Spotify Playlist Classification

Audio and Music

Sneha Pendharkar
Electrical Engineering
snehasp@stanford.edu

SriRaagavi Ragothaman
Computer Science
raagavi@stanford.edu

***Abstract*—The number of features offered by music streaming services has been growing steadily, with companies constantly working to do better than their competitors. A ubiquitous feature of such services is the ability to create playlists so that the user can choose what genre of music to listen to. In this project, we explore the classification of songs into playlists so that a song that a user 'likes' is automatically classified into an existing playlist. We use five different classification algorithms - K-Nearest Neighbors, Naive Bayes, Logistic Regression, Support Vector Machines, and Random Forests, and train them on data obtained from the Spotify dataset. We use accuracy to measure the performance of the algorithms, and see that the random forest classifier performs the best, and the Naive Bayes algorithm performs the worst.**

I. Introduction

With music streaming services becoming more popular by the day, streaming apps such as Apple Music and Spotify are constantly competing with each other to equip their products with features and services that could better suit the tastes, needs, and preferences of the listener. This includes personal playlist curation, music recommendation, and artist suggestions. Our project zooms in on one such potential feature whereby a 'liked' song is automatically classified into one of two or more existing playlists that belong to the user. This is not currently a feature in any streaming apps that we're aware of and could save listeners the hassle of having to sift through and organize each song they 'like' into an appropriate playlist.

Our project essentially uses certain features of songs already in the user's playlists, such as 'danceability' and 'speechiness' to determine which playlist's music a particular song would resemble most. The inputs to each algorithm are the audio features stored in Spotify for a track. We use various supervised learning algorithms to classify this track into a playlist.

II. Related Work

Previous work in the area of music classification includes using neural networks, support vector machines, random forests, and gradient boosting to classify songs based on genre. Audio clips from different datasets like the Free Music Archive, the GTZAN dataset, and the Million Song Dataset were used in past research. These projects extracted the Mel-Frequency Cepstral Coefficients (MFCC) for the songs and used them to train the classification algorithms. MFCCs are a short-time spectral decomposition of an audio signal that conveys the general frequency characteristics important to human hearing. In our project, since we wanted to focus on classification of songs into playlists, we decided to use the Spotify dataset, which contains audio information about the songs. [1] [2] [3] [4] [5]

III. Dataset and Features

For this project, we used the widely available Spotify Web API, via Spotipy, a lightweight Python library for the Web API [6]. With Spotipy, we had full access to the music data provided by the Spotify platform. We used the library to extract the features of songs in 7 different playlists curated by Spotify, each from a different genre:

1) Get Turnt
2) Yoga and Meditation
3) Friday Cratediggers
4) Jazz Vibes
5) Tailgate Party
6) New Metal Tracks
7) Chill Hits

with each playlist consisting of a hundred examples. Each track has the following 12 features, which are represented numerical values:

1) Danceability
2) Energy
3) Liveness
4) Instrumentalness
5) Mode
6) Speechiness
7) Valence
8) Key
9) Loudness
10) Tempo
11) Duration (ms)
12) Time Signature

Features 1 through 7 are represented as values between 0 and 1. Key is represented by an integer that maps to a pitch using the standard pitch class notation. Loudness represents the overall loudness of a track in decibels (dB). Tempo refers to the speed of a song in beats per minute, duration is the length of a song in milliseconds. Time signature is a notational convention to specify how many beats are in a bar.

Figures 1, 2, and 3 on the next page show the distribution of some of these features for the different playlists. [7]

From each playlist, we got a dataset which we split into 2 parts: 80% used as training data and 20% used for validation. The training data from the 7 playlists was combined to make up the overall training data for the project, and similarly for the validation data. Each track was labeled according to the playlist it belonged to.

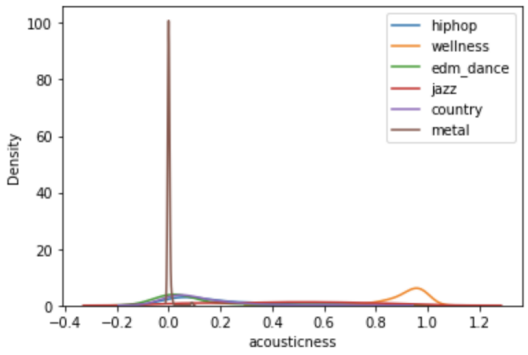

Figure 1. Feature distribution: Acousticness

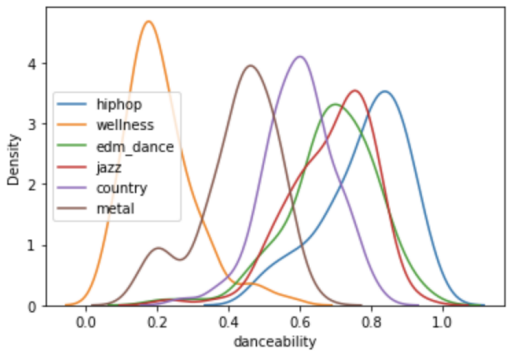

Figure 2. Feature distribution: Danceability

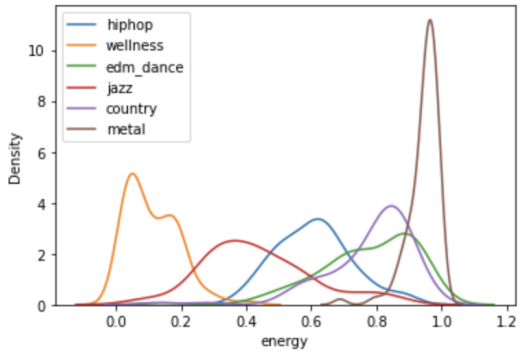

Figure 3. Feature distribution: Energy

IV. Methods

To classify a song into a playlist, we use five different supervised learning algorithms on our dataset: K-Nearest Neighbors, Naive Bayes, Logistic Regression, Support Vector Machines, and Random Forest Classifier. We focus on accuracy as a measure of how well each algorithm performs at classifying the songs.

1) **K-Nearest Neighbors**: This is a common classification algorithm where an input is classified based on the classification of its k nearest neighbors. In some cases it can be beneficial to assign weights to the points that will be considered during classification. For example, we may assign weights to points by the inverse of their distance. This way closer neighbors of a query point will have a greater influence than neighbors which are farther away. Additionally, the value of k also has an effect on training and validation accuracy: a larger value of k reduces the effect of noise, but makes the classification boundary less distinct.

2) **Naive Bayes**: The next algorithm we used was Multinomial Naive Bayes. Naive Bayes classifiers are based on applying Bayes' theorem with the assumption that of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship for a class variable y and features x_1 through x_n

$$P(y|x_1,...,x_n) = \frac{P(y)P(x_1,...,x_n|y)}{P(x_1,...,x_n)} \tag{1}$$

With our assumption, we get

$$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,...,x_n)} \tag{2}$$

Since the denominator is constant given the input, we use the following classification rule:

$$\hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i|y) \tag{3}$$

Multinomial Naive Bayes implements the Naive Bayes algorithm for multinomially distributed data.

3) **Logistic Regression**: Unlike Naive Bayes, this algorithm doesn't assume conditional independence of the features. We define the sigmoid function as follows:

$$\sigma(w^T x + b) = \frac{1}{1 + exp(-(w^T x + b))} \tag{4}$$

For binary logistic regression, we use this as our hypothesis, and we have

$$p_w(y = 1|x) = \sigma(w^T x + b) \tag{5}$$

$$p_w(y = 0|x) = 1 - \sigma(w^T x + b) \tag{6}$$

Multiclass logistic regression is an extension of binary logistic regression. Instead of y = 0, 1, we have n different values for y. We essentially run binary classification multiple times, once for each class. For each subproblem, we select one class, lump the other classes together, and predict the probability that the observation is in that class. To classify an input we take the maximum of these probabilities.

4) **Support Vector Machines**: To perform multiclass classification using support vector machines, we break down the problem into multiple binary classification problems. The one-to-one approach, which is used in the Scikit

learn implementation, sets a binary classifier for each pair of classes [8]. A single SVM does binary classification and can differentiate between 2 classes, so for n classes we require $\frac{n(n-1)}{2}$ classifiers. With an SVM, our goal is to try to find a decision boundary that maximizes the geometric margin. This gives us the following optimization objective:

$$min_{w,b} = \frac{1}{2}||w||^2 \tag{7}$$
$$s.t.\ y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, ..., n$$

5) **Random Forest Classifier**:
The Random Forest Classifier (RFC) is good for applications that involve uncorrelated outcomes. In order to test whether exploiting a potential lack of correlation would increase accuracy, we used the RFC algorithm. The working of the algorithm can be compared to the idea of 'wisdom of crowds' wherein a large number of individual decision trees constitute a random forest. Each tree in the random forest gives a class prediction and the class with the most votes becomes the model's final prediction as shown below in Figure 4.

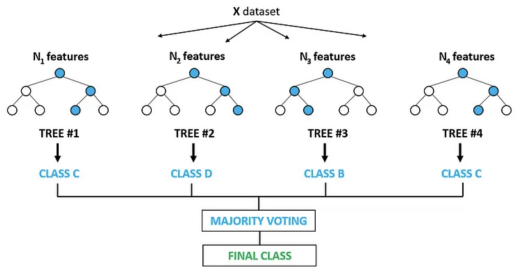

Figure 4. Random Forest Classifier decision trees

V. Experiments/Results/Discussion

For each method outlined above, a simple accuracy metric was employed, defined as:

$$\frac{\text{\# songs in validation set classified correctly}}{\text{\# songs in the validation set}}$$

For the preliminary experiment, the K-Nearest Neighbors algorithm as implemented by Python's Scikit package was used to fit training data from the 7 playlists outlined in Section III, with 20% of the dataset being reserved for validation [8]. The package's feature selection method was used to narrow down the number of features to avoid overfitting while still keeping a substantial number of them to avoid underfitting the model. This feature selection process helped us understand how indicative a feature is of the playlist a song belongs to.

The feature list was narrowed down to the following 7 primary features that were responsible for significant variation in the real value of the playlist each song belonged to:

1) Danceability
2) Energy
3) Loudness
4) Speechiness
5) Instrumentalness
6) Valence
7) Acousticness

As seen from Figure 5 below, with each added or removed feature, the accuracy of the K-Nearest Neighbors model decreased, regardless of the number and category of the playlist included in the training dataset.

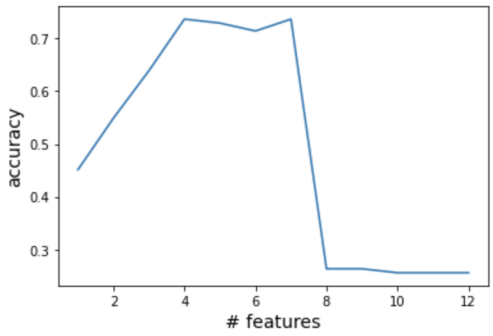

Figure 5. Testing accuracy

This result demonstrated an evident lack of correlation between features such as key, tempo, and duration on the classification of a song in each playlist. However, more intuitively correlated features such as danceability, instrumentalness, and acousticness were indicative of the song's classification. These 7 features were used when training the all of the algorithms.

Given below are our experiments and results for the different algorithms. We give the accuracy of our algorithm after hyperparamter tuning, as well as the confusion matrix. A confusion matrix is often used to describe the performance of a classification model on test data for which the true values are known. Entry i, j in the matrix is the number of observations actually in group i, but predicted to be in group j.

1) **K-Nearest Neighbors**: An important hyperparameter in this algorithm is the number of neighbors considered when performing classification. We used grid search to find the optimal value for this parameter, and found that number of neighbors equal to 9 gave us the best accuracy of **0.78947368**. Figure 6 on the next page shows the confusion matrix for this algorithm.

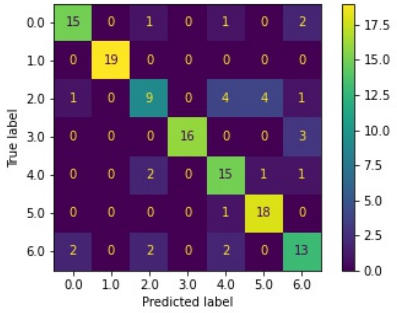

Figure 6. KNN Confusion Matrix

2) **Naive Bayes**: For the Naive Bayes algorithm, we used the Scikit Learn implementation to both fit the model as well as predict the outcomes [8]. We used grid search to find the optimal value for α, which is the additive Laplace smoothing parameter. We found the optimal value to be 0.1, which gave us an accuracy of **0.76691729**. Figure 7 on the next page shows the confusion matrix for this algorithm.

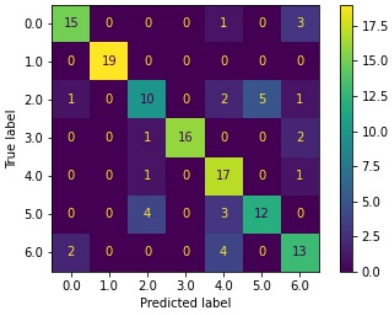

Figure 7. Naive Bayes Confusion Matrix

3) **Logistic Regression**: For the logistic regression algorithm, we used the Logistic Regression CV classifier that is part of Scikit learn [8]. With 3 fold cross validation, we obtained an accuracy of **0.84962406**. Figure 8 below shows the confusion matrix for the logistic regression algorithm.

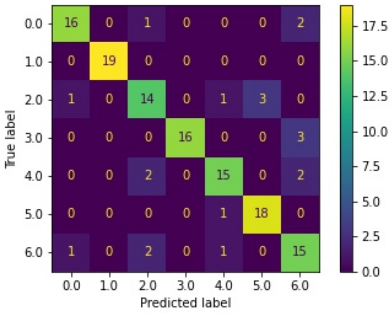

Figure 8. Logistic Regression Confusion Matrix

4) **Support Vector Machine**: For classification using support vector machines, we used the Support Vector Classification implementation that is part of Scikit learn [8]. We tried 3 different kernels - rbf, poly, and linear. Additionally, to tune the hyperparameters we used Randomized Search to try different combinations of C, gamma, and degree (only for the poly kernel) which are parameters of the classifier.

a) RBF Kernel - With optimum values for hyperparameters (gamma = 0.1, C = 50), our accuracy was **0.84210526**.
b) Linear Kernel - With optimum values for hyperparameters (gamma = 0.1, C = 5), our accuracy was **0.83458646**.
c) Poly Kernel - With optimum values for hyperparameters (degree = 3, C = 0.1), our accuracy was **0.81954887**.

Figures 9, 10, 11 show the confusion matrices for classification using the above kernels.

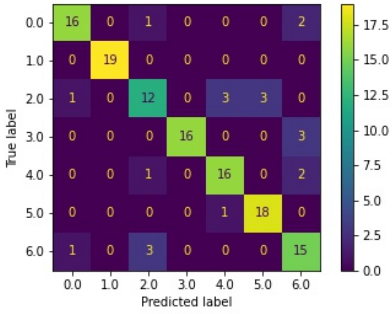

Figure 9. SVM:RBF Kernel Confusion Matrix

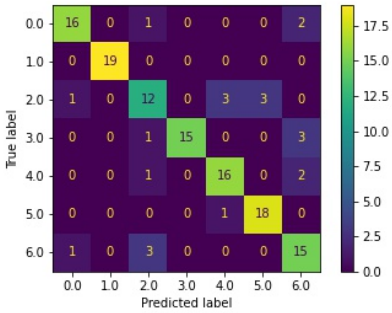

Figure 10. SVM:Linear Kernel Confusion Matrix

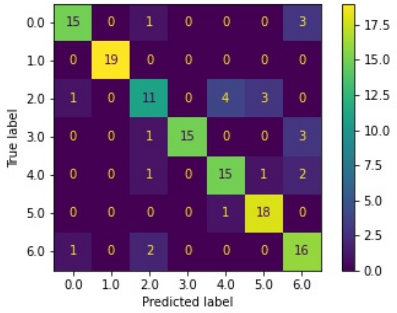

Figure 11. SVM:Poly Kernel Confusion Matrix

5) **Random Forest Classifier** Similar to the previous algorithms, we used the Scikit Learn implementation for classification using random forests [8]. We used randomized search to obtain the optimal values for number of trees in the forest, number of features to consider at every split, maximum number of levels in the tree, minimum number of samples required to split a node, minimum number of samples required at a leaf node, and the method of building trees (using bootstrap or not). Using these tuned hyperparameters, our random forest classifier had an accuracy of **0.86466165**. Figure 12 below shows the confusion matrix for the random forest algorithm.

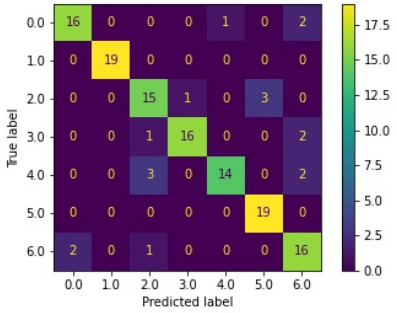

Figure 12. Random Forest Classifier Confusion Matrix

VI. Conclusion

Our algorithms performed fairly well, with the random forest classifier performing the best with an accuracy of 0.864. We saw that the Naive Bayes algorithm performed the worst with an accuracy of 0.766. In the future, we would like to see how these algorithms perform when classifying into more playlists, especially those that are less distinct. Furthermore, we would like to look into non-audio features, such as artist information, release year, etc. that may affect classification.

VII. Contributions

Sneha and Raagavi contributed equally to the research, gathering data, algorithm development, result analysis, and report of this project.

VIII. Code

https://drive.google.com/drive/folders/10Gn9I0DIIuPGLqPYUluywdaF014eoBPL?usp=sharing
Please use a Stanford account to access the code.

References

[1] B. Lansdown. Machine learning for music genre classification. https://www.researchgate.net/publication/337001430_Machine_Learning_for_Music_Genre_Classification, 09 2019.

[2] T. Dang and K. Shirai. Machine learning approaches for mood classification of songs toward music search engine. https://ieeexplore.ieee.org/document/5361715, 2009.

[3] M. Mckinney and J. Breebaart. Features for audio and music classification. https://www.researchgate.net/publication/2889273_Features_for_Audio_and_Music_Classification, 2003.

[4] M. Mandel and D. Ellis. Song-level features and support vector machines for music classification. https://www.ee.columbia.edu/~dpwe/pubs/ismir05-svm.pdf, 2005.

[5] Z. Fu, G. Lu, K. M. Ting, and D. Zhang. A survey of audio-based music classification and annotation. *IEEE Transactions on Multimedia*, 13(2):303–319, 2011.

[6] Spotipy library. https://github.com/plamere/spotipy. Accessed: 2017-08-01.

[7] Michael Waskom and the seaborn development team. mwaskom/seaborn. https://doi.org/10.5281/zenodo.592845, September 2020.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. https://scikit-learn.org/, 2011.