ENGR- E536 : HIGH PERFORMANCE GRAPH ANALYTICS
PROJECT PRESENTATION

MUSIC RECOMMENDATION MODEL USING GNN

MEMBERS :

RISHIKA SAMALA

DEVNA RAMESH

KAMNA CHAUDHARY

MADHURUPA SAMADDAR

PAYAL GHORPADE

INDIANA UNIVERSITY

INTRODUCTION

- **OBJECTIVE**
  - Research and Propose a model that can replicate the recommendation model with GNN on a smaller dataset capturing the underlying relation between users and tracks which can be used to provide nearly accurate recommendations.
- **IDEA**
  - Helps provide personalized and relevant recommendations to users by analyzing the user's listening patterns.
  - Taken into account – Collaborative filtering and Content based filtering to better predict the user's behavior and to overcome the known problems and shortcomings of such a system.

RECOMMENDATION SYSTEMS

- Software applications/algorithms that provide personalized suggestions or recommendations to users based on their preferences, behavior and past interactions with products/ services.
- Relevance : Personalization (Tailored user experience) , Discover new products/ content leading to increased engagement, Increased Efficiency, Increased revenue for businesses.
- Real world Recommendation systems :
  - TikTok – Deep retrieval and Monolith
  - LinkedIn – Talent Search and recommendation Systems at LinkedIn
  - Google – Deep Neural Networks for YouTube Recommendations, Wide and Deep Learning for Recommender Systems.

Why GNN?

- Can model complex relationships between users and items by considering the graph structure of the interaction network.
- Helps overcome the issues faced when using the native method to represent the graph as an adjacency matrix and multiply a dense neural network layer directly.
- Helps learn the best embedding representations of graph nodes for node-level/edge-level/graph-level predictions by providing explainable recommendations by visualizing the graph structure.

Light GCN or GCN Conv

- Light GCN is considered a better implementation if the system has many users/items, with sparse implementation choice due to its lightweight design and efficiency.
- GCN Conv is considered a better implementation if the system has a complex graph structure with both explicit and implicit feedback data due to its ability to handle complex graph structures and explicit feedback data.

CONTENT BASED VS COLLABORATIVE FILTERING

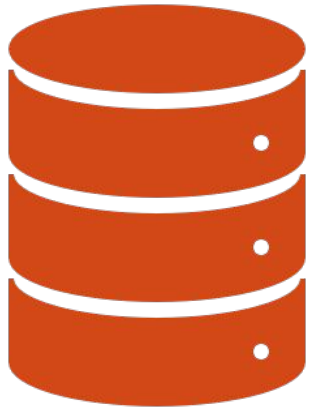
- **Content based Filtering**
  - Based on features of a product (keywords) and a profile of the user's preferred choices (user liking a kind of product)
  - Central Assumption : You will like a similar item if you like a particular item.
  - Computationally expensive
- **Collaborative Filtering**
  - Based on gathering and analyzing data on user's behavior (user's online activity, liked music etc.)
  - Advantage : can recommend complex items precisely without understanding the object itself
  - Takes into account the behavior of other users who have similar listening habits to the users being recommended.

DATASET – SPOTIFY MUSIC DATASET

- **Spotify dataset - Data.csv file**
  - A dataset available from Kaggle
  - Columns: acousticness, danceability, valence, year, artists, instrumentalness etc.

- **Spotify's "The Million Playlist Dataset"**
  - Sampled a subset of the spotify's million playlist dataset, which comprises over 2 million unique tracks by nearly 300,000 artists.
  - Easily accessible, the dataset holds users, their playlists and tracks contained in the playlists.
  - Columns : user_id, artistname, trackname, and playlistname

- **Spotify API**
  - Retrieve user data such as user playlists, saved songs, and recently played tracks, and further process the same to transform into a suitable format for the recommendation system.



CONTENT BASED FILTERING

PRE-PROCESSING:

Dataset - 170653 instances and 19 features.

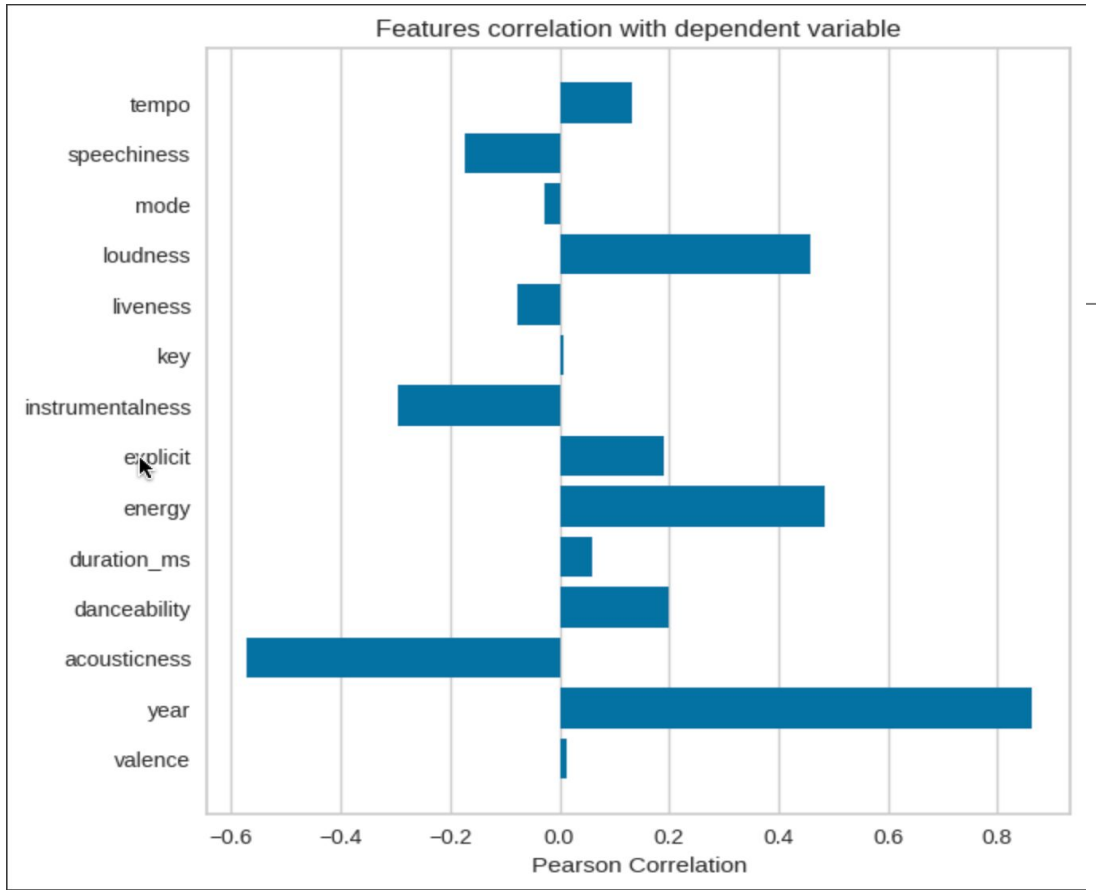
We sampled the dataset into 10000 instances → considering random 1000 songs per year from 2010 to 2020.

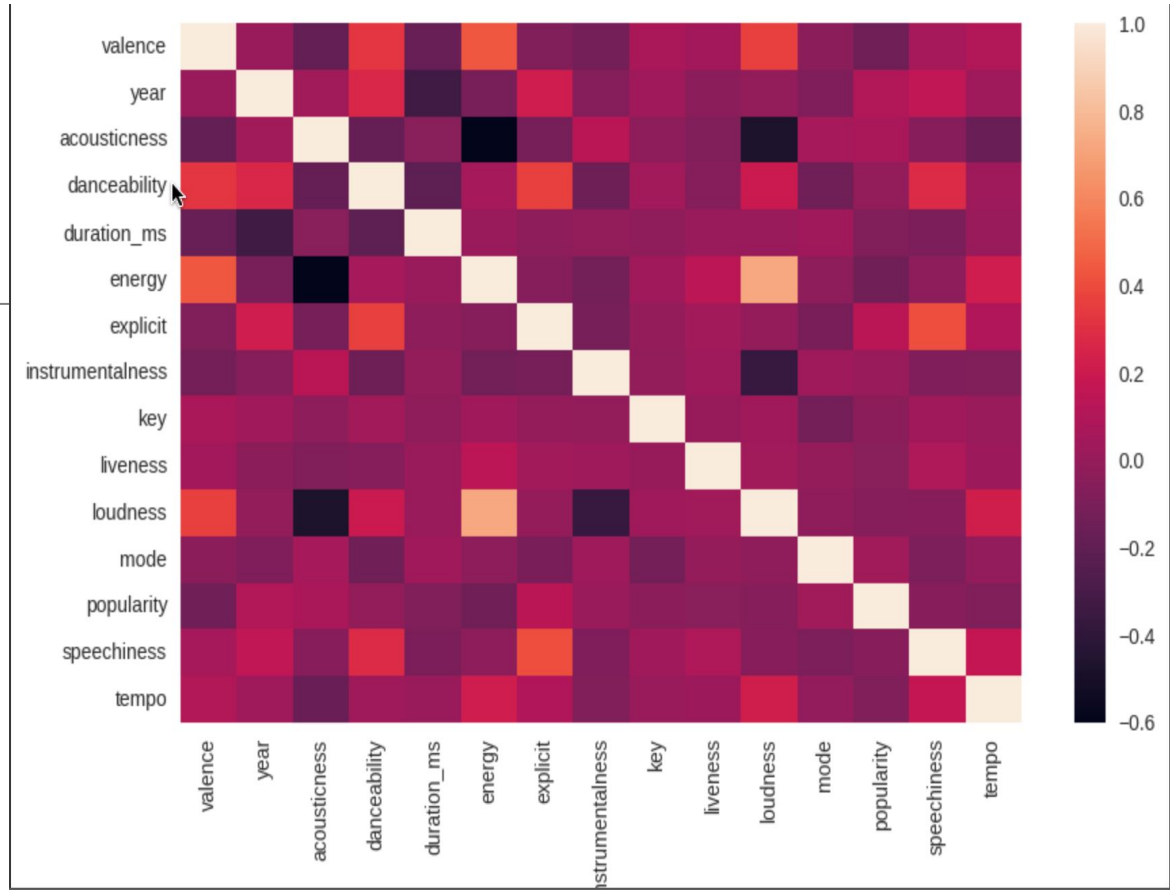
To understand user interactions with songs, it is important to recognize the features that really affect the popularity of songs

```
df = data
df['year'] = df['year'].astype(int)

# Define a function to get the top 1000 songs for each decade
def get_top_songs(df):
    top_songs = pd.DataFrame()
    for decade in range(2010, 2021):
        songs = df.loc[(df['year'] > decade) & (df['year'] < decade + 10)]
        songs = songs[:1000]
        top_songs = pd.concat([top_songs, songs])
    return top_songs

# Call the function to get the top 1000 songs for each decade
top_songs = get_top_songs(df)
```





- Considering correlation of all features with popularity it was found that a few features really affect popularity in this dataset while others like valence, duration_ms, key, mode etc. did not have an adverse effect.

- We also plotted the correlation between one feature with every other feature, to identify the ones that are highly correlated to remove the redundant information.

- Finally, we selected 8 features - danceability , energy, instrumentalness, popularity, liveness, speechiness, tempo, and year.
- We split the data instances into Training data(70%) and test data (30%).
- Networkx was used to form the graph where each song is taken as a node and edges are formed based on the similarity matrix of cosine. But cosine was taking a lot of time and was crashing even for 10000 instances.
- We reduced the memory and computing complexity by using only the feature columns that were considered while loading the data, by converting the data type from float64 to float32 and normalizing the audio features.
- We used Euclidean distance for similarity matrix calculation and formed the graph.
- Extracted node, edge indices by using torch geometric from the graph formed.

We started initially with defining a simple GCN model with 2 hidden layers and number of hidden dimensions as 64 .

Used RELU activation and Mean Square Error Function.

Optimization function we used as Adam with 0.01 learning rate and 5 e-4 as decay.

It was not successful as it was crashing even with Colab GPU.

We tried to reduce the features and the number of instances to 100 and trained the model.

The training is taking a long time and training error was high.

The resources were not sufficient for even handling small data and those many features. We are yet to try on ReD and other resources.

So, we decided to go with Collaborative filtering with user id and track numbers from Spotify API.

COLLABORATIVE FILTERING

PRE-PROCESSING:

- Kaggle Spotify Playlists Dataset:
  - 100 songs chosen per playlist (from the top playlists)
  - Action on_bad_lines while reading csv file
  - Implemented User and Track Encoder/Decoder
  - Train and test data split based on the users
  - Edge Indices tensors created for train/test data
- Spotify API Dataset:
  - Added our users in the Spotify Web Developer dashboard
  - Extracted user id’s and track details against each user
  - Defined dataframe for encoded userID’s and trackID’s

```
df['playlistname'].value_counts()[3:13]
```

```
Rock          30142
2014          22680
Christmas     22281
2013          20895
Work          18410
Jazz          18291
Indie         17859
Classical     16328
everything    16316
Country       15520
Name: playlistname, dtype: int64
```

```
[ ] playlists_counts = df['playlistname'].value_counts()[3:13]

    playlist_names = playlists_counts.index.tolist()

    print(playlist_names)
```

```
['Rock', '2014', 'Christmas', '2013', 'Work', 'Jazz', 'Indie', 'Classical', 'everything', 'Country']
```

```
[ ] df['playlistname'].unique()
```

```
array(['HARD ROCK 2010', 'IOW 2012', '2080', ..., 'Nov 2014 for Alp',
       'Various Artists – Top Latino V.7', 'yoga pop up'], dtype=object)
```

```
user_encoder = {user_id: index for index, user_id in enumerate(df['user_id'].unique())}
track_encoder = {trackname: index for index, trackname in enumerate(df['trackname'].unique())}
df['user_id'] = df['user_id'].apply(lambda x: user_encoder[x])
df['trackname'] = df['trackname'].apply(lambda x: track_encoder[x])
```

[ ]

```
track_id = 5
track_name = track_decoder[track_id]
print(track_name)
```

```
All Be Okay
```

[ ]

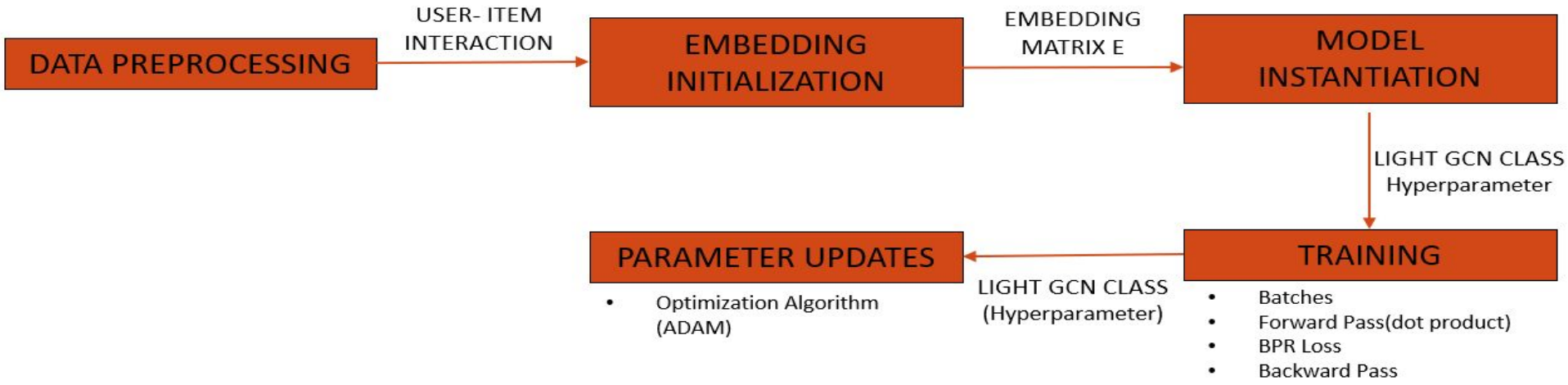
```
user_decoder = {v: k for k, v in user_encoder.items()}
```

```
def get_top_songs(df):
    top_songs = pd.DataFrame()
    for p in playlist_names:
        songs = df.loc[(df['playlistname'] == p)]
        songs = songs[:100]
        top_songs = pd.concat([top_songs, songs])
    return top_songs

# Call the function to get the top 10 songs for each decade
top_songs = get_top_songs(df)
```

MODELLING

- Creation of Sparse Edge indices tensor (/sparse user-item interaction matrix)
- Definition of LightGCN Class (input arguments- number of users, number of songs, and embedding size)
- Initialization of users and items embedding.
- Loss function - BPR loss/Cross Entropy loss defined
- BPR loss formula is L = -\sum_{(u,i,j) \in B} \log \sigma(score_{ui} - score_{uj})



KEY FINDINGS & CONCLUSION

- We observed that graph-based models can provide a more holistic view of the data and captures the underlying relationships between items and users, which makes it more accurate and effective for recommendations compared to general neural networks
- Light GCN seemed to be better than GCN in terms of its simplicity and performance
- We realized that Content Based Recommendation system comes with its own limitations on capturing user preferences

- Graph-based recommendation models like LightGCN leverage the graph structure to learn the latent representations of users and items and make recommendations based on their similarity in the latent space
- GCN model turned out to be computationally heavy compared to LightGCN indicating the upper hand of LightGCN on recommender modelling

FUTURE WORK

- Execute the recommendation function on the trained model
- Include more features like genre, energy etc in the GCN Model
- Explore the combination of content based and collaborative filtering
- Evaluate the model by extending the number of users

REFERENCES

- Analysis of Scenario-Based Recommendation Systems using Graph Neural Networks - Nithin Nataraj Vusirikala1, Rohit Gampa2, Sylvia Boddu3
- https://www.kaggle.com/datasets/ashwinik/spotify-playlist
- https://medium.com/stanford-cs224w/recommender-systems-with-gnns-in-pyg-d8301178e377
- https://arxiv.org/pdf/2002.02126.pdf
- https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

THANK YOU