CMPT353 - Song Popularity Prediction Project Report

Summer 2024

Analyzing and Modeling Audio Features

Group: CMPT353_ALLIN

Lihao Qian 301406445

Lingjie Li 301417109

Qiqi Liu 301256230

Introduction

Music is an essential part of our lives, influencing our emotions, behavior, and even social trends. However, the factors that contribute to a song's popularity can vary significantly from year to year. This report aims to examine the audio features that have the greatest impact on music popularity. By analyzing data extracted from Spotify and combining it with Billboard Chart information, we aim to improve the accuracy of our analysis and gain a better understanding of what makes songs popular.

Leveraging machine learning, we intend to develop models that can predict the popularity level of a song based on its audio features. By inputting a song's features into these trained models, which are built using historical data from Spotify and Billboard, we can forecast its popularity and assess its potential to win awards.

Data Sources

Spotify provides a full set of audio features for each track that can be accessed through its API. These features include:

1. **danceability**: Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.
2. **energy**: A measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
3. **key**: The key the track is in. Integers map to pitches using standard Pitch Class notation.
4. **loudness**: The overall loudness of a track in decibels (dB).
5. **mode**: Modality of the track, indicating the type of scale (major or minor).
6. **speechiness**: Detects the presence of spoken words in a track.
7. **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
8. **instrumentalness**: Predicts whether a track contains no vocals.
9. **liveness**: Detects the presence of an audience in the recording.
10. **valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
11. **tempo**: The overall estimated tempo of a track in beats per minute (BPM).
12. **type, id, uri, track_href, analysis_url**: Various identifiers and links related to the track.

13. **duration_ms**: The duration of the track in milliseconds.
14. **time_signature**: An estimated overall time signature of a track.

The Billboard Hot 100 Chart is a well-recognized measure of a song's popularity in the United States. The Billboard dataset includes:

1. **url, WeekID**: Identifiers for the week.
2. **Week Position**: The position of the song in the chart for that week.
3. **Song, Performer**: The title and performer of the song.
4. **SongID, Instance**: Unique identifiers for the song.
5. **Previous Week Position, Peak Position**: Historical chart positions.
6. **Weeks on Chart**: The number of weeks the song has been on the chart.

Feature Extraction from Spotify API

To facilitate our analysis and model predictions, we leveraged the Spotify API to extract detailed audio features for each song. This process allowed us to obtain a comprehensive set of attributes directly from Spotify, ensuring consistency and accuracy in the features used for our machine learning models.

Process of Feature Extraction

1. **API Connection**: We established a connection to the Spotify API using appropriate authentication methods, which included obtaining access tokens to authorize our requests.
2. **Data Retrieval**: For each song, we used the Spotify track endpoint to fetch detailed audio features. This included querying the API with the song's unique identifier (Spotify URI or track ID).
3. **Data Parsing**: The returned JSON data from the API was parsed to extract the relevant features, which were then stored in a structured format for analysis.
4. **Data Integration**: The extracted features were integrated into our dataset, which included additional information from Billboard charts to enhance the predictive power of our models.

Data Cleaning and Preparation

To improve the accuracy of our analysis, we first cleaned each dataset individually before combining them. The following steps were taken:

Spotify Data Cleaning

1. Normalization of Song Titles and Artist Names: Song titles and artist names were converted to lowercase and non-alphabetic characters were removed to ensure consistency and matchability between datasets.
2. Removal of Unnecessary Columns: Columns such as "explicit", "id", "mode", and "release_date" were deemed unnecessary for our analysis and were removed. Additionally, all missing and duplicate values were identified and removed.

Billboard Data Cleaning

1. Normalization of Song Titles and Artist Names: Similar to Spotify, song titles and artist names were converted to lowercase and non-alphabetic characters were removed.
2. Removal of Unnecessary Columns: Columns that were not relevant to our analysis, such as "url", "Instance", and "SongID", were removed. Missing and duplicate values were also cleaned.
3. Normalization of Chart Data: In the Billboard charts, a lower position number (closer to 1) indicates higher popularity, whereas in Spotify, a higher rank (closer to 100) indicates higher popularity. We reverse normalized the Billboard data to align with Spotify's ranking system.
4. Calculation of Aggregated Metrics: We calculated the maximum Weeks on Chart, average Previous Week Position, and maximum Week Position for each song and performer. Rows containing NaN values were deleted to maintain data integrity.

Combining the Datasets

1. Creation of Combination Key: For better matching between the datasets, we created a combination key from song titles and artist names. This key was used to merge the datasets.
2. Recalculation of Popularity Score: Since a single Spotify popularity score may not be completely trustworthy (e.g., some songs with low Spotify popularity appear in the

Billboard Hot 100 chart), we recalculated a composite score based on the average weights of ‘Weeks on Chart’, ‘Average Previous Week Position’, and ‘Week Position’ from Billboard. This composite score was used to filter out untrustworthy data that couldn't be matched in Spotify.

3. We also used the "qcut" function in pandas to split the popularity into 3 quartiles, ensuring that each category had roughly the same number of values. The label 0 indicates low popularity, 1 indicates medium popularity, and 2 indicates high popularity. This made it easier for our model afterwards to predict these categories.
4. Final Data Preparation: Finally, unnecessary columns were removed to obtain the cleaned and prepared data necessary for our analysis.

By cleaning each dataset individually and then combining them, we ensured that the final dataset is comprehensive and ready for analysis.

Exploratory Data Analysis

Correlation Analysis

A correlation matrix was computed to identify relationships between the numerical features in the dataset. The results were visualized using a heatmap.

Correlation Matrix and Heatmap: The correlation matrix heatmap (Figure 1) provides a visual representation of the relationships between different audio features and song popularity. The heatmap uses color gradients to indicate the strength and direction of correlations, with darker shades representing stronger correlations. Red color indicates positive correlation, the closer the correlation coefficient is to 1, the stronger the positive correlation between the two variables. Blue: negative correlation, the closer the correlation coefficient is to -1, the stronger the negative correlation between the two variables. White color indicates no correlation, the correlation coefficient is close to 0, which means there is almost no correlation between the two variables.

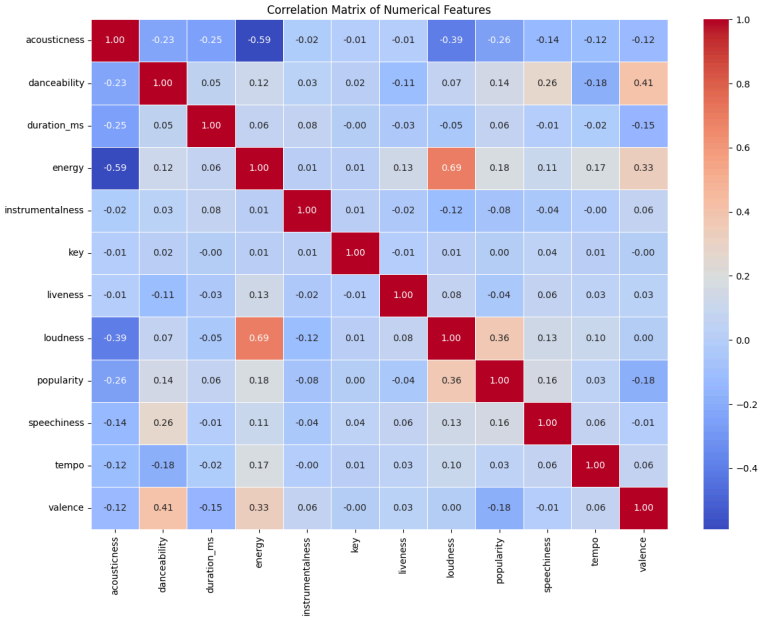

Figure 1: Heatmap of Numerical Features

Regression Analysis

The regression plots (Figure 2) provide a visual representation of the positive correlations between different audio features and song popularity. Each plot demonstrates how individual audio features influence the popularity of a song. The scatter plots use green dots to represent individual data points and a black regression line to indicate the trend of the relationship.

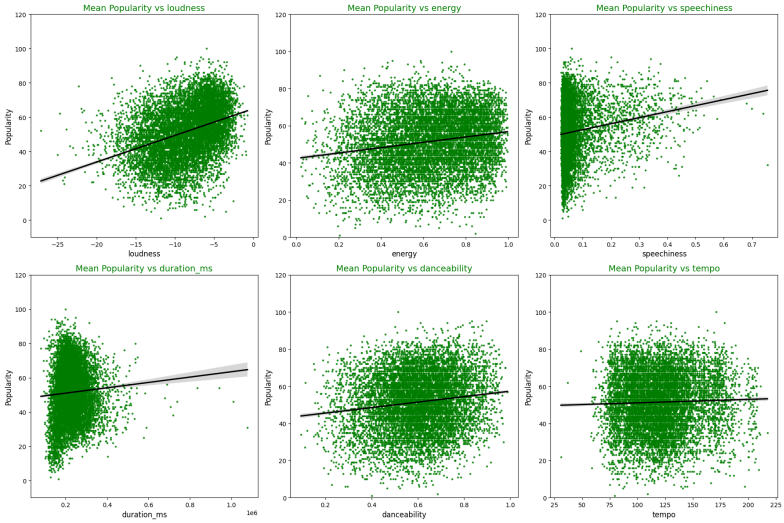

Figure 2: Positive Regression Analysis of Features with Popularity

Key Findings

The correlation matrix, heatmap and regression plots reveal several key insights:

- Loudness: The strongest positive correlation with popularity (0.362830). The regression line in the graph also shows that the louder the song, the more popular it usually is. This may be due to the fact that louder songs are more aurally impactful and more likely to attract the attention of the listener.
- Energy: Shows a moderate positive correlation with popularity (0.182318). High-energy songs may be more popular because they are usually more energetic and upbeat.
- Speechiness: Also positively correlated with popularity (0.160931), suggesting that songs with more spoken words can be more popular.
- Danceability: Has a positive correlation (0.139397), indicating that danceable songs are likely to be more popular.
- Acousticness: Shows a negative correlation with popularity (-0.261853), suggesting that more acoustic songs are generally less popular.
- Valence: Also negatively correlated (-0.179206), implying that happier-sounding songs may not always be the most popular.
- Liveness and Instrumentalness: Both show a slight negative correlation, suggesting that live and instrumental songs may not be as popular as others.

Feature Importance

From the correlation analysis, we can identify which features are most influential in determining a song's popularity. The features with the highest absolute correlation values are loudness, energy, and speechiness, indicating that these characteristics are significant predictors of a song's success.

Correlation with Popularity: The specific correlations with popularity are listed below (Figure 3):

```
Correlation with Popularity:
 popularity          1.000000
loudness             0.362830
energy               0.182318
speechiness          0.160931
danceability         0.139397
duration_mins        0.062887
tempo                0.033210
key                  0.004468
liveness            -0.042276
instrumentalness    -0.081350
valence             -0.179206
acousticness        -0.261853
Name: popularity, dtype: float64
```

Figure 3: Correlation with Popularity

Data Transformer

In this project we use two preprocessing steps for different machine learning models, one is for KNeighborsClassifier and SVC, another preprocessing step is for RandomForestClassifier. The preprocessing consists of using ColumnTransformer in sklearn.compose to scale the numerical features and encode the categorical features. The following is the analysis of the data transformation method based on the given code:

1. **KNeighborsClassifier and SVC Preprocessing**

   For KNeighborsClassifier and SVC, the preprocessing step starts with feature extension, where the target variable y comes from the data frame, and the feature set X is obtained by removing the popularity column and the popularity_level column. The dataset is then split into a training set and a test set using train_test_split with a test size of 25% and a random state of 50 to ensure repeatability.

   To solve the class imbalance problem in the training set, we employ the Synthetic Minority Oversampling Technique (SMOTE).SMOTE generates synthetic samples of the specified few classes, thus balancing the distribution of classes and improving the performance of the model on the imbalanced data (*SMOTE — Version 0.12.3*, n.d.). The ColumnTransformer is set using MinMaxScaler to scale the numerical features to be sure they are in the same range. This scaling is important for distance-based algorithms such

as KNeighborsClassifier and SVC, as it prevents features with larger ranges from dominating the distance calculation. In addition, a OneHotEncoder is used to encode the classification feature keys into a binary format that can be correctly interpreted by the model (*OneHotEncoder*, n.d.).

2. **Random Forest Classifier Preprocessing**

   The preprocessing steps for RandomForestClassifier are similar in that the target variable y and feature set X are extracted from the data frame. "train_test_split" is used to split the dataset into a training set and a test set. The ColumnTransformer is then set up using StandardScaler to normalize the numerical features to have a mean of 0 and a standard deviation of

   Similar to the preprocessing of KNeighborsClassifier and SVC, OneHotEncoder is used to encode the classification feature keys to ensure their correct representation in the model (*OneHotEncoder*, n.d.). StandardScaler is applied to the RandomForestClassifier because it can normalize the features without distorting the feature distribution, thus helping the model to handle features at different scales efficiently.

The preprocessing steps in both sets of models ensure that the digital features are appropriately scaled and encoded for classification. This consistency is important to maintain a uniform approach across models. The choice of MinMaxScaler for KNeighborsClassifier and SVC is appropriate because these algorithms are distance-based, whereas the choice of StandardScaler for RandomForestClassifier ensures numerical stability and correct feature presentation. In addition, applying SMOTE to the training data can solve the class imbalance problem and improve the model performance and generalization ability, especially for KNeighborsClassifier and SVC.

Model Development and Results

1. **K-Nearest Neighbors (KNN)**

K-Nearest Neighbors (KNN) is a simple, yet effective, non-parametric classification algorithm. It works by identifying the k-nearest data points to the query point and classifying the query point based on the majority class among its neighbors.

Model Performance:

The performance of the KNN model on the test set is summarized in Figure 4. The table provides precision, recall, and F1-score for each class, along with overall accuracy and macro-averaged metrics.

- **Accuracy**: The model achieved an accuracy of 43% on the test set.
- **Precision**: Precision values ranged from 0.34 to 0.52 across different classes.
- **Recall**: Recall values varied significantly, with the highest being 0.67.

```
Model: KNeighborsClassifier
Accuracy on Test Set for KNeighborsClassifier = 0.432114333753678
Train Score for KNeighborsClassifier = 0.7154989384288747
Test Score for KNeighborsClassifier = 0.432114333753678

              precision    recall  f1-score   support

           0       0.47      0.67      0.55       844
           1       0.34      0.38      0.36       747
           2       0.52      0.22      0.31       788

    accuracy                           0.43      2379
   macro avg       0.44      0.43      0.41      2379
weighted avg       0.45      0.43      0.41      2379
```

Figure 4: KNN Model Performance Metrics

Learning Curve:

Figure 5 shows the learning curve for the KNN classifier. The training score and cross-validation score are plotted against the number of training examples.

- **Training Score**: The training score remains high, around 70%, indicating that the model fits well to the training data.

- **Cross-Validation Score**: The cross-validation score is consistently lower than the training score, around 40%, suggesting that the model may be overfitting to the training data and not generalizing well to unseen data.

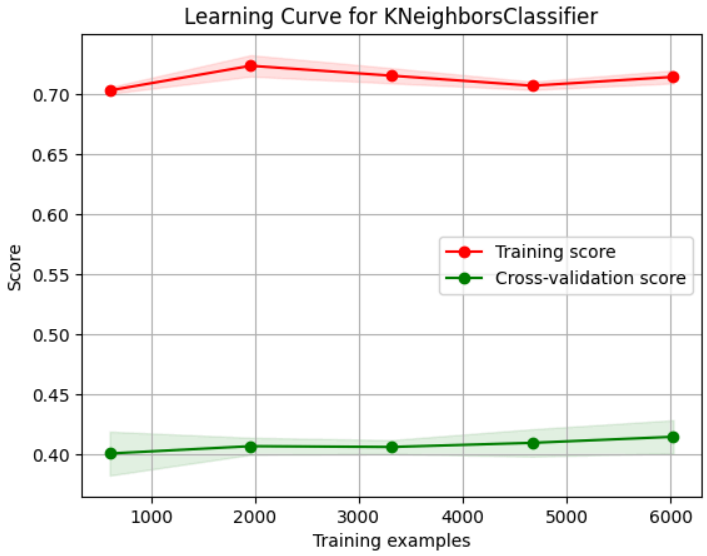

Figure 5: Learning Curve for KNeighborsClassifier

2. Support Vector Classifier (SVC)

Support Vector Classifier (SVC) is a powerful classification algorithm that works by finding the hyperplane that best separates the data into different classes. It is particularly effective for high-dimensional spaces.

Model Performance:

The performance of the SVC model on the test set is summarized in Figure 6. The table provides precision, recall, and F1-score for each class, along with overall accuracy and macro-averaged metrics.

- **Accuracy**: The model achieved an accuracy of 51% on the test set.
- **Precision**: Precision values ranged from 0.39 to 0.53 across different classes.
- **Recall**: Recall values varied, with the highest being 0.70.

```
Model: SVC
Accuracy on Test Set for SVC = 0.5086170659941152
Train Score for SVC = 0.535828025477707
Test Score for SVC = 0.5086170659941152

              precision    recall  f1-score   support

           0       0.53      0.70      0.60       844
           1       0.39      0.20      0.26       747
           2       0.53      0.60      0.56       788

    accuracy                           0.51      2379
   macro avg       0.48      0.50      0.48      2379
weighted avg       0.49      0.51      0.48      2379
```

Figure 6: SVC Model Performance Metrics

Learning Curve:

Figure 7 shows the learning curve for the SVC model. The training score and cross-validation score are plotted against the number of training examples.

- **Training Score**: The training score starts high but gradually decreases as the number of training examples increases, stabilizing around 53%.
- **Cross-Validation Score**: The cross-validation score improves slightly with more training examples, indicating that the model benefits from more data but still struggles to generalize well, stabilizing around 50%.

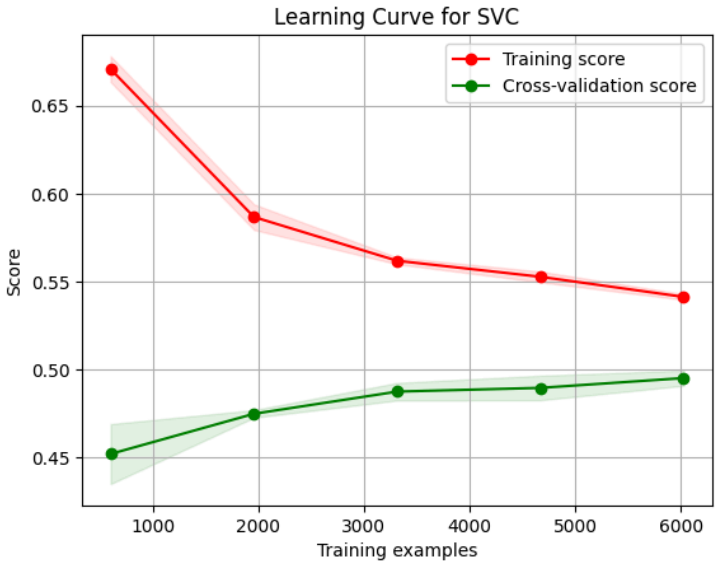

Figure 7: Learning Curve for SVC

3. Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes for classification tasks. It is known for its robustness and ability to handle large datasets with high dimensionality.

Model Performance:

The performance of the Random Forest model on the test set is summarized in Figure 8. The table provides the best parameters obtained through hyperparameter tuning, the train and test scores, the mean squared error, and the R-squared value.

- **Best Parameters**: The optimal parameters found were max_depth of 50, max_features set to 'sqrt', min_samples_leaf of 7, and n_estimators of 250.
- **Train Score**: The model achieved a train score of 81.34%, indicating that it fits the training data well.
- **Test Score**: The model achieved a test score of 52.16%, demonstrating its ability to generalize to unseen data.
- **Mean Squared Error**: The mean squared error was 0.879, reflecting the average squared difference between predicted and actual values.
- **R-squared**: The R-squared value was -0.283, indicating the proportion of variance explained by the model.

```
Best parameters: {'max_depth': 50, 'max_features': 'sqrt', 'min_samples_leaf': 7, 'n_estimators': 250}
Model train score: 0.8133669609079445
Model test score: 0.5216477511559479
Mean Squared Error: 0.8793610760823876
R-squared: -0.2828989697354638
```

Figure 8: Random Forest Model Performance Metrics

Learning Curve:

Figure 9 shows the learning curve for the Random Forest model. The training score and cross-validation score are plotted against the number of training examples.

- **Training Score:** The training score remains high, around 80%, indicating that the model fits well to the training data.
- **Cross-Validation Score:** The cross-validation score is consistently lower than the training score, around 50%, suggesting that the model may be overfitting to the training data and not generalizing well to unseen data.

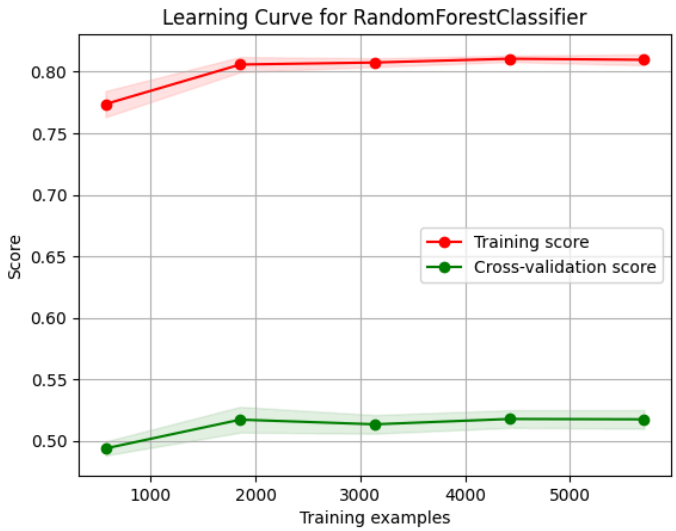

Figure 9: Learning Curve for RandomForestClassifier

4. K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm used to partition the dataset into k clusters, where each data point belongs to the cluster with the nearest mean.

Clustering Results:

The clustering results are visualized in Figure 10, which shows the distribution of data points in different clusters. The clusters are represented in different colors, and the data points are plotted based on their principal component analysis (PCA) components.

- **Clusters**: The dataset was partitioned into k clusters, each represented by a distinct color. The algorithm successfully identified natural groupings within the data.
- **PCA Components**: PCA was used to reduce the dimensionality of the data, allowing for a two-dimensional visualization of the clusters.

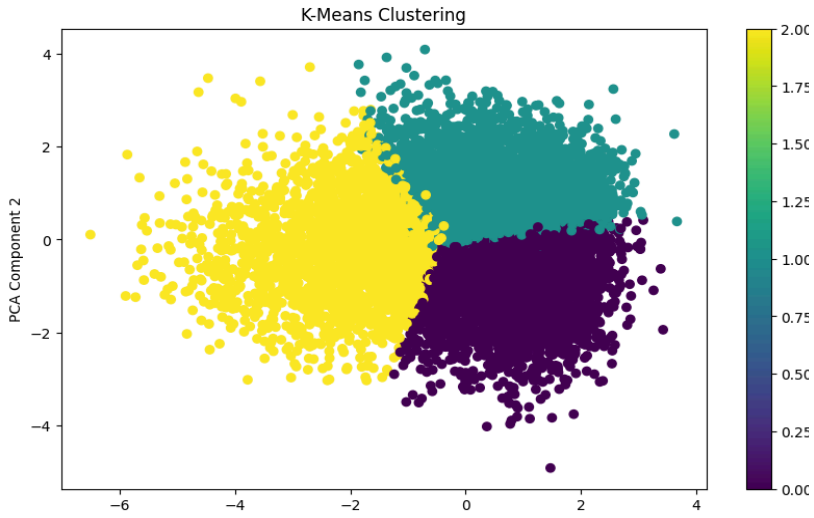

Figure 10:K-Means Clustering

5. Neural Network

Neural Networks are a class of models inspired by the human brain, capable of capturing complex patterns through layers of interconnected nodes (neurons). They are particularly powerful for tasks involving high-dimensional data and non-linear relationships.

Baseline model:

Initially, a simple 3-layer neural network was designed with a structure of 22-64-3 and a dropout rate of 0.01. Since the task involves classification, CrossEntropyLoss() was chosen as the loss function, and the model was optimized using AdamW() with a learning rate of 0.03. These choices are widely recognized as effective for a variety of problems, making them a solid foundation for our baseline model.

By taking a look at the loss graph in Figure 11, there is a strong indication that the model lacks enough understanding of the data mainly because of its simple structure.

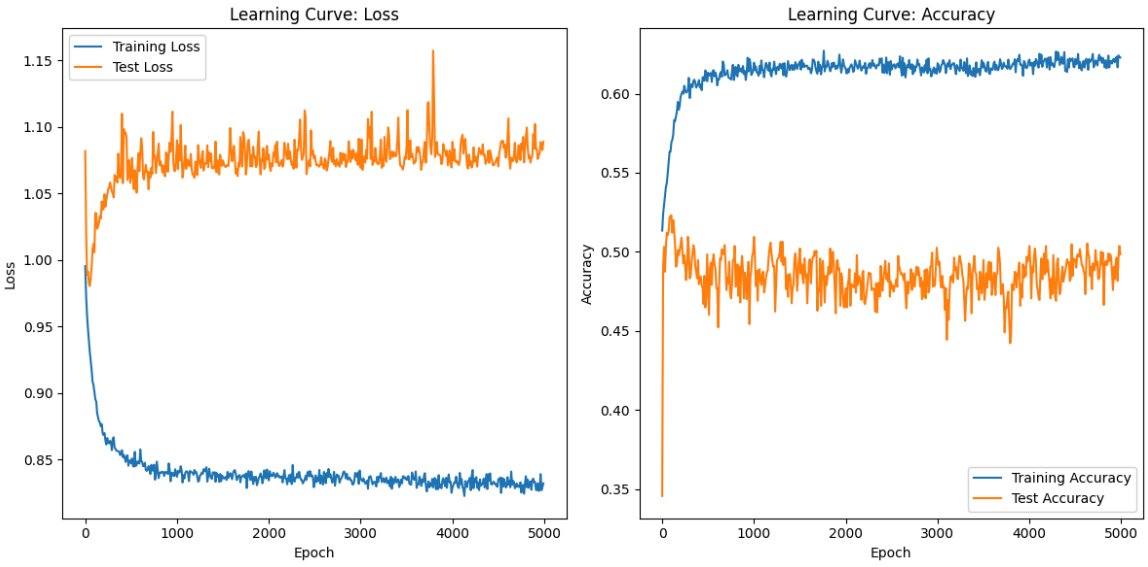

Figure 11: Baseline Model Curve

Fine-Tuned model version 1:

For the first try of our fine-tuned model, extra layers was added, the current structure of neural network is 22-128-64-32-3. In Figure 11, we can see there is a lot of fluctuation in the curve, thus we reduce the learning rate to 0.0001.

In Figure 12, it is interesting to see that at first the test loss curve decays very fast, after around 1000 epochs, the test loss stop decreasing and shows a trend of going up, in the meantime, training loss keep going lower, that's the strong sign of overfitting happening in the model.

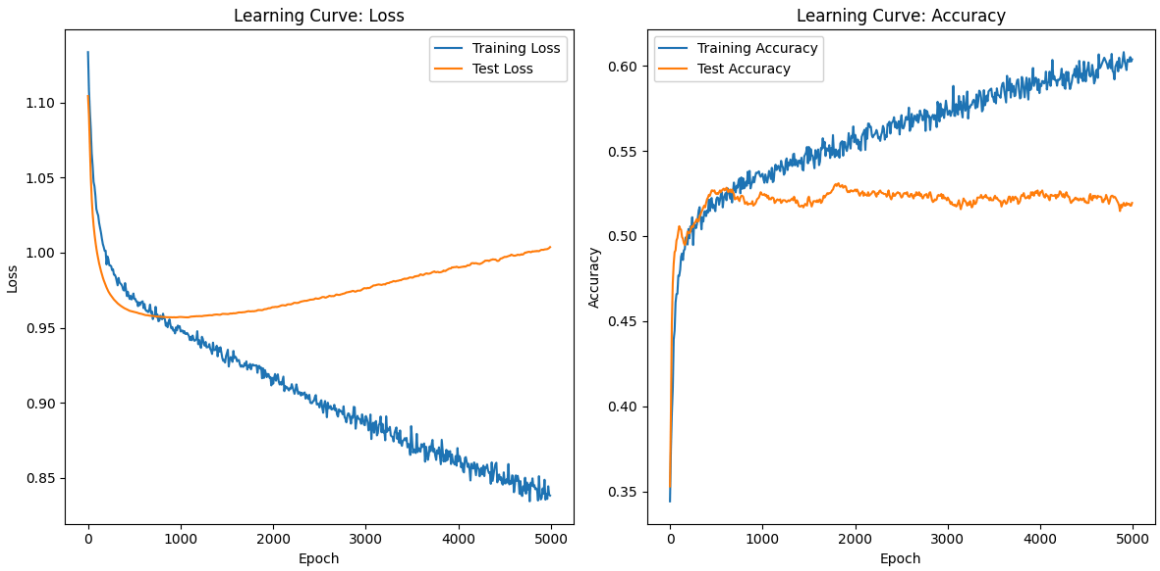

Figure 12: Fine-Tune version 1 Model Curve

Training: 100%| 5000/5000 [01:06<00:00, 75.09it/s, Training Loss=0.8463, Test Loss=1.0186, Training Accuracy=0.5958, Test Accuracy=0.5179]

Fine-Tuned model version 2:

By the lesson we had in version 1(Figure 12), we optimized our model based on that. The dropout rate is tuned higher to 0.45(the optimal value we have found), now in Figure 13, test loss finds a balance where it not getting higher over time.

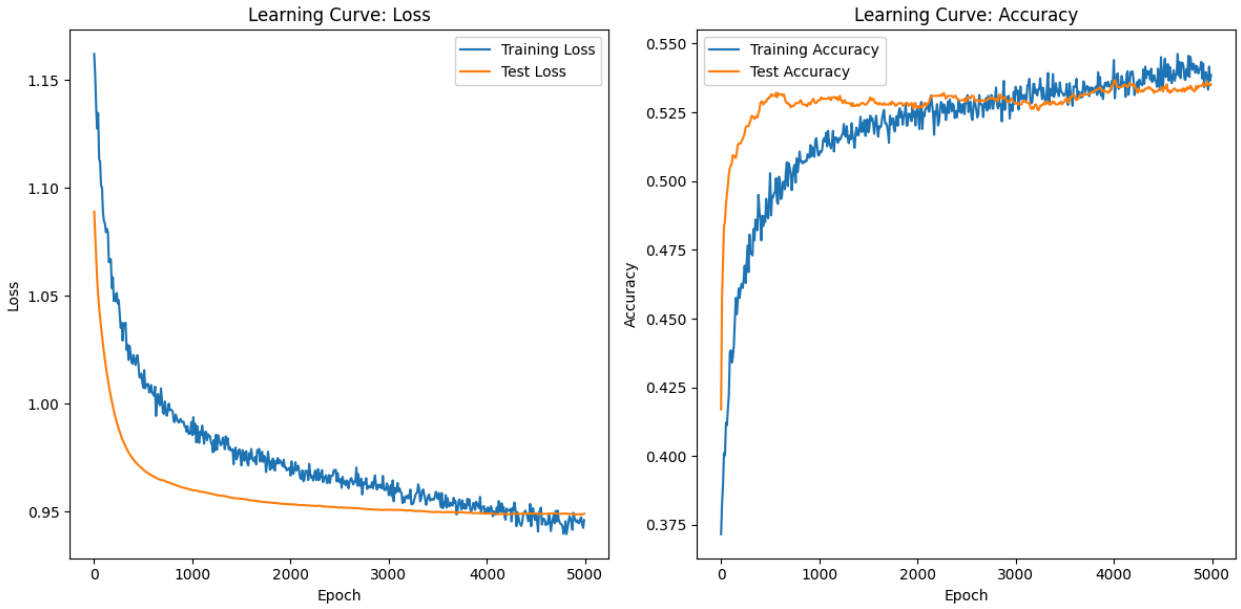

Figure 13: Fine-Tune version 2 Model Curve

Training: 100%|[redacted]| 5000/5000 [01:11<00:00, 69.45it/s, Training Loss=0.9461, Test Loss=0.9490, Training Accuracy=0.5385, Test Accuracy=0.5352]

With this final version of fine-tuned model, we get model accuracy of about 53.5%.

Problems Met and Conclusion

In conclusion, the CMPT353 project on song popularity prediction provided several valuable insights and highlighted key challenges.

1. Initially, our models were trained to predict a single numeric value representing the popularity of a song. This approach was challenging due to the inherent variability and complexity of musical tastes and trends, resulting in a low accuracy of around 20%. To address this, we restructured our approach by categorizing popularity scores into three levels: low, medium, and high. This classification allowed the models to focus on broader trends rather than exact values, significantly improving prediction accuracy. This shift highlighted the importance of appropriately framing the problem to align with the capabilities of machine learning models.
2. When predicting popularity based solely on the Spotify-provided popularity score, the accuracy was notably low. To enhance the accuracy, we incorporated cleaned data from Billboard, including metrics such as Weeks on Chart and Average Previous Week Position.

By combining these metrics with the Spotify popularity score and calculating a weighted average, we derived a more robust popularity level. This multi-faceted approach improved the reliability of our predictions, demonstrating the need for integrating multiple data sources to capture a more comprehensive picture of song popularity.

3. Our dataset contains three categories, however these categories are not uniformly distributed, with a disportional number of low level and high level categories compared to medium level categories. This imbalance causes the model to be biased in favor of the majority of categories, resulting in poorer performance for a few categories, as evidenced by the lower recall and accuracy scores for the medium level categories. To solve the problem of imbalance, we apply the Synthetic Minority Oversampling Technique (*SMOTE — Version 0.12.3*, n.d.). SMOTE creates a more balanced dataset by interpolating existing samples to generate synthetic samples for the minority categories. This approach prevents the model from being biased towards the majority category and improves its ability to generalize to unknown data. After applying SMOTE, the accuracy of the Support Vector Classifier (SVC) model is improved, with significant improvements in recall and accuracy scores for the medium level popularity categories.
4. The quality and accuracy of the extracted features are dependent on Spotify's algorithms, making our model's performance susceptible to any changes or inaccuracies in Spotify's feature extraction process. Additionally, Spotify's API has rate limits that restrict the number of requests that can be made within a given time period, necessitating careful management to avoid hitting these limits during data collection. Furthermore, not all tracks available in our dataset had corresponding entries in Spotify, leading to missing data that needed to be handled appropriately. Despite these challenges, by integrating these detailed audio features from Spotify, we were able to enhance our dataset and improve the accuracy of our models in predicting song popularity. This comprehensive feature set provided a solid foundation for our machine learning analysis.
5. Music popularity is highly dynamic and can change rapidly over time. Our dataset crosses many years, which means that the trends and factors that influenced popularity in earlier years may be quite different from recent years. This time variation may affect the performance of the model, as the characteristics of song popularity change in response to changes in cultural, technological, and social environments. For example, adding time-based

features or training the model over different time periods could provide deeper insights and improve prediction accuracy. However, the data in the audio extractor provided by Spotify does not include the year, which limits our ability to directly incorporate time dynamics into the model. Without year features, we cannot explicitly analyze how the importance of certain audio features changes over time. This limitation requires us to exclude year from consideration, which may miss valuable temporal context that could enhance our understanding of music popularity trends. We were unable to address this issue in the current study, but we hope to do so in future research.

6. One significant challenge encountered was the negative R-squared value obtained from the Random Forest model, indicating that the model performed worse than simply predicting the average popularity for all songs. This poor performance can arise from several factors, including overfitting, where the model captures noise rather than the underlying patterns in the data; inappropriate model selection, where Random Forest may not be the best fit for the dataset; insufficient data, which fails to capture the complexity of song popularity; and data quality issues, such as high variance and irrelevant features.
7. For the neural network, the most apparent issue we encountered was the tuning process. Regardless of the learning rate, dropout rate, or layer configurations we selected, the prediction score consistently plateaued at 53 to 54 percent. Despite experimenting with numerous settings and over 50,000 epochs of training, there was no improvement. This indicates that the problem is not related to overfitting or underfitting. Possible reasons include insufficient data for prediction, weak relationships between the chosen features and song popularity, or the inherent difficulty in predicting whether a song will go viral.

In conclusion, our project successfully identified key audio features influencing song popularity, yet several challenges, such as data variability, feature dependency on Spotify, and class imbalance, were encountered. To enhance the project, incorporating temporal features, integrating additional data sources, refining data cleaning and feature engineering, applying advanced resampling techniques, experimenting with different models and hyperparameter tuning, mitigating overfitting, improving API management, and integrating real-time data and user feedback are essential steps. These improvements can significantly enhance prediction accuracy and reliability, providing deeper insights into musical success factors.

Project Experience Summary | Computational Data Science

Lihao Qian: Discussion with our team for appropriate dataset selection and clean data approach. Connected and integrated the Spotify API and a music file data extraction library to obtain features for machine learning. Learned PyTorch and implemented both baseline and fine-tuned neural network models for the data, enhancing the accuracy of both the neural network and random forest models. Refined all code and integrated it into Predict.py for streamlined execution. Wrote the README and set up the running requirements.

Lingjie Li: Performed data cleaning and normalization on the Billboard dataset, combined it with Spotify audio features, and calculated the average previous position for enhanced predictive features. Trained and evaluated a Random Forest model to predict song popularity and implemented K-Means clustering for data segmentation. Documented all processes and results to ensure accuracy and reliability.

Qiqi Liu: Combine and clean the data, reorganized songs' popularity to make it more credible.. Complete correlation analysis, using heatmaps and regression plots to visualize the relationship between different audio features and song popularity. Complete data transformation and use machine learning methods, specifically k-Nearest Neighbors(k-NN) and Support Vector Classification(SVC) to predict popularity based on these audio features. Plot the learning curves.

Acknowledgments

We would like to sincerely thank Dor Azaria for his invaluable work, as his his blog post "Spotify Popularity Prediction - Machine learning" (Azaria, 2022) provided significant inspiration and laid the groundwork for this project.

In addition, we would like to thank Greg Baker for mentoring us throughout the semester. We would also like to thank our teaching assistants Mariam Bebawy, Ehsan Hoseinzade, and Fatemeh Movafagh, for their support and encouragement, which was instrumental in the successful completion of this project.

References

Azaria, D. (2022, March 5). Spotify Popularity Prediction - Machine learning. Dor Azaria's Tech Blog.
https://dorazaria.github.io/machinelearning/spotify-popularity-prediction/

SMOTE — Version 0.12.3. (n.d.).
https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html#r001eabbe5dd7-1

OneHotEncoder. (n.d.). Scikit-learn.
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

Billboard Hot weekly charts
https://data.world/kcmillersean/billboard-hot-100-1958-2017

Spotify-Data 1921-2020
https://www.kaggle.com/datasets/ektanegi/spotifydata-19212020

Spotify API
https://developer.spotify.com/documentation/web-api

Mutagen Library
https://mutagen.readthedocs.io/en/latest/

Pytorch
https://pytorch.org/tutorials/