CMPE 257
Genre Classification and Song Recommendation on the Free Music Archive Dataset
Final Report

Group 12

Harshit Jangam, Rahul Koonantavida,
Leo Rozay, Prateek Sharma, Paul Junver Soriano

04/28/2025

1. Idea

1.1. Description

Music is deeply ingrained in anthropologic history. From hand clapping to sticks and stones, bone flutes, stringed instruments, and more recently, digitally produced songs – music has been an outlet of expression for what it means to be human. This intertwining of music and human experience has led to the creation of labels called genres, which categorize music by sound, feeling, atmosphere, and more. Over time, as the amount of music being produced has grown, the amount and nuance of genres have also increased, making them prevalent in describing how culture and community emerge within music. However, since genres are curated by humans, they are also subject to inherent biases. The process of classifying which genre or genres a song fits into remains complex.

Genre classification is a growing interest in Music Information Retrieval (MIR), an interdisciplinary field that focuses on conducting computational tasks with music-related data. It is an exceedingly important problem space, with companies like Apple Music, Spotify, and Shazam generating business through their ability to curate tailored playlists and generate song, artist, and album recommendations. In order to deliver high quality results to their many users, it is imperative that these companies are able to effectively classify music based on genres, as this enables them to accurately determine top candidates for suggestions to a user based on their listening history. In the past, music genre labelling was done manually or based on metadata such as artist, album, or user-generated tags. However, modern systems use automated methods to classify and recommend music using audio features and machine learning algorithms. Implementing our own rudimentary version of this song classification and recommendation pipeline is the focus of our project.

In order to build an accurate genre classifier, high quality data is of the utmost importance. The Free Music Archive (FMA) dataset [1] is a comprehensive, open-source collection of musical data designed specifically for research in music retrieval. It includes metadata, audio features, and genre annotations for thousands of tracks, making it suitable for our goal of a supervised learning approach. For example, it contains features such as “mfcc,” “spectral_centroid,” and “chroma_stft,” which describe various characteristics of a given track’s audio signature. This dataset has a wide range of applications, but as mentioned before, we will particularly focus on genre classification and music recommendation. In the following section, we highlight the different ways we aim to take advantage of the FMA dataset to achieve our project goals.

1.2. Goals/Objectives

The primary goals of our project can be divided into two major components. At a high level, we aim to first develop an effective music genre classification model trained on the FMA dataset. Then, given the YouTube URL of a song as input, we aim to leverage this model to recommend similar songs from the dataset. As implied in the prior sentences, these tasks must be completed sequentially, as a key aspect of our objectives is to utilize our trained model as the primary driver of decision making during the song recommendation stage of our project.

1.2.1 Genre Classification Goals/Objectives

1. Conduct exploratory data analysis (EDA) and preprocess the FMA dataset.
2. Explore feature engineering to further augment training data efficacy.
3. Research and identify a suitable model for music genre classification.
4. Design and implement a model for music genre classification.
5. Assess performance metrics for the trained model, and compare performance metrics with baseline model performances.

1.2.2 Song Recommendation Goals/Objectives

1. Develop a Python script to receive the YouTube URL of a song and extract its relevant audio features to provide as input for our trained model.
2. Utilize model classification results to generate song recommendations.
    a. Reclassify tracks of other genres in the dataset into “Rock,” “Electronic,” “Pop,” “Hip-Hop,” and “Folk” genres.
    b. Compute their cosine similarity values with respect to our classification of the input song and rank them to generate output.
3. Establish a baseline recommendation performance metric.
4. Evaluate and compare metrics of baseline and trained model.

2. Work Developed

2.1. Adopted Machine Learning Algorithms/Techniques

2.1.1. Data Preprocessing

For this project, we began by conducting exploratory data analysis on the FMA dataset. An important factor to consider was that the dataset suffers from

imbalancing issues, presenting us with a challenge. For example, the “Rock” genre contains upwards of 20,000 data samples, while the “Jazz” genre contains less than 10,000.

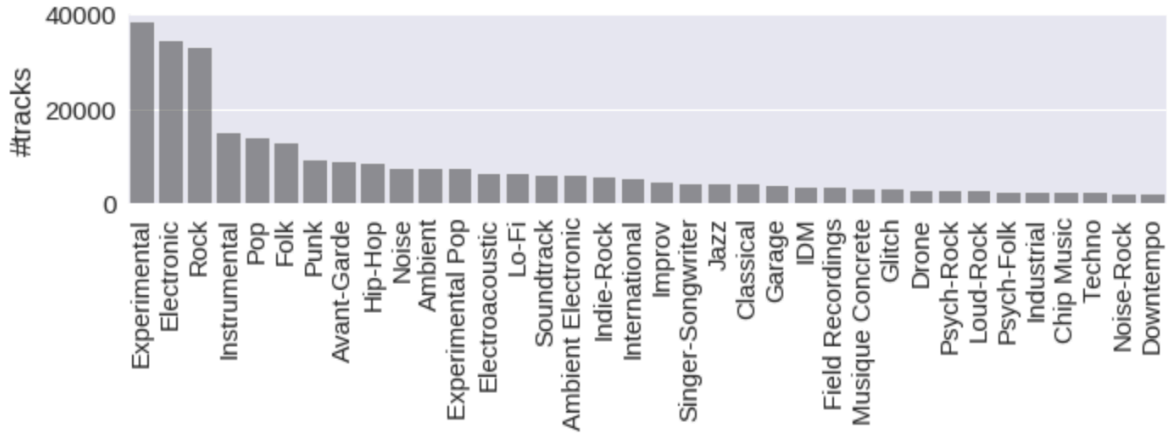

Fig.1. Track distribution for various dataset genres.

Recognizing this imbalance, we attempted one-hot encoding across all genre labels. However, testing the model's performance was a challenge in this scenario, due to difficulty in accurately detecting multiple genres for a single track. For example - A track might be “Rock” and “Pop,” but our model predicting it only as “Pop” would reduce our performance metrics despite correctly predicting one genre. We experimented with SMOTE (Synthetic Minority Over-Sampling Technique), aiming to synthetically balance minority genres but ended with unsatisfactory results and performance.

To simplify our pursuit of the genre classification task, we limited the scope of our model’s targets to five popular and commonly occuring genres: Rock, Electronic, Pop, Hip-Hop, and Folk. We opted not to classify tracks as Experimental, as this is an ambiguous genre that we were eager to attempt potential genre reclassification for. Each track (or dataset row) for our target genres contains representative audio features such as Mel-frequency cepstral coefficients (MFCCs), spectral contrast, and chroma vectors. Outside of the limitation on the scope of target classes, minimal data preprocessing was done. Null and missing values were negligible and remained in our dataset because we wanted our model to have access to as much training data as possible. However, data augmentation occurred in the form of adding additional feature values, which we discuss in the following section.

2.1.2. Feature Engineering

Experimentation with EchoNest and GTZAN Data:

We explored using EchoNest features, but dropped the idea when we realized Spotify deprecated its API, which would hinder feature extraction for new songs and make the solution difficult to scale. We also experimented with the GTZAN dataset, attempting to extract features and merge it with our dataset; however, results were unsatisfactory due to limited data volume and lack of metadata (such as track names), making it unsuitable for recommendation tasks.

Incorporating the Mel Spectrogram Feature:

In order to augment our training data, we pursued the extraction of Mel spectrograms from the raw audio files provided within the FMA dataset. A Mel spectrogram is a representation of a sound signal mapped onto the Mel scale, which approximates human perception of pitch [2]. These spectrograms capture both temporal and spectral characteristics that are important for identifying the unique characteristics that define a song. By incorporating an audio feature as such, our expectation is that our model will be able to more accurately distinguish nuances between genres that are traditionally difficult to perceive. The tradeoff with the pursuit of this method, and a longstanding problem in MIR in general, is that extracting Mel spectrograms is tedious and computationally intensive. As the corpus of training data that a genre classifier may be trained on grows, the pursuit of this method becomes increasingly less appealing.

```
[26]: async def compute_features(tid,zipe):

          features = pd.Series(index=columns(), dtype=np.float32, name=tid)

          warnings.filterwarnings('error', module='librosa')

          def feature_stats(name, values):
              features[name, 'mean'] = np.mean(values, axis=1)
              features[name, 'std'] = np.std(values, axis=1)
              features[name, 'skew'] = stats.skew(values, axis=1)
              features[name, 'kurtosis'] = stats.kurtosis(values, axis=1)
              features[name, 'median'] = np.median(values, axis=1)
              features[name, 'min'] = np.min(values, axis=1)
              features[name, 'max'] = np.max(values, axis=1)

          try:
              filepath = get_audio_path(tid)
              file = await getFile(zipe,filepath)
              x, sr = librosa.load(filepath, sr=None, mono=True)  # kaiser_fast
              stft = np.abs(librosa.stft(x, n_fft=2048, hop_length=512))
              assert stft.shape[0] == 1 + 2048 // 2
              assert np.ceil(len(x)/512) <= stft.shape[1] <= np.ceil(len(x)/512)+1
              del x

              f = librosa.feature.melspectrogram(S=stft)
              feature_stats('mel_spec', f)
              del stft
              os.remove("./"+filepath)

          except Exception as e:
              print('{}: {}'.format(tid, repr(e)))

          return features
```

Fig. 2. Code snippet to extract Mel spectrogram features.

Figure 2 illustrates the process of downloading each audio file in the dataset and generating a dataframe of statistical values extracted from an analysis of each track's Mel spectrogram. These values match the format of the original dataset features, allowing us to merge them efficiently. However, a considerable amount of storage is necessary to extract these features in parallel. We solved this problem by extracting the features of each audio file sequentially at the expense of a long runtime. This tradeoff is important to consider as the scale of input data grows.

2.1.3. Baseline Models

The authors of the FMA dataset provided baseline accuracies for genre classification of the top 8 genres in the dataset, utilizing logistic regression, KNN, and SVM models. However, because we limited the scope of our project to five target genres, we established a new baseline accuracy using logistic regression for this particular data subset. We selected this model for our baseline to maintain consistency with the curators of the FMA dataset and because logistic regressors are simple models that are often effective for illustrating a lower bound of performance for complex tasks. For baseline performance, our model accuracy is 68%.

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
Electronic	0.64	0.75	0.69	1874
Folk	0.60	0.37	0.46	561
Hip-Hop	0.68	0.41	0.51	711
Pop	0.36	0.02	0.03	466
Rock	0.71	0.87	0.78	2836
accuracy			0.68	6448
macro avg	0.60	0.48	0.50	6448
weighted avg	0.65	0.68	0.64	6448

Fig. 3. Classification report for baseline logistic regression model.

An important metric to note is that the f1-score of classification for the pop genre is particularly poor. This is an area of improvement we can focus on and potentially achieve success with through effective model selection for our final model.

2.1.4. Model Selection

Due to the high dimensional nature of audio data, we opted to experiment with tree-based and ensemble algorithms such as Random Forest, which generally perform well with datasets with such characteristics [3]. Immediately, we were greeted with promising results. The accuracy of our model increased by 7% compared to our baseline model, without any hyperparameter tuning. However, the predictive capabilities of our model for the Pop genre specifically continued to struggle. Our key objective moving towards our final model was to implement methods to make our classification of these five genres more well rounded.

```
print(classification_report(y_test, c_y_pred))
```

	precision	recall	f1-score	support
Electronic	0.67	0.84	0.75	1874
Folk	0.69	0.58	0.63	561
Hip-Hop	0.86	0.45	0.59	711
Pop	0.50	0.02	0.04	466
Rock	0.78	0.89	0.83	2836
accuracy			0.74	6448
macro avg	0.70	0.56	0.57	6448
weighted avg	0.73	0.74	0.71	6448

Fig. 4. Classification report for the initial random forest model.

```
[ ]  from sklearn.utils.class_weight import compute_class_weight

     import numpy as np
     classes = np.unique(y_train)
     class_weights = compute_class_weight('balanced', classes=classes, y=y_train)

     sample_weights = np.array([class_weights[label] for label in y_train])
```

Fig. 5. Code snippets for implementation of class weights.

Firstly, we utilized class weights for our target classes to better emulate a balanced training dataset, as even after limiting the scope of our target classes, data imbalance continued to be an issue. Furthermore, we decided to utilize the XGBoost model, which builds upon decision trees. Unlike the random forest algorithm, which uses feature bagging, XGBoost uses a boosting method, where each tree is built upon the previous tree, and each succeeding tree aims to correct the errors of the preceding trees [4]. By combining these methods, we aim to not only improve our model's overall accuracy but also particularly improve performance on target classes that suffer from a lack of data. An evaluation of the final model will be discussed in the upcoming Section 2.3.

2.1.5. Model Pipeline

We employed a traditional machine learning model pipeline, shown in Fig. 6. Important details of the pipeline components are discussed below.

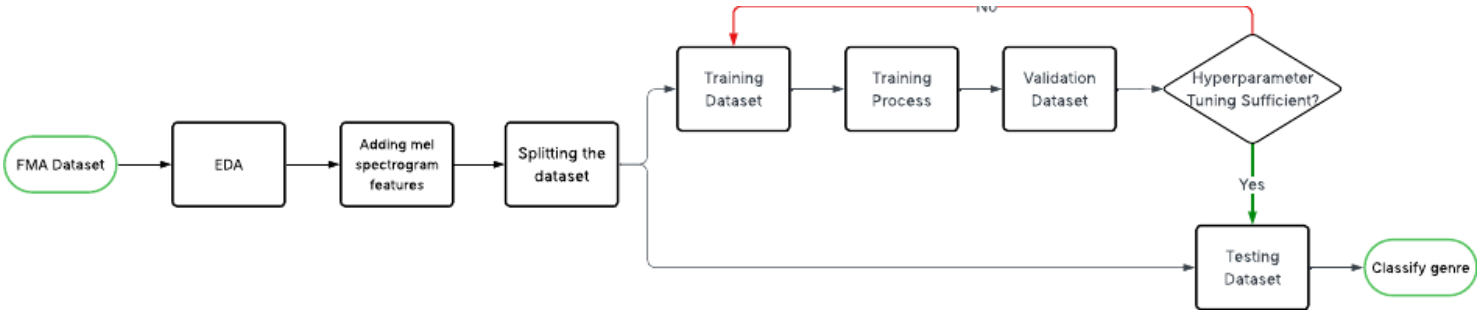

Fig. 6. The model pipeline

1. Exploratory Data Analysis: after acknowledging the data imbalance issues present in our dataset, at this stage, we limited the scope of our target classes to five genres.
2. Feature Engineering with Mel spectrogram: in order to provide our model with more informative features that can be utilized during training, we extracted Mel spectrogram values from audio files to bolster our dataset.
3. Train/Test/Validation Split: we employed a standard dataset split to fairly assess our model's performance and combat model overfitting.
4. Model Training and Hyperparameter Tuning: we experimented with different hyperparameters to improve overall model performance and further combat potential model overfitting on the training dataset.
5. Testing and Evaluation: we assessed the various performance metrics discussed in the upcoming Section 2.2 and compared them to our baseline model performance. Upon suitable performance, we can begin to implement our song recommendation system using our classification model.

2.1.6. Recommender System

Our goal with this stage of the project is to exhibit nuance through our ability to recommend songs that are not easily classified into any of our model's target classes. As such, we utilized our model to reclassify all of the songs in our dataset that did *not* belong to one of our target classes, into our target classes.

```
recommendation_data = reclassify_and_get_recommendation_data()
```

```
recommendation_data
```

track_id	Rock	Electronic	Pop	Hip-Hop	Folk	listens	favorites	interest
20	0.257002	0.125194	0.163086	0.037696	0.417022	361	0	978
26	0.118131	0.064750	0.321094	0.045405	0.450619	193	0	1060
30	0.142537	0.122378	0.454565	0.035769	0.244751	612	0	718
46	0.034783	0.038007	0.361828	0.002768	0.562613	171	0	252
48	0.212671	0.083119	0.235224	0.016218	0.452768	173	0	247
...	...	...	...	...	...	...	...	...
155309	0.399393	0.099351	0.307394	0.061082	0.132779	79	0	90
155310	0.450586	0.036552	0.297130	0.080286	0.135446	84	0	94
155311	0.624200	0.025582	0.142246	0.028306	0.179666	171	0	187
155312	0.253352	0.308128	0.226864	0.148033	0.063623	219	0	230
155320	0.837859	0.046604	0.099830	0.010832	0.004875	705	1	972

Fig. 7. Recommendation data obtained from dataset reclassification.

This reclassification is illustrated above in the form of a probability distribution that a particular song belongs to one of our model's given target classes. By utilizing this data, alongside provided track metadata such as listens, favorites, and interest, we can generate song recommendations from our dataset given a particular input song as a YouTube URL. We achieve this by finding the most similar songs in our dataset to the input song using cosine similarity, and then ranking our recommendations based on a popularity score considering listens, favorites, and interest. This is illustrated in the code snippets below.

```
[ ] from sklearn.preprocessing import MinMaxScaler

    scaler = MinMaxScaler()
    normalized_features['normalized_listens'] = scaler.fit_transform(recommendation_data[['listens']])[:, 0]
    normalized_features['normalized_favorites'] = scaler.fit_transform(recommendation_data[['favorites']])[:, 0]
    normalized_features['normalized_interest'] = scaler.fit_transform(recommendation_data[['interest']])[:, 0]
```

```
[ ] recommendation_data["popularity_score"] = (0.5 * normalized_features["normalized_listens"]
                + 0.3 * normalized_features["normalized_favorites"]
                + 0.2 * normalized_features["normalized_interest"])
```

- Getting top 10 recommendations

```
[ ] top_recommendations = recommendation_data.nlargest(10, "cosine_similarity").copy()
```

- Sorting top 10 recommendation based on popularity.

```
[ ] final_recommendations = top_recommendations.sort_values("popularity_score", ascending=False)
```

Fig. 8. Calculation of popularity score and utilization of cosine similarity and popularity score to provide top 10 recommendations.

```
[ ] track_title_artist.loc[final_recommendations.index]
```

track_id	title	artist	genre_top	listens	favorites	interest
140907	Worries	OpVious	NaN	12678	4	13977
92992	Analytix	Tab & Anitek	NaN	1831	5	2888
123630	The Life and Death of The Party (Edit)	Pimpstrut Local #13	Soul-RnB	1997	1	2373
116209	Darkside Imperials	Toussaint Morrison	NaN	1414	1	1840
34661	Blender Tzivoni	51%	NaN	280	1	664
8175	polybag MFR	James Amoeba	NaN	116	1	225
15922	We're moving	Kukkiva Kunnas	NaN	287	0	760
146190	Revolutionary Letters Part 1	Symbol Of Subversion	NaN	233	0	283
71528	Laxatif Rudimentaire	Doctor Bux	Experimental	69	0	257
154096	To Be	Brian Routh	NaN	11	0	13

Fig. 9. Final recommendation output for “Bank Account” by 21 Savage.

2.2. Performance Metrics

2.2.1. Classification Metrics

To assess our model’s performance, we decided to use the following metrics:

- **Accuracy**: Model accuracy is a standard metric to evaluate model performance. Accuracy measures how often the model predicts the right outcome. We are only using accuracy scores to compare our model's performance to the baseline models' performance; we cannot rely solely on accuracy to evaluate the model, since accuracy is not a suitable metric for unbalanced datasets[1].
- **Precision**: Precision is the number of correctly predicted positive classifications over every positive classification (TP/TP+FP).
- **Recall**: Recall is the number of correctly predicted actual positives over all the actual positive classifications (TP/TP+FN).
- **F1-Score**: F1-score is the harmonic mean of precision and recall. It provides a better measure of a model's performance when dealing with imbalanced classes because it balances the trade-off between missing rare genres (low recall) and falsely predicting them (low precision) [5].
- **Confusion Matrix**: Since our project involves multiclass classification for genres, a confusion matrix is used to show the proportion of classification for every genre. It allows us to gain deeper insights into whether a certain genre is harder to classify.

2.2.2. Recommendation Metrics

For recommendation, cosine similarity was our metric of choice.

- **Cosine Similarity**: Cosine similarity is a measure of the similarity between two vectors computed by taking the cosine of the angle between them. It is a commonly utilized metric for recommendation systems as it enables the capability to distinguish the best possible options for recommendation out of a large corpus of data based on their similarity to user preferences [6].

2.3. Evaluation

2.3.1. Classification Evaluation

To assess our model's performance, we utilized the evaluation metrics discussed in the prior section: accuracy, precision, recall, F1-score, and confusion matrix analysis.

[1]Google, "Classification: Accuracy, recall, precision, and related metrics"

```
[16]: y_pred_train = model.predict(x_train)
```

```
[17]: print(classification_report(y_pred_train,y_train))
```

```
              precision    recall  f1-score   support

           0       0.89      0.98      0.94      9032
           1       0.93      0.94      0.93      6504
           2       1.00      0.80      0.89      2027
           3       0.99      0.92      0.95      2648
           4       0.99      0.83      0.91      2356

    accuracy                           0.93     22567
   macro avg       0.96      0.89      0.92     22567
weighted avg       0.93      0.93      0.93     22567
```

Fig. 10. Training Metric of XGBoost

```
[18]: y_pred_val = model.predict(x_val)
```

```
[20]: print(classification_report(y_pred_val,y_val))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1956
           1       0.82      0.81      0.81      1411
           2       0.41      0.40      0.41       358
           3       0.81      0.77      0.79       576
           4       0.83      0.64      0.72       535

    accuracy                           0.79      4836
   macro avg       0.74      0.70      0.72      4836
weighted avg       0.79      0.79      0.79      4836
```

Fig. 11. Validation Metric of XGBoost

```
[21]: y_pred = model.predict(x_test)
```

```
[26]: print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.82      0.86      2136
           1       0.79      0.83      0.81      1391
           2       0.41      0.43      0.42       348
           3       0.77      0.79      0.78       546
           4       0.62      0.79      0.69       415

    accuracy                           0.79      4836
   macro avg       0.70      0.73      0.71      4836
weighted avg       0.80      0.79      0.79      4836
```

Fig. 12. Testing Metric of XGBoost

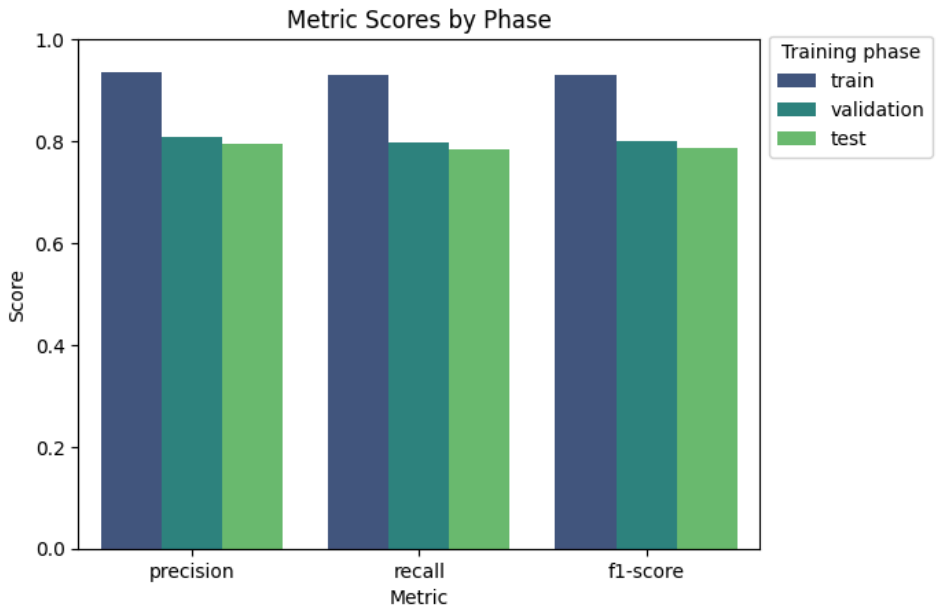

Fig. 13. Precision, recall, and F1-scores of the XGBoost model for the training, validation, and testing iterations.

Figure 13 shows that the precision, recall, and F1-score metrics are high during the training phase but drop slightly during validation and testing. This difference suggests that while our model fits the training data very well, it experiences overfitting when generalized to new data. Regardless, these validation and test scores remain relatively high, all around .80, which is considerably higher than our baseline model.

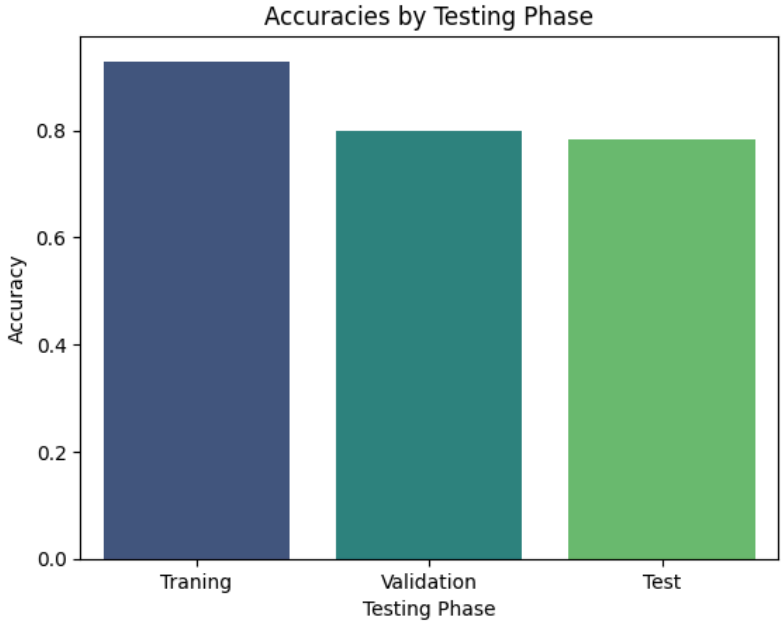

Fig. 14. Accuracy scores of the XGBoost model for the training, validation, and testing iterations.

Figure 14 displays model accuracy scores across phases. Training accuracy is above 90%, while validation and testing accuracies are slightly lower at 79%. This again points towards mild overfitting but still reflects a strong generalization of the data compared to the baseline model referenced in Section 2.1.3 which achieves 68% testing accuracy.

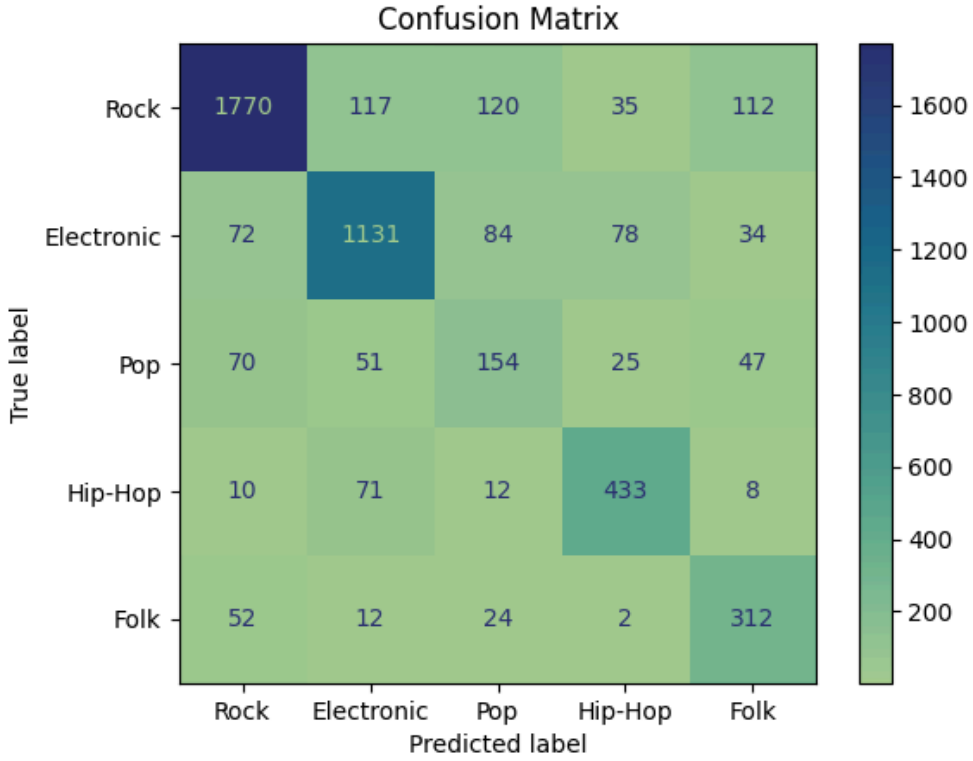

Fig. 15. Confusion matrix.

The confusion matrix, in Figure 15 provides us with deeper insights to the details of our classification model. Rock and Electronic are the two genres that were classified most accurately with large diagonal counts of 1770 and 1131 respectively. Pop exhibits more confusion, with frequent misclassification into Rock and Electronic genres. This aligns with our observation from the classification report, in Table 1, where Pop had the lowest precision (0.39) and recall (0.44). Hip-Hop and Folk also show good classification performance, although Folk seems to have minor confusion with Pop and Rock.

Table 1. Test classification report for XGBoost.

Genre	Precision	Recall	F1-Score
Rock	0.91	0.82	0.86
Electronic	0.79	0.83	0.81
Pop	0.41	0.43	0.42
Hip-Hop	0.77	0.79	0.78
Folk	0.62	0.79	0.69

These results indicate that although genre prediction is generally successful certain genres, especially Pop, are harder to accurately predict. This is likely due to the limited number of Pop samples and their audio similarity to other genres.

Switching from our baseline logistic regression model to XGBoost significantly improved recall and F1-scores across all genres. In addition, feature augmentation using Mel spectrograms provided better input information compared to basic tabular features, allowing the models to better capture the nuances of the audio signals. Restricting the classification task to the top five genres also contributed to a stronger performance by simplifying the problem scope and minimizing some of the effects of our initially heavily imbalanced dataset.

One of the shortcomings we ran into was the gap between training and testing performance which suggest slight overfitting and an inflation of evaluation scores. To address this issue, we could collect additional data for underperforming target classes such as Pop, balancing the dataset and ideally improving classification performance. Another challenge was that the use of Mel spectrogram features and boosting methods like XGBoost increased computation time compared to simpler models; this could be a challenge for scaling this approach to larger datasets.

2.3.2. Recommendation Evaluation

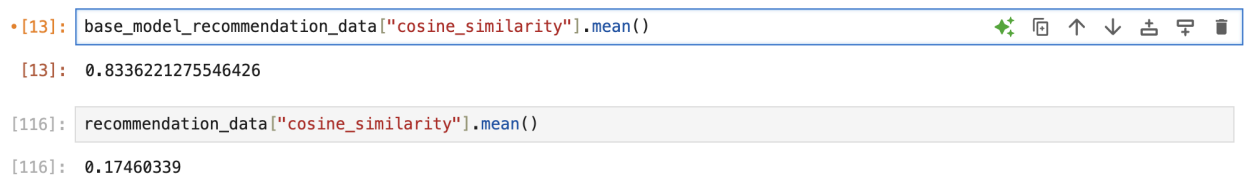

Fig. 16. Comparing cosine similarities between baseline and optimized models.

To assess the quality of our recommendation system, we computed the mean cosine similarity of possible recommended tracks for both the baseline and final models. A higher cosine similarity implies that the model tends to recommend tracks that are broadly similar to many others, while a lower cosine similarity suggests the model is better at distinguishing nuances between different tracks and genres.

Illustrated above, the baseline model had an average cosine similarity of 0.8336, indicating that it tends to deem a large majority of tracks in the dataset as very similar to each other regardless of their actual auditory distinctions. In contrast, our final XGBoost model achieved a much lower mean cosine similarity value of 0.1746. This large drop suggests that the final model learned to distinguish nuanced differences between songs of different genres more effectively, allowing it to make more targeted and meaningful recommendations.

Overall, the lower cosine similarity score mean for the final model shows that our recommendation system becomes more sophisticated and aware of different genres compared to the baseline approach.

2.4. Recommendations / Future Work

Following the results we procured, there are several avenues for enhancing both the genre classification and the recommendation system. Below are few suggested improvements that can be done in the future:

- **Using deep learning approaches** : Utilizing deep learning models like convolutional neural networks (CNNs) or recurrent neural networks (RNNs), on Mel spectrograms would greatly improve feature extraction and classification to a larger extent when compared to traditional machine learning methods.

- **Handling data imbalance** : Even though we attempted data augmentation techniques like SMOTE and limited classification to top five genres, we still had some imbalanced data which led us to lower accuracy performance with infrequently targeted genres. By using additional data balancing techniques like generating synthetic samples or even adjusting the model's loss function, we could ensure there is more consistent prediction capability across all target genres.

- **Incorporating diverse datasets** : The FMA dataset did serve as a strong foundation, even though it had some limitations such as class

imbalancing and null target genres. Integrating other labeled music datasets like the GTZAN Dataset [7] could significantly improve the classification model's performance by providing more data for underrepresented target classes. However, for our particular use case this was not viable because the GTZAN dataset did not provide track names, which presented an issue for the recommendation stage of our pipeline.

- **Conducting more robust feature extraction** : Looking beyond the Mel spectrogram feature, future improvements could include extracting more detailed relevant features. Techniques like Constant-Q transform (CQT) can be used to capture more accurate pitch and frequency information, whereas chroma features may highlight harmonic structures[8] which are unique to different genres.

- **Enhancing recommendation system** : While our recommendation system was based on cosine similarity, it helped establish a working baseline, it was primarily limited to songs within the training dataset. This may have led to slight overfitting and reduced generalizability. Future improvements could include expanding the recommendation scope beyond the top five genres to capture a wider range of musical styles. Additionally, incorporating user-specific data such as listening history and preferences would allow for a more personalized and dynamic recommendation experience that better adapts to individual user behavior.

3. Project Management

3.1. Final Schedule and Task Distribution (Gantt Chart)

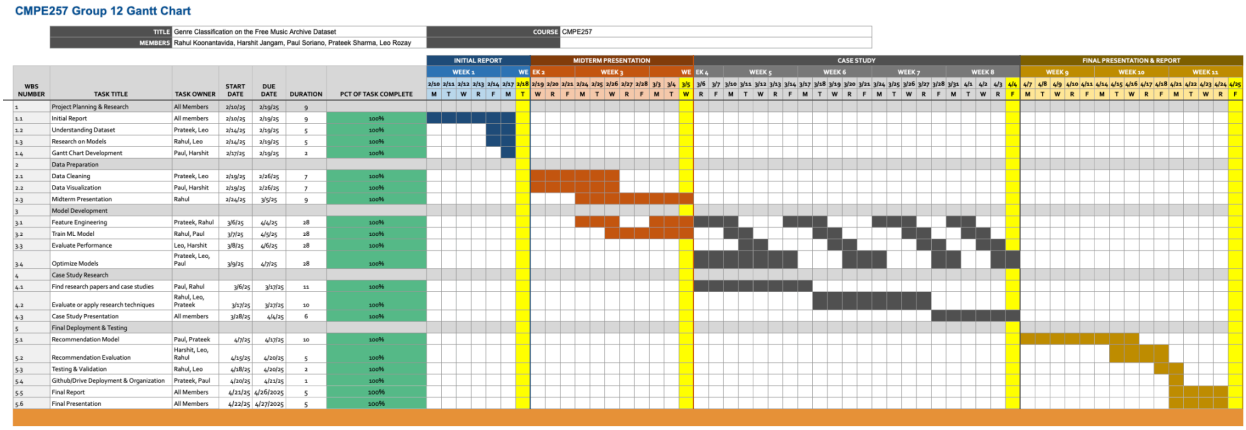

Fig. 17. Gantt Chart.

The final task distribution for our project is illustrated in the Gantt Chart [9] linked above. The phases and highlighted milestones for the Gantt chart are representative of the course deliverables throughout the semester: initial report, midterm presentation, case study presentation, and final presentation.

3.2. Challenges

One of the challenges we faced during this project was aligning every group member's schedule, especially since some of us work full time. Managing deadlines was a challenge but we were able to overcome it by scheduling regular meetings and staying up to date by posting in our group discord. Additionally, from a technical perspective, one of the challenges we faced was that Mel spectrogram extraction and model training was time consuming. Limited time and resources also constrained how much we could test different models or additional data augmentation.

3.3. Learning Outcomes

One of the major takeaways we learned from this project was the importance of front loading project responsibilities early in the timeline. Completing preliminary work sooner allows us to have a buffer in case of technical delays. We also learned the importance of communication and how to successfully operate as a team to deliver a product. Working in a team environment is a skill that will serve us for a long time and certainly will be useful in the industry.

References

[1] M. Defferrard, K. Benzi, P. Vandergheynst, X. Bresson. (2017, October). "FMA: A Dataset for Music Analysis" *International Society for Music Information Retrieval Conference (ISMIR*) [Online]. Available: https://archives.ismir.net/ismir2017/paper/000075.pdf

[2] "Introduction to audio data - Hugging Face Audio Course." https://huggingface.co/learn/audio-course/en/chapter1/audio_data

[3] D. Ghosh and J. Cabrera, "Enriched random forest for high dimensional genomic data," IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 19, no. 5, pp. 2817–2828, Jun. 2021, doi: 10.1109/tcbb.2021.3089417

[4] "XGBoost Advantages and Disadvantages (pros vs cons) | XGBoosting." https://xgboosting.com/xgboost-advantages-and-disadvantages-pros-vs-cons

[5] N. Buhl, "F1 Score in Machine Learning," Nov. 06, 2024. https://encord.com/blog/f1-score-in-machine-learning/

[6] Algolia, "Cosine similarity: what is it and how does it enable effective (and profitable) recommendations?," Algolia, Dec. 07, 2024. https://www.algolia.com/blog/ai/cosine-similarity-what-is-it-and-how-does-it-enable-effective-and-profitable-recommendations

[7] "GTZAN Dataset - Music Genre Classification," Kaggle, Mar. 24, 2020. https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification

[8] M. Müller, "Information Retrieval for Music and Motion," Springer, 2007. https://www.researchgate.net/publication/220694342_Information_Retrieval_for_Music_and_Motion

[9] *Gantt Chart Template* [Online]. Available: https://docs.google.com/spreadsheets/d/1JcX4sHAuBRGsbXIgktxj5n72sMyFQutQyqJ7R_xQCCU/edit?gid=0#gid=0