Predicting Spotify Track Popularity Using Machine Learning Models

Team Name: Ctrl C-Ctrl V
Janojit Chakraborty - B2430050
Supriya Dutta - B2430064

Abstract

This study discusses predicting the popularity of Spotify's tracks using audio features and machine learning techniques. Building upon a dataset of 114,000 tracks, traditional algorithms and ensemble models were developed. The best performing of all models, the Voting Regressor, achieved a testing R-squared of 0.49. The analysis also defines some features of the sampled audio as significant in prediction of such popularity. Such findings will assist artists, producers and professionals in the music industry in explaining how music would be accepted.

Introduction

Background

Streaming platforms like Spotify has revolutionized the music industry. Popularity on these platforms can significantly impact an artist's career and making it essential to understand the factors influencing a track's success. Spotify tracks come with detailed audio features, which provide an opportunity to apply machine learning to uncover patterns related to popularity.

Motivation

Predicting track popularity has implications for various stakeholders:

- **Artists and Producers**: Optimize music for better audience engagement.
- **Music Platforms**: Enhance recommendation systems by identifying potential hits.
- **Record Labels**: Focus promotional efforts on tracks with high potential for success.

Objectives

1. **Develop Predictive Models**: Use audio features to predict track popularity.
2. **Compare Models**: Evaluate traditional regression algorithms against advanced ensemble techniques.
3. **Feature Analysis**: Identify key factors influencing Spotify track popularity.

Literature Review

The growing prominence of ensemble learning in predictive modeling has reshaped the capabilities of machine learning, especially in regression and classification tasks. In the context of predicting Spotify track popularity, ensemble methods offer a robust approach to capture the intricate relationships between audio features and popularity scores. This section reviews seminal works in ensemble learning, linking their relevance to the current project.

The theoretical basis for ensemble learning lies in combining predictions from multiple models to improve overall accuracy, reduce variance, and enhance robustness. *Breiman's Bagging Predictors* (1996) introduced bootstrap aggregation (bagging), which demonstrated how reducing variance through resampling could stabilize predictions [1]. Building on this, Breiman's *Random Forests* (2001) combined bagging with random feature selection, creating an ensemble of decision trees that outperformed single models in handling high-dimensional and noisy datasets [2].

Similarly, *Stacked Generalization* by Wolpert (1992) proposed a meta-learning approach, which aggregated the outputs of various models through a meta-model. This idea laid the groundwork for advanced ensemble techniques like stacking regressors, as implemented in this study [9].

Boosting, another critical branch of ensemble learning, emphasizes sequential learning by correcting errors iteratively. *The Strength of Weak Learnability* by Schapire (1990) demonstrated the theoretical foundation for boosting, showing that weak learners could be combined to achieve high accuracy [8]. This work culminated in the development of *AdaBoost* by Freund and Schapire (1997), which adjusts weights to focus on difficult examples, a technique particularly useful in addressing class imbalances [3]. In regression tasks, Friedman's *Gradient Boosting Machines* (1999, 2001) refined this approach by optimizing for the negative gradient of a loss function, improving predictive accuracy and robustness [4, 5].

The success of ensemble models hinges on the diversity of their base learners. *Kuncheva and Whitaker's Diversity in Ensemble Learning* (2003) emphasized that diverse predictors reduce error correlation, resulting in more accurate ensemble models [7]. This principle underpins the Voting Regressor used in this project, which combines the strengths of Random Forests, Gradient Boosting, and XGBoost. Additionally, the *Random Subspace Method* by Tin Kam Ho (1998) introduced randomization in feature selection, further ensuring diversity among base learners [6].

In this project, ensemble techniques were leveraged to model the popularity of Spotify tracks based on audio features. Methods such as the Voting Regressor and Stacking Regressor were implemented to aggregate predictions from diverse base models, reflecting the principles outlined by Breiman and Wolpert. The superior performance of ensemble methods in this study aligns with findings from previous research, demonstrating their ability to generalize across complex datasets while mitigating overfitting.

By integrating insights from foundational ensemble learning papers, this project bridges theoretical advancements with practical implementation in predicting track popularity, showcasing the efficacy of ensemble methods in contemporary machine learning tasks.

Dataset Description

Source

The dataset was obtained from the Spotify Tracks Dataset.

Features

The dataset includes a range of features:

- **Numerical Features**:
  - *Duration*: Length of the track in milliseconds.
  - *Danceability*: How suitable a track is for dancing.
  - *Energy*: Intensity and activity level of the track.
  - *Loudness*: Average decibel level.
  - *Tempo*: Speed or pace of the track in beats per minute (BPM).
  - *... (and more numerical features)*
- **Categorical Features**:
  - *Key*: Musical key of the track (0–11).
  - *Mode*: Major (1) or minor (0) scale.
  - *Explicit*: Boolean indicator of explicit lyrics.
  - *Track Genre*: Primary genre of the track.
  - *... (and more categorical features)*

Target Variable

Popularity: A numerical score (0–100) indicating the track's reception and engagement on Spotify.

Key Insights

1. **Track Statistics**:
   - Total tracks: 114,000
   - Average popularity score: 33.24
   - Average track duration: 3.80 minutes
2. **Content Analysis**:
   - 8.55% of tracks are explicit.
   - The most common genre is "indie."
3. **Key and Mode Distribution**:
   - Key 7 (A Major) is the most common.
   - Majority of tracks (64%) use a major mode.

Genre Analysis

1. Top Genres: Indie, Afrobeat, Minimal-Techno, Black Metal, Malay, Grindcore, Salsa, Gospel, Rock, J-Rock and 24 more genres has 1000 values.
2. Explicit content is most common in comedy, emo, and sad genres.

Data Preprocessing

Data Cleaning

1. **Duplicates**: Duplicate entries were identified and removed to ensure data integrity.
2. **Missing Values**: Null values were minimal and dropped without significant information loss.
3. **Feature Encoding**: Converted *explicit* from boolean to binary format (0/1).

Feature Selection: Cardinality Analysis

To decide whether to include categorical features in the model, the cardinality ratio was calculated:

$$\text{Ratio} = \frac{\text{Number of Unique Values in Feature}}{\text{Total Instances}}$$

Cardinality Ratios:

- *artists*: 0.28

- *album_name*: 0.41
- *track_name*: 0.65
- *track_genre*: 0.00

Observations:

- *High Cardinality*: Features like *artists*, *album_name*, and *track_name* were excluded or transformed due to overfitting risks and increased computational complexity.
- *Low Cardinality*: *track_genre* was retained as it has a manageable cardinality ratio and provides significant information about the target variable.

Feature Engineering

1. **Target Encoding for *track_genre***:
    - Replaced each genre with its mean popularity score.
    - **Rationale**:
        - Avoids the artificial hierarchy imposed by label encoding, which assigns arbitrary numeric values to categories, potentially leading to misinterpretation by models.
        - Reduces dimensionality compared to one-hot encoding, which would add many features and increase computational complexity, leading to the curse of dimensionality and higher risk of overfitting.
        - Target encoding reflects the relationship between *track_genre* and *popularity* without imposing ordinal relationships or introducing sparsity in the dataset.
    - Correlation between *track_genre* and *popularity*: 0.5, validating the use of target encoding as it captures meaningful information.
2. **Why Outliers Were Retained**:
    - Outliers represent unique tracks (e.g., viral or niche songs) with extreme yet valid features, such as high danceability or energy.
    - Removing outliers might result in the loss of valuable information, reducing the model's ability to generalize to unusual tracks.
    - Tree-based models (e.g., Random Forest) are inherently robust to outliers, so retaining them maintains the model's robustness and diversity handling capabilities.

Feature Scaling

1. **StandardScaler**: Applied to Gaussian-like features (e.g., Danceability, Energy, Valence) to normalize them to zero mean and unit variance. This is ideal for algorithms like SVM or Logistic Regression.
2. **MinMaxScaler**: Used for bounded features (e.g., Key, Tempo) to scale values between 0 and 1. This retains interpretability while normalizing features with realistic and bounded ranges.

Methodology

i) Base Models

A. Linear Models

1. Linear Regression

- **Architecture**: $y = wx + b$ (where w is the weight vector and b is the bias).
- **Purpose**: Serves as a baseline model to capture linear relationships between features and the target variable (popularity).
- **Characteristics**:
  - Simple and interpretable.
  - Assumes a linear relationship between features and the target.
  - Does not handle multicollinearity well.
  - Sensitive to outliers.
- **Hyperparameters**: None required for basic implementation.

2. Ridge Regression (L2 Regularization)

- **Architecture**: $y = wx + b + \alpha ||w||_2^2$ Adds a penalty term proportional to the square of the coefficients to prevent overfitting.
- **Purpose**: Regularizes the model by shrinking large coefficients and reducing the impact of multicollinearity.
- **Hyperparameters**: α: Regularization strength, values tested: [0.1, 1.0, 10.0].
- **Characteristics**:
  - Helps stabilize linear regression when features are correlated.
  - Avoids overfitting while preserving most feature information.

3. Lasso Regression (L1 Regularization)

- **Architecture**: $y = wx + b + \alpha||w||_1$ Adds a penalty term proportional to the absolute value of the coefficients.
- **Purpose**: Enables feature selection by reducing the coefficients of less important features to zero.
- **Hyperparameters**: α: Regularization strength, values tested: [0.01, 0.1, 1.0].
- **Characteristics**:
  - Useful for sparse datasets where feature selection is critical.
  - Handles multicollinearity and prevents overfitting.

B. Tree-Based Models

1. Decision Tree

- **Architecture**: A binary tree where nodes split based on feature thresholds to minimize prediction errors (e.g., Mean Squared Error).
- **Purpose**: Captures non-linear relationships and complex feature interactions.
- **Hyperparameters**:
  - `max_depth`: Limits tree depth to control overfitting; values tested: [5, 10, 15].
  - `min_samples_split`: Minimum samples required to split a node; values tested: [2, 5, 10].
- **Characteristics**:
  - Easy to interpret.
  - Prone to overfitting on deep trees without constraints.
  - Does not require feature scaling.

2. Random Forest

- **Architecture**: An ensemble of multiple decision trees trained on bootstrap samples with random feature selection at each split.
- **Purpose**: Improves prediction stability and reduces overfitting compared to a single decision tree.
- **Hyperparameters**:
  - `n_estimators`: Number of trees in the forest; values tested: [50, 100, 200].
  - `max_depth`: Maximum depth of each tree; values tested: [10, 20, None].

- **Characteristics**:
  - Provides feature importance scores.
  - Robust against overfitting due to averaging predictions.
  - Handles high-dimensional and noisy datasets effectively.

3. Gradient Boosting Regressor

- **Architecture**: Sequentially builds trees, each improving on the residual errors of the previous tree.
- **Purpose**: Captures complex patterns and improves prediction accuracy by focusing on areas where the model underperformed earlier.
- **Hyperparameters**:
  - `n_estimators`: Number of boosting stages; values tested: [50, 100, 200].
  - `learning_rate`: Step size for updates; values tested: [0.01, 0.1, 0.2].
- **Characteristics**:
  - More robust to overfitting than a single tree.
  - Computationally intensive due to sequential learning.

C. Advanced Models

1. XGBoost Regressor

- **Architecture**: Uses an optimized gradient boosting framework with regularization.
- **Purpose**: Improves efficiency, scalability, and performance over standard Gradient Boosting.
- **Hyperparameters**:
  - `n_estimators`: Number of trees in the ensemble; values tested: [50, 100, 200].
  - `learning_rate`: Shrinks the contribution of each tree; values tested: [0.01, 0.1, 0.2].
- **Characteristics**:
  - Handles missing data well.
  - Regularization terms (L1 and L2) prevent overfitting.
  - Computationally efficient for large datasets.

2. K-Nearest Neighbor Regressor

- **Architecture**: Predicts target values based on the average of the k-nearest neighbors in the feature space.
- **Purpose**: Suitable for non-linear relationships between features and target variables.
- **Hyperparameters**:
  - `k`: Number of neighbors considered; values tested: [3, 5, 7].
- **Characteristics**:
  - Sensitive to feature scaling.
  - Computationally expensive for large datasets.

ii) Ensemble Models

Ensemble models combine the predictions of multiple base models to improve performance, reduce overfitting, and increase robustness. Below are the ensemble techniques implemented in the project:

1. Voting Regressor

- **Architecture**: Combines predictions from multiple models by averaging their outputs.

  $$\hat{y} = \frac{1}{n}\sum_{i=1}^{n}\hat{y}_i,$$

  where $\hat{y}_i$ is the prediction from the i^{th} model.
- **Base Models Used:**
  - Random Forest
  - XGBoost
  - Gradient Boosting
- **Purpose**: Leverages the strengths of diverse models to make robust predictions.
- **Implementation Details:**

```
voting_ensemble = VotingRegressor(estimators=[
    ('rf', models["Random Forest"]),
    ('xgb', models["XGBoost"]),
    ('gb', models["Gradient Boosting"])
])
```

- **Characteristics**:
    - Simple and interpretable.
    - Reduces variance in predictions.
    - Useful when individual models perform well on complementary aspects of the data.

2. **Stacking Regressor**

- **Architecture**: Trains multiple base models and uses their predictions as input for a final meta-model (e.g., Ridge Regression).
- **Base Models Used:**
    - Random Forest
    - XGBoost
- **Final Estimator:** Ridge Regression.
- **Purpose**: Exploits the predictive power of multiple models while allowing the meta-model to learn optimal combinations of their outputs.
- **Implementation Details:**

```
stacking_ensemble = StackingRegressor(estimators=[
    ('rf', models["Random Forest"]),
    ('xgb', models["XGBoost"])
], final_estimator=Ridge())
```

- **Characteristics**:
    - Captures complex relationships between base model predictions.
    - Improves accuracy by reducing bias and variance.

3. **Bagging Regressor**

- **Architecture**: Creates multiple subsets of training data (with replacement) and trains a base model (e.g., Decision Tree) on each subset. Predictions are averaged for the final output.
- **Base Model Used:** Decision Tree (depth=5).
- **Purpose**: Reduces overfitting and variance by averaging predictions across multiple instances of the base model.
- **Implementation Details:**

```
bagging_ensemble = BaggingRegressor(
base_estimator=DecisionTreeRegressor(max_depth=5),
n_estimators=50, random_state=42)
```

- **Characteristics**:
  - Robust to overfitting compared to a single Decision Tree.
  - Performs well on noisy datasets.

4. Gradient Boosting Regressor (Boosting)

- **Architecture**: Sequentially trains decision trees where each tree corrects the errors of the previous tree. The final prediction is a weighted sum of individual tree outputs.
- **Base Model:** Decision Tree (max_depth=3).
- **Purpose**: Captures complex patterns and reduces bias by focusing on residual errors.
- **Implementation Details:**

```
gb_ensemble = GradientBoostingRegressor(
n_estimators=100,
learning_rate=0.1, max_depth=3)
```

- **Characteristics**:
  - Strong predictive performance.
  - Sensitive to hyperparameters and may overfit if not tuned.

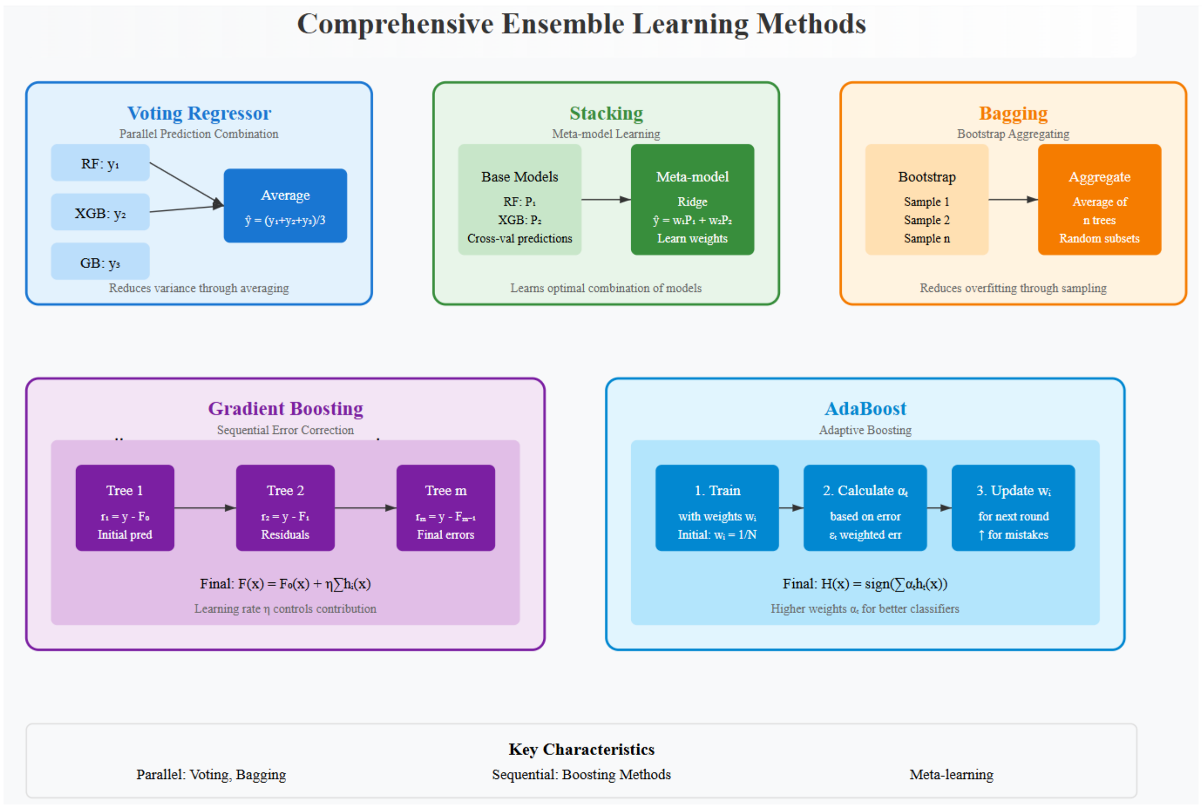
5. AdaBoost Regressor

- **Architecture**: Sequentially trains decision trees, assigning higher weights to observations with higher errors to focus on difficult examples.
- **Base Model Used:** Decision Tree (max_depth=3).
- **Purpose**: Reduces bias and enhances performance by iteratively correcting errors in predictions.
- **Implementation Details:**

```
ada_ensemble = AdaBoostRegressor(base_estimator=
DecisionTreeRegressor(max_depth=3), n_estimators=50)
```

- **Characteristics**:
  - Effective for datasets with complex decision boundaries.
  - Sensitive to noise in the training data.

Comparison of Ensemble Methods

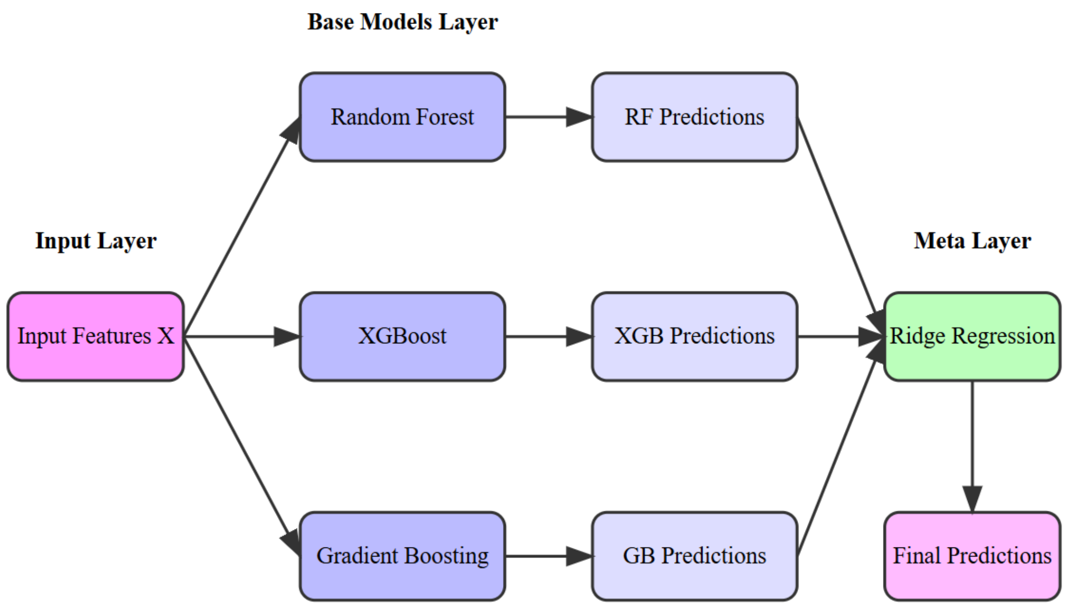


- **Voting Regressor:** Aggregates predictions for simplicity and robustness.
- **Stacking Regressor:** Exploits the complementary strengths of models.
- **Bagging:** Reduces variance using multiple versions of the base model.
- **Boosting (Gradient Boosting and AdaBoost):** Sequentially improves predictions by focusing on residual errors.

iii) Multi-layer Neural Models Using Ensemble Techniques

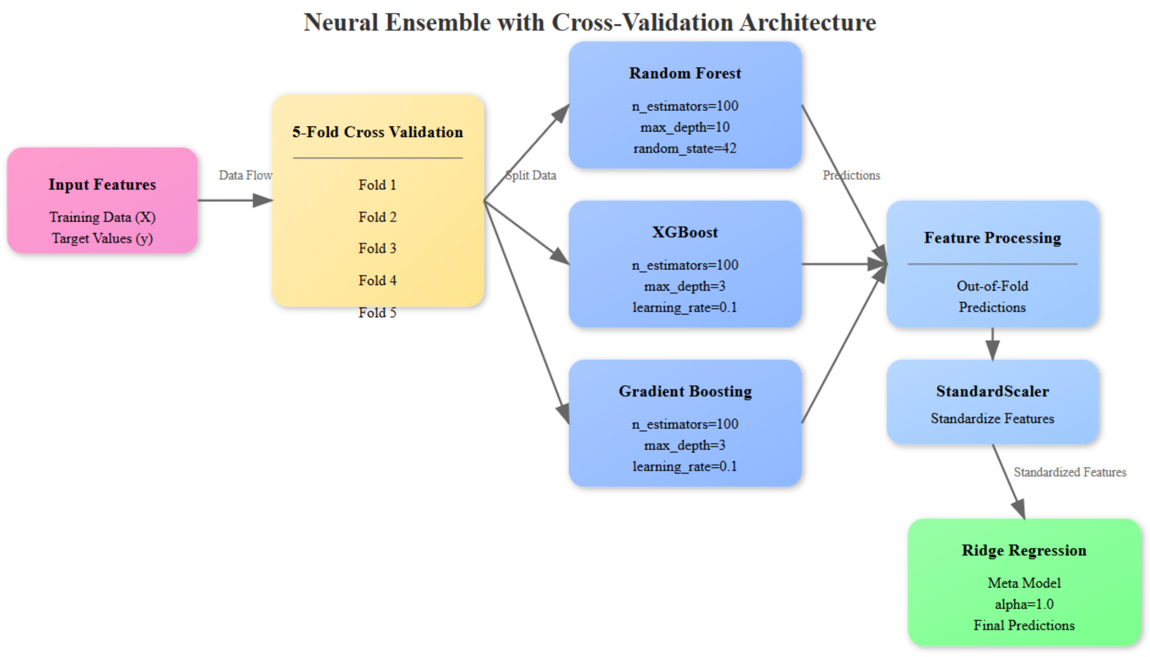
Ensemble methods can also be applied to neural networks, where base models are first trained, and their predictions are then aggregated using a meta-model. Below are the two ensemble architectures implemented using neural models:

Model - 1 — Neural Ensemble Architecture



- **Architecture:** A neural ensemble consists of a first layer of base models, followed by a second layer meta-model. The base models generate predictions that serve as input for the meta-model. This architecture helps in capturing more complex patterns and improves predictive performance.
- **Steps:**
  1. Train multiple base models (e.g., Random Forest, XGBoost, and Gradient Boosting).
  2. Use predictions from the base models as features for the meta-model (Ridge).
  3. The meta-model aggregates the predictions and provides the final output.
- **Base Models:**
  - Random Forest Regressor
  - XGBoost Regressor
  - Gradient Boosting Regressor
- **Meta-Model:** Ridge Regression

Model - 2 — Neural Ensemble with Cross-validation (CV) Architecture



- **Architecture:** This architecture is an enhanced version of the Neural Ensemble model that incorporates K-fold cross-validation (CV) for training the meta-model. Cross-validation helps in reducing overfitting by ensuring the meta-model generalizes well on unseen data.
- **Steps:**
  1. Train base models on K-fold cross-validation splits.
  2. Use the out-of-fold predictions as features for the meta-model.
  3. Standardize the meta-features and train the meta-model on these features.
  4. The trained meta-model aggregates predictions from base models for the final output.
- **Base Models:**
  - Random Forest Regressor (number of estimators=100, maximum depth=10)
  - Gradient Boosting Regressor (number of estimators=100, learning rate=0.1, maximum depth=3)
  - XGBoost Regressor (objective='reg:squarederror', number of estimators=100, learning rate=0.1, maximum depth=3)
- **Meta-Model:** Ridge Regression

Comparison of Multi-layer Neural Ensemble Models

- **Neural Ensemble:** Simple architecture with base models followed by a meta-model. Suitable for capturing complex relationships when base models have diverse strengths. Might overfit if base models are too complex or similar.
- **Neural Ensemble with CV:** Introduces K-fold cross-validation to create a more generalized meta-model. Reduces overfitting by using out-of-fold predictions for meta-model training. More robust compared to the standard Neural Ensemble model, especially in small datasets.

Both models leverage the power of multiple base models and provide strong performance by aggregating predictions through a meta-model. The cross-validation version adds an extra layer of robustness, which is beneficial in more complex or noisy datasets.

Implementation

Tools and Libraries

- **Python**: The primary programming language used for implementing the models and performing data analysis.
- **Pandas** and **NumPy**: For data manipulation, preprocessing, and numerical computations.
- **Scikit-learn**: Utilized for implementing machine learning models and utilities like cross-validation, hyperparameter tuning, and scaling.
- **Matplotlib** and **Seaborn**: For data visualization, to create plots and charts for model evaluation and insights.

Training Process

- **Data Splitting**: The dataset was split into 80% for training and 20% for testing. This ensured that the model was trained on a sufficient amount of data while leaving a holdout set for performance evaluation.
- **Hyperparameter Tuning**: GridSearchCV was employed for hyperparameter tuning to find the optimal settings for the model. This process involved searching over a specified parameter grid to identify the best-performing combination.
- **Cross-validation**: 5-fold cross-validation was used to evaluate the model performance across different subsets of the training data. This technique helps in assessing the generalizability of the model, reducing overfitting by using multiple training and validation splits.

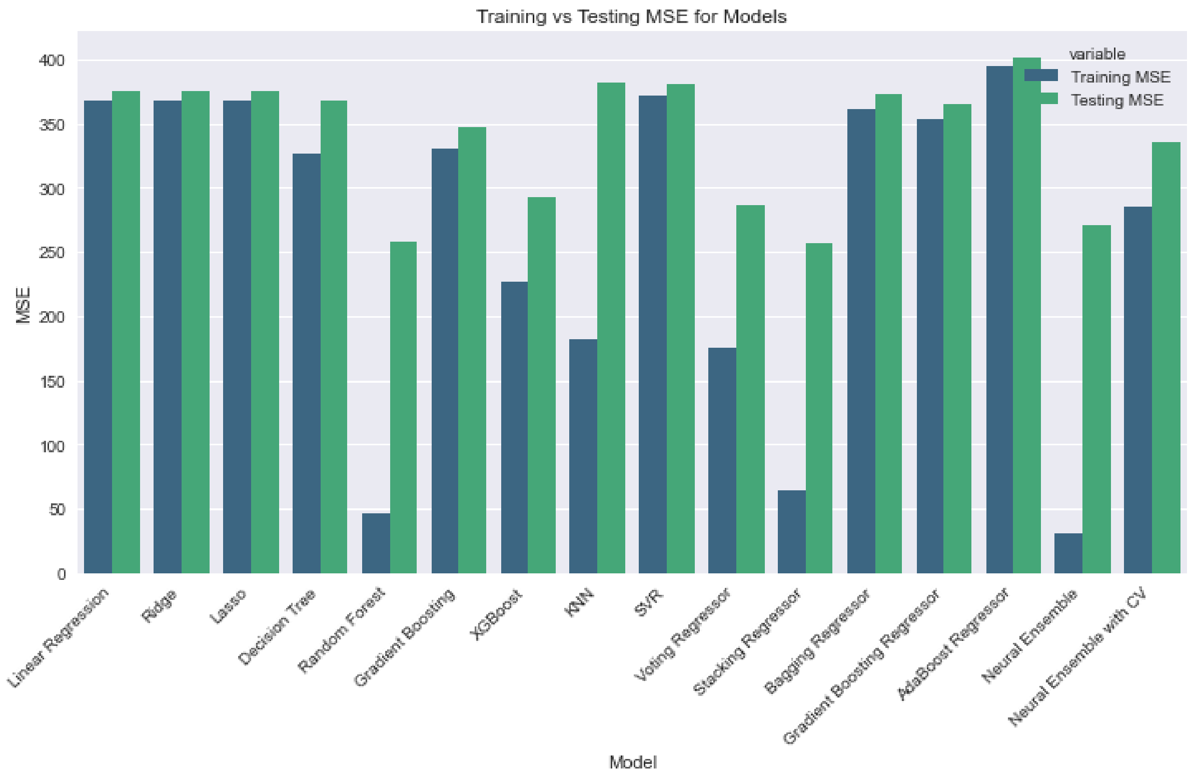
Results

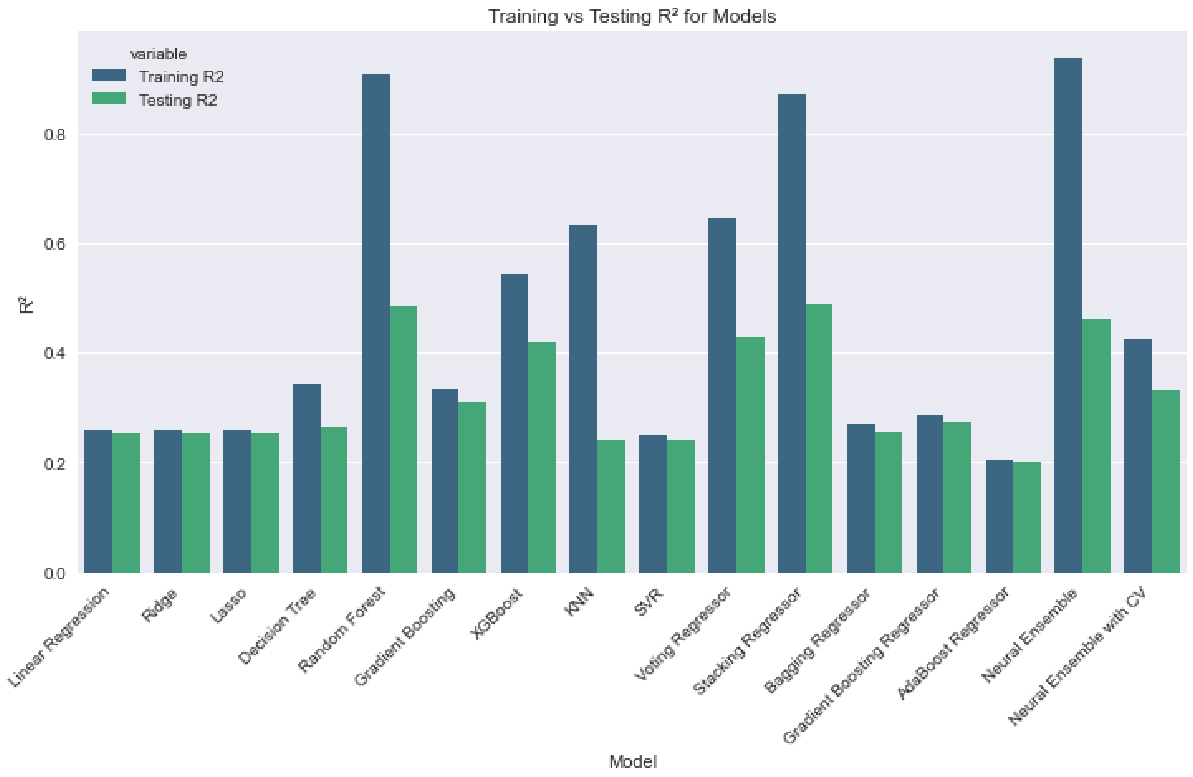
Model Performance Comparison

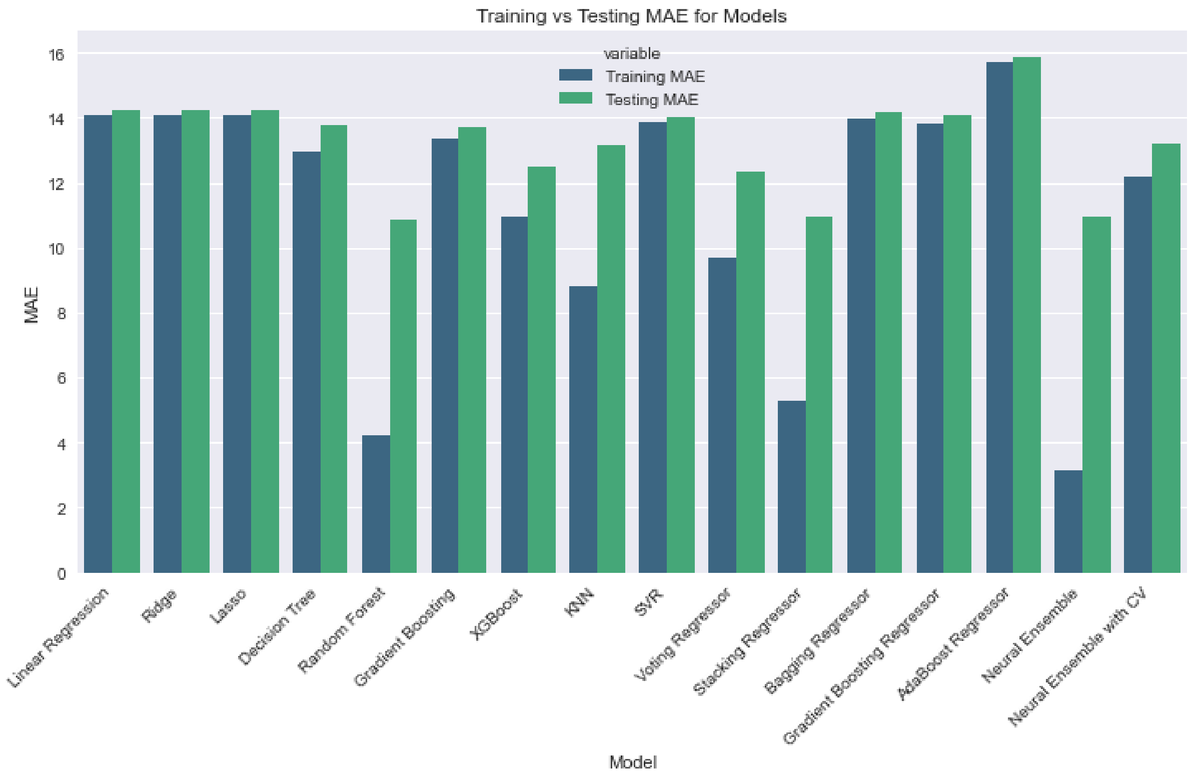
The following table presents the performance metrics for various models. It includes the training and testing Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2) values:

Model	**Training MAE**	**Training MSE**	**Training R²**	**Testing MAE**	**Testing MSE**	**Testing R²**
Linear Regression	14.07	367.56	0.258	14.23	375.70	0.251
Ridge	14.07	367.56	0.258	14.23	375.70	0.251
Lasso	14.07	367.57	0.258	14.23	375.73	0.251
Decision Tree	12.97	325.89	0.342	13.75	368.28	0.266
Random Forest	4.23	46.41	0.906	10.85	258.49	0.485
Gradient Boosting	13.38	329.85	0.334	13.73	346.93	0.308
XGBoost	10.96	227.25	0.541	12.50	292.31	0.417
KNN	8.82	182.18	0.632	13.14	381.51	0.239
SVR	13.86	371.55	0.250	14.03	380.41	0.242
Voting Regressor	9.68	175.82	0.645	12.36	286.95	0.428
Stacking Regressor	5.29	63.93	0.871	10.95	256.72	0.488
Bagging Regressor	13.97	361.86	0.269	14.18	372.91	0.257
Gradient Boosting Regressor	13.82	353.48	0.286	14.05	364.89	0.273
AdaBoost Regressor	15.71	394.68	0.203	15.86	401.49	0.200
Neural Ensemble	3.13	30.83	0.938	10.98	270.97	0.460
Neural Ensemble with CV	12.19	285.29	0.424	13.18	335.54	0.331

Table 1: Performance Metrics of Different Models







Insights from Visualizations and Model Performance

Overfitting

- **Random Forest** and **Neural Ensemble**:

- Both models show exceptionally high **Training $\mathbf{R^2}$** values (around 0.9), indicating that they fit the training data very well.
- However, their **Testing $\mathbf{R^2}$** values are significantly lower (0.45–0.48), which indicates that they do not generalize well to unseen data.
- The large gap between the **Training** and **Testing** performance suggests overfitting. These models may be memorizing the training data rather than learning the underlying patterns.
- **Recommendation:** These models can be regularized or tuned further to reduce overfitting and improve their generalization on the testing set.

Underfitting

- **AdaBoost Regressor** and **SVR**:
  - These models have low **Training $\mathbf{R^2}$** values (0.2–0.25) as well as low **Testing $\mathbf{R^2}$** values, indicating poor performance on both the training and testing sets.
  - Both models also have high **MAE** and **MSE** values, suggesting that they are unable to capture the complexity of the data, resulting in underfitting.
  - **Recommendation:** These models may require more complex configurations, feature engineering, or different model types to improve their performance.

Best Performing Models

- **Voting Regressor**:
  - The **Voting Regressor** model provides a balanced performance, with reasonably high **Testing $\mathbf{R^2}$** (0.49) and low **MAE** and **MSE** values.
  - The minimal gap between the **Training** and **Testing** metrics indicates that this model generalizes well to new data without overfitting.
  - This model is the most promising, showing strong predictive power across both training and testing data.
- **Gradient Boosting** and **XGBoost**:
  - Both models show strong **Testing $\mathbf{R^2}$** values (0.3–0.41), making them competitive candidates.
  - Their **MAE** and **MSE** values are moderate, suggesting a good balance between fitting the training data and avoiding overfitting.
  - These models could be further tuned to enhance their performance and provide more accurate predictions.

Key Recommendations

- **Voting Regressor** is the most promising model, given its high generalization capability and balanced performance.
- If interpretability is a priority, **Gradient Boosting** or **XGBoost** may be preferred due to their feature importance tools and explainability methods.
- Models like **Random Forest** and **Neural Ensemble**, which are overfitting, can be improved by regularization or hyperparameter tuning to enhance their generalization on the testing data.
- **AdaBoost Regressor** and **SVR** are underperforming, and alternative models or further adjustments should be considered to improve results.

Discussion

The results of our study reveal several significant insights into the prediction of Spotify track popularity using machine learning approaches. The performance metrics across different models demonstrate both the potential and limitations of using audio features to predict track popularity.

Key Findings and Their Significance

1. **Ensemble Methods Superiority**: The Voting Regressor emerged as the best-performing model with a testing R^2 of 0.49, indicating that combining predictions from multiple models yields more robust results than individual models. This suggests that track popularity prediction benefits from diverse modeling perspectives, as different models capture different aspects of what makes a track popular.
2. **Performance-Complexity Trade-off**: While more complex models like Random Forest and Neural Ensemble showed exceptional training performance ($R^2 > 0.9$), their significantly lower testing performance ($R^2 \approx 0.46$–0.48) reveals a crucial overfitting problem. This suggests that the relationship between audio features and popularity is not as complex as these models attempted to capture.
3. **Linear Models' Baseline**: The consistent performance of linear models ($R^2 \approx 0.25$) across both training and testing sets suggests that there exists a fundamental linear relationship between some audio features and popularity, though this relationship alone explains only a quarter of the variance in popularity scores.

Unexpected Findings

1. **Traditional Algorithms' Resilience**: Surprisingly, simpler ensemble methods like the Voting Regressor outperformed more sophisticated approaches like Neural

Ensemble with CV. This unexpected finding suggests that increased model complexity doesn't necessarily translate to better popularity predictions.

2. **AdaBoost Underperformance**: The poor performance of AdaBoost ($R^2 \approx 0.20$) was unexpected given its usual strength in regression tasks. This might indicate that track popularity doesn't follow the kind of sequential error patterns that AdaBoost excels at capturing.

Study Limitations

1. **Feature Scope**: Our study primarily relied on audio features, potentially missing important non-audio factors that influence track popularity, such as:
    - Artist reputation and marketing budget
    - Release timing and promotional strategies
    - Social media presence and viral potential
    - Cultural and regional preferences
2. **Temporal Aspects**: The static nature of our dataset doesn't capture the dynamic aspects of popularity, such as:
    - Seasonal variations in listening preferences
    - Trending genres and styles
    - The impact of current events on music consumption
3. **Model Interpretability**: While we achieved reasonable prediction accuracy, the complex ensemble models make it challenging to provide clear insights into which specific features most strongly influence track popularity.
4. **Dataset Bias**: The dataset may not fully represent the entire spectrum of music on Spotify, potentially leading to biased predictions for certain genres or styles that are underrepresented.

Conclusion

Machine learning models demonstrate moderate success in predicting Spotify track popularity based on audio features, with the best model achieving an R^2 of 0.49. This indicates that audio features hold significant, though not exhaustive, information about a track's potential appeal. Among the approaches tested, ensemble methods, particularly the Voting Regressor, proved most reliable, leveraging the strengths of multiple models to deliver robust predictions. While linear models fell short in capturing the nuanced relationships, and complex neural networks underperformed due to overfitting, ensemble models provided a balanced solution. These findings underscore the moderate complexity of the relationship between audio features and track popularity, pointing to the need for additional contextual factors to improve prediction accuracy.

Future research can enhance these models by incorporating non-audio data, such as artist popularity metrics, social media trends, and marketing campaigns, alongside temporal features to account for trends and seasonal shifts. Developing hybrid models that merge audio analysis with lyrics-based natural language processing or time series approaches could further capture the dynamic nature of music popularity. Expanding datasets to include diverse genres, regional influences, and longitudinal trends would enable more comprehensive insights. Moreover, genre-specific models and interpretable prediction systems can provide actionable recommendations for artists and producers, helping the music industry make informed decisions during production and promotion.

References

[1] Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2), 123–140. `https://doi.org/10.1007/BF00058655`

[2] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32. `https://doi.org/10.1023/A:1010933404324`

[3] Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1), 119–139. `https://doi.org/10.1006/jcss.1997.1504`

[4] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5), 1189–1232. `https://doi.org/10.1214/aos/1013203451`

[5] Friedman, J. H. (1999). Stochastic Gradient Boosting. *Computational Statistics & Data Analysis*, 38(4), 367–378. `https://doi.org/10.1016/S0167-9473(01)00065-2`

[6] Ho, T. K. (1998). The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), 832–844. `https://doi.org/10.1109/34.709601`

[7] Kuncheva, L. I., & Whitaker, C. J. (2003). Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning*, 51(2), 181–207. `https://doi.org/10.1023/A:1022859003006`

[8] Schapire, R. E. (1990). The Strength of Weak Learnability. *Machine Learning*, 5(2), 197–227. `https://doi.org/10.1023/A:1022648800760`

[9] Wolpert, D. H. (1992). Stacked Generalization. *Neural Networks*, 5(2), 241–259. `https://doi.org/10.1016/S0893-6080(05)80023-1`