****

**IS424 Data Mining and Business Analytics**

**Sequential Skip Prediction Using Supervised Learning Techniques**

**Prepared for:**

Professor Wang Zhaoxia

**G2T3**

Sean Chai Shong Hee

Siew Wei Lun

Jazreel Toh Wei Wen

Lee Cheng Leng

Ngoh Yi Long

22 November 2019

Table of Contents

[Abstract 3](#_Toc25352354)

[1. Introduction 4](#_Toc25352355)

[2. Motivation 4](#_Toc25352356)

[3. Literature Review 4](#_Toc25352357)

[4. Dataset 4](#_Toc25352358)

[5. Methodology 5](#_Toc25352359)

[5.1 Exploratory Data Analysis (EDA) 5](#_Toc25352360)

[5.2 Data Pre-processing 6](#_Toc25352361)

[5.2.1 Data Integration 6](#_Toc25352362)

[5.2.2 Data Cleaning 7](#_Toc25352363)

[5.2.3 Feature Selection 8](#_Toc25352364)

[5.2.3.1 Correlation Matrix 8](#_Toc25352365)

[5.2.3.2 Principal-Component Analysis 8](#_Toc25352366)

[5.3 Algorithms and Implementations 9](#_Toc25352367)

[5.3.1 Naive Bayes 9](#_Toc25352368)

[5.3.2 Logistic Regression 10](#_Toc25352369)

[5.3.3 k-Nearest Neighbours (kNN) 11](#_Toc25352370)

[5.3.4 Gradient Boosted Decision Trees (XGBoost) 11](#_Toc25352371)

[5.3.6 Long-Short Term Memory Model 13](#_Toc25352372)

[6. Results and Discussion 14](#_Toc25352373)

[7. Conclusion and Future Work 14](#_Toc25352374)

[8. References 15](#_Toc25352375)

### 

### Abstract

A central challenge for Spotify is to recommend the right music to each of its users, in order to retain its competitive edge. While most recommender systems study similarities in user characteristics and song metadata, few have studied the sequential user interactions with tracks in their listening session.

This report provides an outline of various algorithms used to predict the binary skip variable (skip\_2) of a track in the Spotify Sequential Skip Prediction Challenge. The models were trained on one day’s worth of user sessions, which resulted in 178,342 unique listening sessions to be analysed. From there, the sessions were split into the first half and second half to be used as the training dataset and testing dataset respectively.

The algorithms used include Naive Bayes, Logistic Regression, k-Nearest Neighbours, XGBoost and Long-Short Term Memory Model. The training methodology employed was to develop an initial model, followed by conducting feature selection and hyperparameter tuning for selected models. After which, the tuned models will be run to compare its accuracy against the initial model.

The resultant highest accuracy was attained by the XGBoost model, with an accuracy of 86.55%.

**Keywords – Sequential Prediction, Music, Recommender Systems**

### 1. Introduction

Traditionally, one had to rely on word-of-mouth to get song recommendations from their friends. In recent times, music streaming platforms have evolved to provide song recommendations for its users based on the songs they already listen to. Today, this standard has become an expectation of music consumers using these platforms. A central challenge for Spotify is to gain differential advantage over other music streaming platforms. To do so, Spotify needs to improve its music recommendations to consumers as recommending the right music to each user will improve user engagement and customer retention rates.

### 2. Motivation

Currently, Spotify uses collaborative filtering and natural language processing to recommend songs.

Little research has been done on how users’ sequential interactions with tracks can be used to improve music recommendations made to them. Our group feels that more focus can be put into this area of research. Discovery of patterns between users’ sequential interactions with tracks might be helpful in improving current music recommendation systems. Hence, we would like to utilise the skip behaviour of users as a measure to improve Spotify’s current music recommendation system.

### 3. Literature Review

Based on prior research, we found that most of the top solutions for the Spotify Sequential Skip Prediction Challenge featured the use of a Long-Short Term Memory (LSTM) model. LSTM models are widely used for problems with a sequential nature as it considers a time dependency. They also alleviate the vanishing gradient problem of tradition Recurrent Neural Network (RNN) models.

Other solutions featured the use of gradient boosted decision trees to predict a label for each track position. This involved multiple independent decision trees, from which majority decision is used to prevent overfitting of the model. This model was also a favourite as it is optimised for computational speed and performance.

### 4. Dataset

The training dataset downloaded from AIcrowd was 56GB, while the testing dataset was 14GB. The dataset consisting of track features was an additional 1.2GB. As the datasets provided by the challenge were too large to be used on the Google Colab GPU server without it crashing, we had to extract a subset of the data for processing.

We compiled 1 day’s worth of data from the provided training dataset. The final training dataset we used had 2,990,609 rows. We used the test dataset for the corresponding date and used the 1.2GB dataset that consisted of track features.

Studying the train and test dataset in closer detail, each row represents a track within a unique listening session. The training dataset is derived from the first half of a user’s listening session, and the test set to be predicted is based on the second half of the session.

Meanwhile, the dataset consisting of track features provides acoustic features of each track, derived from the Spotify API. This includes features like key, tempo, loudness and acousticness. This provides us with more accurate features of which we can use to train our model.

The ground truth variable in this challenge is the skip\_2 variable, which is a variable which indicates if the song was played briefly or not.

### 5. Methodology

#### 5.1 Exploratory Data Analysis (EDA)

To explore the data, we first obtained summary statistics and checked for missing values in the training dataset. We found that there were no missing values in the training dataset and the track features dataset. However, we noticed that the track\_id was a string that would not be easily identifiable, and there were columns with boolean values and categorical variables as well.

As such, we set out to encode these values before continuing with the EDA in order to ensure that the values were all numeric, which would allow for easier comparison, processing and model fitting thereafter.

To explore the data further, we plotted some bar charts for the discrete variables and histograms for the continuous variables (as shown in Fig. 1). This also helped us to find out what data processing steps we had to undertake in order to produce a more suitable dataset on which to run our models.

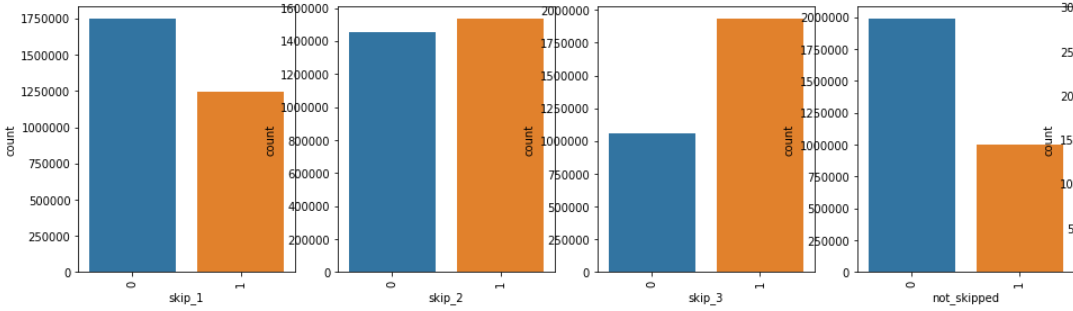


Fig. 1: Distributions for Skip Variables

We first studied the distribution of the skip variables and found that the values were most equally distributed for the skip\_2 variable. This is also the ground truth variable for the dataset as specified for the competition.

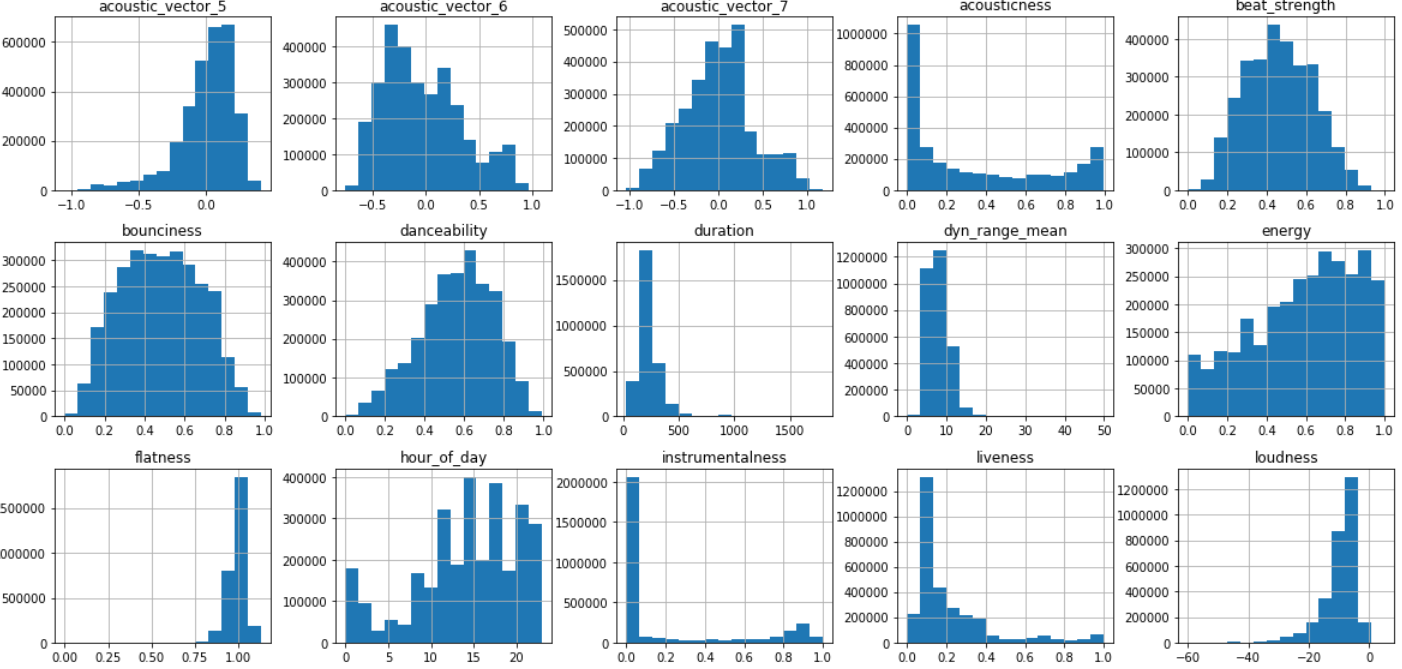


Fig. 2: Distributions for Continuous Variables

Upon studying the distribution of the continuous features in the dataset, we noticed that some of them were left or right skewed (as shown in Fig. 2). We decided to conduct variable transformation to transform these skewed variables to symmetric ones. We also noticed that the scales for each variable were different.

#### 5.2 Data Pre-processing

Before we continued with training our model, we first had to undertake some data pre-processing steps in order to prepare our data in the correct format for our models.

##### 5.2.1 Data Integration

As the training dataset, test dataset and track features were provided in different files, we had to first merge the track features to the training dataset. This allowed us to obtain a more complete representation of the tracks in the training dataset.

##### 5.2.2 Data Cleaning

To tackle the issue of skewed variable distribution mentioned in Section 5.1, we conducted transformation and standardisation on the continuous variables of the dataset. Transformation would ensure that the variables follow a normal distribution, allowing the algorithms to converge faster. Furthermore, we conducted standardisation to ensure that differences in scale would not assign unintended weights to variables that have larger scales (as shown in Fig. 3 and Fig. 4)

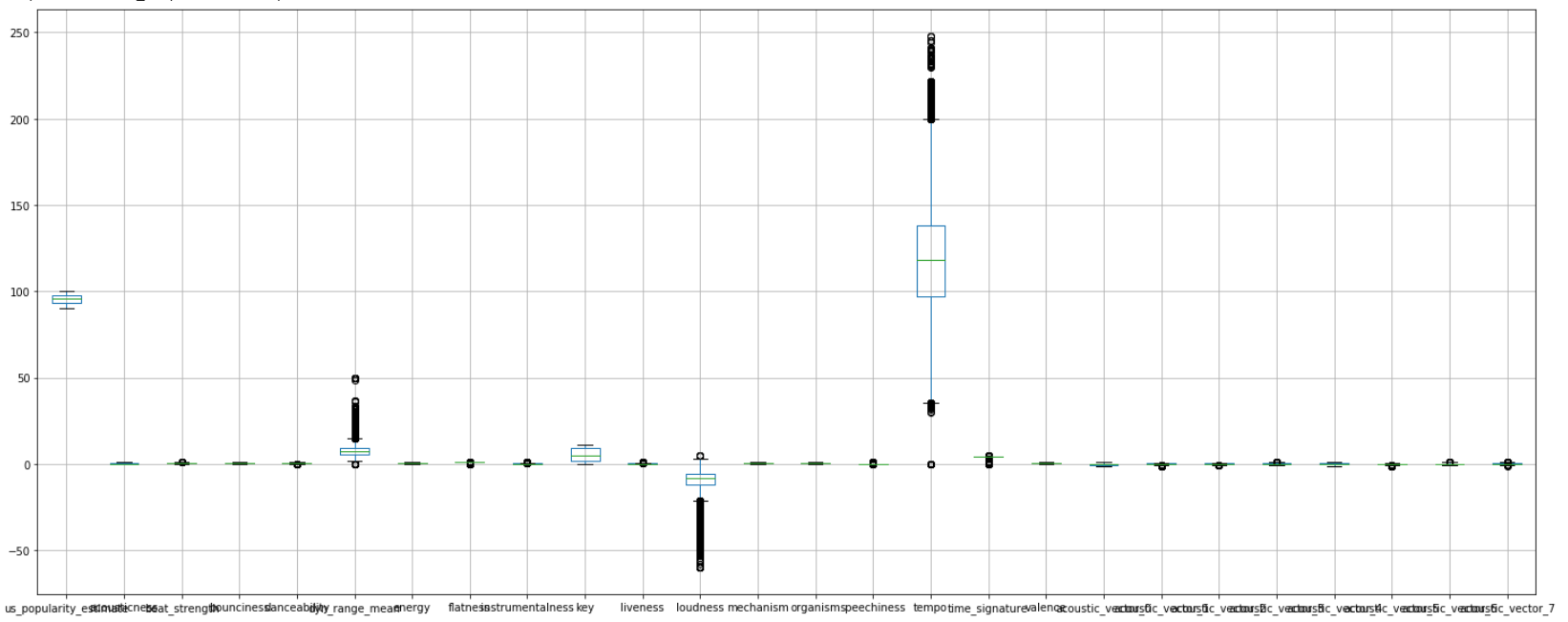


Fig. 3: Distribution of Continuous Variables before Normalisation and Standardisation

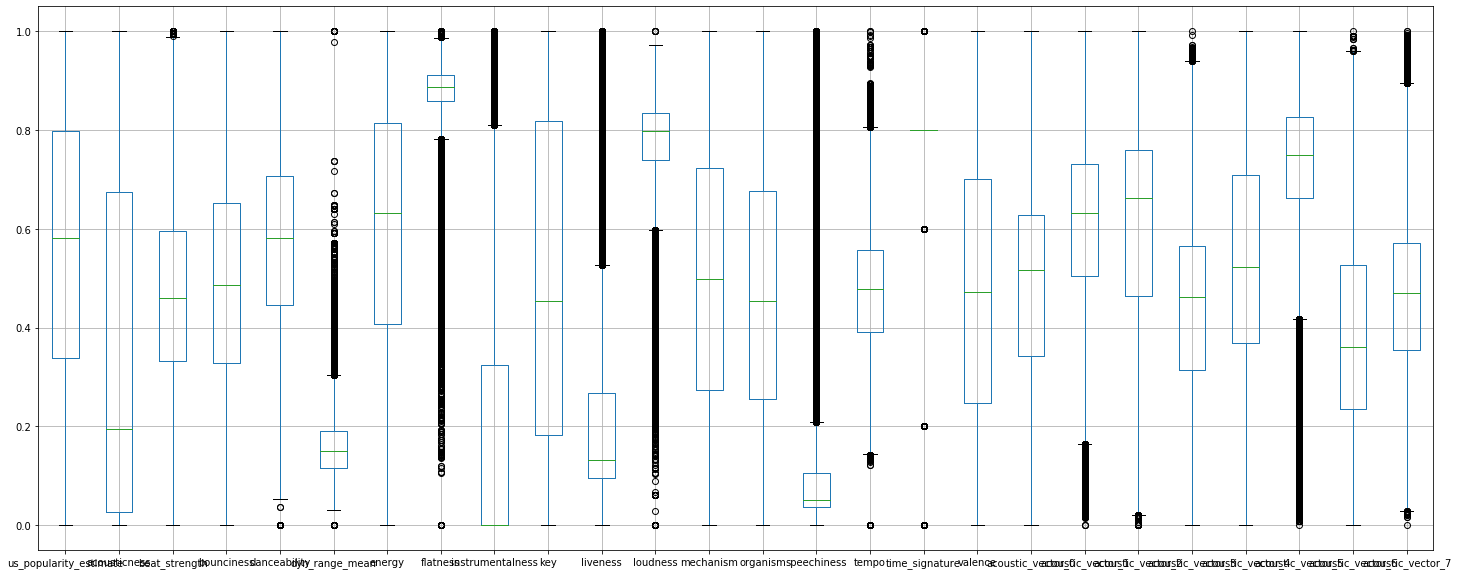


Fig. 4: Distribution of Continuous Variables after Normalisation and Standardisation

##### 5.2.3 Feature Selection

Feature selection was performed to reduce the effects of the curse of dimensionality, which is highly applicable to our dataset which has 46 features [1]. To do so, we studied the correlation between the independent features and conducted principal-component analysis. Feature selection was also conducted for each model, as the feature importance would vary according to the model used.

###### 5.2.3.1 Correlation Matrix

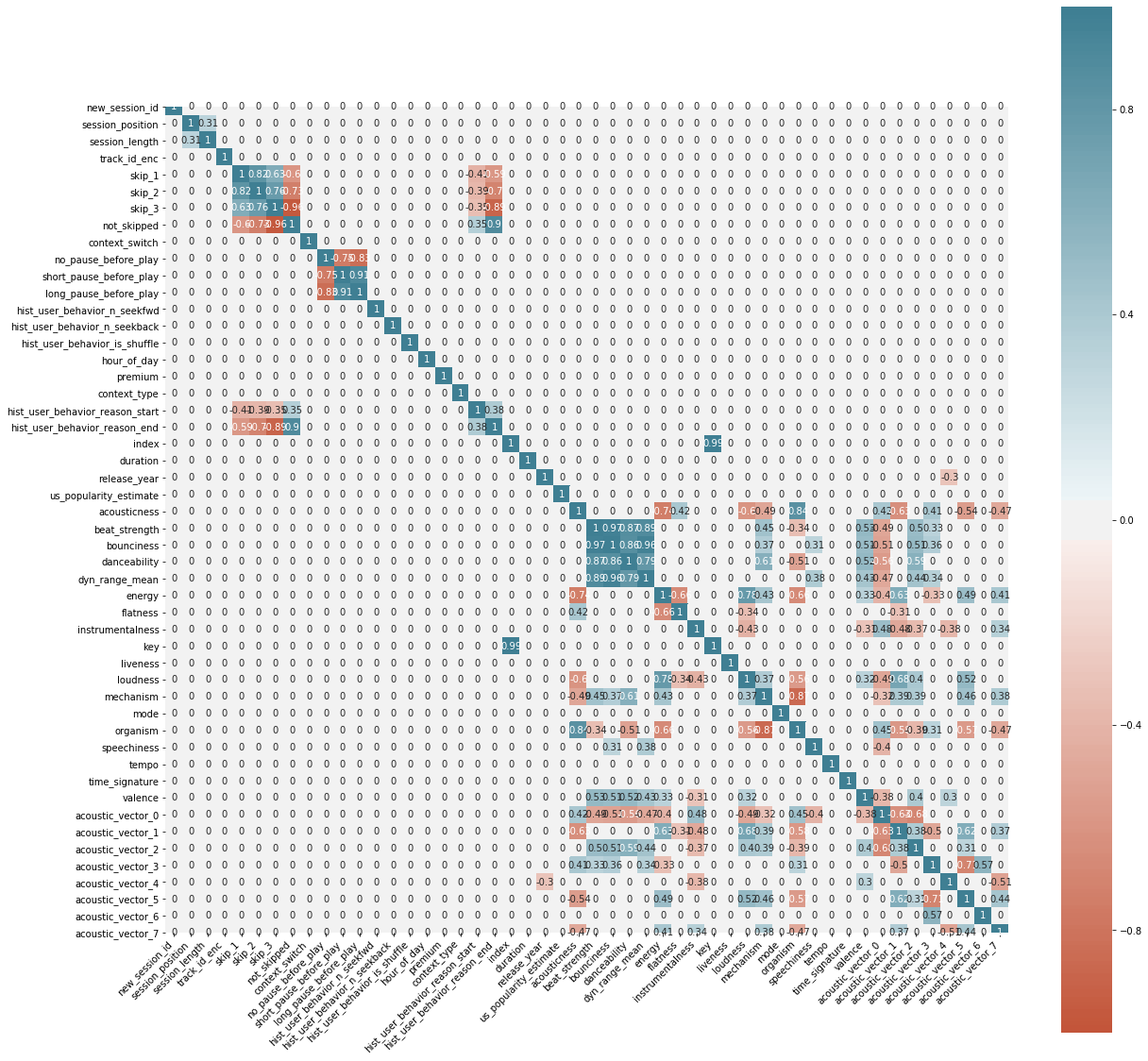


Fig. 5: Correlation Heatmap of all Variables

We decided to study the relationships between the variables in the dataset, in order to identify independent variables that are highly correlated. We would then have to eliminate these correlated to reduce multicollinearity, as this may hide significant variables, or lead to variable inflation.

From the correlation matrix in Fig. 5, we noticed that the skip variables were highly correlated. As such, we decided to drop the skip variables other than the skip\_2 column, as the other skip variables were redundant and only added to the data noise.

We also decided to drop the date column, as we had only utilised the training dataset from one date to reduce the computational power required to run our models. As such, the column would be irrelevant to our predictions.

###### 5.2.3.2 Principal-Component Analysis

As the number of features in our training dataset was around 47 columns, we set to reduce the dimensionality of the data by conducting feature extraction. This would allow us to reduce the execution time for our algorithms in the future. We first tried using Principal Component Analysis (PCA) in order to generate main dimensions of variation from the large number of features.

The top three PCA features provided an explained variance of 41.79%, while the top five PCA features gave us an explained variance of 57.98%. We decided to use a different approach, which is to use a combination of PCA components which provided an explained variance of 95% in order to run some tests to see if we should use the PCA components or not.

Using a logistic regression model on the PCA components, we achieved an accuracy score of 51.2%, which was not as accurate as our baseline naïve bayes and logistic regression model. As such, we decided not to use the features generated by the PCA.

#### 5.3 Algorithms and Implementations

##### 5.3.1 Naïve Bayes

Naïve Bayes is a linear classifier that assumes all features are independent and equal. Of all Naïve Bayes models, we chose Gaussian Naïve Bayes as the variables are mostly continuous variables in our dataset. Gaussian Naïve Bayes was chosen as a baseline model for us to obtain a preliminary result and expectation for our model performance on the dataset.

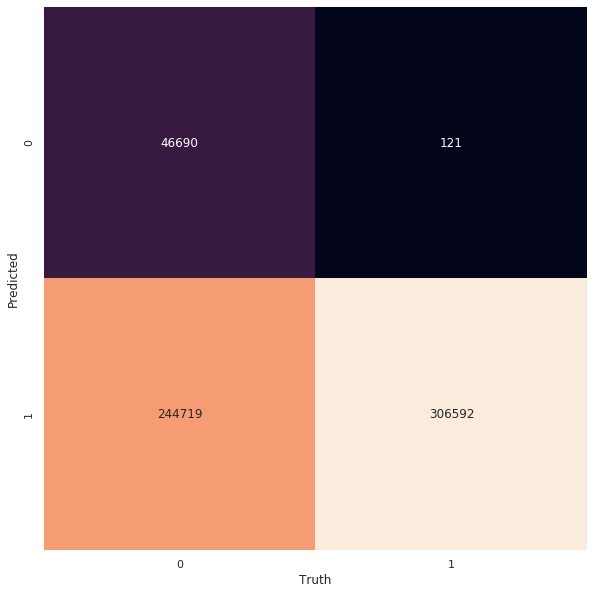


Fig 6. Confusion Matrix

We obtained an accuracy of 59.07%, with the confusion matrix as shown above in Fig. 6.

##### 5.3.2 Logistic Regression

Since Naïve Bayes is not designed to simultaneously support both categorical and continuous features, we decided to use logistic regression as it can handle both types of features and is also suited for binary class problems due to its usage of the sigmoid function.

Data was split into training (0.7) and test (0.3) to be fitted into the logistic regression model with “L2 regularization”. The initial model achieved an accuracy of 57.44%.

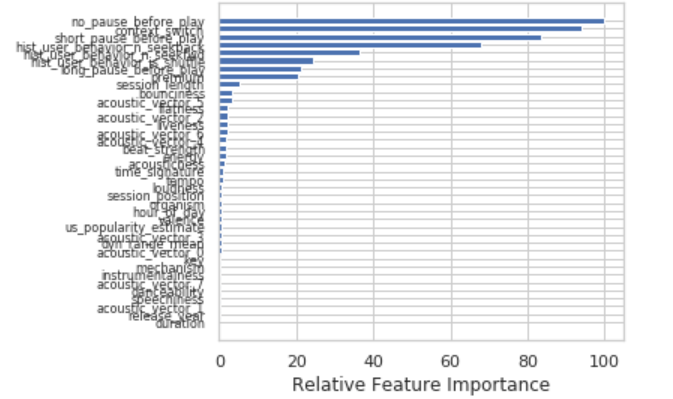


Fig. 7: Feature Importance for Logistic Regression model

After which, we decided to select important features which were specific to this model. With reference to Fig. 7, the variables with zero importance are 'duration', 'release\_year', 'acoustic\_vector\_1', ‘speechiness', 'danceability', 'acoustic\_vector\_7', 'instrumentalness', ‘'mechanism' and 'key'. Thus, we decided to drop those variables with zero importance and analyse if there are any significant differences in model performance. Upon dropping the above variables, we obtained a lower accuracy rate of 57.32% instead. Thus, we decided to use all training features in our model.

To further enhance the model, we performed GridSearch hypertuning that determines the optimal value of the logistic regression model through altering various parameters. Upon hypertuning the model with several possible parameters, the best was:

1. 'C' value of 0.001
2. 'Penalty' of l2

With Gridsearch Hypertuning, we achieved an accuracy rate of 57.46%, which was a 0.02% increase from the initial logistic regression model.

##### 5.3.3 k-Nearest Neighbours (kNN)

We chose kNN as it is a classification algorithm that is easily implemented for binary class problems due to its lack of explicit training. The distance measure used in this model is Euclidean distance.

From our research, we found that kNN works best with lower number of features. When the number of features increase, the increase in dimensionality might lead to the problem of overfitting. To achieve better results, we attempted feature selection to identify important features from the initial kNN model and re-run the model using those identified features. [2]

A limitation of kNN is that the optimal value of k is not known, which may affect the model accuracy. As such, we experimented with four values of k, namely 5, 7, 9 and 11 to compare the model results to indicate the best value of k.

The results are summarised as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | k = 5 | k = 7 | k = 9 | k=11 |
| Accuracy  Without Feature Selection | 50.888% | 50.906% | 50.941% | 50.921% |
| Accuracy with Feature Selection | 51.361% | 51.406 % | 51.411% | 51.363% |

Fig. 8: Model Accuracy for Different Iterations

From the results displayed in Fig. 8, we noticed that models run after feature selection generally achieved a higher accuracy. Furthermore, the best value of k identified was 9. However, the best model accuracy of 51.411% was given upon feature selection, with a k value of 9.

##### 5.3.4 Gradient Boosted Decision Trees (XGBoost)

XGBoost is an implementation of gradient boosted decision trees. They are optimised for execution speed and model performance. Our team felt it was apt to use the XGBoost classifier as our data was structured with our target being a dichotomous variable.

XGBoost was performed on one day’s worth of dataset containing user listening sessions. A XGBoost model was initially generated using our pre-processed data. Using feature\_importances*\_* from the xgboost library, we selected the top 20 features that resulted in the largest gain for our XGBoost model. In theory, this would help us reduce the dimensionality of our data and at the same time, it would improve the training speed of our model at the expense of some accuracy. [3]

We also carried out hyperparameter tuning using GridSearch. Due to high time complexities associated with GridSearch, we conducted a 3-Fold Cross Validation, with a smaller number of parameters to be tuned. We carried out a total number of 24 data-fitting runs to find us the best hyperparameters to be used for our XGBoost model.

The results obtained from the initial XGBoost model was 88.678%. From there, we decided to conduct feature extraction and GridSearch in order to improve the performance of the model, since this was the model which gave the highest accuracy and was also a model that many high-placing solutions achieved the most success with.

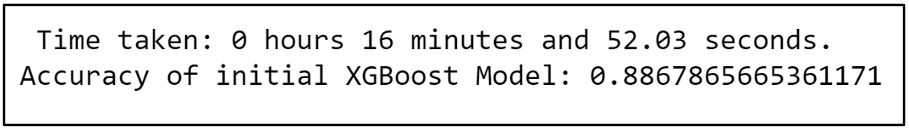


Fig. 9: Initial XGBoost model performance

After selection of important features based on our initial XGBoost model, our group reran our model with the selected features and the results are as shown:

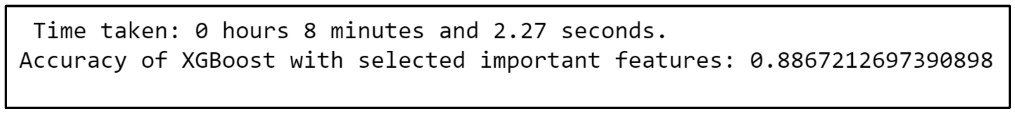


Fig. 10: XGBoost model performance after Feature Selection

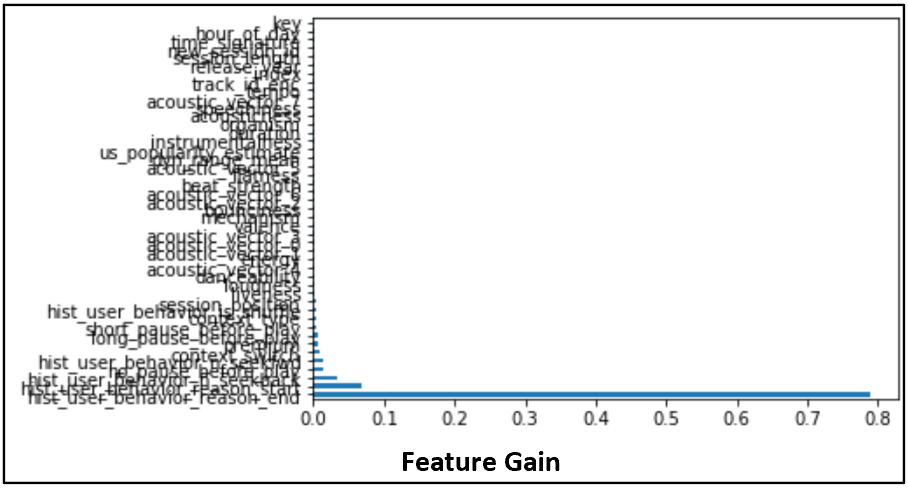


Fig. 11: Feature Gain for each independent feature

Important features extracted:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hist\_user\_behavior\_reason\_end | hist\_user\_behavior\_reason\_start | hist\_user\_behavior\_n\_seekback | no\_pause\_before\_play | hist\_user\_behavior\_n\_seekfwd |
| context\_switch | premium | long\_pause\_before\_play | short\_pause\_before\_play | context\_type |
| hist\_user\_behavior\_is\_shuffle | session\_position | liveness | loudness | danceability |
| acoustic\_vector\_4 | energy | acoustic\_vector\_1 | acoustic\_vector\_0 | acoustic\_vector\_3 |

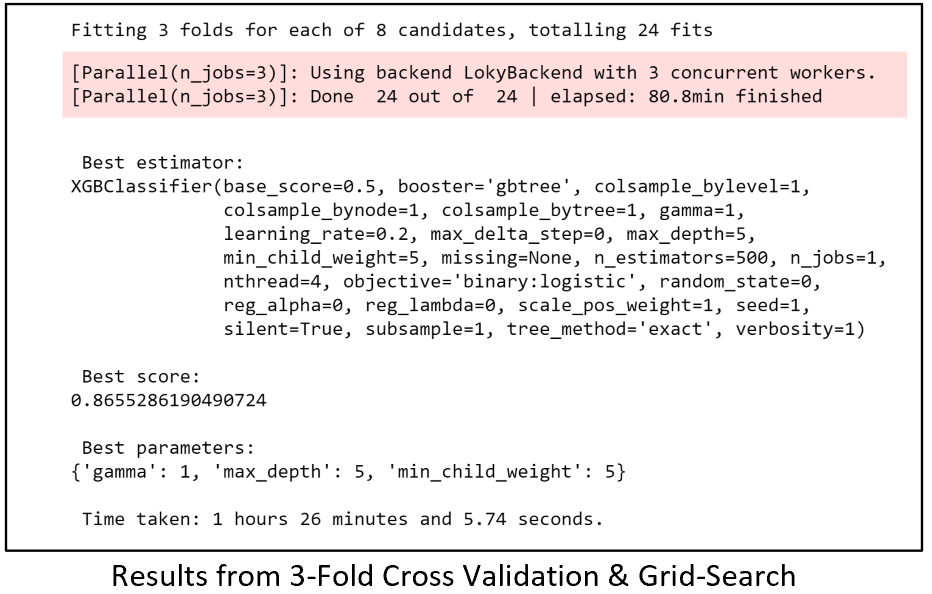


Fig 12: Results from 3-Fold Cross Validation & Grid-Search

Due to time constraints, our group only carried out a hyperparameter grid-search on a few selected hyperparameters. The accuracy obtained from our XGBoost model after performing 3-Fold Cross Validation with GridSearch was 86.553% with the hyperparameters as described above. The reason for the decrease in accuracy can be attributed to the use of 3-fold cross validation.

In the essence of time, our group also decided to use 3-folds as opposed to a higher number of folds. Nevertheless, we acknowledge that 3-fold cross validation might not be adequate in ensuring that our model was not overfitted.

Ultimately, our group felt that our accuracy score of 86.55% was a good measure of how accurate our model was at predicting our skip\_2 variable.

##### 5.3.6 Long-Short Term Memory Model

A recurrent neural network (RNN) was used to model the sequence of skip predictions. Long-short term memory (LSTM) model was chosen as the most suitable model as it can retain important information across time steps which minimizes the vanishing gradient problem faced by standard RNN models. [4] Unlike the aforementioned tree models, this model utilizes multi-labels to conduct predictions of skips instead of a single model.

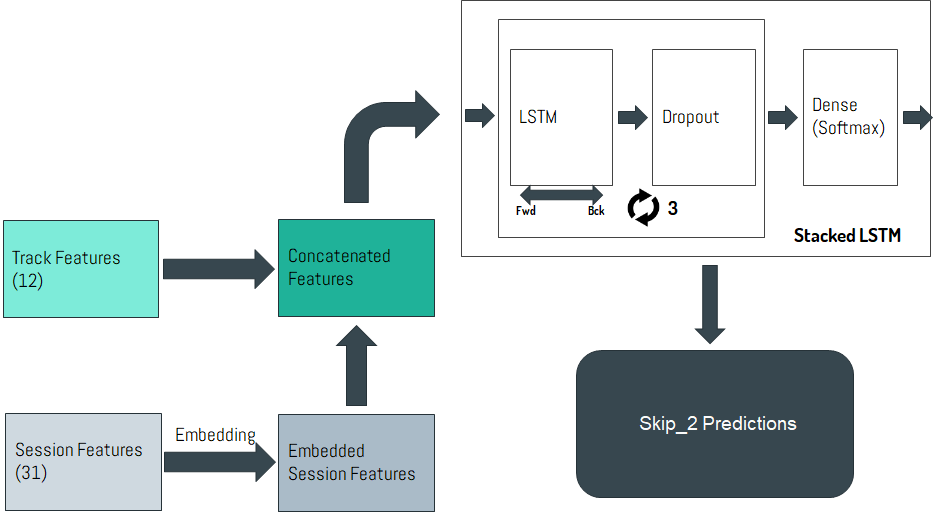


Fig. 13: LSTM Model Architecture

First, the data was transformed into a three-dimensional array *(n, t, f)* where *n* is the number of listening sessions, *t* is the number of tracks within the first half of the listening session, and *f* is the number of features utilized in the model. [5]

An embedding layer was applied to the track features (categorical variables) to compress the track features’ space into a smaller dimensional space. The embedded track features are then concatenated with the session features. [6]

These concatenated features are then passed into three bidirectional LSTM layers with *f* units and a dropout layer of 0.2 was placed after every LSTM layer to reduce overfitting of the model (as shown in Fig. 13).

The final parameters used in this model are as follows:

|  |  |
| --- | --- |
| Batch Size | 128 |
| Epochs | 5 |
| Optimizer | 'nadam' |
| Loss | 'binary\_crossentropy' |

This stacked LSTM model achieved an accuracy of 46.7% during our initial run. However, after subsequent attempts in running the model with varying parameters, the model was able to achieve an accuracy of 47.4%.

### 6. Results and Discussion

Comparing the results obtained from our models, XGBoost attained the highest accuracy of 86.55%. Meanwhile, the LSTM model did not perform as well as we had expected. We believe that this could be due to insufficient data size used to train the model, bad set of hyperparameters used and possible excessive regularization, causing the network to underfit badly. Furthermore, the given test dataset did not include target variables, which prevents us from validating our respective models against the test dataset. Thus, we are unable to verify how well our models perform on the test dataset to measure how well it performs on new data.

### 7. Conclusion and Future Work

We have used a combination of classification and prediction models to identify the best supervised learning algorithm for this task. The models we have developed showed a marked improvement over the baseline model GaussianNB, from 54.17% to 86.55% in the XGBoost implementation.

Given more time, we would also like to focus our efforts on improving the XGBoost accuracy by experimenting with ensembles to combine independent gradient boosted trees generated for each track position­. We could also conduct ensembling of the better-performing models to evaluate its results.

Furthermore, to improve the performance of the LSTM model, we could fine tune the model hyperparameters of our existing LSTM using GridSearch. We could also implement Multi-RNNs in the LSTM model to compare the accuracy of the model against our existing one.

### 8. References

[1] Shakih, R. (2018, Oct 28). Feature Selection Techniques in Machine Learning with Python. Retrieved from:<https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e>

[2] Navlani, A. (2018, Aug 3). KNN Classification using Scikit-learn. Retrieved from: <https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn>

[3] Morde, V. (2019, April 8). XGBoost Algorithm: Long May She Reign! Retrieved from:   
[https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d](about:blank)

[4] Nguyen, M. (2019, July 10). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Retrieved from <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>.

[5] Brownlee, J. (2019, August 24). Gentle Introduction to Models for Sequence Prediction with RNNs. Retrieved from <https://machinelearningmastery.com/models-sequence-prediction-recurrent-neural-networks/>.

[6] Béres, F., Kelen, D. M., & Benczúr, A. A. (n.d.). Sequential skip prediction using deep learning and ensembles. Retrieved from  
<https://people.eng.unimelb.edu.au/jianzhongq/wsdm19-cup-reports/reports/report16.pdf.>