SPOTIFY SEQUENTIAL SKIP PREDICTION

Project repo: Spotify Skip Action Prediction

BY : ENGI *NORHAN SWAR*

Email: eng-nourhan.sowar1722@alexu.edu.eg

ABSTRACT

Music consumption habits have changed dramatically in the past decade with the rise of streaming services such as Spotify, Apple Music, and Tidal.
Today, these services are ubiquitous. This music providers are also incentivized to provide songs that users like in order to create a better user experience and increase time spent on their platform.
Thus, an important challenge to music streaming is determining what kind of music the user would like to hear. The skip button is one feature instance on these music services that play a large role in the user's experience and helps identify what a user likes, as with this button, users are free to abandon songs as they choose.

Thus, in this project, we will build a machine learning model that will predict if a user skips a song or not give information about the user's previous actions during a listening session along with
acoustic features of the previous songs.

INTRODUCTION

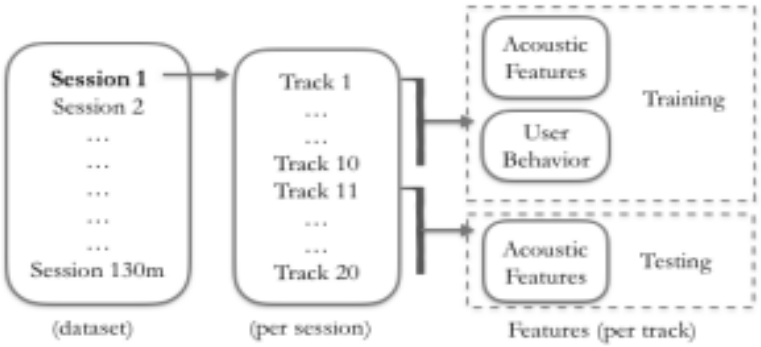
As streaming services like Spotify become the go-to providers for music, it is increasingly important that these platforms can recommend the right music to their users. Machine learning has long been leveraged to build recommender systems in order to recommend music. This can be seen through improvements made to music streaming platforms, such as customized playlists and suggesting a user new releases that they might like. However, there has not been much research done on how a user interacts with music sequentially, over the course of one listening session.

Skipping behavior serves as a powerful signal about what the user does and doesn't like. For instance, in the afternoon, a user might be looking for classical study music, and thus skipping hip-hop. Later that evening, the user might skip classical for hip-hop. Being able to use skip behavior in the context of an entire listening session is key to recommending relevant content, and yet there has been a lack of research into this specific recommendation task.

The focus of our project is to build a sequential skip prediction model to predict if a user will skip over a track based on the user's interactions with previous songs and the song's musical qualities in an individual music listening session.
The input will be an embedding of audio features and user behavior and the output will be predicting whether a track is 'skipped' or 'not skipped'. We have implemented simpler sequential-based models like GBTs and Decision Trees.

DATASET STRUCTURE

I have used the Spotify Sequential Skip Prediction dataset for this project, which consists of roughly 130 million listening sessions. Each session has at most 20 music tracks. All the user behavior features and track ids are provided for the first half of the session, but only the track ids are provided for the second half. The track ids are associated with the track's acoustic features that can be extracted via the Spotify API. The user behavior features consist of actions like whether the user paused, an hour of day, etc. and acoustic features consist of data like danceability, tempo, etc.



A schema for the log is provided in below table. Each session is defined to be a period of listening with no more than 60 seconds of inactivity between consecutive tracks. Additionally, for this dataset, we set a cut-off of at most 20 tracks per session as part of our privacy strategy. Sessions included in the dataset are sampled uniformly at random from eligible listening sessions on the contexts included in the dataset over an 8-week period. We exclude sessions that include tracks that did not meet a minimum popularity threshold. The logs, therefore, contain a mix of listening sessions based on users' personally curated collections; expertly curated playlists; contextual, but non-personalized recommendations; and finally, personalized recommendations.

Column name	Column description
session id	unique session identifier
session position	position of track within session
session length	length of session
track id	unique track identifier
skip 1	whether the track was only played very briefly
skip 2	whether the track was only played briefly
skip 3	whether most of the track was played
not skipped	whether the track was played in its entirety
context switch	whether the user changed context between the previous row and the current row
no pause	whether there was no pause between playback of the previous track and current track
short pause	whether there was a short pause between playback of the previous track and current track
long pause	whether there was a long pause between playback of the previous track and current track
num seekfwd	the number of times the user scrubbed forward during playback
num seekbk	the number of times the user scrubbed backward during playback
shuffle	whether the track was played with shuffle mode activated
hour of day	hour of day (integers between 0 and 23)
date	date in YYYY-MM-DD format
premium	whether the user was on premium or not
context type	what type of context the playback occurred within
reason start	cause of this track play starting
reason end	cause of this track play ending
uniform random	whether shuffle would be uniformly random for this session

DATA PREPROCESSING

In order to pre-process the data for training, I merged user behavior and acoustic features using track ids for each track in the first half of a listening session. We then pre-processed categorical features into one-hot representations and normalized them (using either z-score and min/max).

This process created an input track embedding for our model. For the tracks used in the testing phase (tracks from the second half of the session), we only used the pre-processed acoustic features embeddings. We chose to use the 'skip 2' feature as our output label for whether a track was skipped/not skipped. This is because 'skip 2' indicates whether a user skipped a track early on (i.e. a third of the way through a track), whereas other the skip features 'skip 1' and 'skip 4' respectively indicate whether a user skipped the track almost as soon as it played or near the end of the track. 'Skip 2' is more ideal in indicating the user's actions of listening to a song for a little bit and figuring out if they like/dislike that song (since many people tend to skip songs they like even halfway through).

DATA WRANGLING

- **HANDLING CATEGORICAL DATA**

I have used One-Hot Encoding and ordinal Encoding to handle categorical data. One-Hot Encoding is the most common, correct way to deal with non-ordinal categorical data. It consists of creating an additional feature for each group of the categorical feature and mark each observation belonging (Value=1) or not (Value=0) to that group.

- **HANDLING MISSING DATA**

I have used Imputation Techniques to handle missing values by replacing it with mean values. The Simple Imputer class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located. This class also allows for different missing values encodings.

- **NORMALIZING AND SCALING FEATURES**

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.
Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

MODEL ARCHITECTURE

- **GRADIENT BOOSTING**

Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modeling problems.
Ensembles are constructed from decision tree models. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. This is a type of ensemble machine learning model referred to as boosting.

Extreme Gradient Boosting, or XGBoost for short, is an efficient open-source implementation of the gradient boosting algorithm. As such, XGBoost is an algorithm, an open-source project, and a Python library.

- **DECISION TREE CLASSIFICATION**

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

MODEL DEPLOYMENT

Deployment is the method by which you integrate a machine learning model into an existing production environment to make practical business decisions based on data. It is one of the last stages in the machine learning life cycle and can be one of the most cumbersome.

- USING FLASK

Flask is a web application framework written in python, in simple terms it helps end users interact with your python code (in this case our ML models) directly from their web browser without needing any libraries,code files, etc.

Flask enables you to create web applications very easily, hence enabling you to focus your energy more on other important parts of a ML lifecycle like EDA, feature engineering, etc. Here in this blog I will give you a walkthrough on how to build a simple web application out of your ML Model and deploying it eventually.

FUTURE IMPORVEMENT

Given more time, I want to explore the transformer architecture more. Based on the transformers implemented successfully in NLP applications and the performance of our feature-forcing transformer, I could combine aspects of the traditional NLP transformer with the feature-forcing transformer in order to create a better model. I could dynamically append the output prediction for each track from the decoder with the track's audio features and injected these concatenated embeddings into our decoder for prediction.

This architecture would perhaps better follow the transformers that are known to work well in NLP, as well as would better represent our model by considering the audio features that are inputting into the decoder. Finally, exploring variable length sessions and improved feature analysis would help us better understand and improve our model