Spotify Recommendation Project

Roshen Nair[1], **Julia Xi**[1], **Olivia Xu**[1]

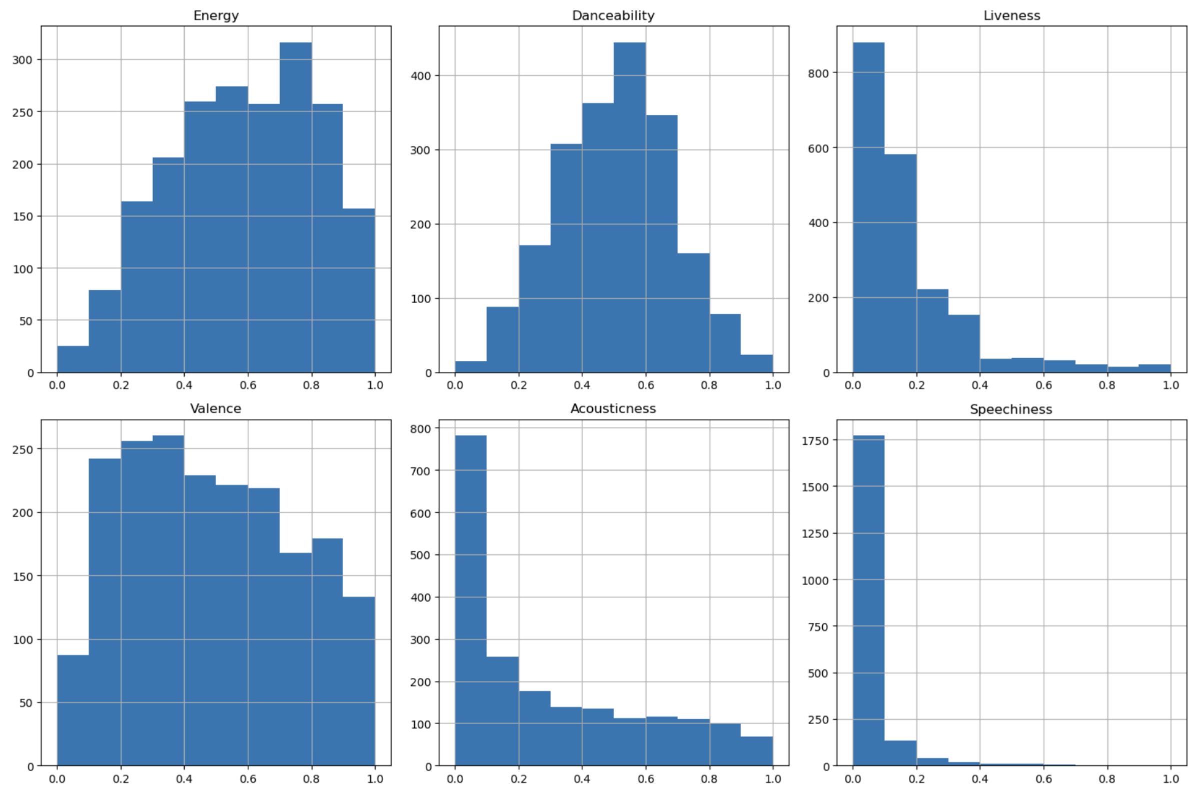
[1]Stanford University

roshen03@stanford.edu, juliaxi@stanford.edu, olivialx@stanford.edu

1. Choose Your Features

We only used the suggested features ('Energy', 'Danceability', 'Liveness', 'Valence', 'Acousticness', 'Speechiness') in our analysis and recommendation process. The remaining variables would still take into account different factors that might influence the user's music preferences. For example, a user might enjoy the lyrics in more recent songs. This relation would be hard to model in any other way besides year. Additionally, they could prefer newer songs because of they are more upbeat. In this case, keeping track of the year and the BPM of each song would allow us to propose a better prediction model (via PCA) to determine what types of songs the user would most likely prefer. The disadvantages, however, include the fact that it would likely require more computation to factor in these other features, and that it may reduce the influence of the other factors previously included (so, we might choose to ignore "Popularity", "Year", and "BPM"to focus on more important measurements that will help us predict whether a given audience will like a song or not).

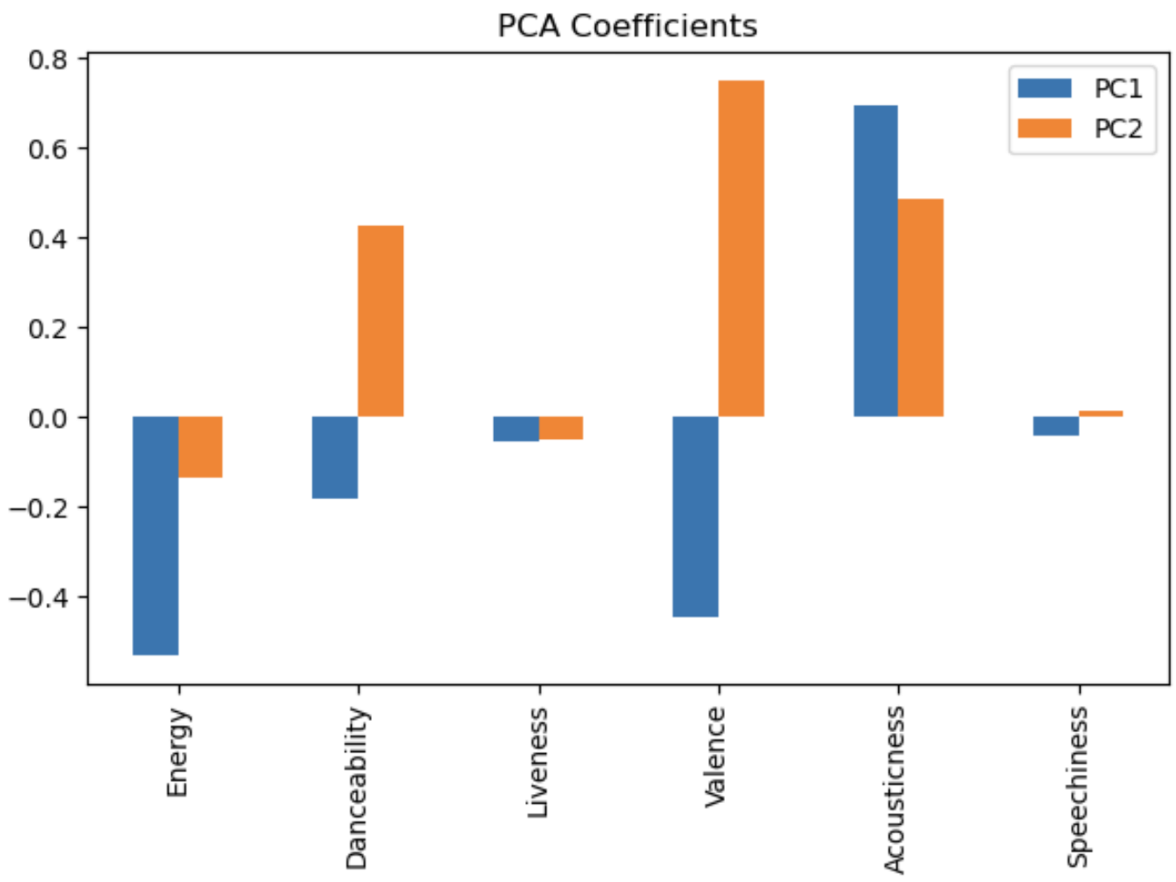
2. Visualizing the Data



(a) From the visualized plots of the data distributions, it's more clear that features like the "speechiness" of the song are more densely concentrated, while other features like "energy", "danceability", and "valence" are more spread out across the range of values we plotted above.

(b) We chose to use MinMaxScaler to scale our features (i.e. normalizing and centering), since it helps make our PCA's calculation of a dimensionality reduction model consider each characteristic of the song more equally. In the graphs of each feature, it is clear that they are not all normally distributed, so the MinMaxScaler is a better option over StandardScaler, since StandardScaler assumes that the data is normally distributed while MinMaxScaler better preserves the shape of the data. It also helped make the coefficients more interpretable (giving a better idea of how exactly each coefficient contributes to each principal component - the relative ratios will provide actually meaningful information).

3. Principal Component Analysis



The larger the magnitude of the coefficient of a principal component, the significant that feature's contribution is to that principal component. By visualizing the coefficients of the two principal components, we can interpret the main pieces of information that each principal component conveys. The first principal component (PC1) primarily focuses on the acousticness, energy, and valence of the song (in that order of priority), while mostly disregarding the other information about the song. The second principal component (PC2) is strongly influenced by valence, while also roughly equally considering the acousticness and danceability (at half the consideration level as the valence). The rest of the information is relatively unused for this particular PCA. Since Speechiness and Liveness were mostly similar amongst all the songs (they were extremely left skewed), it followed that those two metrics would not help distinguish between two songs. Thus, extremely small coefficients associated with these two properties is notably small. In this way, the principal components account for different factors of the song to help provide a recommendation system maximizing the explained variance between songs.

We wrote a function that computes the dot product of the feature values and the PCA coefficients of each song, to compute the song's principal component values (PC 1 and PC 2). In the first principal component, the song with the lowest value was 'Just Can't Get Enough (Live in Hammersmith) - 2018 Remaster' (with value -1.0574) and the song with the highest value was 'Theme from Harry's Game' (with value 0.6139). One of these songs is very cheerful song (high valence, higher energy) while the other one (the lowest value) is a more sad, depressed song (low valence, low energy), which makes sense given how the first principal component is calculated with the relative coefficients analyzed above.

In the second principal component, the song with the lowest value was 'Live Forever - Remastered' (with value -0.0056) and the song with the highest value was 'Thank God I'm a Country Boy' (with value 1.3683). The one with the lower value is much less acoustic (uses more electronic musical instruments) and less danceable and the one with the higher value is more acoustic (uses more acoustic, non-electronic musical instruments), more danceable, and has more valence (cheerfulness), which makes sense given how the second principal component is calculated.

4. K-Nearest Neighbors

We tested our recommendation system by considering an example song, "High Hopes" by Panic! At the Disco, and listening to our algorithm's recommendations based on the five most similar songs (using PCA and then shortest Euclidean distance):

"High Hopes" *by Panic! At The Disco* song recommendations:

1. "I Would Die 4 U" by Prince
2. "Our House" by Madness
3. "Stay - Remastered by Jackson" Browne
4. "One Way Wind" by The Cats
5. "I Will Survive - Single Version" by Gloria Gaynor

These recommendations were generally well-accepted, as they reflected some qualities like the liveliness and the overall energy, valence, and tone of the song. Many of these songs shared an upbeat undertone and positive energy. However, there were also some major differences between the songs. This could primarily be a result of differing genres (a feature we did not use in analyzing our song data), which could result in the song being less liked by the same audience. To address this inconsistency, one could attempt to use other features (e.g.: the genre of the song, quantified in a way that it can be assigned a numerical coefficient when doing PCA) in order to provide a better recommendation for the audience.

5. Code

We published our code on GitHub. To write the program for this project, we referenced many websites (including but not limited to Stack Overflow, scikit-learn, and GeeksforGeeks) to help with syntax and implementation for various functions. We used pandas to read and work with the csv file, matplotlib to create the bar charts for each feature, sklearn to scale and do PCA, and numpy to compute the 5 nearest neighbors for the inputted song.

We also include our program below:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA


def process_data(filename):
    df = pd.read_csv(filename)
    columns = ['Energy', 'Danceability', 'Liveness',
     'Valence', 'Acousticness', 'Speechiness']

    df[columns] = MinMaxScaler().fit_transform(df[columns])
    # scale data

    pca = PCA(n_components = 2)
    pca_components = pca.fit_transform(df[columns])
    # use 2 PCs

    return df, columns, pca_components, pca

def print_data(df, features, pca):
    plt.figure(figsize=(15, 10))
    for i, curr in enumerate(features, 1):
    # loop over 6 features and create plots
        plt.subplot(2, 3, i)
        df[curr].hist()
        plt.title(curr)
    plt.show()

    pca_coef = pd.DataFrame(pca.components_, columns=
    features, index=['PC1', 'PC2'])

    plt.figure(figsize=(10, 6))
    pca_coef.T.plot(kind='bar')
    plt.title('PCA Coefficients')
    plt.show()

    plt.figure(figsize=(6, 4))
    plt.bar(range(2), pca.explained_variance_ratio_)
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.tight_layout()
    plt.show()

def recommend_songs(song_title, df, features, pca_components, n=5):
    song_title = song_title.lower()
    if song_title not in df['Title'].str.lower().values:
        print(f"'{song_title}' not in song database")
        return

    for index, row in df.iterrows():
    # find index of song
        if row['Title'].lower() == song_title:
            curr_index = index
            break

    curr_pca = pca_components[curr_index]

    distances = []
    for index, row in df.iterrows():
    # get distances between all pairs with input song
        if index != curr_index:
            song_pca = pca_components[index]
            distance = np.linalg.norm(song_pca - curr_pca)
            distances.append((index, distance))

    distances = sorted(distances, key=lambda x: x[1])
    # sort songs by distance (not index)

    print("Here are your recommendations:")

    for i, (index, _) in enumerate(distances[:n], 1):
    # print out 5 nearest neighbors
        song = df.loc[index]
        print(f"{song['Title']}, {song['Artist']}, {distances[i][1]}")

def main():
    df, features, pca_components, pca = process_data('Spotify-2000.csv')

    while True:
        userinput = input("\nType 'song lookup' or 'print data'").strip().lower()

        if userinput == 'song lookup':
            title = input("favorite song?").strip()
            recommend_songs(title, df, features, pca_components)

        elif userinput == 'print data':
            print_data(df, features, pca)

        else:
            print("try again?")

if __name__ == "__main__":
    main()
```