nowej kolumnie ‘spent_time’.
10. Na podstawie kolumny ‘timestamp’ uzyskanie informacji o dniu tygodnia oraz miesiącu, w którym miała miejsce sesja. Zapis tych danych w kolumnach ‘day’ oraz ‘month’, w postaci numerycznej, dla dnia tygodnia liczba z zakresu 0-6, dla miesiąca 1-12.
11. Przepisanie do osobnej zmiennej *categorical_columns_one* informacji o mieście, dniu tygodnia oraz miesiącu. Do kolejnej zmiennej *categorical_columns_many* informacji o unikalnych odwiedzonych kategoriach głównych. Pozostałe dane numeryczne połączone w jeden wiersz dla sesji.
12. Zapisanie wiersza wartości numerycznych uzyskanego w poprzednim punkcie jako tensor liczb zmiennoprzecinkowych.
13. Pobranie z listy użytkowników listy wszystkich miast oraz z listy produktów listy wszystkich głównych kategorii.
14. Za pomocą funkcji one hot z modułu torch.nn.functional zamiana danych z *categorical_columns_one* na wektory wypełnione zerami z jedynką na miejscu wartości uzyskanych z sesji.
15. Za pomocą funkcji *categories_to_tensor* wyprodukowanie wektora z liczbą jedynek odpowiadającą liczbie odwiedzonych kategorii głównych przechowywanych w zmiennej *categorical_columns_many*.
16. Konkatenacja wektorów uzyskanych w punktach 14 i 15 w jeden tensor opisujący dane kategoryczne.

W ten sposób wprowadzając jedną sesję użytkownika składającą się z kilku wierszy uzyskujemy jeden tensor opisujący sesję nadający się do treningu oraz ewaluacji modeli (mimo, iż mamy tak naprawdę trzy tensory: opisujący dane numeryczne, dane kategoryczne, wynik sesji to tak naprawdę możemy uznać złączenie tych wektorów za jeden wektor opisujących całą sesję. To rozgraniczenie jest nam potrzebne wyłącznie dla ułatwienia dalszego przebiegu naszego programu).

Wygląd sesji przed transformacją

	session_id	timestamp	user_id	product_id	event_type	offered_discount	purchase_id
0	124	2021-02-13 05:31:27	102	1289	VIEW_PRODUCT	0	NaN
1	124	2021-02-13 05:34:39	102	1288	VIEW_PRODUCT	0	NaN
2	124	2021-02-13 05:38:20	102	1287	VIEW_PRODUCT	0	NaN
3	124	2021-02-13 05:39:02	102	1285	VIEW_PRODUCT	0	NaN
4	124	2021-02-13 05:42:43	102	1292	VIEW_PRODUCT	0	NaN
5	124	2021-02-13 05:47:03	102	1292	BUY_PRODUCT	0	20001.0

Sesja o id 124 w postaci pandas.DataFrame

Po transformacji

```
(tensor([936.0000,   0.0000,   0.0000,   5.0000, 837.9900,  82.9900, 189.0000]),
 tensor([0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]),
 tensor(1.))
```

Ta sama sesja po transformacji do postaci tensorowej

Do tworzenia modeli, w szczególności do ich treningu każda z sesji otrzymanych w danych została poddana wyżej opisanym operacjom, następnie dane te zostały zapisane w postaci dwuwymiarowych tensorów.

Za pomocą prostej funkcji numpy.random.rand() (biblioteka numpy) wyznaczającej liczby losowe z rozkładem jednostajnym U(0,1) wylosowaliśmy indeksy rozdzielające nasze dane na te przeznaczone do treningu oraz do ewaluacji naszych modeli.

Następnie korzystając z modułu torch.utils.data.Dataset stworzyliśmy dwa zbiory danych, treningowy oraz testowy, zawierające dane numeryczne, kategoryczne oraz wynik sesji.

W przypadku modelu bazującego na sieci neuronowej użyliśmy dodatkowo modułu torch.utils.data.DataLoader() pozwalającego nam na łatwe losowanie danych z utworzonych zbiorów oraz tworzenie odpowiednich rozmiarów serii danych używanych do treningu i ewaluacji.

Opis modeli

- **Naive model**

Prosty model naiwny. Na wejście przyjmuje dane numeryczne sesji. Na wyjściu produkuje 0 lub 1. Bazuje swoje przewidywania na średniej długości czasu trwania sesji w danych, na których został wytrenowany. Jeżeli czas sesji poddawanej predykcji jest większy od tej średniej zwraca 1 – przewidywanie sesji zakończonej zakupem, jeżeli czas jest mniejszy lub równy zwraca 0 – przewidywanie braku zakupu.

- **NN model**

Model bazujący na sieci neuronowej. Na wejście przyjmuje dane numeryczne oraz kategoryczne. Na wyjściu produkuje predykcje, która jest liczbą z zakresu [0, 1]. Zwracanie 0 lub 1 realizowane jest za pomocą treshold'u.

Specyfikacja sieci neuronowej:

- Warstwa gęsta działająca wraz z funkcją aktywacji (tangens hiperboliczny) jako embedding – przyjmuje na wejście dane kategoryczne, na wyjściu zwraca wektor „zanużony”.

- Dwie warstwy ukryte

- Liczba neuronów ukrytych: 40, 20

- Funkcja aktywacji dla warstw ukrytych: LeakyReLU

- Dla warstw ukrytych zastosowany Dropout o współczynniku 0.4

- Funkcja aktywacji dla warstwy wyjściowej: Sigmoid

- Threshold = 0.5

- Funkcja straty: BCEWithLogitsLoss

- Optymalizator: Adam, lr = 0.0003

Model sieci neuronowej uczony był z wykorzystaniem DataLoadera o parametrze batch_size ustalonym na 64. Uczenie modelu trwało 101 epok.

Ewaluacja modeli

Ewaluacja polegała na wydzieleniu z otrzymanych danych zbiorów treningowego oraz testowego. Zbiór testowy został ustalony na około 30% wszystkich danych. Naive model oraz NN model zostały przetrenowane na danych treningowych, gdzie trening modelu naiwnego polegał na wyznaczeniu średniego czasu trwania sesji. Wyniki dla danych testowych były następujące:

nn_model – 82.7 %

naive_model – 46.4 %

Dodatkowo w analitycznym kryterium sukcesu posługiwaliśmy się współczynnikiem alpha, który dla modelu naiwnego wynosił około 0.3. Dla modelu nn_model współczynnik ten wynosi (dla danych treningowych) około 0.65, a więc mamy dwukrotny wzrost tego współczynnika.

Mikroserwis

W ramach drugiego etapu stworzyliśmy aplikację serwującą predykcje czy dana sesja zakończy się zakupem. Przed wysłaniem requesta w formie JSON do serwisu musimy poddać dane pewnym transformacją, by uzyskać tensor (opisane było to wcześniej). Następnie tak przygotowane dane możemy przesłać do serwisu. Przykładowy wygląd requesta:


```
{
  "x_cat": [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
  "x_num": [224.0000,  20.0000,   0.0000,   1.0000, 199.9800,  99.9900,  99.9900],
  "model": "nn_model"
}
```


Gdzie w polu:

X_cat – podajemy tensor wartości kategorycznych,

X_num – podajemy tensor wartości numerycznych,

Model – podajemy, z którego modelu chcemy skorzystać. Do wyboru są „nn_model” albo „naive_model”.

Użycie mikroserwisu

Serwer ma jeden endpoint:

http://localhost:5000/predict

Który zwraca obiekt JSON z jednym lub dwoma polami. Zależne jest to od wybranego modelu. W przypadku modelu naiwnego zwracane jest tylko pole result (1 – przy zakupie, 0 – w innym przypadku), a w przypadku modelu bazującego na sieci neuronowej zwracana jest również pewność predykcji wyrażona w procentach (certainty). Przykład odpowiedzi widoczny poniżej:


```
{
  "result": 1
}
```


Wynik dla modelu naiwnego


```
{
  "certainty": 99.82,
  "result": 1
}
```


Wynik dla modelu bazującego na sieci neuronowej

Certainty wyznaczone jest na podstawie wyjścia sieci neuronowej, która jest z przedziału [0;1]. Tak więc np. dla wyjścia 0.12 model zwróci result = 0, certainty = (1-0.12)*100=88%. Funkcje realizujące zapytania do mikroserwisu zawarte są w tym samym pliku co przygotowanie i trening modeli (plik ium_etap2.ipynb). Wytrenowane modele zapisujemy kolejno do plików ‘model.pth’ oraz ‘naive.pkl’ a następnie ładujemy te pliki po stronie mikroserwisu (*load_models()*).

Logowanie

W ramach każdego zapytania zapisywane są logi w pliku results.txt. W każdej linii tego pliku zapisujemy dane w formacie JSON, w których zawarte są następujące dane:

1. Dane kategoryczne
2. Dane numeryczne
3. Jaki model zawarty był w żądaniu
4. Wynik dla modelu naive_model
5. Wynik dla modelu nn_model
6. Pewność modelu nn_model (certainty)


```
{"x_cat": [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0], "x_num": [224.0, 20.0, 0.0, 1.0, 199.98, 99.99,
99.99], "model": "naive_model", "naive_model": 0, "nn_model": 1,
"nn_model_certainty": 100.0}
```


Przykładowy log

Testy mikroserwisu

W ramach testu wykonaliśmy kilka zapytań zawierających dane z sesji. Poniżej widać wysłanie tego samego zapytania dla dwóch różnych modeli (nn_model/naive_model). **Uwaga:** Widoczne dane są tylko z pliku sessions.jsonl. Oczywiście cały request tworzony jest dodatkowo na podstawie danych z products.jsonl oraz users.jsonl.

Zapytanie I

	session_id	timestamp	user_id	product_id	event_type	offered_discount	purchase_id
97	149.0	2020-03-28 17:51:26	102.0	1278.0	VIEW_PRODUCT	10.0	0
98	149.0	2020-03-28 17:52:47	102.0	1278.0	BUY_PRODUCT	10.0	1

```
{'certainty': 99.96, 'result': 1}
```

	session_id	timestamp	user_id	product_id	event_type	offered_discount	purchase_id
97	149.0	2020-03-28 17:51:26	102.0	1278.0	VIEW_PRODUCT	10.0	0
98	149.0	2020-03-28 17:52:47	102.0	1278.0	BUY_PRODUCT	10.0	1

```
{'result': 0}
```

Zapytanie II

	session_id	timestamp	user_id	product_id	event_type	offered_discount	purchase_id
2650	856.0	2020-08-24 19:14:12	116.0	1007.0	VIEW_PRODUCT	15.0	0
2651	856.0	2020-08-24 19:14:28	116.0	1009.0	VIEW_PRODUCT	15.0	0
2652	856.0	2020-08-24 19:15:55	116.0	1005.0	VIEW_PRODUCT	15.0	0
2653	856.0	2020-08-24 19:18:45	116.0	1013.0	VIEW_PRODUCT	15.0	0
2654	856.0	2020-08-24 19:22:22	116.0	1008.0	VIEW_PRODUCT	15.0	0

```
{'certainty': 100.0, 'result': 0}
```

	session_id	timestamp	user_id	product_id	event_type	offered_discount	purchase_id
2650	856.0	2020-08-24 19:14:12	116.0	1007.0	VIEW_PRODUCT	15.0	0
2651	856.0	2020-08-24 19:14:28	116.0	1009.0	VIEW_PRODUCT	15.0	0
2652	856.0	2020-08-24 19:15:55	116.0	1005.0	VIEW_PRODUCT	15.0	0
2653	856.0	2020-08-24 19:18:45	116.0	1013.0	VIEW_PRODUCT	15.0	0
2654	856.0	2020-08-24 19:22:22	116.0	1008.0	VIEW_PRODUCT	15.0	0

```
{'result': 1}
```

Zapytanie III

	session_id	timestamp	user_id	product_id	event_type	offered_discount	purchase_id
5618	1685.0	2020-04-14 13:14:49	137.0	1047.0	VIEW_PRODUCT	5.0	0
5619	1685.0	2020-04-14 13:18:55	137.0	1046.0	VIEW_PRODUCT	5.0	0
5620	1685.0	2020-04-14 13:19:24	137.0	1041.0	VIEW_PRODUCT	5.0	0
5621	1685.0	2020-04-14 13:21:06	137.0	1042.0	VIEW_PRODUCT	5.0	0
5622	1685.0	2020-04-14 13:25:21	137.0	1084.0	VIEW_PRODUCT	5.0	0

```
{'certainty': 100.0, 'result': 0}
```

```
      session_id           timestamp  user_id  product_id    event_type  offered_discount  purchase_id
5618      1685.0 2020-04-14 13:14:49    137.0      1047.0  VIEW_PRODUCT               5.0            0
5619      1685.0 2020-04-14 13:18:55    137.0      1046.0  VIEW_PRODUCT               5.0            0
5620      1685.0 2020-04-14 13:19:24    137.0      1041.0  VIEW_PRODUCT               5.0            0
5621      1685.0 2020-04-14 13:21:06    137.0      1042.0  VIEW_PRODUCT               5.0            0
5622      1685.0 2020-04-14 13:25:21    137.0      1084.0  VIEW_PRODUCT               5.0            0

{'result': 1}
```

Dodatkowe uwagi

- W analitycznym kryterium sukcesu ustalaliśmy, że dotychczas konsultanci dokonują wyboru sesji na podstawie długości jej trwania (wybierali daną sesję jak trwa dłużej niż średnia wartość sesji). W ten sposób wyznaczaliśmy współczynnik alpha = **#poprawne_przewidzenie_zakupy / #wszystkie_sesje_zakończone_zakupem**. Jak już wcześniej pisaliśmy wynosił on około 0.3. Nowy model daje nam alpha = 0.65. Podsumowując myślę, że spełniliśmy analityczne kryterium biznesowe. Dodatkowo dokładność predykcji tego modelu jest niemalże dwa razy większa niż dotychczasowy model bazowy (82.7% do około 46%).