Bop or Flop? Predicting Hits and Flops on Spotify

Justin Lee
jlee363@u.rochester.edu

Sammy Potter
spott14@u.rochester.edu

April 27, 2023

Abstract

This paper outlines the data mining process done on the The Spotify Hit Predictor Dataset (1960-2019), complete with 28,774 tracks, each with 19 normalized attributes. This paper was done as a part of the final project of the CSC 240: Data Mining course at the University of Rochester in Spring 2023.

Keywords– decision tree; gradient boosting; data preprocessing; machine learning; classification model

1 Introduction

The ability to predict whether or not a song will be a popular hit has many benefits, including (1) helping music industry executives make better decisions regarding artist signings, song promotions, and other investments, (2) increased revenue for music companies and musicians alike, and (3) improved music discovery for listeners by being able to make improved recommendations.

In this paper, we first outline the process by which we explored, cleaned, and preprocessed the data. After, we provide an overview of the model training process used to train two types of classification models, Decision Tree and Gradient Boosted Tree. Then, we conclude with an analysis and discussion of our results, and explore the implications of our findings.

2 Exploratory Data Analysis

We are first provided with a description of the dataset, which includes its number of songs, its number of attributes, and a description of each attribute. We are also told that there are no missing data values, which makes our data cleaning process much easier. Finally, the author of the dataset also provides us with their definition of a 'flop', which is a song that (1) does not appear in the 'hit' list of that decade, (2) whose artist does not appear in the 'hit' list of that decade, (3) belongs to a non-mainstream genre, (4) whose genre does not have a song in the 'hit' list, and (5) has the US as one of its markets.

Our first step was to gain a better understanding of the characteristics of the dataset beyond the description provided by the author. Generally, there were 3 parts to this process:

1. General Data Visualization and Exploration
2. Distribution of Songs Across Decades
3. Distribution of Songs Across Popularity

2.1 General Data Visualization and Exploration

Our initial step was to create a correlation matrix in order to identify any obvious linear relationships between any two attributes. This was done by utilizing the Matplotlib Python library. The correlation matrix is presented in figure 1. We identified two variables with a particularly high linear relationship, namely duration_ms and sections. As such, we knew that they were redundant and that we were able to remove one of them.

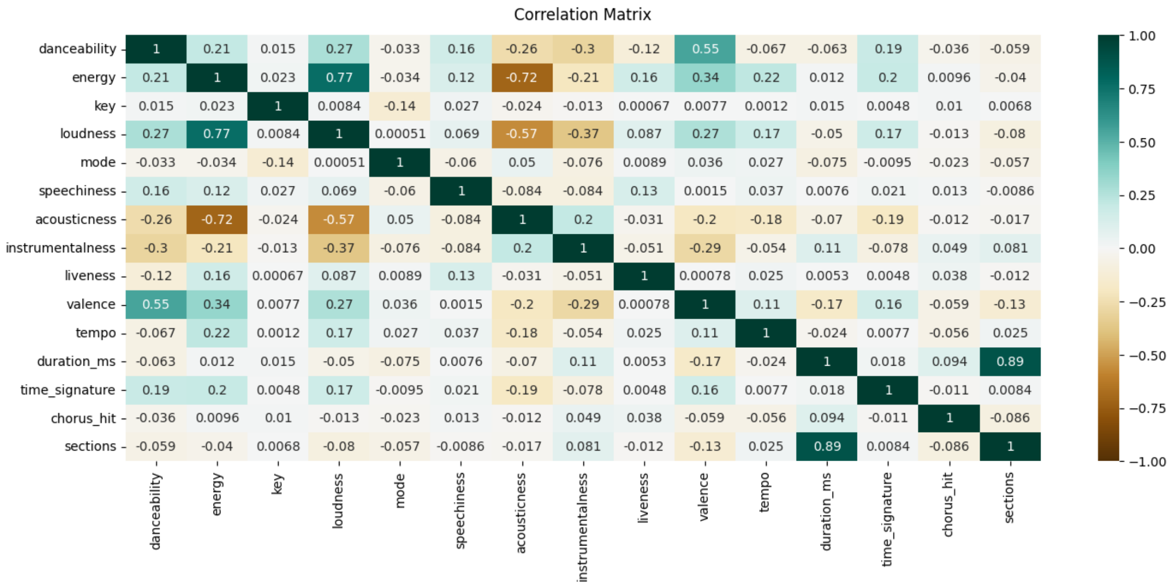

Figure 1: Correlation Matrix of Track Attributes

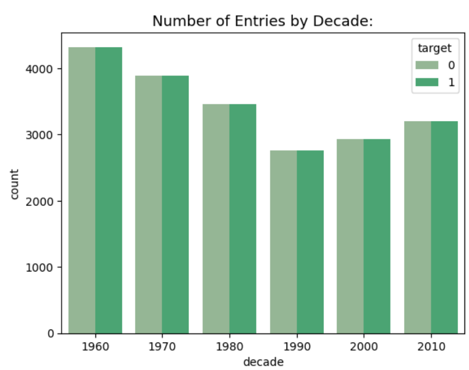

Figure 2: Correlation Matrix of Track Attributes

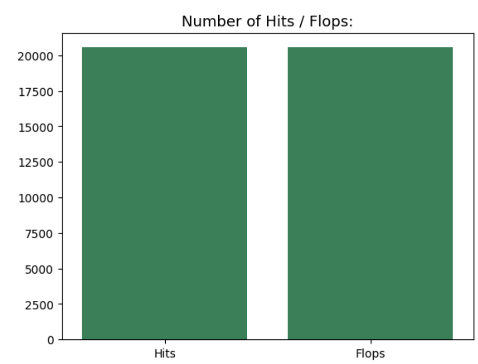

Figure 3: Correlation Matrix of Track Attributes

2.2 Distribution of Songs Across Decades

In order to avoid skewed data, we needed to make sure that the tracks were roughly evenly distributed across all the decades. Once again, we utilize the Matplotlib Python library to visualize the distribution of songs across decades. The results are presented in figure 2. From this, we knew that the distribution of songs across the decades from the 1960's to 2010's was acceptable for the purposes of this project.

2.3 Distribution of Songs Across Popularity

In addition to ensuring that our data was evenly distributed across decades, we needed to make sure that the number of 'hits' were close to equal the number of 'flops' in order to avoid skewed results. Once again, we utilized the Matplotlib Python library to visualize the distribution of songs across 'hits' and 'flops'. The results are presented in figure 3. From this, we ensured that the data would not be skewed in favor of either 'hits' or 'flops'.

3 Data Cleaning and Preprocessing

3.1 Data Concatenation

The dataset came in the form of 6 separate CSVs of tracks for each of the decades. Our first step to data cleaning and preprocessing was to parse all the data into a singular mutable data structure. However, by concatenating all the tracks from the different into a singular data structure, we lose information about which decade the song is from. This problem is later solved via feature engineering.

3.2 Feature Selection and Engineering

After concatenating all of our data into a single data structure, we then began the process of dropping any unwanted attributes in our data. We identified the attributes track, artists, and URI as irrelevant nominal values. As such, we dropped those.

By concatenating our data, we lost information about every given track's decade. To fix the problem of losing information about the tracks' decades (as stated above), we add another attribute called decade for each song, which contains information about what decade the track is from.

4 Training Models

To begin the process of training our models, we first partitioned the dataset into different sets for training and testing. In short, 80% of the original dataset was used for training, and the other 20% was used for testing. Then, we trained two models: a decision tree and a gradient boosted tree.

4.1 Decision Tree

Decision trees recursively classify data based on individual attributes based on branch purity. Generally, they are efficient, widely used in practical applications, and can be combined into more powerful models. To train our decision tree, we utilize the scikit-learn Python library.

4.2 Gradient Boosted Tree

Gradient Boosting uses an ensemble of Decision Trees, and combines them using a loss function. Compared to Decision Trees, Gradient Boosted Trees have a higher model capacity and can model more complex relationships. Once again, in order to train our decision tree, we utilize the scikit-learn Python library.

5 Results, Analysis, and Discussion

To gauge our models' performance, we took measures of the following: classification accuracy, classification error, precision, recall/sensitivity, true positive rate, false positive rate, and specificity. Our results are presented in the following tables.

Decision Tree Analysis	
Classification accuracy:	0.72148
Classification error:	0.27852
Precision:	0.71474
Recall / Sensitivity:	0.72530
True Positive Rate:	0.72530
False Positive Rate:	0.28225
Specificity:	0.71775

Gradient Boosting Analysis	
Classification accuracy:	0.79141
Classification error:	0.20859
Precision:	0.71231
Recall / Sensitivity:	0.84700
True Positive Rate:	0.84700
False Positive Rate:	0.24905
Specificity:	0.75095

To better visualize the true positive, false positive, true negative, and false negative rates, we create confusion matrices for both models. These confusion matrices are presented in figures 4 and 5.

The Decision Tree's performance included the following: 2970 true positives, 1110 false

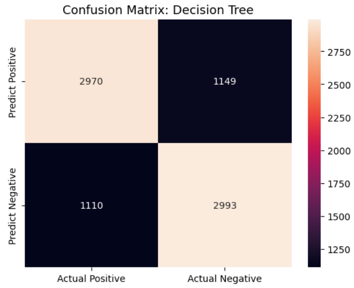

Figure 4: Confusion Matrix – Decision Tree

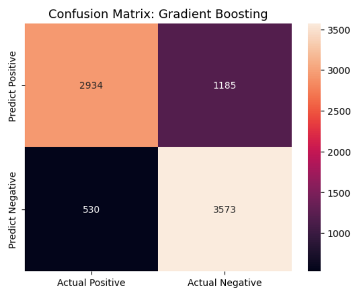

Figure 5: Confusion Matrix – Gradient Boosting

positives, 1149 false negatives, and 2993 true negatives. The Gradient Boosted performance included the following: 2934 true positives, 1185 false positives, 530 false negatives, 3573 true negatives.

We also wanted to better visualize and understand the importance and weight of each attribute for both models. We did so by generating the weights for each attribute for both models. The results are shown in the following tables:

Decision Tree Feature Importance	
instrumentalness	0.228866
acousticness	0.103122
danceability	0.091549
speechiness	0.076447
duration_ms	0.073270
energy	0.070725
loudness	0.054187
valence	0.052145
tempo	0.049916
liveness	0.048602
chorus_hit	0.046746
decade	0.044127
key	0.022992
sections	0.021951
mode	0.010178
time_signature	0.005175

Gradient Boosting Feature Importance	
instrumentalness	0.446068
acousticness	0.134828
danceability	0.117856
speechiness	0.066420
duration_ms	0.061402
decade	0.060054
energy	0.038931
loudness	0.025344
valence	0.020281
mode	0.010614
sections	0.007140
tempo	0.006818
liveness	0.001457
time_signature	0.001239
chorus_hit	0.000981
key	0.000569

References

[Ansari(2020)] Farooq Ansari. 2020. *The Spotify hit Predictor Dataset (1960-2019).* https://www.kaggle.com/datasets/theoverman/the-spotify-hit-predictor-dataset

[Econoscent(2015)] Econoscent. 2015. Visual guide to gradient boosted trees (xgboost). Online video. https://www.youtube.com/watch?v=TyvYZ26alZs

[Econoscent(2020)] Econoscent. 2020. Visual guide to decision trees. Online video. https://www.youtube.com/watch?v=zs6yHVtxyv8

[Spotify([n. d.])] Spotify. [n. d.]. *Spotify: Music for everyone.* https://spotify.com/

Appendix:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import confusion_matrix

df60 = pd.read_csv('60s.csv')
df70 = pd.read_csv('70s.csv')
df80 = pd.read_csv('80s.csv')
df90 = pd.read_csv('90s.csv')
df00 = pd.read_csv('00s.csv')
df10 = pd.read_csv('10s.csv')

dfs = [df60,df70,df80,df90,df00,df10]

decades = [1960,1970,1980,1990,2000,2010]
for i in range(len(decades)):
    dfs[i]['decade'] = pd.Series(decades[i], index=dfs[i].index)

df = pd.concat(dfs).reset_index(drop=True)

plt.title(f"Number of Entries by Decade:", fontsize=13)
sns.countplot(data=df, x="decade", hue="target", palette=["darkseagreen", "
    mediumseagreen"]) # color="seagreen"

# Choose the attribute you want to count and assign it to a variable
attribute = 'target'

# Count the number of items that have the chosen attribute
hitCount = len(df[df[attribute] == 1])
flopCount = len(df.index)-hitCount

plt.title(f"Number of Hits / Flops:", fontsize=13)
sns.barplot(x=['Hits', 'Flops'], y=[hitCount, flopCount], color="seagreen")

df = df.select_dtypes(include=['float64','int64'])

X = df.drop("target", axis=1)
Y = df["target"]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8,
    shuffle=True, random_state=1)

scaler = StandardScaler()
scaler.fit(X_train)
X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns
    =X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=
    X_test.columns)

df.head()
```

```
# Correlation Matrix

plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(df.drop(['decade', 'target'], axis=1).corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Correlation Matrix', fontdict={'fontsize':12}, pad=12);

print("Training Decision Tree...")
decisionTree = DecisionTreeClassifier()
decisionTree.fit(X_train, Y_train)

print("Training Gradient Boosting...")
gradientBoosting = GradientBoostingClassifier()
gradientBoosting.fit(X_train, Y_train)

print("Done.")

def analyze_model(model, modelName):
    print(f"{modelName} analysis:")
    print('-'*41)

    Y_pred = model.predict(X_test)

    conf_matrix = confusion_matrix(Y_test, Y_pred)

    TP = conf_matrix[0,0]
    TN = conf_matrix[1,1]
    FP = conf_matrix[0,1]
    FN = conf_matrix[1,0]

    classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
    classification_error = (FP + FN) / float(TP + TN + FP + FN)
    precision = TP / float(TP + FP)
    recall = TP / float(TP + FN)
    true_positive_rate = TP / float(TP + FN)
    false_positive_rate = FP / float(FP + TN)
    specificity = TN / (TN + FP)

    print(f"{'Classification accuracy':<25} {classification_accuracy:>15.5f}")
    print(f"{'Classification error':<25} {classification_error:>15.5f}")
    print(f"{'Precision':<25} {precision:>15.5f}")
    print(f"{'Recall / Sensitivity':<25} {recall:>15.5f}")
    print(f"{'True Positive Rate':<25} {true_positive_rate:>15.5f}")
    print(f"{'False Positive Rate':<25} {false_positive_rate:>15.5f}")
    print(f"{'Specificity':<25} {specificity:>15.5f}")

    cm_square = [[TP,FP],[FN,TN]]

    cm_matrix = pd.DataFrame(data=cm_square, columns=['Actual Positive', 'Actual Negative'],
                                    index=['Predict Positive', 'Predict Negative'])

    plt.title(f"Confusion Matrix: {modelName}", fontsize=13)
    sns.light_palette("seagreen", as_cmap=True)
    sns.heatmap(cm_matrix, annot=True, fmt='d')

    # Compute the feature importances
    feature_importances = model.feature_importances_

    # Create a DataFrame of feature importances
```

```
    feat_imp_df = pd.DataFrame({'feature': X_train.columns, 'importance': feature_importances})

    # Sort the DataFrame by feature importance in descending order
    feat_imp_df = feat_imp_df.sort_values('importance', ascending=False)

    top_feat_imp = feat_imp_df

    top_feat_imp = top_feat_imp.reset_index(drop=True)

    print()
    print(f"{modelName} Feature Importance:")
    print('-'*41)
    print(top_feat_imp[['feature', 'importance']].to_string(index=False))

analyze_model(decisionTree, "Decision Tree")
analyze_model(gradientBoosting, "Gradient Boosting")
```