

The background is a light cream color with a fine, dotted texture. It is decorated with various hand-drawn elements: a large orange abstract shape with teal dots in the top left; a yellow bean-like shape in the top center; a teal abstract shape in the top right; a red treble clef on the left; a blue treble clef at the bottom center; and several musical notes in teal, blue, and red scattered throughout. Abstract shapes in teal, orange, and yellow are also present at the bottom.

# Song Recommender System

## Cosine Similarity

Afra Farkhooy & Victoria Stråberg  
*Machine Learning for Social Media TNM108*

# What is a Recommender System?

A recommendation system is a data filtering tool that recommend the most relevant items to a particular user. It uses machine learning algorithms and operates on the principle of finding patterns in consumer behavior data.

It can be about being recommended which movie to watch, text to read, products to buy or something else depending on the industry. In fact, some of the biggest brands we engage with every day are built around a recommender system, for example Netflix, Amazon and Spotify. **The possibilities are truly endless when it comes to recommender systems!**

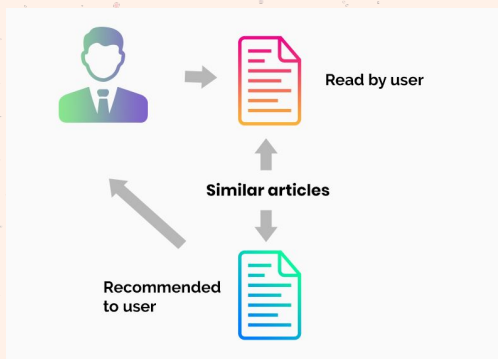


# Different Types of Recommender Systems

There are two primary types of recommender systems: Content-based method and Collaborative-based method. A hybrid system system, combining these two systems also exists.

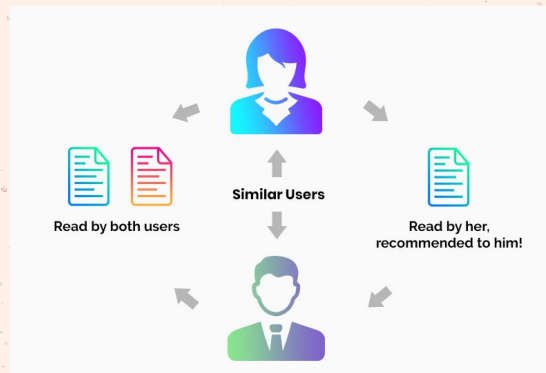
## Content-based method

Content-based filtering selects items based on the similarities between the content description of an item and the user's preferences



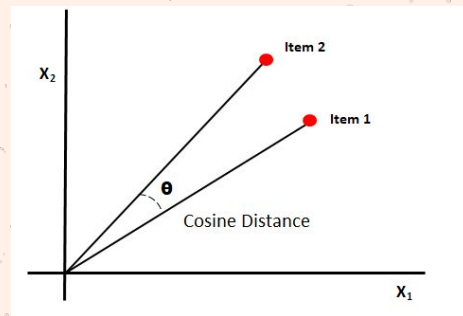
## Collaborative-based method

Collaborative filtering select items based on the similarities between the preferences of different users [1]



# Cosine Similarity ?

The cosine similarity metric finds the normalized dot product of two attributes [2]. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects. The cosine of  $0^\circ$  is 1, and it is less than 1 for any other angle.



**Advantage with cosine similarity** is that it even if the two similar data objects are far apart by the Euclidean distance because of the size, they could still have a smaller angle between them. Smaller the angle, higher the similarity. When plotted on a multi-dimensional space, the cosine similarity captures the orientation (the angle) of the data objects and not the magnitude. Two vectors with the same orientation have a cosine similarity of 1, two vectors at  $90^\circ$  have a similarity of 0.

The background is a light cream color with a fine, speckled texture. It is decorated with several musical-themed elements: a red eighth note in the top left, a teal eighth note in the top right, a yellow treble clef in the top right, and a yellow eighth note in the bottom right. There are also abstract shapes, including a yellow blob with blue dots in the top left and a teal circle in the bottom right.

# Our Work

In this work, we want to explore which songs are similar to each other according to their **sounds** - for example the tempo, energy and instrumentalness. We have worked with content-based filtering and the chosen method is cosine similarity. To obtain our results, we do the work in the programs *Visual Studio Code* and *MATLAB*.

## Research questions:



1. **Is it possible to create a music recommender based on cosine similarity?**
2. **How accurate are the similarities?**

# The Data Set


The data set comes from Kaggle, which is a website that provides code and data you need to be able to do science work or research. On Kaggle you can basically find anything you need for your purpose.

Since we wanted to use a data that contained songs and song attributes we searched through a few music data on Kaggle and compared how big the dataset were. The more songs and different song attributes, the greater the probability of finding the right suitable song. We finally chose a dataset that contained 18 different categories and about 230.000 songs [3].





The dataset contains the genre of a song, the name of the artist, the song title and some song attributes. The attributes and their description can be found on Spotify's web API reference [4].



### **Acousticness number <float>**

A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic

### **Danceability number <float>**

Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable


### **Duration\_ms integer**

The duration of the track in milliseconds.


### **Energy number <float>**

Measure from 0.0 to 1.0 and represents a perceptual measure of intensity and high activity. Typically, energetic tracks feel fast, loud, and noisy. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate and general entropy

### **Mode integer**



Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.



### **Instrumentalness number <float>**

Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.

### **Liveness number <float>**

The duration of the track in milliseconds. Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.

### **Loudness number <float>**

The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks.

### **Speechiness number <float>**

Detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.





### **Tempo number<float>**

The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.


### **Time signature integer**

An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4".



### **Valence number<float>**

A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).



# Data Processing

Before using the data set it is important to prepare it with different steps. Our dataset contains unnecessary columns, and we decided to **drop** the columns we did not want.

Instead, we wanted to focus on these nine features: 'acousticness', 'danceability', 'energy', 'speechiness', 'tempo', 'instrumentalness', 'loudness', 'valence', 'liveness'

We used **drop\_duplicates** because we discovered that there were duplicates of a song that were placed in different genres but also that there were different variations of the selected song. We wanted to get similar songs based on other artists! For example Avicii "Levels" was placed in both the Dance genre and the Pop genre. The song had also different versions – the Radio edit and the Original.

```
#Dropping unnecessary columns
raw_data_with_measures = raw_data_with_measures.drop(["genre", "popularity",
"duration_ms", "key", "time_signature", "mode"], axis="columns")
```

Output gives  
duplicates of song/artist

```
Artist      Avicii
Song        Levels - Radio Edit
Name: 14391, dtype: object
Cosine Similarity: 1.0

Artist      Avicii
Song        Levels - Edit
Name: 111024, dtype: object
Cosine Similarity: 1.0

Artist      BROHUG
Song        Dust
Name: 26758, dtype: object
Cosine Similarity: 0.99860555745

Artist      Avicii
Song        Levels
Name: 15749, dtype: object
Cosine Similarity: 0.99853025240

Artist      Avicii
Song        Levels
Name: 150800, dtype: object
Cosine Similarity: 0.99853025240
```

# Data Processing

We also checked missed data with a for loop and used pandas **isnull()**. The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly, then we may end up drawing an inaccurate inference about the data. This may lead to biased results and loss of power.

Many algorithms perform better when numerical input variables are scaled to a standard range. We wanted to **scale** the relevant numerical features in the data with the sklearn **MinMaxScaler()**. This transforms features by scaling each feature to a given range (0,1). This scaling will preserve the shape of the dataset without distortion.

At least we joined the scaled data into the first dataset and dropped the old ones. We also used the pandas **Dropna()** to drop missing values.

**Now the data is ready to use!**

```
#Checking missing values
def display_missing(data):
    for col in data.columns.tolist():
        print('{} column missing values: {}'.format(
            col, raw_data_with_measures[col].isnull().sum()))
    print('\n')
display_missing(raw_data_with_measures)

#Scaling the data
song_data = raw_data_with_measures.loc[:, ['acousticness', 'danceability', 'energy',
                                           'liveness', 'speechiness', 'valence',
                                           'tempo', 'instrumentalness', 'loudness']]

scaler = MinMaxScaler()
song_features = pd.DataFrame()

for col in song_data.iloc[:, :].columns:
    scaler.fit(song_data[col].values)
    song_features[col] = scaler.transform(song_data[col].values.reshape(-1, 1)).ravel()

#Merging the new scaled data to the original dataframe and dropping the unscaled data. final_merge_df is our new dataframe.
data_to_merge = raw_data_with_measures.drop(['acousticness', 'danceability', 'energy',
                                              'liveness', 'speechiness', 'valence',
                                              'tempo', 'instrumentalness', 'loudness'], axis="columns")
final_merge_df = data_to_merge.join(song_features).dropna()
```

# Modelling

1.

```
def rank_song_similarity_by_measure(data, song, artist):  
    data.rename(columns={'track_name': f'Song', 'artist_name': f'Artist'}, inplace=True)
```

We defined a function where the input arguments are the data we are using and the name of the song and artist we want to find similar songs to.

We started by renaming the columns for song and artist to have a more good looking output.

2.

```
#Picking out the artist and the song  
artist = data.loc[data['Artist'] == artist]  
song = artist.loc[artist['Song'] == song]
```

We picked out the artist and the title of the song from the dataset.

3.

```
#Creating new data including only the chosen song attributes  
song_and_artist_data = song.drop(["Song", "Artist", "track_id"], axis="columns")
```

Then we created a new data set that only includes the audio features we have chosen to work with and we therefore had to drop song, artist and track-id.

# Modelling

4.

```
#Copy of input data
similar_data = data.copy()

#Picking out the chosen song attributes from the input data
data_values = similar_data.loc[:, ['acousticness', 'danceability', 'energy', 'liveness', 'speechiness', 'valence', 'tempo', 'instrumentalness', 'loudness']]
```

A copy of the input data were made and from that we picked out all the audio features of the chosen song.

5.

```
#Cosine similarity
cos_sim = cosine_similarity(song_and_artist_data, data_values)

#Printing out the similar songs
for i in range(int(5)):

    most_similar = np.argmax(cos_sim[0], axis=0)
    # print(data.iloc[[most_similar], [0,1]]) #0 column artist, 1 column for song
    print(data.iloc[most_similar, [0,1]]) #0 column artist, 1 column for song
    print('Cosine Similarity:', (cos_sim[0][most_similar]))
    print("")
    cos_sim[0][most_similar] = 0
```

Then we calculated the cosine similarity between the chosen song audio features with all the other songs in the data's features.  
The for loop prints out the top 5 most similar songs and the cosine similarity between them.

Python library scikit-learn has a built-in cosine similarity calculator.

# Modelling

```
##### COSINE SIMILARITY #####

def rank_song_similarity_by_measure(data, song, artist):

    data.rename(columns={'track_name': f'Song', 'artist_name': f'Artist'}, inplace=True)

    #Picking out the artist and the song
    artist = data.loc[data['Artist'] == artist]
    song = artist.loc[artist['Song'] == song]

    #Creating new data including only the chosen song attributes
    song_and_artist_data = song.drop(["Song", "Artist", "track_id"], axis="columns")

    #Copy of input data
    similar_data = data.copy()

    #Picking out the chosen song attributes from the input data
    data_values = similar_data.loc[:, ['acousticness', 'danceability', 'energy', 'liveness', 'speechiness', 'valence', 'tempo', 'instrumentalness', 'loudness']]

    #Cosine similarity
    cos_sim = cosine_similarity(song_and_artist_data, data_values)

    #Printing out the similar songs

    for i in range(int(5)):

        most_similar = np.argmax(cos_sim[0], axis=0)
        # print(data.iloc[[most_similar, [0,1]]] #0 column artist, 1 column for song
        print(data.iloc[[most_similar, [0,1]]] #0 column artist, 1 column for song
        print('Cosine Similarity:', (cos_sim[0][most_similar]))
        print("")
        cos_sim[0][most_similar] = 0

rank_song_similarity_by_measure(final_merge_df, "You Give Love A Bad Name", "Bon Jovi")
```

The full function. In order to use it we call it with the three necessary arguments, like in this case the datafile, the song title and the name of the artist.

# Evaluation

The results show both the output in Python code in *Visual Studio Code* and the Cosine similarity calculations in *MATLAB*.

We wanted to compare and manually calculate cosine formula using MATLAB to know if the result was correct. We wrote all nine relevant features we compared in every song and put it in the cosine similarity formula.

Chosen artist and song by user:

Artist: Bon Jovi

Song: You Give Love A Bad Name





# Evaluation

We notice that the calculation in MATLAB matches with the result in the Python code! Value: 0.9992

In this case MATLAB made it easier to interpret and understand the result in a numerical way!

First song is chosen by user.  
Songs listed below are similar ones!

Info about the artist/song:

Manual calculations of the nine features:

Artist Bon Jovi  
Song You Give Love A Bad Name  
Name: 166234, dtype: object  
Cosine Similarity: 1.0

Artist Ty Herndon  
Song Living In A Moment  
Name: 4157, dtype: object  
Cosine Similarity: 0.9992375688964952

Artist Devin Dawson  
Song All On Me - Acoustic  
Name: 3933, dtype: object  
Cosine Similarity: 0.9989063888772058

Artist Motohiro Hata  
Song Dear Mr. Tomorrow  
Name: 45573, dtype: object  
Cosine Similarity: 0.9987888409725741

Artist Comedian Bob Marley  
Song I'm From Maine  
Name: 167930, dtype: object  
Cosine Similarity: 0.9986592889932835

artist\_name Bon Jovi  
track\_name You Give Love A Bad Name  
track\_id 3KYiA4vq6RP01dE2XROxd8  
acousticness 0.01747  
danceability 0.503272  
energy 0.688682  
liveness 0.257823  
speechiness 0.003493  
valence 0.52  
tempo 0.534867  
instrumentalness 0.0  
loudness 0.845519  
Name: 166234, dtype: object

artist\_name Ty Herndon  
track\_name Living In A Moment  
track\_id 1QIYR4q1Pt19yLHG4Ef4rg  
acousticness 0.009347  
danceability 0.534385  
energy 0.670664  
liveness 0.26994  
speechiness 0.009843  
valence 0.487  
tempo 0.553533  
instrumentalness 0.00003  
loudness 0.830822

```
check_cosine.m x +
1 A = [0.01747 0.503272 0.688682 0.257823 0.003493 0.52 0.534867 0.0 0.845519]; %Bon Jovi - You Give Love A Bad Name
2 B = [0.009347 0.534385 0.670664 0.26994 0.009843 0.487 0.553533 0.00003 0.830822]; %Ty Herndon - Living In A Moment
3
4 AB = sum(A.*B);
5
6 floor_A = sqrt(0.01747^2 + 0.503272^2 + 0.688682^2 + 0.257823^2 + 0.003493^2 + 0.52^2 + 0.534867^2 + 0.0^2 + 0.845519^2);
7 floor_B = sqrt(0.009347^2 + 0.534385^2 + 0.670664^2 + 0.26994^2 + 0.009843^2 + 0.487^2 + 0.553533^2 + 0.00003^2 + 0.830822^2);
8
9 cos_sim = (AB)/(floor_A*floor_B);
10 disp(cos_sim);
```

Command Window

```
>> check_cosine
0.9992
```



# Conclusion

As you noticed, the 4 best songs that are similar to the selected song are almost 1. This in itself looks like a very correct match, but when you listen to the songs, it is not the perfect match. You can hear some similarities, for example all of the that were most similar to Bon Jovi's song had a clear instrumental analogy with guitar. Therefore it may not be a good song recommender for a private user looking for new songs.



## Improvements

Use more audio features or compare the songs with genres to get matching songs within specific genres. Could be more accurate.

Use a dataset that contains even more songs and is up to date.

User testings where users could rate the songs they were recommended based on how accurate they were to see what we could improve. .





# Conclusion – what is the answer to our research questions?

- 
1. **Is it possible to create a music recommender based on cosine similarity?**
  2. **How accurate are the similarities?**

We think that cosine similarity is the best algorithm to use in this type of case. We could also have used Euclidean distances but as we mentioned before, it can be misleading and affect the recommender in a bad way

Therefore the answers to our research questions is that it is possible to build a recommender system with cosine similarity and based on the mathematics the results are accurate but whether the songs are accurate for a user is a matter of taste.



# References

- [1] Medium, 2021, "Types of Recommendation Systems & Their Use Cases", retrieved 2021-12-08  
<https://medium.com/mlearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9>
- [2] Saimadhu Polamuri, 2015, "FIVE MOST POPULAR SIMILARITY MEASURES IMPLEMENTATION IN PYTHON", Dataaspirant, retrieved 2021-12-10  
<https://dataaspirant.com/five-most-popular-similarity-measures-implementation-in-python/>
- [3] Dmitri Ivanov, 2021, "Similar songs clustering (K-Means, Cosine sim.)", Kaggle, retrieved 2021-11-25  
<https://www.kaggle.com/dimasikson/similar-songs-clustering-k-means-cosine-simil/data>
- [4] "Get Track's Audio Features", Spotify for Developers, retrieved 2021-12-10  
<https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features>
- Code:**  
<https://gitlab.liu.se/afrfa108/TNM108>