Inżyniera uczenia maszynowego

Projekt

Zadanie 5.1

Michał Matak, Jakub Robaczewski

Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA

14.01.2021

1 Github

Kod źródłowy projektu znajduje się pod linkiem https://github.com/Robak132/IUM-z5w1

2 Problem biznesowy i zadanie modelowania

Dostaliśmy od klienta następujące polecenie:

Mamy co prawda dodatkowe benefity dla naszych najlepszych klientów, ale może dałoby się ustalić kto potencjalnie jest skłonny wydawać u nas więcej?

2.1 Pierwsza iteracja

W pierwszej iteracji zadania zdecydowaliśmy się na próbie znalezienia informacji i budowie modeli, który by określał jakie jest prawdopodobieństwo, że klient wróci do naszego serwisu. Okazało się jednak, że dane niewiele mówią na ten temat, a kryterium sukcesu z kolei jest ściśle zależne od długości okresu, w które wpływa na kwalifikacje użytkownika. Po zebraniu danych okazało się, że w ciągu całego roku tylko 4 użytkowników nie wróciło do serwisu (w tym 2, którzy pojawili się w grudniu), co stanowi 2% wszystkich użytkowników. Na podstawie tak małej ilości danych nie udałoby się odpowiednio nauczyć modelu.

2.2 Druga iteracja

Z wyżej wymienionego względu zmieniliśmy nasz cel i potraktowaliśmy zadanie klienta bardziej literalnie. Po rozmowie z klientem ustaliliśmy, że zadowoli go model, który pozwoli na oszacowanie wydatków klienta w przyszłej jednostce czasu.

3 Mikroserwis

3.1 Uruchamianie

1. Uruchmić skrypt `microservice/main.py`
2. Wysłać żądanie POST na adres `localhost:8080/predict`, podając dane w formacie JSON lub uruchomić przykładowy POST w pliku `tests/send_ab_data.py`.
3. Wysłać żądanie POST na adres `localhost:8080/good_results`, podając słownik danych w postaci `user_id: poprawna_wartość` lub uruchomić przykładowy POST w pliku `tests/send_good_data.py`.
4. Wysłać zapytanie GET na adres `localhost:8080/predict`, żeby obejrzeć raport z testu AB.

3.2 Endpointy

POST /predict/A	uzyskanie predykcji z modelu A
POST /predict/B	uzyskanie predykcji z modelu B
POST /predict	przesłanie danych do testu AB (losowy model)
GET /predict	wygenerowanie raportu z testu AB (skuteczność modeli)
POST /predict/reset_log	usunięcie wszystkich danych z poprzednich testów AB
POST /predict/good_results	wgranie poprawnych wartości w celu ewaluacji i obliczenia skuteczności modelu
GET /predict/good_results	wyświetlenie wgranych poprawnych wartości dla użytkowników

3.3 Raport


```
{
    "model_results": {
        "2022-01-14T21:47:54": [
            {
                "model": "microservice.model.model_a.predict_expenses",
                "MSE": 181809324.247733
            },
            {
                "model": "microservice.model.model_b.predict_expenses",
                "MSE": 197982658.33597
            }
        ],
        "2022-01-14T21:48:16": [
            {
                "model": "microservice.model.model_a.predict_expenses",
                "MSE": 147252715.770525
            },
            {
                "model": "microservice.model.model_b.predict_expenses",
                "MSE": 242182553.012622
            }
        ],
        "_total": [
            {
                "model": "microservice.model.model_a.predict_expenses",
                "MSE": 164531020.00912902
            },
            {
                "model": "microservice.model.model_b.predict_expenses",
                "MSE": 220082605.674296
            }
        ]
    },
    "log": [
        {
            "timestamp": "2022-01-14T21:47:54",
            "user_id": 139,
            "model": "microservice.model.model_a.predict_expenses",
            "result": 180.71,
            "good": 23400.85,
            "error": 539174901.6195999
        },
        {
            "timestamp": "2022-01-14T21:47:54",
            "user_id": 242,
            "model": "microservice.model.model_a.predict_expenses",
            "result": 533.24,
            "good": 44677.14,
            "error": 1948683907.21
        },
        {
            "timestamp": "2022-01-14T21:47:54",
            "user_id": 143,
            "model": "microservice.model.model_a.predict_expenses",
            "result": 830.67,
            "good": 7273.05,
            "error": 41504260.0644
        },
        {
```

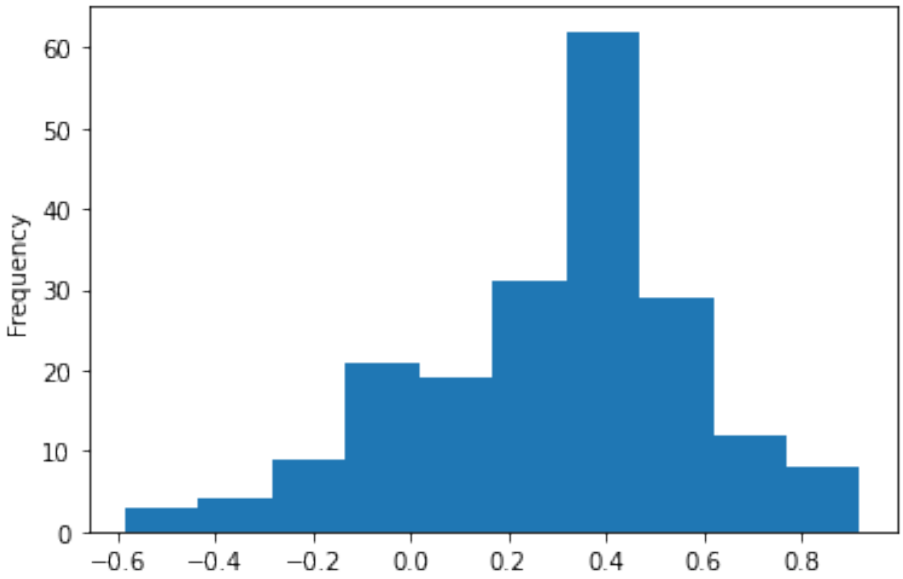

Rysunek 1: Przykładowy raport z testów AB

Raport przekazuje informacje dotyczące błędu średniokwadratowego modeli użytych w teście AB. Oprócz tego można z niego pobrać dokładne logi pokazujące jakie wartości otrzymywały modele dla poszczególnych użytkowników i jakie wartości powinny uzyskać bazując na poprawnych danych ewaluacyjnych.

4 Modele

4.1 Model A

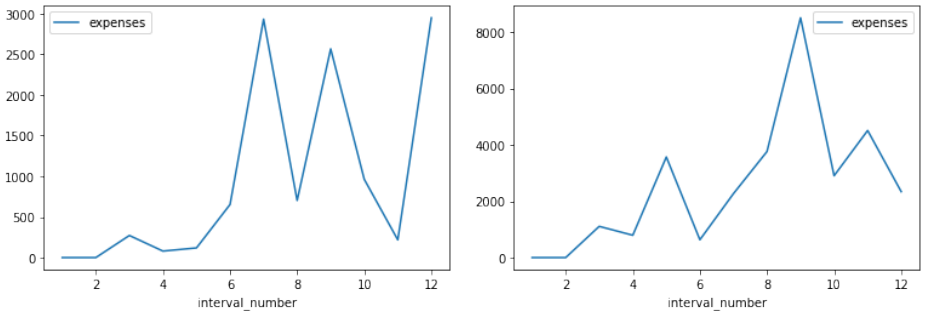
W pierwszym modelu staraliśmy się zastosować podejście polegające na tym, że analizujemy szereg czasowy wydatków użytkownika pochodzący z kolejnych miesięcy. Dla każdego z użytkowników sprawdziliśmy czy istnieje korelacja pomiędzy kolejnymi miesiącami, a jego wydatkami. Zebrane wyniki przedstawione są na poniższym wykresie:



Rysunek 2: Zebrane korelacje wydatków z upływem miesięcy dla wszystkich użytkowników

Widzimy tutaj, że jest stosunkowo mało przypadków, gdzie korelacja nie występuję dlatego zdecydowaliśmy się na to, że naszą pierwszą metodą predykcji będzie regresja liniowa.

Kod, dzięki, któremu doszliśmy do powyższych wniosków znajduje się w `notebooks/users_linear.ipynb`



(a) Wykres wydatków dla użytkownika o id równym 200, dla którego współczynnik korelacji wynosił 0.60

(b) Wykres wydatków dla użytkownika o id równym 161, dla którego współczynnik korelacji wynosił 0.62

Rysunek 3: dwa obrazki

4.2 Model B

W drugim modelu natomiast zdecydowaliśmy się na inne podejście. Na podstawie danych sesji z pewnego okresu czasu dla każdego użytkownika wyznaczyliśmy pewne jego zagregowane cechy takie jak ilość kupionych produktów, wydanych pieniędzy, średnich zniżek, okres aktywności itp. Przykład tak zbieranych i wyliczanych danych znajduje się na obrazku poniżej:

	user_id	expenses	products_bought	events_number	sessions_number	average_discount	average_discount_on_bought	city
0	103	19009.38	26	163	41	9.907975	9.423077	Warszawa
1	104	25259.70	36	205	63	11.073171	9.583333	Gdynia
2	106	12342.84	16	129	35	9.302326	8.750000	Wrocław
3	107	968.67	8	86	20	9.534884	7.500000	Radom
4	108	16312.03	23	129	32	9.418605	9.347826	Radom
...	...	...	...	...	...	...	...	...
8	297	9589.30	17	97	27	9.536082	9.117647	Kraków
9	298	11714.33	20	137	35	9.854015	9.250000	Warszawa
0	299	11498.74	19	154	47	8.084416	8.947368	Gdynia
1	300	13844.80	11	76	25	11.973684	11.363636	Radom
2	301	13363.17	10	85	24	11.352941	12.000000	Wrocław

Rysunek 4: Przykład zagregowanych danych

Powyższe dane podaliśmy następnie na wejście sieci neuronowej, na której wyjściu znajdowały się kwoty wydane przez tych użytkowników w następnym okresie czasu. Kod, dzięki, któremu doszliśmy do powyższych wniosków znajduje się w `notebooks/users_aggregation.ipynb`, a uczenie sieci odbywało się w `models/neural_networks/train.ipynb`.

4.3 Budowa modelu

Aby zapewnić sobie elastyczność we wdrażaniu modeli ich podmianie oraz aby zrównoleglić pracę zdecydowaliśmy się na stworzenie interfejsu, który określa jakie metody ma udostępniać model (`microservice/models/model_interface.ipynb`). Interfejs definiuje metody `predict_expenses`, `load_model` oraz `save_model`. Dzięki temu mogliśmy zaimplementować dwie kompletnie różne metody i użyć ich w celu predykcji. Pierwszy model korzysta z prostej implementacji regresji liniowej zaimportowanej z biblioteki `scikit`, zatem nie musi być trenowany i zwraca predykcje od razu na podstawie otrzymanego szeregu czasowego. Natomiast wagi do sieci neuronowej trzeba wprowadzić korzystając z wyżej wymienionej metody `load_model`. Jeśli chcemy zmienić architekturę sieci bez zmiany struktury zaagregowanych danych (wejście i wyjście muszą pozostać takie same) to można to łatwo zrobić podmieniając zaagregowaną klasę w konstruktorze `model/model_b.py`.

5 Wyniki

Niespodziewanie model A daje lepsze predykcje od modelu B, który jest tylko nieznacznie lepszy od modelu zwracającego stałą równą średniej wydatków w analizowanym miesiącu.