Predicting Music Listening Behaviour on Deezer: From Competition Pipelines to Production Recommender Systems

Vimerlin Govender[1], Meleknur Özgü[1] and Xinmeng Song[1]

[1]*Lucerne University of Applied Sciences and Arts (HSLU)*

Abstract

We address the Deezer skip prediction task from the 2017 Data Science Game, where the goal is to predict whether a user will listen to or skip the first track recommended by Deezer's Flow feature. We frame this as a binary classification problem evaluated by ROC AUC. Our competition-focused pipeline engineers 47 features spanning temporal context, release metadata, user engagement statistics, target-encoded categorical variables, item popularity, and user-item affinity signals. An ensemble of XGBoost and LightGBM achieves a cross-validation AUC of 0.935 on a random training split, though we critically examine why this substantially exceeds the winning competition score of 0.686, identifying evaluation methodology as the key differentiator. We then contrast this competition solution with a production-grade recommendation architecture for Deezer, and analyse where competition-driven and production-driven approaches overlap and diverge.

Keywords

recommender systems, music recommendation, skip prediction, gradient boosting, feature engineering

1. Introduction

Music streaming platforms such as Deezer serve hundreds of millions of tracks to users with diverse tastes and listening contexts. Deezer's *Flow* feature provides a personalised, infinite stream of music—a mix of familiar favourites and new discoveries. Predicting whether a user will engage with (listen to) or reject (skip) a Flow recommendation is central to improving this experience [1].

The Deezer Data Science Game 2017 (DSG17) formalises this as a binary classification task: given a user's one-month listening history, predict whether they will listen to or skip the first track Flow recommends [2]. A track is considered "listened" if the user hears more than 30 seconds of it (`is_listened` = 1); pressing skip before 30 seconds yields `is_listened` = 0. The competition attracted strong international participation, with the winning team ("lebed i 3 raka", MSU Russia) achieving a test AUC of 0.686.

In this report, we pursue three objectives aligned with the course requirements. First, we develop a competition-focused solution that maximises AUC through feature engineering and model ensembling (**Q1**, Section 3). Second, we propose a production-grade recommendation system for Deezer's broader needs (**Q2**, Section 4). Third, we critically compare these two perspectives (**Q3**, Section 5).

2. Dataset and Problem Setup

2.1. Dataset Description

The training set contains 7,558,834 interactions from 19,914 users across 451,867 tracks, representing one month of listening history. Each record includes user and item identifiers (`user_id`, `media_id`, `artist_id`, `album_id`, `genre_id`), a Unix timestamp, track release date, song duration, and the binary target `is_listened`. The overall listen rate is 68.4%.

Recommender Systems Course Project, February 2026

vimerlin.govender@stud.hslu.ch (V. Govender); meleknur.oezgue@stud.hslu.ch (M. Özgü); xinmeng.song@stud.hslu.ch (X. Song)

The test set contains exactly 19,918 rows—approximately one row per user—each representing the *first track Flow recommended* to that user. This is a fundamentally different distribution from training: the training data captures general listening behaviour across many tracks per user, while the test set captures a single, curated recommendation event per user.

2.2. Problem Formulation

Given a user u, a track t, and contextual features $\mathbf{c}$, we predict $P(\text{is\_listened} = 1 \mid u, t, \mathbf{c})$. Predictions are evaluated using ROC AUC. The challenge lies in generalising from broad listening history (training) to the specific context of a Flow recommendation (test).

3. Q1: How Would We Win This Competition?

If our sole objective were to maximise the Kaggle leaderboard score, our strategy would centre on three pillars: (1) comprehensive feature engineering, (2) strong gradient-boosted tree models, and (3) rigorous validation that mirrors the test distribution.

3.1. Feature Engineering

We engineer 47 features in seven categories. All user-level and interaction-level statistics are computed exclusively from training data to prevent data leakage.

3.1.1. Temporal Features (9 features)

From the listening timestamp: `hour`, `day_of_week`, `day_of_month`, `month`, `is_weekend`, `is_late_night` (1–5 AM), `is_evening` (6–11 PM), `is_commute_time` (7–9 AM), and `time_of_day` (categorical). Listening behaviour varies strongly with temporal context—users are more selective during commutes than evening relaxation.

3.1.2. Release Features (7 features)

From the track release date: `release_year`, `release_month`, `release_decade`, `days_since_release`, `is_pre_release_listen`, `is_new_release` (within 30 days), and `track_age_category`. New releases exhibit higher engagement, while deep catalogue tracks tend to be intentionally sought out.

3.1.3. Duration Features (3 features)

`duration_minutes`, categorical `duration_category`, and `is_extended_track` (>5 minutes). Track length correlates with skip probability.

3.1.4. User Engagement Features (9 features)

Per-user statistics from training data: `user_listen_rate`, `user_skip_rate`, `user_session_count`, `user_total_listens`, `user_genre_diversity`, `user_artist_diversity`, `user_context_variety`, `user_engagement_segment`, and `user_engagement_score`. These capture the insight that some users are inherently more selective than others.

3.1.5. Target-Encoded Categoricals (3 features)

High-cardinality identifiers (`genre_id`, `artist_id`, `album_id`) are encoded using smoothed target encoding:

$$\text{enc}(c) = \frac{n_c \cdot \bar{y}_c + m \cdot \bar{y}}{n_c + m} \tag{1}$$

where n_c is the category count, $\bar{y}_c$ the category mean, $\bar{y}$ the global mean, and $m = 50$ the smoothing parameter. This converts identifiers into informative numerical features while regularising rare categories toward the global mean.

3.1.6. Item-Level Features (4 features)

Smoothed listen rates and log-transformed play counts for tracks (`media_listen_rate_smooth`, `media_play_count_log`) and artists (`artist_listen_rate_smooth`, `artist_play_count_log`).

3.1.7. User–Item Affinity Features (3 features)

`user_artist_affinity` (smoothed user-specific artist listen rate), `user_genre_affinity` (user-specific genre listen rate), and `user_knows_artist` (binary prior exposure flag). These approximate collaborative filtering within a feature-based framework.

3.2. Model Selection

3.2.1. XGBoost and LightGBM

Our primary models are XGBoost [3] and LightGBM [4], both gradient-boosted decision tree frameworks. Both use 500 estimators, max depth 7, learning rate 0.05, subsample 0.8, and L1/L2 regularisation. Early stopping with patience 30 prevents overfitting.

3.2.2. Neural Network

We also train a feedforward neural network (256–128–64 units) with batch normalisation [5], dropout [6], and Adam optimiser [7]. Despite being trained on the same 47 features, it underperforms tree-based models, consistent with findings that gradient boosting dominates on tabular data.

3.2.3. Ensemble

We combine XGBoost and LightGBM via weighted averaging: $\hat{p} = w \cdot \hat{p}_{\text{XGB}} + (1 - w) \cdot \hat{p}_{\text{LGB}}$, with w optimised by grid search on the validation set.

3.3. Results and Comparison with Competition Winners

Table 1 summarises our model progression.

3.3.1. Why Our Validation AUC Far Exceeds the Competition Scores

Our validation AUC of 0.935 dramatically exceeds the winning test score of 0.686. This discrepancy is *not* because our model is superior—it reflects a fundamental difference in evaluation methodology:

1. **Distribution mismatch**: Our validation set is a random sample from the same training distribution (general listening history). The competition test set consists of *first Flow recommendations*—a curated, algorithmically selected context that differs from organic listening behaviour.

Table 1
Model performance on a stratified 90/10 random split of the training data

Model	Features	Data Size	Val AUC
XGBoost v1 (baseline)	35	100K	0.872
XGBoost v2	47	7.5M	0.934
LightGBM v2	47	7.5M	0.935
Ensemble (XGB + LGB)	47	7.5M	0.935
Neural Network	47	7.5M	0.926
Competition winner (test)	*?*	*7.5M*	*0.686*

2. **Inflated validation from user overlap**: In our random split, the same users appear in both train and validation sets with many interactions each, allowing user-level features (listen rate, affinity scores) to be highly predictive. On the actual test set, the model must predict a *single* new context per user.
3. **Target leakage through item features**: Our item-level features (media listen rate, artist listen rate) are computed from the full training set. In a random split, validation items likely appear in training. On the held-out test set, items may be entirely new to the model.

3.3.2. What Competition Winners Likely Did Differently

The top teams (AUC 0.68–0.69) likely employed strategies we did not:

- **Temporal validation**: Splitting by time rather than randomly, since the test set represents future events. This gives a more realistic estimate of generalisation.
- **Sequence-aware models**: Modelling the *order* of listening events, not just aggregated statistics, since Flow's first recommendation depends on recent session context.
- **Context-specific features**: Engineering features that capture *why Flow selected this track*—e.g., novelty relative to the user's history, genre transition patterns, and session-level momentum.
- **Robust regularisation**: Avoiding over-reliance on user-item statistics that inflate validation but fail on distribution-shifted test data.

This analysis highlights a critical lesson: **a high validation score on a convenient split does not imply competition success**. Winning requires evaluation methodology that mirrors the test distribution.

4. Q2: What Would We Propose to Solve Deezer's Recommendation Problems?

If we worked at Deezer, we would not deploy our competition pipeline directly. A production recommendation system must address challenges that a Kaggle competition ignores entirely.

4.1. System Architecture

We propose a two-stage architecture common in industrial recommender systems [8]:

1. **Candidate generation**: A lightweight retrieval model selects $\sim$1000 candidate tracks from the full catalogue using approximate nearest neighbour search over learned user and item embeddings [9]. This stage prioritises recall.
2. **Ranking**: A more expressive model scores each candidate using rich features—similar to our competition features but augmented with real-time session context. The top-k items are presented after a diversity-aware re-ranking step.

This decomposition enables sub-second latency even with catalogues of tens of millions of tracks.

4.2. Cold-Start Problem

The competition conveniently provides training data for every test user. In production, new users and new tracks arrive constantly [10]:

- **New users**: Without listening history, we rely on onboarding preferences (selected genres/artists), demographic signals, and popularity-based recommendations. As interactions accumulate, collaborative signals gradually take over.
- **New tracks**: For tracks with zero play history, we use content-based features extracted from audio (tempo, energy, valence), artist metadata, and editorial tags. Our smoothed target encoding naturally handles this by falling back to the global mean when $n_c = 0$.
- **Exploration**: An ϵ-greedy strategy occasionally surfaces new or less popular tracks, accelerating data collection for cold items while preventing the system from converging on a narrow set of safe recommendations.

4.3. Diversity and User Experience

A system that only optimises listen probability will converge on a narrow set of popular, familiar tracks—the "filter bubble" [11]. Flow's value proposition is specifically a *mix of favourites and discoveries*. A production system must therefore:

- **Enforce diversity**: Re-rank to ensure variety in artist, genre, tempo, and mood within each session [12].
- **Balance familiarity and novelty**: Surface known artists alongside new discoveries, calibrated to each user's openness to exploration.
- **Ensure fairness**: Avoid systematic bias against niche artists or less popular genres, which harms both user experience and the artist ecosystem.

4.4. Scalability and Long-Term Engagement

- **Incremental updates**: User statistics and item features must update incrementally as new interactions arrive, not require full recomputation.
- **Real-time serving**: Tree-based models serve predictions in <1 ms; embedding-based retrieval uses approximate nearest neighbour libraries (e.g., FAISS) for sub-linear catalogue search.
- **User evolution**: Musical tastes change over time. Models should weight recent interactions more heavily and adapt to evolving preferences rather than anchoring on historical averages.
- **Session awareness**: Users listen in sessions with internal coherence (workout music vs. study music). Context-aware and session-based models [8] capture these patterns that pointwise classifiers miss.

5. Q3: Do These Two Solutions Overlap? Why or Why Not?

The competition solution and production system share some foundations but diverge in fundamental ways.

5.1. Where They Overlap

- **Feature engineering transfers**: Temporal context, user engagement statistics, and item popularity are valuable in both settings. The insight that user-artist affinity is the strongest predictor applies universally.
- **Leakage-free design**: Our strict separation of training and evaluation statistics mirrors the production requirement that models must never use future information.

Table 2
Competition vs. production: fundamental differences

Dimension	Competition	Production
Objective	Maximise AUC on fixed test set	Maximise long-term user retention and satisfaction
Cold start	Not a concern (all test users in training)	Critical—new users and tracks arrive daily
Diversity	Irrelevant to AUC	Essential for Flow's "favourites + discoveries" promise
Latency	Batch prediction, unlimited time	Must be $<$100 ms per request
Fairness	Not evaluated	Legal and ethical requirement
Feedback loop	Static dataset	Dynamic: predictions influence future user behaviour
Evaluation	Single offline metric	A/B tests, retention, session length, user surveys

- **Scalable model families**: XGBoost and LightGBM are widely used in production ranking systems at companies like Airbnb and Microsoft due to their speed, interpretability, and robustness [3, 4].
- **The ranking stage**: Our competition pipeline could serve directly as the second-stage ranker in the production architecture, scoring pre-selected candidates with rich features.

5.2. Where They Diverge

5.2.1. Accuracy Is Necessary but Not Sufficient

The most important divergence is in *what success means*. In the competition, a model that perfectly separates listens from skips wins—regardless of whether it recommends the same ten songs to every user. In production, such a model would destroy user engagement within weeks [11]. Real recommendation quality requires balancing accuracy, diversity, novelty, and fairness—none of which AUC captures.

5.2.2. Competition Tricks That Hurt in Production

Several techniques that boost competition scores are counterproductive in production:

- **Heavy target encoding**: Our target-encoded features are powerful but create feedback loops in production—popular items get higher encoded scores, receive more recommendations, and become even more popular, starving niche content.
- **Complex ensembles**: Our XGBoost + LightGBM ensemble adds marginal AUC improvement but doubles model maintenance, deployment complexity, and debugging difficulty. In production, a single well-tuned model is usually preferred.
- **Overfitting to distribution**: Competition models overfit to the specific train/test split. Production data is non-stationary—user tastes shift, new music releases change the catalogue, and seasonal patterns create distribution drift.

5.2.3. Production Requirements That Competitions Ignore

Conversely, several production-critical concerns are entirely absent from competitions:

- **Latency constraints**: Flow must generate recommendations in real time as users press play. Our batch pipeline has no such constraint.
- **Explainability**: When Flow recommends a track, users benefit from understanding why ("Because you liked Artist X"). Tree models offer feature importance, but not per-prediction explanations suitable for users.

- **Artist ecosystem**: Deezer must balance user satisfaction with fair exposure for artists and labels—a multi-stakeholder optimisation that no competition metric captures.

5.3. Synthesis

The competition pipeline provides a strong *component* for production—specifically, the ranking model that scores candidate tracks. However, it cannot function as a standalone system. Production deployment requires wrapping it in a candidate generation layer, diversity-aware re-ranking, cold-start fallbacks, and continuous monitoring infrastructure. The core lesson is that *winning a Kaggle competition and building a good recommender system are related but distinct objectives*, and understanding this distinction is essential for practitioners.

6. Conclusion

We presented a comprehensive approach to the Deezer skip prediction challenge, developing a 47-feature pipeline with an XGBoost/LightGBM ensemble that achieves 0.935 AUC on a random validation split. We critically examined why this far exceeds the competition winner's 0.686, attributing the gap to evaluation methodology: our random split inflates user-item feature utility, while the actual test set requires generalisation to a distribution-shifted recommendation context.

We contrasted this competition solution with a production architecture addressing cold start, scalability, diversity, and long-term engagement. Our comparison shows that competition solutions provide excellent ranking components but require fundamental architectural augmentation for production deployment. The strongest teams would recognise that *building a system users love requires more than maximising a single offline metric.*

Declaration on Generative AI

During the preparation of this work, the authors used AI-assisted tools for code development and report drafting. The authors reviewed and edited all content and take full responsibility for the publication's content.

Author Contributions

- **Vimerlin Govender**: [TODO: Describe contributions]
- **Meleknur Özgü**: [TODO: Describe contributions]
- **Xinmeng Song**: [TODO: Describe contributions]

References

[1] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, M. Elahi, Current challenges and visions in music recommender systems research, in: International Journal of Multimedia Information Retrieval, volume 7, 2018, pp. 95–116.

[2] Deezer, Deezer data science game 2017: Predicting user listening behavior, 2017. Online challenge, available at https://challengedata.ens.fr/.

[3] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785–794.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, volume 30, 2017, pp. 3146–3154.

[5] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, Proceedings of the 32nd International Conference on Machine Learning (2015) 448–456.
[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014) 1929–1958.
[7] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, Proceedings of the 3rd International Conference on Learning Representations (2015).
[8] P. Covington, J. Adams, E. Sargin, Deep neural networks for YouTube recommendations, Proceedings of the 10th ACM Conference on Recommender Systems (2016) 191–198.
[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, Proceedings of the 26th International Conference on World Wide Web (2017) 173–182.
[10] A. I. Schein, A. Popescul, L. H. Ungar, D. M. Pennock, Methods and metrics for cold-start recommendations, Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2002) 253–260.
[11] S. M. McNee, J. Riedl, J. A. Konstan, Being accurate is not enough: How accuracy metrics have hurt recommender systems, CHI'06 Extended Abstracts on Human Factors in Computing Systems (2006) 1097–1101.
[12] P. Castells, N. J. Hurley, S. Vargas, Novelty and diversity in recommender systems, in: Recommender Systems Handbook, Springer, 2015, pp. 881–918.