RAPORT WALIDACYJNY

Daria Bartkowiak, Aleksander Brandt

1 Wprowadzenie

Naszym zadaniem była walidacja projektu klasteryzacji treści dostępnych na platformie Netflix. Niniejszy raport ocenia poprawność, efektywność oraz jakość implementacji tego projektu. Wskazujemy zarówno jego mocne strony, jak i potencjalne błędy, nieefektywności oraz obszary do usprawnienia. Celem naszej analizy jest wskazanie grupie budowy rzeczy, które walto/należy poprawić aby model działał płynniej i skuteczniej.

2 Mocne strony projektu

1. **Jasno zdefiniowane cele biznesowe**:
   Grupa budowy precyzyjnie określa cele, takie jak poprawa retencji użytkowników i optymalizacja rekomendacji, co jest zgodne z potrzebami biznesowymi Netflixa.
2. **Analiza danych i preprocessing:**
   Dobrze przeprowadzona wstępna analiza danych oraz preprocessing, które stanowią solidny fundament dla dalszych etapów modelowania. Dane zostały poprawnie wczytane, połączone i odpowiednio oczyszczone.
3. **Przetwarzanie wstępne danych**:
   Połączenie cech tekstowych (tytuł, opis, gatunek) w jedną zmienną (`text`) na plus.
4. **Różnorodność algorytmów klastrowania**:
   Projekt testuje różne algorytmy (KMeans, MiniBatchKMeans, Gausian Mixture, Birch, DBSCAN), co wskazuje na wszechstronne podejście do modelowania.
5. **Zastosowanie Optuny**:
   Wykorzystanie narzędzia Optuna do optymalizacji hiperparametrów zwiększa precyzję doboru parametrów modelu.
6. **Optymalizacja metryki Silhouette**:
   Wniosek, że 7 klastrów jest optymalne na podstawie metryki Silhouette bardzo trafne.

3 Zidentyfikowane problemy

1. **CPU**:
   W trakcie pracy nad notebookiem konieczne było przejście z wykorzystania GPU na CPU, nasze ponieważ środowisko nie posiadało wsparcia dla CUDA. W związku

z tym usunęliśmy zależności od bibliotek takich jak cuml, cupy i cudf, a wszystkie operacje (np. redukcja wymiarowości, t-SNE, modele BERT) dostosowaliśmy do wersji CPU. Dzięki temu kod jest w pełni kompatybilny i możliwy do uruchomienia na dowolnym sprzęcie, niezależnie od dostępności GPU.

2. **Redundantne importy bibliotek**:
   Biblioteki takie jak `numpy`, `pandas` czy `torch` są importowane wielokrotnie w różnych komórkach, co zwiększa złożoność kodu.

3. **Wielokrotne wykorzystywanie redukcji wymiarowości**:
   Metody redukcji wymiarowości — t-SNE, PCA oraz UMAP — są wywoływane wielokrotnie w różnych częściach kodu. Znacząco wydłuża to czas wykonania.

4. **Błąd w wyświetlaniu opisów klastrów**:
   Kod do wyświetlania opisów najbliższych centrom klastrów nie działa z powodu braku zmiennej.

4 Rekomendacje

1. **Konsolidacja importów**:
   Umieszczenie wszystkich importów w jednej komórce na początku notatnika w celu poprawy czytelności kodu.

2. **Usprawnienie redukcji wymiarowości**:
   Wybór jednej techniki redukcji wymiarowości (np. UMAP) z wyraźnym uzasadnieniem jej zastosowania.

3. **Wykorzystanie podziału danych**:
   Zastosowanie zbiorów train, val, test do generalizacji klastrów, np. za pomocą metryki Adjusted Rand Index.

4. **Usunięcie kodu GPU**:
   Eliminacja fragmentów kodu związanych z GPU i zmiana na CPU (wówczas wszystkie komputery będą mogły odpalić kod).

5. **Naprawa wyświetlania klastrów**:

```
cluster_centroids = minikmeans_silhouette.cluster_centers_
closest_docs_indices = {}

for cluster_label in range(minikmeans_silhouette.n_clusters):
    cluster_indices =
       df_optymalizacja[df_optymalizacja['minikmeans_silhouette']
       == cluster_label].index

    if len(cluster_indices) > 0:
        X_cluster = X[cluster_indices]
        distances_to_centroid = pairwise_distances(
            cluster_centroids[cluster_label].reshape(1, -1),
            X_cluster,
            metric='euclidean'
        )
```

```
        closest_doc_index_in_cluster =
            np.argmin(distances_to_centroid)
        original_index =
            cluster_indices[closest_doc_index_in_cluster]
        closest_docs_indices[cluster_label] = original_index

def wrap_text_by_words(text, words_per_line=20):
    words = text.split()
    wrapped_lines = [' '.join(words[i:i + words_per_line])
        for i in range(0, len(words), words_per_line)]
    return '\n'.join(wrapped_lines)

print("Opisy najblizsze centroidom klastrow Silhouette:")
for cluster_label, doc_index in closest_docs_indices.items():
    full_description = df_optymalizacja.loc[doc_index,
        'description']

    if isinstance(full_description, pd.Series):
        full_description = full_description.iloc[0]

    wrapped_description =
        wrap_text_by_words(full_description, words_per_line=20)

    print(f"\n Klaster {cluster_label}:\n")
    print(wrapped_description)
    print("-" * 100)
```

6. **Konsolidacja redukcji wymiarowości**: Z punktu widzenia efektywności i przejrzystości kodu wystarczy zastosować każdą z technik redukcji jednorazowo i zapisać wyniki do zmiennych (np. X_tsne, X_pca, X_umap).

5 Wyniki walidacji

Przeprowadziliśmy walidację dla modeli MiniBatchKMeans zoptymalizowanych przez grupę budowy względem różnych metryk (Silhouette, Calinski-Harabasz, Davies-Bouldin oraz wersji domyślnej). Oto wyniki, które uzyskaliśmy:

Tabela 1: Wyniki walidacji modeli MiniBatchKMeans na zbiorze walidacyjnym X_{val}

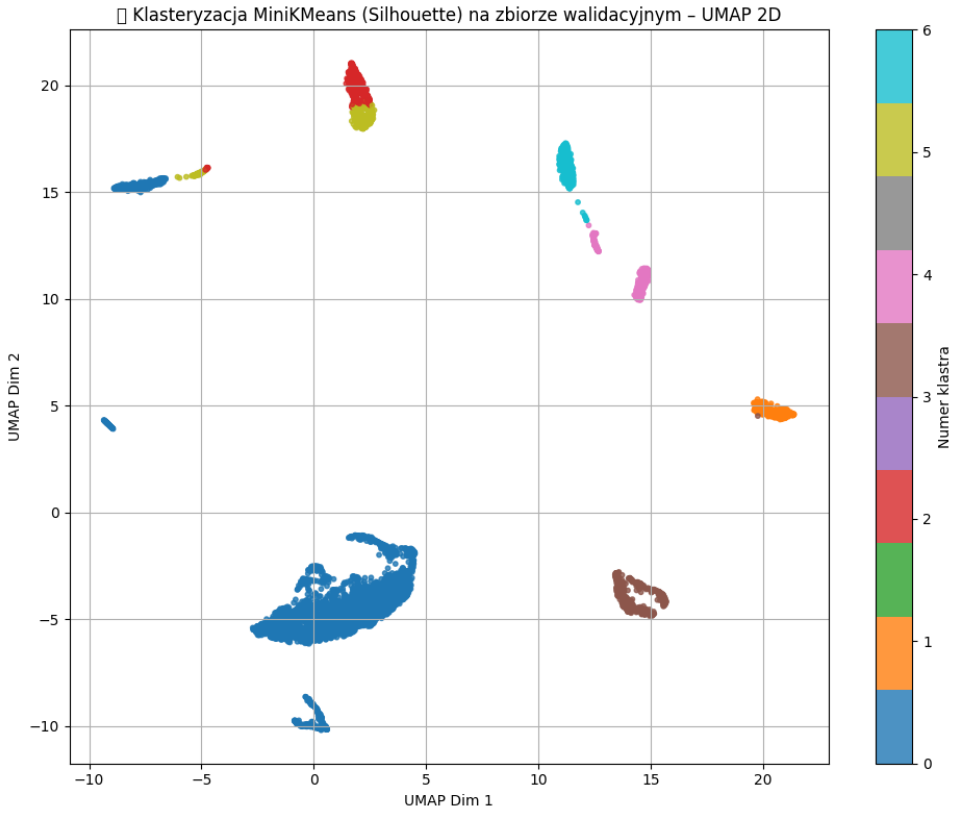
Model	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
Silhouette	**0.483**	14 279.464	1.083
Davies	0.357	**17 663.950**	**0.969**
Calinski	0.357	17 656.625	**0.969**
Default	0.323	16 700.642	1.120

Za najlpeszy uznaliśmy model zoptymalizowany względem Silhouette Score. Pomimo że inne modele osiągały nieco wyższe wyniki w metrykach Calinski-Harabasz oraz Davies-

Bouldin, to Silhouette Score najlepiej równoważy zwartą strukturę klastrów i ich separację. Model silhouette osiągnął:

- najwyższy Silhouette Score = 0.483, co świadczy o dobrej spójności wewnętrznej klastrów,
- nadal bardzo dobre wartości w pozostałych metrykach, nawet jeśli nie były najwyższe.

Dodatkowo, wizualizacja wyników na zbiorze walidacyjnym (przy pomocy UMAP) potwierdza, że model ten tworzy czytelne i wyraźnie rozdzielone klastry (wizualizacja poniżej). Dlatego to zalecamy wybranie wersji silhouette jako finalnego modelu do klasteryzacji.



Rysunek 1: Wizualizacja klastrów przy użyciu UMAP i modelu MiniKMeans (Silhouette)