IUM 2023Z

Jan Filipecki (305969), Jakub Niezabitowski (304082)

Etap 2

Biznesowe kryterium sukcesu

Tak jak założyliśmy w dokumentacji wstępnej, przeprowadziliśmy analizę założonego biznesowego kryterium sukcesu o treści: *"Model powinien co tydzień generować nową playlistę utworów, która docelowo ma generować więcej łącznych odsłuchań niż zbiór tych samych utworów w poprzedzającym tygodniu (Eksperyment A/B)"*, jednak zauważyliśmy małą niedokładność w tym założeniu. Podane kryterium sukcesu w rzeczywistości nie ma zbyt wiele wspólnego z eksperymentem A/B, gdyż Pozytywka do czasu wdrożenia naszego rozwiązania nie posiadała jakiejkolwiek funkcjonalności generującej playlisty. Założyć trzeba byłoby, że grupa A nie otrzymywałaby playlist a grupa B otrzymywałaby je wykorzystując jeden z modeli. Tym samym postanowiliśmy rozszerzyć analizę jakości rozwiązania o faktyczny eksperyment A/B, doprecyzowując założenia poprzednio ustalonego kryterium.

Doprecyzowanie założeń kryterium

Wcześniej ustalone założenia posiada również jeden mankament. Nie jest ono odporne na czynniki wpływające na popularność utworów niezwiązane z dodaniem playlist. Dlatego postanowiliśmy przeformułować to kryterium:
"Model powinien co tydzień generować nowe playlisty utworów, które docelowo mają generować większy wzrost liczby odsłuchań utworów przez grupę badaną, niż zbiór tych samych utworów w poprzedzającym tygodniu (Eksperyment A/B). Dodatkowo chcemy zauważyć, że ten wzrost większy jest dla modelu docelowego."
Innymi słowy, chcemy sprawdzić, czy słuchacze otrzymujący playlisty stworzone przez model docelowy będą słuchać tych utworów więcej, niż Ci którzy otrzymają je od modelu bazowego, oraz Ci, którzy playlist nie otrzymają.

W tym miejscu warto również wspomnieć, że rozważyliśmy przeprowadzenie eksperymentu Casual Impact by stworzyć syntetyczną grupę kontrolną, jednak zważając na ilość użytkowników **ogromnego** serwisu pozytywka, i możliwość przeprowadzenia dokładnej kontroli przepływu w eksperymencie A/B, uznaliśmy, że nie jest to konieczne.

Założenia eksperymentu A/B

W celu przeprowadzeniu eksperymentu założyliśmy, że model wdrożony został w ostatnim tygodniu, aby mieć dane do prowadzenia eksperymentów.

Nowy eksperyment zakłada porównanie jakości dwóch modeli: bazowego i docelowego oraz grupy kontrolnej. W ten sposób oprócz sprawdzenia wpływu samej koncepcji rozwiązania, jaką jest wdrożenie systemu generującego playlisty, sprawdzimy jakość bardziej kosztownego rozwiązania jakim jest model docelowy, w stosunku do modelu bazowego.

Eksperyment odbywa się w interwale dwutygodniowym, mierząc odsłuchania w pierwszym i drugim tygodniu. Tutaj teoretycznie powinniśmy przeprowadzić symulacje Monte Carlo by zbadać, jak długo powinniśmy mierzyć ilość odsłuchań, by mieć pewność, że zebraliśmy odpowiednią ilość próbek danych. Nie umiemy tego jednak zrobić, więc tego nie zrobiliśmy.

Eksperyment zakłada serwowanie playlist przez mikroserwis z pomocą wylosowanego modelu i jednoczesne zapisanie danych o pobierającym playlisty użytkowniku.

> $Grupa\ A$ - Użytkownicy którym zaserwowano playlisty stworzone przy pomocy **modelu bazowego**.
> $Grupa\ B$ - Użytkownicy którym zaserwowano playlisty stworzone przy pomocy **modelu docelowego**.
> $Grupa\ C$ - Użytkownicy którym **nie zaserwowano** playlist.

Rozmiary populacji wszystkich trzech są sobie równe. Podział przepływu na 3 grupy pozwoli nam na jednoczesne przeprowadzenie eksperymentów porównujących dwa modele, oraz sprawdzenie wpływu całego rozwiązania na serwis. Metryki obliczane są według następujących wzorów:

> P_A - ilość odsłuchań utworów zawartych w playlistach przez użytkowników, którym zaserwowano playlisty przy pomocy **modelu bazowego**, w tygodniu **poprzedzającym** wdrożenie i początek eksperymentu
>
> P_B - ilość odsłuchań utworów zawartych w playlistach przez użytkowników, którym zaserwowano playlisty przy pomocy **modelu docelowego**, w tygodniu **poprzedzającym** wdrożenie i początek eksperymentu
>
> P_{CA} - ilość odsłuchań utworów zawartych w playlistach **wygenerowanych przez model bazowy** przez użytkowników, którym **nie zaserwowano** playlist, w tygodniu **poprzedzającym** wdrożenie i początek eksperymentu
>
> P_{CB} - ilość odsłuchań utworów zawartych w playlistach **wygenerowanych przez model docelowy** przez użytkowników, którym **nie zaserwowano** playlist, w tygodniu **poprzedzającym** wdrożenie i początek eksperymentu

N_A - ilość odsłuchań utworów zawartych w playlistach przez użytkowników, którym zaserwowano playlisty przy pomocy **modelu bazowego**, w tygodniu **następującym po** wdrożeniu i początku eksperymentu

N_B - ilość odsłuchań utworów zawartych w playlistach przez użytkowników, którym zaserwowano playlisty przy pomocy **modelu docelowego**, w tygodniu **następującym po** wdrożeniu i początku eksperymentu

N_{CA} - ilość odsłuchań utworów zawartych w playlistach **wygenerowanych przez model bazowy** przez użytkowników, którym **nie zaserwowano** playlist, w tygodniu **następującym po** wdrożeniu i początku eksperymentu

N_{CA} - ilość odsłuchań utworów zawartych w playlistach **wygenerowanych przez model docelowy** przez użytkowników, którym **nie zaserwowano** playlist, w tygodniu **następującym po** wdrożeniu i początku eksperymentu

Następnie obliczane są cztery metryki:

$$T_{B/A} = \frac{\frac{N_B}{P_B}}{\frac{N_A}{P_A}}, \qquad T_{B/CB} = \frac{\frac{N_B}{P_B}}{\frac{N_{CB}}{P_{CB}}}, \qquad T_{A/CA} = \frac{\frac{N_A}{P_A}}{\frac{N_{CA}}{P_{CA}}}$$

Trzecia ze zdefiniowanych metryk $T_{A/CA}$ jest jedynie pomocnicza i diagnostyczna, stosowana jedynie w przypadku, gdy znajdziemy wyższość modelu bazowego nad docelowym.
Zakładamy, że konieczne jest spełnienie trzech kryteriów:

$$\left(T_{B/A} \geq 1.1\right) \wedge \left(T_{B/CB} \geq 1.1\right) \wedge \left(\frac{N_B}{P_B} \geq 1\right)$$

Ostatnie z nich zapewnia, że dla modelu docelowego Pozytywka odnotuje **większy wzrost, a nie mniejszy spadek** odsłuchań.

Realizowanie eksperymentu przez więcej niż jedną iterację może być narażone na wpływ sprzężenia zwrotnego, związanego ze zmianą popularności piosenek dodanych na playlisty. Wobec tego realizacje eksperymentów powinny być oddalone od siebie w czasie o co najmniej miesiąc, według naszej studenckiej intuicji.

Analityczne kryterium sukcesu

Tym samym zakładamy **jednoczesne** spełnienie czterech kryteriów, by móc stwierdzić sukces. Przypominając, te kryteria to:

- Chcemy zauważyć co najmniej 10% poprawę jakości w metryce Sihlouette dla modelu docelowego w stosunku do modelu bazowego:

$$\frac{SC_B}{SC_A} \geq 1.1$$

SC_B - współczynnik Silhouette dla playlist wygenerowanych przez model **docelowy**

SC_A - współczynnik Silhouette dla playlist wygenerowanych przez model **bazowy**

- Chcemy zobaczyć, że wzrost ilości odsłuchań po wdrożeniu modelu docelowego jest o 10% większy niż po wdrożeniu modelu bazowego:

$$T_{B/A} \geq 1.1$$

- Chcemy zobaczyć, że wzrost ilości odsłuchań po wdrożeniu modelu docelowego jest o 10% większy niż w przypadku nie podania playlist:

$$\left(T_{B/C} \geq 1.1\right) \wedge \left(\frac{N_B}{P_B} \geq 1\right)$$

Realizacja zadań modelowania

Po takim doprecyzowaniu kryteriów przeprowadziliśmy zadania modelowania, trzymając się wcześniej zdefiniowanych założeń:

- Czas trwania playlisty to maksymalnie jedna godzina
- Playlisty generowane mają być co tydzień
- Playlisty z poprzedzającego tygodnia zastępowane są nowo-wygenerowanymi playlistami
- Jeden utwór może występować w wielu playlistach
- Model generuje od 5 do 15 playlist

Ostatecznie przyjęliśmy, że oba modele generować będą po 10 playlist.

Model bazowy

W dokumentacji wstępnej założyliśmy, że playlisty generowane będą na podstawie częstości występowania gatunków w listach gatunków opisujących autorów utworów. Uznaliśmy że lepiej i prościej będzie jednak wybranie playlist na podstawie średniej popularności, po uprzednim stworzeniu zbiorów utworów przypisanych do każdego z gatunków, który opisuje ich autorów.

Pierwszym krokiem wobec tego jest przypisanie utworom list gatunków, które opisują ich autorów. Następnie dla każdego gatunku tworzone są grupy utworów, które w listach gatunków posiadają dany gatunek. Utwory, które przypisane mają kilka gatunków znajdować się więc będą w wielu grupach. Potem utwory w grupach sortowane są według atrybutu popularity (podanego przez Pozytywkę) i wybierane jest tyle najbardziej popularnych utworów, by dla każdego gatunku utworzyć około godzinną playlistę (nie krótszą, dłuższą jedynie o część długości jednej piosenki). Jeśli w danej playliście brakuje utworów, są one losowo dosztukowywane ze zbioru wszystkich utworów. Na koniec dla każdej playlisty obliczana jest średnia popularność i wybierane jest 10 najbardziej popularnych playlist.

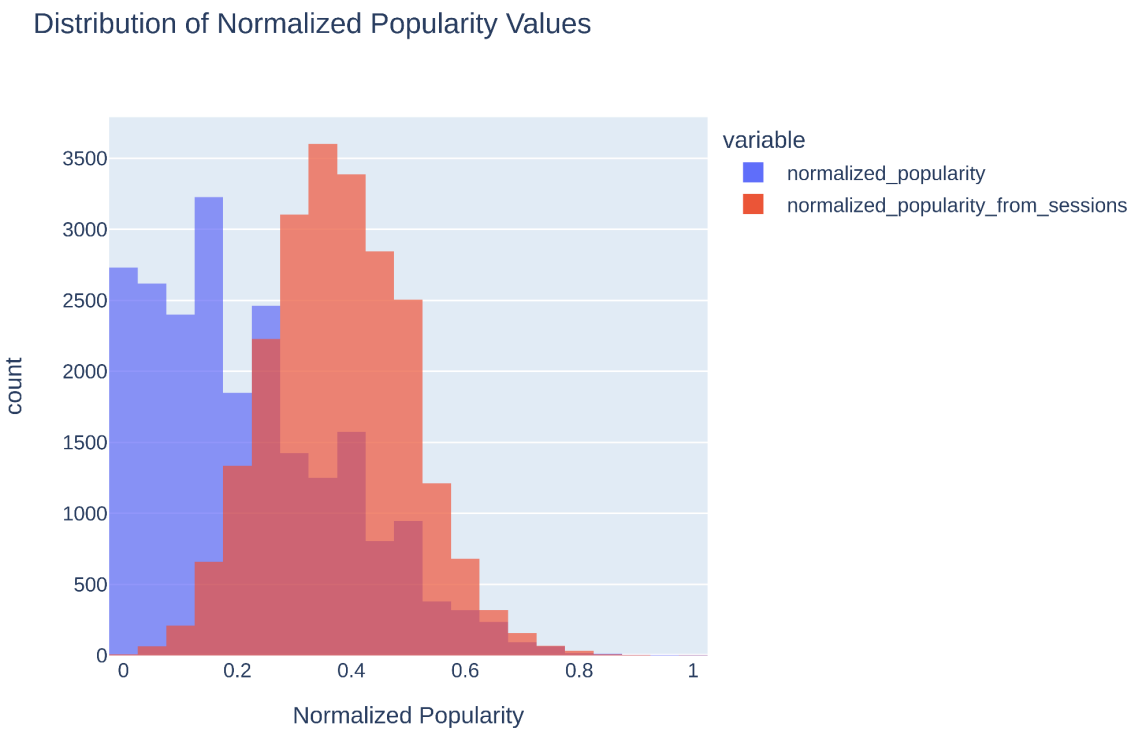
Model bazowy zaimplementowany został z wykorzystaniem biblioteki **pandas** i przygotowane zostały skrypty umożliwiające tworzenie playlist poprzez uruchomienie w mikroserwisie.

Model docelowy

Tak jak wspominaliśmy we wstępnym sprawozdaniu, zaczęliśmy od obliczenia dodatkowej metryki popularności, wyznaczonej na podstawie ilości akcji odtworzeń i pominięć w bazie sesji użytkowników. Akcje punktowane były w następujący sposób:

- Odtworzenie → +1
- Pominięcie→ -1
- Polubienie→ +2
- Reklama→ 0

Po drobnym eksperymentowaniu z dobranymi wartościami zauważyliśmy znaczną różnicę pomiędzy rozkładem wartości wcześniej podanego parametru popularity a tym stworzonym przez nas. Nowy rozkład przypomina rozkład normalny, co daje dużo więcej informacji na temat danych utworów. Popularność znajdująca się w danych Pozytywki jest tak naprawdę prawą częścią rozkładu normalnego, gdyż Pozytywka jest serwisem **premium**, wobec czego zajmuję się tylko ponadprzeciętnymi utworami.

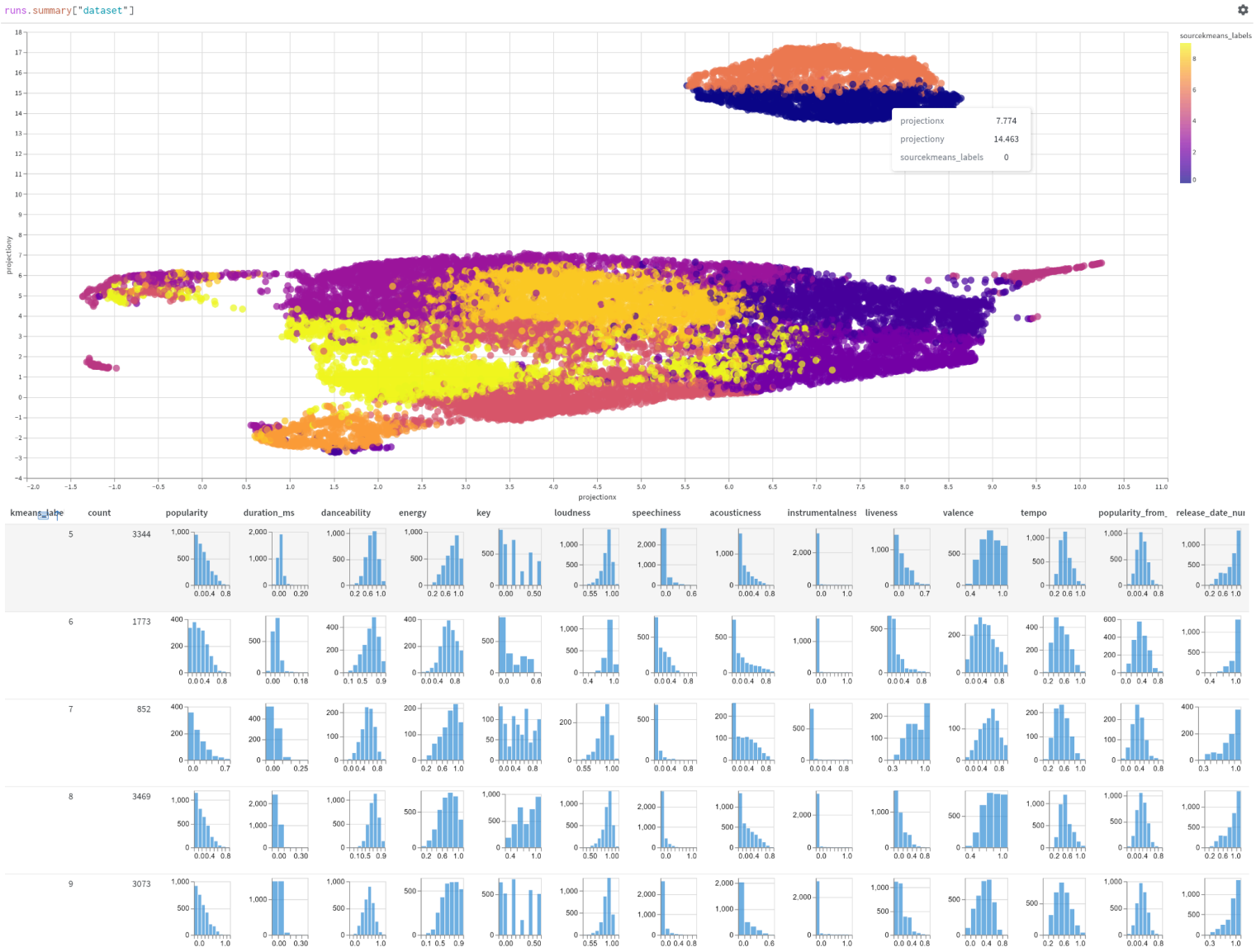


Następnie dane liczbowe zostały znormalizowane, oraz w przypadku daty wydania również uprzednio przekonwertowane do wartości liczbowych. Upewniliśmy się również, że dane nie zawierają brakujących pól.

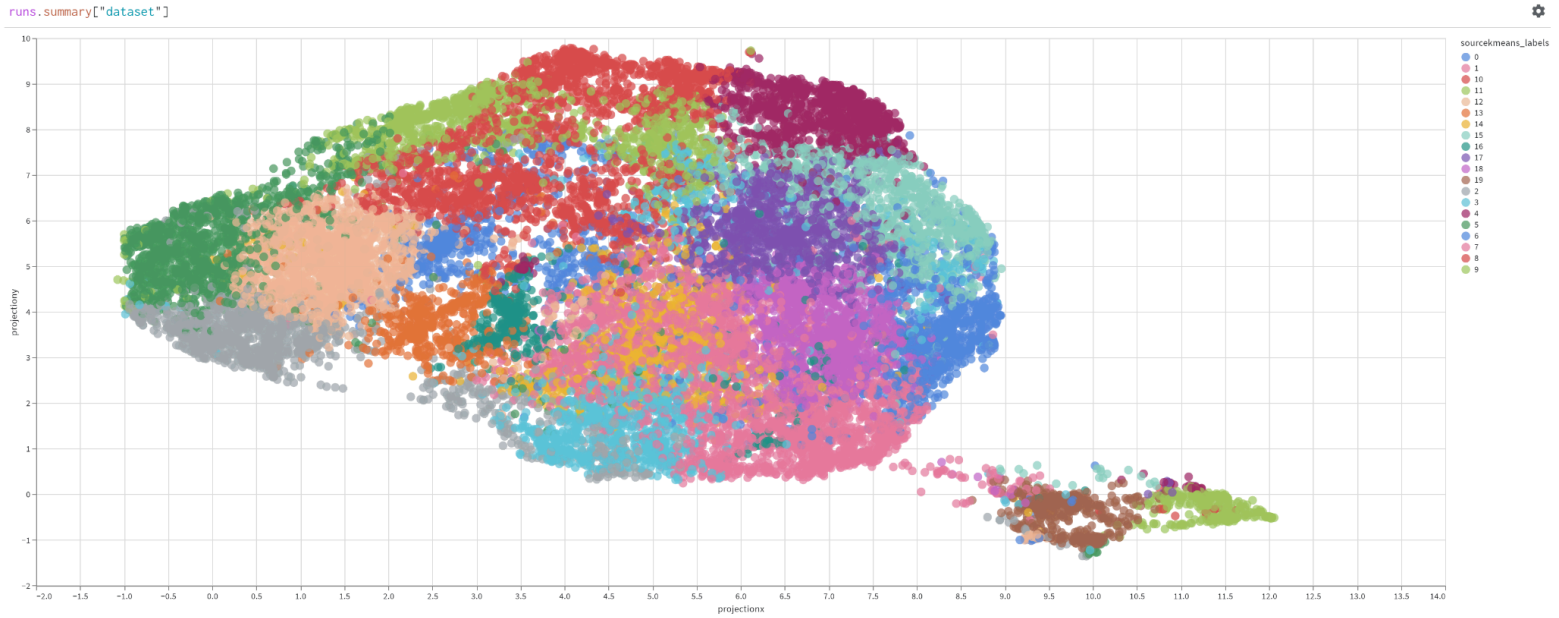
Później przeszliśmy do eksperymentów związanych z klasteryzacją. Pomimo wcześniejszego założenia o korzystaniu z metody klasteryzacji gęstościowej DBSCAN postanowiliśmy najpierw sprawdzić jak sprawować się będzie metoda centroidowa k-średnich, gdyż tak naprawdę nie mamy pewności, które z tych podejść będzie lepsze. Natura danych na których pracujemy może nie być odpowiednia dla klasteryzacji gęstościowej, jeżeli zawierają one zbyt dużo szumu. Wówczas metoda centroidowa może okazać się lepsza.
Podczas eksperymentów postanowiliśmy skorzystać z narzędzia do śledzenia wyników eksperymentów **Weights & Biases** zwanym również **wandb** (dla przyjaciół Wandzia).
Dla algorytmu k-średnich sprawdzaliśmy jakość klasteryzacji korzystając z metody **UMAP**, metryki **inercji** oraz metryki **Sihlouette**.

Po przeprowadzeniu wstępnej klasteryzacji dla 10 klastrów reprezentacja danych UMAP pokazała całkiem niezłą separację klastrów, jednak rzuciłą się nam w oczy wyspa danych widoczna na górze wykresu. Po weryfikacji z rozkładami danych w klasach ustaliliśmy, że są to dane “wulgarne”. Jako że Pozytywka jest serwisem **premium**, uznaliśmy, że nie chcemy, by modele generowały

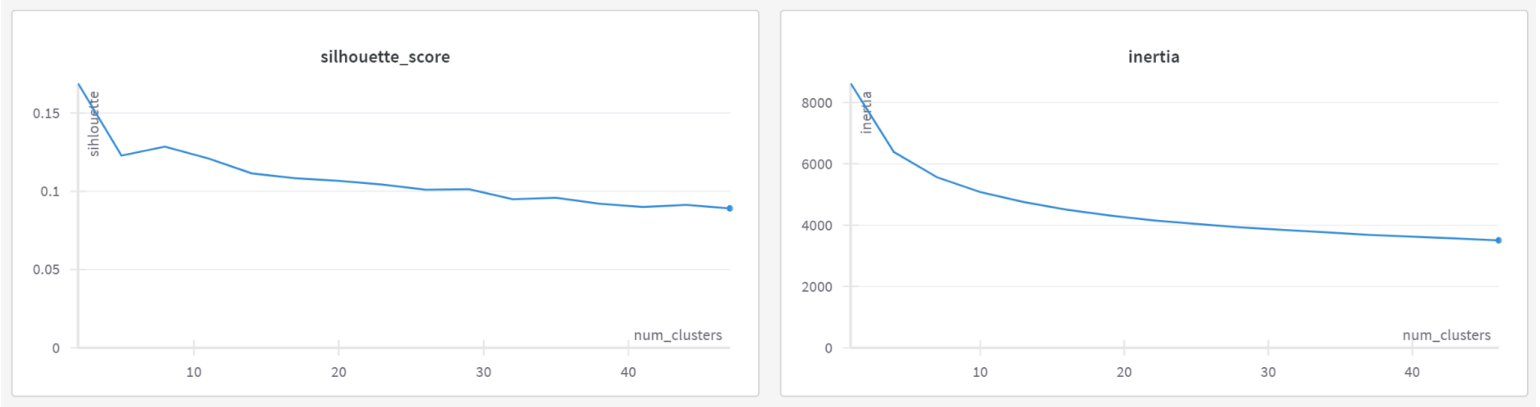
jakiekolwiek playlisty, których główną cechą jest wulgarność, dlatego uznaliśmy że usuniemy kolumnę “explicit” na potrzeby naszej klasteryzacji.



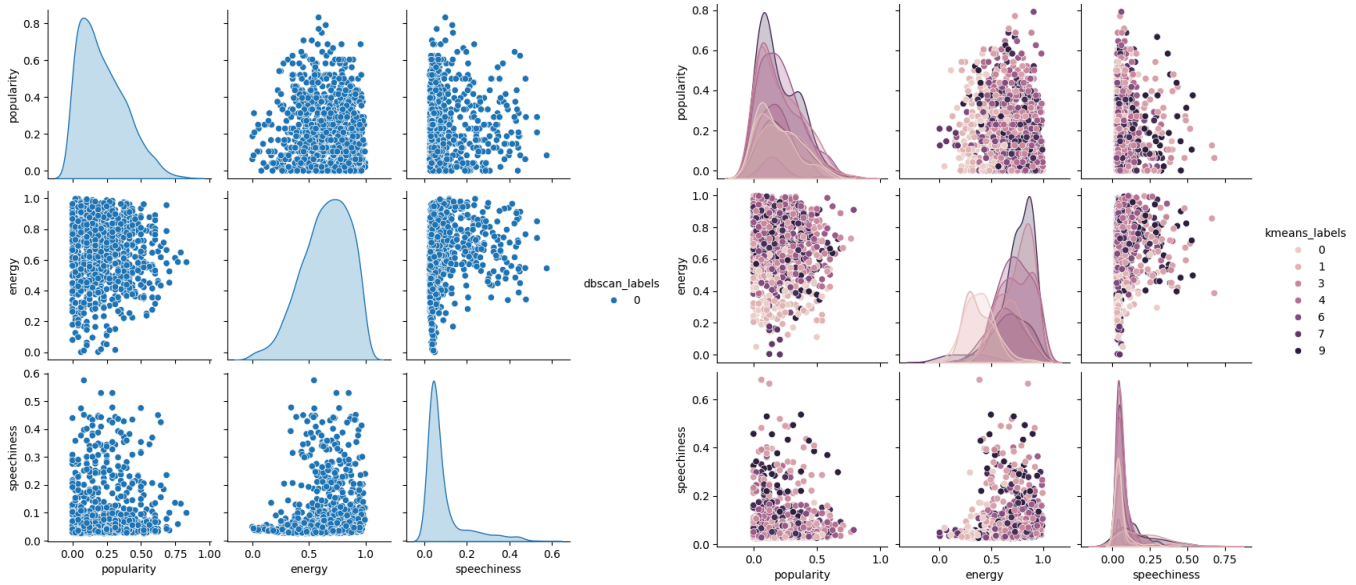
Po usunięciu kolumny “explicit” reprezentacja danych wygląda tak. Wyspa nie jest już widoczna, a reprezentacja danych przypomina kontynent Australii i Oceanii, co jest zastanawiające zważając na fakt, że Taylor Swift nie pochodzi z Australii, a z Ameryki Północnej. Niemniej dane dalej zawierają pewien segment “odklejony” od reszty. Z naszych obserwacji wynika, że są to utwory posiadające charakterystykę instrumentalno akustyczną, wobec tego nie stanowią problemu, gdyby zostać miały przypisane do jednej playlisty.



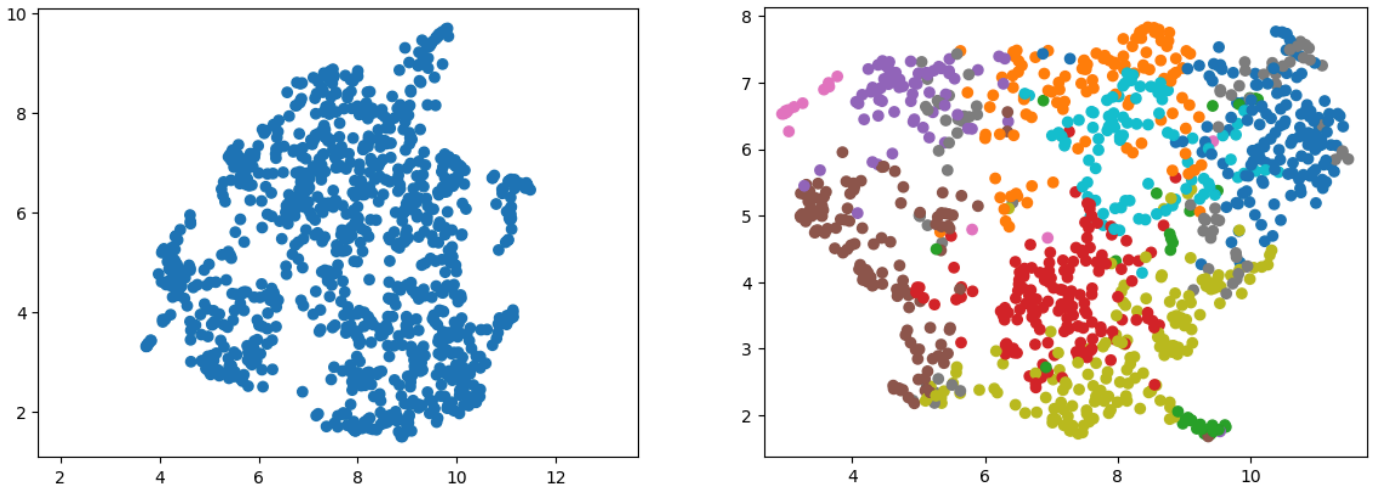
Hiperparametr ilości klastrów postanowiliśmy dobrać korzystając z “metody łokciowej”, dla metryk inercji i Silhouette. Odpowiednia ilość klastrów jest wówczas balansem pomiędzy spójnością wewnątrz klastrów oraz różnicą danych pomiędzy klastrami. Odpowiednia ilość klastrów dla naszych danych to około 11 klastrów.



Następnie przystąpiliśmy do próby klasteryzacji za pomocą metody gęstościowej DBSCAN. Pomimo próby stosowania różnych metryk miary odległości (euklidejska, manhattan, etc.) oraz różnych wartości parametru epsilon, nie udało nam się otrzymać klastrów, które nie zostaną sklasyfikowane jako szum.



Widoczne powyżej wykresy reprezentują rozkłady klas dla obu metod. Eksperymenty przeprowadzane na metodzie DBSCAN nie przyniosły żadnego skutku.



Powyżej widoczny jest absolutny brak podziału na klastry przy pomocy metody DBSCAN w porównaniu do klastrów otrzymanych metodą k-średnich. Nasze przypuszczenie jest takie, że dane zawierają zbyt dużo szumu, lub nie posiadają cech, które umożliwiałyby klasteryzację gęstościową. Jako że metoda k-średnich jest odporna na szum, efekty jej wykorzystania są dużo lepsze. Wobec powyżej przedstawionych wyników postanowiliśmy w modelu docelowym wykorzystać metodę k-średnich z parametrem ilości klastrów 11.

Następnie, podobnie jak w modelu bazowym, sortujemy utwory w klastrach na bazie naszego wewnętrznego atrybutu popularności, po czym wybieramy w każdym klastrze wszystkie najpopularniejsze utwory, których łączny czas nie przekracza godziny. W przeciwieństwie do założonego mechanizmu algorytmu, postanowiliśmy nie korzystać z wbudowanego atrybutu

“popularity”, a jedynie z tego stworzonego przez nas. Postanowiliśmy również nie zmieniać ilości klastrów w zależności od ilości danych, ponieważ założyliśmy, że ilość playlist ma być stała i ma wynosić 10 playlist. Wobec tego, gdy utworów w którymś z klastrów jest zbyt mało, są one losowo dosztukowywane ze zbioru wszystkich utworów. Tak zbudowane playlisty są zwracane w formie listy list “id” utworów. Na koniec ze wszystkich powstałych playlist, których jest tyle, ile klastrów (zakładamy że klastrów zawsze będzie więcej niż playlist), wybierane jest 10, które posiada największą średnią popularność.

Realizując powyższe ustalenia skorzystaliśmy z biblioteki **sklearn**, z której zaczerpnęliśmy algorytmy k-średnich, DBSCAN oraz metryki Silhouette oraz inercję. Do obróbki danych został przygotowany skrypt “preprocess.py” wykorzystujący bibliotekę **pandas**, oraz ponownie sklearn do normalizacji danych. Powstał również skrypt “prepare_playlists.py”, który przperowadza klasteryzację oraz wyżej wymienione procesy by na koniec zwrócić żądaną listę playlist.

Implementacja mikroserwisu

Postanowiliśmy stworzyć mikroserwis o architekturze REST API wykorzystując **FastAPI** i umożliwić uruchomienie jego w środowisku docker.

Wszystkie testy przeprowadzone były w środowisku **Swagger UI** dostarczanym przez FastAPI.

Mikroserwis odpowiada na zapytania GET wysłane na odpowiedni URL, zawierające “id” użytkownika w zawartości zapytania, i w zależności od wylosowanej ścieżki - A, B, lub C, czyli playlisty stworzone przez model bazowy, playlisty stworzone przez model docelowy, i brak playlist zwracają odpowiednie dane o piosenkach w playlistach w formacie JSON. “id” użytkownika zapisywane jest w pliku w kontenerze mikroserwisu, razem z przypisaną do niego wylosowaną ścieżką. Na podstawie tych danych później generowane są metryki pozwalające na ewaluację eksperymentu A/B.


```
{
  "user_id": 911,
  "group": "A",
  "playlists": [
    [
      "7lPN2DXiMsVn7XUKtOW1CS",
      "5QO79kh1waicV47BqGRL3g",
      "6tDDoYIxWvMLTdKpjFkc1B",
      "0VjIjW4GlUZAMYd2vXMi3b",
      "6f3Slt0GbA2bPZlz0aIFXN",
      "3FAJ6O0NOHQV8Mc5Ri6ENp",
      "60ynsPSSKe6O3sfwRnIBRf",
      "3YJJjQPAbDT7mGpX3WtQ9A",
      "1XXimziG1uhM0eDNCZCrUl",
      "35mvY5S1H3J2QZyna3TFe0",
      "4u4NyuceXP7Uzh7XFJKCr1",
      "6UelLqGlWMcVH1E5c4H7lY",
      "6Im9k8u9iIzKMrmV7BWtlF",
      "1diS6nkxMQc3wwC4G1j0bh",
      "45bE4HXI0AwGZXfZtMp8JR",
      "1KixkQVDUHggZMU9dUobgm",
      "54bFM56PmE4YLRnqpW6Tha",
      "31qCy5ZaophVA81wtlwLc4",
      "7qEHsqek33rTcFNT9PFqLf",
```

```
{
  "user_id": 1006,
  "group": "B",
  "playlists": [
    [
      "4V84hb0KLUwGgLoXfX0YMa",
      "3NrfU7FvNF0w6mGcFVnlUj",
      "5LME7YULt0enp6UAB8VoDn",
      "15k1TDabqSEmyXOwMq9RM7",
      "2IT0T0EqPaUxasjl2o8J2G",
      "4odiyU3myG29Ld0wurMfE8",
      "4tqBQD2QG0IYLU08rpkU6X",
      "6FI3RJ58Ztl0X1VtA6pVs9",
      "0i5el041vd6nxrGEU8QRxy",
      "5cTsXX5qwa6zmG80OCz4hR",
      "2aadUB2b4hugbdIPU8Aypk",
      "3G0yz3DZn3lfraledmBCT0",
      "5OOxMbmz5txzE78oZbGQhY",
      "0CaBBQsaAiRHhiLmzi7ZRp",
      "1YP719l4JjsOmyU4PGv3c0",
      "2O8EVtQtfsKBABn0hOxs8Y"
    ],
    [
      "1q8E1FfFuhd12c5JcJwPxQ",
```

```
{
  "user_id": 191,
  "group": "C",
  "playlists": null
}
```

```
{
  "user_id": 1,
  "group": null,
  "playlists": null
}
```

Stworzyliśmy również endpoint pozwalający na wysłanie zapytania GET i wygenerowanie metryki opartej na stosunku wartości Sihlouette dla modelu bazowego i docelowego, tak jak wcześniej zakładaliśmy.


```
{
  "silhouette_score": {
    "criteria": "(SC_b / SC_a) >= 1.1",
    "value": 1.382380856055529,
    "is_met": true
  },
  "t_metrics": {
    "name": "t_ba",
    "value": 1.9580419580419581
  },
  "criteria": {
    "criteria": "(t_ba >= 1.1) and ((n_b / p_b) >= 1)",
    "value": null,
    "is_met": true
  }
}
```


Zakładamy, że dane które Pozytywka przesyła nam co tydzień dodawane są do plików kontenera seryjnie, poprzez inne mechanizmy niż endpointy API, stąd brak odpowiednich wejść dla wczytania danych.

Mankamentem tej implementacji jest to, że nie weryfikuje ona tego, czy dane dostarczone przez pozytywkę nie zmieniły struktury, bądź nie pojawiły się w nich nowe braki lub błędy. Poprawka która powinna zostać wprowadzona w przyszłości to odpowiednia weryfikacja wczytywanych danych przed realizacją modelowania.

Porównanie jakości modeli i jakości rozwiązania

Przy pomocy stworzonych skryptów zaimplementowanych w naszym mikroserwisie obliczyliśmy wartość $\frac{SC_B}{SC_A}$.

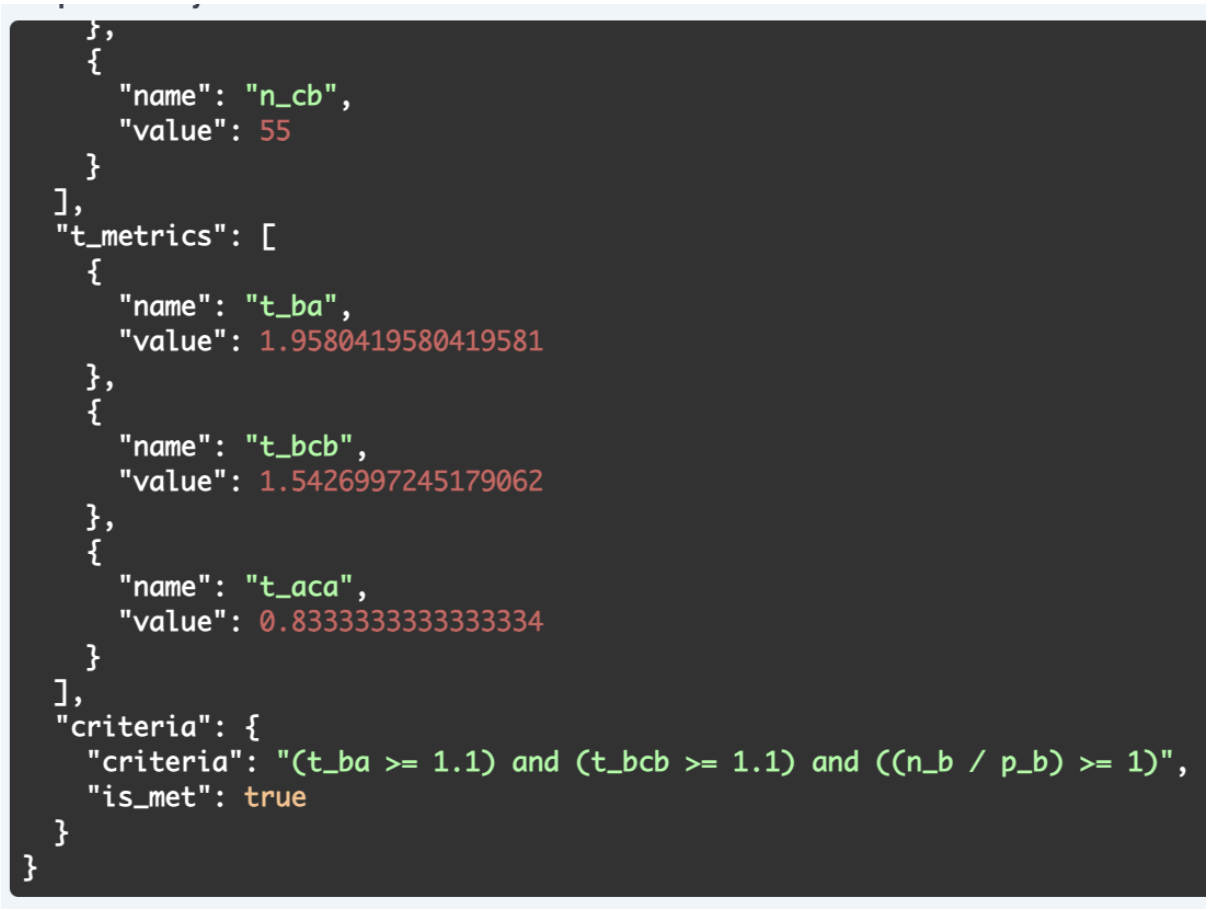
Wartość ilorazu współczynników Shilouette wyniósł **1,38**, co spełnia postawione przez nas analityczne kryterium sukcesu.
Wartości metryk w eksperymencie A/B prezentują się następująco:

$$T_{B/A} = 1.96,\ T_{B/CB} = 1.54,\ T_{A/CA} = 0.83,\ \frac{N_B}{P_B} = 1.7$$

Oznacza to, że kryterium zamieszczone poniżej zostało spełnione.

$$\left(T_{B/A} \geq 1.1\right) \wedge \left(T_{B/CB} \geq 1.1\right) \wedge \left(\frac{N_B}{P_B} \geq 1\right)$$

Przetestowaliśmy również działanie endpointów umożliwiających realizację eksperymentu A/B.



Łącznie, nasze API posiada 7 endpointów:

- /
  Służy on do inicjalizowania mikroserwisu. Jeśli w kontenerze Dockera nie znajdują się gotowe już pliki .json z wynikami predykcji modeli, API samo uruchomi modele i wygeneruje predykcje.
- **/update**
  Służy do aktualizowania predykcji. Danych w serwisie Pozytywka cały czas przybywa. Endpoint ten umożliwia ręczną aktualizację predykcji z uwzględnieniem najnowszych danych.
- **/base-model**
  Endpoint ten zwraca odpowiedź z przygotowanymi przez model bazowy playlistami.
- **/target-model**
  Endpoint ten zwraca odpowiedź z przygotowanymi przez model docelowy playlistami.
- **/abc-test**
  Służy do przeprowadzenia eksperymentu A/B. Wczytuje od bazę grup użytkowników z pliku .json, a jeśli takowy nie istnieje, to samemu grupuje użytkowników z ostatnich 2 tygodni. Następnie przeprowadza odpowiednie obliczenia i jako odpowiedź zwraca wyniki eksperymentu A/B.

- **/asc**
  Jest to skrót od Analytical Success Criteria, czyli analityczne kryteria sukcesu. Jak sama nazwa wskazuje, przeprowadza on obliczenia, na podstawie których można stwierdzić czy zakładane kryteria są spełnione
- **/get-predictions/{user_id}**
  Ten endpoint pozwala nam otrzymać dokładne informacje o użytkowniku pod kątem grupy badawczej eksperymentu A/B. Podając w adresie id użytkownika można sprawdzić do jakiej grupy należy i jakie playlisty zostały do niego przypisane (zrzuty ekranu w sekcji **Implementacja Mikroserwisu**)
- **/docs**
  Endpoint ten uruchamia środowisko Swagger UI, które wykorzystywaliśmy do testowania odpowiedzi mikroserwisu.

Kod implementacji oraz analizy danych

Kod aplikacji oraz kod analizy danych znajduje się w załączonym pliku .zip.
Skrypt mikroserwisu należy najpierw zbudować poleceniem
docker compose up –build
A następnie uruchamiać już za pomocą
docker compose up
Po uruchomieniu dockera, API możemy badać pod adresem:
http://0.0.0.0:8000/docs/