Songs Classifier: Machine Learning Models for Spotify Song Genre Classification

Yidong Wu
Department of Statistics
University of California, Berkeley
Berkeley, CA 94720
areswu@berkeley.edu

Abstract

This project presents the development of a machine learning-based classifier designed for categorizing Spotify songs into genres. Various machine learning models, including NB, DT, RF, GBM, MLP, and CNN, are explored and compared regarding classification accuracy and computational efficiency. The findings contribute to understanding the trade-offs between model complexity and performance in machine learning, offering insights for optimizing song classification in digital music platforms.

1 Introduction

In the digital era, music streaming services like Spotify have transformed how we access music, with song classification playing a pivotal role in enhancing user experience through personalized recommendations. In this paper, I will try to build a Spotify Songs Classifier, which applies machine learning techniques to categorize songs based on their intrinsic characteristics.

The significance of this project lies in its potential to contribute to the domain of music recommendation systems. By accurately classifying songs, the classifier aids in constructing more precise user profiles, thereby refining the recommendation algorithms used by Spotify.

Furthermore, this project explores various machine learning paradigms, ranging from traditional methods like Naive Bayes and Decision Trees to advanced neural network approaches. I will compare these models in terms of accuracy and computational efficiency. This analysis contributes to the optimization of the song classification process and provides valuable insights into the trade-offs between model complexity and performance efficiency in machine learning applications.

The following sections will detail the dataset and preprocessing steps, describe the machine learning and neural network methods employed, discuss the results obtained, and conclude with reflections on the project's outcomes and potential future work in this exciting field.

2 Dataset and Features

2.1 Brief Description

In this project, I use the "30000 Spotify Songs" dataset from Kaggle. This dataset contains nearly 30,000 songs from the Spotify API, and primarily focuses on a wide range of musical characteristics that are crucial for song classification. These include attributes such as ‘danceability‘, ‘energy‘, ‘loudness‘, ‘tempo‘, ‘valence‘, ‘key‘, ‘mode‘, and other perceptual features that define the musical quality of the songs. Meanwhile, the dataset also contains essential track identification and contex-

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

tual information such as track names, album details, and songs genres. It ensures a rich and relevant data source for the classification task.

2.2 Data Preprocessing

After reading the dataset, a custom function is applied to convert the album release dates into years for convenience. Following this, the dataset is streamlined by removing several columns, focusing exclusively on retaining the songs' characteristics and release years. The columns are then renamed for better clarity. To ensure data integrity, any rows with missing values are removed. The genre data, initially in text format, is transformed into a numerical representation through a mapping process, assigning a unique integer (from 0 to 5) to each genre.

The dataset is then divided into a feature matrix X containing songs' characteristics and a target vector y, representing song genres. I also normalize the features in X, which helps equalize each feature's influence on the model, avoiding biases toward variables with larger scales. In this step, 'year', 'key', and 'mode' are not normalized as they are categorical. Finally, the dataset is split into training and testing sets in an 80%-20% ratio, crucial for training the model and evaluating its performance.

In preparation for neural network training, one-hot encoding is applied to specific categorical columns in the dataset. Specifically, the 'year' and 'key' columns and the target vector y undergo this transformation. One-hot encoding is crucial for neural networks as it converts categorical data into a format that can be effectively processed, avoiding the ordinal implications of numerical encoding. Note that this step is done after performing the traditional ML methods.

3 Methods

3.1 Gaussian Naive Bayes

Gaussian Naive Bayes is one of the most commonly used Naive Bayes classifiers for continuous variables, assuming that features are normally distributed. The classifier is built on Bayes' theorem, which provides a way of finding the probability of a label given some observed features, denoted as $P(Y|X)$.

The probability density of a feature X_i given a class k is expressed as:

$$P(X_i = x_i|Y = k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} \exp\left(-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}\right)$$

where $\mu_{k,i}$ is the mean and $\sigma_{k,i}^2$ is the variance of the feature X_i in class k.

The Gaussian Naive Bayes classifier incorporates the Naive assumption of conditional independence between every pair of features given the class label. This assumption simplifies the posterior probability computation, as the feature vector's joint probability can be calculated as the product of individual probabilities. Given the independence assumption, the posterior probability that an instance x belongs to class k is then given by:

$$P(Y = k|X = x) = \frac{P(Y = k)\prod_{i=1}^{p} P(X_i = x_i|Y = k)}{P(X = x)}$$

3.2 Decision Tree

Decision Trees model decisions and their possible consequences as a tree structure, where each internal node denotes a test on an attribute, each branch represents the outcome of the test, and each leaf node corresponds to a class label or regression value.

Constructing a Decision Tree involves splitting the dataset into subsets based on an attribute value. This process, known as recursive partitioning, continues until the subsets are pure or until further splitting provides no additional benefit. The decision of where to split the data is made using criteria such as the Gini impurity for classification tasks: $\text{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2$, where p_i is the proportion of instances of class i in the dataset D. The algorithm will choose the attribute with the lowest Gini impurity to split on.

3.3 Random Forest

Random Forest is an ensemble learning technique that improves upon bagging by creating a collection of decision trees with controlled variance. Each tree in a Random Forest is built from a random subset of the data, chosen with replacement, and uses a subset of features at each split, typically $\sqrt{n}$, to determine the best split. This introduces variation among the trees, which, when aggregated, results in a model with high accuracy and robustness to overfitting. The decorrelation between the trees comes from this randomness in the selection of both samples and features. By doing so, Random Forest ensures that the behavior of any single tree does not dominate the ensemble's prediction, thus enhancing the overall performance.

The final prediction for this classification task is obtained by performing majority voting across all trees. The class that receives the majority of votes from the individual trees is chosen as the final prediction for each input sample.

3.4 Gradient Boosting Machine

Gradient Boosting Machine implements an ensemble of decision trees sequentially. The process starts with a simple initial model, typically a decision tree, which makes predictions and identifies errors. Successive models are then added, each targeting the residuals or errors left by the previous ones. This iterative approach continues for a predetermined number of steps or until the model performs satisfactorily.

The entire theory and algorithm are complex, and they are not the primary focus of this paper. Therefore, I will not discuss them in detail. Readers are referred to [Author's Name, Year] for an in-depth understanding.

3.5 Multilayer Perceptron

A Multilayer Perceptron (MLP) is a feedforward artificial neural network integral to complex pattern recognition and classification tasks. It consists of multiple layers of nodes, each fully connected to the next. Nodes apply a nonlinear activation function to their inputs' weighted sum. MLPs employ backpropagation for training, adjusting weights iteratively to minimize the output prediction error.

In this project, the MLP model I built is configured for a classification task. It commences with an input layer corresponding to the feature count of the scaled training data. The architecture includes two hidden layers, with 128 and 256 neurons, respectively, using the ReLU activation function. After each hidden layer, a dropout layer with a 0.3 rate is incorporated for regularization. The output layer, tailored for multi-class classification, comprises neurons equal to the class count, utilizing the softmax activation function. It is compiled with the Adam optimizer and categorical cross-entropy loss function.

3.6 Convolutional Neural Network

CNNs are a class of deep neural networks that are highly effective in processing data, such as images and sequences, with a grid-like topology. Unlike MLP, CNNs are designed to learn spatial hierarchies of features from data automatically and adaptively. This is achieved through the use of convolutional layers that apply a convolution operation to the input, capturing spatial dependencies and pooling layers that reduce the dimensions of the data, hence reducing the number of parameters and computational cost.

In this project, a 1D CNN model is built for classifying structured data. The model includes convolutional layers, max-pooling layers for downsampling, a flattening layer, and fully connected dense layers for classification. Specifically, the model starts with a convolutional layer with 32 filters, followed by a max pooling layer, another convolutional layer with 64 filters, a second max pooling layer, and then fully connected layers.

Before training the model, I reshaped the 2D feature matrix into a 3D format. The original data, in a 2D shape of [samples, features], is reshaped to [samples, features, 1]. Each feature in the original dataset is treated as a single sequence step in the reshaped data, allowing the 1D CNN to process each feature independently while maintaining the overall structural integrity of the data.

3.7 Stacked Ensemble Model

Building upon the previously described MLP and 1D CNN models, a stacked ensemble approach is implemented to enhance predictive performance. This model integrates predictions from the MLP and CNN using a weighted average method, with weights of 0.7 for the MLP and 0.3 for the CNN. The combined predictions form a new feature set for training an additional neural network. The stacked model's architecture comprises a 64-neuron dense layer, a dropout layer with a rate of 0.5, a second 32-neuron dense layer, and another dropout layer at a rate of 0.3, concluding with a softmax output layer for multi-class classification.

4 Results and Discussion

4.1 Performance

Table 1: Accuracy and Running Time across different models

Metric	NB	DT	RF	GBM	MLP	CNN	Ensemble
Test Accuracy	0.4801	0.4612	0.5835	0.5858	0.5922	0.5688	0.6616
CV Accuracy	0.4702	0.4560	0.5743	0.5743	0.5763	0.5527	0.6508
Running Time (s)	0.008	0.280	4.591	31.485	31.214	84.414	27.008

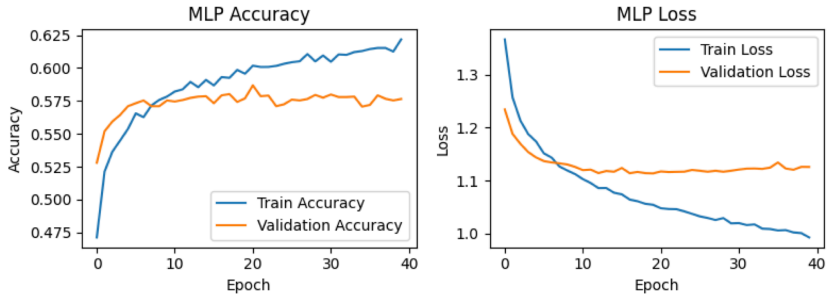

Figure 1: MLP Results

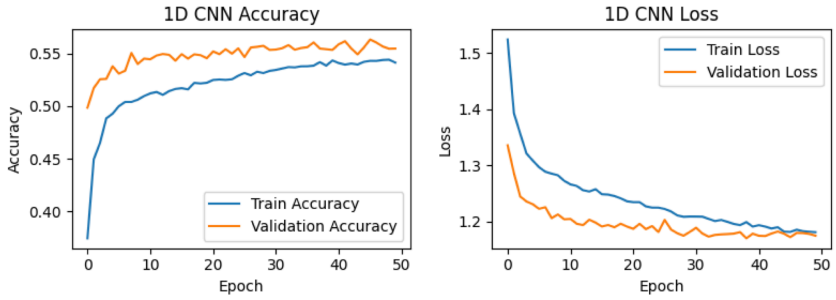

Figure 2: CNN Results

Test Accuracy is Correct Classifications / Total Classifications on the test set. For Cross Validation, I use a 5-folds CV. Running time is the time to train one model. I run the project on M1 Pro, with Python 3.11. Note that the running time for the ensemble model only accounts for the time spent training itself, which combines the predictions of the MLP and CNN models. However, this does not include the individual training times for the MLP and CNN models, which are also part of the overall computational cost of the ensemble approach.

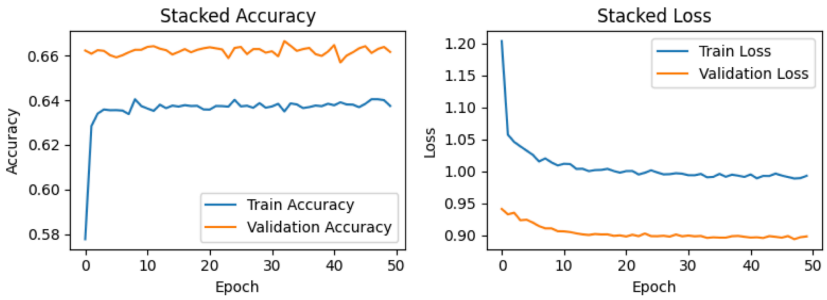

Figure 3: Ensemble Results

4.2 Discussion

Naive Bayes, known for its simplicity and speed, demonstrated the fastest running time. However, this model's simplicity also leads to lower accuracy, particularly when the feature independence and Gaussian distribution assumptions do not hold.

Decision Trees, offering interpretability and handling non-linear relationships, show the worst accuracy. It may suffer the problem of overfitting and has high variance. Random Forest mitigates some of DT's overfitting issues through ensemble learning, achieving better test accuracy but at the expense of increased computational time (not much).

The GBM model, which iteratively corrects errors of the weak learners, obtained a high accuracy. However, the trade-off is evident in its substantial running time, suggesting higher computational complexity. The same thing also happens in MLP.

Interestingly, CNN, despite its sophistication and capacity for feature extraction, did not surpass the MLP in accuracy and required the longest running time. This outcome may be attributed to CNN's convolutional layers being more suitable for spatial data processing, such as image recognition, rather than the dataset at hand.

The Ensemble model, which combines MLP and CNN predictions, achieved the highest test accuracy, illustrating that ensemble methods can leverage the strengths of individual models to improve overall performance. Although this model only runs a bit, it is based on MLP and CNN, so we need to have these two models upfront. In this way, the ensemble model needs much computation.

5 Conclusion and Future Work

This project presented an extensive study on applying various machine learning models to classify Spotify song genres. The analysis demonstrated that while traditional models like Gaussian Naive Bayes and Decision Trees provide foundational insights, advanced models like Multilayer Perceptrons and Convolutional Neural Networks show a more incredible promise in handling complex data structures. The ensemble model, combining predictions from both MLP and CNN, emerged as a robust solution, although with a higher computational requirement.

In future research, it would be interesting to use Convolutional Neural Networks (CNNs) for processing sound files. Sound data has unique time-based and frequency-based properties that could work well with the layers in CNNs, which are good at processing patterns. Using CNNs could help better identify and categorize detailed sound characteristics, which might improve how accurately different types of music are classified. Trying out various designs of CNNs and adjusting their settings for sound data could also improve how well they work. Combining features specific to sound with CNNs could lead to new advancements in classifying and recommending music.

References

[1] Thompson, C., Parry, J., Phipps, D., & Wolff, T. (2020) Spotify Songs. Retrieved from https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-01-21/spotify_songs.csv

[2] James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning. Vol. 112. New York: springer, 2013.

[3] Hastie, Trevor, Robert Tibshirani, Jerome H. Friedman, and Jerome H. Friedman. The elements of statistical learning: data mining, inference, and prediction. Vol. 2. New York: springer, 2009.

[4] Ayyadevara, V. Kishore. (2018). Gradient Boosting Machine. In *Pro Machine Learning Algorithms: A Hands-On Approach to Implementing Algorithms in Python and R* (pp. 117–134). Berkeley, CA: Apress. DOI: https://doi.org/10.1007/978-1-4842-3564-5_6.

[5] Huang, Derek A., Arianna A. Serafini, and Eli J. Pugh. "Music Genre Classification." CS229 Stanford (2018). Retrieved from https://github.com/derekahuang/Music-Classification

[6] Rahardwika, Dewangga Satriya, Eko Hari Rachmawanto, Christy Atika Sari, Candra Irawan, Desi Purwanti Kusumaningrum, and Swapaka Listya Trusthi. "Comparison of SVM, KNN, and NB classifier for genre music classification based on metadata." In 2020 international seminar on application for technology of information and communication (iSemantic), pp. 12-16. IEEE, 2020.

[7] N. Pelchat and C. M. Gelowitz, "Neural Network Music Genre Classification," in Canadian Journal of Electrical and Computer Engineering, vol. 43, no. 3, pp. 170-173, Summer 2020, doi: 10.1109/CJECE.2020.2970144.

Code and Results

For code and results in this project, please refer to https://github.com/Areswu16/Spotify-Songs-Classifier/blob/main/Spotify_songs_classifier.ipynb

You can also find them at https://www.kaggle.com/code/areswu16/spotify-songs-classifier. However, the code was run on the GPU provided by Kaggle, and the results might differ slightly from those in this report.