Using Adversarial Autoencoders for Multi-Modal Automatic Playlist Continuation

Iacopo Vagliano
ZBW – Leibniz Information Centre for Economics
Kiel, Germany
i.vagliano@zbw.eu

Lukas Galke
Kiel University
Germany
lga@informatik.uni-kiel.de

Florian Mai
Kiel University
Germany
stu96542@informatik.uni-kiel.de

Ansgar Scherp
University of Stirling
Scotland UK
ansgar.scherp@stir.ac.uk

ABSTRACT

The task of automatic playlist continuation is generating a list of recommended tracks that can be added to an existing playlist. By suggesting appropriate tracks, i. e., songs to add to a playlist, a recommender system can increase the user engagement by making playlist creation easier, as well as extending listening beyond the end of current playlist. The ACM Recommender Systems Challenge 2018 focuses on such task. Spotify released a dataset of playlists, which includes a large number of playlists and associated track listings. Given a set of playlists from which a number of tracks have been withheld, the goal is predicting the missing tracks in those playlists. We participated in the challenge as the team *Unconscious Bias* and, in this paper, we present our approach. We extend adversarial autoencoders to the problem of automatic playlist continuation. We show how multiple input modalities, such as the playlist titles as well as track titles, artists and albums, can be incorporated in the playlist continuation task.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Computing methodologies → Neural networks**; *Learning from implicit feedback*;

KEYWORDS

music recommender systems; neural networks; adversarial autoencoders; multi-modal recommender; automatic playlist continuation

ACM Reference Format:
Iacopo Vagliano, Lukas Galke, Florian Mai, and Ansgar Scherp. 2018. Using Adversarial Autoencoders for Multi-Modal Automatic Playlist Continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18), October 2, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3267471.3267476

1 INTRODUCTION

The task of automatic playlist continuation is adding one or more tracks, i. e., songs, to a playlist while keeping the same target characteristics of the original playlist [16]. Given a set of playlist features and initial tracks, the system generates a list of recommended tracks that can "continue" that playlist. This is particularly interesting in services such as Spotify[1]. Automatic playlist continuation is becoming more and more important as user tend to prefer being recommended musical experiences rather than single tracks [15]. This is a challenging task, because it is unclear to which extent and when the sequential order of the tracks helps to create better models for recommendation and it is difficult to assess the quality of a playlist [16].

Recent advances in autoencoders on images have shown that adversarial regularization can improve the performance of autoencoders [11]. Adversarial autoencoders [11] are not only trained to reconstruct the input, but also to match the code with a selected prior distribution. In a previous study [6], we successfully transferred adversarial autoencoders to recommendation tasks in the context of citations and subject labels and we showed that smoothness on the code aids autoencoders to reconstruct highly sparse item vectors for citation and subject labels recommendation.

In this paper, we analyze whether adversarial autoencoders can be applied to automatic playlist continuation in the context of the ACM Recommender Systems Challenge 2018 [4]. In our prior work [6], we did not yet exploit item attributes. In this work, we explore the potential of aggregating item attributes (track title, album title, artist name) to the playlist-level. This can be useful because playlist titles may be not as meaningful as titles are, e. g., for research papers [6].

The challenge focuses on music recommendation, specifically on automatic playlist continuation. Spotify released the Million Playlist Dataset[2] consisting of a large number of playlist titles and associated track listings. The evaluation set contains playlists from which a number of tracks have been withheld. The task is predicting the missing tracks in those playlists.

Traditionally, the recommendation problem is modeled as the prediction of missing ratings in a $\boldsymbol{U} \times \boldsymbol{I}$ matrix with set of users $\boldsymbol{U}$ and set of items $\boldsymbol{I}$ (matrix completion). In our case, we consider

[1]https://www.spotify.com
[2]http://recsys-challenge.spotify.com/dataset

Table 1: Notation

Symbol	Description
$\mathbb{P}$	Set of m playlists
$\mathbb{T}$	Set of n tracks
$\boldsymbol{X} \in \{0,1\}^{m \times n}$	Sparse ratings matrix
$\boldsymbol{S} \in \mathbb{R}^{m \times d}$	Side information of the playlist
$\boldsymbol{x}, \boldsymbol{s}$	Row vectors of $\boldsymbol{X}$ or $\boldsymbol{S}$, respectively
$[\boldsymbol{x}; \boldsymbol{s}]$	Concatenation of vectors $\boldsymbol{x}$ and $\boldsymbol{s}$
$\bowtie$	Natural join (on document identifiers)
$\boldsymbol{I}$	Identity matrix

a matrix $\boldsymbol{P} \times \boldsymbol{T}$ which indicates the occurrences of tracks in the playlists, where $\boldsymbol{P}$ is the set of playlist and $\boldsymbol{T}$ is the set of tracks. We aim to predict the missing occurrences.

We show how adversarial autoencoders can be applied to the challenge task and how multiple input modalities can be incorporated. We consider metadata like the playlist titles as well as track titles, artists and albums, as content-based features. We performed various experiments for the challenge task to study how adversarial autoencoders perform while exploiting metadata along with the partial list of tracks. Our results show that adversarial autoencoders consistently outperform other models and that their capability of incorporating multiple input modalities increase the performance of the models.

The remainder of this paper is organized as follows. In Section 2, we formally state the problem. We introduce the employed models in Section 3 and describe the experiments conducted in Section 4. We conclude in Section 5.

2 PROBLEM STATEMENT

In the following, we provide a formal problem statement for the considered automatic playlist continuation task. The playlists can be considered users in a traditional recommendation scenario, while the items are the tracks. This means that we regard tracks as a bipartite graph, as shown in Figure 1.

Given a set of m playlists $\mathbb{P}$ and a set of n tracks $\mathbb{T}$, the typical recommendation task is to model the spanned space, $\mathbb{P} \times \mathbb{T}$. We consider a sparse matrix $\boldsymbol{X} \in \{0,1\}^{m \times n}$, where X_{jk} means that the track k is included in the playlist j. We only consider the binary occurrence of a track in a playlist, i. e. X_{jk} is 1 if the track k appears one or more times in the playlist j, and we normalize by the number of tracks in the playlist by the L1 norm.

For training, the models are supplied with the complete occurrences of the tracks in the playlists, $\boldsymbol{X}_{\text{train}} = \mathbb{P}_{\text{train}} \bowtie \boldsymbol{X}$, along with side information, $\boldsymbol{S}_{\text{train}} = \mathbb{P}_{\text{train}} \bowtie \boldsymbol{S}$. As side information, we use the title of the playlist, as well as the aggregated artists names, track names, and album names of the playlist's tracks. At test time, the representations $\boldsymbol{X}_{\text{test}}$ and $\boldsymbol{S}_{\text{test}}$ are obtained analogously.

To conduct preliminary experiments, we create an artificial test set that is similar to the setting of our own prior work [6]: we remove randomly selected items in $\boldsymbol{X}_{\text{test}}$ by setting one non-zero entry in each row to zero. We denote this test set by $\tilde{\boldsymbol{X}}_{\text{test}}$. For the purpose of the challenge, the real test set is provided by the organizers. However, we artificially construct a development set

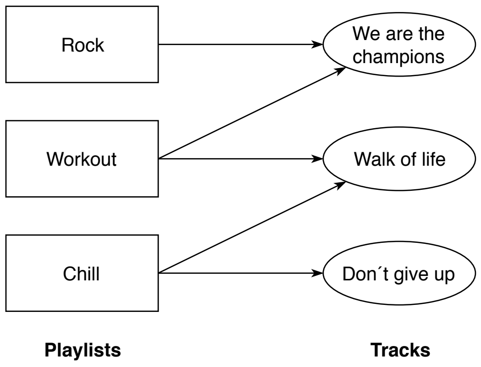

Figure 1: Exemplary bipartite graph of tracks occurrences in playlists.

that is representative for the challenge test set (more details are provided in Section 4).

The model ought to predict values $\boldsymbol{X}_{\text{pred}} \in [0,1]^{m_{\text{test}} \times n}$, given the test set $\tilde{\boldsymbol{X}}_{\text{test}}$ along with the additional information $\boldsymbol{S}_{\text{test}}$. The goal is that the correct tracks (continuations) are highly ranked in $\boldsymbol{X}_{\text{pred}}$.

3 OUR APPROACH

In the following, we describe the employed models. We initially developed and applied them to citation recommendation and subject indexing tasks [6]. We also exploited two baselines based on item co-occurrence in the preliminary experiments. We first present those baselines, then we introduce the multi-layer perceptron as a building block for the two autoencoder variants. Finally, we briefly describe undercomplete autoencoders and we show how they can be extended to obtain adversarial autoencoders and how side information, such as the playlist title, track titles, artist names, and album titles can be incorporated in both models.

Item Co-Occurrence. As one baseline, we consider the co-occurrence score [17] that is purely based on track co-occurrence. The rationale is that two tracks, which have occurred together in the same playlist in the past, are more likely to occur together in the future. Given training data $\boldsymbol{X}_{\text{train}}$, we compute the full item co-occurrence matrix $\boldsymbol{C} = \boldsymbol{X}_{\text{train}}{}^{T} \cdot \boldsymbol{X}_{\text{train}} \in \mathbb{R}^{n \times n}$. At prediction time, we obtain the scores by aggregating the co-occurrence values via matrix multiplication $\boldsymbol{X}_{\text{test}} \cdot \boldsymbol{C}$. On the diagonal of $\boldsymbol{C}$, the (squared) occurrence count of each item is retained to model the prior probability.

Singular Value Decomposition. Singular value decomposition (SVD) is an approach that factorizes the co-occurrence matrix of items $\boldsymbol{X}^{T} \cdot \boldsymbol{X}$. Caragea et al. showed that SVD can be successfully used for recommendation [3]. We extend SVD by the capability of incorporating side information. We concatenate the textual features as TF-IDF weighted bag-of-words with the items and perform singular value decomposition on the resulting matrix. To obtain predictions, we only use those indices of the reconstructed matrix that are associated with items.

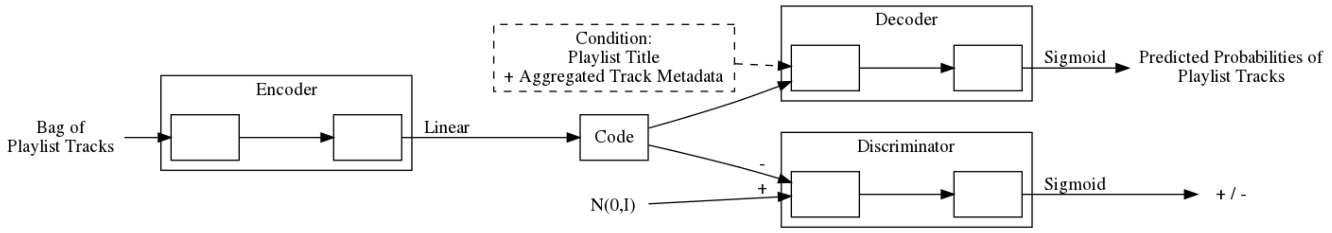
Bag of Playlist Tracks
Encoder
Linear
Code
Condition: Playlist Title + Aggregated Track Metadata
Decoder
Sigmoid
Predicted Probabilities of Playlist Tracks
N(0,I)
-
+
Discriminator
Sigmoid
+ / -

Figure 2: Adversarial autoencoder for item-based recommendations [6]. Each edge resembles a parametrized mapping $f(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$ with activation function f and parameters W, b. When not labeled differently, the activation function is rectified linear followed by dropout.

Multi-Layer Perceptron. A multi-layer perceptron (MLP) is a fully-connected feed-forward neural network with one or multiple hidden layers. The output is computed by consecutive applications of $\boldsymbol{h}^{(i)} = f(\boldsymbol{h}^{(i-1)} \cdot \boldsymbol{W}^{(i)} + \boldsymbol{b}^{(i)})$ with f being a nonlinear activation function. In the description of the following models, we abbreviate a two hidden-layer perceptron module by MLP-2. This MLP-2 module is not only used as a building block for subsequent architectures, but also as a full model that only operates on the playlist titles. In this case, we optimize binary cross-entropy $\mathrm{BCE}(x, \text{MLP-2}(s))$, where the playlist titles s are used as input and tracks x as target outputs. We chose to operate on an TF-IDF weighted embedded bag-of-words representation [7] for a fair comparison with the autoencoder variants, which are described below.

Undercomplete Autoencoders. The general concept of an autoencoder (AE) involves two components: the encoder enc and the decoder dec. The encoder transforms the input into a hidden representation (the code) $z = \mathrm{enc}(x)$. Then the decoder reconstructs the input from the code $r = \mathrm{dec}(z)$. The two components are jointly trained to minimize the loss function $BCE(x, r)$. To avoid learning to merely copy the input x to the output r, the code has a lower dimensionality (undercomplete). Autoencoders are trained to capture the most important factors of variation for reconstruction [2].

For both the encoder and the decoder we chose an MLP-2 module, such that the model function is $r = \text{MLP-2}_{\text{dec}}(\text{MLP-2}_{\text{enc}}(x))$. When side information s is available, we supply it as additional input to the decoder $r = \text{MLP-2}_{\text{dec}}([\text{MLP-2}_{\text{enc}}(x); s])$. We embed the textual features into a lower dimensional space by using pre-trained word embeddings [13]. The rationale here is that the rather low code dimension is not overwhelmed by the high amount of vocabulary terms. For a fair comparison of the models, also the MLP described above is supplied the same text representation as input. More precisely, we employ a TF-IDF weighted bag of embedded words representation which has proven to be useful for information retrieval [7]. The usage of side information in an undercomplete autoencoder is comparable to the approach by Barbieri et al. [1]. A minor difference is that we supply the side information (titles, artists and albums) only to the decoder, yet use two hidden layers for both the encoder and the decoder to enable a fair comparison to the adversarial variant, which is described below.

Adversarial Autoencoders. We extend the work of Makhzani et al. on adversarial autoencoders (AAE) [11], who combine generative adversarial networks [8] with autoencoders. The autoencoder component reconstructs the sparse item vectors, while the discriminator distinguishes between the generated codes and samples from a selected prior distribution (see Figure 2). Hence, the distribution of the latent code is shaped to match the prior distribution. We hypothesize that the latent representations learned by distinguishing the code from a smooth prior lead to a model that is more robust to sparse input vectors than undercomplete autoencoders. The rationale is that smoothness is a main criterion for good representations that disentangle the explanatory factors of variation [2].

Formally, we first compute $h = \text{MLP-2}_{\text{enc}}(x)$ and $r = \text{MLP-2}_{\text{dec}}(h)$ and then update the parameters of the encoder and the decoder with respect to binary cross-entropy $\mathrm{BCE}(x, r)$. Hence, in the regularization phase, we draw samples $z \sim \mathcal{N}(0, \boldsymbol{I})$ from independent Gaussian distributions matching the size of h. The parameters of the discriminator $\text{MLP-2}_{\text{disc}}$ are then updated, to minimize $\log \text{MLP-2}_{\text{disc}}(z) + \log(1 - \text{MLP-2}_{\text{disc}}(h))$ [8]. Finally, the parameters of the encoder are updated to maximize $\log \text{MLP-2}_{\text{disc}}(h)$, such that the encoder is trained to fool the discriminator. As a result, the encoder is jointly optimized for matching the prior distribution and for reconstruction of the input [11]. At prediction time, we discard the discriminator and perform one pass of encoding and decoding to obtain a prediction over all considered items.

Application to playlist continuation. In the case of playlist continuation, the considered items are tracks. On the input side, the tracks are repsented by an L1-normalized bag-of-tracks vector. The desired output is an estimated probability $p(\text{track}|\text{playlist})$ for each track. Since the number of distinct tracks is large, we limit the amount of considered tracks to the n_{tracks} most frequent tracks, which is controlled by a hyperparameter. After the probabilities are estimated, we eliminate (for all methods) those tracks that were already present in the original playlist, before the tracks are ranked by descending probability.

We incorporate side information in the model, namely playlist titles, track titles, artists, and albums. To this end, we first aggregate this information by generating a string with all these metadata for each track in the playlist. Similar to our prior work on scientific documents [6], the side information is concatenated with the code of the autoencoders, before the decoder uses it to make a prediction. In this way, during training, the models' parameters are optimized to draw the necessary information for prediction either from the already prevalent track set or the supplied side information.

As an example for a prediction step, we consider a "Workout" playlist with the track "Walk of life" and the desired continuation by "We are the champions" (see Figure 1). In this case, the input track set consists only of the single track "Walk of life", whose (one-hot) vector is mapped to the code. The code is then concatenated with the bag-of-words representation composed of the playlist title "Workout" along with all words of the already prevalent tracks, in this case: "Walk", "of", "life". The decoder takes both the code and the side information as input and estimates a high probability for the track "We are the champions".

4 EXPERIMENTS

The goal of the challenge is automatic playlist continuation. Given a set of playlist features and some initial tracks, the system generates a list of recommended tracks that can be added to the playlist. The input is a user-created playlist, represented by some playlist metadata (see Section 4.1) and a list of the K tracks in the playlist, where K can be equal to 0, 1, 5, 10, 25, or 100.

The output is a list of 500 recommended candidate tracks, ordered by relevance in decreasing order. It is necessary to cope with playlists for which no initial seed tracks are given. To assess the performance of a submission, the output track predictions are compared to the ground truth tracks from the original playlist.

4.1 Dataset

The Million Playlist Dataset[3] contains 1,000,000 playlists created by users on the Spotify platform between January 1, 2010 and December 1, 2017. Every playlist has a title and includes at least three unique artists and two unique albums, has a minimum number of followers and listeners and has at least 5 tracks and no more than 250 tracks. Each playlist is caracterized by the playlist identifier, the playlist title, the total number of tracks in the playlist and the number of tracks included in the playlist. For each track in the playlist, the following information is included: the position in the playlist, the title, the Spotify URI, the name and Spotify URI the of the primary artist, the title and the Spotify URI of the album, the duration. The main dataset statistics are summarized in Table 2.

Table 2: Dataset statistics

Playlists	1,000,000
Tracks	2,262,292
Albums	734,684
Artists	295,860
Titles	17,381
Playlists with descriptions	18,760

The challenge set contains 10,000 incomplete playlists and the task is to recommend tracks for each of these playlists. These 10,000 playlists are grouped in 10 clusters of 1,000 elements with similar characteristics. Specifically the clusters consist of: (1) empty playlists with titles; (2) playlists with titles and the first track; (3) playlists with titles and the first 5 tracks; (4) playlists without titles but with the first 5 tracks; (5) playlists with titles and the first 10 tracks; (6) playlists with first 10 tracks but without titles; (7) playlists with titles and the first 25 tracks; (8) playlist with title and 25 random tracks; (9) playlists with titles and the first 100 tracks; (10) playlists with title and 100 random tracks.

We created a development set for internal testing that resembles the challenge set. We reproduced the distribution of tracks in playlists with titles and playlists without titles within the challenge set. The goal was then to sample a development set that is representative of the challenge set.

We separated 10,000 random playlists for creating our development set. For 2,000 of them, we removed their title. We retained either five tracks or ten tracks at random (with a 0.5 probability each). For the remaining 8,000, we randomly selected the number of retained tracks according to the basic statistics of the test set: we retain either 100 or 25 tracks with a 0.2 probability each, while we retain either zero, one, five, or ten tracks, with probability 0.1 each. Due to our random sampling approach, the resulting distribution of tracks slightly differs from the challenge set.

We did not distinguish between selecting the first tracks or random tracks. We decided to always select the tracks at random. We also employed a naive approach for dealing with playlists with few tracks. In these cases, we cannot remove more tracks than available. However, we consider these effects to be negligible.

4.2 Procedure

Preliminary experiments. The challenge task is similar to the task addressed in an our previous study [6]. In that case, the goal was recommending citations or subject labels starting from a partial list of references or labels. In this case, the system is given a playlist with some tracks as input. In our previous work, we split the data on the time axis of the citing documents to resembles the natural constraint that publications cannot cite other publications that do not exist yet. Similarly, no subject labels are available for papers still not published. All documents that were published before a certain year were used as training, and the remaining documents as test data. For the challenge, time information is available in the from of the last modification of a playlist, but it is not so meaningful as the publication year. Thus, we did not split based on time. However, these settings are more demanding than the challenge ones as they correspond to a new user scenario: the system has to recommend tracks for a playlist he has never seen before, which correspond to the situation in which no ratings are given for some users.

As preliminary experiments, we compared AAE on the Million Playlist Dataset in these more challenging settings against the baselines used in the previous experiments, i. e. item co-occurrence (IC) and singular value decomposition (SVD), as well as the decoder (MLP) and the classical undercomplete autoencoder. This enabled us to verify that the approach is also effective on this dataset and to check whether using additional metadata together with the list of tracks is beneficial. Specifically, we run all the methods once using the title of the playlist and once without additional information to compare their performance. The results showed that using playlist title is more effective, as summarized in Section 4.4.

We randomly split the data in order to obtain a 90:10 ratio between training and test items guaranteeing that all playlists used as training, do not appear in the test data. In other words, instead

[3]http://recsys-challenge.spotify.com/dataset

Table 3: Values tested for each parameter

Hyperparameter	Values
n_{tracks}	25 k, 50 k, 75 k, 100 k
Hidden units	50, 100, 200
Epochs	10, 20
Code size	50, 100

of cutting the $\boldsymbol{P} \times \boldsymbol{T}$ matrix along the time axis, as we did for documents [6], we randomly select some of its rows (playlists) for the test set. The hyperparameters are initially selected based on our previous experiments on other datasets [6]. While we do not exclude that a certain set of hyperparameters may perform better in a specific scenario, we select the following, most robust, hyperparameters: hidden layer sizes of 100 with ReLU [14] nonlinearities and drop probabilities of 0.2 after each hidden layer. The optimization is carried out by Adam [10] with initial learning rate 0.001. The two autoencoder variants use a code size of 50. We further select a Gaussian prior distribution for the adversarial autoencoder. For SVD, we consecutively increased the number of singular values up to 1,000. Higher amounts of singular values decreased the performance.

As a preprocessing step, we extracted the 50,000 most frequent tracks in the training set. We filtered both the training and test set to retain only those items. Furthermore, we remove playlists that have less than one track left, so that there is at least one track to drop out for testing. In the extreme case, this results in no track as input for a given playlists.

Optimization on the development set. We selected the most promising approach from the preliminary experiments, namely AAE, and optimized it on the challenge task exploiting our development set. The training set includes all the playlists that do not belong to the development set. For preprocessing the dataset, we first built a vocabulary on the training set including only the 50,000 most frequent distinct words from the metadata considered (playlist title, track title, artist name, and album title). Subsequently, we restrict our model to the n_{tracks} most frequent distinct tracks. The rationale is that for the less frequent items, there is too few training data available such that taking these into account would harm the overall performance. We tested different configurations of considered tracks (n_{tracks}), hidden units, number of training epochs, and code sizes. The values tested for each parameter are listed in Table 3. We tried the different values of hidden units and epochs as well as the code size on a predefined vocabulary based on Google news and with n_{tracks} equal to 50,000. Then, we choose the best-performing values (200, 20 and 100, respectively) while varying n_{tracks}.

As using playlist titles is effective, we expected that exploiting even more metadata further improve the results. For this reason, we run every configuration of the AAE model both relying on the playlist title only and on playlist title together with track title, artist, and album. When considering more metadata, we aggregated this information generating a string with all these metadata (playlist title as well as title, artist, and album for each of its track), which is concatenated to the code. Overall, we run 20 different configurations.

Final experiments. Finally, we conducted some experiments on the challenge set. We provided playlist titles together with tracks metadata to the model as this proved to be effective during the optimization on the development set. The training now takes all the playlists of the challenge set into account. We apply the same preprocessing steps carried out for the optimization on the development set and we tested several configurations varying the n_{tracks} as before, with hidden units, epochs and code size set to 200, 20 and 100, respectively.

4.3 Evaluation metrics

The metrics considered in the challenge were R-precision [12], normalized discounted cumulative gain (NDCG) [9], and song clicks. All these metrics were evaluated at both the track level (exact track must match) and the artist level (any track by the artist is a match). Additionally, we relied on the mean reciprocal rank (MRR) [5] for the preliminary experiments to reuse a module implementing the evaluation of the models through this measure. Thus, we could focus on adapting the models tested on the new scenario. As MRR considers the single highest-ranked relevant item, it is an appropriate choice because in the preliminary experiments we predict only one track. In the following, we recall all the measures exploited denoting the set of tracks considered as ground truth by G, the ordered list of recommended tracks by R, and the size of a set or list by $|\cdot|$.

R-precision. R-precision (Equation 1) is the number of retrieved relevant tracks divided by the number of known relevant tracks, i. e., the number of withheld tracks. The metric is averaged across all playlists in the challenge set and rewards the total number of retrieved relevant tracks regardless of their order.

$$R - precision = \frac{\left|G \bigcap R_{1:|G|}\right|}{|G|} \tag{1}$$

NDCG. The discounted cumulative gain (DCG) measures the ranking quality of the recommended tracks. It increases when relevant tracks are placed higher in the list. The normalized DCG (NDCG) is determined dividing the DCG by the ideal DCG (IDCG). In the latter, the recommended tracks are perfectly ranked. Equation 2 shows the DCG, while Equation 3 shows the IDCG. When a recommended track is relevant, r_i is 1, while it is 0 otherwise. If the intersection between G and R is empty, then the DCG is equal to 0. The NDCG metric is given in Equation 4.

$$\text{DCG} = r_1 + \sum_{i=2}^{|R|} \frac{r_i}{\log_2(i+1)} \tag{2}$$

$$\text{IDCG} = 1 + \sum_{i=2}^{|G|} \frac{1}{\log_2(i+1)} \tag{3}$$

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}} \tag{4}$$

Recommended Songs clicks. Recommended Songs is a Spotify feature that, given a set of tracks in a playlist, recommends 10 tracks to add to the playlist. The list can be refreshed to produce 10 more tracks. Recommended Songs clicks is the number of refreshes needed before a relevant track is encountered (Equation 5). If the metric does not exist, i. e., there is no relevant track in R, a value of 51 is picked, which is 1 plus the maximum number of clicks possible.

$$\text{clicks} = \left\lfloor \frac{\arg\min_i \{R_i : R_i \in G\} - 1}{10} \right\rfloor \tag{5}$$

Mean reciprocal rank. The reciprocal rank (RR) assesses the reciprocal of the position at which the first relevant track occurs in *R*. It is 1 if a relevant track is in the first position, 0.5 if a relevant track occurs in the second position, and so on [5]. The mean reciprocal rank (MRR), shown in Equation 6, is the RR averaged across the playlists. The position of the first relevant track in the list of recommendations for the i-th playlist is denoted by r_i.

$$\text{MRR} = \frac{1}{|G|} \sum_{i=1}^{|G|} \frac{1}{r_i} \tag{6}$$

4.4 Results

Table 4 shows the results for the preliminary experiments. We run each model once using the title of playlists and once without the titles. Item co-occurrence (IC) cannot exploit the titles, MLP can only rely on titles. In all the cases tested, using titles increased the performance. AAE obtained the highest MRR both with and without titles. Thus, we decided to further optimize this model on the development set.

Table 4: Results of the preliminary experiments both with playlist titles and without playlist titles.

Method	**MRR**	
	No titles	Titles
IC	0.0515 (0.1700)	-
SVD	0.0658 (0.1946)	0.0662 (0.1953)
AE	0.0645 (0.1855)	0.0679 (0.1913)
AAE	**0.0682** (0.1937)	**0.0700** (0.1958)
MLP	-	0.0300 (0.1310)

Table 5: Results of the best configuration AAE on our development set (dev) and on the challenge set (final) with playlist titles only (Titles) and with aggregated metadata (Aggr.).

Set	**R-Prec**		**NDCG**		**Clicks**	
	Titles	Aggr.	Titles	Aggr.	Titles	Aggr.
dev	0.1063	0.1205	0.2092	0.2319	9.9477	7.9350
final	-	**0.1787**	-	**0.3201**	-	**5.3510**

Table 5 lists the best-performing AAE configurations on the development set and on the challenge set. On the development set, we run each model once using playlist titles and once using aggregated metadata (playlist titles as well as title, artist, and album of each track in the considered playlist). As on the development set using aggregated metadata was more effective, we run only experiments with aggregated metadata on the challenge set. In both configurations, the vocabulary contained the 50,000 most frequent distinct words from the metadata considered, and we restricted our model to the 75,000 most frequent distinct tracks (n_{tracks}). The code size, hidden units, and training epochs were 100, 200, 20, respectively.

5 CONCLUSION

We extended adversarial autoencoders to the problem of automatic playlist continuation and we showed how multiple input modalities can be incorporated. We executed various experiments for the challenge task to study how adversarial autoencoders perform while exploiting metadata along with the partial list of tracks. Our approach outperformed other models and considering additional metadata such as the playlist titles as well as track titles, artists, and albums increased the performance of the models.

Reproducibility. The source code for reproducing our experiments is openly available on GitHub[4].

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation under project number 311018540 (Linked Open Citation Database) as well as by the EU H2020 project MOVING (contract no 693092).

REFERENCES

[1] Julio Barbieri, Leandro G. M. Alvim, Filipe Braida, and Geraldo Zimbrão. 2017. Autoencoders and recommender systems: COFILS approach. *Expert Syst. Appl.* 89 (2017), 81–90.

[2] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2012. Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives. *CoRR* abs/1206.5538 (2012).

[3] Cornelia Caragea, Adrian Silvescu, Prasenjit Mitra, and C. Lee Giles. 2013. Can't see the forest for the trees?: a citation recommendation system. In *JCDL*. ACM, 111–114.

[4] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. RecSys Challenge 2018: Automatic Music Playlist Continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA.

[5] Nick Craswell. 2009. *Mean Reciprocal Rank.* Springer US, Boston, MA, 1703–1703.

[6] Lukas Galke, Florian Mai, Iacopo Vagliano, and Ansgar Scherp. 2018. Multi-Modal Adversarial Autoencoders for Recommendations of Citations and Subject Labels. In *Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization (UMAP '18)*. ACM, New York, NY, USA, 197–205.

[7] Lukas Galke, Ahmed Saleh, and Ansgar Scherp. 2017. Word Embeddings for Practical Information Retrieval. In *GI-Jahrestagung (LNI)*, Vol. P-275. GI, 2155–2167.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS*. 2672–2680.

[9] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446.

[10] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[11] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. 2015. Adversarial Autoencoders. *CoRR* abs/1511.05644 (2015).

[12] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Evaluation in information retrieval.* Cambridge University Press, 139–161.

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.

[14] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*. Omnipress, 807–814.

[15] Markus Schedl, Peter Knees, and Fabien Gouyon. 2017. New Paths in Music Recommender Systems Research. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. ACM, New York, NY, USA, 392–393.

[16] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval* 7, 2 (01 Jun 2018), 95–116.

[17] Henry Small. 1973. Co-citation in the scientific literature: A new measure of the relationship between two documents. *JASIS* 24, 4 (1973), 265–269.

[4]https://github.com/lgalke/mpd-aae-recommender