Industrial Engineering and Operations Research
COLUMBIA | ENGINEERING

Predicting Sequential Track-Skipping Behavior on Spotify

(Zi Ying) Sheila Teo

Master of Science in Business Analytics

Abstract

Spotify is an online music streaming service with over 190 million active users interacting with a library of over 40 million tracks. A central challenge for Spotify is to recommend the right music to each user. While there exists a large related body of work on recommender systems, there is at present little work describing how users sequentially interact with the streamed content they are presented with. Music content, in particular, is unique in that the question of if, and when, a user skips a track is an important implicit feedback signal on the quality of the system's recommendation.

In this paper, I explore this important and understudied problem in music streaming using Spotify's data, by building a machine learning model that predicts whether a user will skip or listen to tracks that they are streamed, given their immediately preceding interactions in a listening session.

This paper finds significant model performance with a gradient boosted tree model using XGBoost after engineering features that capture this sequential nature of the data. In particular, these engineered features are found to be among the most important predictors of track-skipping behavior. This paper therefore further highlights the significance of feature engineering during preprocessing as a crucial complement to the commonly emphasized hyperparameter tuning in order to achieve high model performance.

Additionally, this paper presents the significance of threshold-moving, as opposed to simply using the default decision threshold of 0.5 to classify a positive condition. In contrast, I obtain an optimal decision threshold of 0.2655 that minimizes expected costs in the real-world deployment of this XGBoost model to predict track-skipping on Spotify.

Dataset

The dataset used in this paper is obtained from the Spotify Sequential Skip Prediction Challenge jointly hosted by Spotify, WSDM, and CrowdAI[1]. As the datasets made public for this Challenge together contain roughly 130 million listening sessions, only a subset of this data is used for the purposes of this paper. This subset contains 10,000 listening sessions, corresponding to 168,000 user interactions on Spotify. Each user interaction is represented by a row in the data, and the target variable is *skip_2*, a binary variable indicating if the track was skipped by the user.

The dataset contains both session-based information and track-based information for each user interaction. Examples of session-based information include the type of playlist the user interaction takes place in and the time of day of the interaction; examples of track-based information include

[1] https://www.aicrowd.com/challenges/spotify-sequential-skip-prediction-challenge

acoustic features of each track and its popularity measure. A detailed explanation of all features used in this paper is attached in Appendix A.

The segregation of this dataset into training and test sets is performed based on listening sessions. Of the 10,000 training sessions in the dataset, 8000 are subsetted to be training data (corresponding to 134,000 rows of user interactions), while the remaining 2000 are used as test data (corresponding to 34,000 rows of user interactions).

Exploratory Data Analysis

I start by exploring the data in order to better understand patterns and uncover important relationships among predictors. I create visualizations of each predictor's relationship with the target, including kernel density plots, box plots, grouped bar charts, and percent stacked bar charts, where appropriate. Figures 1 to 5 in Appendix B are some examples of the more interesting relationships within the dataset.

For instance, a kernel density plot is created in Figure 1 to visualize the relationship between *session_position* (position of current track within the user's listening session, ie. after how many tracks does the user encounter the current track?) and the target. This allows us to understand if track-skipping behavior is influenced by the number of tracks the user encounters before the current one. A kernel density plot acts akin to a smooth histogram by visualizing this relationship after smoothing out noise; the peaks of the density plot display where values are concentrated, similar to a histogram. From the figure, we observe that session position does indeed play a role in track-skipping behavior- users are more likely to skip a track if they have streamed more than 3 songs prior to it in a listening session.

Another compelling instance is the relationship between *hist_user_behavior_n_seekback* (number of times the user did a seek back within track) and the target, depicted in Figure 2. I create a percent stacked bar chart to visualize the probability of a user skip against the number of times a seek back is performed by the user. On average, there exists a strong negative relationship between the two- the more times a user seeks back on a track, the less likely he is to skip it. Intuitively, this can be justified in that a user seeking back enjoys the track enough to merit hearing the same parts of it multiple times.

Feature Engineering & Transformation

Since the aim of this paper is to predict sequential track-skipping behavior, it is crucial to engineer smart features that are able to capture the sequential nature of our data. This is further pivotal when utilizing algorithms that are not inherently optimized for modelling sequential data. For instance, Long Short-Term Memory Networks (LSTM) would be an advantageous model in this aspect since it is able to remember past relationships when predicting the future. However, since this paper explores methods outside of deep learning for the task at hand, engineering features to capture this temporal nature of user action becomes critical for model performance.

I start by creating a naïve binary variable, *skip_previous*, that depicts if the track encountered right before the current one was skipped by the user. This is built on the fact that user action on previously encountered tracks are correlated with actions on future tracks. Given that this correlation decays

with time, this variable captures only the user action made right before encountering the present track, which is assumed to be the action that is most predictive of user action on the current track.

Next, I relax the above assumption by accounting for user actions on all tracks encountered prior to the current one, through creating a continuous variable, *skip_prop_prior_to_track.* This variable represents the proportion of skips on all tracks previously encountered.
However, simply obtaining this proportion of skips is not enough to tell the whole story. I therefore additionally engineer a new continuous variable, *skip_prop_prior_to_track_SD*, representing the standard deviation of the previously created variable, *skip_prop_prior_to_track*, at each point in time. This depicts the consistency of the user's skipping action prior to the current track. For instance, *skip_prop_prior_to_track_SD=0* on track 5 implies that the user had made the same action on all of the 4 prior tracks (either skipped all or not skipped all). We can therefore be more certain of the predictive power of *skip_prop_prior_to_track* for forecasting user action on this 5th track, since the user is very likely to also make the same action. As such, the lower the *skip_prop_prior_to_track_SD*, the more consistent the user's past track-skipping behavior, and the more certain we are in predicting user action on the next track. Creating this variable thus allows us to feed such information into our model.

Lastly, I investigate the relationship between *date* and the target. The predictive power of the column *date* in forecasting track-skipping likely comes from its representation of the day of week, instead of the date in itself. I therefore convert this column into a 7-level categorical variable, each level being a day of the week. This allows us to not only reduce the cardinality of the variable, but also condense the information it provides for predicting track-skipping into a more targeted format for our model. I further create a percent stacked bar chart, shown in Figure 4, to visualize skipping behavior over each day of the week.

Algorithm: eXtreme Gradient Boosting (XGBoost)

This paper implements Gradient Boosted Trees for the task at hand. The choice of GBMs is simple- due to its high model performance. While Random Forests build an ensemble of deep independent trees, GBMs build an ensemble of shallow trees in sequence, with each tree learning and improving on the previous one. These shallow trees are then boosted to produce a powerful "committee" that, when appropriately tuned, often outperforms any other common classifiers such as Logistic Regression or Random Forests.

In particular, this paper utilizes eXtreme Gradient Boosting (XGBoost) due to its advantage over traditional boosting algorithms. XGBoost offers additional regularization hyperparameters that provide added protection against overfitting, improving model performance.

Encoding Categorical Variables: Regularized K-Fold Target Encoding

Since XGBoost requires a matrix input for predictors, the encoding of all categorical predictors is essential before being able to run the algorithm. There exists a number of high cardinality categorical predictors in our dataset:

Categorical Predictor	**Number of Categories**
context_type	6
hour_of_day	24

hist_user_behavior_reason_end	7
day_of_week	7
hist_user_behavior_reason_start	9
release_year	69
key	12
time_signature	5
session_length	11

To this end, k-fold target encoding is chosen as an extension of naïve target encoding in order to reduce overfitting. In naïve target encoding, each category of a categorical predictor is replaced by the mean of the target corresponding to it. However, this method suffers from high overfitting tendency, since each observation's target value is used in the computation of the mean target value for that same observation's category. This thus introduces a "data leak" for all observations, wherein the target value for each observation is leaked onto its predictors. This results in an overdependence on the target-encoded predictors and overfitting.

In order to prevent such data leakage, k-fold target encoding instead ensures that the target value of an observation is not used to compute its target-encoded predictors. Instead, other observations in the training data are used. The dataset is first split into k folds, and the means for each category in the i^{th} fold is computed using data in the other *k-1* folds. To further prevent overfitting, I additionally add random noise into the target-encoded values for each category. This provides extra regularization to the k-fold target encoding.

Baseline XGBoost

I first decide on an appropriate loss function to use in building the XGBoost. I choose to utilize log loss, due to the fact that it penalizes by taking into account the uncertainty of model predictions based on how much they vary from the actual label. For instance, the log loss penalizes predicted probabilities closer to 0 then it does for those closer to 1, for a true positive case. This pushes our model to learn the "correct" probabilities. This is as opposed to the commonly chosen misclassification error metric in building classification models.

I start by building an XGBoost with its default hyperparameters of *eta=0.3, gamma=0, max_depth=6, subsample=1,* and *colsample_bytree=1*. This provides a baseline against which to compare our subsequently tuned model. Using these default hyperparameters, I conduct 5-fold cross validation on the training set with an early stopping round of 30. This ensures that the training process stops when the mean log loss of the holdout set has not improved in 30 rounds. This iteration that it stops at is then taken as a proxy for the optimal *nrounds* value (maximum number of iterations) that go with the above default hyperparameter values to produce a minimum log loss on the unseen test set. Figure 6 depicts the mean log loss obtained on the training set and holdout set for each *nrounds* value, and the shaded ribbon around the line represents its standard deviation. The optimal *nrounds* value is 79, producing a minimum mean log loss on the holdout set of 0.416.

Next, I train an XGBoost using this optimal *nrounds* value and evaluate it on the test set, which has been left unseen until now. Figure 7 depicts the log loss obtained on the training set and test set respectively for each *nrounds* value. We observe that at *nrounds*=79, log loss on the test set is 0.674. This is much higher than the value of 0.416 obtained on the holdout set from our previous 5-fold cross validation, depicting that our error from cross validation is more optimistic than the true test

error, which is often the case in practice. Furthermore, we observe from the figure that the minimum log loss produced from the test set is 0.550 and this occurs at less than 10 *nrounds*, instead of 79 that was estimated using cross validation on the training set. Additionally, we observe that training set log loss falls at a much higher rate than test set log loss. This indicates that our baseline XGBoost is greatly overfitting on the training set, and simply using these default hyperparameter values is insufficient to obtain optimal model performance on our data.
This baseline XGBoost performs as follows on the test set, using a default decision threshold of 0.5 for classifying the positive condition. Figure 8 depicts the confusion matrix used in calculating these metrics.

Metric	**Score**
Accuracy	64.89%
Specificity (TNR)	70.12%
Recall (TPR)	60.06%
Precision	68.49%
F2-Score	61.58%

Of the five metrics above, three are of particular significance in our specific business case:

1) **Recall**: It is of greater business utility to identify tracks that will be skipped over ones that will not. Identification of these songs can pave the way for Spotify to improve on their track recommendation system.
2) **Precision**: It is important for the model to be certain when identifying a track that will be skipped.
3) **F2-Score**: Given our combined emphasis on both Recall and Precision, I ultimately look at the F2-Score as a combined measure of both. In particular, choosing the F2-Score over the more commonly utilized F1-Score implies Recall is weighed twice as heavily as compared to Precision. I particularly choose this due to the very likely differences in the real-world business costs of False Positives (FP) and False Negatives (FN) for Spotify in the context of track-skipping.

   A FP occurs when a song is identified to be skipped and thus not recommended to a user, but in fact would not have been skipped. In this context, costs are incurred due to the user not being exposed to a song he would have enjoyed. However, this presents a relatively lower cost to Spotify since user satisfaction is unaffected. As such, Precision is less important in this business case.

   On the other hand, a FN occurs when a song identified to not be skipped is in fact skipped by a user. Costs are incurred due to lower user satisfaction with the streaming service. This presents a much higher cost to Spotify. As such, Recall is far more important in this business case.

   Given this imbalanced importance of Recall against Precision, I therefore choose to utilize F2-Score as opposed to the more commonly used F1-Score, which instead weighs Recall and Precision equally.

From the scores on the above 3 important metrics, we observe that the baseline XGBoost performs relatively poorly. In addition, the ROC curve for this baseline XGBoost is depicted on Figure 9, which gives the model's performance as the decision threshold for classifying positives varies. The area under the ROC curve (ROC-AUC) is 0.7049. This ROC-AUC value is particularly interesting due to its probabilistic interpretation. A value of 0.7049 implies that 70.49% of outcomes are

correctly ordered according to the probabilities outputted by the baseline XGBoost. This ordering of outcomes is particularly applicable to our analysis, wherein Spotify can choose to act on the first N outcomes ranked with the highest probability of user skips. Next, I seek to improve model performance through hyperparameter tuning.

Tuning XGBoost: Bayesian Optimization with K-Fold Cross Validation

I utilize a Bayesian Optimization search method with 5-fold cross validation in order to obtain the optimal hyperparameters in the most efficient manner. This is as compared to using simpler hyperparameter search methods such as Grid Search and Random Search, where each hyperparameter combination searched during tuning is independent of the last. In contrast, Bayesian Optimization uses knowledge of previous iterations of the algorithm such that each new search is guided from previous search results. This allows the algorithm to obtain optimal hyperparameters in as few iterations as possible, thus reducing computational cost.

Specifically, Bayesian Optimization works by first constructing a posterior distribution of functions that best describe the intended function to optimize in the XGBoost model. This function then acts as a surrogate such that the algorithm simply optimizes hyperparameters to this surrogate function, instead of building a new model and going through the training and evaluation loop for each hyperparameter combination to be searched. The hyperparameters that minimize this surrogate function then provide an estimate of the optimal hyperparameters in the model.

The range of each hyperparameter searched using Bayesian Optimization are:

- *eta* range (learning rate): 0.1 to 0.3
- *nrounds* range (maximum number of iterations): 80 to 500
- *max_depth* range (maximum depth each iteration's tree): 1 to 4
- *subsample* range (proportion of training rows used to build each iteration's tree): 0 to 1
- *colsample_bytree* range (proportion of predictors used to build each iteration's tree): 0 to 1

The choice of the above hyperparameter ranges to search stem from insights obtained from the previous baseline XGBoost created. Given that this baseline model suffered from overfitting, I first lowered *eta* from its previously used default value of 0.3 and increased *nrounds* to compensate for this. Lowering *eta* reduces the rate of overfitting. Next, I reduced *max_depth* from its previously used default value of 6 in order to ensure that trees grown in each iteration are even weaker learners than before. Lastly, I reduced both *subsample* and *colsample_bytree* from their previously used default values of 1, so as to decorrelate the trees built in each iteration. All these together serve to reduce the problem of overfitting present in the baseline XGBoost.

Upon completion of the Bayesian Optimization search, the optimal hyperparameter values obtained are as follows:

- *eta: 0.3*
- *nrounds: 473*
- *max_depth: 1*
- *subsample: 0.964*
- *colsample_bytree*: *0.998*

Regularizing XGBoost

Lastly, I explore the key feature of the XGBoost algorithm that gives it an advantage over other traditional boosting algorithms- its additional regularization hyperparameter, *gamma*, that allows for added protection against overfitting. *gamma* is the minimum loss reduction required to create a further partition on a leaf node of the tree and can take values from 0 to infinity. The larger the *gamma*, the greater the regularization effect. Using 5-fold cross validation, I search for the optimal *gamma* value across the range of 0 to 100. This optimal *gamma* is found to be 14.

Final XGBoost Model

The final XGBoost model produces the following performance, using a default decision threshold value of 0.5 for classifying the positive condition. Figure 10 depicts the confusion matrix used in calculating these metrics.

Metric	**Score**
Accuracy	76.80%
Specificity (TNR)	71.44%
Recall (TPR)	81.76%
Precision	75.58%
F2-Score	80.44%

We observe an improvement on all of the above metrics. In particular, the metric most appropriate in the context of our analysis, F2-Score, increases by 30.63% from before. This underscores the significance of hyperparameter tuning in achieving higher model performance for our specific business goal.

The ROC curve for this final XGBoost model is depicted on Figure 11, achieving an ROC-AUC of 0.7874. This indicates that 78.74% of outcomes are correctly ordered according to the probabilities outputted by the final XGBoost.

Figure 12 depicts the log loss obtained on the training set and test set respectively for each *nrounds* value, across 3 versions of the XGBoost: Baseline (using default hyperparameters), Tuned (using optimal hyperparameters found from Bayesian Optimization), and Final (extra *gamma* regularization added on top of the tuned model). We observe that tuning and regularization allows for test set log loss to fall across all values of *nrounds* (iterations). However, the optimal *nrounds* for the test set to attain a minimum log loss occurs at a value much lower than the value of *nrounds=473* as obtained from hyperparameter tuning via conducting cross validation on the training set. This suggests that our final model is not yet entirely optimal and can be further tuned for improved performance on the test set.

Threshold-Moving to Minimize Expected Costs for Spotify

Thus far, I have used the default decision threshold of 0.5 to classify the positive condition (track is skipped). However, this default threshold will only be optimal if the business costs to Spotify of a False Negative (FN) and False Positive (FP) are equal. However, given the likely very different costs of a FP and FN as previously stated, I instead explore alternative decision thresholds to minimize expected costs for Spotify in the real-world deployment of this XGBoost model to predict track-skipping.

In particular, this paper has previously discussed the higher cost of a FN relative to FP in the context of track-skipping. For the following analysis, I assume that a FN generates twice the cost of a FP for Spotify. For each decision threshold, I obtain the resulting confusion matrix and subsequently compute expected costs as $2FN + FP$. The optimal threshold is then defined as the one resulting in the minimum expected cost. Figure 13 depicts this change in expected cost for Spotify as the decision threshold varies.

Following the above methodology and varying thresholds to the 4th decimal place, the optimal decision threshold on the test set is found to be 0.2655, ie. a track is classified to be skipped as long as its predicted probability from the final XGBoost model is above 0.2655. This results in the following model performance. Figure 14 depicts the new confusion matrix obtained at this optimal threshold.

Metric	**Score**
Accuracy	72.40%
Specificity (TNR)	48.09%
Recall (TPR)	94.88%
Precision	66.40%
F2-Score	87.38%

Importantly, F2-Score has increased by a further 8.63% compared to simply using the default decision threshold of 0.5 from before, stemming from a significant increase in Recall due to a lower decision threshold. This depicts higher model performance while achieving minimum expected costs for Spotify, in order to increase the real-world business impact of deploying the XGBoost model.

Importantly, this paper assumes that the cost of a FN is twice that of a FP. However, the above analysis can be similarly repeated using the true relative costs of a FN and FP to Spotify, should these quantities become available with additional data.

Feature Importances

Last but not least, I wrap up my analysis by inspecting the feature importance plot of the final XGBoost model, in order to determine the most important features in predicting track-skipping. Importantly, I choose to utilize "gain" instead of the commonly used "frequency" metric for computing feature importance. This is due to the fact that the "frequency" metric refers to the frequency with which a feature is used to split a node within the model. However, categorical features- especially ones with minimal cardinality- will naturally have low frequency scores since they possess lesser predictor-value combinations to be used in node splitting. On the other hand, continuous features or categorical features with high cardinality have a larger range of values which naturally increases the likelihood of these features being used to split nodes. Hence, the "frequency" metric is biased towards high cardinality features. Instead, the "gain" metric measures the relative contribution of the feature to the model, as calculated by taking its contribution for each tree in the model. This provides a far less biased feature importance ranking, which is depicted on Figure 15.

From this figure, we observe that *hist_user_behavior_reason_start* (the user action which led to the current track being played), *session_id* (unique identifier for each listening session), and *skip_previous* (whether the track right before the current one was skipped) are the top three most important features in predicting track-skipping on Spotify. These serve as important business insights

into user interactions on Spotify that can pave the way for increasing the quality of its streaming recommendations.

In particular, we observe that two of the features engineered to take into account the sequential nature of the data, namely *skip_prop_prior_to_track* (proportion of tracks prior to current track that were skipped by user) and *skip_previous* rank highly on the feature importance plot. This depicts the crucial importance of sequential information in predicting track-skipping behavior.

Conclusion & Future Improvements

This paper explores utilizing XGBoost to predict sequential track-skipping on Spotify. The increased performance of the model after hyperparameter tuning serves to underscore the importance of model tuning for machine learning. However, one cannot overlook the significance of feature engineering for model performance. This is particularly important in the context of this analysis, wherein sequential and temporal user behavior was fed into our models through sheer feature engineering.

Additionally, this paper presents the significance of threshold-moving in order to obtain an optimal decision threshold that minimizes expected costs in the real-world deployment of a machine learning model that predicts track-skipping on Spotify. This is particularly relevant in this context, given that the relative cost of a False Negative and False Positive likely differ greatly.

A worthy extension of this paper would be to utilize deep learning algorithms to better capture the sequential behavior in track skipping. In particular, the usage of Long Short-Term Memory Networks (LSTMs) would be highly relevant and an intriguing pursuit.

Appendix A: Dataset Features

Session-Based Information

Feature	Description
session_id	E.g. 65_283174c5-551c-4c1b-954b-cb60ffcc2aec - unique identifier for the session that this row is a part of
session_position	{1-20} - position of row within session
session_length	{10-20} - number of rows in session
track_id_clean	E.g. t_13d34e4b-dc9b-4535-963d-419afa8332ec - unique identifier for the track played. This is linked with track_id in the track features and metadata table.
skip_2	Boolean indicating if the track was only played briefly
not_skipped	Boolean indicating that the track was played in its entirety
context_switch	Boolean indicating if the user changed context between the previous row and the current row. This could for example occur if the user switched from one playlist to another.
no_pause_before_play	Boolean indicating if there was no pause between playback of the previous track and this track
short_pause_before_play	Boolean indicating if there was a short pause between playback of the previous track and this track
long_pause_before_play	Boolean indicating if there was a long pause between playback of the previous track and this track
hist_user_behavior_n_seekfwd	Number of times the user did a seek forward within track

hist_user_behavior_n_seekback	Number of times the user did a seek back within track
hist_user_behavior_is_shuffle	Boolean indicating if the user encountered this track while shuffle mode was activated
hour_of_day	{0-23} - The hour of day
date	E.g. 2018-09-18 - The date
day_of_week	Engineered from *date* feature
premium	Boolean indicating if the user was on premium or not. This has potential implications for skipping behavior.
context_type	E.g. editorial playlist - what type of context the playback occurred within
hist_user_behavior_reason_start	E.g. fwdbtn - the user action which led to the current track being played
hist_user_behavior_reason_end	E.g. trackdone - the user action which led to the current track playback ending
skip_previous	Boolean indicating if the track encountered right before the current one was skipped by the user
skip_prop_prior_to_track	Proportion of tracks prior to current track that were skipped by user
skip_prop_prior_to_track_SD	Standard deviation of *skip_prop_prior_to_track* - Represents the consistency of user action (skipping) prior to the current track

Track-Based Information

Feature	**Description**
track_id	E.g. t_13d34e4b-dc9b-4535-963d-419afa8332ec - unique identifier for the track played. This is linked with track_id_clean in the session logs
duration	Length of track in seconds
release_year	Estimate of year the track was released
us_popularity_estimate	Estimate of the US popularity percentile of

	the track as of October 12, 2018
acousticness	See https://developer.spotify.com/documentation/ web-api/reference/tracks/get-audio-features/
beat_strength	See acousticness
bounciness	See acousticness
danceability	See acousticness
dyn_range_mean	See acousticness
energy	See acousticness
flatness	See acousticness
instrumentalness	See acousticness
key	See acousticness
liveness	See acousticness
loudness	See acousticness
mechanism	See acousticness
mode	See acousticness
organism	See acousticness
speechiness	See acousticness
tempo	See acousticness
time_signature	See acousticness
valence	See acousticness
acoustic_vector_0	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_1	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_2	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_3	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_4	See http://benanne.github.io/2014/08/05/spotify-cnns.html and

	http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_5	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_6	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-
acoustic_vector_7	See http://benanne.github.io/2014/08/05/spotify-cnns.html and http://papers.nips.cc/paper/5004-deep-content-based-

Appendix B: Figures

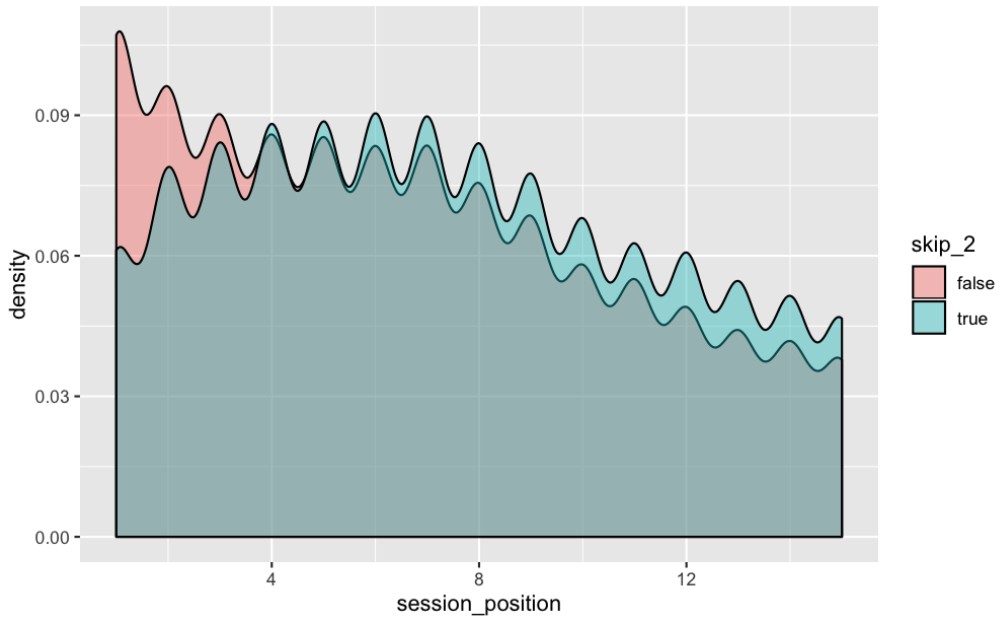

Figure 1: Relationship between *session_position* and target (*skip_2*)

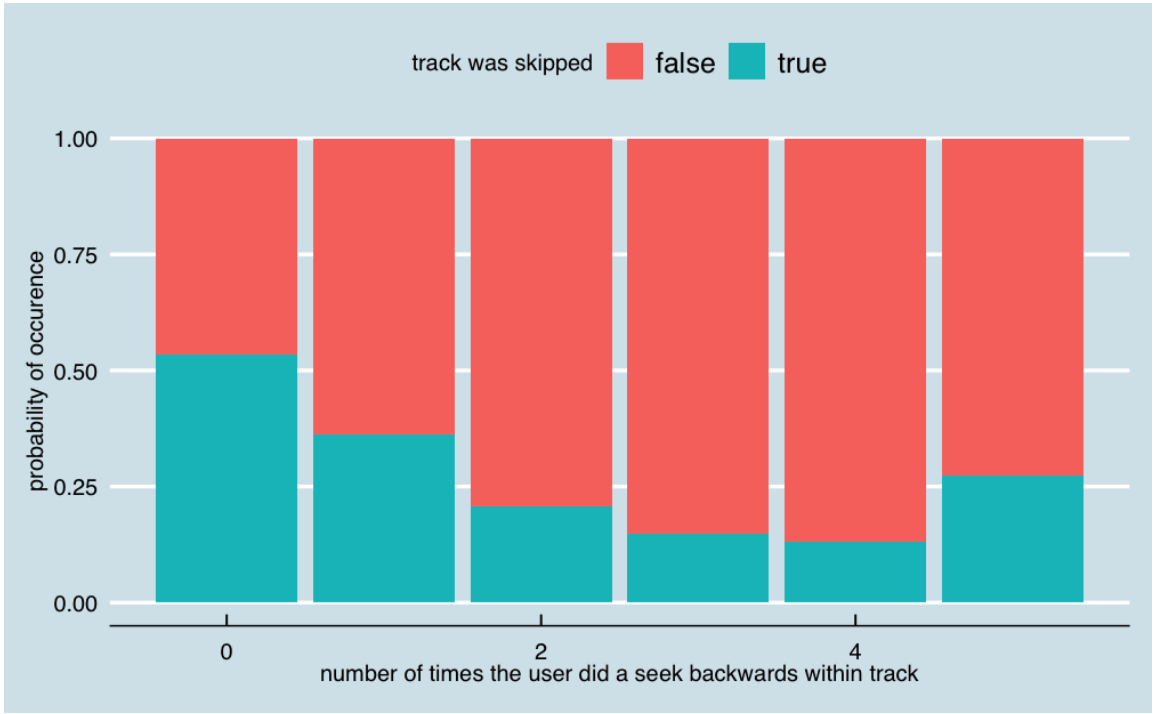

Figure 2: Relationship between hist_user_behavior_n_seekback and target (skip_2)

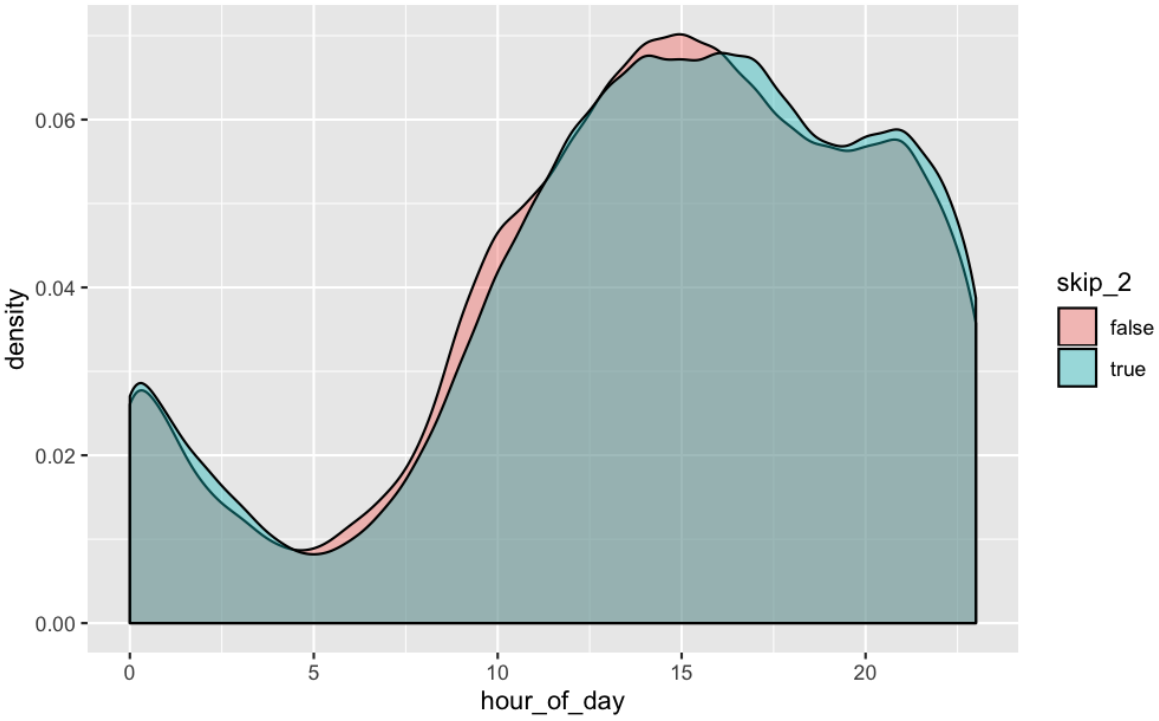

Figure 3: Relationship between *hour_of_day* and target (*skip_2*)

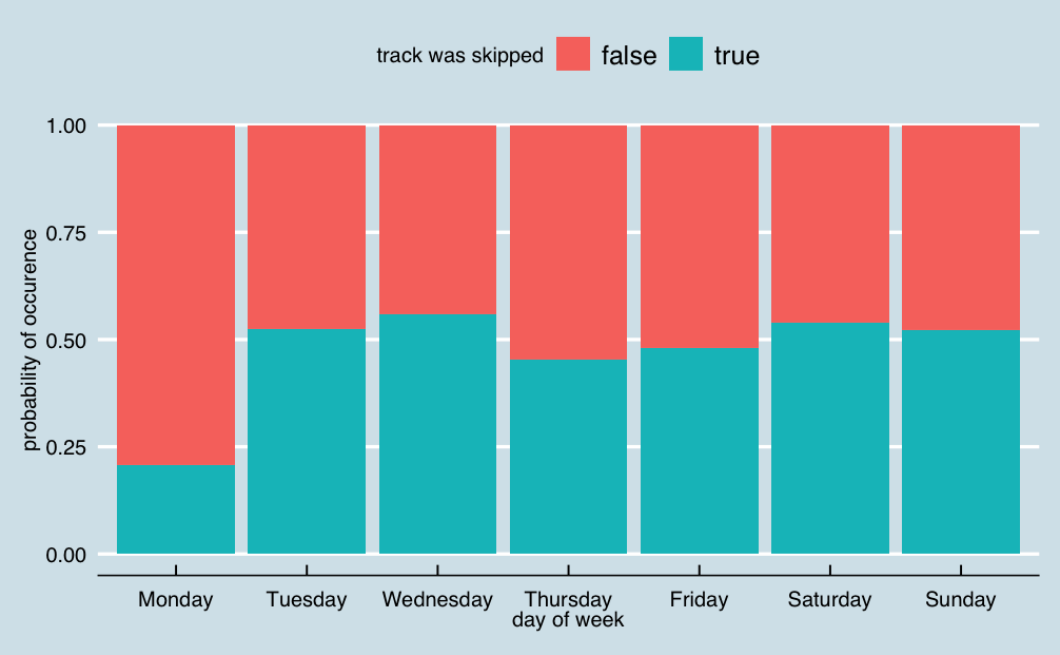

Figure 4: Relationship between *day_of_week* and target (*skip_2*)

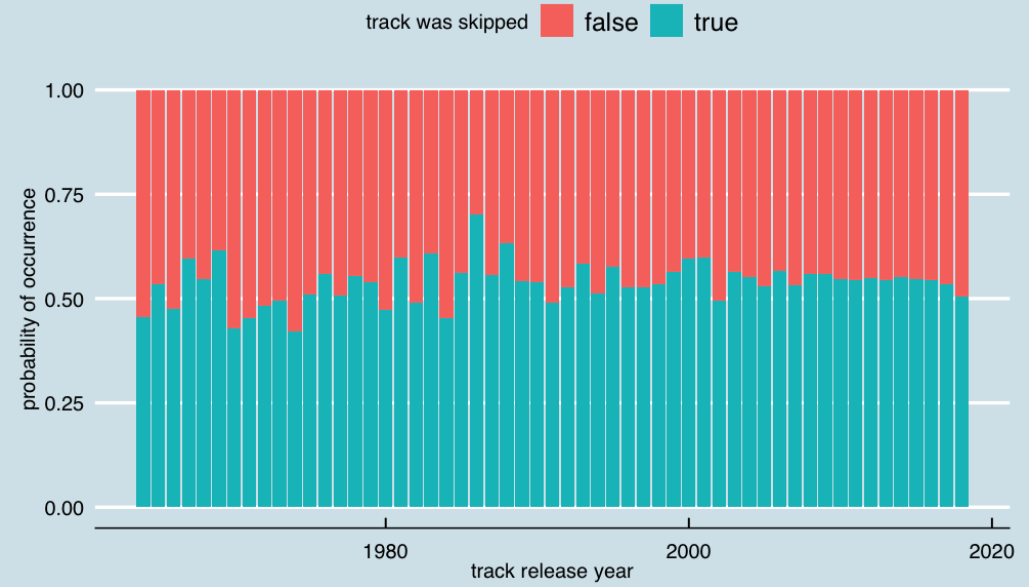

Figure 5: Relationship between *release_year* and target (*skip_2*)

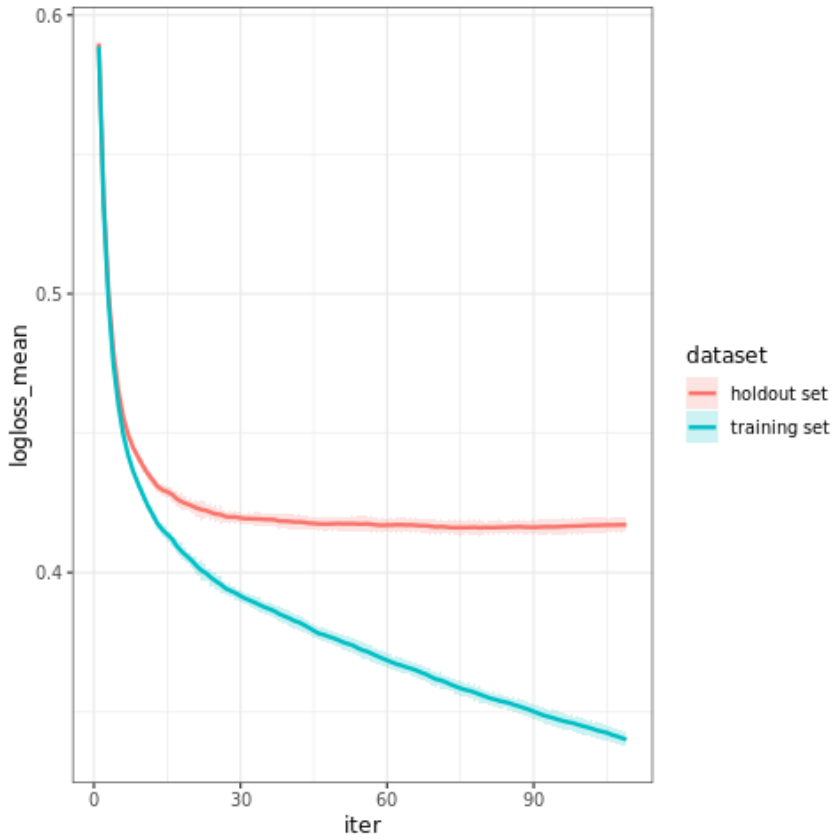

Figure 6: Mean log loss obtained on the training set and holdout set for each *nrounds* value (maximum number of iterations)

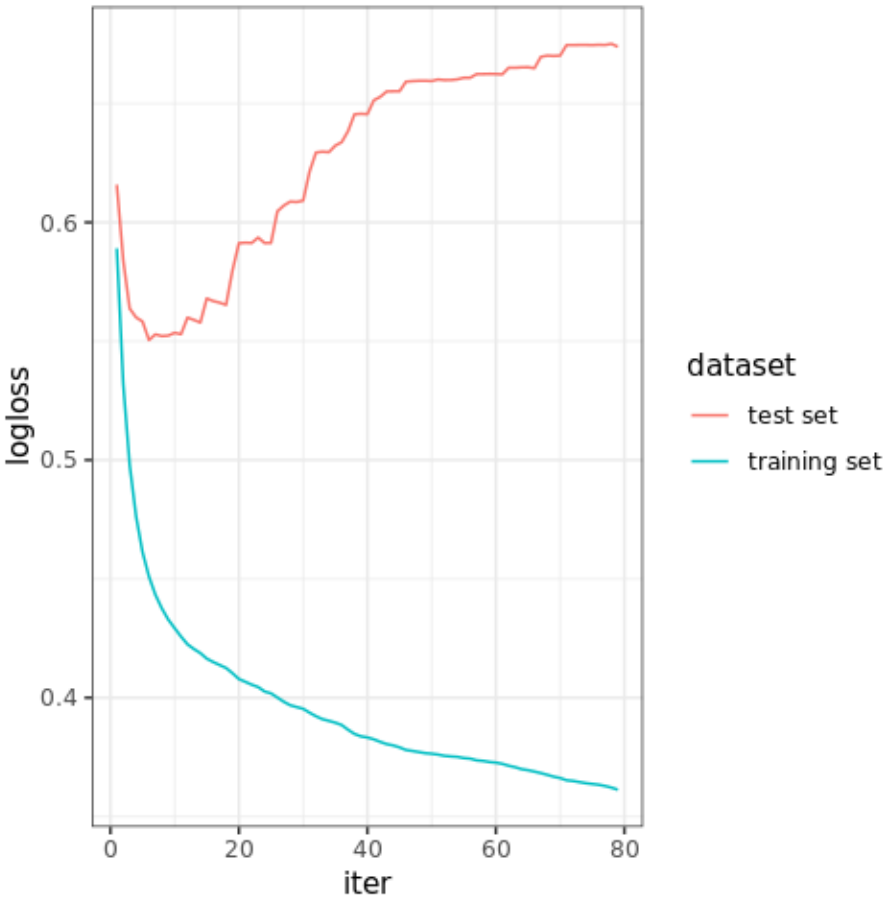

Figure 7: Log loss obtained on training set and test set for each *nrounds* value in Baseline XGBoost

	True Positive	**True Negative**
Predicted Positive	10460	4812
Predicted Negative	6955	11292

Figure 8: Confusion Matrix for Baseline XGBoost (Using Decision Threshold of 0.5)

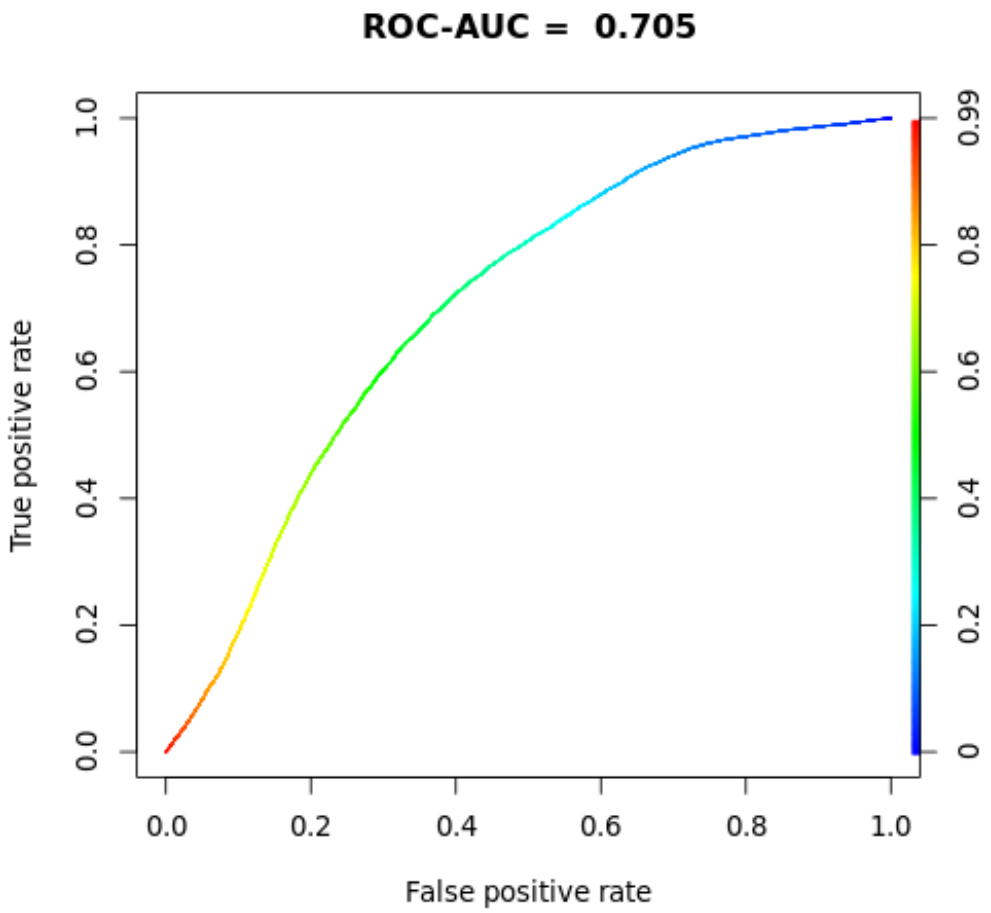

Figure 9: ROC Curve of Baseline XGBoost

	True Positive	**True Negative**
Predicted Positive	14238	4600
Predicted Negative	3177	11504

Figure 10: Confusion Matrix for Final XGBoost (Using Decision Threshold of 0.5)

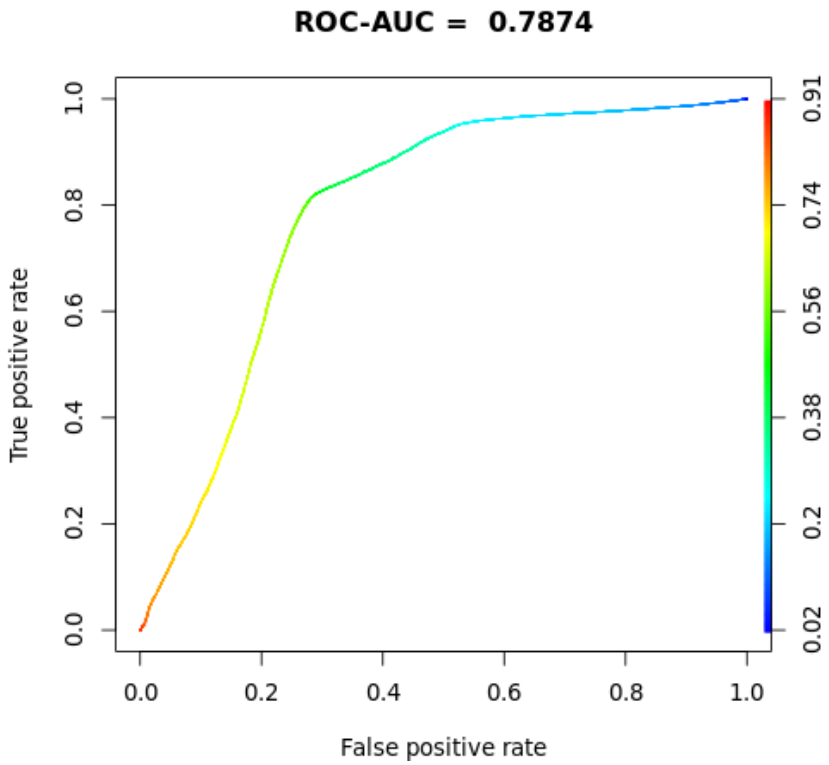

Figure 11: ROC Curve of Final XGBoost

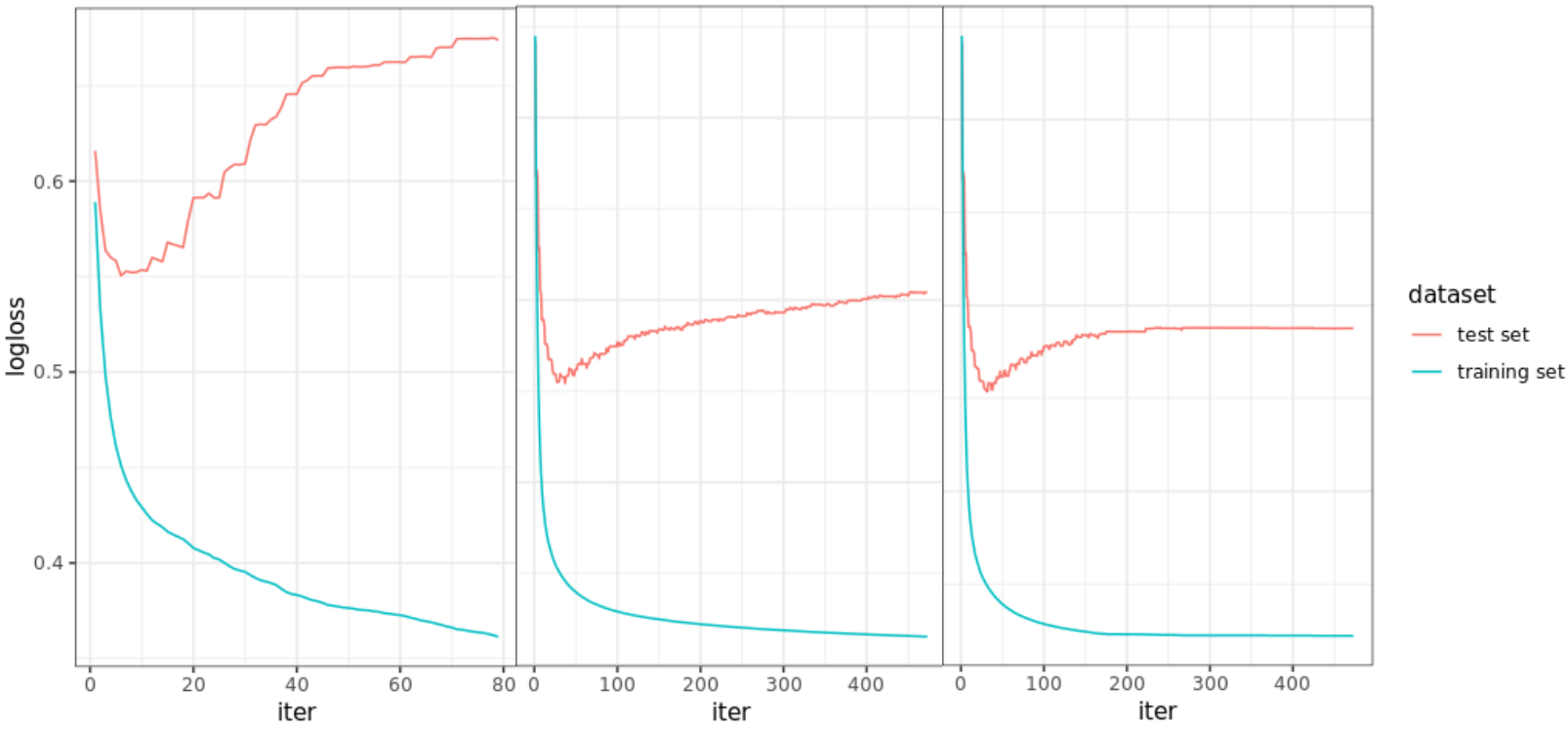

Figure 12: Log Loss for 3 Versions of XGBoost: Baseline, Tuned, Final (From left to right)
Note: Constant Y-axis for each plot

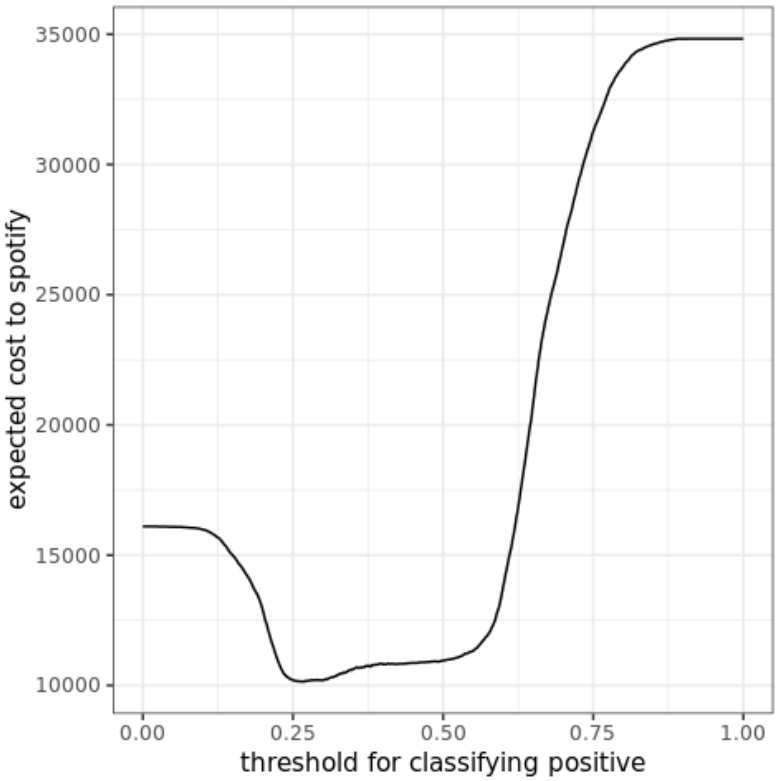

Figure 13: Expected Cost to Spotify against Decision Threshold

	True Positive	**True Negative**
Predicted Positive	16524	8360
Predicted Negative	891	7744

Figure 14: Confusion Matrix for Final XGBoost
(Using Optimal Decision Threshold of 0.2655)

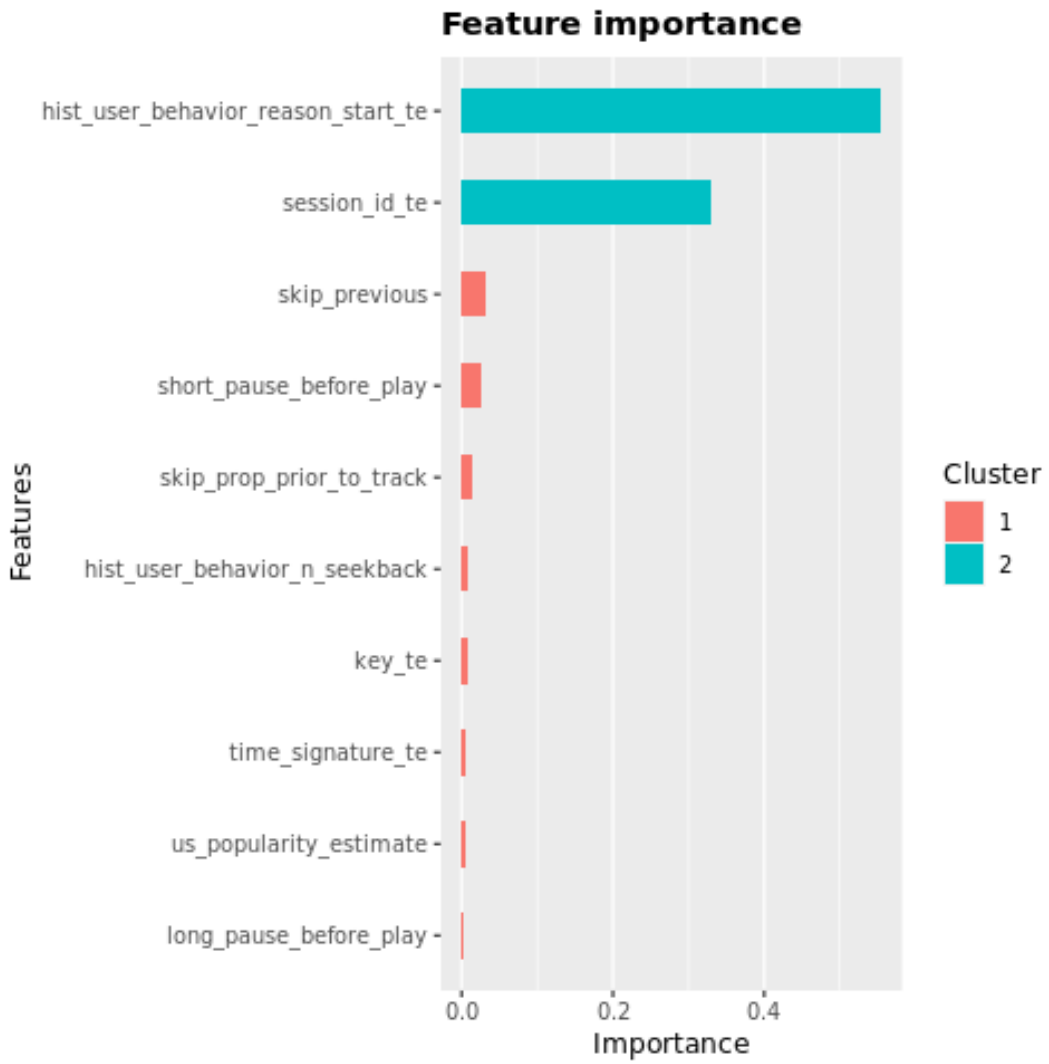

Figure 15: Feature Importance Plot Using 'Gain' Metric

References

"Spotify Sequential Skip Prediction Challenge." AIcrowd, *www.aicrowd.com/challenges/spotify-sequential-skip-prediction-challenge/leaderboards?challenge_round_id=240*.

Saraswat, Manish. "Beginners Tutorial on XGBoost and Parameter Tuning in R Tutorials & Notes: Machine Learning." HackerEarth, *www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/*.

Laurae. "XGBoost and the Tuning of Regularization." Medium, Data Science & Design, 3 Jan. 2017, *medium.com/data-design/xgboost-hi-im-gamma-what-can-i-do-for-you-and-the-tuning-of-regularization-a42ea17e6ab6.*

"Tuning Hyperparameters." Mlr, *mlr.mlr-org.com/articles/tutorial/tune.html.*

Abdelkader, Metales. "Bayesian Hyperparameters Optimization." Modeling with R, 11 June 2020, *modelingwithr.rbind.io/bayes/hyper_bayes/bayesian-hyperparameters-method/#random-forest-model.*

Arasanipalai, Ajay Uppili. "A Practical Guide To Hyperparameter Optimization." AI & Machine Learning Blog, AI & Machine Learning Blog, 8 Mar. 2021, *nanonets.com/blog/hyperparameter-optimization/.*

Insight. "Visualizing Machine Learning Thresholds to Make Better Business Decisions." Medium, Insight, 30 Sept. 2017, *blog.insightdatascience.com/visualizing-machine-learning-thresholds-to-make-better-business-decisions-4ab07f823415*.

Antico, Adrian. "The Easiest Way to Create Thresholds And Improve Your Classification Model." R-Bloggers, 11 Apr. 2019, *www.r-bloggers.com/2019/04/the-easiest-way-to-create-thresholds-and-improve-your-classification-model/.*