# Analisis Perubahan Pola Leksikal pada Lirik Lagu Populer di Amerika Serikat (1960–2025) Menggunakan *Singular Value Decomposition* (SVD)

Eliana Natalie Widjojo - 13524116[1,2]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
[1]13524116@std.stei.itb.ac.id, [2]elianawidjojo@gmail.com

***Abstract*—Studi ini menganalisis perubahan pola leksikal pada lirik lagu populer di Amerika Serikat dari tahun 1960 hingga 2025. SVD digunakan untuk mengidentifikasi struktur laten yang merepresentasikan tren leksikal dominan serta perubahannya tiap tahun. Melalui analisis nilai singular dan vektor singular, penelitian ini mengidentifikasi pergeseran penggunaan kosakata, tema, dan gaya bahasa yang mencerminkan perubahan sosial budaya seiring zaman. Penelitian ini juga menunjukkan bahwa SVD merupakan pendekatan aljabar linear yang efektif untuk menganalisis perubahan pola leksikal dalam data teks berskala besar.**

***Keywords*—Analisis leksikal, dekomposisi nilai singular, lirik lagu, musik populer**

## I. Pendahuluan

Musik adalah seni mengkombinasi bunyi vokal atau instrumental untuk keindahan atau ekspresi emosional [1]. Musik populer adalah musik yang ditulis dan dipasarkan dengan tujuan mencapai distribusi massal, terutama dalam bentuk rekaman [2]. Musik populer umumnya memiliki ciri khas dekat dengan pengalaman dan perasaan masyarakat. Melodi, tema, dan terutama lirik dapat dipahami dan dirasakan oleh pendengar dari berbagai latar belakang, baik secara emosional maupun budaya.

*Single* adalah sebuah rekaman musik yang hanya memiliki satu lagu utama dan kadang satu atau lebih lagu tambahan. *Single* juga dapat berarti sebuah lagu dari rekaman yang memiliki banyak lagu [3].

*Billboard* adalah publikasi musik terkemuka di dunia yang berasal dari Amerika Serikat. "Billboard Hot 100" adalah artikel yang menampilkan daftar 100 *single* terpopuler di Amerika Serikat setiap minggu, dengan posisi paling atas menunjukkan musik yang paling populer berdasarkan penjualan, *online streaming*, dan pemutaran di radio. Sejak tahun 1960, *Billboard* juga merilis daftar berisi 100 *single* paling populer dalam tahun tersebut yang dinamakan "*Billboard Year-End Hot 100 Chart*". Tabel ini dirilis setiap akhir tahun, yaitu sekitar awal Desember.

Musik selalu mencerminkan kondisi sosial dan budaya di masanya. Salah satu komponen terpenting di dalam musik adalah lirik. Lirik tidak hanya menjadi pendamping melodi, tapi juga sebagai sarana komunikasi pesan, cerita, dan emosi kepada para pendengar. Oleh karena itu, lirik musik populer dapat digunakan untuk menganalisis tema, tren, dan gaya bahasa yang juga populer di masa itu.

*Singular value decomposition* (SVD) adalah cabang dari aljabar linear dan geometri yang memanfaatkan *eigenvalue* dan *eigenvector*. SVD adalah proses penguraian matriks menjadi tiga komponen, yaitu U, Σ, dan V. SVD digunakan untuk mencari struktur tersembunyi pada data, mengurangi noise, dan mengukur similaritas antar lagu dengan efektif.

*Latent semantic analysis* (LSA) adalah teknik menganalisis hubungan antara kata dan dokumen. LSA memanfaatkan konsep SVD dan dapat digunakan untuk mencari pola leksikal pada musik, mengukur kemiripan antar lagu antar dekade, dan melacak pergeseran leksikal antar dekade.

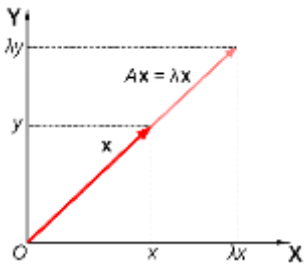
## II. Dasar Teori

### A. Eigenvalue dan Eigenvector

*Eigenvector* (vektor eigen) adalah vektor tidak-nol $x$ di $R^n$ dari sebuah matriks $A$ berukuran $n \times n$, jika $Ax$ sama dengan perkalian suatu skalar $\lambda$ dengan $x$.

$$Ax = \lambda x \tag{1}$$

Skalar $\lambda$ disebut sebagai e*igenvalue* (nilai eigen), sementara $x$ dinamakan vektor eigen yang berkoresponden dengan $\lambda$. Nilai eigen menyatakan nilai karakteristik dari sebuah matriks yang berukuran $n \times n$. Vektor eigen $x$ adalah vektor yang jika dikalikan dengan sebuah matriks, hanya berubah skalanya. Arah dari vektor tetap sama jika nilai eigen positif atau berlawanan jika nilai eigen negatif.




Gambar 1. Ilustrasi hubungan vektor eigen dan nilai eigen
Sumber: https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors

Nilai eigen pada matriks dapat dihitung dengan rumus

$$det(I\lambda - A)x = 0 \tag{2}$$

Untuk menemukan vektor eigen, cari $x \neq 0$ yang memenuhi

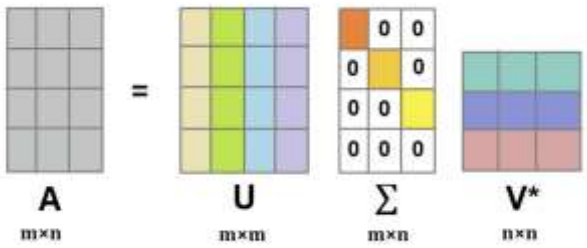
$$(I\lambda - A)x = 0 \tag{3}$$

### *B. Singular Value Decomposition (SVD)*

SVD menguraikan matriks $A$ berukuran $m \times n$ menjadi $U$, Σ, dan $V$. SVD dapat ditulis sebagai

$$A = U\Sigma V^T \tag{4}$$

Pada persamaan (1), $U$ adalah matriks ortogonal berukuran $m \times m$, $V$ adalah matriks ortogonal berukuran $n \times n$, dan Σ adalah matriks $m \times n$ dengan elemen diagonal utamanya berupa *singular value* dari $A$, sedangkan elemen lainnya bernilai nol.




Gambar 2. Ilustrasi *singular value decomposition*
Sumber: https://www.askpython.com/python/examples/singular-value-decomposition

Matriks ortogonal adalah matriks persegi di mana vektor-vektor kolom dan barisnya saling tegak lurus dan bersifat ortonormal. Ortonormal berarti setiap vektor memiliki panjang satuan.

Berikut adalah langkah-langkah untuk menghitung SVD:

1. *Transpose* $A$ menjadi $A^T$, lalu kalikan $A$ dengan $A^T$.
2. Menghitung *eigenvalue* dan *eigenvector* dari $A^T A$.
3. Menghitung *singular value* $\sigma_i$ menggunakan rumus

$$\sigma_i = \sqrt{eigenvalue\ ke - i\ dari\ A^T A} \tag{5}$$

4. Normalisasi *eigenvector* $v_i$ dari $A^T A$ untuk mendapatkan vektor singular kanan

$$\hat{v}_i = \frac{v_i}{\|v_i\|} \tag{6}$$

5. Menyusun matriks $V$ dengan $\hat{v}_i$ sebagai kolom-kolomnya.
6. *Transpose* matriks $V$ menjadi $V^T$.
7. Menentukan vektor singular kiri $u_i$ menggunakan rumus

$$u_i = \frac{Av_i}{\|Av_i\|} = \frac{1}{\sigma_i} Av_i \tag{7}$$

8. Normalisasi vektor-vektor $u_i$ dengan cara

$$\hat{u}_i = \frac{u_i}{\|u_i\|} \tag{8}$$

9. Menyusun matriks $U$ adalah $\hat{u}_i$ sebagai kolom-kolomnya.
10. Membuat matriks Σ berukuran $m \times n$ dengan nilai-nilai diagonalnya berupa $\sigma_i$ tidak nol dengan susunan dari yang terbesar ke terkecil. Isi nilai-nilai yang masih kosong dengan angka 0.

### *C. Latent Semantic Analysis (LSA)*

LSA adalah metode analisis teks dan informasi yang digunakan untuk menemukan hubungan laten antara kata dan dokumen. Pada umumnya, LSA digunakan untuk *information retrieval*, *text mining*, dan *natural language processing*.

Berikut adalah langkah-langkah untuk melakukan LSA dengan sederhana:

1. Mengumpulkan dan mempersiapkan dokumen dengan metode *preprocessing*. *Preprocessing* terdiri dari tokenisasi (memisahkan kata), normalisasi (membuat kata menjadi *lowercase*), *stemming*/lematisasi (mengubah kata menjadi bentuk dasar), dan penghapusan *stopword* (menghapus kata-kata umum).
2. Membuat matriks *term-document*. Baris dari matriks mewakili kata (*term*), sementara kolom dari matriks mewakili dokumen (*document*). Elemen dari matriks tersebut berupa frekuensi kata (TF) atau frekuensi kata – frekuensi dokumen invers (TF-IDF). TF digunakan jika fokus terhadap kata yang paling sering muncul di dokumen. Sementara TF-IDF digunakan jika fokus terhadap kata-kata penting dan unik. Berikut adalah rumus-rumus yang akan digunakan

$$TF(t,d) = \frac{frekuensi\ term\ t\ dalam\ dokumen\ d}{total\ kata\ dalam\ dokumen\ d} \tag{9}$$

$$DF(t) = jumlah\ dokumen\ di\ mana\ t\ muncul \tag{10}$$

$$IDF(t) = \log_{10} \frac{N}{1 + DF(t)} \tag{11}$$

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t) \tag{12}$$

3. Menguraikan matriks *term-document* menjadi tiga matriks menggunakan SVD.
4. Mereduksi dimensi dengan membentuk

$$X_k \approx U_k \Sigma_k {V_k}^T \tag{13}$$

$k$ adalah nilai singular terbesar, sementara $X$ adalah matriks *term-document*. Tujuan langkah ini adalah untuk mereduksi *noise*, mengungkap hubungan semantik tersembunyi, dan mempermudah analisis.

5. Menghitung similaritas antara dokumen atau kata menggunakan *cosine similarity*, yaitu

$$sim(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\| \|d_j\|} \tag{14}$$

6. Menggunakan representasi *term–concept* dan *document–concept* untuk menemukan dokumen yang relevan, mengelompokkan dokumen, dan mengekstrak topik dan konsep tersembunyi.

## III. Metode Penelitian

Penelitian dilakukan secara kuantitatif, menggunakan metode analisis numerik dan statistik. Tujuan dari penelitian ini adalah menemukan musik yang paling mirip antar tahun *(cross year similarity)* untuk melihat perubahan pola musik tahun ke tahun. Selain itu, penelitian ini juga membandingkan tingkat kemiripan lagu dalam satu dekade dan antar dekade, serta mendeteksi pergeseran leksikal antar dekade. Data penelitian dibatasi dari tahun 1960 hingga 2025, dengan hanya 20 *single* teratas yang tercatat pada "*Billboard Year-End Hot 100 Chart*" untuk setiap tahun. Hal ini dilakukan untuk mengurangi *noise* yang dapat mengganggu hasil penelitian. Apabila terdapat *single* yang memuat dua lagu pada satu CD, hanya lirik lagu pertama yang dianalisis untuk menjaga konsistensi data. Jika *single* tidak memiliki lirik lagu, maka data akan dilewatkan. Program dibuat dalam bahasa Python.

## IV. Implementasi

### A. Pengumpulan Data

Penelitian dimulai dengan membuat tabel yang berisi tahun, peringkat, judul, dan nama musisi pada setiap 20 *single* teratas yang tercatat pada "*Billboard Year-End Hot 100 Chart*" dalam format .xslx menggunakan Microsoft Excel.

Tabel 1. Lima Baris Pertama dari metadata.xslx

| *Year* | *Rank* | *Title* | *Artist* |
|---|---|---|---|
| 1960 | 1 | "Theme from A Summer Place" | Percy Faith |
| 1960 | 2 | "He'll Have to Go" | Jim Reeves |
| 1960 | 3 | "Cathy's Clown" | The Everly Brothers |
| 1960 | 4 | "Running Bear" | Johnny Preston |
| 1960 | 5 | "Teen Angel" | Mark Dinning |

Kemudian, lirik lagu dikumpulkan secara otomatis menggunakan Genius API. Lagu yang tidak memiliki lirik akan dilewatkan dan ditampung *file ID*-nya dalam sebuah *file* dengan format .csv, sehingga data tetap terdokumentasi dan dapat ditinjau kembali jika diperlukan. Berikut adalah kode yang digunakan dalam pengumpulan lirik lagu.

```
import lyricsgenius
import pandas as pd
import os
import re
import time

# Cleaning
def clean_title(title: str) -> str:
    title = title.split('/')[0]

    title = title.replace('"', '').strip()

    title = re.sub(r'\s+', ' ', title)

    return title.strip()

def clean_artist(artist: str) -> str:
    artist = artist.replace('"', '').strip()
    artist = re.sub(
        r'(featuring|feat\.|with|and|&).*',
        '',
        artist,
        flags=re.I
    )
    artist = re.sub(r'\s+', ' ', artist)
    return artist.strip()

def clean_lyrics(text: str) -> str:
     text = re.sub(r'You might also like.*', '',
text, flags=re.DOTALL)
    text = re.sub(r'\n{2,}', '\n', text)
    return text.strip()

# Configuration
GENIUS_TOKEN =
 "5VyYKj1RHfKJBtGInW_e5MGQ0_FuyDYhSsVMxxJATtHI2ek
  P1SyP5o5CEIsJoaOZ"
METADATA_FILE = "../data/metadata.xlsx"
OUTPUT_DIR = "../data/lyrics"
SLEEP_TIME = 1.0

# Genius Client
genius = lyricsgenius.Genius(
    GENIUS_TOKEN,
    remove_section_headers=True,
    skip_non_songs=True,
    excluded_terms=["(Remix)", "(Live)"],
    timeout=15
)

# Load Metadata
if METADATA_FILE.endswith(".xlsx"):
    df = pd.read_excel(METADATA_FILE)
else:
    df = pd.read_csv(METADATA_FILE)

os.makedirs(OUTPUT_DIR, exist_ok=True)

failed = []

# Download lyrics
for _, row in df.iterrows():
    year = int(row["Year"])
    rank = int(row["Rank"])
    raw_title = str(row["Title"])
    raw_artist = str(row["Artist"])

    title = clean_title(raw_title)
    artist = clean_artist(raw_artist)

    file_id = f"{year}_{rank:03d}"

    year_dir = os.path.join(OUTPUT_DIR, str(year))
    os.makedirs(year_dir, exist_ok=True)
```

```
    out_path = os.path.join(year_dir,
               f"{file_id}.txt")

    if os.path.exists(out_path):
        continue

    try:
        song = genius.search_song(title, artist)
        if song and song.lyrics:
            lyrics = clean_lyrics(song.lyrics)
            with open(out_path, "w", encoding="utf-
                      8") as f:
                f.write(lyrics)
        else:
            failed.append(file_id)

    except Exception:
        failed.append(file_id)

    time.sleep(SLEEP_TIME)

# Log Gagal
if failed:
    pd.DataFrame({"file_id":
    failed}).to_csv
    ("../data/failed_downloads.csv", index=False)
    print(f"{len(failed)} lagu gagal. Lihat
          failed_downloads.csv")
else:
    print("Semua lirik berhasil diunduh.")
```

*B. Preprocessing*

```
import re

# 1. Tokenisasi & 2. Normalisasi
def tokenize(text):
    text = text.lower()
    text = re.sub(r"[']", "", text)
    text = re.sub(r'[^a-z\s]', ' ', text)
    tokens = text.split()

    return tokens

# 3. Stemming/Lematisasi
def simple_stem(word):
    suffixes = [
        "ization", "isation",
        "ational", "fulness", "ousness",
        "iveness", "ational", "tional",
        "lessly", "lessly", "lessly",
        "lessly", "ologist", "ologies",
        "lessly", "lessly",
        "ically", "ically",
        "ing", "ers", "ies", "ied",
        "ness", "ment", "able", "ible",
        "less", "tion", "sion", "ment",
        "ally", "edly", "ward", "wards",
        "est", "ous", "ive", "ish",
        "ed", "ly", "er", "es", "s"
    ]

    for suffix in suffixes:
        if word.endswith(suffix) and len(word) >
        len(suffix) + 2:
            return word[:-len(suffix)]
    return word

# 4. Penghapusan Stopword
STOPWORDS = { # set kata2 yang umum
    "a","an","the","is","are","and","or","to",
    "in","of","that","it","on","for","with","as",
    "by","this","was","be","from","at","which",
    "but","not","have","has","had","were","will",
    "their","its","oh", "na", "la", "yeah"
}

def remove_stopwords(tokens):
    return [t for t in tokens if t not in STOPWORDS]

# Process document
def process_document(text, use_stem=False):
    tokens = tokenize(text)

    if use_stem:
        tokens = [simple_stem(t) for t in tokens]

    tokens = remove_stopwords(tokens)

    return tokens
```

Kode di atas digunakan untuk memproses lirik lagu agar siap dianalisis secara komputasional. Proses dimulai dengan tokenisasi dan normalisasi. Kemudian, *stemming*/lematisasi sederhana dilakukan untuk mengubah kata menjadi bentuk dasarnya. Setelah itu, penghapusan *stopword* dilakukan untuk fokus ke kata-kata penting. Kata-kata pada *array* STOPWORDS berisi kata-kata yang umum digunakan dalam lagu berbahasa Inggris.

*C. Pembentukan Matriks Term-Document*

```
import numpy as np
from preprocessing import process_document

def tf_idf(texts, use_stem=True):
    docs_tokens = [
        process_document(t, use_stem=use_stem)
        for t in texts
    ]

    vocab = sorted(set(token for doc in docs_tokens
            for token in doc))
    vocab_index = {w: i for i, w in
                  enumerate(vocab)}
    num_terms = len(vocab)
    num_docs = len(docs_tokens)
    A = np.zeros((num_terms, num_docs), dtype=float)

    for j, tokens in enumerate(docs_tokens):
        for token in tokens:
            A[vocab_index[token], j] += 1

    col_sums = A.sum(axis=0)
    valid = col_sums > 0
    A = A[:, valid]
    col_sums = col_sums[valid]
    num_docs = A.shape[1]
    tf = A / col_sums
    df = np.count_nonzero(A > 0, axis=1)
    idf = np.log10(num_docs / (1 + df))
    X = tf * idf[:, np.newaxis]

    return X, valid
```

Pertama, *preprocessing* dilakukan pada setiap dokumen. Selanjutnya, *vocabulary* dibuat dari semua token unik dalam korpus dan matriks *term-document* (matriks *A*) dibuat dengan ukuran jumlah *terms* dikali jumlah *document*, di mana setiap elemen mewakili frekuensi token dalam dokumen. Kolom yang kosong (*single* yang tidak memiliki lirik) dihapus, kemudian dihitung TF dengan membagi setiap elemen matriks dengan jumlah kata dalam dokumen. Kemudian, dihitung DF dan IDF sesuai dengan rumus. Matriks TF dikalikan dengan IDF untuk menghasilkan matriks TF-IDF (matriks *X*). Fungsi yang dibuat mengembalikan matriks TF-IDF dan *boolean array* valid untuk menandai dokumen yang tidak kosong.

*D. Singular Value Decomposition (SVD)*

```
import numpy as np

def power_iteration(A, num_iters=100):
    n = A.shape[0]
```

```
        b = np.random.rand(n)
        b /= np.linalg.norm(b)

        for _ in range(num_iters):
            b = A @ b
            norm = np.linalg.norm(b)
            if norm == 0:
                break
            b = b / norm

        eigenvalue = b.T @ A @ b
        return eigenvalue, b

    def svd(A, k):
        A = np.array(A, dtype=float)

        ATA = A.T @ A
        eigenvalues = []
        eigenvectors = []

        A_copy = ATA.copy()
        for _ in range(k):
            eigval, eigvec = power_iteration(A_copy)
            eigenvalues.append(eigval)
            eigenvectors.append(eigvec)
            A_copy = A_copy - eigval * np.outer(eigvec,
                     eigvec)

        #sorting
        eigenvalues = np.array(eigenvalues)
        eigenvectors = np.column_stack(eigenvectors)

        idx = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idx]
        eigenvectors = eigenvectors[:, idx]

        V = eigenvectors
        S = np.diag(np.sqrt(eigenvalues))

        eps = 1e-10
        S_inv = np.diag(1 / (np.sqrt(eigenvalues) +
                eps))
        U = A @ V @ S_inv

      return U, S, V.T
```

Program ini adalah implementasi SVD secara manual. Fungsi power_iteration adalah menghitung *eigenvector* dan *eigenvalue* terbesar dari sebuah matriks. Fungsi SVD berfungsi untuk menguraikan matriks $A$ menjadi $U$, $\Sigma$, dan $V$. SVD dilakukan sesuai dengan rumus yang ada.

### *E. Perhitungan Similaritas*

```
import numpy as np
from read_mapper import load_texts
from term_document import tf_idf
from svd import svd

texts, labels = load_texts()
X, valid = tf_idf(texts)
labels = [labels[i] for i in range(len(labels)) if
          valid[i]]
doc_names = [l["file"] for l in labels]
years = np.array([l["year"] for l in labels])
assert X.shape[1] == len(labels)
U, S, Vt = svd(X, k=100)
D = Vt.T @ S
norms = np.linalg.norm(D, axis=1, keepdims=True)
D_norm = D / np.maximum(norms, 1e-10)
sim_matrix = D_norm @ D_norm.T

# Fungsi menemukan lagu paling mirip lintas tahun
def most_similar_cross_year(target_year):
    idx_src = np.where(years == target_year)[0]
    idx_oth = np.where(years != target_year)[0]

    results = []
    for i in idx_src:
        sims = sim_matrix[i, idx_oth]
        j = idx_oth[np.argmax(sims)]
        results.append({
            "source_doc": doc_names[i],
            "source_year": target_year,
            "most_similar_doc": doc_names[j],
            "target_year": int(years[j]),
            "similarity": float(sim_matrix[i, j])
        })
    return results

# Fungsi konversi tahun ke dekade, khusus 2021-2025
def year_to_decade(year):
    if year >= 2021:
        return 2025  # Dekade khusus 2021-2025
    return (year // 10) * 10

decades = np.array([year_to_decade(y) for y in
          years])

# Rata-rata similarity dalam dekade dan lintas
dekade
def average_similarity_by_decade():
    within = []
    cross = []
    n = len(doc_names)
    for i in range(n):
        for j in range(i + 1, n):
            if decades[i] == decades[j]:
                within.append(sim_matrix[i, j])
            else:
                cross.append(sim_matrix[i, j])
    return {
        "within_decade_avg":
           float(np.mean(within)),
        "cross_decade_avg": float(np.mean(cross))
    }

# Centroid per dekade
def decade_centroids():
    centroids = {}
    for d in np.unique(decades):
        idx = np.where(decades == d)[0]
        centroids[d] = D_norm[idx, :2].mean(axis=0)
    return centroids

# Pergeseran leksikal antar dekade
def decade_shift_distance():
    centroids = decade_centroids()
    decs = sorted(centroids.keys())

    shifts = []
    for i in range(len(decs) - 1):
        d1, d2 = decs[i], decs[i + 1]
        dist = np.linalg.norm(centroids[d2] -
               centroids[d1])
        shifts.append({
            "from": d1,
            "to": d2,
            "distance": float(dist)
        })
    return shifts

if __name__ == "__main__":
    print("Pergeseran leksikal antar dekade:")
    for shift in decade_shift_distance():
      print(f"{shift['from']} -> {shift['to']}  |
      jarak pergeseran = {shift['distance']:.4f}")
```

Fungsi most_similar_cross_year(target_year) dibuat untuk menentukan lagu-lagu dari tahun tertentu yang paling mirip dengan lagu dari tahun lain. Setiap lagu pada tahun yang ditargetkan akan dihitung nilai similaritasnya dengan semua lagu dari tahun lain, lalu akan dipilih lagu dengan nilai similaritas yang tertinggi.

Fungsi average_similarity_by_decade() menghitung rata-rata kemiripan lagu dalam dekade sama dan lintas dekade. Semua pasangan lagu akan dibandingkan. Jika keduanya berasal dari dekade yang sama, nilai akan dimasukkan ke dalam kelompok *within-decade*.

Sementara, jika keduanya berasal dari dekade yang berbeda, nilai akan dimasukkan ke dalam kelompok *cross-decade*.

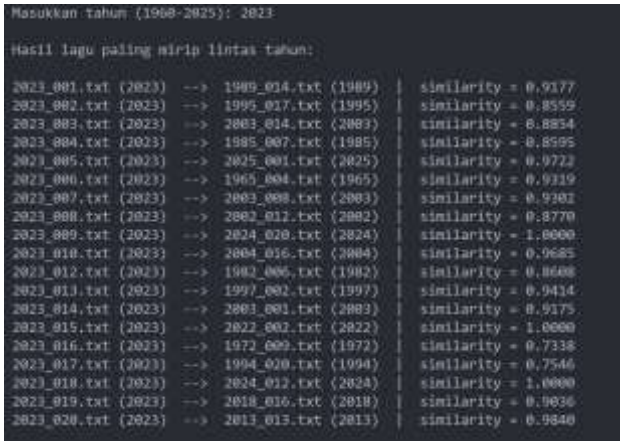
Fungsi decade_centroids() menentukan pusat (*centroid*) lagu per dekade dalam ruang LSA. Untuk setiap dekade, posisi semua lagu yang termasuk dekade itu dirata-ratakan untuk memperoleh *centroid*. *Centroid* merepresentasikan lokasi rata-rata lagu-lagu dalam dekade di ruang fitur.

Fungsi decade_shift_distance() mengukur pergeseran leksikal antar dekade dengan menghitung jarak Euclidean antar *centroid* dekade berurutan. Jarak ini menunjukkan seberapa besar perubahan pola leksikal atau representasi lagu dari satu dekade ke dekade berikutnya, sehingga dapat digunakan untuk menganalisis evolusi bahasa atau tema lirik musik dari waktu ke waktu.

## V. PEMBAHASAN

### *A. Most Similar Cross Year*

Misal, tahun yang ditargetkan adalah 2023. Berikut adadlah hasil dari perhitungan *most similar cross year* untuk setiap lagu di tahun 2023




Gambar 3. Hasil perhitungan *most similar cross year* pada tahun 2023

Nilai similaritas antar tahun untuk tiap lagu cukup tinggi. Nilai similaritas terendah adalah 0,7338. Sementara nilai similaritas tertinggi selain 1 adalah 0,9840. Hal ini menunjukkan bahwa lagu populer memiliki tema dan penggunaan kata yang cukup mirip, meskipun dirilis di tahun yang berbeda. Fenomena ini mengindikasikan terdapatnya pola leksikal yang konsisten dalam musik populer untuk menarik pendengar. Meskipun musik berkembang secara gaya dan aransemen, aspek leksikal tetap mempertahankan kemiripan yang tinggi antar lagu lintas tahun.

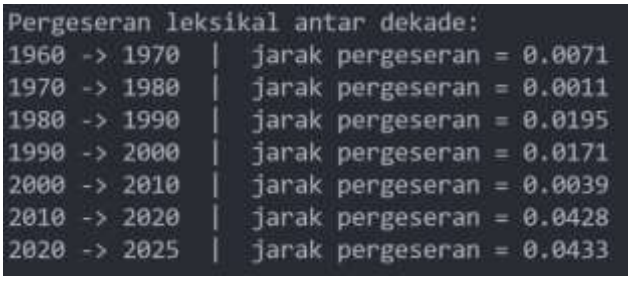
Terdapat beberapa *single* yang memiliki tingkat similaritas yang sama. Hal ini disebabkan karena terdapat beberapa *single* ke dalam daftar "*Billboard Year-End Hot 100 Chart"* lebih dari satu kali. Contohnya lagu "Snooze" oleh SZA yang memperoleh peringkat 9 di tahun 2023 dan peringkat 20 di tahun 2024. Begitu juga dengan lagu "As It Was" oleh Harry Styles (memperoleh peringkat 15 di 2023 dan peringkat 2 di 2022) dan "Cruel Summer" oleh Taylor Swift (memperoleh peringkat 18 di 2023 dan peringkat 12 di 2024).

### *B. Similaritas dalam Dekade dan Lintas Dekade*


Gambar 4. Hasil perbandingan similaritas dalam dekade dan lintas dekade

Hasil ini menunjukkan rata-rata similaritas dalam dekade yang sama lebih tinggi dibandingkan lintas dekade. Lagu-lagu dalam dekade yang sama cenderung memiliki tema, kosakata, dan pola leksikal yang mirip. Perbedaan nilai similaritas yang tidak terlalu besar menunjukkan pola leksikal lirik lagu mengalami perubahan secara lambat dari waktu ke waktu.

### *C. Pergeseran Leksikal antar Dekade*




Gambar 5. Hasil pergeseran leksikal antar dekade

Hasil perhitungan menunjukkan jarak pergeseran leksikal antar dekade dari tahun 1960 hingga 2020. Pergeseran terkecil terjadi antar 1970 dan 1980. Artinya, tidak banyak perubahan pola leksikal yang digunakan pada dekade tersebut. Sementara itu, pergeseran terbesar terjadi antar 1980 dan 1990. Hal ini menunjukkan adanya perubahan yang lebih besar dalam pola leksikal yang digunakan.

## VI. KESIMPULAN

Analisis lirik lagu dari 1960 hingga 2025 menunjukkan bahwa lagu-lagu populer cenderung memiliki pola leksikal yang mirip, terutama dalam dekade yang sama, sehingga rata-rata similaritas antar lagu dalam satu dekade lebih tinggi dibandingkan lintas dekade. Pergeseran leksikal antar dekade relatif lambat, dengan perubahan paling kecil terjadi pada dekade 1970–1980 dan perubahan paling besar terjadi pada dekade 1980-1990. Dari penelitian ini, dapat disimpulkan bahwa perubahan pola leksikal pada lirik lagu populer di Amerika Serikat (1960-2025) terjadi dengan sangat lambat.

## VI. LAMPIRAN

Berikut adalah pranala Github *repository* berisi program yang telah dibuat dan digunakan untuk menganalisis

perubahan pola leksikal pada lirik lagu populer di Amerika Serikat (1960-2025) menggunakan SVD:
https://github.com/EliWidjojo/Analisis-Perubahan-Pola-Leksikal-pada-Lirik-Lagu-Populer-Menggunakan-SVD

Selain itu, berikut adalah pranala video YouTube yang menjelaskan penelitian ini:
https://youtu.be/JBAARGsTrJY\

## VII. Ucapan Terima Kasih

Pertama, penulis ingin mengucapkan syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat yang telah diberikan-Nya sehingga makalah ini dapat selesai dengan baik. Penulis juga ingin mengucapkan terima kasih kepada Pak Rinaldi Munir, Pak Rila Mandala, dan Pak Arrival Dwi Sentosa atas materi Aljabar Linear dan Geometri yang telah disampaikan. Terakhir, penulis ingin mengucapkan terima kasih kepada para pembaca dan berharap semoga makalah ini dapat bermanfaat bagi pembaca.

## References

[1] Encyclopaedia Britannica, "Music," https://www.britannica.com/art/music. Diakses pada 24 Desember 2025 pukul 17.30.
[2] Merriam-Webster, "Popular Music," https://www.merriam-webster.com/dictionary/popular%20music. Diakses pada 24 Desember 2025 pukul 17.30.
[3] Encyclopaedia Britannica, "Single," https://www.britannica.com/dictionary/single. Diakses pada 24 Desember 2025 pukul 17.30.
[4] Genius Docs, "Getting Started," https://docs.genius.com/#/getting-started-h1. Diakses pada 24 Desember 2025 pukul 17.30.
[5] R. Munir. "Aljabar Linear dan Geometri 2024-2025," STEI ITB, https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2024-2025/MakalahAlgeo2024.htm. Diakses pada 24 Desember 2025 pukul 17.30.
[6] S. Deerwester, et al., "Indexing by Latent Semantic Analysis," Journal of the American Society for Information Science, vol. 41, no. 6, pp. 391–407, 1990.
[7] Billboard, "Music Charts, News, Photos & Video," https://www.billboard.com/. Diakses pada 24 Desember 2025 pukul 17.30

## Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Eliana Natalie Widjojo - 13524116