User Modeling and User-Adapted Interaction
https://doi.org/10.1007/s11257-020-09269-1

Using autoencoders for session-based job recommendations

Emanuel Lacic[1] · Markus Reiter-Haas[2] · Dominik Kowald[1] · Manoj Reddy Dareddy[3] · Junghoo Cho[3] · Elisabeth Lex[4]

Received: 15 April 2019 / Accepted in revised form: 1 June 2020

Abstract
In this work, we address the problem of providing job recommendations in an online session setting, in which we do not have full user histories. We propose a recommendation approach, which uses different autoencoder architectures to encode sessions from the job domain. The inferred latent session representations are then used in a k-nearest neighbor manner to recommend jobs within a session. We evaluate our approach on three datasets, (1) a proprietary dataset we gathered from the Austrian student job portal Studo Jobs, (2) a dataset released by XING after the RecSys 2017 Challenge and (3) anonymized job applications released by CareerBuilder in 2012. Our results show that autoencoders provide relevant job recommendations as well as maintain a high coverage and, at the same time, can outperform state-of-the-art session-based recommendation techniques in terms of system-based and session-based novelty.

Keywords Job recommendations · Session-based recommendation · Autoencoders · Session embeddings · Accuracy · Novelty

1 Introduction

People increasingly use business-oriented social networks such as LinkedIn[1] or XING[2] to attract recruiters and to look for jobs (Kenthapadi et al. 2017). Users of such networks make an effort to create personal profiles that best describe their skills, interests, and previous work experience. Even with such carefully structured content, it remains a non-trivial task to find relevant jobs (Abel 2015). As a consequence, the field of job recommender systems has gained much traction in academia

[1] http://linkedin.com.
[2] http://xing.com.

✉ Elisabeth Lex
elisabeth.lex@tugraz.at

Extended author information available on the last page of the article

Published online: 01 July 2020

and the industry (Lacic et al. 2019; Siting et al. 2012). The main challenge that job recommender systems tackle is to retrieve a list of jobs for a user based on her preferences or to generate a list of potential candidates for recruiters based on the job's requirements (Hong et al. 2013).

Besides, most online job portals offer the option to browse the available jobs anonymously in order to attract users to the portal. As a consequence, the only data a recommender system can exploit are anonymous user interactions with job postings during a session. In other words, the problem of recommending jobs is a session-based recommendation problem (Jannach and Ludewig 2017). That is, the aim is to recommend the next relevant job in an anonymous session.

In our ongoing work with the Austrian start-up Studo,[3] we have started to address the problem of recommending jobs in a session-based environment. In their student job portal Studo Jobs,[4] we have observed an increasing volume of anonymous user sessions that look for new jobs.[5] For example, over the past six months, anonymous job-related browsing has doubled from approximately 30,000 to 60,000 job interactions. Therefore, in this paper, we address the problem of recommending jobs in a session-based environment.

Recently, neural networks have gained attention in the context of session-based recommender systems (e.g., Hidasi et al. 2015; Li et al. 2017; Lin et al. 2018; Wu et al. 2018, 2019; Yuan et al. 2019). The idea is to extract latent information about a user's preferences from anonymous, short-lived sessions. For example, autoencoders (Kramer 1991) are neural networks designed to learn meaningful representations, i.e., embeddings, and to reduce the dimensionality of input data. Example applications are data compression (Theis et al. 2017), clustering and dimensionality reduction (Makhzani et al. 2015) as well as recommender systems, where they have been used to find latent similarities between users and items and to predict user preferences (Sedhain et al. 2015; Strub et al. 2016).

Their ability to preserve the most relevant features, while reducing dimensionality, inspired our idea to explore the use of autoencoders to infer latent session representations in the form of embeddings and to use these embeddings to generate recommendations in a k-nearest-neighbor manner. To that end, in this paper, we introduce a recommendation approach, which employs different autoencoder architectures, (1) a classic autoencoder (Kramer 1991), (2) a denoising autoencoder (Vincent et al. 2008) and (3) a variational autoencoder (Jordan et al. 1999), to learn embeddings of job browsing sessions. The inferred latent session representations are then used in a k-nearest neighbor manner to recommend jobs within a session. Besides, we use two types of input data to train and test our approach, i.e., interaction data from sessions and content features of job postings, for which interactions took place during a session. We assess the performance of our approach in the form

[3] https://studo.co.

[4] The jobs platform in Studo, which is the predecessor of the Talto career platform (https://talto.com)

[5] We observe this trend independent from the changes in authenticated sessions, which fluctuate heavily over the year. The cause of this trend is that both the total number of sessions and the average ratio of anonymous sessions to authenticated sessions are growing.

of offline evaluations on three datasets from the job domain: firstly, a dataset collected from the Austrian online student job portal Studo Jobs; secondly, the job dataset that was provided by XING after the RecSys Challenge 2017 (Abel et al. 2017); and finally, a dataset from a Kaggle competition on job recommendation sponsored by CareerBuilder. Our approach is compared to the state-of-the-art session-based recommender approaches (Hidasi and Karatzoglou 2018; Hidasi et al. 2015; Jannach and Ludewig 2017; Ludewig and Jannach 2018; Rendle et al. 2009) not only with respect to accuracy but also in terms of system-based and session-based novelty as well as coverage (Zhang et al. 2012). This is grounded in the growing awareness that factors other than accuracy contribute to the quality of recommendations (Herlocker et al. 2004; McNee et al. 2006). Moreover, novelty is especially an important metric for the job domain since applying to popular jobs may decrease a user's satisfaction due to high competition and less chance of getting hired (see e.g., Kenthapadi et al. 2017).

Contributions and findings The main contributions of this paper and the corresponding findings are as follows:

- We present a recommendation approach, which uses different autoencoder architectures to encode sessions from the job domain. We use the inferred latent session representations in a k-nearest neighbor manner to recommend jobs within a session.
- We compare our approach to methods from recent work (Hidasi and Karatzoglou 2018; Hidasi et al. 2015; Jannach and Ludewig 2017; Ludewig and Jannach 2018; Rendle et al. 2009) on the state-of-the-art session-based recommendation.
- We evaluate the efficacy of our approach on three datasets: firstly, a proprietary dataset collected from the online student job portal Studo Jobs; secondly, a publicly available job dataset that was provided by XING after the RecSys Challenge 2017; and thirdly, a publicly available job dataset from the job platform CareerBuilder.
- We train and test the autoencoders on two sources of job-related data: (1) interaction data from sessions and (2) content features of job postings, for which interactions took place during a session. Our results show that variational autoencoders provide competitive job recommendations in terms of accuracy compared to the state-of-the-art session-based recommendation algorithms.
- We additionally evaluate all session-based job recommender approaches in terms of the beyond-accuracy metrics with system-based and session-based novelty as well as coverage. We find that autoencoders can produce more novel and surprising recommendations compared to the baselines and, at the same time, provide relevant jobs for the user while maintaining a high coverage.
- We provide the implementation of our approach as well as a more detailed hyperparameter description in a public GitHub repository[6] in order to foster reproducible research.

[6] https://github.com/lacic/session-knn-ae.

Organization of the paper The remainder of the paper is structured as follows: In Sect. 2, we discuss related work. Section 3 outlines our approach to employ autoencoders for session-based job recommendation. Section 4 describes the baseline approaches, datasets, evaluation protocol and performance metrics. Section 5 elaborates on the results of our experiments. Finally, in Sect. 6, we conclude the paper and provide an outlook on our plans for future work.

2 Related work

At present, we identify two lines of research that are related to our work: (1) job recommender systems and (2) session-based recommender systems.

Job recommender systems Job recommender systems address a particular recommendation problem, in that a company might want to hire only a few candidates, while classic recommender systems typically recommend items that are relevant for a large number of users (Kenthapadi et al. 2017). There are two directions of the recommendation problem: One is to recommend jobs to a user given her user profile, while the other is to recommend candidates for a job posting. The directions of both problems can even be combined using a reciprocal recommender (Mine et al. 2013).

Research on recommending jobs to users has mostly focused on improving accuracy with methods like collaborative- and content-based filtering or hybrid combinations of both (Al-Otaibi and Ykhlef 2012; Zhang and Cheng 2016). One example of a hybrid job recommendation system that uses interaction data as well as content data is the work of Liu et al. (2017). Here, the recommendation problem corresponds to first searching for matching candidates for a given job and then recommending this job to these candidates. In another job recommender system presented in Hong et al. (2013), the authors propose to first cluster user profiles based on their characteristics and then to design separate recommendation strategies for each cluster.

In 2016, XING (a career-oriented social networking site based in Europe) organized a challenge for the ACM RecSys conference to build a job recommendation system (Abel et al. 2016) that recommends a list of job posts with which a user might interact in the upcoming week. The winning approach (Xiao et al. 2016) used a hierarchical learning-to-rank model to generate the recommendations, which captures semantic relevance, temporal characteristics of a user's profile information, the content of job postings and the complete log of user activities. The anonymized challenge dataset has since been employed, for instance, by Mishra and Reddy (2016), who built a gradient boosting classifier to predict if a given user will like a particular job posting. In 2017, XING organized another recommender challenge for the ACM RecSys conference (Abel et al. 2017). Here, the recommendation problem was turned into a search for suitable candidates when a new job posting is added to the system (i.e., the task constitutes a cold-start problem (Lacic et al. 2015)). The winning approach (Volkovs et al. 2017) spent considerable effort on feature engineering to train a gradient boosting algorithm, which determines the probability of whether or not a given candidate user profile is suited for a target job posting.

In our work, we employ the most recent version of the dataset provided by XING after the RecSys challenge 2017 to evaluate a range of approaches to provide job

recommendations in anonymous sessions. Besides, in our experiments, we use a proprietary dataset gathered from Studo Jobs, an Austrian student job portal, as well as a publicly available dataset from the job portal CareerBuilder.

Since in our work, we focus on session-based job recommendations, in the next paragraph, we summarize related work on session-based recommender systems.

Session-based recommender systems Most recommender systems require a user preference history in the form of explicit or implicit user interactions. Based on the user preference history, a user profile is created, which is the basis for approaches such as matrix factorization (Koren et al. 2009). However, it is not always possible to create such user profiles, e.g., to protect the privacy of users or due to inadequate resources. As a remedy, session-based recommender systems (Hidasi et al. 2015) have been proposed, which model a user's actions within a session, i.e., a short period when the user is actively interacting with the system. A simple approach toward session-based recommendation is to recommend similar items using item–item similarity as proposed by Sarwar et al. (2001). Hidasi and Tikk (2016) propose a general factorization framework that models a session using the average of the component latent item representations. Shani et al. (2005) use Markov decision processes to compute recommendations that incorporate the transition probability between items. Jannach and Ludewig (2017) use co-occurrence patterns as a basis for session-based recommendations. They report comparable and often even a superior performance of a heuristics-based nearest neighbor method (KNN) to generate recommendations in a session-based setting in comparison with competitive, state-of-the-art methods based on neural architectures. Hence, in our work, we also use two KNN-based methods, i.e., sequential session-based KNN and vector multiplication session-based KNN (Ludewig and Jannach 2018) as baseline algorithms due to their good performance and scalability as reported in related works (Jannach and Ludewig 2017; Kamehkhosh et al. 2017; Ludewig and Jannach 2018).

In general, applying neural networks in session-based recommendation systems has gained much attention in recent years. For instance, recent work (Tuan and Phuong 2017; Yuan et al. 2019) uses convolutional networks to produce session-based item recommendations. Song et al. (2016) proposed a neural architecture that combines both long-term and short-term temporal user preferences. They model these preferences through different long short-term memory (LSTM) networks in a stepwise manner. In this vein, Lin et al. (2018) introduce STAMP (short-term attention/memory priority) that simultaneously incorporates a user's general interest (i.e., long-term memory) and current interest (i.e., short-term memory). Wu et al. (2018) present an architecture for session-based recommendations that is based on graph neural networks. Here, using an attention network, each session is also represented by a session user's global preference and their current interest. The authors of Li et al. (2017) propose NARM (neural attentive recommendation machine), which uses an attention mechanism in a hybrid encoder to model the sequential behavior of a user and to extract the user's main purpose from the current session. As the authors show, this approach is specifically well suited to model long sessions.

Out of the different neural architectures, recurrent neural networks have become particularly popular for the task at hand (Chatzis et al. 2017; Hidasi et al. 2015; Smirnova and Vasile 2017). In the earlier mentioned work of Hidasi

et al. (2015), the authors showed that a recurrent neural network (RNN)-based approach can model variable-length session data. Other related papers on sequential data either improve the original algorithm (Hidasi and Karatzoglou 2018; Tan et al. 2016) or extend it by capturing additional information such as context (Twardowski 2016) or attention (Li et al. 2017). In later work, Hidasi et al. (2016) introduce an architecture (i.e., pRNN) that combines multiple RNNs to model sessions via clicks as well as via features of the clicked items such as content information. Here, each RNN handles a particular feature, such as the clicked item's textual representation. The authors show that, given the optimal training strategy, pRNN architectures can result in higher performance compared to feature-less session models. Due to its ability to incorporate content features of job postings in its model in addition to interactions within sessions, in our work, we use pRNN as a baseline approach as we also take into account content features of job postings as well as interactions.

In our work, we employ autoencoders, a type of neural network that can reduce the dimensionality of data (Kramer 1991), to infer latent session representations and to generate recommendations. Specifically, we propose to employ a classic autoencoder (Kramer 1991), a denoising autoencoder (Vincent et al. 2008) and a variational autoencoder (Jordan et al. 1999) to model and encode sessions. In this vein, we find that collaborative denoising autoencoders (CDAE) (Wu et al. 2016) are related to our work. CDAE utilize a denoising autoencoder (Vincent et al. 2008) by adding a latent factor for each user to the input. A denoising autoencoder can learn representations that are robust to small, irrelevant changes in the input. In CDAE, the number of parameters grows linearly with the number of users and items, which makes it prone to overfitting (Liang et al. 2018). Also related to our work is neural collaborative filtering (He et al. 2017), where a neural architecture, which can learn any function from data, replaces the dot product between the latent user and item features. However, this model has a similar issue as CDAE and, thus, grows linearly with the number of sessions and jobs as the authors of Liang et al. (2018) describe.

Finally, with respect to evaluation, to the best of our knowledge, related work on evaluating session-based recommender systems with beyond-accuracy metrics such as system-based and session-based novelty, or coverage is scarce. Only in recent work, Ludewig and Jannach (2018) evaluate session-based recommender systems in light of coverage and popularity bias. With this work, we aim to contribute to this sparse line of research as we evaluate all approaches in this work with respect to system-based and session-based novelty as well as coverage, in addition to accuracy.

3 Approach

In this section, we describe our approach toward a session-based job recommender system using autoencoders. In Sect. 3.1, we first describe how we encode sessions with autoencoders. Then, in Sect. 3.2, we outline our method to model the input session vectors from interactions and content features. Finally, Sect. 3.3 details how we compute session-based job recommendations.

3.1 Encoding sessions using autoencoders

Autoencoders are a type of neural network, which were popularized by Kramer (1991) as a more effective method than principal component analysis (PCA) with respect to describing and reducing the dimensionality of data. Autoencoders are trained in an unsupervised manner where the network is trying to reconstruct the input by passing the information to the output layer through a bottleneck architecture. For our work, we employ three variants of autoencoder architectures to represent a session: (1) a classical autoencoder (AE), (2) a denoising autoencoder (DAE) and (3) a variational autoencoder (VAE).

Autoencoder (AE) The simplest form of an autoencoder has only one hidden layer (i.e., the latent layer) between the input and output (Bengio et al. 2007). The latent layer takes the vector $x_s \in \mathbb{R}^D$, which represents the session and maps it to a latent representation $z_s \in \mathbb{R}^K$ using a mapping function:

$$z_s = h(x_s) = \sigma(W^T x_s + b)$$

where W is a $D \times K$ weight matrix, $b \in \mathbb{K}$ is an offset vector and σ is usually a non-linear activation function. Using z_s, the network provides a reconstructed vector $\hat{x_s} \in \mathbb{R}^D$, which is calculated as:

$$\hat{x_s} = \sigma(W' z_s + b')$$

By adding one or more layers between the input and latent layer, we create an *encoder* and, correspondingly, a *decoder* by doing the same between the latent and output layer, hence the name autoencoder. During inference, we use the output of the latent layer (i.e., the information bottleneck) to represent the latent session vector z_s.

In our experiments, for σ, we use rectified linear units (ReLU[7]) (Nair and Hinton 2010) activation function for all layers except the final output layer, where a sigmoid activation function is used. Furthermore, we use a $D_s - 256 - 100 - 256 - D_s$ network architecture,[8] where D_s is the dimension of the original vector representation of the session that is encoded using job interactions with or without the corresponding job content data. To train the network, we use RMSprop (Tieleman and Hinton 2012) and minimize the Kullback–Leibler divergence (Fischer and Igel 2012).

We also experimented with adding additional encoder/decoder layers as well as increasing the layer size (e.g., layers with a size of 1000) but did not see any major performance differences besides an increased training complexity. Both Adam and RMSProp are two of the most popular adaptive stochastic algorithms for training deep neural networks. In our work, we focused on RMSProp.

Denoising Autoencoder (DAE) As shown by Vincent et al. (2008), extending autoencoders by corrupting the input can show surprising advantages. The idea of a

[7] For an input x, $relu(x) = max(0, x)$.

[8] We also tested higher values for the dimension of the latent layer (e.g., layers with a size of 1000) as well as adding additional encoder/decoder layers, but did not find enough accuracy improvement that would justify the additional computation burden when calculating session similarities in real time.

denoising autoencoder is to learn representations that are robust to small, irrelevant changes in the input. Corrupting the input can be done on either one or multiple layers before we calculate the final output.

In our DAE model, we get a corrupted input $\hat{x}$ using the commonly employed additive Gaussian noise on the input layer with a probability of 0.5. Like earlier, we use the same $D_s - 256 - 100 - 256 - D_s$ architecture, ReLU and sigmoid activation functions, the RMSprop optimization algorithm and the Kullback-Leibler divergence as loss function.

Variational Autoencoder (VAE) Another approach to extract the latent representation z_s is to use variational inference (Jordan et al. 1999). For that, we approximate the intractable posterior distribution $p(z_s|x_s)$ with a simpler variational distribution $q_\phi(z_s|x_s)$, for which we assume an approximate Gaussian form with an approximately diagonal covariance:

$$\log q_\phi(z_s|x_s) = \log \mathcal{N}(z_s; \mu, \sigma^2 I)$$

where μ and σ^2 is the encoded output given the input vector representation x_s of a session. To be more precise, we use additional neural networks as probabilistic encoders and decoders. Most commonly, this is done using a multilayered perceptron (MLP). For the above-mentioned $q_\phi(z_s|x_s)$, we calculate:

$$\begin{aligned} \mu &= W_2\, h + b_2 \\ \log \sigma^2 &= W_3\, h + b_3 \\ h &= \text{ relu } (W_1\, x_s + b_1) \end{aligned}$$

where $\{W_1, W_2, W_3, b_1, b_2, b_3\}$ are weights and biases of the MLP and are part of variational parameters ϕ. While decoding, we sample the latent representation and produce a probability distribution $\pi(z_s)$ over all features from the input session vector x_s. As we deal with implicit data, to calculate the probabilities, we let $p_\theta(x_s|z_s)$ be a multivariate Bernoulli (Kingma and Welling 2013), whose probabilities in the MLP we calculate as:

$$\log p(x_s|z_s) = \sum_{i=1}^{D} x_{si} \log y_i + (1 - x_{si}) \cdot \log(1 - y_{si})$$

$$y_s = f_\theta(\, W_5 \text{ relu } (W_4\, z_s + b_4) + b_5\,)$$

where f_θ is an element-wise nonlinear activation function (i.e., in our case a sigmoid) and $\theta = \{W_4, W_5, b_4, b_5\}$ are weights and biases of the MLP.

The generative model parameters θ are learned jointly with variational parameters ϕ by optimizing the marginal likelihood of the data. The objective is thus to minimize the distance between the variational lower bound $\mathcal{L}(\theta, \phi, x)$ and a certain prior (Kingma and Welling 2013; Liang et al. 2018), which in case of VAEs is the Kullback–Leibler divergence (Fischer and Igel 2012) of $q_\phi(z_s|x_s)$ and $p(z_s|x_s)$. As we are sampling z_s from q_ϕ in the variational lower bound, in order to learn the model, we need to apply the reparametrization trick (Kingma and Welling 2013; Rezende et al. 2014) by sampling $\epsilon \sim \mathcal{N}(0, I_K)$ (also seen later in Fig. 2) and reparametrize

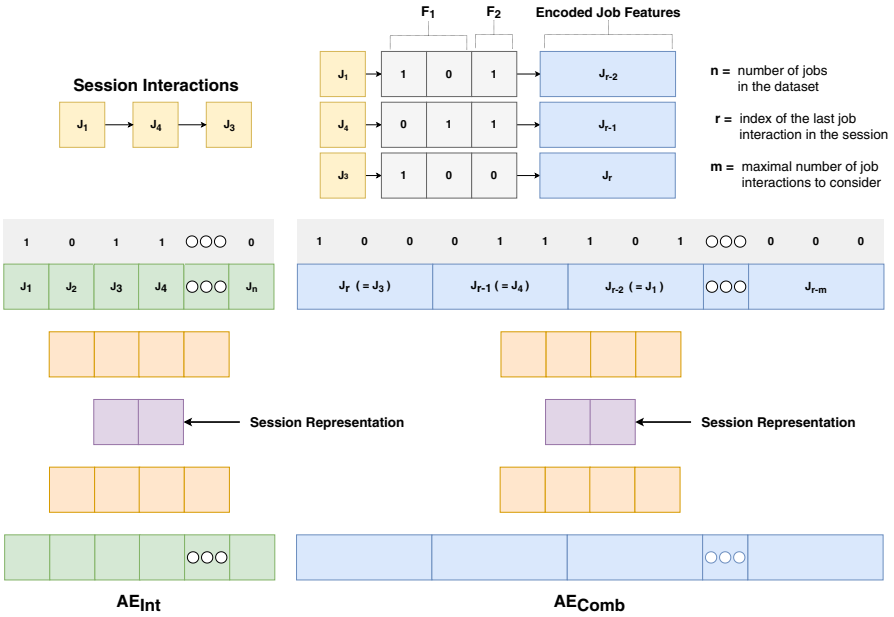


Fig. 1 Modeling session vectors. The input of the utilized autoencoder is a session representation, which can be binary-encoded using job interactions with or without the corresponding job content data. For example, a standard autoencoder that only considers interaction data (denoted as AE_{Int}) will expect a binary encoded vector with a dimension that equals the number of jobs in the underlying dataset. To combine this with job content data (denoted as AE_{Comb}), we use the most recent m job interactions within the session and generate a binary encoding of the job content features in descending order

$z_s = \mu_\phi(x_s) + \epsilon \odot \sigma_\phi(x_s)$. Hence, the gradient with respect to ϕ can be back-propagated through the sampled z_s.

In our experiments, we utilize the described VAE model with a similar architecture as previously mentioned: $D_s - 256 - 100 - 256 - D_s$ (i.e., the encoder and decoder MLPs are symmetrical). Furthermore, for all three autoencoder architectures, we experiment on additionally incorporating the self-attention mechanism (e.g., as Lin et al. 2017; Parikh et al. 2016; Vaswani et al. 2017 do in their work) on the encoder layer.

3.2 Modeling session vectors

The input for any of the three autoencoder variants is a binary-encoded representation of the session x_s. As shown in Fig. 1, we propose the following two variants of how to train the autoencoder models that will be used to infer the latent representation z_s.

Variant 1: Modeling from interactions We construct x_s by only using the job interaction data of a given session. In the remaining paper, we denote the three autoencoder models, which only use job interaction data as AE_{Int}, DAE_{Int} and VAE_{Int}. We create session vectors of size n_s, where n_s is the number of jobs in the

underlying dataset. Each job is then assigned an index in this vector. The interactions on the corresponding job indices are set to 1, while we set the rest to 0. One possible drawback of this approach is that due to the ephemeral nature of job postings, we would need to frequently retrain the utilized model in order to consider new jobs coming to the system (Matuszyk et al. 2015). Moreover, this will also impact the size of the input vector x_s, which will constantly be increasing with every new job.[9]

Variant 2: Modeling from interactions and content In order to mitigate the need to retrain the autoencoder models frequently, we also propose to leverage the content of job postings, with which anonymous users have interacted during a session (i.e., combine interaction data with content data). Given a set of content features $F = \{f_1, \ldots, f_l\}$, we first convert each job interaction in a session to a binary vector of size $n_j = \sum_{i=1}^{l} dist(f_i)$, where $dist(f_i)$ gives the number of distinct values of a job feature f_i. Each feature value is then assigned an index in this vector, and the existing feature values are set to 1, while the rest are 0. To create the session vector x_s, starting from the most recent job interaction, we concatenate the last m converted job interactions. In case the number of job interactions is less than m, x_s is right-padded with 0-filled job vectors, which results in x_s being of size $n_j \times m$. We denote the three autoencoder models that use the content features of job interactions as AE_{Comb}, DAE_{Comb} and VAE_{Comb}. Note also that we introduce the parameter m to end up with an input vector x_s that has a fixed length and a model that is less sensitive to new job postings that are added to the system.

3.3 Computing session-based job recommendations

We formulate the recommendation problem as follows: Given a target session s_t, in which there was an interaction with at least one job j_i from the set of available jobs $J = \{j_1, \ldots, j_n\}$, the task is to predict the next jobs this user will likely interact with. In order to compute recommendations, as shown in Fig. 2, we first extract the output z_s for the sessions that are available in the training set. During prediction time, for a given target session s_t, we proceed to infer its latent representation first to find the top-k similar past sessions. In order to reduce the computational burden and allow for efficient recommendation,[10] we extract a subset of all sessions, where the users have interacted with the last job in s_t. Using z_s, we compute the cosine similarity between the respective target and candidate session and use the top-k similar sessions to recommend jobs. Jobs are then ranked based on the following score:

$$sKNN(s_t, j_i) = \sum_{i=1}^{n} sim(s_t, s_i) \times 1_{s_i}(j_i)$$

[9] This effect can, however, be damped by removing obsolete job postings, but would still result in a constantly changing input dimension.

[10] The number of stored sessions can easily pass the million mark and cause for unnecessary calculations once a recommender system is running for a longer period.

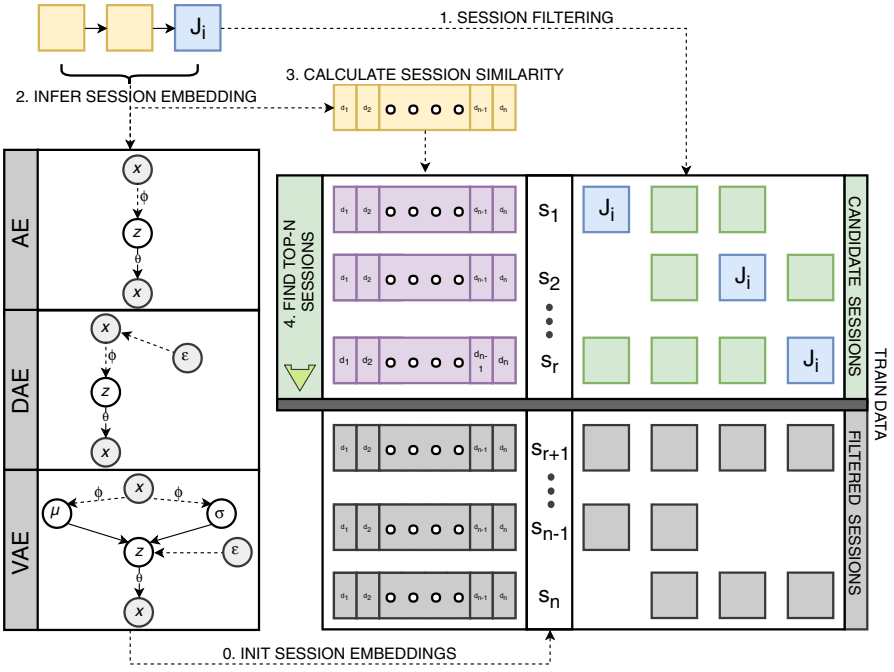


Fig. 2 Computing session-based job recommendations. Using the trained autoencoders, we infer latent representation for (1) sessions in the training data and (2) the current target session for which we recommend jobs. Jobs from the top-k similar candidate sessions (filtered by the currently interacted job posting) are recommended to the target session

where $1_{s_i}(j_i)$ is 1 if the candidate session s_i contains the job j_i and 0 otherwise (as in Bonnin and Jannach 2015; Jannach and Ludewig 2017).

4 Experimental setup

In this section, we present the baseline approaches and the datasets we used for this study. We outline the evaluation protocol and the performance measures, which we employed to compare all approaches. In our evaluation, we contribute to the limited amount of related work (e.g., like Ludewig and Jannach 2018) as we evaluate all approaches both concerning accuracy and beyond-accuracy measures (i.e., system-based and session-based novelty as well as coverage).

4.1 Baseline approaches

We utilize well-known baselines and compare our approach to the following state-of-the-art methods (Ludewig and Jannach 2018) for session-based recommendation:

POP A simple and yet often strong baseline for session-based recommendation is the popularity-based approach. As in Hidasi et al. (2015), the results are always the same top-k popular items from the training dataset.

iKNN The item-KNN approach recommends jobs that are similar to the actual job that is interacted with during the session. As in Hidasi et al. (2015), we use the cosine similarity and include regularization to avoid coincidental high similarities between rarely visited jobs.

BPR-MF One of the commonly used matrix factorization methods for implicit feedback is Bayesian personalized ranking (Rendle et al. 2009). As in Hidasi et al. (2015), we use the average of job feature vectors of the jobs that had occurred in the current session as the user feature vector to apply it directly to generate a session-based recommendation. That is, similarities of the feature vectors are averaged between a candidate job and the jobs of the current session.

Bayes Following the Bayesian rule, we calculate the conditional probability of a job x_i being clicked based on the previous r interactions of the current session s:

$$P(x_i|x_{s_1}, \ldots, x_{s_r}) = \frac{\prod_{j=1}^{r} P(x_{s_j}|x_i) \times P(x_i)}{\prod_{j=1}^{r} P(x_{s_j})}$$

This approach is, from a computational perspective, inexpensive to calculate and run in an online setting.

GRU4Rec Recently, Hidasi et al. (2015) showed that recurrent neural networks are excellent models for data generated in anonymous sessions. GRU4Rec combines gated recurrent units with a session-parallel mini-batch training process, and it incorporates a ranking-based loss function. For our study, we use the most recent improvement in GRU4Rec (Hidasi and Karatzoglou 2018). This GRU4Rec version employs a new class of loss functions tied together with an improved sampling strategy.

pRNN Another recent advancement of Hidasi et al. (2016) shows how to incorporate item features into the representation of neural networks. They propose several different architectures based on GRU units and ways to train them. We use a parallel architecture with simultaneous training for our experiments. This approach utilizes both a one-hot encoding of the current item interaction and an item representation as inputs for the subnets. The trained model uses the TOP1 loss function as defined in Hidasi et al. (2015).

sKNN Recent research has shown that computationally simple nearest-neighbor methods can be effective for session-based recommendation Jannach and Ludewig (2017). The session-based KNN approach first determines the k most similar past sessions in the training data. Sessions are encoded as binary vectors of the item space, and a set of k nearest sessions is retrieved for the current session using cosine similarity. The final job score is calculated by aggregating the session similarity over all the sessions that contain the candidate job.

V-sKNN Vector multiplication session-based KNN (V-sKNN) is a variant of sKNN that considers the order of the elements in a session. The idea here is to create a real-valued vector by putting more weight on recent interactions, where

Table 1 Statistics of the datasets Studo, RecSys Challenge 2017 (i.e., RecSys17) and CareerBuilder12

Dataset	# Interactions	# Sessions	# Jobs	Sparsity (%)
Studo	191,259	26,785	1111	99.36
RecSys17	55,380	16,322	15,686	99.98
CareerBuilder12	661,910	120,147	197,590	99.99

While Studo has more sessions and job interactions, the RecSys Challenge 2017 dataset has more job postings that can be recommended. CareerBuilder12 is the largest dataset, but also has the highest sparsity

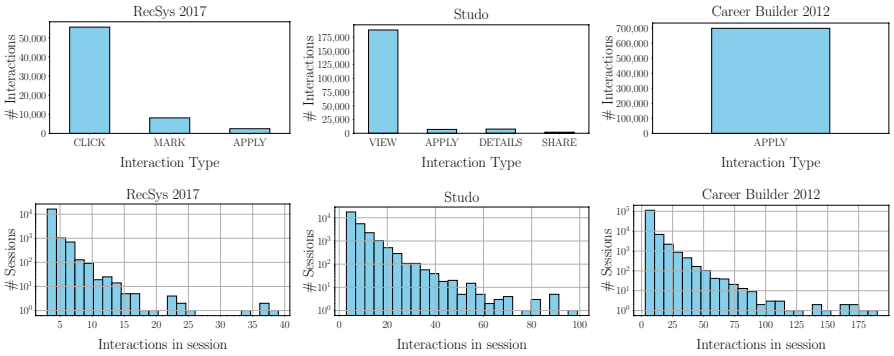


Fig. 3 Number of interactions based on the interaction type (top) and the distribution of session sizes (bottom) is shown for the RecSys Challenge 2017 (left) and Studo (middle) and CareerBuilder12 datasets. Overall, the distribution of interaction types is similar between the datasets where the *click*, *view* and *apply* interactions mostly dominate

only the very last element of the session obtains a value of "1" (Ludewig and Jannach 2018). For this, a linear decay function is used that depends on the position of an element within the session.

S-sKNN Sequential session-based KNN (S-sKNN) puts more weight on elements that appear later in the session in a similar way as V-sKNN (Ludewig and Jannach 2018). This effect is, however, achieved by giving more weight to neighboring sessions which contain recent items of the current session.

4.2 Datasets

For this study, we employ three different datasets from the job domain. The first dataset, *Studo*, is a proprietary dataset collected from the online platform Studo Jobs, a job-seeking service for university students. The second dataset *RecSys17* is the latest version of the data provided by XING after the RecSys Challenge

Table 2 Binary-encoded content features of our three datasets

Studo		RecSys17		CareerBuilder12	
Content feature	Encoding	Content feature	Encoding	Content feature	Encoding
Job state†	10	Region†	17	State†	55
Job country‡	1	Country‡	4	Requirement topic	20
Job begins now	1	Is payed	2	Title topic	20
Job effort	1	Career level	6	Description topic	20
Job language	1	Industry Id††	23		
Job discipline††	40	Discipline Id††	22		
Employment type‡‡	6	Employment‡‡	5		

For Studo, concatenating all job features results in a job vector with a dimensionality of 60. For the RecSys17 dataset, this results in a job vector with a dimension of 79. For the CareerBuilder12 dataset, this results in 115 dimensionality vectors. We also put the same annotation on content features, which have a similar meaning in both datasets. The *Job Discipline* feature is the only one in Studo, which represents a combination of the *Discipline Id* and *Industry Id* features from the RecSys17 dataset

2017 (Abel et al. 2017). The third dataset *CareerBuilder12* is from an open Kaggle competition, called Job Recommendation Challenge,[11] provided by the online employment Web site CareerBuilder. The statistics of all three datasets are given in Table 1. As seen, all datasets have a high sparsity: 99.36% for Studo, 99.98% for RecSys17 and 99.99% for CareerBuilder12. Studo contains a higher number of sessions when compared to RecSys17 but has a much smaller number of available jobs that can we can recommend. CareerBuilder12 is the largest dataset of the three, but only contains job applications as interactions. In the next paragraphs, we describe the three datasets in more detail.

Studo The dataset contains job interactions from anonymous user sessions from a period of three months between September 2018 and December 2018. All job interactions in this dataset have an anonymous session id assigned to them. As seen in the top row of Fig. 3, the Studo dataset contains four interaction types, i.e., job *view*, show company *details*, *apply* and *share* job. As shown at the bottom row of Fig. 3, the log histogram of session sizes follows a skewed pattern, which means that most sessions have a small number of interactions. In particular, every session has 6.98 interactions on average and a median of 5 interactions.

Concerning content features reported in Table 2, in the Studo dataset, we utilize seven content features of job postings. The *Job State* determines 1 out of 9 Austrian federal states. *Job Country* indicates whether the job is in Austria or some other country. The *Job Begins Now* feature specifies whether the job candidate can start immediately working on the advertised position. We relate this feature to the *Is Payed* feature from the RecSys17 dataset as companies typically pay for job postings to be shown if they urgently need candidates. Studo's *Job State* is similar to the *Region* feature from RecSys17, the same holds true for Studo's *Employment Type*,

[11] https://www.kaggle.com/c/job-recommendation.

which can be related to the *Employment* feature from RecSys17. *Job Effort* indicates whether the concrete working hours are specified; otherwise, the default working hours are assumed. The *Job Language* feature specifies whether the job requires the usage of either the German or English language. Furthermore, a job posting can also be described by a subset of 40 different *Job Discipline* labels and a subset of 6 different *Employment Type* labels. The *Job Discipline* feature can actually be regarded as a combination of the *Discipline Id* and *Industry Id* features from the RecSys17 dataset. As described in Sect. 3.2, we use all content features from the Studo dataset to create a binary-encoded job vector with a dimensionality of 60. Finally, we have 77.7% uniquely encoded job vectors, which consist, on average, of 11.8% assigned feature values.

RecSys17 The dataset contains six different interaction types that were performed on the job items. For this study, we only keep the *click*, *bookmark* and *apply* interactions (as seen on the top of Fig. 3). We remove the *delete recommendation* and *recruiter interest* interactions as these are irrelevant in our setting. Moreover, we discard *impression* interactions as they are created when XING shows a job to a user. As stated by Bianchi et al. (2017), an impression does not imply that the user has interacted with the job. The dataset consists of interactions from a period of three months (from November 6, 2016, until February 3, 2017). We manually partition the interaction data of the RecSys dataset into sessions using a 30-minute idle threshold (as in Quadrana et al. 2017). The resulting sessions have, on average, 3.62 interactions per session and a median of 3 interactions.

Also, the RecSys17 dataset contains content features about the job postings, such as career level or type of employment. From this set, we select seven features as content-based input for our approaches and discarded the numeric IDs of title and tags, since those would lead to very big encodings. The chosen features closely resemble the features that are present in the Studo dataset. More specifically, from RecSys17, we use the following features, as shown in Table 2: *Region*, *Employment*, *Is Payed*, *Discipline Id*, *Career Level*, *Industry Id* and *Country*. The *Region* content feature is a categorical feature with 17 possible value, like the *Employment* feature with 5 values. The *Is Payed* content feature indicates if the posting is a paid for by a company. The *Discipline Id* is a categorical feature with 22 different values that represent disciplines such as consulting or human resources. The categorical feature *Career Level* can take 7 values, *Industry Id* represents industries such as finance, and *Country* denotes the code of the country in which the job is offered.[12] Overall, we end up with a job vector that has dimensionality of 79. We find that only 33.58% of the job vectors are unique and have on average 8.86% assigned feature values.

CareerBuilder12 The dataset contains job applications from a period of almost three months. No other interaction types are present in this dataset. The sessions are created via a time-based split of 30 min. Due to the nature of job applications, most sessions contain very few interactions. The interactions in the dataset happen over 13 weeks. Thus, similar to the other two datasets, it happens over almost three

[12] https://www.recsyschallenge.com/2017.

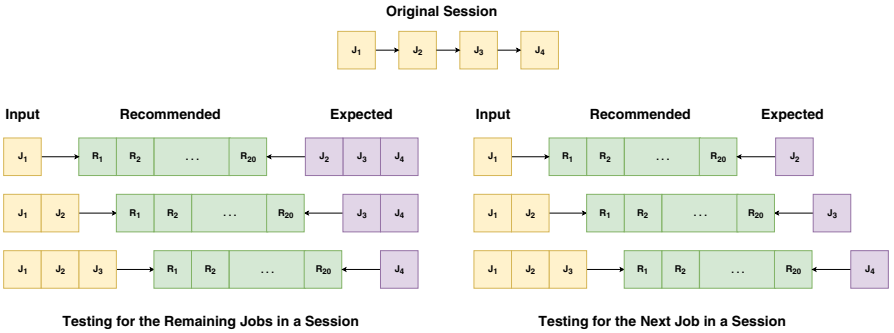


Fig. 4 Our evaluation protocol for one exemplary session consisting of four jobs. We distinguish between (1) comparing the recommended jobs with the remainder of the interactions in a session (left) and (2) comparing the recommended jobs with the next job interaction (right)

months, i.e., from April 2012 to June 2012. The sessions have, on average, 5.64 job applications per session, whereas the median is 4 applications per session.

Regarding content features, the CareerBuilder12 dataset contains textual descriptions of the jobs as well as categorical data for the location. From the content data, the 55 different states are used in the form of one-hot encodings. Since in our work, we utilize categorical job features as input to our models, we additionally inferred categorical topics for each of the 3 textual features (i.e., title, description and requirements). That is, for every textual feature, we trained a separate latent Dirichlet allocation (LDA) model from which we extracted 20 distinct topics. This procedure resulted in every job posting having a requirement, title and description topic assigned to them. Thus, the resulting feature vector of a job posting is of size 115. For this largest dataset, 13.46% of vectors are unique, and those vectors have only 2.58% assigned feature values.

4.3 Evaluation protocol

We employ a time-based split on all three datasets to create train and test sets. For this, we put the sessions from the last 14 days (i.e., 2 weeks) in the test set of the respective dataset and use the remaining sessions for training. For each set, we keep only sessions with a minimal number of 3 interactions.[13] Like (Quadrana et al. 2017), we filter items in the test set that do not belong to the train set as this enables a better comparison with model-based approaches (e.g., RNNs), which can only recommend items that have been used to train the model. In Studo, this procedure results in 23, 738 sessions to train and 3047 to test the approaches. For the RecSys17 dataset, this results in 12, 712 sessions for training and 3610 sessions for testing. In

[13] We chose 3 for the minimum amount of interaction as it is the lowest median of interactions in a session across all three datasets, as reported in Sect. 4.2.

the case of the much larger CarrerBuilder12 dataset, the train set contains 108, 783 sessions, whereas the test set has 11, 364 sessions.

Training and testing the algorithms We first train all approaches on the respective training data. In order to evaluate the performance of the utilized session-based recommendation algorithms, for each session in the test data, we iteratively subsample its interactions. That is, after each session interaction, we recommend 20 jobs for the current target session state and compare the predictions with the remaining interactions. We start this procedure for every session after the first interaction and end before the last one. In this setting, as shown in Fig. 4, we explore two evaluation cases: comparing the recommended jobs with (1) the remainder of the interactions in the session and (2) with the next job interaction (i.e., next item prediction; same as in Hidasi and Karatzoglou 2018; Hidasi et al. 2015).

For our proposed method, that uses content features in combination with user interactions to encode the input for the autoencoders (i.e., as described in Sect. 3.2), we use the top 25 recent job interactions to infer the session representation. That is, we set the parameter $m = 25$ as more than 98% of all sessions in Studo, and almost all sessions in the RecSys17 dataset do not have more than 25 job interactions (i.e., as shown in Fig. 3).

Hyperparameter optimization To optimize hyperparameters, we further split the train sets by the same time-based split to generate validation sets. Thus, we use the last 2 weeks of the train set as a separate validation set and the remaining sessions to train our models. The resulting split for the Studo dataset is 19, 245 sessions in the validation train set and 3273 sessions in the validation test set. For the RecSys17 dataset, we have 8001 sessions in the validation train set and 2046 sessions in the validation test set. In case of CareerBuilder12, the validation train set contains 51, 717 sessions and the validation test set 10, 574 sessions. Note that some sessions did no longer have the minimal number of 3 interactions and were filtered out. As a consequence, the combination of the validation train and validation test set is smaller than the original train set. The results of the hyperparameter optimization step are described in Sect. 5.3.

4.4 Evaluation metrics

We quantify the recommendation performance of each approach concerning accuracy and beyond-accuracy metrics like system-based and session-based novelty. More specifically, in our study, we use the following performance measures:

Normalized Discounted Cumulative Gain (nDCG) nDCG is a ranking-dependent metric that measures how many jobs are predicted correctly. Also, it takes the position of the jobs in the recommended list into account (Parra and Sahebi 2013). It is calculated by dividing the DCG of the session's recommendations with the ideal DCG value, which is the highest possible DCG value that can be achieved if all the relevant jobs would be recommended in the correct order. The nDCG metric is based on the *Discounted Cumulative Gain (DCG@k)*, which is given by Parra and Sahebi (2013):

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel(i)} - 1}{log(1+i)}$$

where $rel(i)$ is a function that returns 1 if the recommended job at position i in the recommended list is relevant. nDCG@k is calculated as DCG@k divided by the ideal DCG value iDCG@k, which is the highest possible DCG value that can be achieved if all the relevant jobs would be recommended in the correct order. Over all the sessions, it is given by:

$$nDCG@k = \frac{1}{|S|} \sum_{s \in S} \left(\frac{DCG@k}{iDCG@k} \right)$$

Mean reciprocal rank (MRR) MRR is another metric for measuring the accuracy of recommendations and is given as the average of the reciprocal ranks of the first relevant job in the list of recommended jobs, i.e., 1 for the first position, $\frac{1}{2}$ for the second position, $\frac{1}{3}$ for the third position and so on. This means that a high MRR is achieved if relevant jobs occur at the beginning of the recommended jobs list (Voorhees 1999). Formally, it is given by Aggarwal (2016):

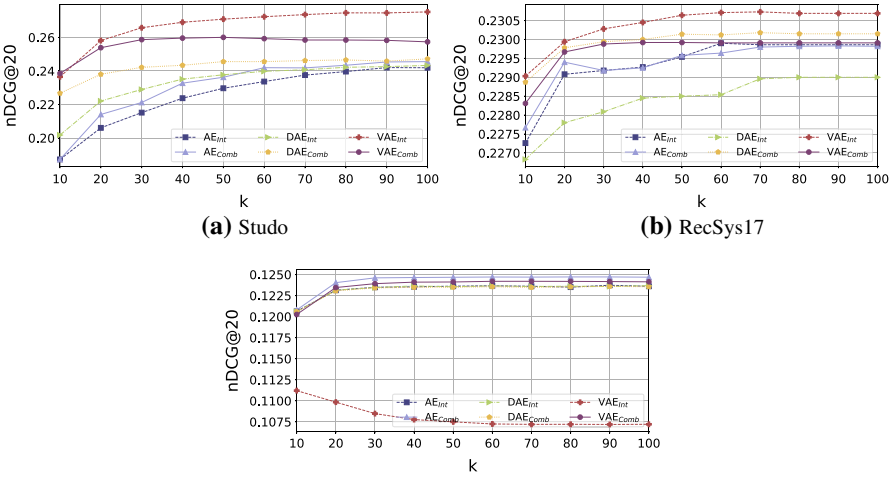
$$MRR@k = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|H_s|} \sum_{H_j \in H_s} \frac{1}{rank(H_j, R_k)} \tag{1}$$

Here, H_s is the history of the current session s and $rank(H_j, R_k)$ is the position of the first relevant job H_j in the recommended job list R_k.

System-based novelty (EPC) System-based novelty denotes the ability of a recommender to introduce sessions to job postings that have not been (frequently) experienced before in the system. A recommendation that is accurate but not novel will include items that the session user enjoys, but (probably) already knows. Optimizing system-based novelty has been shown to have a positive, trust-building impact on user satisfaction (Pu et al. 2011). Moreover, system-based novelty is also an important metric for the job domain since applying to popular jobs may decrease a user's satisfaction due to high competition and less chance of getting hired (see, e.g., Kenthapadi et al. 2017). In our experiments, we measure the system-based novelty using the expected popularity complement (EPC) metric introduced by Vargas and Castells (2011). In contrast to solely popularity-based metrics (e.g., Zhou et al. 2010), EPC also accounts for the recommendation rank and the relevance for the current session. Thus, the system-based novelty $nov_{system}(R_k|s)$ for the recommendation list R_k of length k for session s is given by:

$$EPC@k = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|R_k|} \sum_{R_i \in R_k} disc(i) p(rel|R_i, s)(1 - p(seen|R_i))$$

Here, $disc(i)$ is a discount factor to weight the recommendation rank i [i.e., $disc(i) = 1/log_2(i+1)$] and $p(rel|R_i, s)$ is 1 if the recommended job R_i is relevant for session s or 0 otherwise (i.e., only jobs that are in the current session history are taken



(a) Studo **(b)** RecSys17

(c) CareerBuilder12

Fig. 5 The figures show the influence of the neighborhood size k for picking top-k similar sessions when comparing the three autoencoder variations on both interaction data and combined data. We find that the recommendation accuracy converges when k is picked to be around 60 or more

into account). Finally, $p(seen|R_i)$ defines the probability that a recommended job R_i was already seen in the system, i.e., $p(seen|R_i) = log_2(pop_{R_i} + 1)/log_2(pop_{MAX} + 1)$.

Session-based novelty (EPD) In contrast to system-based novelty, session-based novelty incorporates the semantic content of jobs and represents how *surprising* or *unexpected* the recommendations are for a specific session history (Zhang et al. 2012). Given a distance function $d(H_i, H_j)$ that represents the dissimilarity between two jobs H_i and H_j, the session-based novelty is given as the average dissimilarity of all job pairs in the list of recommended jobs R_k and jobs in the current session history H_s (Zhou et al. 2010). In our experiments, we use the cosine similarity to measure the dissimilarity of two job postings using a raw job vector, which contains 1 if a session interacted with it and 0 otherwise. Again, we use the definition by Vargas and Castells (2011) that takes the recommendation rank as well as the relevance for the current session into account. Hence, we measure the session-based novelty $nov_{session}(R_k|s)$ for the recommendation list R_k of length k for session s by the expected profile distance (EPD) metric:

$$EPD@k = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|R_k||H_s|} \sum_{R_i \in R_k} \sum_{H_j \in H_s} disc(i) p(rel|R_i, s) d(R_i, H_j)$$

Here, H_s is the current history of a session s and $disc(i)$ as well as $p(rel|R_i, s)$ are defined as for the EPC metric for measuring the system-based novelty.

Coverage With coverage (Adomavicius and Kwon 2012; Ludewig and Jannach 2018), we assess how many jobs a recommender approach can cover with its predictions. As such, we additionally report the job coverage of each evaluated algorithm. We define the coverage as the ratio between the jobs that have been recommended

and jobs that would be available for recommendation. Here, we make a distinction between coverage types and report the job coverage (1) on the full dataset, i.e., how many of all available jobs can we recommend, and (2) on the test dataset, i.e., how many of the jobs can we recommend that we expect to be interacted with during a session.

5 Results

In this section, we present our experimental results. We first compare the performance of the respective models when used in a k-nearest neighbor manner and then analyze the embedding space of the best-performing autoencoder model. After that, we show the best hyperparameter configurations used for the baseline approaches and then discuss the performance of our approach compared to these baselines.

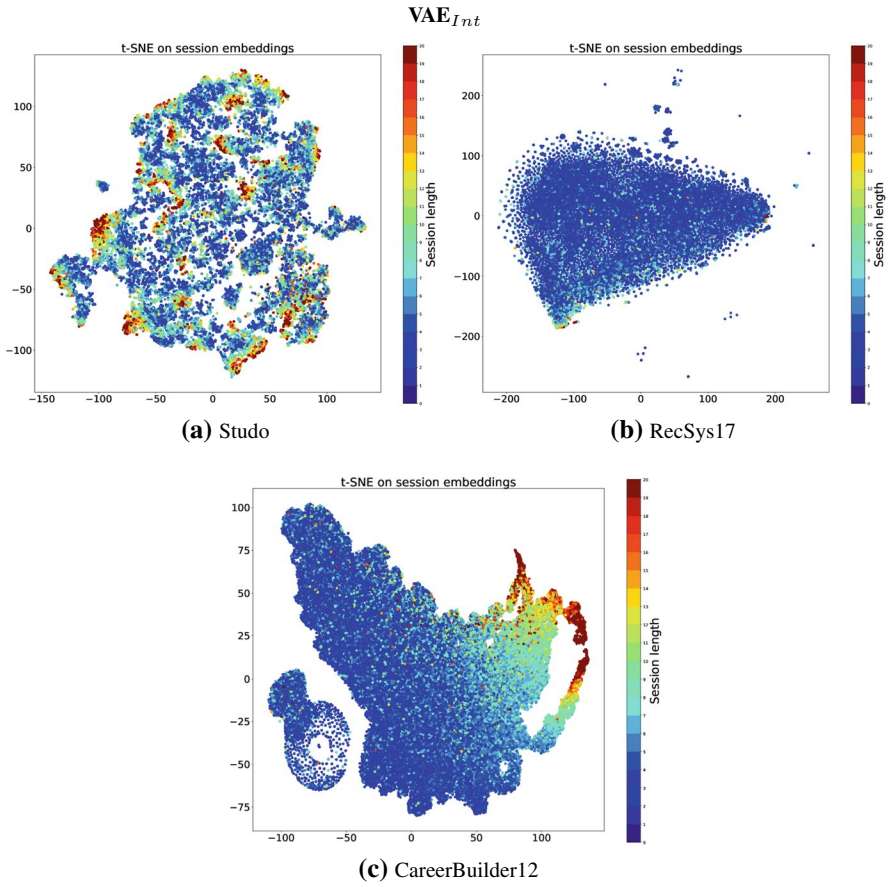
5.1 Comparing the recommendation performances of autoencoders

We compare the recommendation performance of all three variants of autoencoders, i.e., AE, DAE, VAE, trained on interactions as well as on content. This results in six autoencoder variants in total. We train all autoencoder models for a maximum of 50 epochs or until the error on the validation test set converges. We made additional experiments and incorporated the self-attention mechanism on the encoder layer (e.g., as in Lin et al. 2017; Parikh et al. 2016; Vaswani et al. 2017). We did not find any major improvements, so we do not report the results of these 6 additional autoencoder models.

Figure 5 shows the results of the autoencoder comparison in terms of *nDCG*@20. We compare the results across different values for the neighborhood size k, ranging from 10 to 100. We find that VAE_{Int}, which only uses interactions to encode the input vector, outperforms all other approaches on the Studo and RecSys17 datasets. When combining interaction data with content features (i.e., AE_{Comb}, DAE_{Comb} and VAE_{Comb}), VAE_{Comb} performs the best on the Studo dataset and slightly worse than DAE_{Comb} on the RecSys17 dataset. For the CareerBuilder12 dataset, all approaches except the VAE_{Int} approach have a similar performance. Such accuracy performance for VAE_{Int} suggests that having a much larger item space can be problematic for the generative autoencoder variant. As the variational autoencoder outperforms the other approaches in the majority of the configurations, in the next step, we compare it to the baseline methods. Furthermore, we find that for all autoencoders, accuracy converges after $k = 60$. Thus, in Sect. 5.4, we report the results of VAE_{Int} and VAE_{Comb} using top-60 similar sessions for recommendation.

5.2 Embedding analysis

To better understand the autoencoder models' actual effectiveness, we employ the t-SNE algorithm (Maaten and Hinton 2008) to visualize the embedding spaces. The t-SNE method enables us to visualize high-dimensional data. It reduces the



(a) Studo **(b)** RecSys17

(c) CareerBuilder12

Fig. 6 The plots show t-SNE embeddings for latent session representations produced with the VAE autoencoder models trained on all three datasets using only interaction data. The colors of the sessions reflect the session length, where the same red color is used for sessions with 20 or more interactions

dimensionality of the latent session representations and lets us explore embeddings in a 2*D* space. In t-SNE plots, similar items are modeled by neighboring points with high probability. In our case, we expect similar sessions to form clusters of neighboring points in the 2*D* space.

Figures 6 and 7 show the variational autoencoder models as t-SNE plots for all three datasets, i.e., VAE$_{Int}$ trained on interactions and VAE$_{Comb}$ trained on interactions combined with job content (see "Appendix B" for a more detailed embedding analysis). In the case of the smallest dataset Studo, when we train the autoencoder only on interactions, more clusters are produced with sessions of different sizes close to each other (e.g., Fig. 6a). If the variational autoencoders are additionally trained on the job content, we can observe rainbow-colored shapes that are based on session length (e.g., as shown in Fig. 7a, b). In the larger CareerBuilder12 dataset, we end up with several sub-clusters that exhibit

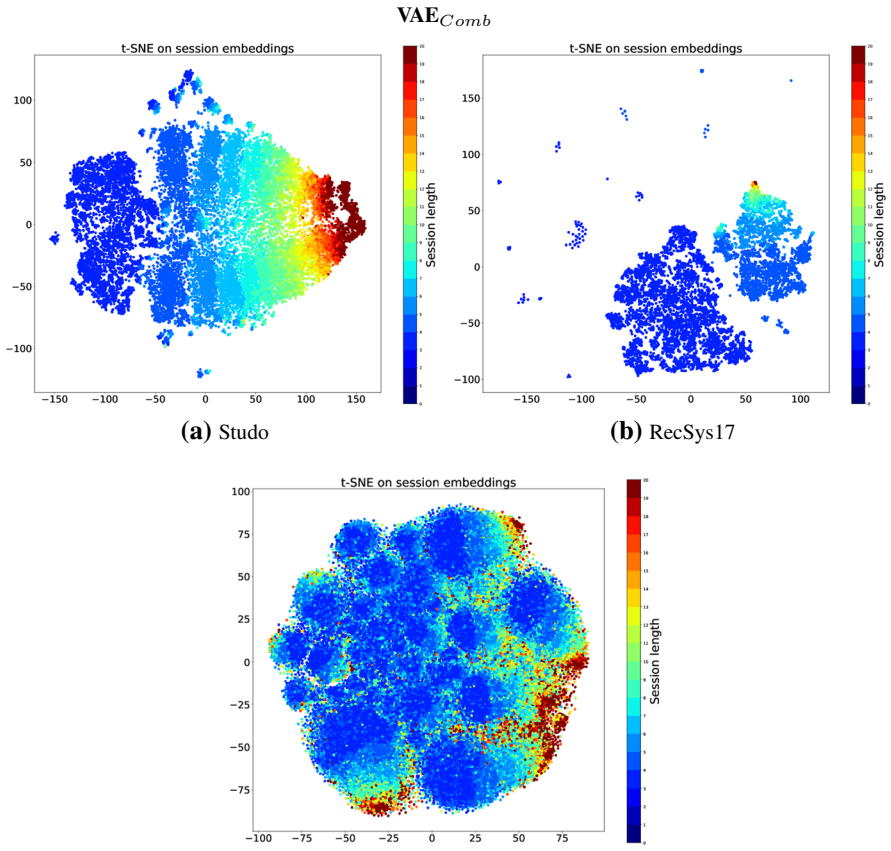


Fig. 7 The plots show t-SNE embeddings for latent session representations produced with the VAE autoencoder models trained on all three datasets using interaction data combined with the job content. The colors of the sessions reflect the session length, where the same red color is used for sessions with 20 or more interactions

this rainbow pattern. In other words, when we encode the input with content features, sessions of similar length tend to cluster. We attribute this to sessions of similar length having similar patterns of input vectors (e.g., many right-padded zeros for short sessions).

Next, we investigate the difference in recommendation accuracy between VAE$_{Int}$ and VAE$_{Comb}$ in light of the clustering patterns. The results suggest that when sessions cluster by similar size in the 2D space, as in the case of VAE$_{Comb}$ in the Studo and RecSys17 datasets and VAE$_{Int}$ in the CareerBuilder12 dataset, recommendation accuracy drops.

Table 3 Best performing hyperparameter settings for each evaluated baseline approach and dataset based on nDCG@20

Approach	Parameter	Studo	RecSys17	CareerBuilder12
BPR	$\lambda_{SESSION}$	0.25	0	0
	λ_{ITEM}	0.25	0	0
iKNN	λ	80	50	20
	α	0.75	0.75	0.75
sKNN	k	100	500	1000
	SAMPLING	Recent	Random	Random
	SIMILARITY	Cosine	Cosine	Jaccard
	POPULARITY BOOST	No	No	Yes
S-sKNN	k	100	500	1000
	SAMPLING	Recent	Random	Random
	SIMILARITY	Cosine	Jaccard	Cosine
	POPULARITY BOOST	No	No	Yes
V-sKNN	k	100	100	100
	SAMPLING	Recent	Random	Random
	SIMILARITY	Cosine	Cosine	Cosine
	POPULARITY BOOST	No	No	No
	WEIGHTING	Quadratic	Quadratic	Logarithmic
GRU4Rec	LOSS	top1-max	bpr-max-0.5	top1-max
	LAYERS	[100]	[100]	[1000]
	DROPOUT	0.2	0.2	0.2
	BATCH SIZE	32	32	32
pRNN	ACTIVATION	tanh	tanh	softmax
	LAYERS	[1000]	[100]	[1000]
	α	0.001	0.01	0.001
	BATCH SIZE	512	512	512

5.3 Hyperparameter optimization of the baseline approaches

We conducted a grid search on the hyperparameters for the baseline approaches using the validation set, i.e., two weeks of user interactions. As such, Table 3 reports on the best performing configurations for each approach and dataset in terms of recommendation accuracy (see "Appendix A" for more details).

BPR We performed a grid search that includes three different values for the regularization of session features $\lambda_{SESSION} \in \{0.0, 0.25, 0.5\}$ and the regularization of item feature $\lambda_{ITEM} \in \{0.0, 0.25, 0.5\}$.

iKNN For the iKNN approach, we evaluated the values for regularization (i.e., to avoid coincidental high similarities of rarely visited items) $\lambda \in \{20, 50, 80\}$ and the normalization factor for the support between two items $\alpha \in \{0.25, 0.5, 0.75\}$.

sKNN, S-sKNN and V-sKNN For all the sKNN variations that we utilize in this paper, we conducted a grid search for the parameter k (i.e., 100, 200, 500 or 1000),

Table 4 Prediction results ($k = 20$) of remaining jobs that will be subject to interaction within a session. (Color table online)

Studo						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	.2724	.1636	.0214	.0174	70.12	100
VAE_{Comb}	.2593	.1597	.0198	.0162	82.27	100
sKNN	.2552	.1403	.0193	.0165	55.18	99.67
V-sKNN	.2766	.1679	.0209	.0178	60.67	100
S-sKNN	.2687	.1532	.0203	.0172	56.26	99.67
GRU4Rec	.2909	.1682	.0230	.0202	53.74	99.02
pRNN	.0895	.0500	.0060	.0058	20.70	29.97
Bayes	.1560	.0758	.0138	.0144	72.73	100
iKNN	.1718	.0864	.0153	.0158	62.65	99.67
BPR-MF	.0700	.0485	.0059	.0028	40.59	69.71
POP	.0501	.0276	.0014	.0048	1.80	2.61
RecSys Challenge 2017						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	.2307	.1962	.0468	.0070	35.82	90.68
VAE_{Comb}	.2299	.1948	.0466	.0068	35.84	90.53
sKNN	.2180	.1782	.0129	.0033	36.33	91.03
V-sKNN	.1931	.1521	.0115	.0033	36.24	88.93
S-sKNN	.2221	.1829	.0131	.0034	36.91	91.43
GRU4Rec	.1040	.0811	.0065	.0024	46.63	73.45
pRNN	.1024	.0593	.0003	.0021	7.87	11.17
Bayes	.0446	.0336	.0031	.0019	28.56	61.62
iKNN	.0565	.0414	.0038	.0024	35.04	70.59
BPR-MF	.2530	.1900	.0086	.0028	76.58	93.79
POP	.2294	.2278	.0001	.0047	0.13	0.30
CareerBuilder 2012						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	.1072	.0393	.0194	.0160	18.20	96.98
VAE_{Comb}	.1242	.0438	.0209	.0177	16.04	96.62
sKNN	.1406	.0454	.0161	.0146	13.77	92.45
V-sKNN	.1458	.0566	.0173	.0154	16.06	95.67
S-sKNN	.1428	.0462	.0166	.0150	14.61	95.03
GRU4Rec	.1415	.0554	.0172	.0159	20.46	83.60
pRNN	.0005	.0002	.0001	.0001	0.02	0.14
Bayes	.0842	.0383	.0094	.0077	11.87	78.53
iKNN	.1386	.0577	.0164	.0144	16.61	90.13
BPR-MF	.0005	.0001	.0001	.0001	78.75	95.36
POP	.0004	.0001	.0001	.0001	0.01	0.08

Table 4 (continued)

Coverage is reported for the ratio of recommended jobs compared to all jobs available in the data set (left) and jobs expected in the test set (right)

the sampling method of sessions (i.e., recent or random), the similarity function (i.e., cosine or Jaccard) and if popular items from neighboring sessions should be boosted. For V-sKNN, we also optimized the decay weighting function (i.e., division, logarithmic or quadratic).

GRU4Rec In the case of GRU4Rec, we experimented with two different loss functions {*top*1-*max*, *bpr-max*-0.5}, four variations of the number of GRU layers and their sizes {[100], [100, 100], [1000], [1000, 1000]}, a dropout applied to the hidden layer of {0.0, 0.2, 0.5} and batch sizes of {32, 128, 512}.

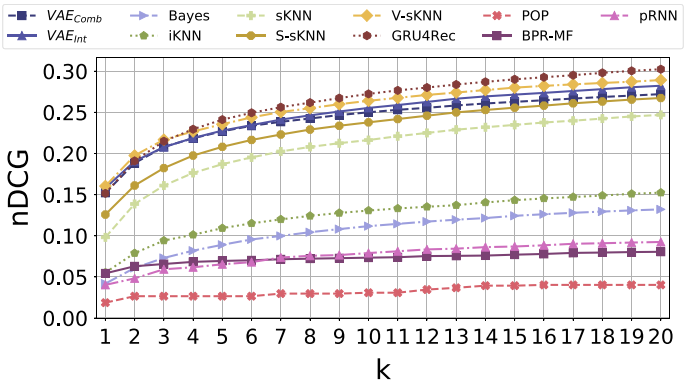
pRNN For the pRNN approach, we explored two activation functions {*softmax*, *tanh*} for the output layer, two sizes for the GRU layers {[100], [1000]}, a learning rate $\alpha \in \{0.01, 0.001\}$ and batch sizes of {32, 128, 512}. With respect to the batch size, however, due to the computational complexity of pRNN and the size of CareerBuilder12, we were only able to tune this hyperparameter for Studo and RecSys17. As we received the best results for a batch size of 512 for both datasets, we also used a batch size of 512 in case of CareerBuilder12.

5.4 Comparison with baseline approaches

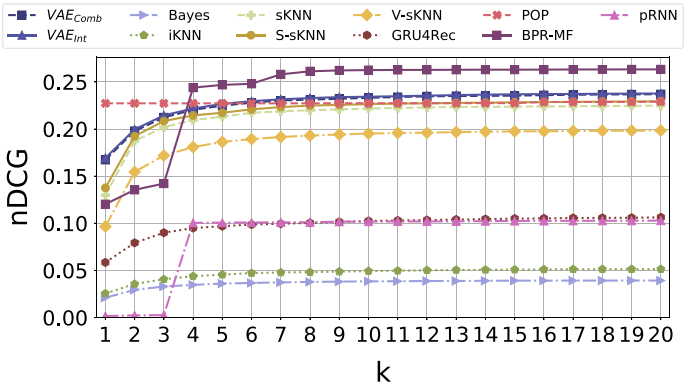
Table 4 shows the results of comparing VAE_{Int} and VAE_{Comb} with all baseline methods when we evaluate against the remaining jobs in the session. We report recommendation accuracy in terms of nDCG and MRR, as well as a system-based novelty (EPC), session-based novelty (EPD) and coverage. In the case of the next job prediction problem, in Figs. 8 and 9, we show nDCG and EPC results for different values of k (i.e., number of recommended jobs).

Accuracy (nDCG & MRR) On all datasets, the *sKNN*-based approaches achieve high accuracy in terms of nDCG and MRR, as shown in Table 4. In terms of both nDCG and MRR, VAE_{Int} performs second best in RecSys17, while it performs third best in Studo. For the Studo dataset, *GRU4Rec* has the highest accuracy for both metrics. In the RecSys17 dataset, *BPR-MF* performs best concerning nDCG, while *POP* performs best in terms of MRR. In CareerBuilder12, *V-sKNN* achieves the highest nDCG, while *iKNN* achieves the highest MRR. In this dataset, VAE_{Int} achieves medium performance, which we attribute to the ample item space and sparsity of CareerBuilder12. The VAE_{Comb} method, however, results in a higher recommendation accuracy, while training the model is much less expensive.

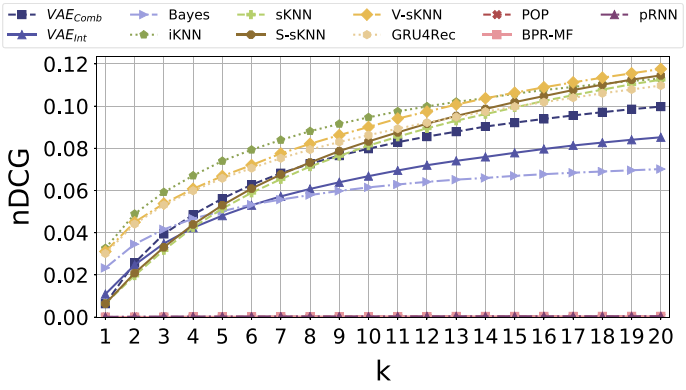
While the performance of *sKNN*-based approaches is rather stable, several baselines algorithms, namely *POP*, *BPR-MF*, *iKNN*, *Bayes*, *GRU4Rec* and *pRNN*, show



(a) Studo

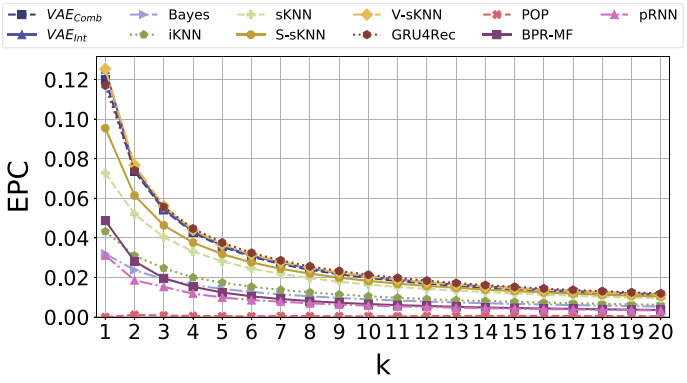


(b) RecSys Challenge 2017

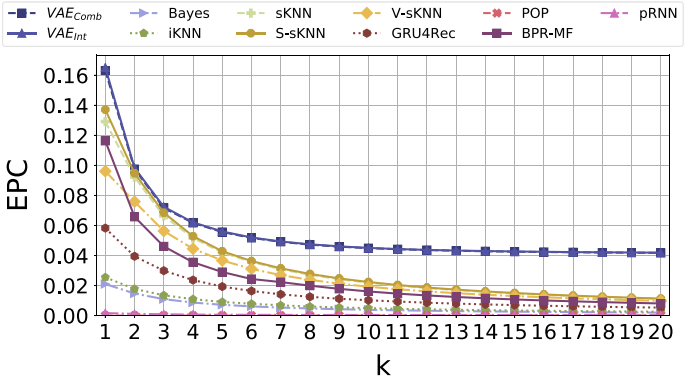


(c) CareerBuilder 2012

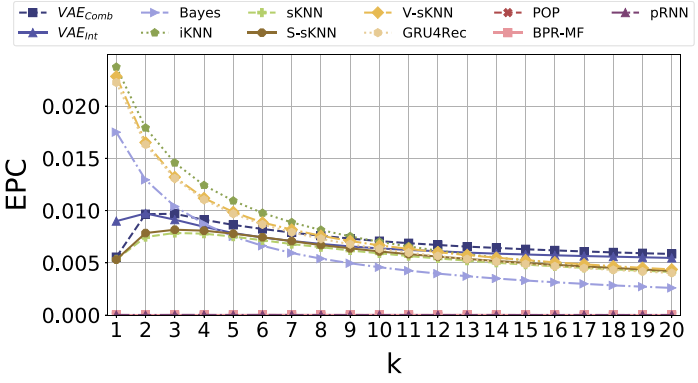
Fig. 8 nDCG results for different recommendation list sizes (i.e., values of k) when predicting the next job in the session. On all three datasets, both our proposed *VAE* approaches achieve competitive results concerning accuracy (i.e., nDCG) metrics



(a) Studo



(b) RecSys Challenge 2017



(c) CareerBuilder 2012

Fig. 9 EPC results for different recommendation list sizes (i.e., values of k) when predicting the next job in the session. On all three datasets, both our proposed *VAE* approaches achieve good results concerning beyond-accuracy (i.e., EPC) metrics

notable differences among the datasets. First, the *Bayes* approach establishes itself as a competitive baseline in the Studo dataset, whereas it results in a poor performance for the two larger datasets (i.e., RecSys17 and CareerBuilder12). In fact, for the RecSys17 dataset, it results in the worst performance. Hence, when the domain has a small number of items, it can be reasonable to employ such a simple and computationally inexpensive method.

Second, the accuracy of *POP* in the RecSys17 dataset is noteworthy.[14] The reason for this is that in the RecSys17 dataset, the most popular job from the train set was also the one with the highest number of interactions in the test set (i.e., around 21.5%). However, this approach will likely not result in high user satisfaction, just by predicting the same items repeatedly. Moreover, the *BPR-MF* performs best in terms of nDCG in the RecSys17 dataset, but it has the second worst performance in the other two datasets. Also, *GRU4Rec* performs worse for the RecSys17 dataset when compared to Studo and CareerBuilder12. We attribute this to bias toward popularity (Ludewig and Jannach 2018). The performance of *GRU4Rec* is low, while the performance of *BPR-MF* is high in the RecSys17 dataset. The *pRNN* method performs low on all three datasets, but its recommendation accuracy is especially weak on the CareerBuilder12 dataset. Finally, the performance of the *iKNN* differs among all three datasets. While it has the highest MRR for the CareerBuilder12 dataset, the performance in the RecSys17 dataset is the second lowest for both accuracy metrics.

For the next job prediction problem shown in Fig. 8, in all three datasets, all approaches show a similar accuracy performance. The results confirm the presence of bias toward popular items in the RecSys17 dataset as the popularity approach outperforms the other algorithms until $k = 3$, after which *BPR-MF* becomes the best performing approach. We also attribute the sudden increase in the nDCG values for *BPR-MF* and *pRNN* at the recommendation list of length 4 to this popularity bias in the dataset. A closer inspection revealed that both approaches often recommend highly popular items from the train set at the beginning of the recommendation list. The top-1 (i.e., most popular) item that is shared between the train and test set is also the one which gets recommended most frequently as the fourth item in the recommendation list of *BPR-MF* and *pRNN*. Besides that, for all values of k (i.e., the number of recommended jobs), the session-based KNN approaches and *GRU4Rec* achieve competitive accuracy values.

System-based novelty (EPC) As shown in Table 4, both *VAE* approaches achieve top results in terms of EPC for all three datasets. VAE_{Int} performs best on the RecSys17 dataset, while VAE_{Comb} outperforms all approaches in the CareerBuilder12 dataset. In the Studo dataset, VAE_{Int} achieves second best to *GRU4Rec*. Especially in the RecSys17 dataset, the difference in novelty is considerably high when compared to other baselines. For the baselines, the *sKNN* approaches and *GRU4Rec* both exhibit a good performance concerning the novelty of the recommended jobs. The *pRNN* method, as well as *POP* and *BPR-MF*, produces recommendations that have the lowest system-based novelty.

[14] Quadrana et al. (2017) report that their popularity approach outperforms session-based RNN (Hidasi et al. 2015) in the XING dataset used in the ACM RecSys Challenge 2016.

Table 5 Summary of the rankings of the session-based algorithms evaluated in the job domain. (Color table online)

	Accuracy	Beyond Accuracy	Coverage
VAE_{Int}	++	++	++
VAE_{Comb}	+	++	++
sKNN	+	o	+
V-sKNN	++	+	++
S-sKNN	++	+	+
GRU4Rec	++	+	+
pRNN	- -	- -	- -
Bayes	- -	- -	o
iKNN	o	-	+
BPR-MF	-	- -	++
POP	- -	- -	- -

"++" indicates best, "+" good, "o" average, "-" low and "- -" the worst ranking with respect to (1) accuracy (i.e., nDCG and MRR), (2) beyond-accuracy (i.e., EPC and EPD) and (3) coverage

In Fig. 9, we see that both our proposed *VAE* approaches outperform all others in the CareerBuilder12 dataset after $k = 9$. The *sKNN* baselines, as well as *GRU4Rec*, show a better novelty performance for a smaller number of recommended jobs.

Session-based novelty (EPD) As depicted in Table 4, both *VAE* approaches provide the best session-based novelty for the RecSys17 and CareerBuilder12 datasets and are competitive in the Studo dataset. The VAE_{Comb} method generates the most *surprising* recommendations in the largest dataset (i.e., CareerBuilder12) and *GRU4Rec* in the smallest dataset (i.e., Studo). In all cases, the *sKNN*-based approaches are a competitive baseline. We can observe the most notable difference between accuracy and EPD, however, in the CareerBuilder12 dataset, where the *VAE* approaches result in a rather average accuracy while performing very well concerning session-based novelty. Overall, the results indicate that both *VAE* approaches are suitable for cases when we aim to generate novel session-based recommendations.

Coverage In Table 4, we report the percentage of jobs, which were recommended and are a part of (1) all jobs available in the dataset (i.e., the complete item catalog), and (2) the jobs that we know anonymous session users will interact within the test set (i.e., the expected item catalog).

In terms of the coverage of all possible job postings, VAE_{Comb} performs best in the Studo dataset. *BPR-MF* covers at the most the entire item catalog in the RecSys17 and CareerBuilder12 dataset. Concerning the coverage of items in the test set (i.e., expected items), the session-based KNN approaches achieve almost perfect coverage in the Studo dataset. Only in the case of the RecSys17 dataset, the

BPR-MF baseline has an even higher coverage. As expected, the *POP* baseline results in the worst coverage. While this baseline has high accuracy values in the RecSys17 dataset (due to the popularity bias inherent in this dataset), it effectively covers only a small fraction of jobs in the system. It also has to be noted that the *pRNN* baseline always has the second-worst coverage. As the available item catalog grows, the coverage drops, which suggests that the trained model focuses on a specific (i.e., relatively small) set of items, which explains the worse performance in the largest dataset (i.e., CareerBuilder12).

5.5 Performance overview

To provide a better overview of the performance of the different session-based job recommendation approaches, we summarize all results in Table 5 with respect to three metric categories. That is, we report the performance on accuracy (i.e., *nDCG* and *MRR*), beyond accuracy (i.e., *EPC* and *EPD*) and coverage (of the whole dataset and the test set). For every approach, we assign a rank (i.e., from 1 to 11) for the particular metric in a dataset. We then aggregate these rankings across all three metric categories and datasets. The final rankings are then normalized and assigned into five performance buckets (i.e., from worst "- -" to best "++"; see "Appendix C" for the calculation steps).

Concerning accuracy, the best performance is achieved by *V-sKNN*, our VAE_{Int} variant, *S-sKNN* and *GRU4Rec*. This is then followed by VAE_{Comb} and *sKNN*. All other baselines achieve worse accuracy. For the beyond accuracy metric category, both of our *VAE* variants achieve the best performance. This is followed by *GRU4Rec* and the *sKNN* variants. A similar observation can be made for the metric category coverage. Here, however, *BPR-MF* also shows the best, *iKNN* good and the simple *Bayes* baseline medium coverage. Noteworthy is also the ranking score of the VAE_{Comb}, as with our proposed method it is possible to train the autoencoder models faster (i.e., even with a large item space) and without the need to frequently retrain the utilized model to consider new jobs coming to the system. The *pRNN* approach did not achieve a good rank in any metric category. The same is true for *POP*.

6 Conclusion and future work

In this work, we addressed the problem of providing job recommendations in an anonymous, online session setting. In three datasets, i.e., Studo, RecSys17 and CareerBuilder12, we evaluated the efficacy of using different autoencoder architectures to produce session-based job recommendations. Specifically, we utilized autoencoders to infer latent session representations, which are used in a k-nearest neighbor manner to recommend jobs within a session. We evaluated two types of

input for the autoencoders: (1) interactions with job postings within browsing sessions and (2) a combination of interactions with job postings and content features extracted from these job postings.

We found that variational autoencoders trained on interaction and content data, and used in a *k*-nearest neighbor manner, led to very good results in terms of accuracy compared to other autoencoder variants. A visual analysis of the embedding spaces with t-SNE revealed that we could attribute a lower accuracy performance when similar-sized sessions form clusters in the 2D space. Although this was mostly the case for autoencoders trained on content features, in practice, however, such an approach has the advantage of fixed size vectors, which means retraining is needed less often. Consequently, depending on the application scenario, one can decide which input for the variational autoencoder to take, i.e., to balance frequent retraining and accuracy.

Furthermore, we evaluated all autoencoder and baseline approaches with respect to beyond-accuracy metrics, i.e., system-based and session-based novelty as well as coverage, in two settings: Firstly, we compared the recommendation performance of the approaches on all remaining interactions within a session, and secondly, we predicted the next job interaction in the session. We find that our proposed variational autoencoder methods can outperform state-of-the-art approaches for sessions-based recommender systems with respect to system-based and session-based novelty. Besides, the session-based KNN approaches are a competitive baseline for the variational autoencoder methods with respect to accuracy and coverage.

For future work, we aim to explore the use of generative variational autoencoder models to directly recommend jobs from the reconstructed session vector (e.g., in a similar way as in Liang et al. 2018). Other ideas for future work include investigating all approaches used in this study in an online evaluation. We plan to conduct an online study to ask users how satisfied and surprised they are with job recommendations generated by autoencoders. Also, we plan to evaluate the accuracy in an A/B test to conclude whether a higher system-based and session-based novelty in a session-based offline setting leads to higher user satisfaction. Additionally, we also plan to directly optimize for the beyond-accuracy metrics by incorporating re-ranking techniques (e.g., maximum marginal relevance Carbonell and Goldstein 1998). These evaluations are planned to be carried out in the Talto [15] career platform. In summary, we hope that the approach presented in this paper will attract further research on the effectiveness of dimensionality reduction techniques

[15] Talto (https://talto.com) is the successor of the jobs platform in Studo (http://www.studo).

for session-based job recommender systems and the effect of such methods on beyond-accuracy metrics such as system-based and session-based novelty as well as coverage.

Limitations Our work has several limitations. So far, we only focused on autoencoders to infer the latent representation of the anonymous user session. While autoencoders are a popular choice to reduce the dimensionality of data, other deep neural networks such as restricted Boltzmann machines (Nguyen et al. 2013), deep belief networks (Srivastava and Salakhutdinov 2012) or convolutional neural networks (Shen et al. 2014) could also serve well for this task. Furthermore, additional metadata information about jobs (e.g., textual content of job postings) could potentially enhance recommendations, which we did not tackle due to the unavailability of such data in all datasets. So far, we did not compare the approaches used in this study concerning computational performance, like the authors of (Ludewig and Jannach 2018) did. Moreover, in this work, we did not investigate how to model repeated interactions on the same job postings. Although this is implicitly considered by the autoencoder variants that combine interactions with job content features, such actions are not taken into account by the autoencoders that solely rely on interaction data. Also, in this work, we extracted the candidate sessions based on the last job interaction, which is a limitation of our work. For the evaluation procedure, we used a single time-based split for our experiments. One approach to assess the robustness of our results would be to apply a sliding window approach to generate splits with varying lengths. However, the size of the Studo and RecSys17 datasets is limited, which makes such an approach infeasible. For the larger CareerBuilder dataset, a sliding-window-based evaluation approach could be applied to test the robustness of the method. Due to computational constraints, for the present work, we used the same time-based split as for the Studo and the RecSys17 datasets. We leave the exploration of more splits to future work.

Another limitation is that one of the datasets we used for our study, the Studo dataset, is proprietary, and due to the terms of service of Moshbit, the owner of Studo, it cannot be made available for others at this point.

Acknowledgements Open access funding provided by Graz University of Technology. This work is supported by the Know-Center, the Institute of Interactive Systems and Data Science (ISDS) of Graz University of Technology and Moshbit. We thank Moshbit for granting access to their dataset in Studo Jobs. We thank Simone Kopeinik, Dieter Theiler, Tomislav Duricic and Leon Fadjevic for their feedback on this manuscript.

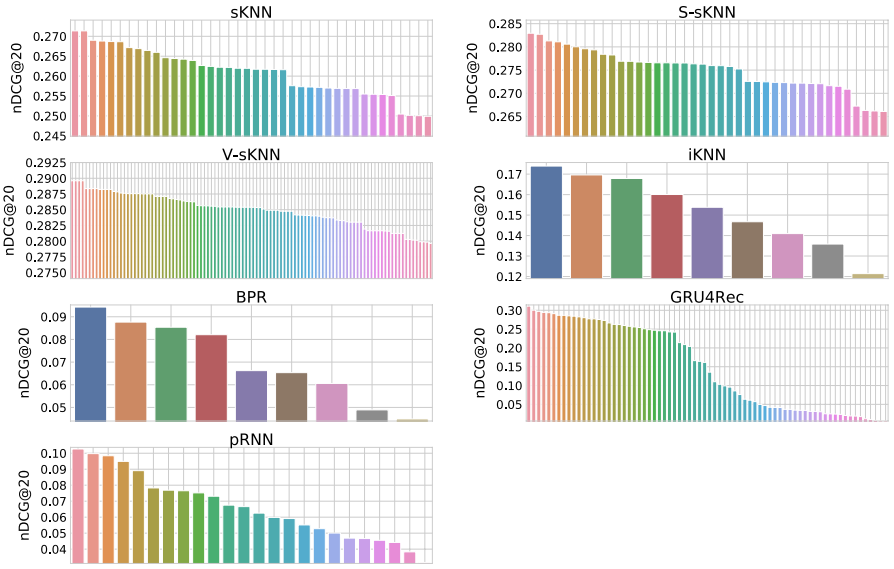


Fig. 10 Accuracy results for the different hyperparameters of the baseline approaches on the Studo dataset

Appendices

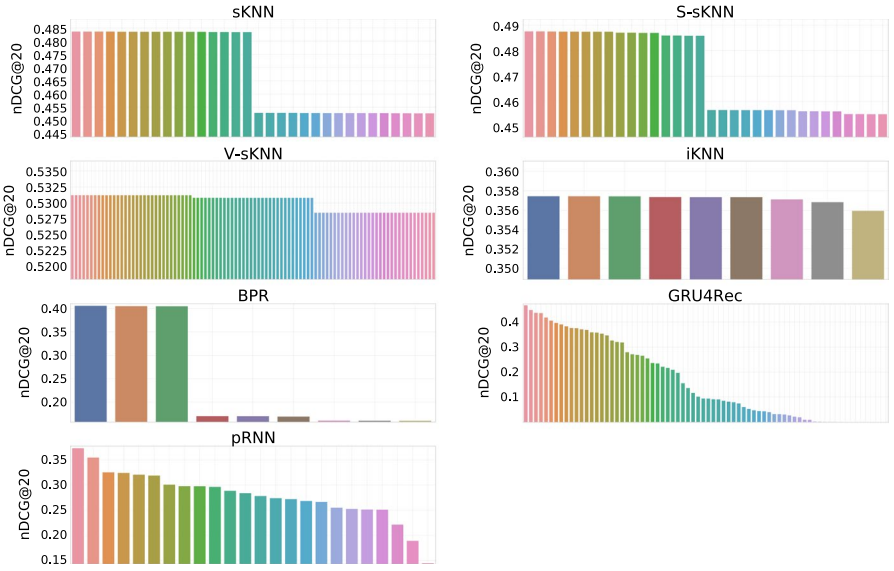


Fig. 11 Accuracy results for the different hyperparameters of the baseline approaches on the RecSys17 dataset

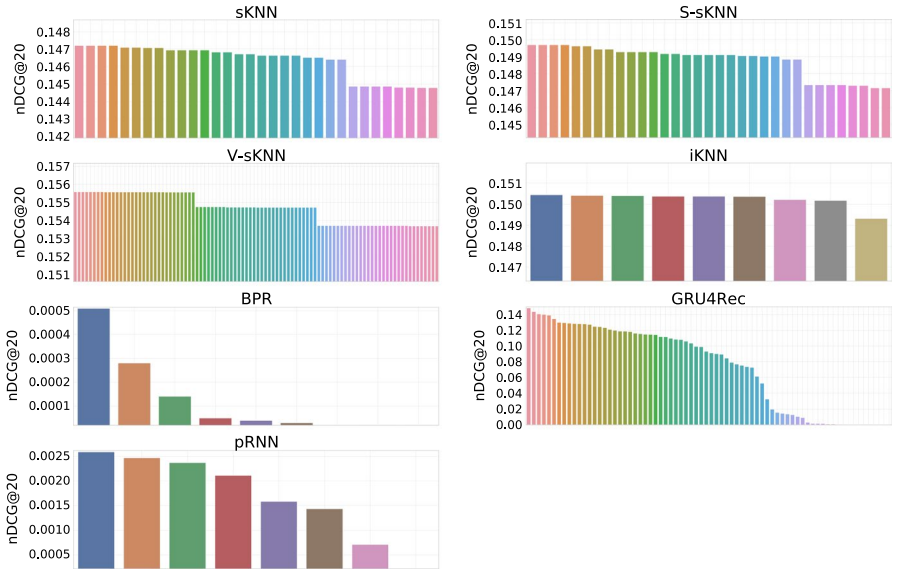


Fig. 12 Accuracy results for the different hyperparameters of the baseline approaches on the CareerBuilder12 dataset

Hyperparameter optimization results

In this section, we report the distribution of the accuracy results achieved by optimizing the hyperparameters for the baseline approaches in Sect. 5.3. For each baseline approach, we pick those hyperparameters which showed the best performance with respect to nDCG@20. As such, Fig. 10 shows the differences between the evaluated baseline configurations on the Studo dataset. Respectively, Fig. 11 depicts the results for the RecSys17 and Fig. 12 for the CareerBuilder12 dataset.

Autoencoder embedding analysis

Figure 13 shows all autoencoder models as t-SNE plots for the Studo dataset, i.e., AE_{Int}, DAE_{Int} and VAE_{Int} trained on interactions and AE_{Comb}, DAE_{Comb} and VAE_{Comb} trained on the combination of interactions and job content. The same is reported for RecSys17 in Fig. 14 and CareerBuilder12 in Fig. 15.

The results indicate that both denoising autoencoders and variational autoencoders tend to produce more session clusters than a classic autoencoder, which creates more of a linear pattern of neighboring sessions. In some cases, we can observe that both the classic and denoising autoencoder models produce shapes without clear structure and large dispersion (e.g., see Fig. 13d or 15b), which indicates that it is hard to find a clear neighborhood of similar sessions. For the smaller Studo dataset, if the autoencoders are solely trained on interactions, i.e., AE_{Int}, DAE_{Int} and VAE_{Int}, more clusters are produced with sessions of different

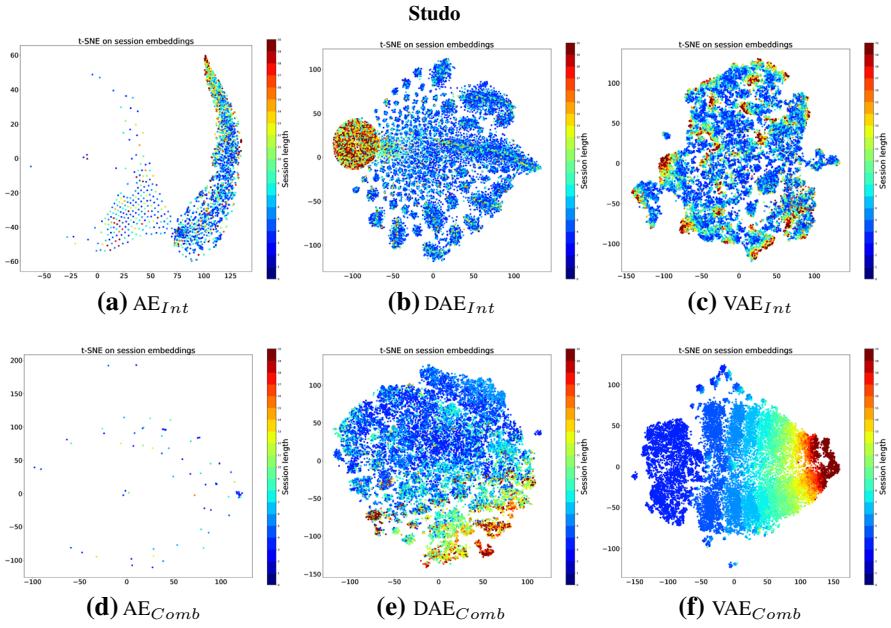


Fig. 13 t-SNE embeddings for latent session representations produced with the three autoencoder models trained on interaction and content data from the Studo dataset. Sessions are colored according to their length, where the same red color is used for sessions with 20 or more interactions

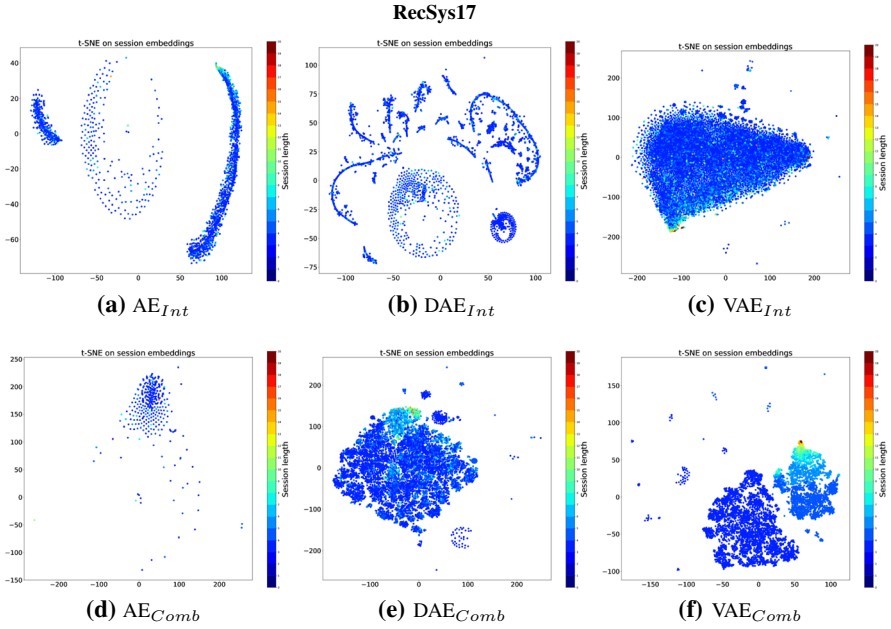


Fig. 14 t-SNE embeddings for latent session representations for the RecSys17 dataset

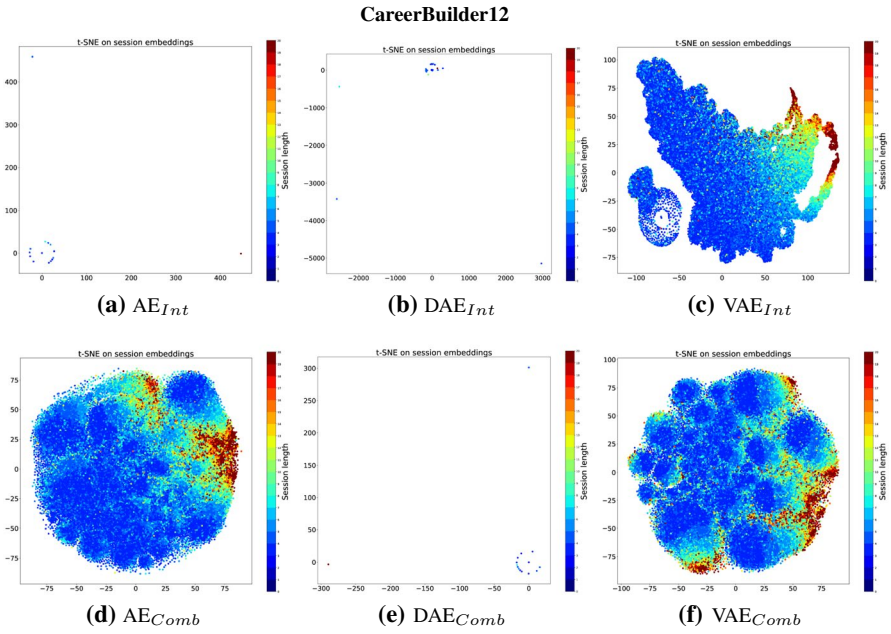


Fig. 15 t-SNE embeddings for latent session representations for the Careerbuilder12 dataset

sizes close to each other (e.g., Fig. 13b, c). Interestingly, if autoencoders are trained on content, we can observe rainbow-colored shapes that are based on session length (e.g., as shown in Fig. 13e, f). In case of a larger dataset like CareerBuilder12, we end up with several sub-clusters that exhibit this rainbow pattern. This shows that when we encode the input with content features, sessions of similar length tend to cluster. We attribute this to sessions of similar length having similar patterns of input vectors (e.g., many right-padded zeros for short sessions).

Aggregation of rankings

In Sect. 5.5, we report the aggregated performance of the different approaches. For this, in Table 6 we first rank the results from each dataset (i.e., based on Table 4). We then sum the rankings for each dataset (i.e., Studo, RecSys17 and CareerBuilder12) for the accuracy metrics (i.e., nDCG and MRR), the beyond-accuracy metrics (i.e., EPC and EPD) and both coverage, respectively. The aggregated rankings are outlined in Table 7. The rankings are then normalized with the equation

Table 6 Ranking of the results per metric and dataset, which are derived from numerical results. (Color table online)

Studo						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	3	3	2	3	3	1
VAE_{Comb}	5	4	5	6	1	1
sKNN	6	6	6	5	7	5
V-sKNN	2	2	3	2	5	1
S-sKNN	4	5	4	4	6	5
GRU4Rec	1	1	1	1	8	8
pRNN	9	9	9	9	10	10
Bayes	8	8	8	8	2	1
iKNN	7	7	7	7	4	5
BPR-MF	10	10	10	11	9	9
POP	11	11	11	10	11	11
RecSys Challenge 2017						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	2	2	1	1	7	4
VAE_{Comb}	3	3	2	2	6	5
sKNN	6	6	4	5	4	3
V-sKNN	7	7	5	5	5	6
S-sKNN	5	5	3	4	3	2
GRU4Rec	8	8	7	8	2	7
pRNN	9	9	10	10	10	10
Bayes	11	11	9	11	9	9
iKNN	10	10	8	8	8	8
BPR-MF	1	4	6	7	1	1
POP	4	1	11	3	11	11
CareerBuilder 2012						
	nDCG	MRR	EPC	EPD	Coverage (%)	
VAE_{Int}	7	7	2	2	3	1
VAE_{Comb}	6	6	1	1	6	2
sKNN	4	5	7	6	8	6
V-sKNN	1	2	3	4	5	3
S-sKNN	2	4	5	5	7	5
GRU4Rec	3	3	4	3	2	8
pRNN	9	9	9	9	10	10
Bayes	8	8	8	8	9	9
iKNN	5	1	6	7	4	7
BPR-MF	9	10	9	9	1	4
POP	11	10	9	9	11	11

Coloring is according to the rank within each dataset

Table 7 Aggregated rankings across the three different datasets and per metric type (i.e., accuracy, beyond accuracy and coverage)

	Accuracy		Beyond Accuracy		Coverage	
	Aggregated	Normalized	Aggregated	Normalized	Aggregated	Normalized
VAE_{Int}	24	0.1176	11	0.0217	19	0.0208
VAE_{Comb}	27	0.2059	17	0.1522	21	0.0625
sKNN	33	0.3824	33	0.5000	33	0.3125
V-sKNN	21	0.0294	22	0.2609	25	0.1458
S-sKNN	25	0.1471	25	0.3261	28	0.2083
GRU4Rec	24	0.1176	24	0.3043	35	0.3542
pRNN	54	1.0000	56	1.0000	60	0.8750
Bayes	54	1.0000	52	0.9130	39	0.4375
iKNN	40	0.5882	43	0.7174	36	0.3750
BPR-MF	44	0.7059	52	0.9130	25	0.1458
POP	48	0.8235	53	0.9348	66	1.0000

Results are then normalized by a min-max scaling

$Norm(x) = \frac{x-min+1}{max-min+1}$, where *min* is the lowest aggregated rank and *max* is the highest aggregated rank. Thus, lower results are considered better, while the worst results receive the value 1. The results are then put into five buckets according to their values. A double plus (i.e., ++) is assigned to values between 0.0 and 0.2, while values between 0.2 and 0.4 get assigned a single plus (i.e., +), followed by *o* (i.e., 0.4 until 0.6), − (i.e., 0.6 until 0.8) and for the worst results a −− (i.e., 0.8 until 1).

References

Abel, F.: We know where you should work next summer: job recommendations. In: Proceedings of the 9th ACM Conference on Recommender Systems, pp. 230–230 (2015)

Abel, F., Benczúr, A., Kohlsdorf, D., Larson, M., Pálovics, R.: Recsys challenge 2016: job recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 425–426. ACM (2016)

Abel, F., Deldjoo, Y., Elahi, M., Kohlsdorf, D.: Recsys challenge 2017: offline and online evaluation. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 372–373 (2017)

Adomavicius, G., Kwon, Y.: Improving aggregate recommendation diversity using ranking-based techniques. IEEE Trans. Knowl. Data Eng. **24**(5), 896–911 (2012)

Aggarwal, C.C.: Evaluating recommender systems. In: Recommender Systems, pp. 225–254. Springer (2016)

Al-Otaibi, S.T., Ykhlef, M.: A survey of job recommender systems. Int. J. Phys. Sci. **7**(29), 5127–5142 (2012)

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems, pp. 153–160 (2007)

Bianchi, M., Cesaro, F., Ciceri, F., Dagrada, M., Gasparin, A., Grattarola, D., Inajjar, I., Metelli, A.M., Cella, L.: Content-based approaches for cold-start job recommendations. In: Proceedings of the Recommender Systems Challenge 2017, p. 6. ACM (2017)

Bonnin, G., Jannach, D.: Automated generation of music playlists: survey and experiments. ACM Comput. Surv. (CSUR) **47**(2), 26 (2015)

Carbonell, J., Goldstein, J.: The use of mmr, diversity-based reranking for reordering documents and producing summaries. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 335–336 (1998)

Chatzis, S.P., Christodoulou, P., Andreou, A.S.: Recurrent latent variable networks for session-based recommendation. In: Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems, pp. 38–45. ACM (2017)

Fischer, A., Igel, C.: An introduction to restricted Boltzmann machines. In: Iberoamerican Congress on Pattern Recognition, pp. 14–36. Springer (2012)

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web, pp. 173–182. International World Wide Web Conferences Steering Committee (2017)

Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. (TOIS) **22**(1), 5–53 (2004)

Hidasi, B., Karatzoglou, A.: Recurrent neural networks with top-k gains for session-based recommendations. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 843–852. ACM (2018)

Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939 (2015)

Hidasi, B., Quadrana, M., Karatzoglou, A., Tikk, D.: Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 241–248. ACM (2016)

Hidasi, B., Tikk, D.: General factorization framework for context-aware recommendations. Data Min. Knowl. Discov. **30**(2), 342–371 (2016)

Hong, W., Zheng, S., Wang, H., Shi, J.: A job recommender system based on user clustering. JCP **8**(8), 1960–1967 (2013)

Jannach, D., Ludewig, M.: When recurrent neural networks meet the neighborhood for session-based recommendation. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 306–310 (2017)

Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods for graphical models. Mach. Learn. **37**(2), 183–233 (1999)

Kamehkhosh, I., Jannach, D., Ludewig, M.: A comparison of frequent pattern techniques and a deep learning method for session-based recommendation. In: RecTemp@ RecSys, pp. 50–56 (2017)

Kenthapadi, K., Le, B., Venkataraman, G.: Personalized job recommendation system at linkedin: practical challenges and lessons learned. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 346–347 (2017)

Kenthapadi, K., Le, B., Venkataraman, G.: Personalized job recommendation system at linkedin: practical challenges and lessons learned. In: Proceedings of the 11th ACM Conference on Recommender Systems, pp. 346–347 (2017)

Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint. arXiv:1312.6114 (2013)

Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)

Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE J. **37**(2), 233–243 (1991)

Lacic, E., Kowald, D., Traub, M., Luzhnica, G., Simon, J., Lex, E.: Tackling cold-start users in recommender systems with indoor positioning systems. In: Poster Proceedings of the 9th ACM Conference on Recommender Systems (2015)

Lacic, E., Reiter-Haas, M., Duricic, T., Slawicek, V., Lex, E.: Should we embed? A study on the online performance of utilizing embeddings for real-time job recommendations. In: Proceedings of the 13th ACM Conference on Recommender Systems, pp. 496–500. ACM (2019)

Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., Ma, J.: Neural attentive session-based recommendation. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 1419–1428. ACM (2017)

Liang, D., Krishnan, R.G., Hoffman, M.D., Jebara, T.: Variational autoencoders for collaborative filtering. arXiv preprint. arXiv:1802.05814 (2018)

Lin, Z., Feng, M., Santos, C.N.d., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding. arXiv preprint. arXiv:1703.03130 (2017)

Liu, Q., Zeng, Y., Mokhosi, R., Zhang, H.: Stamp: short-term attention/memory priority model for session-based recommendation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1831–1839. ACM (2018)

Liu, R., Rong, W., Ouyang, Y., Xiong, Z.: A hierarchical similarity based job recommendation service framework for university students. Front. Comput. Sci. **11**(5), 912–922 (2017)

Ludewig, M., Jannach, D.: Evaluation of session-based recommendation algorithms. User Model. User-Adap. Inter. **28**(4–5), 331–390 (2018)
Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. J. Mach. Learn. Res. **9**(Nov), 2579–2605 (2008)
Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders. arXiv preprint. arXiv:1511.05644 (2015)
Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting methods for incremental matrix factorization in recommender systems. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 947–953. ACM (2015)
McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: ACM CHI'06 (2006)
Mine, T., Kakuta, T., Ono, A.: Reciprocal recommendation for job matching with bidirectional feedback. In: 2013 Second IIAI International Conference on Advanced Applied Informatics, pp. 39–44. IEEE (2013)
Mishra, S.K., Reddy, M.: A bottom-up approach to job recommendation system. In: Proceedings of the Recommender Systems Challenge, p. 4. ACM (2016)
Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, pp. 807–814 (2010)
Nguyen, T.D., Tran, T., Phung, D., Venkatesh, S.: Learning sparse latent representation and distance metric for image retrieval. In: 2013 IEEE International Conference on Multimedia and Expo (ICME), pp. 1–6. IEEE (2013)
Parikh, A., Täckström, O., Das, D., Uszkoreit, J.: A decomposable attention model for natural language inference. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 2249–2255 (2016)
Parra, D., Sahebi, S.: Recommender systems: sources of knowledge and evaluation metrics. In: Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis, pp. 149–175. Springer (2013)
Pu, P., Chen, L., Hu, R.: A user-centric evaluation framework for recommender systems. In: Proceedings of the fifth ACM conference on Recommender systems, pp. 157–164 (2011)
Quadrana, M., Karatzoglou, A., Hidasi, B., Cremonesi, P.: Personalizing session-based recommendations with hierarchical recurrent neural networks. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 130–137. ACM (2017)
Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: Bpr: Bayesian personalized ranking from implicit feedback. In: Proceedings of the Twenty-fifth Conference on Uncertainty in Artificial Intelligence, pp. 452–461. AUAI Press (2009)
Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. arXiv preprint. arXiv:1401.4082 (2014)
Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, pp. 285–295. ACM (2001)
Sedhain, S., Menon, A.K., Sanner, S., Xie, L.: Autorec: autoencoders meet collaborative filtering. In: Proceedings of the 24th International Conference on World Wide Web, pp. 111–112. ACM (2015)
Shani, G., Heckerman, D., Brafman, R.I.: An mdp-based recommender system. J. Mach. Learn. Res. **6**(Sep), 1265–1295 (2005)
Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G.: A latent semantic model with convolutional-pooling structure for information retrieval. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 101–110. ACM (2014)
Siting, Z., Wenxing, H., Ning, Z., Fan, Y.: Job recommender systems: a survey. In: 2012 7th International Conference on Computer Science Education (ICCSE), pp. 920–924 (2012)
Smirnova, E., Vasile, F.: Contextual sequence modeling for recommendation with recurrent neural networks. arXiv preprint. arXiv:1706.07684 (2017)
Song, Y., Elkahky, A.M., He, X.: Multi-rate deep learning for temporal recommendation. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 909–912. ACM (2016)
Srivastava, N., Salakhutdinov, R.: Learning representations for multimodal data with deep belief nets. In: International Conference on Machine Learning Workshop, Vol. 79 (2012)
Strub, F., Gaudel, R., Mary, J.: Hybrid recommender system based on autoencoders. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 11–16. ACM (2016)

Tan, Y.K., Xu, X., Liu, Y.: Improved recurrent neural networks for session-based recommendations. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 17–22 (2016)
Theis, L., Shi, W., Cunningham, A., Huszár, F.: Lossy image compression with compressive autoencoders. arXiv preprint. arXiv:1703.00395 (2017)
Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop, coursera: neural networks for machine learning. University of Toronto, Technical Report (2012)
Tuan, T.X., Phuong, T.M.: 3d convolutional networks for session-based recommendation with content features. In: Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 138–146. ACM (2017)
Twardowski, B.: Modelling contextual information in session-aware recommender systems with neural networks. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 273–276. ACM (2016)
Vargas, S., Castells, P.: Rank and relevance in novelty and diversity metrics for recommender systems. In: Proceedings of the Fifth ACM Conference on Recommender Systems, pp. 109–116. ACM (2011)
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, pp. 1096–1103. ACM (2008)
Volkovs, M., Yu, G.W., Poutanen, T.: Content-based neighbor models for cold start in recommender systems. In: Proceedings of the Recommender Systems Challenge 2017, p. 7. ACM (2017)
Voorhees, E.: Proceedings of the 8th Text Retrieval Conference. TREC-8 Question Answering Track Report, pp. 77–82 (1999)
Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. arXiv preprint. arXiv:1811.00855 (2018)
Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, pp. 346–353 (2019)
Wu, Y., DuBois, C., Zheng, A.X., Ester, M.: Collaborative denoising auto-encoders for top-n recommender systems. In: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, pp. 153–162. ACM (2016)
Xiao, W., Xu, X., Liang, K., Mao, J., Wang, J.: Job recommendation with hawkes process: an effective solution for recsys challenge 2016. In: Proceedings of the Recommender Systems Challenge, p. 11. ACM (2016)
Yuan, F., Karatzoglou, A., Arapakis, I., Jose, J.M., He, X.: A simple convolutional generative network for next item recommendation. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 582–590 (2019)
Zhang, C., Cheng, X.: An ensemble method for job recommender systems. In: Proceedings of the Recommender Systems Challenge, p. 2. ACM (2016)
Zhang, Y.C., Séaghdha, D.Ó., Quercia, D., Jambor, T.: Auralist: introducing serendipity into music recommendation. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, pp. 13–22. ACM (2012)
Zhou, T., Kuscsik, Z., Liu, J.G., Medo, M., Wakeling, J.R., Zhang, Y.C.: Solving the apparent diversity-accuracy dilemma of recommender systems. Proc. Natl. Acad. Sci. **107**(10), 4511–4515 (2010)

Emanuel Lacic is a Senior Researcher and Recommender Systems Architect in the Social Computing team at the Know-Center. He is a PhD student at Graz University of Technology and a former visiting researcher at the Computer Science department of the University of California, Los Angeles. He has an M.Sc. and B.Sc. in Software Engineering and Information Systems from the University of Zagreb. His research interests are in the fields of Recommender Systems, Deep Learning and Social Network Analysis.

Markus Reiter-Haas is a researcher at Moshbit GmbH and is responsible for the recommender system of the Talto career platform. He has a background in Computer Science at the Graz University of Technology with a focus on Knowledge Technologies. His master thesis tackled the evaluation of student job recommendations on the Talto predecessor Studo Jobs. His current research concentrates on creating low-dimensional embeddings for effective retrieval in the job domain.

Dominik Kowald is a post-doctoral researcher and deputy research area manager of the Social Computing team at the Know-Center. He has a Ph.D. (with hons), M.Sc. (with hons) and B.Sc. in Computer Science from Graz University of Technology. His research interests are in the fields of recommender systems, fairness and biases in algorithms, and computational social science, in which he has published more than 60 papers so far.

Manoj Reddy Dareddy is a Ph.D. Candidate in the Computer Science department at the University of California Los Angeles. His research interest is in recommender systems and applied machine learning. More specifically, Manoj works on emerging frontiers in personalization such as privacy and explainability. He received his Masters from the University of Michigan Ann Arbor and Bachelors from Carnegie Mellon University in Qatar.

Junghoo Cho is a professor in the Department of Computer Science at the University of California, Los Angeles. He received a Ph.D. degree in Computer Science from Stanford University and a B.S. degree in physics from Seoul National University. His research interest is in the theory and practice of learning, particularly in the area of language acquisition and understanding. He is a recipient of prestigious awards such as the 10-Year Best Paper Award at VLDB 2010, NSF CAREER Award or IBM Faculty Award.

Elisabeth Lex is an assistant professor and head of the Social Computing Lab at Graz University of Technology, Austria. She received a Ph.D. and an M.Sc. degree in Computer Science from Graz University of Technology. Her research interests are in the development of personalized recommender systems, in particular, algorithms based on psychological theory, as well as in computational social science, more specifically, using behavioral and network data to investigate human activity and social dynamics.

Affiliations

Emanuel Lacic[1] · Markus Reiter-Haas[2] · Dominik Kowald[1] · Manoj Reddy Dareddy[3] · Junghoo Cho[3] · Elisabeth Lex[4]

Emanuel Lacic
elacic@know-center.at

Markus Reiter-Haas
markus.reiter-haas@moshbit.com

Dominik Kowald
dkowald@know-center.at

Manoj Reddy Dareddy
mdareddy@cs.ucla.edu

Junghoo Cho
cho@cs.ucla.edu

1 Know-Center GmbH, Graz, Austria

2 Moshbit GmbH, Graz, Austria

3 University of California, Los Angeles, USA

4 Graz University of Technology, Graz, Austria