IUM 25Z DOKUMENTACJA PROJEKTOWA

Etap II

Bartosz Żelazko (331457)

Amadeusz Lewandowski (331397)

Treść zadania projektowego (nr. 5)

“Niewątpliwie nasza wyszukiwarka ofert jest najlepsza w branży, jednakże jakiś czas temu okazało się, że nowo dodane opinie są słabo pozycjonowane przez silnik wyszukujący. Czy da się coś z tym zrobić?”

Cel projektu

Celem projektu było stworzenie rozwiązania, który pomoże klientowi usprawnić wyszukiwarkę poprzez nowy sposób pozycjonowania nowych ofert, które nie mają żadnych opinii lub mają ich bardzo mało.

W tym celu stworzyliśmy oprogramowanie umożliwiające wytrenowanie modelu uczenia maszynowego regresji i mikroserwis będący przedłużeniem tej wyszukiwarki w którym można ten model umieścić, aby mógł wykonywać swoje zadanie.

Bazowa wyszukiwarka klienta pozycjonuje oferty względem średniej wszystkich ocen danej oferty. Dla ofert o znikomej liczbie lub braku ocen model przewiduje oceny i dostosowuje te przewidywania do istniejących ocen o ile oferta je posiada. Następnie mikroserwis odpowiednio je sortuje, zwracając przy okazji wartość korelacji spearmana dla statystyk. Po dokładniejsze informacje na ten temat należy się udać do dokumentacji pierwszego etapu tego projektu.

Struktura projektu

Projekt został napisany w języku Python i oprócz licznych modułów biblioteki standardowe wykorzystuje zewnętrzne biblioteki takie jak:

- **FastApi** - mikroserwis
- **matplotlib** - wizualizacja danych statystycznych
- **pandas** - wczytywanie i obróbka danych do nauki modelu
- **pandera** - walidacja danych do nauki modelu

- **pydantic** - walidacja danych w klasach
- **requests** - obsługa zapytań klienta do mikro serwisu
- **scikit-learn** - trenowanie i obsługa modelu ML
- **scipy** - obliczanie korelacji spearmana

Aby kod był jak najwyższej jakości użyliśmy narzędzie takich jak Ruff do formatowania kodu i MyPy do statycznej analizy typów. Dodatkowo projekt używa uv do uruchamiania i zarządzania bibliotekami.

Kod można znaleźć w repozytorium na GitHubie.

Oprócz programów pythonowych przygotowane zostało kilka skryptów shellowych w celu łatwiejszej obsługi niektórych czynności związanych z serwisem.

Skrypty:

- **run-service.sh** - umożliwia uruchomienie mikroserwisu
- **train-multiple-models.sh i train-param-sweep.sh** - pozwalają na wytrenowanie kilkunastu różnych modeli automatycznie co znacznie przyspiesza procedurę tworzenia i trenowania nowych modeli

Kod źródłowy znajduje się w folderze src. Wszystko co jest poza folderami z modułami takimi jak:

- **constants** - stałe używane w całym projekcie
- **data** - wczytywanie i obróbka plików csv
- **model** - trenowanie modelu, predykcja i preprocessing danych, aby były z nim kompatybilne
- **schemas** - schematy danych do walidacji i typowania
- **service** - mikroserwis

Można uruchomić bezpośrednio. Są to między innymi:

- **analize_logs.py** – analiza danych z mikroserwisu
- **delete_model.py** – usuwanie modelu z mikroserwisu
- **get_models.py** – sprawdzanie aktualnie dostępnych modele w mikroserwisie
- **run_service.py** – definicja mikroserwisu
- **send_random_requests.py** – program służący do wysyłania losowych zapytań do mikroserwisu celem testowania usługi oraz do symulacji danych dla testów A/B,
- **test_model.py** – testowanie modelu na zbiorze testowym
- **train_model.py** – trenowanie modelu
- **upload_model.py** – wgrywanie modelu do mikroserwisu

Wszystkie wyżej wymienione programy poza analize_logs.py mogą być wywoływane z linii poleceń wraz z obowiązkowymi bądź opcjonalnymi parametrami.

Aby zobaczyć instrukcję dotyczącą konkretnego programu należy uruchomić go z parametrem - -help.

Opis działania

Na początku musimy wytrenować model. Modelem jakiego używamy jest **GrandientBoostingRegressor** z biblioteki sklearn. Po użyciu skryptu do trenowania dane zostaną zaczytane, zwalidowane (na wypadek gdyby ktoś chciał używać jakiegoś losowego zbioru) i odpowiednio obrobione (one-hot encoding itd.), aby były kompatybilne z modelem i dostosowane do niego.

Po wytrenowaniu model zapisuje się do pliku, a my uzyskujemy ustalone w pierwszym etapie metryki tj. MAE i RMSE zmierzone na zbiorze walidacyjnym. Dzięki temu możemy wytrenować kolejny model (lub napisać ten istniejący) w celu strojenia hiper parametrów (odpowiednie argumenty przy uruchomieniu programu).

Dodatkowymi hiper parametrami modelu, które nie są jego integralną częścią ale są bardzo istotne są:

- **min_reviews** - liczba opinii, które uznajemy za wystarczającą - opinie poniżej tej liczby nie będą brały udziału w trenowaniu ani testowaniu modelu, a oferty z większą lub równą ich liczbą nie będą wysyłane do predykcji przez serwis
- **rating_weight** - waga rankingu, używane przy ofertach, które mają już jakieś opinie, ale jest ich bardzo mało, określa stopień w jakim stopniu przy finalnym rankingowaniu będzie pod uwagę brana predykcja, a w jakim istniejące oceny

Gdy już mamy dość trenowania możemy przetestować nasze modele na zbiorze testowym. Jeśli wyniki nie są zadowalające oczywiście możemy powrócić do trenowania, w przeciwnym wypadku uruchamiany nasz serwis. Następnie przy pomocy kolejnych skryptów (a raczej zapytań http, które się pod nim kryją) możemy wstawiać nasze modele do serwisu i nimi zarządzać (wstawiać, usuwać i sprawdzać jakie są wstawione). W rzeczywistym projekcie należałoby te wszystkie odpowiednio zabezpieczyć, ale ze względu na to, że to tylko uproszczenie zdecydowaliśmy się na brak mechanizmów autoryzacji. Wszystkie endpointy, które powinny być dostępne tylko dla administratora mają w swojej ścieżce słowo “admin”.

Mikro serwis umożliwia wstawienie do dwóch modeli w celu wykonania testów A/B (o których niżej). W celu rozróżniania, która grupa użytkowników ma korzystać z którego modelu, zostało użyte podejście consistent hashing na identyfikatorach, który to zawsze, obowiązkowo musi być dołączony do zapytania o sortowanie ofert. Takie podejście eliminuje używanie jakiejkolwiek bazy danych do zapisywania identyfikatorów i umożliwia przeprowadzenie testów A/B. Użytkownik nie wie z jakiego modelu korzysta, wszystko jest rozróżniane już we wnętrzu jedynego endpointu wystawionego dla zwykłego użytkownika (do sortowania ofert).

Endpointy mają też wbudowaną obsługę wszelkich błędów (np. próba skorzystania z predykcji przy braku modeli w serwisie).

Wstawienie modelu na serwis odbywa się przez rzeczywiste przesłanie plików z modelem do serwisu tak, aby można go było używać tak jak w rzeczywistym projekcie.

Gdy już mamy modele na serwisie możemy używać predykcji. Kiedy użytkownik użyje wyszukiwarki ofert klienta ta uderza do naszego endpointu, aby posegregował oferty według średniej ocen. Te które mają wystarczająco dużo ofert (parametr modelu) są pomijane i bierzemy je pod uwagę dopiero przy finalnym sortowaniu, w innych używana jest predykcja średniej ocen i dostosowanie jej za pomocą zmodyfikowanej średniej bayesowskiej.

Po posortowaniu odsyłamy gotową listę do wyświetlenia dla użytkownika, gdzie nowe oferty mają odpowiednie pozycje. W celach analitycznych zwracamy również korelację spearmana.
Nowe oferty nie są teraz ani faworyzowane będąc na samym początku, ani nic nie tracą będąc na końcu. Wszystko jest zrobione uczciwie zarówno dla oferentów jak i klientów serwisu, tak więc cel został osiągnięty. Jeśli chodzi o kryteria analityczne to wraz z testami A/B zostały omówione poniżej.

Testy A/B

W ramach prac stworzone zostały dwa modele. Pierwszy model został wytrenowany na domyślnych parametrach, i na potrzeby testów A/B został on nazwany modelem A. Model B to model który został wytrenowany wraz ze strojeniem parametrów które było oparte o wyniki modelu na zbiorze walidacyjnym. Oba modele zostały wgrane do mikro serwisu a następnie wysłano 10 000 zapytań od różnych użytkowników. Każde zapytanie zawierało losowo od 1 do 100 ofert. Oczywiście cały ruch został zasymulowany odpowiednim programem.

Następnie przy użyciu programu analyze_logs.py sprawdzono wyniki działania tych modeli w ramach mikro serwisu.

Otrzymano następujące wyniki:

Model	A	B	Predykcja naiwna (średnia ocen)	Predykcja naiwna (mediana)
Całkowita liczba predykcji	250340	254786	X	X
Średnia predykcja	4,7555	4,7581	X	X
MAE	0,0657	0,0230	0,086	0,0782
RMSE	0,2177	0,0785	0,2728	0,279

Dla porównania w tabeli wstawiono także wartości błędów dla predykcji naiwnych z wykorzystaniem średniej i mediany z pierwszego etapu projektu.

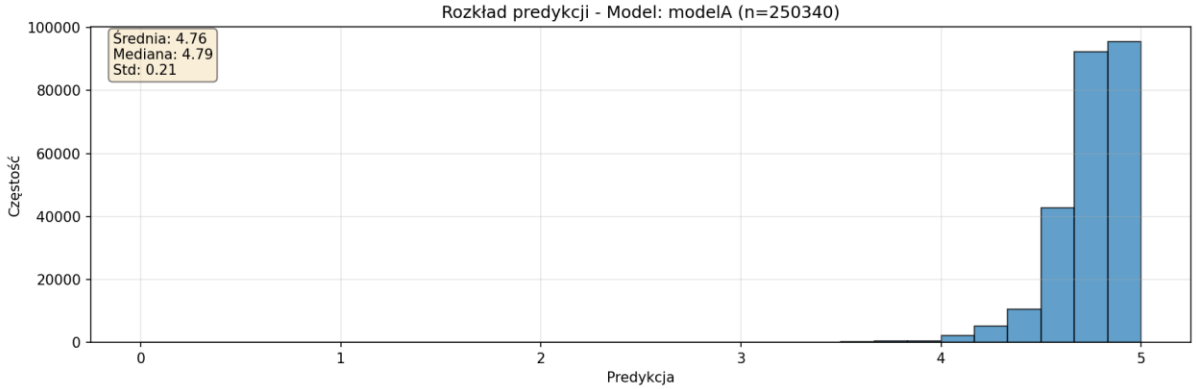
Jak możemy zauważyć oba modele bazujące na regresji liniowej (model A i B) osiągają zdecydowanie lepsze wyniki w przypadku miar MAE i RMSE co powoduje że oba nasze modele spełniają analityczne kryterium sukcesu które zostało przyjęte na etapie pierwszego etapu projektu. Dodatkowo możemy zauważyć, że model B ma ponad dwukrotnie lepsze miary jakości niż model A.

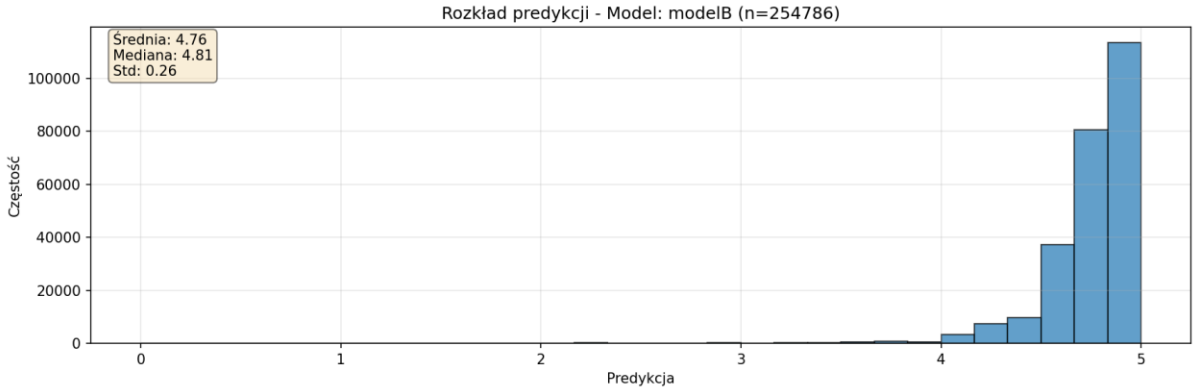
Tak więc model docelowy radzi sobie lepiej niż model bazowy co jest pożądane.

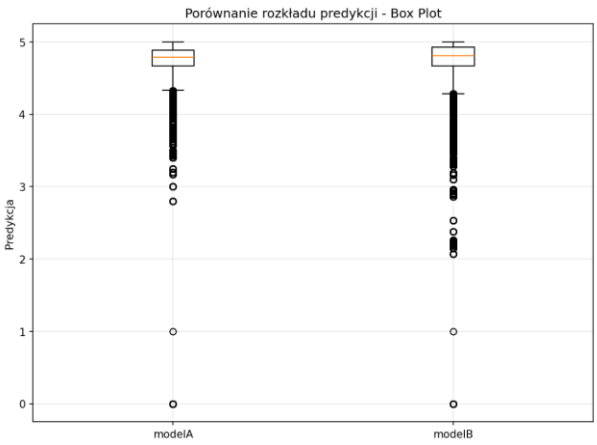
Przeprowadzono także testy statystyczne w postaci testu t-Studenta oraz test Manna-Whitney'a. Dla testu t-Studenta osiągnięto wartość parametru t na poziomie -3.9 co przekłada się na wartość krytyczną p na poziomie 0.0001 co oznacza że test jest istotny statystycznie.

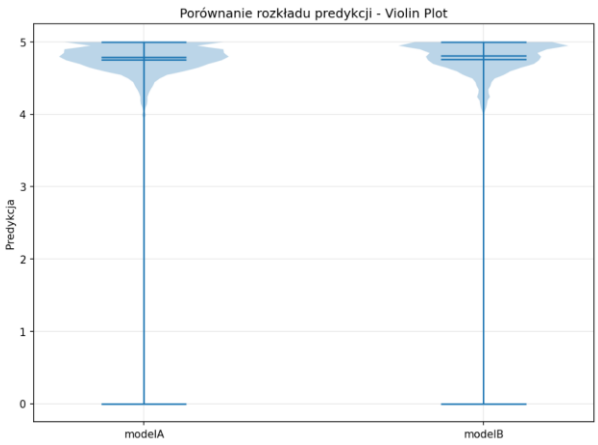
Metoda Manna-Whitney'a osiągnęła wartość U równą 29933373395 co sprowadza wartość P do zera a więc ta metoda także potwierdza statystyczną istotność naszego testu.

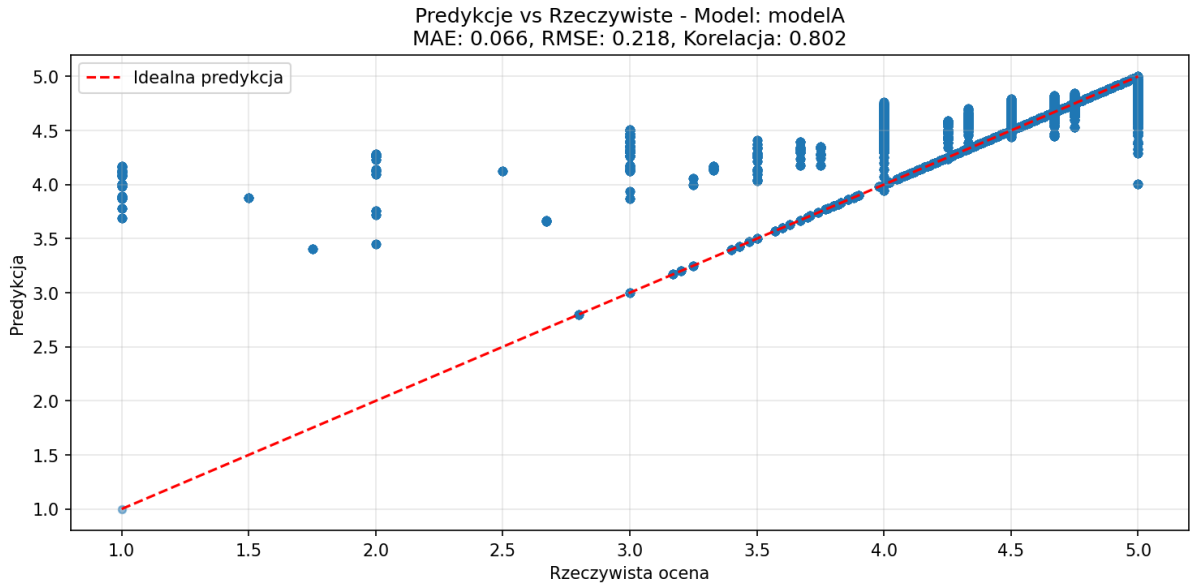
Dodatkowo poniżej zamieszczone zostały wykresy przedstawiające rozkłady predykcji dla modelu A i modelu B:

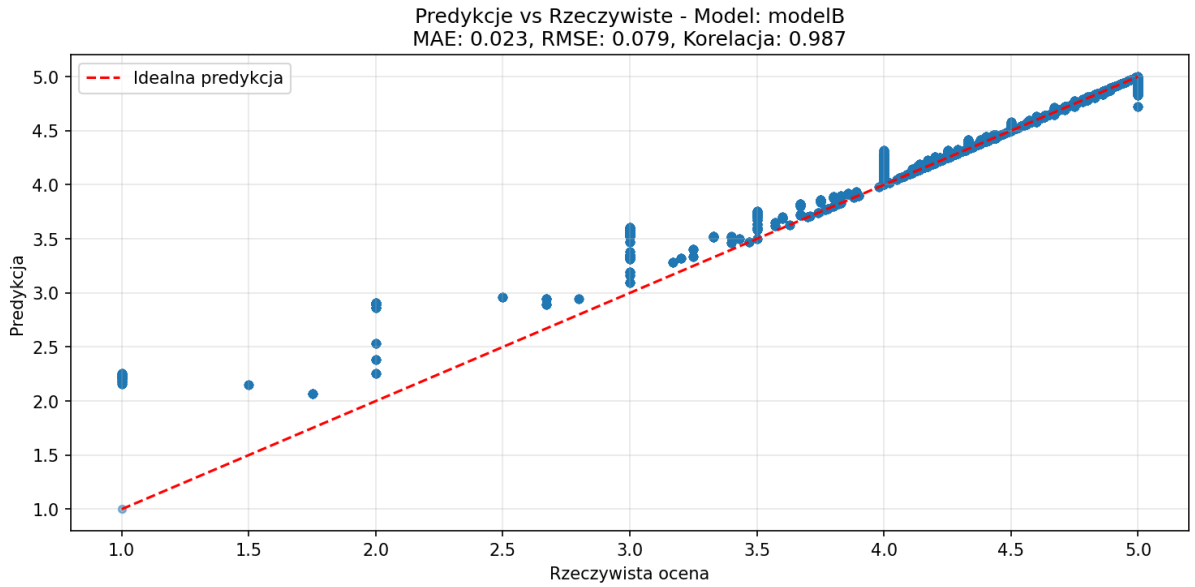

Rozkład predykcji - Model: modelA (n=250340)
Średnia: 4.76
Mediana: 4.79
Std: 0.21
Częstość
Predykcja


Rozkład predykcji - Model: modelB (n=254786)
Średnia: 4.76
Mediana: 4.81
Std: 0.26
Częstość
Predykcja


Porównanie rozkładu predykcji - Box Plot
Predykcja
modelA
modelB


Porównanie rozkładu predykcji - Violin Plot
Predykcja
modelA
modelB





Podsumowanie

Jak widać kryteria sukcesu zakładane w pierwszym etapie projektu zostały osiągnięte więc całe przedsięwzięcie zakończyło się pomyślnie i serwis “Nocarz” może szczycić się, że ich wyszukiwarka jest teraz najlepsza z najlepszych w branży i nie ma ona żadnych problemów z pozycjonowaniem nowych ofert, a to wszystko dzięki użyciu technik uczenia maszynowego.