Solving Spotify Multiclass Genre Classification Problem

BEGINNER CLASSIFICATION DEEP LEARNING MACHINE LEARNING PANDAS RECOMMENDATION STATISTICS

Introduction

The music industry has become more popular, and how people listen to music is changing like wildfire. The development of music streaming services has increased the demand for automatic music categorization and recommendation systems. Spotify, one of the world's leading music streaming sites, has millions of subscribers and a massive song catalog. Yet, for customers to have a personalized music experience, Spotify must recommend tracks that fit their preferences. Spotify uses machine learning algorithms to guide and categorizes music based on the Genre.

Source: www.analyticsvidhya.com

This project will focus on the Spotify Multiclass Genre Classification problem, where we download the Dataset from Kaggle.

Goal – This project aims to develop a model that classifies the Genre that can accurately predict the Genre of a music track on spotify.

Learning Objectives

- To investigate the link between music genres on Spotify and their acoustic characteristics.
- To create a classification model based on auditory characteristics to predict the genre of a given song.
- To investigate the distribution of various spotify music genres in the dataset.
- To clean and preprocess data in order to prepare it for modeling.
- To assess the categorization model's performance and improve its accuracy.

This article was published as a part of the Data Science Blogathon.

Table of Contents

Prerequisites

Before we begin implementation, we must install and import some of the libraries. The libraries listed below are required:

Pandas: A library for data manipulation and analysis.

NumPy: A scientific computing package used for matrix computations.

Matplotlib: A plotting library for the Python programming language.

Seaborn: A data visualization library based on matplotlib.

Sklearn: A machine learning library for building models for classification

TensorFlow: A popular open-source library for building and training deep learning models.

To install these, we run this command.

```
!pip install pandas !pip install numpy !pip install matplotlib !pip install seaborn !pip install sklearn !pip install tensorflow
```

Project Pipeline

Data Preprocessing: Clean and preprocess the “genres_v2” dataset to prepare it for machine learning.

Feature Engineering: Extract meaningful features from the audio files that will help us train our model.

Model Selection: Evaluate several machine learning algorithms to find the best-performing model.

Model Training: Train the selected model on the preprocessed Dataset and evaluate its performance.

Model Deployment: Deploy the trained model in an online application that can recommend music tracks on Spotify based on the user’s preferences

So, let’s get started doing some code.

Project

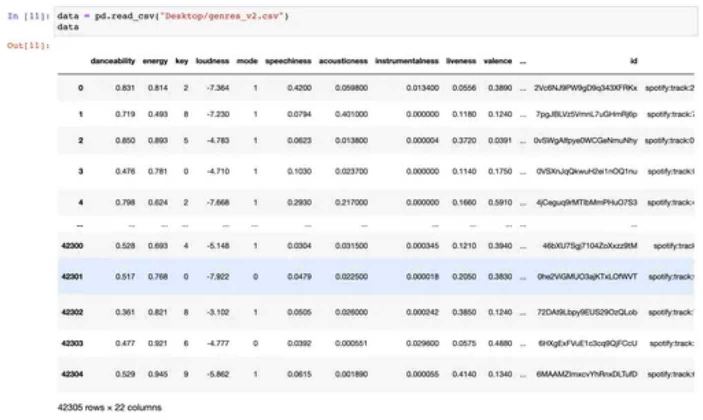
First, we need to download the data set. You can download the Dataset from Kaggle. We need to import the necessary libraries to perform our tasks.

```
import pandas as pd import seaborn as sns import matplotlib.pyplot as plt from sklearn.model_selection import train_test_split from sklearn.preprocessing import MinMaxScaler from sklearn import preprocessing from sklearn import metrics import numpy as np import tensorflow as tf from tensorflow import keras from sklearn.decomposition import PCA, KernelPCA, TruncatedSVD from sklearn.manifold import Isomap, TSNE, MDS import random from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis import warnings warnings.simplefilter("ignore")
```

Load the Dataset

We load the Dataset using pandas read_csv, and the data set contains 42305 rows and 22 columns and consists of 18000+ tracks.

```
data = pd.read_csv("Desktop/genres_v2.csv") data
```



```
In [11]: data = pd.read_csv("Desktop/genres_v2.csv")
         data
Out[11]:
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	id	
0	0.831	0.814	2	-7.364	1	0.4200	0.059800	0.013400	0.0556	0.3890	...	2Vc6NJ9PW9gD9q343XFRKx	spotify:track:2
1	0.719	0.493	8	-7.230	1	0.0794	0.401000	0.000000	0.1180	0.1240	...	7pgJBLVz5VmnL7uGHmRj6p	spotify:track:7
2	0.850	0.893	5	-4.783	1	0.0623	0.013800	0.000004	0.3720	0.0391	...	0vSWgAlfpye0WCGeNmuNhy	spotify:track:0
3	0.476	0.781	0	-4.710	1	0.1030	0.023700	0.000000	0.1140	0.1750	...	0VSXnJqQkwuH2ei1nOQ1nu	spotify:track:
4	0.798	0.624	2	-7.668	1	0.2930	0.217000	0.000000	0.1660	0.5910	...	4jCeguq9rMTlbMmPHuO7S3	spotify:track:
...	...	...	...	...	...	...	...	...	...	...	...	...	
42300	0.528	0.693	4	-5.148	1	0.0304	0.031500	0.000345	0.1210	0.3940	...	46bXU7Sgj7104ZoXxzz9tM	spotify:track
42301	0.517	0.768	0	-7.922	0	0.0479	0.022500	0.000018	0.2050	0.3830	...	0he2ViGMUO3ajKTxLOfWVT	spotify:track:
42302	0.361	0.821	8	-3.102	1	0.0505	0.026000	0.000242	0.3850	0.1240	...	72DAt9Lbpy9EUS29OzQLob	spotify:track:
42303	0.477	0.921	6	-4.777	0	0.0392	0.000551	0.029600	0.0575	0.4880	...	6HXgExFVuE1c3cq9QjFCcU	spotify:track:
42304	0.529	0.945	9	-5.862	1	0.0615	0.001890	0.000055	0.4140	0.1340	...	6MAAMZImxcvYhRnxDLTufD	spotify:track:

42305 rows × 22 columns

Exploring the Data

I use the 'iloc' method to select the rows and columns that form a data frame by their integer index positions. I am choosing the first 20 columns of the df.

```
data.iloc[:,:20] # this is for the first 20 columns data.iloc[:,20:] # this is for the 21st column data.info()
```

When you call data.info(), it will print the following information:

- The number of rows and columns in the data frame.
- The name of each column, its data type, and the number of non-null values in that column.
- The total number of non-null values in the data frame.
- The memory usage of the DataFrame.

```
data.nunique() # number of unique values in our data set.
```

Data Cleaning

Here, we want to clean our data by removing unnecessary columns that add no value to the prediction.

```
df = data.drop(["type","type","id","uri","track_href","analysis_url","song_name", "Unnamed: 0","title", "duration_ms", "time_signature"], axis =1) df
```

We have removed some columns that add no value to this particular problem and put axis = 1, where it drops the columns rather than rows. We are again calling the Data Frame to see the new Data Frame with helpful information.

The df. describe( ) method generates descriptive statistics of a Pandas Data Frame. It provides a summary of the central tendency and dispersion and the shape of the distribution of a dataset.

After running this command, you can see all the descriptive statistics of the Data Frame, like std, mean, median, percentile, min, and max.

```
df.describe()
```

To display a summary of a Pandas DataFrame or Series, use the df.info() function. It gives Dataset information like the number of rows and columns, the data types of each column, the number of non-null values in each column, and the utilization of the dataset's memory.

```
df.info() df["genre"].value_counts()
```

axe = sns.histplot(df["genre"]) generates a histogram of the distribution of values in a Pandas DataFrame named df's "genre" column. This code may be used to visualize the frequency of some Spotify genres in a music dataset.

```
ax = sns.histplot(df["genre"]) _ = plt.xticks(rotation=90) _ = plt.title("Genres")
```

The following code eliminates or deletes all rows in a Pandas DataFrame where the value in the "genre" column is equal to "Pop". The DataFrame's index is then reset to the range where it starts with 0. Lastly, it computes the correlation matrix of the DataFrame's remaining columns.

This code helps study a dataset by deleting unnecessary rows and finding correlations between the remaining variables.

```
df.drop(df.loc[df['genre']=="Pop"].index, inplace=True) df = df.reset_index(drop = True) df = df.corr()
```

The following code sns. heatmap (df, cmap='coolwarm, annot=True) plt. show() generates a heatmap depicting a Pandas DataFrame df's correlation matrix.

This code helps to find and display the strength and direction of correlations between variables in a dataset. The heatmap color coding makes it easy to see which pairs of variables are highly correlated and which are not.

```
sns.heatmap(df, cmap='coolwarm', annot=True ) plt.show()12python
```

The following code picks a subset of columns in a Pandas DataFrame df named x, which contains all columns from the DataFrame's beginning, including the "tempo" column. Then it chooses the DataFrame's "genre" as the target variable and assigns it to y.

The x variable represents a Pandas DataFrame with a subset of the original columns, and the y variable represents a Pandas Series with the "genre" column values.

The methods x.unique() and y.unique() retrieve the unique values in the x and y variables, respectively. These routines can be helpful for determining the number of unique values in the variables of a dataset.

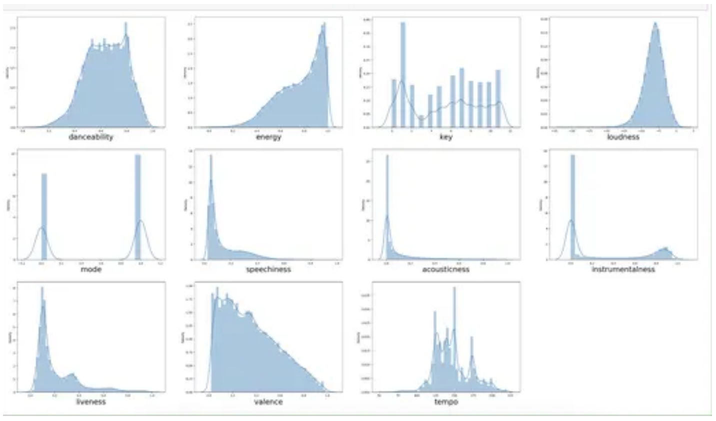
```
x = df.loc[:,:"tempo"] y = df["genre"] x y x.unique() y.unique()
```

I am not giving all the images. You can check the notebook down below.

The given code generates a grid of distribution plots that allow users to view the distribution of values over several columns in a dataset. Discovering patterns, trends, and outliers in the data by showing the distribution of values in each column. These are helpful and beneficial for exploratory data analysis and finding valuable and potential faults or inaccuracies in a dataset.

```
k=0  plt.figure(figsize  =  (40,30))  for  i  in  x.columns:  plt.subplot(4,4,  k  +  1)  sns.distplot(x[i])
plt.xlabel(i, fontsize=24) k +=1
```

Here, we are plotting for each x_columns, by using the for loop.



Model Training

The following code divides a dataset into training and testing subsets. It divides the input variables and target variables into 80% training and 20% testing groups at random. The descriptive statistics of the training data are then outputted to aid in data exploration and the identification of possible problems.

```
xtrain,  xtest,  ytrain,  ytest  =  train_test_split(x,  y,  test_size=  0.2,  random_state=42,  shuffle  =  True)
xtrain.columns xtrain.describe()
```

Here we are splitting the data into training and testing (size = 20%), and we are using the describe function to see the descriptive statistics.

The MinMaxScaler() function from the sklearn.preprocessing module is used to do feature scaling. It stores the training data's column names in the variable ol. The scaler object is then used to fit and convert the xtrain data while changing xtest data.

Lastly, the alternative xtrain and xtest data are converted into pandas DataFrames with the original column names (col). This is a critical step in the preprocessing and standardizing of data for machine learning models.

```
ol   =   xtrain.columns   scalerx   =   MinMaxScaler()   xtrain   =   scalerx.fit_transform(xtrain)   xtest   =
scalerx.transform(xtest) xtrain = pd.DataFrame(xtrain, columns = col) xtest = pd.DataFrame(xtest, columns =
col)
```

Here we use the MinMaxScaler, mainly for scaling and normalizing the data.

The following allows us to see the descriptive statistics of the xtrain and xtest.

```
xtrain.describe() xtest.describe()
```

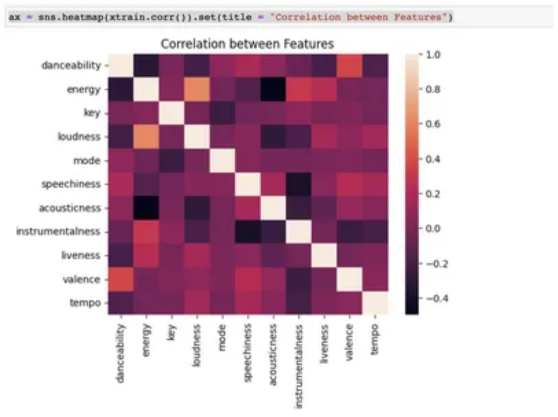
The LabelEncoder() function from the sklearn.preprocessing package is used to encode labels. It uses the fit transform() and transform() routines to encode the category target variables (ytrain and ytest) into numerical values.

The training and testing data for input (x) and target (y) variables are then concatenated. The numerical labels are then inversely transformed into their original category values (y train, y test, and y org).

Next, we use the np.unique() method, which returns the individual categories in the training data.

Lastly, using the seaborn library generates a heatmap graphic to illustrate the relationship between the input characteristics. This is a critical stage when we examine and prepare data for machine-learning models.

```
le = preprocessing.LabelEncoder() ytrain = le.fit_transform(ytrain) ytest = le.transform(ytest) x =
pd.concat([xtrain, xtest], axis = 0) y = pd.concat([pd.DataFrame(ytrain), pd.DataFrame(ytest)], axis = 0)
y_train = le.inverse_transform(ytrain) y_test = le.inverse_transform(ytest) y_org =
pd.concat([pd.DataFrame(y_train), pd.DataFrame(y_test)], axis = 0) np.unique(y_train) csvax =
sns.heatmap(xtrain.corr()).set(title = "Correlation between Features")
```

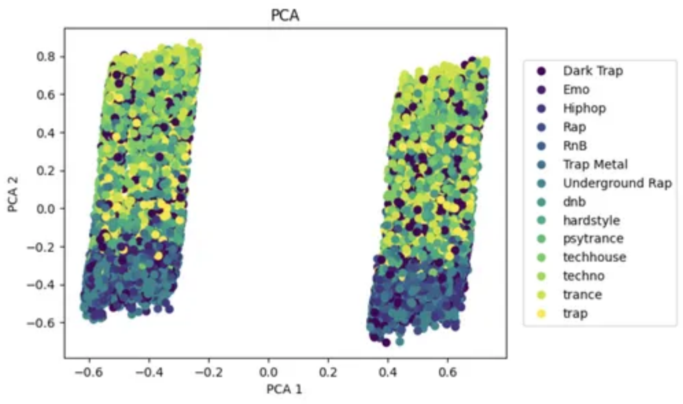


```
ax = sns.heatmap(xtrain.corr()).set(title = "Correlation between Features")
```

PCA is a popular dimensionality reduction approach that may assist in decreasing the complexity of large datasets and increasing the performance of machine learning models.

With input data x, the algorithm uses PCA to minimize the number of features to two parts that explain the variation. The reduced Dataset is shown on a 2D scatter plot, with dots colored by class labels in y. This aids in visualizing the dividing of some classes in the reduced feature space.

```
pca = PCA(n_components=2) x_pca = pca.fit_transform(x, y) plot_pca = plt.scatter(x_pca[:,0], x_pca[:,1], c=y)
handles, labels = plot_pca.legend_elements() lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center
right', bbox_to_anchor=(1.4, 0.5)) plt.xlabel("PCA 1") plt.ylabel("PCA 2") _ = plt.title("PCA")
```

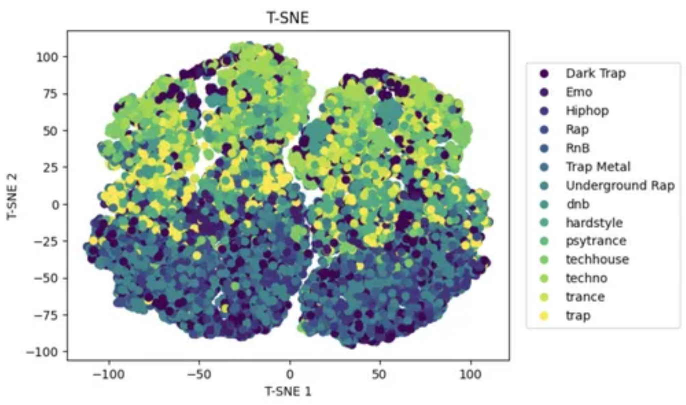


t-SNE is a popular nonlinear dimensionality reduction approach that may assist in decreasing the complexity of large datasets and improve the performance of machine learning models.

Using t-Distributed Stochastic Neighbor Embedding (t-SNE) on the input data x reduces the number of features in the high-dimensional space to 2D while maintaining similarity between Data points.

A 2D scatter plot shows the reduced Dataset, with dots colored according to their y-class labels. It helps visualize the division of some classes in the reduced feature space.

```
tsne = TSNE(n_components=2) x_tsne = tsne.fit_transform(x, y) plot_tsne = plt.scatter(x_tsne[:,0], x_tsne[:,1], c=y) handles, labels = plot_tsne.legend_elements() lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5)) plt.xlabel("T-SNE 1") plt.ylabel("T-SNE 2") _ = plt.title("T-SNE")
```

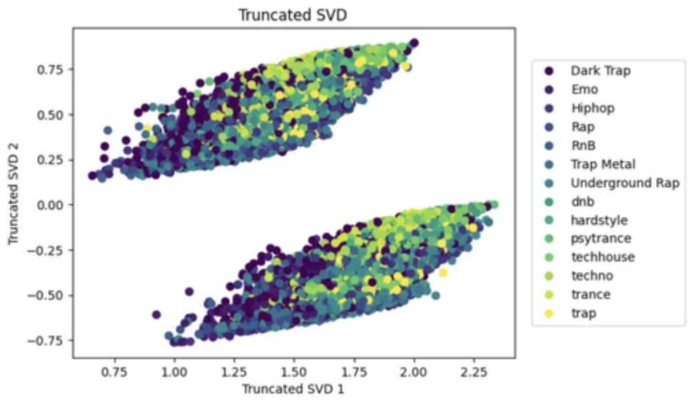


SVD is a popular dimensionality reduction approach that may assist in decreasing the complexity of large datasets and increasing the performance of machine learning models.

The following code applies Singular Value Decomposition (SVD) on the input data x with n components=2, reducing the number of input features to two that explain the most variance in the data. The reduced Dataset is then shown on a 2D scatter plot, with the dots colored based on their y-class labels.

This facilitates visualizing the division of multiple classes in the reduced feature space, and the scatter plot is made with the matplotlib tool.

```
svd = TruncatedSVD(n_components=2) x_svd = svd.fit_transform(x, y) plot_svd = plt.scatter(x_svd[:,0], x_svd[:,1], c=y) handles, labels = plot_svd.legend_elements() lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5)) plt.xlabel("Truncated SVD 1") plt.ylabel("Truncated SVD 2") _ = plt.title("Truncated SVD")
```

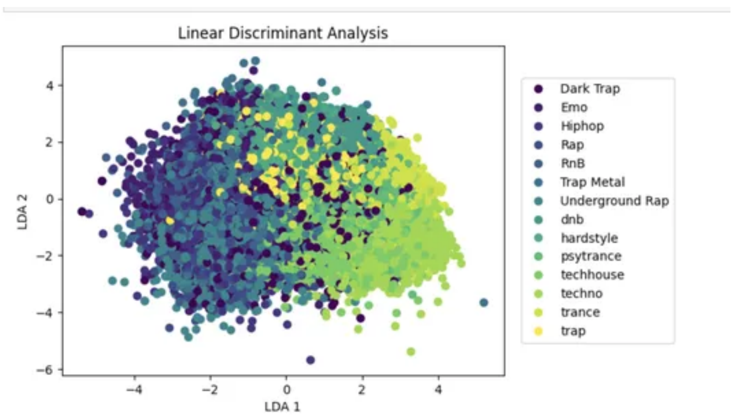


LDA is a popular dimensionality reduction approach that can increase machine learning model performance by decreasing the influence of irrelevant information.

The following code does Linear Discriminant Analysis (LDA) on the input data x with n components=2, which reduces the number of input features to two linear discriminants that maximize the division between the different classes in the data.

The reduced Dataset is then shown on a 2D scatter plot, with the dots colored based on their y-class labels. This aids in visualizing the division of some classes in the reduced feature space.

```
lda = LinearDiscriminantAnalysis(n_components=2) x_lda = lda.fit_transform(x, y.values.ravel()) plot_lda =
plt.scatter(x_lda[:,0], x_lda[:,1], c=y) handles, labels = plot_lda.legend_elements() lg =
plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5)) plt.xlabel("LDA
1") plt.ylabel("LDA 2") _ = plt.title("Linear Discriminant Analysis")
```



The following code substitutes some values in a Data Frame column called 'genre' with the new deal 'Rap.' Specifically, it replaces the values "Trap Metal," "Underground Rap," "Emo," "RnB," etc., with "Rap." This is useful for grouping genres under a single name for analysis or modeling.

```
df = df.replace("Trap Metal", "Rap") df = df.replace("Underground Rap", "Rap") df = df.replace("Emo", "Rap")
df = df.replace("RnB", "Rap") df = df.replace("Hiphop", "Rap") df = df.replace("Dark Trap", "Rap")
```

The code below generates a histogram using the seaborn library to illustrate the variable "genre" distribution in the input dataset df. The figure has been rotated by 80 degrees to improve the visibility of the x-axis labels. "Genres" is a title.

```
ax = sns.histplot(df["genre"]) _ = plt.xticks(rotation=80) _ = plt.title("Genres")
```

The provided code removes the rows from the Data Frame. Specifically, it eliminates rows with a frequency of 0.85 where the genre column value is "Rap," using a random number generator.

The rows to be discarded are saved in a list of rows dropped before being removed from the Data Frame using the drop function. The code then prints a histogram of the remaining genre values with the seaborn plot function and changes the title and rotation of the x-axis labels with matplotlib's title and xticks methods.

```
rows_drop = [] for i in range(len(df)): if df.iloc[i]['genre'] == 'Rap': if random.random()<0.85:
rows_drop.append(i) df.drop(index = rows_drop, inplace=True) ax = sns.histplot(df["genre"]) _ =
plt.xticks(rotation=80) _ = plt.title("Genres")
```

The code provided preprocesses the data. The first step is to divide the input data into training and testing sets using the Sklearn library's train test split function.

It then adjusts the numerical characteristics in the supplied data using the MinMaxScaler function from the same package. The code encodes the category target variable using the preprocessing module's LabelEncoder function.

As a result, the training and testing sets are preprocessed previously are merged into a single dataset that the machine learning algorithm can process.

```
x = df.loc[:,:"tempo"] y = df["genre"] xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size= 0.2, random_state=42, shuffle = True) col = xtrain.columns scalerx = MinMaxScaler() xtrain = scalerx.fit_transform(xtrain) xtest = scalerx.transform(xtest) xtrain = pd.DataFrame(xtrain, columns = col) xtest = pd.DataFrame(xtest, columns = col) le = preprocessing.LabelEncoder() ytrain = le.fit_transform(ytrain) ytest = le.transform(ytest) x = pd.concat([xtrain, xtest], axis = 0) y = pd.concat([pd.DataFrame(ytrain), pd.DataFrame(ytest)], axis = 0) y_train = le.inverse_transform(ytrain) y_test = le.inverse_transform(ytest) y_org = pd.concat([pd.DataFrame(y_train), pd.DataFrame(y_test)], axis = 0)
```

This code creates two early stopping callbacks for model training, one based on validation loss and the other on validation accuracy. Keras' Sequential API makes a NN model with various connected layers using the ReLU activation function, batch normalization, and dropout regularisation. The summary of the model is printed on the console.

The final output layer outputs class probabilities using the softmax activation function. The summary of the model is printed on the console.

```
early_stopping1 = keras.callbacks.EarlyStopping(monitor = "val_loss", patience = 10, restore_best_weights = True) early_stopping2 = keras.callbacks.EarlyStopping(monitor = "val_accuracy", patience = 10, restore_best_weights = True) model = keras.Sequential([ keras.layers.Input(name = "input", shape = (xtrain.shape[1])), keras.layers.Dense(256, activation = "relu"), keras.layers.BatchNormalization(), keras.layers.Dropout(0.2), keras.layers.Dense(128, activation = "relu"), keras.layers.Dense(128, activation = "relu"), keras.layers.BatchNormalization(), keras.layers.Dropout(0.2), keras.layers.Dense(64, activation = "relu"), keras.layers.Dense(max(ytrain)+1, activation = "softmax") ]) model.summary()
```

The following code block uses Keras to compile and train a neural network model. The model is a sequential model with multiple dense layers with relu activation function, batch normalization, and dropout regularisation. "sparse categorical cross entropy" is the loss function utilized. At the same time, "Adam" is the optimizer. The model is trained for 100 epochs, with callbacks that end early based on validation loss and accuracy.

```
model.compile(optimizer = keras.optimizers.Adam(), loss = "sparse_categorical_crossentropy", metrics = ["accuracy"]) model_history = model.fit(xtrain, ytrain, epochs = 100, verbose = 1, batch_size = 128, validation_data = (xtest, ytest), callbacks = [early_stopping1, early_stopping2])
```

The training data is sent as xtrain and ytrain, whereas the validation data is sent as xtest and ytest. The training history of the model is saved in the model history variable.

```
print(model.evaluate(xtrain, ytrain)) print(model.evaluate(xtest, ytest))
```

The following code generates a plot using matplotlib; on the x_axis, we have the epoch, and on the y_axis, we have the sparse Categorical Cross Entropy.

```
plt.plot(model_history.history["loss"])    plt.plot(model_history.history["val_loss"])    plt.legend(["loss", "validation loss"], loc ="upper right") plt.title("Train and Validation Loss") plt.xlabel("epoch") plt.ylabel("Sparse Categorical Cross Entropy") plt.show()
```

Same as above, but here we are plotting between the epoch and the accuracy.

```
plt.plot(model_history.history["accuracy"])                    plt.plot(model_history.history["val_accuracy"]) plt.legend(["accuracy", "validation accuracy"], loc ="upper right") plt.title("Train and Validation Accuracy") plt.xlabel("epoch") plt.ylabel("Accuracy") plt.show()
```

The following code ypred, which predicts the xtest.

```
ypred = model.predict(xtest).argmax(axis=1)
```

The following code evaluates the classification metrics on the test and ypred, where we can see the precision, recall, and F1score. Based on the values, we can proceed with our model.

```
cf_matrix = metrics.confusion_matrix(ytest, ypred) _ = sns.heatmap(cf_matrix, fmt=".0f", annot=True) _ = plt.title("Confusion Matrix")
```

Finally, we do the model Evaluation.

Model Evaluation

The following code evaluates the classification metrics on the test and ypred, where we can the precision, recall, F1score. Based on the values we can proceed with our model.

```
print(metrics.classification_report(ytest, ypred))
```

Conclusion

In conclusion, we could categorize Spotify music genres with an accuracy of 88% using the analysis and modeling done in this study. Given the complexity and subjectivity in defining music genres, this is a reasonable level of accuracy. Yet, there is always an opportunity for improvement, and our analysis has a few limitations.

One disadvantage is the need for more diversity in our Dataset, primarily rap and hip-hop music on Spotify. This influenced our research and modeling in favor of specific genres. We must incorporate a wider variety of music genres into the Dataset to improve our model.

Another restriction is the likelihood of human mistakes in classifying the data, which might have resulted in genre categorization discrepancies. We may utilize more sophisticated approaches, such as deep learning models, to automatically label music based on auditory attributes to address this.

Our analysis and modeling give a solid foundation for categorizing Spotify music genres, but more study and improvements are required to increase the model's accuracy and resilience.

Key Takeaways

- Auditory characteristics such as pace, danceability, energy, and valence can be distinguished across Spotify music genres.
- Data cleaning and preprocessing are critical processes in preparing data for modeling and can significantly influence model performance.
- Early stopping approaches, such as monitoring validation loss and accuracy, can help to prevent model overfitting.
- Increase the dataset size, add features, and experiment with alternative methods and hyperparameters to enhance the classification model's performance.

Article Url - https://www.analyticsvidhya.com/blog/2023/03/solving-spotify-multiclass-genre-classification-problem/

Tarak Ram