Obliczenia Rozpowszechnione: Zaawansowany System Rekomendacji Piosenek

Michał Kaniowski Łukasz Kochańczyk Łukasz Kołodziej

February 5, 2025

Spis treści

Obliczenia Rozpowszechnione: Zaawansowany System Rekomendacji Piosenek **1**
Wstęp . . . 2
Opis Danych . . . 2
Inteligentny System (Rekomendacja Piosenek) . . . 2
Przetwarzanie Danych i Inżynieria Cech . . . 2
Logika Rekomendacji i Miara Cosinusowa . . . 3
Omówienie Kodu . . . 4
Symulator . . . 5
Przebieg Symulacji . . . 5
Agregacja i Ocena Rekomendacji . . . 6
Omówienie Kodu Symulatora . . . 6
Wizualizator . . . 8
Analiza Eksploracyjna Danych . . . 8
Szczegóły Implementacji . . . 8
Wnioski i Przyszłe Prace . . . 9

Obliczenia Rozpowszechnione: Zaawansowany System Rekomendacji Piosenek

Niniejszy dokument przedstawia szczegółowy raport dotyczący projektowania, wdrażania oraz ewaluacji systemu rekomendacji piosenek opartego na technikach rozproszonych obliczeń . Projekt składa się z trzech głównych komponentów:

1. **Inteligentny System (Rekomendacja Piosenek):**
   System rekomendacji oparty na zawartości, który wykorzystuje zarówno cechy audio, jak i metadane artysty, aby generować rekomendacje dla każdej piosenki osobno. Dzięki temu każdy gatunek czy styl obecny w liście odtwarzania jest odpowiednio reprezentowany.
2. **Symulator:**
   Ramy symulacyjne służące do walidacji systemu rekomendacji poprzez symulowanie warunków rzeczywistych – wybieranie losowych list odtwarzania, celowe usuwanie fragmentu utworów oraz ocena, jak dobrze system potrafi odzyskać lub dopasować usuniętą zawartość.
3. **Wizualizator:**
   Wizualizator, będący w zasadzie funkcją która, umożliwia prezentację wyników symulacji w czytelnej formie. Użytkownik otrzymuje podsumowanie wybranej listy odtwarzania, usuniętych

utworów, wygenerowanych rekomendacji oraz szczegółowy ranking rekomendowanych piosenek wraz z miarami podobieństwa.

Wstęp

Celem tego projektu jest stworzenie zaawansowanego systemu rekomendacji piosenek, który jest zarówno odporny na różnorodne gatunki muzyczne, jak i zdolny do obsługiwania rozbudowanych zbiorów danych. Zamiast opierać się na uśrednionej reprezentacji całej listy odtwarzania, system generuje rekomendacje dla każdej piosenki osobno, co pozwala na zachowanie różnorodności stylów muzycznych. Dodatkowo, komponent symulacyjny oraz wizualizator umożliwiają ocenę jakości rekomendacji w warunkach zbliżonych do rzeczywistych oraz prezentację wyników w przystępnej formie.

Dzięki zastosowaniu technik rozproszonych obliczeń system potrafi przetwarzać duże zbiory danych, zawierające miliony rekordów, zapewniając skalowalność i responsywność nawet w złożonych scenariuszach rekomendacyjnych.

Opis Danych

System korzysta z dwóch głównych źródeł danych:

1. **Główny Zbiór Danych Piosenek (`tracks_features.csv`):**
   - **Wielkość i Zakres:** Zawiera ponad milion rekordów.
   - **Cechy Audio:**
     - *Danceability, Energy, Loudness, Tempo:* Opisują właściwości rytmiczne i dynamiczne utworów.
     - *Acousticness, Speechiness:* Oddają subtelne cechy dźwiękowe.
     - *Duration i Year:* Dostarczają informacji o długości utworu oraz epoce, z której pochodzi.
   - **Metadane:**
     - *Name, Album, Artists, Artist IDs, Release Date:* Umożliwiają identyfikację i dodatkowe filtrowanie.
2. **Dane z List Odtwarzania/Do Symulacji (`playlists_filtered.csv`):**
   - Zawierają skomponowane listy odtwarzania z polami takimi jak `playlistname`, `trackname` oraz `artistname`.
   - Dane te są wykorzystywane głównie w eksperymentach symulacyjnych, które naśladują zachowania użytkowników i scenariusze testowe.

Dodatkowe notatniki eksploracyjne (`recommendation.ipynb` oraz `sim.ipynb`) zawierają szczegółowe analizy struktury danych, wizualizacje (histogramy, mapy ciepła korelacji, wyniki klasteryzacji) oraz eksperymenty, które pomogły w doborze cech wykorzystywanych w systemie.

Inteligentny System (Rekomendacja Piosenek)

Rdzeniem projektu jest **Inteligentny System**, wdrożony głównie w klasie `SongRecommender` (patrz: src/services/SongRecommender.py). Poniżej przedstawiono szczegółowy opis jego budowy, w tym przetwarzanie danych, inżynierię cech oraz logikę rekomendacyjną.

Przetwarzanie Danych i Inżynieria Cech

1. **Wczytywanie Danych i Normalizacja:**

- **Wczytywanie Danych:**
  Główny zbiór danych jest ładowany z pliku CSV do obiektu DataFrame biblioteki Pandas.

  ```
  self.data = pd.read_csv(csv_path)
  ```

- **Wybór Cech:**
  Wybierany jest podzbiór cech numerycznych, takich jak `danceability`, `energy`, `loudness`, `tempo`, `duration_ms`, `acousticness`, `speechiness` oraz `year`, które są wykorzystywane do obliczania podobieństwa.

- **Normalizacja:**
  Cechy numeryczne są normalizowane za pomocą `MinMaxScaler`, co powoduje, że wszystkie wartości mieszczą się w przedziale [0, 1]. Normalizacja ta jest kluczowa, aby uniknąć błędów wynikających z różnych zakresów wartości:

  ```
  scaler = MinMaxScaler()
  self.data[self.numeric_features] =
   ↪  scaler.fit_transform(self.data[self.numeric_features])
  ```

2. **Ekstrakcja Cech Artysty i Hashing:**

   - **Ekstrakcja Głównego Artysty:**
     Kolumna `artists`, która przechowuje listę artystów jako ciąg znaków (np. `"['Eminem']"`), jest przetwarzana przy użyciu funkcji `ast.literal_eval`. Zachowywany jest tylko pierwszy artysta, traktowany jako „główny artysta.”

     ```
     def _extract_main_artist(artist_str):
         try:
             artist_list = ast.literal_eval(artist_str)
             if isinstance(artist_list, list) and len(artist_list) > 0:
                 return artist_list[0]
         except Exception:
             pass
         return ""
     ```

   - **Hashing Cech Artysty:**
     Główny artysta jest przekształcany w wektor numeryczny o stałej długości za pomocą `FeatureHasher` z biblioteki scikit-learn. Dzięki temu metoda ta zamienia dane kategoryczne na dane liczbowe bez konieczności stosowania kosztownego kodowania one-hot, co umożliwia skalowanie na dużych zbiorach danych:

     ```
     hasher = FeatureHasher(n_features=self.n_artist_features,
      ↪  input_type='string')
     artist_hashed = hasher.transform(self.data['main_artist'].apply(lambda x:
      ↪  [x])).toarray()
     ```

3. **Łączenie Cech:**

   Znormalizowane cechy numeryczne oraz zhashowany wektor artysty są łączone w jeden wektor cech dla każdej piosenki. Taki połączony wektor jest wykorzystywany do dalszych obliczeń podobieństwa.

Logika Rekomendacji i Miara Cosinusowa

1. **Przetwarzanie Wejścia:**

   Dla każdej piosenki podanej jako wejście system pobiera odpowiadający jej wektor cech. Jeśli piosenka nie zostanie znaleziona w zbiorze danych, jest pomijana.

2. **Obliczanie Podobieństwa Cosinusowego:**
    - System oblicza podobieństwo cosinusowe między wektorem cech piosenki wejściowej a wektorami cech wszystkich piosenek w zbiorze danych.
    - Miara cosinusowa jest wykorzystywana, ponieważ dobrze mierzy kątową odległość między wektorami w przestrzeniach wielowymiarowych.
    - Obliczone wyniki podobieństwa są wykorzystywane do rankingu rekomendacji:

      ```
      similarity = cosine_similarity(input_vector, self.data[self.features])
      ```

3. **Generowanie Rekomendacji:**
    - **Wykluczenie Własnego Utworu:**
      Piosenka, dla której generowane są rekomendacje, jest usuwana z listy, aby nie rekomendować jej samej sobie.
    - **Sortowanie i Selekcja:**
      Pozostałe piosenki są sortowane według wartości podobieństwa (w kolejności malejącej), a następnie wybierane jest n najlepszych wyników.

      ```
      top_recs = filtered_data.sort_values(by='similarity',
      ↪  ascending=False).head(top_n)
      ```

    - **Struktura Wyniku:**
      Dla każdej piosenki wejściowej rekomendacja zawiera nazwę piosenki, szczegóły artysty oraz wartość podobieństwa.

Omówienie Kodu

Poniżej przedstawiono uproszczoną wersję metody rekomendującej:

```
def recommend(self, song_names, top_n=1):
    global_recommendations = []
    for song_name in song_names:
        print(f"Przetwarzanie rekomendacji dla: {song_name}")
        song_data = self.data[self.data['name'] == song_name]
        if song_data.empty:
            continue
        # Dla uproszczenia wykorzystujemy tylko pierwszy znaleziony rekord.
        song_data = song_data.iloc[[0]]
        input_vector = song_data[self.features].values.reshape(1, -1)

        # Obliczanie podobieństwa cosinusowego.
        similarity = cosine_similarity(input_vector, self.data[self.features])
        temp_data = self.data.copy()
        temp_data['similarity'] = similarity[0]

        # Wykluczenie piosenki wejściowej z rekomendacji.
        filtered_data = temp_data[temp_data['name'] != song_name]

        # Wybór top_n najbardziej podobnych piosenek.
        top_recs = filtered_data.sort_values(by='similarity',
↪   ascending=False).head(top_n)
        rec_list = []
        for _, row in top_recs.iterrows():
```

```
            rec_list.append({
                'recommended_song': row['name'],
                'artists': row['artists'],
                'similarity': row['similarity']
            })

        global_recommendations.append({
            'input_song': song_name,
            'recommendations': rec_list
        })
    return global_recommendations
```

Metoda ta została zaprojektowana do generowania spersonalizowanych rekomendacji dla każdej piosenki na podstawie połączonej przestrzeni cech.

Symulator

Komponent **Symulator** został stworzony w celu przetestowania wydajności i odporności systemu rekomendacji w warunkach zbliżonych do rzeczywistych. W tej sekcji opisano przebieg symulacji, metody agregacji rekomendacji oraz kryteria oceny.

Przebieg Symulacji

1. **Losowy Wybór Listy Odtwarzania:**
   - Losowo wybierana jest lista odtwarzania z danych symulacyjnych (`playlists_filtered.csv`). Proces ten odbywa się poprzez losowe próbkowanie, co zapewnia testowanie różnych list przy kolejnych uruchomieniach.

     ```
     random_playlist = self.sim_data['playlistname'].sample(n=1).iloc[0]
     playlist_songs_df = self.sim_data[self.sim_data['playlistname'] ==
      ↪  random_playlist]
     ```

2. **Proces Usuwania Utworów:**
   - Z wybranej listy losowo usuwana jest określona liczba utworów (`removal_count`).
   - Usunięcie to symuluje sytuacje takie jak utrata danych, selektywna kuracja czy celowe pominięcie utworów.

     ```
     removed_songs =
      ↪  playlist_songs_df['trackname'].sample(n=removal_count).tolist()
     reduced_playlist_df =
      ↪  playlist_songs_df[~playlist_songs_df['trackname'].isin(removed_songs)]
     ```

   - Aby ograniczyć koszty obliczeniowe, jeśli pozostała lista zawiera więcej niż 30 utworów, losowo wybierany jest podzbiór 30 utworów.

3. **Generowanie Rekomendacji na Podstawie Pozostałych Utworów:**
   - Pozostałe utwory (tzw. „utwory wejściowe") są przekazywane do systemu rekomendacji.
   - Dla każdego z tych utworów system generuje top n rekomendacji.

     ```
     input_song_names = reduced_playlist_df['trackname'].tolist()
     global_recs = self.recommend(input_song_names, top_n=top_n)
     ```

Agregacja i Ocena Rekomendacji

1. **Łączenie Rekomendacji:**
   - Rekomendacje wygenerowane dla każdego utworu wejściowego są łączone w unikalny zbiór.
   - Taka agregacja zapobiega powtarzaniu się rekomendacji i tworzy skonsolidowaną listę.
2. **Obliczanie Podobieństwa Cosinusowego do Usuniętych Utworów:**
   - Dla każdej rekomendowanej piosenki system oblicza podobieństwo cosinusowe pomiędzy jej wektorem cech a wektorami cech usuniętych utworów.
   - Maksymalna wartość podobieństwa uzyskana dla danej rekomendacji (spośród porównań z wszystkimi usuniętymi utworami) służy jako wskaźnik, jak dobrze rekomendacja odpowiada usuniętej zawartości.

     ```
     similarity_matrix = cosine_similarity(rec_features, removed_features)
     aggregated_sim = similarity_matrix.max(axis=1)
     ```
3. **Ranking Końcowych Rekomendacji:**
   - Rekomendowane utwory są sortowane według zagregowanych wyników podobieństwa.
   - Top n utworów z posortowanej listy stanowią najlepsze rekomendacje pod względem podobieństwa do usuniętych utworów.
   - Dodatkowo, system rejestruje, do którego usuniętego utworu dana rekomendacja osiągnęła najwyższą wartość podobieństwa.

     ```
     max_indices = similarity_matrix.argmax(axis=1)
     removed_song_names_arr = removed_subset['name'].values
     compared_to = [removed_song_names_arr[idx] for idx in max_indices]
     ```

Omówienie Kodu Symulatora

Poniżej znajduje się fragment metody `simulate`, który ilustruje przebieg symulacji:

```
def simulate(self, top_n=5, removal_count=5):
    if self.sim_data is None:
        raise ValueError("Dane symulacyjne nie zostały wczytane. Proszę najpierw
        ↪  załadować dane symulacyjne.")

    # Krok 1: Wybierz losowo listę odtwarzania.
    random_playlist = self.sim_data['playlistname'].sample(n=1).iloc[0]
    playlist_songs_df = self.sim_data[self.sim_data['playlistname'] ==
↪   random_playlist]

    # Krok 2: Losowo usuń utwory z listy.
    removed_songs = playlist_songs_df['trackname'].sample(n=removal_count).tolist()

    # Krok 3: Użyj pozostałych utworów jako wejścia (ograniczenie do 30 utworów).
    reduced_playlist_df =
↪   playlist_songs_df[~playlist_songs_df['trackname'].isin(removed_songs)]
    if len(reduced_playlist_df) > 30:
        reduced_playlist_df = reduced_playlist_df.sample(30)
    input_song_names = reduced_playlist_df['trackname'].tolist()
```

```
    # Krok 4: Wygeneruj rekomendacje dla każdego pozostałego utworu.
    global_recs = self.recommend(input_song_names, top_n=top_n)

    # Krok 5: Połącz unikalne rekomendacje.
    recommended_songs = {}
    for rec in global_recs:
        for song in rec['recommendations']:
            song_name = song['recommended_song']
            if song_name not in recommended_songs:
                recommended_songs[song_name] = song
    recommended_song_names = list(recommended_songs.keys())

    # Krok 6: Oblicz podobieństwo cosinusowe między rekomendowanymi utworami a
     ↪ usuniętymi utworami.
    removed_subset = self.data[self.data['name'].isin(removed_songs)]
    if removed_subset.empty:
        raise ValueError("Żaden z usuniętych utworów nie został znaleziony w głównym
         ↪ zbiorze danych.")
    rec_subset = self.data[self.data['name'].isin(recommended_song_names)]
    if rec_subset.empty:
        raise ValueError("Żadne rekomendowane utwory nie zostały znalezione w
         ↪ głównym zbiorze danych.")

    rec_features = rec_subset[self.features].values
    removed_features = removed_subset[self.features].values
    similarity_matrix = cosine_similarity(rec_features, removed_features)
    aggregated_sim = similarity_matrix.max(axis=1)
    max_indices = similarity_matrix.argmax(axis=1)
    removed_song_names_arr = removed_subset['name'].values
    compared_to = [removed_song_names_arr[idx] for idx in max_indices]

    # Dołącz zagregowane podobieństwo oraz informację referencyjną.
    rec_subset = rec_subset.copy()
    rec_subset['aggregated_similarity'] = aggregated_sim
    rec_subset['compared_to'] = compared_to

    # Sortuj i wybierz najlepsze rekomendacje.
    top_rec_df = rec_subset.sort_values(by='aggregated_similarity',
↪   ascending=False).head(top_n)
    top_recommendations = []
    for _, row in top_rec_df.iterrows():
        top_recommendations.append({
            'recommended_song': row['name'],
            'artists': row['artists'],
            'aggregated_similarity': row['aggregated_similarity'],
            'compared_to': row['compared_to']
        })

    # Zwróć szczegółowy wynik symulacji.
    return {
        'playlist': random_playlist,
        'removed_songs': removed_songs,
```

```
        'input_songs': input_song_names,
        'global_recommendations': global_recs,
        'top_recommendations': top_recommendations
    }
```

Powyższa metoda nie tylko waliduje logikę rekomendacji, ale również dostarcza informacji o tym, jak dobrze rekomendacje odpowiadają usuniętej zawartości, umożliwiając tym samym rygorystyczną ocenę wydajności systemu.

Wizualizator

System zawiera dodatkowy komponent – **Wizualizator**, który jest realizowany poprzez funkcję `main()` w pliku `simulate2.py`. Wizualizator odpowiada za prezentację wyników symulacji w czytelnej formie. Główne zadania wizualizatora obejmują:

- Wyświetlenie nazwy wybranej listy odtwarzania.
- Prezentację listy usuniętych utworów oraz utworów wejściowych użytych do generowania rekomendacji.
- Wypisanie szczegółowych wyników globalnych rekomendacji dla poszczególnych piosenek.
- Przedstawienie rankingu rekomendowanych utworów, które zostały posortowane według zagregowanych wyników podobieństwa, wraz z informacją, z którym usuniętym utworem każda rekomendacja jest najbardziej zbliżona.

Dzięki tej funkcji użytkownik może szybko uzyskać wizualne podsumowanie działania systemu, co jest szczególnie przydatne podczas testowania oraz prezentacji wyników.

Analiza Eksploracyjna Danych

Przed implementacją systemu rekomendacji przeprowadzono szczegółową analizę eksploracyjną (EDA) danych przy użyciu notatników Jupyter. Kluczowe kroki analizy obejmowały:

- **Profilowanie Danych:**
  Analiza typów danych, wykrywanie wartości pustych oraz statystyki opisowe.
- **Wizualizacje:**
  - Histogramy cech audio, takich jak `danceability`, `energy`, `loudness`.
  - Mapy ciepła korelacji, które pokazują zależności między poszczególnymi cechami.
- **Analiza Klasteryzacji:**
  Zastosowanie metody KMeans do identyfikacji naturalnych grup w zbiorze danych na podstawie cech audio. Dzięki temu można było potwierdzić, że wybrane cechy mają istotną wariancję między różnymi stylami muzycznymi.
- **Eksperymenty z TF-IDF:**
  W jednym z notatników zastosowano wektoryzację TF-IDF na skombinowanych metadanych utworów, a następnie obliczono podobieństwo cosinusowe. Chociaż metoda ta miała charakter eksperymentalny, dostarczyła cennych informacji przy projektowaniu głównego systemu.

Szczegóły Implementacji

Projekt został zorganizowany w sposób modułowy:

- **Warstwa API (main.py):**
  Usługa FastAPI umożliwia udostępnienie systemu rekomendacji jako serwisu webowego. Dzięki temu zewnętrzni użytkownicy mogą wysyłać żądania rekomendacyjne za pomocą REST API.
- **Rdzeń Systemu Rekomendacji:**
  Klasa `SongRecommender` (zlokalizowana w `src/services/SongRecommender.py`) odpowiada za przetwarzanie danych, inżynierię cech oraz obliczanie podobieństwa.
- **Moduł Symulacyjny:**
  Skrypt `simulate2.py` wykorzystuje system rekomendacji do testowania jego wydajności w realistycznych scenariuszach list odtwarzania. Moduł ten wywołuje metodę `simulate` klasy `SongRecommender` w celu oceny rekomendacji względem usuniętych utworów.
- **Wizualizator:**
  Funkcja `main()` w pliku `simulate2.py` pełni rolę wizualizatora, prezentując wyniki symulacji w czytelny sposób.
- **Modele Wstępnie Wytrenowane:**
  Niektóre modele, takie jak macierze podobieństwa cosinusowego czy modele TF-IDF, są wstępnie trenowane i zapisywane za pomocą joblib. Przyspiesza to proces rekomendacji i zmniejsza czas obliczeń podczas fazy inferencji.
- **Notatniki Wspomagające:**
  Notatniki (`recommendation.ipynb` oraz `sim.ipynb`) dokumentują proces analizy eksploracyjnej oraz eksperymentalne podejścia, które wpłynęły na projekt systemu.

Wnioski i Przyszłe Prace

Przedstawiony system rekomendacji piosenek demonstruje efektywne podejście oparte na zawartości, łączące znormalizowane cechy audio z metadanymi artystów przetworzonymi metodą hashującą. Dzięki generowaniu rekomendacji dla każdej piosenki osobno system zachowuje różnorodność list odtwarzania, zapewniając, że każdy styl muzyczny jest odpowiednio reprezentowany.

Główne Wnioski: - Solidne Przetwarzanie Danych:
Normalizacja oraz hashing cech umożliwiają przetwarzanie różnorodnych i wielowymiarowych danych.

- **Szczegółowa Rekomendacja:**
  Generowanie rekomendacji dla każdej piosenki z osobna, zamiast uśredniania całej listy, gwarantuje uzyskanie zróżnicowanego wyniku.
- **Realistyczna Symulacja:**
  Moduł symulacyjny zapewnia rygorystyczne testowanie systemu, symulując warunki zbliżone do rzeczywistych, co umożliwia ocenę jakości rekomendacji.

Przyszłe Kierunki Rozwoju: - Modele Hybrydowe:
Integracja technik filtracji kolaboratywnej z obecnym podejściem opartym na zawartości, aby lepiej uwzględniać zachowania użytkowników.

- **Rozszerzenie Zestawu Cech:**
  Włączenie dodatkowych cech audio oraz kontekstowych, takich jak analiza sentymentu tekstów piosenek, wykrywanie nastroju czy metryki zaangażowania użytkowników.
- **Skalowalność:**
  Wykorzystanie frameworków rozproszonych obliczeń oraz architektur przetwarzania w czasie rzeczywistym w celu obsługi większych zbiorów danych i szybszej inferencji.

- **Integracja Opinii Użytkowników:**
  Wdrożenie mechanizmów zbierania opinii na temat rekomendacji i ciągłe ulepszanie algorytmu na ich podstawie.

Projekt stanowi solidną podstawę do dalszych badań oraz praktycznych zastosowań inteligentnych systemów rekomendacyjnych w ramach *obliczeń rozpowszechnionych*.