Classifying 'danceable' Spotify genres with Random Forest (RF) models*

Christina Nguyen

December 14, 2024

1 Introduction

Music classification is a useful way to organize large digital music libraries. Streaming platforms like Spotify use algorithms to classify tracks into specific genres based on track metadata ("sound features") such as danceability, tempo, and energy. In this analysis, I focus on using Random Forest (RF) models to classify Spotify tracks into ten highly "danceable" genres. This highlights the process from data preparation to the evaluation of multiple Random Forest models, ultimately identifying the best-performing model for this particular dataset, using confusion matrices.

This analysis's objectives are:

1. Explore the structure of the Spotify dataset and identify the relevant metadata fields.
2. Preprocess the data to ensure it is suitable for classification.
3. Build Random Forest models and assess their performance.@rf-models
4. Investigate the mis-classification trends to understand which genres are most frequently confused with one another; consider some contextual reasons why these genres are frequently confused with one another.
5. Compare model performance across different tree sizes (100, 500, and 1000 trees) to determine the optimal configuration.

2 Data preprocessing

The data was downloaded from Kaggle's "Spotify Tracks Dataset" (**Kaggle?**). The downloaded dataset contained over 100,000 tracks from 125 different genres. Each track has audio

*Code and data are available at: https://github.com/ChristinaDNguyen/spotify-metadata-analysis.

features listed.

2.1 Exploration

Some preliminary explorations examined the audio features assigned to tracks, to understand how the entire tracklist was represented across these features, on average.

2.1.1 The summary statistics

Seeing the multiple audio track features as a summary helps visualize what kind of data we are working with:

```
library(readr)
library(knitr)

#Set the path to the file to display summary statistics of audio features:
file_pathificate <- "output1_summary_statistics.csv"
#Put the path into an object
summary_statis <- read_csv(file_pathificate)
```

```
Rows: 1 Columns: 45
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (45): danceability_mean, energy_mean, loudness_mean, speechiness_mean, a...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#Display that object in a horizontal chart
kable(summary_statis)
```

[illegible]										
[illegible] -8.25896 [illegible] -7.004 [illegible] 05 [illegible]	-49.531	0	0	0	0	0	0	0.9[illegible]	[illegible]	0.9[illegible]

```
#Make that in a vertical chart to make it easy to read
transposed_statis <- as.data.frame(t(summary_statis))
kable(transposed_statis)
```

	V1
danceability_mean	0.5668001
energy_mean	0.6413828
loudness_mean	-8.2589604
speechiness_mean	0.0846521
acousticness_mean	0.3149101
instrumentalness_mean	0.1560496
liveness_mean	0.2135528
valence_mean	0.4740682
tempo_mean	122.1478373
danceability_median	0.5800000
energy_median	0.6850000
loudness_median	-7.0040000
speechiness_median	0.0489000
acousticness_median	0.1690000
instrumentalness_median	0.0000416
liveness_median	0.1320000
valence_median	0.4640000
tempo_median	122.0170000
danceability_sd	0.1735422
energy_sd	0.2515291
loudness_sd	5.0293366
speechiness_sd	0.1057324
acousticness_sd	0.3325227
instrumentalness_sd	0.3095548
liveness_sd	0.1903777
valence_sd	0.2592611
tempo_sd	29.9781969
danceability_min	0.0000000
energy_min	0.0000000
loudness_min	-49.5310000
speechiness_min	0.0000000
acousticness_min	0.0000000
instrumentalness_min	0.0000000
liveness_min	0.0000000
valence_min	0.0000000
tempo_min	0.0000000
danceability_max	0.9850000
energy_max	1.0000000
loudness_max	4.5320000
speechiness_max	0.9650000
acousticness_max	0.9960000

	V1
instrumentalness_max	1.0000000
liveness_max	1.0000000
valence_max	0.9950000
tempo_max	243.3720000

```
#Correct the column name
colnames(transposed_statis) <- c("Value")

#Print that vertical chart again
kable(transposed_statis)
```

	Value
danceability_mean	0.5668001
energy_mean	0.6413828
loudness_mean	-8.2589604
speechiness_mean	0.0846521
acousticness_mean	0.3149101
instrumentalness_mean	0.1560496
liveness_mean	0.2135528
valence_mean	0.4740682
tempo_mean	122.1478373
danceability_median	0.5800000
energy_median	0.6850000
loudness_median	-7.0040000
speechiness_median	0.0489000
acousticness_median	0.1690000
instrumentalness_median	0.0000416
liveness_median	0.1320000
valence_median	0.4640000
tempo_median	122.0170000
danceability_sd	0.1735422
energy_sd	0.2515291
loudness_sd	5.0293366
speechiness_sd	0.1057324
acousticness_sd	0.3325227
instrumentalness_sd	0.3095548
liveness_sd	0.1903777
valence_sd	0.2592611
tempo_sd	29.9781969

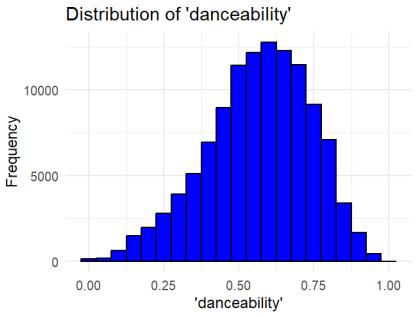
	Value
danceability_min	0.0000000
energy_min	0.0000000
loudness_min	-49.5310000
speechiness_min	0.0000000
acousticness_min	0.0000000
instrumentalness_min	0.0000000
liveness_min	0.0000000
valence_min	0.0000000
tempo_min	0.0000000
danceability_max	0.9850000
energy_max	1.0000000
loudness_max	4.5320000
speechiness_max	0.9650000
acousticness_max	0.9960000
instrumentalness_max	1.0000000
liveness_max	1.0000000
valence_max	0.9950000
tempo_max	243.3720000

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. The mean danceability score of 0.5668 suggests that, on average, the tracks in our dataset are moderately danceable. Further seperation into categories of "low," "moderate," and "high" danceability will be performed to see the spread of "danceability." The median value of 0.5800 indicates that half of the tracks have a danceability score above this value, meaning they are more likely to be suitable for dancing. The standard deviation of 0.1735 shows moderate variability in danceability scores across tracks, indicating that while some tracks are highly danceable, others may not be as suitable for dancing. The minimum value of 0.0000 and the maximum value of 0.9850 highlight the range of danceability in the dataset, from tracks that are not danceable at all to those that are almost perfectly suited for dancing. Interestingly, however, danceability, while quantified by Spotify's algorithm, is inherently subjective. The algorithm considers factors such as tempo, rhythm stability, beat strength, and overall regularity to assign a danceability score. These measurements are detailed in Spotify's API documentation (**SpotifyAPIDocumentation?**). However, what makes a track danceable can vary greatly from person to person. For instance, some listeners might find a track with a slower tempo and smooth rhythm highly danceable, while others might prefer fast-paced tracks with strong beats.

2.1.2 A focus on danceability's spread

Indeed the 'spread' of 'danceability,' as seen with the standard deviation of 0.1735 and mean, can be seen as a frequency cluster around danceability = 0.6:

```
library(knitr)
knitr::include_graphics("output2_distribution_of_danceability.png")
```



Other audio track features included in the summary analysis included energy, loudness, speechiness, acousticness, instrumentalness, temp, liveness, and valence. Energy measures the intensity and activity of a track. A mean energy score of 0.6414 suggests that our tracks are generally energetic. The median value of 0.6850 suggests that more than half of the tracks have high energy levels. But the standard deviation of 0.2515 indicates considerable variability in energy levels among the tracks. (**SpotifyAPIDocumentation?**)

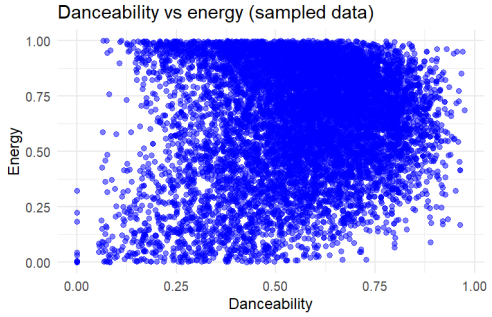
Loudness is the overall volume of a track in decibels (dB). The mean loudness of -8.2590 dB suggests that the tracks are, on average, relatively quiet. The median value of -7.0040 dB indicates that half of the tracks are louder than this value. (**SpotifyAPIDocumentation?**)

Speechiness detects the presence of spoken words in a track. The mean speechiness score of 0.0847 indicates that most tracks have low speech content. Acousticness measures the confidence that a track is acoustic. The mean acousticness score of 0.3149 suggests that the tracks are moderately acoustic. Instrumentalness predicts whether a track contains no vocals. The mean instrumentalness score of 0.1560 suggests that most tracks have some vocal content. Liveness detects the presence of an audience in the recording. The mean liveness score of 0.2136 suggests that the tracks are generally not live recordings. Valence describes the musical positiveness conveyed by a track. The mean valence score of 0.4741 suggests that the tracks have a balanced emotional tone. Tempo is the speed or pace of a track in beats per minute (BPM). The mean tempo of 122.1478 BPM suggests that the tracks have a moderate tempo on average. (**SpotifyAPIDocumentation?**)

2.1.3 Danceability vs. energy

The comparison of danceability vs. energy in data analysis, especially in fields like music analysis or audio feature analysis, is useful because these two concepts can capture distinct aspects of a song's "feel" or "mood." In future analysis and use, when building a music recommendation engine (like Spotify already does), understanding the relationship between energy and danceability can help classify music for different moods or activities (like "Chilled-Out Beats" vs. "Workout Jams"). Instead of using 100,000 points (for the 100,000 tracks), a random sampling of 10,000 tracks was used to keep a readable point-graph.

```
library(knitr)
knitr::include_graphics("output6_danceabilityvsenergy_10kpointssamplednot100kpoints-foreaseo
```



Overall, the points are distributed across the entire range of both danceability (x-axis) and energy (y-axis), from 0.00 to 1.00. There's a general pattern where most of the data points fall somewhere in the upper-middle of both the axes, with danceability and energy showing moderate to high values. Danceability and energy are relatively independent, but show some overlap in trends. Higher energy tracks (e.g., more intense, fast-paced music) are not necessarily highly danceable, and vice versa. Plus, the large cluster in the upper middle section of the graph - where danceability is 0.50 and is between 0.75 and 1.00 - reveals much overlapping points, so there is a high density of tracks with similar danceability and energy values here.

2.2 Genre selection:

To focus the exploration, only tracks from the top ten "danceable" genres were included in data preprocessing. Despite previous objections to the subjectivity of the measure, danceability can still be chosen as a frame for analysis, as it is a key feature that significantly influences listener engagement and enjoyment. To determine which ten genres were the most "danceable," two passes were made to the data. The first pass set each track into either a "low," "medium," or "high" danceability: values lower than 0.4 were set to "low," values equal to or greater than 0.4 were set to "medium," and values higher than 0.7 were set to "high". Then I filtered the "high" danceability songs to display the head(), which gives us this:

```
#| include: true
#| warning: false
#| message: false

# Load packages
library(readr)  #for reading CSV files
library(knitr)  #for rendering tables nicely
library(dplyr)  #for good data manipulation
```

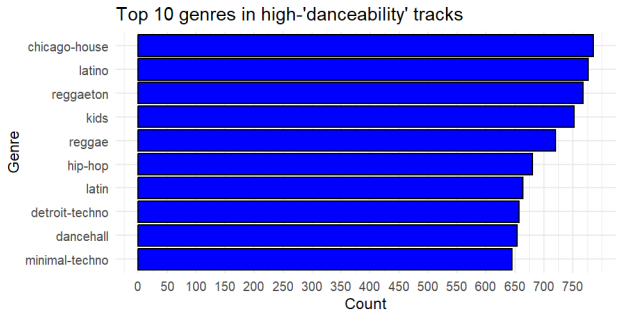
```
Warning: package 'dplyr' was built under R version 4.4.2
```

```
Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
#Set the path to the file
file_path <- "output3_top10genres_inhighdancetracks.png"

#Read in the file
knitr::include_graphics("output3_top10genres_inhighdancetracks.png")
```



2.3 Data cleaning:

From the dataset featuring only the tracks from the top ten 'danceable' genres, any rows with missing data was removed using `drop_na()`. The `track_genre` variable was encoded as a "factor" to ensure it could be used as a categorical target variable in the classification model.

2.4 Data splitting:

The data was split into a test set (20% of the original dataset) and a training set (80% of the original dataset), using a random seed to maintain reproducibility. This split allows the model to be trained on a large set of the data while holding back some portion to test the models' performances.

3 Building and evaluating Random Forest models

The Random Forest algorithm was used to classify tracks into one of the ten genres ("target") using the following predictors:

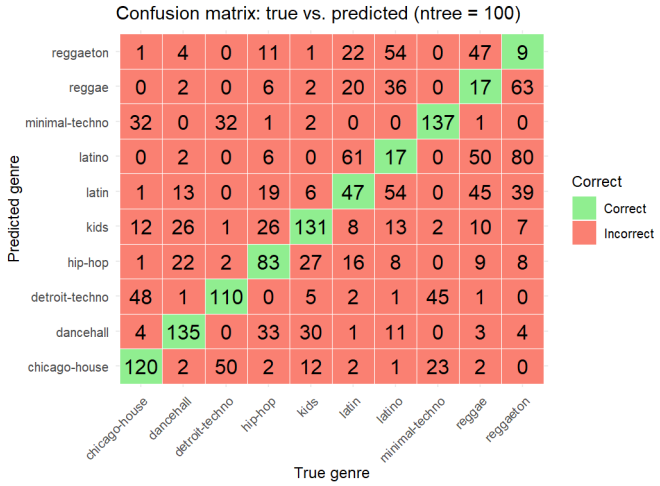
- danceability
- energy
- loudness
- speechiness
- acousticness
- instrumentalness
- liveness
- valence
- tempo

3.0.1 Working with "number of trees = 100"

The initial Random Forest model was trained with 100 trees (ntree = 100) to gauge its performance. For `ntree = 100`, evaluating the model shows 40.3% accuracy, and this confusion matrix:

```
#| include: true
#| warning: false
#| message: false

library(readr)
library(knitr)
```

```
#Set the path to the file
file_path_image_confusionmatrix_ntree100 <- "output13_generic_first_random_forest_model_visua

#Read in the file
knitr::include_graphics("output13_generic_first_random_forest_model_visualized_confusion_mati
```



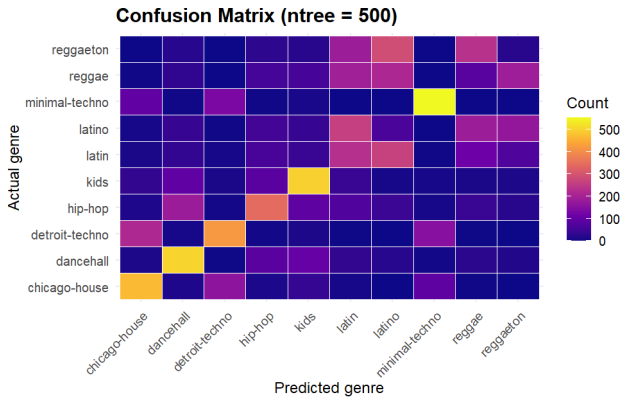
The confusion matrix shows how well the model predicted each genre, with diagonal values indicating correct predictions. The off-diagonal values show where the model made errors.

3.0.2 Using ntree = 500

As the accuracy of first trained RF model was low, I changed the parameter., to control how many decision trees the model builds when training. This can generally lead to more stable prediction.

For 500 trees, the OOB error rate is 60.2%, so better than the ntree = 100. For "chicago-house", the model correctly predicted 473 tracks as "chicago-house", but misclassified 158 as "detroit-techno", 10 as "dancehall", etc. Compared to the 100-tree model, the number of misclassifications decreased slightly.

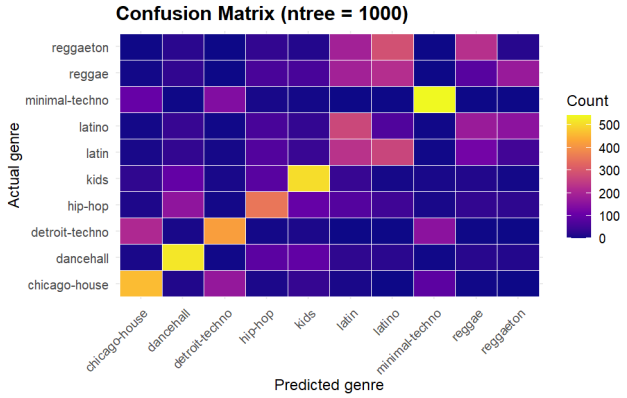
```
#Set the path to the file
file_path_image_confusionmatrix_ntree500 <- "output14_matrix_ntree500.png"

#Read in the file
knitr::include_graphics("output14_matrix_ntree500.png")
```



3.0.3 Using ntree = 1000

To test if more trees would help the accuracy go up, I went further with 1000 trees. The OOB error rate was 60.34 %, so, unfortunately, beyond 500 trees, increasing the number of trees to 1000 does not lead to significant improvements in OOB error or class error, suggesting that the model has converged or probably reached optimal point at 500 trees.

```
#Set the path to the file
file_path_image_confusionmatrix_ntree1000 <- "output15_matrix_ntree1000.png"

#Read in the file
knitr::include_graphics("output15_matrix_ntree1000.png")
```



Since the 500-tree model achieved an accuracy of ~40%, it seems reasonable given the complexity of multi-class classification and the similarity between certain genres.

4 Results and discussion

4.1 Multi-class classification challenges in genre prediction

Achieving high accuracy in multi-class classification is inherently challenging, especially in scenarios where classes exhibit overlapping characteristics. In the case of the 500-tree random forest model, the overall accuracy of approximately 40% aligns with expectations for multi-class classification tasks with complex class overlaps. Musical genres such as "Chicago House" and "Minimal Techno," for instance, share similar tempo and rhythm features, making them difficult for the model to distinguish. Additionally, overlaps in key metadata features—such as energy, tempo, and danceability—can blur the boundaries between genres like "Latin" and "Latino," further complicating classification. In the ideal 500-tree model:

4.1.1 Misclassification Between Chicago House and Detroit Techno

The genres Chicago House and Detroit Techno are frequently confused in the 500-tree model:

Chicago House is misclassified as Detroit Techno 158 times. Detroit Techno is misclassified as Chicago House 211 times. This mutual misclassification is likely due to their shared roots in electronic dance music. Both genres exhibit similar musical features, including tempo, energy, and rhythmic patterns. Historically, Chicago House and Detroit Techno developed in parallel as part of the electronic music movement, which may contribute to overlapping stylistic elements that challenge the model's ability to distinguish between them.

4.1.2 Misclassification Between Dancehall and Hip-Hop

The model also struggles to differentiate between Dancehall and Hip-Hop:

Dancehall is misclassified as Hip-Hop 173 times. Both genres share rhythmic characteristics, particularly in terms of beat and tempo. While Dancehall is rooted in reggae rhythms such as the one-drop and riddim, Hip-Hop similarly emphasizes heavy beats with grooves that may overlap stylistically. These shared rhythmic structures likely contribute to the model's difficulty in accurately classifying tracks between the two genres.

4.1.3 Misclassification between Latin and Latino

The genres Latin and Latino show significant overlap in their classifications:

Latin is misclassified as Latino 271 times. Latino is misclassified as Latin 68 times. This mutual confusion is likely driven by shared musical features, such as rhythm, tempo, and instrumentation, which stem from common Latin American musical traditions. Both genres

often emphasize danceable rhythms and lively energy, which can make them difficult to distinguish based on metadata alone. The overlap in terminology may also reflect ambiguities in the genre labeling process itself.

4.1.4 Misclassification Between Latino and Reggae

The model also highlights overlaps between Latino and Reggae:

Latino is misclassified as Reggae 252 times. Reggae is misclassified as Latino 216 times. Despite their distinct cultural roots, these genres share certain rhythmic and instrumental characteristics that may account for the misclassifications. Both genres frequently utilize syncopated rhythms and emphasize the offbeat, characteristics that could cause the model to group them together. Additionally, both are highly dance-oriented, which may further complicate classification based on features like tempo and energy.

4.2 Implications of Misclassification Patterns

These results highlight the complexities of multi-class genre classification and underscore the importance of understanding the underlying similarities between genres. Misclassifications reveal meaningful insights into the shared features and stylistic overlaps of musical genres, which could inform future model refinements. By exploring these overlaps further, it could be possible to make more robust feature engineering techniques or alternative modeling approaches (besides Random Forest, for example) that better capture subtle distinctions between these genres (which, at best, could be described as 'arbitrarily assigned').

4.3 Weaknesses and next steps

Moving forward, enhancing the feature engineering process, incorporating additional metadata, and exploring alternative machine learning algorithms—such as deep learning models or ensemble methods—could improve classification accuracy. Furthermore, I could try expanding the dataset and refining genre definitions, to help the model better differentiate between "overlapping" genres, allowing for more precise predictions.

5 Conclusion

This study examined the use of random forest models for classifying music genres based on features like danceability, energy, and tempo. While the 500-tree model achieved reasonable accuracy, challenges arose due to overlapping characteristics between certain genres, such as "chicago-house" and "detroit-techno." These misclassifications highlight the complexities of

multi-class classification tasks. Future work could focus on refining the feature set, incorporating additional data, and exploring alternative machine learning algorithms to improve model performance and address these limitations.

6 References