Language Models for Next-Track Music Recommendation

Maximilian Mayerl, Michael Vötter, Eva Zangerle, Günther Specht
Databases and Information Systems
Department of Computer Science
University of Innsbruck, Austria
{firstname.lastname}@uibk.ac.at

ABSTRACT

Next-track music recommendation is the task of automatically determining the next song to play in a music listening session. Almost all music streaming platforms on the web provide their users with such a feature today. In this work, we propose the use of language modeling techniques for this task and investigate how well these techniques perform in the context of popular and also more diverse music. For this, we implement two basic language models, one based on n-grams and the other based on a recurrent neural network. We evaluate these models on two datasets, one limited to popular music and one consisting of more diverse tracks. Further, we also compare them with a nearest-neighbor model. Our results suggest that language models perform well in the context of popular music and can be used both as a basis for more sophisticated models and as a strong comparative baseline.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing

General Terms

Music Recommendation, Playlist Continuation

Keywords

Music Recommendation, Playlist, Next-Track, Algorithm, N-Gram, Language Model

1. INTRODUCTION

Nowadays, almost all music streaming services on the web provide their users with a feature to automatically determine the next song to listen to, based on the user's current listening session and possibly their past listening behavior and general preferences. This feature is an example of a recommender system, and the specific problem is known as next-track recommendation.

A single listening session (or playlist) X_i is given as a temporally ordered sequence of tracks, i.e., $X_i = [t_1, t_2, ..., t_n]$. The task of a next-track recommender is then to take the current listening session of the user and predict the most likely—in other words, most appropriate—next track to continue the session with. Effectively, such a recommender can be considered as a model learning a probability distribution $p(X_j)$ over all possible listening sessions and the task of recommending the most likely continuation as finding the solution to

$$\texttt{continuation}(X_i) = \texttt{argmax}_{t \in T} p(X_i + t) \quad (1)$$

where T is the set of all available tracks and $X_i + t$ is the listening session resulting from appending track t to session X_i, i.e., $X_i + t = [t_1, t_2, ..., t_n, t]$. This can further be generalized to not only determine the single most likely continuation, but a list of the n most likely continuations.

Many different approaches for solving this task have been proposed over the years. Quadrana et al. [11] provide an overview of approaches for sequence-aware recommendation, which can be regarded as a generalized form of the next-track recommendation problem. These approaches include pattern mining [14], Markov models [9], and recurrent neural networks [17]. Other approaches are based on autoencoder networks [13, 18]. Another popular technique, most often used as a comparative baseline, are nearest-neighbor models [2, 6, 5, 7].

Existing approaches can be divided into those that view listening sessions as pure lists of track identifiers, and those that make use of additional features of tracks, including track metadata, audio features or lyrical features. In this work, we take the former approach—i.e., we do not make use of any features of the tracks in a session. Specifically, we regard listening sessions as sentences, formed from words given by track identifiers, as has been done before for example by McFee and Lanckriet [9]. We then train a language model on those sentences and use that to predict continuations for other sentences (listening sessions). Language models are statistical models that learn a probability distribution $p(s)$ over the set of all possible sentences S of a language. Note that this is exactly the kind of distribution we need for the next-track recommendation task. We hypothesize that listening histories and playlists—at least those of users focusing on popular music, due to them having a relatively limited "vocabulary" but at the same time a relatively large number

31^{st} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 11.06.2019 - 14.06.2019, Saarburg, Germany.

of patterns this vocabulary appears in—have a structure similar to natural language and therefore lend themselves to the use of language models.

With this work, we attempt to answer the following research questions:

- **RQ1:** Can language models perform well for the next-track recommendation task?
- **RQ2:** Are there differences in performance for language models between different types of listening sessions/playlists and if yes, can we explain them?

The remainder of this paper is structured as follows: In Section 2, the different language models we propose for solving the next-track recommendation problem are explained. Section 3 details the experiments we performed to evaluate our approaches as well as the datasets we used for the evaluation. In Section 4, the results of these experiments are given and discussed. Lastly, we give a conclusion of those results in Section 5.

2. METHODS

In this section, the language models we use to tackle the next-track recommendation problem are explained. We implemented two different types of language models. The first model, explained in Section 2.1, is an n-gram based model using backoff smoothing. The second model, presented in Section 2.2, is a neural language model using a recurrent architecture.

2.1 N-Gram Language Model

An n-gram is a subsequence of length n of a longer sequence. Consider for example the listening session $X = [t_1, t_2, t_3, t_4]$. This session can be decomposed into three 2-grams: $[t_1, t_2]$, $[t_2, t_3]$, and $[t_3, t_4]$. An n-gram language model is a language model which makes the assumption that a given word in a sentence only depends on the $n-1$ previous words in the sentence, i.e.,

$$p(t_i|t_1, t_2, ..., t_{i-1}) = p(t_i|t_{i-(n-1)}, ..., t_{i-1}) \tag{2}$$

In other words, such a language model can be trained on all n-grams of a set of sentences instead of the complete sentences. The probabilities can then be determined as

$$p(t_i|t_{i-(n-1)}, ..., t_{i-1}) = \frac{c(t_{i-(n-1)}, ..., t_{i-1}, t_i)}{c(t_{i-(n-1)}, ..., t_{i-1})} \tag{3}$$

where $c(s)$ is the number of occurences of sequence s in the training set. For real language models, these probabilities are often discounted slightly to "set aside" probability mass for words that were not encountered during training. We make the simplifying assumption that all tracks available to the system will be seen during training and therefore don't use discounting. Such models have the advantage of being very easy and fast to train even on very large data sets and are known to work remarkable well considering their simplicity.

One limitation of a simple n-gram model is that it is limited to a fixed sequence length n. This limitation can be overcome by training multiple models for different values of n and then combining them via a procedure called *backoff*. In a simple backoff model, the probability $p(t_i)$ is first determined using the largest value for n for which a model was trained. If this probability is 0, because the sequence $t_{i-(n-1)}, ..., t_{i-1}, t_i$ was never encountered in the training data, the probability is instead determined using the next-lower value for n for which a model was trained etc.

For our approach, we decided to train n-gram models for $n \in \{1, 2, 3, 4\}$. We also implement a backoff model, for which we further introduce a parameter k_n, inspired by Katz' backoff model [8], that gives a minimum threshold for how often a given sequence must have been encountered in the training data to be considered. The probabilities in our model are then finally given by

$$p_n(t_i|t_{i-(n-1):i-1}) = \begin{cases} \frac{c(t_{i-(n-1):i})}{c(t_{i-(n-1):i-1})} & c(t_{i-(n-1):i}) \geq k_n \\ p_{n-1}(t_i|t_{i-(n-2):i-1}) & \texttt{otherwise} \end{cases} \tag{4}$$

where $t_{a:b}$ is the sequence of words t_a to t_b.

This was implemented in Python. For the implementation, a further simplification is possible. Since we don't actually need probability values, but only want to output candidate continuations in order of probability, it is possible to simply save lists of all continuations for prefixes of length $n-1$, ordered by frequency, and draw predictions from these lists.

2.2 Neural Language Model

The second kind of language model we propose is a neural language model based on a recurrent architecture. A neural language model is a neural network which produces a vector of probability values as its output. The vector has the same number of elements as there are distinct words in the modeled language (i.e., the language's vocabulary size). Every element of the vector thus corresponds to one word, and the value of that element is the probability that this word is the next work in a text, given the history of previous words that the network got as its input. In other words, a neural language model learns the probability distribution

$$p(\mathbf{t}_i|t_{i-n}, ..., t_{i-1}) \tag{5}$$

For our approach, we use a recurrent architecture. An illustration of this architecture is given in Figure 1. Our network takes a sequence of track identifiers as input. The first layer of the network calculates a semantic vector space embedding of the inputs. This embedding is then fed into a recurrent layer using GRU cells [3] with tanh activation. This layer is effectively responsible for recognizing the relevant patterns in the input sequence. Both of these layers consist of $\sqrt{|T|}$ units, where T is the set of all tracks; preliminary experiments have shown that this number of units performs well. The recurrent layer utilizes variational dropout as described by Gal and Ghahramani [4], on both the input and recurrent connections. Finally, the network has a dense layer of $|T|$ units, with softmax activation to obtain probabilities as output.

The network was implemented in Python using Keras[1]. For training, `categorical_crossentropy` was used as loss function and `Adam` as optimizer.

For training the network, the listening sessions in the training data are split into subsequences of a predefined length, and all of those subsequences are fed to the network for training, using the last element of the sequence as the target value and the elements before that (the *prefix*) as the input to the network. The length of those subsequences is controlled via a hyperparameter specifying how long the prefixes

[1]https://www.keras.io/

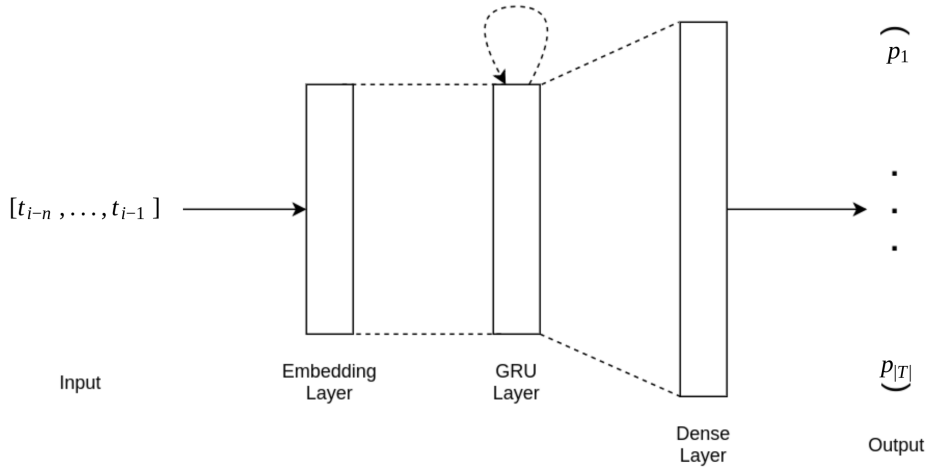

Figure 1: The architecture of the neural language model.

should be (the subsequences are then one element longer then the prefix).

3. EXPERIMENTS

In this section, the experiments we conducted to test the applicability of language models for the next-track recommendation task are described. First, in Section 3.1, the datasets we used for the experiments are presented. Section 3.2 discusses the nearst-neighbor baseline we use to compare our models against. After that, we explain our experimental setup in Section 3.3.

3.1 Datasets

As we attempt to assess the applicability of language models to the next-track recommendation problem for different types of playlists/listening sessions, we use two different datasets for our experiments. As our results should also be comparable to already existing work, we decided to use data from sources that are used extensively in the literature [9, 10, 1, 6]. For these reasons, we decided to use data obtained from the two music playforms Last.fm[2] and Art of the Mix[3] for conducting the experiments.

The Last.fm dataset we used is based on the LFM-1b [12] dataset and we relied on listening sessions extracted by Jacob Winder [16] as follows. First, listening sessions are constructed from singular listening events such that if two listening events belong to the same user and are no longer than 30 minutes apart they are considered to belong to the same listening session. Also, if the same track occurs more than once in direct succession, all the repeat occurences are dropped. After further removing all the listening sessions with only one track, this gives a dataset of approximately 62 million sessions.

We then filtered this dataset as follows to make it meet our purposes. First, we dropped all sessions with fewer than three tracks. Then, we extracted a chunk of the first three million remaining sessions. Lastly, we dropped all the sessions containing tracks that occur in fewer than 840 sessions in this chunk. This has two benefits: First, it restricts the dataset to listening sessions consisting of popular music only, which is exactly the type of sessions for which we hypothesize that language models should work well. Second, it makes the dataset small enough so that the hardware available to us can handle training our models on it.

As the second dataset, we used the AotM-2011 dataset [10] as published by Vall et al. [15]. As Art of the Mix is a platform for music enthusiasts, the playlists users upload to it tend to be more diverse on average than listening sessions on Last.fm. Therefore, this dataset is not limited to popular music and contains many tracks that occur only a few times. It is therefore a good choice for testing the inverse of our hypothesis, namely that language models should not work so well on such playlists. More details about the makeup of the two datasets are given in Table 1.

Table 1: Detailed dataset information.

Dataset	Playlists	Tracks	Avg. Occurences per Song
LFM-1b	20,824	2,673	56.19
AotM-2011	2,715	12,355	2.05

3.2 kNN Baseline

To have a baseline to compare the performance of our models against, we also train and evaluate a nearest-neighbor model (kNN) [2] on the same datasets as our models. We chose this baseline because nearest-neighbor models are prevalent throughout existing literature [2, 6, 5, 7].

A kNN is a simple supervised machine learning algorithm. It works by finding—using some arbitrary distance metric—the k nearest neighbors in feature space to a given data point and then determining the class label (in case of classification) or the value (in case of regression) for that given data point based on the labels/values of those neighbors. In the case of classification, this is often done by (weighted) majority vote among the neighbors.

To implement this, we used the Python package implicit[4]. This package provides different distance metrics for determining neighbors. We used *cosine similary* and *item-item* for the baseline, as preliminary experiments showed them to

[2]https://www.last.fm/
[3]http://www.artofthemix.org
[4]https://pypi.org/project/implicit/0.3.8/

Table 2: Results for the LFM-1b dataset.

Model	R@1	R@5	R@20	MRR@5	MRR@20
N-Gram	**0.757**	**0.843**	0.882	**0.789**	**0.794**
Neural	0.733	0.820	0.878	0.766	0.776
kNNi	0.619	0.786	**0.883**	0.619	0.695
kNNc	0.635	0.799	0.869	0.635	0.708

Table 3: Details for language models on LFM-1b.

Model	R@1	R@5	R@10	R@20	R@100
N-Gram	**0.757**	**0.843**	**0.871**	**0.882**	0.886
Neural	0.733	0.820	0.854	0.878	**0.911**

work best. As with the language models, the kNN did not make use of any audio features; for the similarity computations, the list of song identifiers in a listening session was used as the vector representation of that listening session.

3.3 Setup

We performed one experiment for every combination of model (n-grams, neural language model, and kNN baseline) and dataset (LFM-1b, AotM-2011). This gives a total of six experiments.

For each experiment, we performed a grid search over a set of reasonable hyperparameter values to find the best parameter settings for every model-dataset combination. To eliminate possible bias resulting from a single, fixed train-test split, we used 5-fold cross validation for the grid search. To make results reproducible, a fixed seed was used for the cross validation. For the n-gram model, we decided to always train the model on n-grams ranging from length one to four and grid search over the backoff tresholds k_n ranging from zero to six. For the neural language model, we performed the grid search over the dropout rate (0.2, 0.4), and the prefix length (2, 3, and 4). For the kNN baseline, we performed a grid search over k, the number of neighbors, with possible values of 10, 20, 50, 100, 200, and 300.

In every experiment (and every fold within it) the model is first trained on the training portion of the data. After that, the evaluation is performed as follows. For every listening session in the test portion of the data, the last track of the session is removed and the trained model is asked to produce a ranked list of candidate continuations for the shortened session. Based on these predictions and the known real continuations—those that were removed—we then calculate the metrics recall (R) and mean reciprocal rank (MRR), taking into account only the first, the top five and the top twenty of the predicted candidate continuations, respectively.

4. RESULTS

In this section, we present and discuss the results of our experiments. The results for the evaluation on the LFM-1b dataset are summarized in Table 2. In this table, kNNi is the kNN model with item-item distance, using $k = 20$, and kNNc is the same with cosine similary, using $k = 50$. We can see that both language models (n-grams and the neural model) work well on this dataset and significantly outperform the kNN baseline in all but one metric (Recall@20). Thus, we can already positively answer RQ1—language models can indeed perform well for the next-track recommendation task.

Table 4: Results for the AotM-2011 dataset.

Model	R@1	R@5	R@20	MRR@5	MRR@20
N-Gram	0.021	0.029	0.044	0.024	0.025
Neural	0.017	0.024	0.034	0.019	0.020
kNNi	**0.027**	**0.044**	**0.062**	**0.033**	**0.035**
kNNc	0.018	0.024	0.036	0.020	0.022

The better performance for recall at lengths 1 and 5 suggest that the language models are better than kNNs at learning and correctly ranking the exact patterns in people's music listening behavior—at least for people leaning heavily towards popular music—and are therefore more likely to find the correct continuation as one of their first predictions. This is further supported by the results for the mean reciprocal rank (MRR), where the language models beat kNNs event at length 20, suggesting that they tend to rank the correct continuation higher on average. For recall at length 20, the kNN model using item-item distance shows the same performance as the n-gram model and slightly better performance than the neural model. This suggests that the kNN model does indeed learn the relevant patterns, but is not as good at ranking them correctly.

Between the two language models, the n-gram model consistently outperforms the neural one on this dataset, despite its simplicity. Looking at the different metrics, we identify an apparent trend—the gap between the n-gram model and the neural model seems to become smaller at larger lengths. To further investigate this, we also calculated the recall for both models on the LFM-1b dataset for lengths 10 and 100. The results for this are given in Table 3. They support this trend and show that the neural model even overtakes the n-gram model at some point and outperforms it at length 100, suggesting that the neural model is better at detecting patterns, but worse at ranking them.

The results for the experiments on the AotM-2011 dataset show a different picture. They are summarized in Table 4. Here, the kNN model using item-item distance consistently outperforms both language models. Due to this, we can partially answer RQ2—there apparently are differences in performance for language models on different types of listening sessions. The fact that language models perform better than nearest-neighbor models for our filtered Last.fm dataset but worse for the AotM-2011 dataset supports our hypothesis that language models work well in the context of users leaning towards popular music, where we deal with a limited number of frequently occurring patterns, but not so well for scenarios where users listen to more diverse music.

5. CONCLUSION

In this work, we proposed to use common language modeling techniques to solve the next-track music recommendation task. We were able to show that these approaches perform well in the context of popular music, but not so well for scenarios in which users prefer playlists that are more diverse. Both of the language models we proposed are simple and easy to train and are therefore also well suited for use as a baseline for the development of more sophisticated models.

For future work, we will have to perform further experiments to test our hypothesis—namely that language models

work well for popular music contexts—by evaluating our models on additional datasets obtained from different sources. Another possible direction for future work is the use of more sophisticated language models for the task. We believe that a language model that is better optimized for this specific task could show significantly better performance still.

6. REFERENCES

[1] G. Bonnin and D. Jannach. Evaluating the Quality of Playlists Based on Hand-crafted Samples. In *Proc. ISMIR*, pages 263–268, 2013.

[2] G. Bonnin and D. Jannach. Automated Generation of Music Playlists: Survey and Experiments. *ACM Computing Surveys (CSUR)*, 47(2):26, 2015.

[3] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] Y. Gal and Z. Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027, 2016.

[5] D. Jannach, I. Kamehkhosh, and G. Bonnin. Biases in Automated Music Playlist Generation: A Comparison of Next-Track Recommending Techniques. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, pages 281–285. ACM, 2016.

[6] D. Jannach, L. Lerche, and I. Kamehkhosh. Beyond Hitting the Hits: Generating Coherent Music Playlist Continuations with the Right Tracks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 187–194. ACM, 2015.

[7] I. Kamehkhosh and D. Jannach. User Perception of Next-Track Music Recommendations. In *Proceedings of the 25th conference on user modeling, adaptation and personalization*, pages 113–121. ACM, 2017.

[8] S. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401, 1987.

[9] B. McFee and G. R. Lanckriet. The Natural Language of Playlists. In *ISMIR*, volume 11, pages 537–541, 2011.

[10] B. McFee and G. R. Lanckriet. Hypergraph Models of Playlist Dialects. In *ISMIR*, volume 12, pages 343–348. Citeseer, 2012.

[11] M. Quadrana, P. Cremonesi, and D. Jannach. Sequence-aware Recommender Systems. *ACM Computing Surveys (CSUR)*, 51(4):66, 2018.

[12] M. Schedl. The LFM-1b Dataset for Music Retrieval and Recommendation. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 103–110. ACM, 2016.

[13] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. AutoRec: Autoencoders Meet Collaborative Filtering. In *24th Intl. Conf. on World Wide Web*, WWW, pages 111–112. ACM, 2015.

[14] R. Turrin, A. Condorelli, P. Cremonesi, R. Pagano, and M. Quadrana. Large Scale Music Recommendation. In *Workshop on Large-Scale Recommender Systems (LSRS 2015) at ACM RecSys*, 2015.

[15] A. Vall, H. Eghbal-zadeh, M. Dorfer, M. Schedl, and G. Widmer. Music Playlist Continuation by Learning from Hand-curated Examples and Song Features: Alleviating the Cold-Start Problem for Rare and Out-of-Set Songs. In *Proceedings of the 2Nd Workshop on Deep Learning for Recommender Systems*, DLRS 2017, pages 46–54. ACM, 2017.

[16] J. Winder. Session-based Track Embedding for Context-aware Music Recommendation. Master's thesis, University of Innsbruck, 2018.

[17] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan. A Dynamic Recurrent Model for Next Basket Recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 729–732. ACM, 2016.

[18] S. Zhang, L. Yao, X. Xu, S. Wang, and L. Zhu. Hybrid Collaborative Recommendation via Semi-AutoEncoder. In *Intl. Conf. on Neural Information Processing*, ICONIP, pages 185–193. Springer, 2017.