On the Value of Encoding Sessions for Job Recommendation

ABSTRACT

In this work, we address the problem of recommending jobs in an anonymous session setting. For this, we propose to use autoencoders in order to extract latent session embeddings which can be used in a k-nearest neighbor manner. We demonstrate with different variations of autoencoders that by applying dimensionality reduction on session information, we outperform current state-of-the-art session-based recommendation techniques with respect to recommendation accuracy. We also investigate the lack of non-accuracy evaluations on session-based recommenders and show that such recommendations do not need to suffer from a usually far too high serendipity.

KEYWORDS

Job Recommendations; Session-based Recommendation; Autoencoders; Session Embeddings; Novelty; Serendipity;

1 DETAILED DESCRIPTION OF THE RESEARCH PROJECT

Research on recommender systems has gained tremendous popularity in recent years. Especially since the hype of the social Web and the rise of social media and networking platforms like Twitter or Facebook, recommender systems are acknowledged as an essential feature helping users to, for instance, discover new connections between people and resources (e.g., products, news articles, job advertisements, etc.). The advent of the Big Data era has additionally posed the need for high scalability and real-time processing of frequent data updates, and thus, has brought new challenges for the recommender systems research community. Traditional recommender systems usually analyze the data offline and update the generated model in regular time intervals. However, in many domains, choices made by users depend on factors which are susceptible to change anytime. Lets take a shopping mall for example, where a user triggers frequent indoor location updates via a smartphone application while moving through the mall. Employing an offline model update strategy that lasts hours or days may potentially miss the current location context of the user and fail to provide the right recommendations to match users real-time shopping demand. As a consequence, being able to capture users real-time interests is gaining momentum and is currently of high demand.

In many online systems where recommendations are applied (e.g., like in the example mentioned above), interactions between a user and the system are organized into sessions. A session is a group of interactions that take place within a given time frame. Sessions from a user can occur on the same day, or over several days, weeks, or months. A session usually has a goal, such as finding a product to buy, reading news articles of a certain style (e.g., sports or politics) or searching for a new job position to apply . But, when user identifiers are not available, to propagate information from the previous user session to the next is not possible and makes inherently the recommendation problem even harder. Providing a recommendation in such a setting and under real-time constraints poses unique challenges for classical methods. For example, the popular matrix factorization approach breaks down in the session-based setting when no user profile can be constructed from past user behavior.

The session-based recommendation problem shares some similarities with some NLP-related problems in terms of modeling as long as they both deals with sequences. The past few years have seen the tremendous success of deep neural networks in a number of sequential data modeling tasks (e.g., speech recognition). A lot of attention as the model of choice for this type of data has been especially received by various flavors of Recurrent Neural Networks (RNN). While RNNs have been applied to different NLP domains with remarkable success, only recently has some attention been paid to the area of recommender systems. In the session-based recommendation we can consider the first item a user clicks when entering a web-site as the initial input of the RNN, we then would like to query the model based on this initial input for a recommendation. Each consecutive click of the user will then produce an output (a recommendation) that depends on all the previous clicks. Typically the item-set to choose from in recommenders systems can be in the tens of thousands or even hundreds of thousands. Apart from the large size of the item set, another challenge is that click-stream datasets are typically quite large thus training time, scalability and especially run-time performance are really important.

As seen, session-based recommendation is a characteristic challenge that cannot be properly addressed by conventional methodologies employed in the context of real-time recommender systems. Specifically, under a session-based setup, a recommendation is based only on the actions of a user during a specific browsing session. Indeed, this type of recommendation generation approach is based on tracking user actions during an active session. Based on the captured and inferred session-based user behavioral patterns, the aim of this work is to predict the following user actions during that session, and proactively recommend items/actions to them. A higher degree of difficulty lies additionally on making the developed algorithms suitable for running in a live setting. To test this out, this work will investigate the problem of session-based job recommendations and show how Deep Learning methods can be utilized and adapted to recommend jobs in real-time.

2 INTRODUCTION

With the advent of the Web, majority of our activities are being performed online. One important such activity is looking for employment. The nature of the job market is highly competitive and finding a new job is not an easy decision as it usually depends on many factors like salary, job description or geographical location. This has led to the recent rise of business-oriented social networks

WSDM '19, February 11, 2019, Melbourne, Australia

DOI: N/A

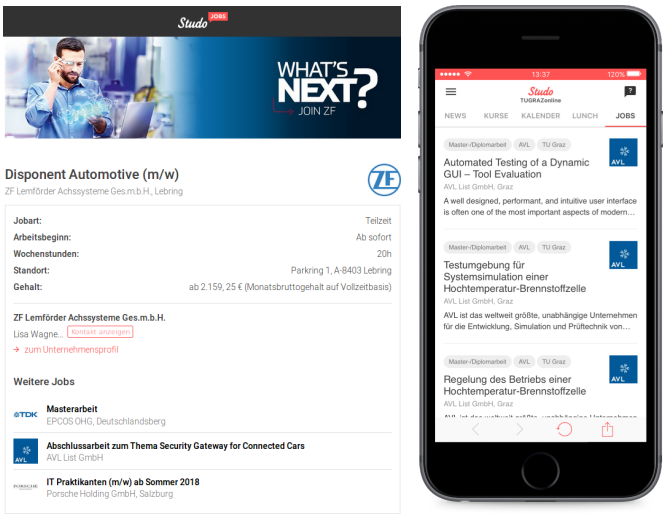


Figure 1: Job recommendation in Studo shown either on the web or within the mobile application.

like LinkedIn[1] or XING[2]. Users of such networks organize and look after their profile by describing their skills, interests and previous work experiences. But finding relevant jobs even for users with such carefully structured content is a non-trivial task to perform [1]. To make the problem even harder, most job portals offer their users the option to browse the available jobs in an anonymous fashion and the only information available to the system are the interactions within a short-lived session. Since users interact with the system anonymously, it is not possible to correlate any past interactions of the users with the current one. Moreover, users do not change jobs quite often and when they do, it can easily be in a session-based manner where we are basically constantly tackling the cold-start problem. Thus, tackling job recommendation in a session-based setting is an important task for the recommendation community to solve.

Present work. In this paper, we present our work on session-based job recommendations with the focus on utilizing interaction data. We recently started to tackle this topic on the Austrian online platform Studo[3]. Studo provides guidance and support for about one third of Austrian students by offering services such as finding the right job (as seen in Figure 1). But tackling the job recommendation problem gets more difficult for university students, as they normally have only some or no relevant work experience at all. This has become a real issue for students as they get more aware that having a degree does not automatically guarantee them their desired job after graduation (e.g., [20]). As such, it becomes more important to provide relevant job recommendations even when they browse anonymously in a session-based environment.

Unlike movies, songs, or books, where items of recommendation have a reasonably long lifespan, job postings are ephemeral in nature. Once the employer has selected a candidate, the corresponding job posting will be removed from the system. As new job postings are added, their lifespan depends on various factors such as demand and supply in the job market. In order to cope with the ephemeral nature of the job domain in an online system, we investigate a novel utilization of autoencoders for recommending jobs in a session-based environment and compare them to other state-of-the-art approaches used for session-based recommendation (e.g., [16, 19]). By looking at the majority of work on session-based recommendation, we also notice that the sole focus of the investigated approaches has been on evaluating the accuracy (e.g., [15, 17, 40]). With the growing awareness that factors other than accuracy contribute towards the quality of recommendations [14, 29, 46], in this paper, we further address the lack of non-accuracy evaluations in session-based recommendations and compare our approach as well as current state-of-the-art approaches on how novel and serendipitous the recommendations are. Thus, we raise the following two research questions:

RQ1: How can autoencoders be utilized for session-based job recommendation?
RQ2: How novel and serendipitous are session-based job recommendations?

In order to address these research questions, we employ the interaction data collected on the Studo platform. In addition to that, we also experiment on the dataset provided by XING after the RecSys Challenge 2017 [3]. Since only interaction data is considered, the models depends only on the session behaviour and thus external factors, such as content length of the job posting, do not need to be considered for the training of the models. We show that utilizing autoencoders can result in the best accuracy performance for both datasets. Moreover, we show that session-based algorithms usually suffer from a potentially too high serendipity which can alleviate the risk of having a distrust effect in the underlying recommendation system [11]. Using or adapting a session-based KNN approach however results in recommendations which are not geared towards such an extreme case of unexpectedness. This effect however comes at the cost of slightly sacrificing novelty in the recommended jobs.

In summary, this study may help researchers to get an insight on how to apply autoencoders in a session-based setting. Moreover, we address the lack of non-accuracy evaluations in session-based recommendations and compare our approach as well as current state-of-the-art approaches on how novel and serendipitous their recommendations are.

Outline. This paper is structured as follows: we first discuss related work in Section 3. Then, we present our methodology on using autoencoders for session-based job recommendation in Section 4. Then, we present the dataset, utilized baselines and metrics which we use for evaluation in Section 5 and the results of our experiments in Section 6. Finally, in Section 7, we conclude the paper and provide an outlook on future work.

3 RELATED WORK

At present, we identify three main lines of research related to our work: (i) job recommendation, (ii) session-based recommendation, and (iii) recommending with neural networks.

Job recommendation. Research on job recommmendations has mostly focused on improving the accuracy by applying various

[1] http://linkedin.com
[2] http://xing.com
[3] http://studo.co/

Collaborative- and Content-Based Filtering approaches or their hybrid combinations [5, 45]. In [27], the problem was inverted to search for the right users for a given job and then to recommend this job to the users. Hong et al. [18] proposed a clustering based approach whereby they cluster user profiles and apply several recommendation algorithms on each cluster depending on their effectiveness. In 2016 and 2017, XING (a career-oriented social networking site based in Europe) organized a challenge for the ACM RecSys conference to build a job recommendation system [2, 3]. Their anonymized dataset has sparked a lot of interest among researchers to build recommendation systems geared towards the job domain. For instance, Mishra et al. [30] looked into using machine learning-based methods and proposed to build a gradient boosting classifier that can predict if a given user will like a particular job posting. They design a set of hand-picked features based on information such as education level, experience or location, just to name a few.

Session-based recommendation. Most recommender systems assume the presence of user preference history in the form explicit or implicit interactions that can be used to determine accurate recommendations using techniques such as matrix factorization. In such situations, the assumption is that there exists a profile for every user based on their feedback. In many cases, such user profile information cannot be constructed due to various reasons like, to protect privacy of users, inadequate resources, etc. In such scenarios, session-based recommender systems are appropriate as they only model the user within a session, i.e., a short period of time when the user is actively interacting with the system. A naive approach for session-based recommendation is to recommend similar items using item-item similarity as proposed by Sarwar et al. [36]. Hidasi et al. [17] propose a general factorization framework that models a session using the average of the component latent item representations. Other use Markov Decision Processes to compute recommendations that are based on the transition probability between items [37] or adapt nearest neighbor methods for a session-based setting [19, 39]. As discussed next, applying neural networks in recommendation systems has gained a lot more attention in recent years. Out of the different neural architectures, recurrent neural networks have become especially popular when providing session-based recommendation [9, 16, 38].

Recommending with neural networks. Earlier work has focused on explicit feedback and solving the rating prediction problem with neural networks. For that, Restricted Boltzmann Machines was a popular choice to serve as a basis for Collaborative Filtering [12, 47]. Newer work however identified the importance of implicit feedback. One recent approach related to our work is Collaborative Denoising Autoencoder (CDAE) [44]. CDAE utilizes a denoising autoencder (see also later in Section 4.1) by adding a latent factor for each user to the input. In CDAE, the number of parameters grows linearly with the number of users and items. As the number of sessions is normally much larger than the number of users, this would make it much more prone to overfitting than modeling sessions as we do in Section 4, where the model grows linearly with the number of items (see also [26]). Another related approach is neural collaborative filtering [13]. The authors explore applying non-linear interactions between user and item latent factors instead of using the dot product. However, their model would also grow linearly with the number of sessions and jobs if applied in our case. As Liang et al. [26] recently found out, this becomes problematic when the datasets gets much larger. In their work, it was also shown how to use the generative variational autoencoder (VAE) model together with a β multiplier for regularization for collaborative filtering [26]. Although we focus on learning latent session representations for KNN, as we discuss later in Section 6, we did try to use the generative VAE model for directly recommending jobs but did not achieve a competitive accuracy performance in a session-based setting. Other recent work like the one from Hidasi et al. [16] propose a Recurrent Neural Network (RNN) based approach to model variable-length session data. They showed that recurrent neural networks (e.g., Gated Recurrent Unit) can be adapted for this task. Other papers on sequential data mostly build on this idea and either improve the original algorithm [15, 40] or extend it by capturing additional information like context [41] or attention [25].

4 METHOD

With respect to *RQ1*, in this work, we investigate the impact of applying autoencoders within a session-based KNN job recommender. We formulate our problem as follows: given a target session s_t, which interacted with at least one job j_i from the set of available jobs $J = \{j_1,...,j_n\}$, our goal is to predict the jobs that the target session user will interact with next. Due to the ephemeral nature of job postings, sessions interact with jobs in an implicit way, thus allowing us to represent the current session as a binary encoded vector with a dimension that is equal to the number of jobs in the underlying dataset. In previous work [19], the highly dimensional sparse session vectors were directly used to calculate the similarity between sessions.

In this paper, we propose to reduce the dimensionality needed to represent the session vectors using autoencoders[4]. The main idea behind this is that knowledge relative to session preferences is hidden in raw data and dimensionality reduction techniques can exploit this. An overview of the utilized autoencoder variants is given in the following section 4.1. For each one, we specify an output z that is the latent representation of a session. As seen in Figure 2, we first extract z 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 to first find the top-k similar past sessions. In order to reduce the computational burden and allow for real-time recommendation[5], we extract a subset of all sessions that have interacted with the last job in s_t. Using z, we calculate the cosine similarity between the respective target and candidate session and use the top-k similar sessions to recommend jobs. The score of a job that is used to rank the recommendations is then calculated as follows:

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

where $1_{s_i}(j_i)$ is 1 if the candidate session s_i contains the job j_i and 0 otherwise (as in [8, 19]).

[4]Implementation of our approach as well as a more detailed hyperparameter description available at: https://github.com/sbreco/session_rec_ae

[5]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 of time.

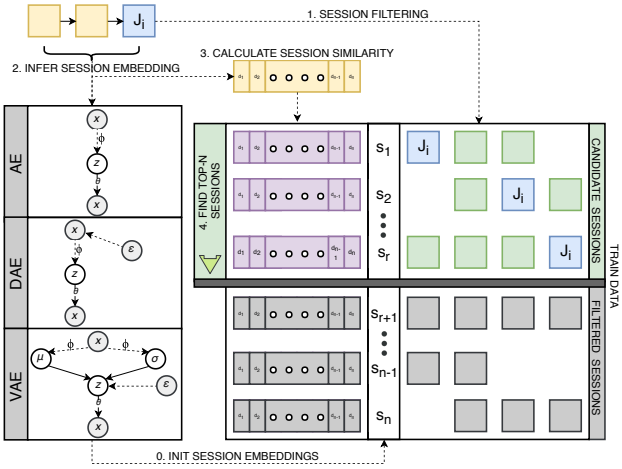


Figure 2: Using the trained autoencoders, we infer latent representation for (i) sessions in the training data and (ii) the current target session for which we recommend jobs. Jobs from the top-k similar candidate sessions (filtered by the currently interacted job posting) are recommended to the target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

Online job recommendations. In order to compute the session similarity using the latent representations in real-time, for Studo we store the learned session vectors in Lucene (i.e., as part of the utilized Apache Solr search engine) in the form of payloads. Payloads are a general purpose array of bytes that are associated with a Lucene token at a certain position. Each session is thus annotated with multiple positions of the latent vector dimensions. Using a custom made plugin[6] , the latter positional information is used for a speedy retrieval and calculation of vector similarities (i.e., using the cosine similarity) at runtime. Moreover, for a lower computational burden, we do not limit the candidate sessions using their recency as similar work does [19]. The reason for that is the ephemeral nature of job postings. Sessions are tied to browsing available jobs and filtering on a specific job posting should already result in sessions that are recent enough. Actually, due to the inverted-index data structure available in search engines like Apache Solr, it is computationally much cheaper to filter sessions out rather than sorting them based on time.

4.1 Overview of Autoencoders

In this section, we describe how we extract the latent session representation z using autoencoders. Autoencoders are a type of neural network, which were popularized by Kramer [24] as a more effective method than PCA when it comes to describing and reducing the dimensionality of data. An autoencoder is learned 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. As seen in Figure 2, in this work, we investigate three variations of autoencoder architectures to represent job sessions: (i) a classical Autoencoder (AE), (ii) a Denoising Autoencoder (DAE), and (iii) a Variational Autoencoder (VAE).

[6]https://github.com/sbreco/session_rec_ae/tree/master/vec_plugin

Autoencoder (AE). The simplest form of an autoencoder is with only one hidden layer between the input and output [6]. The hidden layer takes the session vector $x_s \in \mathbb{R}^D$ and maps it to a latent representation $z_s \in \mathbb{R}^K$ through a mapping function:

$$z = h(x) = \sigma(W^T x + b)$$

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:

$$\hat{x} = \sigma(W' z + b')$$

By adding one or more layers between the hidden and input layer, we create an *encoder* and by doing the same between the hidden and output layer, we create a *decoder*, hence the name autoencoder. During inference, we use the output of the hidden layer (i.e., bottleneck) to represents the latent session vector z_s. In our experiments, we use a sigmoid activation function and a $D_s - 256 - 100 - 256 - D_s$ architecture[7], where D_s is the dimension of the original session vector that equals the number of jobs in the underlying dataset. We stack the layers in the network using Restricted Boltzmann Machines (RBMs) and train the model using stochastic gradient descent by minimizing the Kullback-Leibler divergence [10].

Denoising Autoencoder (DAE). As shown by Vincent et al. [42], extending autoencoders by corrupting the input can show surprising advantages. The idea of a 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 encoder layer with a probability of 0.2. We use the same $D_s - 256 - 100 - 256 - D_s$ architecture as above, but this time, we use classical autoencoder layers instead of RBMs (as noted in [43], stacking classical autoencoders yields almost as good a performance as when stacking RBMs). We again apply a sigmoid activation function and stochastic gradient descent but minimize the cross entropy [44].

Variational Autoencoder (VAE). Another approach to extract the latent representation z is by using variational inference [21]. 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:

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

where μ and σ^2 is the encoded output given the input x of a session. To be more precise, additional neural networks are here utilized as probabilistic encoders (and decoders). Most commonly this is done using a multi-layered perceptron (MLP) and for the above mentioned $q_\phi(z|x)$ we calculate:

[7]We also tested higher values for the dimension of the hidden layer but did not find enough accuracy improvement that would justify the additional computation burden when calculating session similarities in real-time.

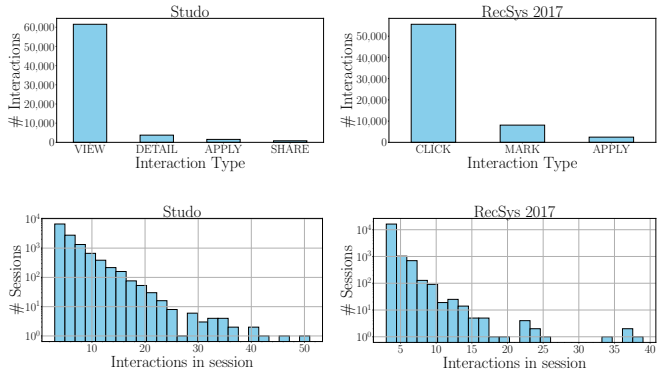


Figure 3: Number of interactions based on the interaction type (top) and the distribution of session sizes (bottom) is shown for the Studo (left) and RecSys Challenge 2017 (right) datasets.

$$\mu = W_2\, h + b_2$$
$$\log \sigma^2 = W_3\, h + b_3$$
$$h = \tanh(W_1\, x + b_1)$$

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 jobs from J. As we deal with implicit data, to calculate the probabilities we let $p_\theta(x|z)$ be a multivariate Bernoulli [23] whose probabilities in the MLP we calculate as:

$$\log p(x|z) = \sum_{i=1}^{D} x_i \log y_i + (1 - x_i) \cdot \log(1 - y_i)$$
$$y = f_\theta(\, W_5 \tanh(W_4\, z + b_4) + b_5\,)$$

where f_θ is an element-wise non-linear activation function (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 [23, 26], which in case of VAEs is the Kullback-Leibler divergence [10] of $q_\phi(z_s|x_s)$ and $p(z_s|x_s)$. Because we are sampling z_s from q_ϕ in the variational lower bound, in order to learn the model, we also need to apply the reparametrization trick [23, 35] by sampling $\epsilon \sim \mathcal{N}(0, I_K)$ (also seen in Figure 2) and reparametrize $z_s = \mu_\phi(x_s) + \epsilon \odot \sigma_\phi(x_s)$. 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).

5 EXPERIMENTS

In this section, we provide a detailed description of our empirical study of using session encodings. First, we describe the datasets. Second, we introduce state-of-the art baselines for session-based recommendation that we compare against. Finally, we introduce the evaluation metrics that we use to compare the utilized session-based job recommendation approaches as well as explain how we setup the experiments.

5.1 Dataset

We employ two different datasets from the job domain. The first dataset utilizes the data from the Studo platform, which provides guidance and support for about one third of Austrian students. The proprietary dataset contains job interactions from anonymous user sessions for a period of 3 months between December 2017 and March 2018. The second dataset that we use is the latest version of the data provided by XING after the RecSys Challenge 2017 [3].

As seen on the top of Figure 3, in Studo, we have four different interaction types, i.e., job *view*, show company *details*, *apply* and *share* job. The dataset from the RecSys Challenge 2017 had originally six different interaction types, but for the purpose of this paper, we only kept the *click*, *bookmark* and *apply* interactions. We removed the *delete recommendation* and *recruiter interest* interactions as these were not relevant in our setting. Moreover, we also discarded the *impression* interaction as these denote that XING showed the corresponding job to a user. As stated by [7], the presence of an impression does not actually imply that the user interacted with the job and would thus introduce bias and possibly lead to learning a model that mimics XINGs recommender engine. Overall, the distribution of interaction types is similar between the datasets where the *click* or *view* interactions mostly dominate. With respect to sessions, we manually partitioned the interaction data of the RecSys dataset into sessions by using a 30-minute idle threshold (as in [33]). For Studo, we already assign the session ids to the job interactions when collected. As seen on the bottom of Figure 3, the log histogram of session sizes follows a similar skewed pattern, where the longest sessions was in the Studo dataset with 51 interactions. The statistics of the utilized datasets are given in Table 1. As seen, both of them are sparse. However, Studo has a much smaller number of available jobs that can be recommended.

5.2 Baselines

We utilize well-known baselines and compare our approach to current state-of-the art methods for session-based recommendation:

POP. A simple and yet often a strong baseline for session-based recommendation is the popularity approach. 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 within the session. As in [16], 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 [34]. As in [16], we use the average of job feature vectors of these jobs that had occurred in the current session as the user feature vector to apply it directly for session-based recommendation. That is, similarities of the feature vectors are averaged between a recommendable job and the jobs of the current session.

Dataset	# Interactions	# Sessions	# Jobs	Sparsity
Studo	59,192	11,832	596	99.16%
RecSys17	55,380	16,322	15,686	99.98%

Table 1: Statistics of the Studo and RecSys Challenge 2017 datasets.

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:

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

This is a simple approach which is, from a computational perspective, inexpensive to calculate and run in an online real-time system. Although not a popular choice, as we see later in Section 6 such an approach can be a competitive baseline when session-based recommendations are provided within a smaller item space.

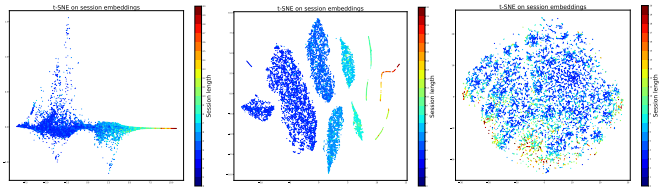
GRU4Rec. Recently Hidasi et al. [16] showed that recurrent neural networks are excellent models for data that is generated from users in an session-based manner. They utilize Gated Recurrent Units with a session-parallel mini-batch training process as well as ranking-based loss functions. In this work, we use their most recent improvement of GRU4Rec [15] from 2017 where they improved the performance with a new class of loss function tied together with an improved sampling strategy. We decided for this RNN approach as a baseline as the authors state that their model is the current best performer so far. Moreover, methods which utilize auxiliary information and also compare themselves to Tan et al. [40] (e.g., the NARM model with an attention-based mechanism [25]) do not have in all evaluation scenarios the same or better relative accuracy improvement.

sKNN. Recent research has shown that computationally simple nearest-neighbor methods can still be effective for session-based recommendation [19]. The session-based KNN first determines the k most similar past sessions in the training data. By encoding the sessions as binary vector of the item space and using cosine similarity, the set of k nearest sessions are found for the current session. The final job score is calculated by aggregating the session similarity over all sessions that contain the candidate job.

5.3 Evaluation metrics

As already stated, session-based recommendation quality is most commonly measured using one of a number of accuracy metrics. To quantify the performance of each of our recommendation approaches with respect to *RQ2*, we not only focus on measuring accuracy but also look into the impact on non-accuracy metrics like novelty and serendipity.

Measuring accuracy. nDCG is a ranking-dependent metric that not only measures how many jobs are correctly predicted but also takes the position of the jobs in the recommended list into account [31]. 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 Cummulative Gain (DCG@k)* which is given by [31]:



(a) AE **(b) DAE** **(c) VAE**

Figure 4: t-SNE embedding for latent session representations produced with the three autoencoder models for Studo. Sessions are colored by their length where the same red color is shared for sessions with 20 or more interactions.

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

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. Taken together over all sessions, it is given by:

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

Measuring novelty. Recommendation novelty can be seen as the ability of a recommender to introduce sessions to job postings that have not been (frequently) experienced before. A recommendation that is accurate but not novel will include items that the session user enjoys, but (probably) already knows of. Optimizing on novelty has been shown to have a positive, trust-building impact on user satisfaction [32]. Moreover, applying to too popular jobs may decrease the session user's satisfaction due to high competition and less chance of getting hired (see e.g., [22]). In our experiments, we measure novelty with a normalized metric previously introduced by [48]:

$$Novelty@k = 1 - \frac{1}{|S|} \sum_{s \in S} \frac{1}{k} \sum_{i \in k} \frac{\log_2(pop_i + 1)}{\log_2(pop_{MAX} + 1)}$$

This way, we quantify the average information content of job recommendations, where higher values mean that more globally *unexplored* jobs are being recommended. If the likelihood that a session user has experienced a job is proportional to its global popularity, this can serve as an approximation of true novelty.

Measuring serendipity. In contrast to novelty, serendipity incorporates the semantic content of jobs and represents how *surprising* or *unexpected* the recommendations are [46]. Given a distance function $d(i,j)$ that represents the dissimilarity between two jobs i and j, the serendipity is given as the average dissimilarity of all job pairs in the list of recommended jobs and jobs in the current session history [48]. In our experiments, we use the cosine similarity to measure the dissimilarity of two job postings using a raw job vector which contains a 1 if a session interacted with it and 0 otherwise:

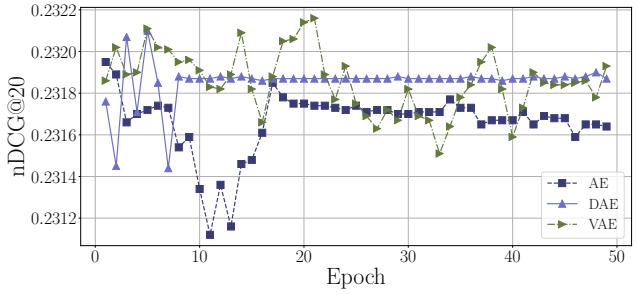


Figure 5: Prediction accuracy when comparing the three autoencoder variations when training the models for 50 epochs on the RecSys Challenge 2017 dataset.

$$Serendipity@k = \frac{1}{|S|}\sum_{s\in S}\left(\frac{1}{|R_k|\cdot|H_s|}\sum_{i\in R_k}\sum_{j\in H_s} d(i,j)\right)$$

where H_s is the current history of a session s, R_k is the set of the first k recommended jobs and $d(i,j) = 1 - sim(i,j)$. As it can be seen later in Section 6, session-based recommendations can get highly serendipitous. However, providing the highest possible level of *unexpectedness* would be unreasonable and cause problems because it might lead to users being dissatisfied with the underlying recommendation system [4, 11].

Measuring coverage. Another important factor of a recommender is how many jobs it 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 (i) on the full dataset, i.e., how many jobs of all available ones did we recommend and, (i) on the test dataset, i.e., how many jobs of the ones that we would expect to be interacted with did we manage to recommend.

5.4 Experimental setup

We first preprocessed both datasets from Section 5.1 to create a train and test set using a time-based split. The sessions from the last 14 days (i.e., 2 weeks) were put in the test set of the respective dataset and the rest was used for training. For each set, we also kept only sessions with a minimal number of 3 interactions. Like [33], we filtered items in the test set that do not belong to the training set as this allows better comparison against model based approaches (e.g., RNNs), which can only recommend items that have been used to train the model with. For the RecSys Challenge 2017 dataset, this resulted in 12,712 sessions to be used for training and 3,610 sessions for testing. In Studo there are 9,620 session to train and 1,752 to test on.

We initially train the models on the respective train data. In order to evaluate the performance of the utilized session-based recommendation algorithms, for each session in the test data, we iteratively sub-sample 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 (i.e., same as in [15, 16]). We start this procedure for every session after the first interaction and end before the last one. In this setting, we explore two evaluation cases: comparing the recommended jobs with (i) the remainder of the interactions in the session and (ii) with the next job interaction (i.e., next item prediction).

6 RESULTS

In this section, we discuss the results of our session-based job recommender approach. The overall results when evaluating recommendations against the remaining session interactions are presented in Table 2. In case of next item prediction, we only report accuracy results for different values of k in Figure 6 since we observe little difference between both evaluation cases.

Embedding analysis. We initially analyzed the shared embedding space by applying the t-SNE algorithm [28] to reduce the dimensionality of the inferred latent session representations within a $2D$ space. As seen in Figure 4, employing a denoising or varational autoencoder results in a better clustering of session embeddings. By utilizing the denoising autoencoder, we get visible clusters of sessions with similar lengths (i.e., the number of job interactions within the session). In contrast, the variational autoencoder enables more sessions of different sizes to be closer to each other.

Accuracy performance. We report the accuracy of our experiments in Table 2 with respect to *nDCG*@20. On both datasets, by inferring the latent representation of a session from the trained autoencoder models and then using it in a KNN manner yields the best results. The accuracy performance between the individual autoencoder variations does not show major differences. It needs to be noted that we report the results after training the autoencoders for 5 epochs and using top-100 similar sessions for recommendation. We experimented on different values for the k similar sessions to use when extracting candidate jobs to recommend (i.e., setting k from 50 to 500), but found that we usually achieve the best accuracy when k is 100 or larger. We did an additional analysis of the performance between the autoencoder variations with respect to training the models longer for up to 50 epochs. As depicted in Figure 5, $sKNN_{VAE}$ achieves the highest peaks while $sKNN_{DAE}$ stabilizes after 10 epochs.

One interesting observation when looking at the baselines is that the *Bayes* approach has established itself as competitive baseline in the Studo dataset, whereas for the RecSys Challenge 2017 dataset, it resulted in the worst performance. This suggests that in cases when the domain where we provide real-time session-based recommendations has a small number of items, it is very much reasonable to also utilize such a simple and computationally inexpensive method.

The accuracy of the simple popularity algorithm for the RecSys dataset is also noteworthy[8]. While this approach will likely not result in high user satisfaction, just by predicting the same items over and over again, we can beat all other baselines in terms of nDCG. This is also apparent in Figure 6, which presents the accuracy for predicting the next job in the session. The popularity approach actually outperforms the other algorithms until $k = 3$.

With respect to the next job prediction, in both datasets, we can see similar accuracy results for all of the utilized approaches. The *iKNN* did manage to outperform *GRU4Rec* in the smaller Studo

[8] Actually, Quadrana et al. [33] also report that their popularity approach outperforms the utilized session-based RNN [16] on the older XING dataset.

Approach	Studo					RecSys Challenge 2017				
	nDCG	Novelty	Serendipity	Coverage (%)		nDCG	Novelty	Serendipity	Coverage (%)	
sKNN$_{AE}$	.3121	.2182	.8173	46.48	100	.2317	.6840	.6309	37.13	91.33
sKNN$_{DAE}$	.3112	.2179	.8170	47.99	100	.2321	.5946	.5187	36.97	91.33
sKNN$_{VAE}$	.3135	.2190	.8142	48.49	100	.2321	.5941	.5382	36.82	91.29
sKNN	.2829	.2090	.7976	43.62	100	.2181	.7663	.8786	36.32	91.03
GRU4Rec	.1624	.3427	.9110	93.46	99.39	.0878	.7629	.9491	45.60	72.85
Bayes	.2334	.2709	.8595	47.48	100	.0446	.8207	.9463	28.56	61.62
iKNN	.2435	.2687	.8423	47.99	98.78	.0565	.7904	.9344	35.03	70.59
BPR-MF	.0671	.2284	.9186	58.39	78.66	.1324	.6831	.9127	77.08	94.59
POP	.0445	.1047	.9444	3.34	3.66	.2294	.3513	.9126	0.13	0.30

Table 2: Prediction results ($k = 20$) of remaining jobs that will be interacted within a session. Coverage is reported for the ratio of recommended jobs when compared to all jobs available in the data set (left) and jobs expected in the test set (right).

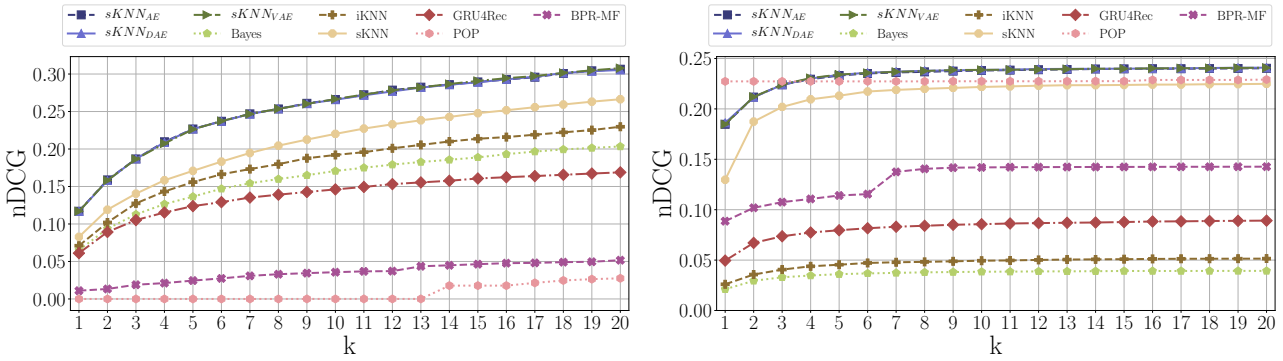
dataset but not in the one from the RecSys Challenge 2017. Beside the already mentioned popularity approach, the highest accuracy of the baselines had the *sKNN*.

Non-accuracy performance. Although the *GRU4Rec* approach did not achieve the highest accuracy, it needs to be noted that it results in novel recommendations. That is, for Studo it recommended the most towards the long tail of the job distribution and for RecSys it was pretty competitive with the *sKNN* baseline. Also to note for the RecSys dataset, *Bayes* and *iKNN* did have the highest novelty but also the lowest accuracy. With respect to our proposed method, it needs to be acknowledged that while we achieve the highest accuracy, utilizing autoencoders for session-based *KNN* does not lead to the best results with respect to novelty. Between the autoencder variations, a higher novelty was achieved using a *VAE* in the smaller Studo dataset, where a classical *AE* was performing better in the larger RecSys dataset. Still, when compared to *GRU4Rec* this yields between 1.12 and 1.57 times less novel recommendations.

With regards to serendipity, we can see that almost all of the session-based baseline approaches provide highly *surprising* or *unexpected* job recommendations. Due to the limited amount of information about the user that is available in a session-based setting, this can also be expected. But as already noted, providing such a high level *unexpectedness* might lead to a dissatisfaction and distrust with the underlying recommender system [4, 11]. If we take that a simple popularity approach (i.e., like the one we utilize as a baseline) is completely unrelated to the current interaction behavior of a given session user, the aim for a session-based recommender algorithm would be to have a serendipity that is lower and in fact not totally *surpassing* given the current browsing behaviour. Actually, the least *surprising* recommendations for a session are achieved when utilizing or adapting an *sKNN* approach. Indeed, the *sKNN* baseline has the lowest serendipity value in the smaller Studo dataset. Using our proposed method with utilizing autoencoders, we are slightly more serendipitous but still better than all the other baselines. In case of the RecSys Challenge 2017 dataset, we achieve a clear difference with respect to serendipity when utilizing the encoded session representations in a *KNN* manner. Actually, using a *DAE* or *VAE* model, we can reach almost perfect balance of how *surprising* the recommendations are.

Job coverage. Being able to provide a good coverage for a session-based recommender algorithm is especially important in the job domain, as the aim is also to help a company with the respective job posting to be detected by potential candidates. As such, in Table 2, we report the ratio of the recommendations with all jobs available in the dataset and jobs that we know (i.e., expect) will be interacted with in the test set. Looking at the coverage of all possible job postings, the best performance is achieved with the *BPR-MF* and *GRU4Rec* baselines. Apart from those two, our session-based *KNN*, which utilizes autoencoders provided the 3rd best coverage. When looking at the jobs, which we expect to be interacted with in the test set, we can see that our proposed method covers all of them in case of the Studo dataset. In case of the RecSys Challenge 2017 dataset, our proposed method with around 91.3% coverage was only topped by the *BPR-MF* baseline, which in turn performed much worse with respect to all the other evaluation metrics. By looking at the coverage, we can also see why the previously mentioned high accuracy results of the popularity baseline in the RecSys Challenge 2017 dataset does not mean that much. To cover only such a small portion of possible job postings in the system will not only lead to unsatisfied users but also lose trust from companies that actually post jobs in the system.

Generating recommendations with VAEs. We also explored to recommend jobs using the generative model described in Section 4.1. For a given target session s_t we generate job probabilities using the mean of the variational distribution z. In our session-based setting this has however resulted in a noticeably lower accuracy than the rest of the utilized approaches. When predicting the remainder of the session, the best performing model when trained for 50 epochs had an accuracy of $nDCG@20 = 0.1451$ in case of Studo and 0.1165 for the RecSys Challenge 2017 dataset. With respect to the non-accuracy metrics, for Studo we got $Novelty@20 = 0.1664$ and $Serendipity@20 = 0.8983$. For RecSys it was $Novelty@20 = 0.3950$ and $Serendipity@20 = 0.9359$. As these results are already



(a) Studo **(b) RecSys Challenge 2017**

Figure 6: Accuracy results for different recommendation list sizes when predicting the next clicked job for the Studo (left) and RecSys Challenge 2017 (right) dataset.

outperformed by the utilized baselines, this suggests that additional research needs to be done on applying the generative model of VAEs for session-based recommendation. For instance, we did not experiment on β-VAE with the adapted multinomial likelihood and Bayesian inference for parameter estimation as recently proposed by Liang et al.[26]. It could also be possible that as session vectors are much sparser than user vectors, we would need to adapt the learning method to be longer and introduce sub-sampling mechanisms to provide more data examples.

7 CONCLUSION AND FUTURE WORK

In this work, we have investigated how to utilize autoencoders on interaction data for session-based job recommendations. We have shown that our proposed method augments existing KNN techniques through a better representation of sessions and yields the best results with respect to recommendation accuracy. In addition to that, we addressed the lack of non-accuracy evaluations in session-based recommendations. As such, we evaluated all approaches on non-traditional metrics such as novelty and serendipity and also showed that session-based recommendations do not need to be highly surprising for the session user.

For future work, we plan to investigate the impact of the offline measured serendipity in the online setting. As such, we plan to conduct an online study to ask users how satisfied and surprised they were when we utilize autoencoders for recommending jobs with latent session representations. Also, we plan to further look into how to directly adapt the generative model of variational autoencoders for session-based recommendation.

One limitation of our approach is that the autoencoder model would need to be retrained when allot of input items become obsolete. Given the nature of domains other than from the job market, this may pose an undesirable computational overhead. As such, we will look into extending our approach by exploiting auxiliary information for dimensionality reduction when inferring latent session representations. For example, this could be done by utilizing solely the content of job postings or even including temporal dynamics into this (e.g., by utilizing an attention mechanism similar as the authors of [25] have done it).

In summary, we hope that future work will be attracted by our insights on how dimensionality reduction techniques can be applied for session-based job recommendation as well as their effect on non-accuracy metrics like novelty and serendipity.

REFERENCES

[1] F. Abel. We know where you should work next summer: job recommendations. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 230–230. ACM, 2015.

[2] F. Abel, A. Benczúr, D. Kohlsdorf, M. Larson, and R. Pálovics. Recsys challenge 2016: Job recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 425–426. ACM, 2016.

[3] F. Abel, Y. Deldjoo, M. Elahi, and D. Kohlsdorf. Recsys challenge 2017: Offline and online evaluation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 372–373. ACM, 2017.

[4] P. Adamopoulos and A. Tuzhilin. On unexpectedness in recommender systems: Or how to expect the unexpected. In *DiveRS@ RecSys*, pages 11–18, 2011.

[5] S. T. Al-Otaibi and M. Ykhlef. A survey of job recommender systems. *International Journal of Physical Sciences*, 7(29):5127–5142, 2012.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[7] M. Bianchi, F. Cesaro, F. Ciceri, M. Dagrada, A. Gasparin, D. Grattarola, I. Inajjar, A. M. Metelli, and L. Cella. Content-based approaches for cold-start job recommendations. In *Proceedings of the Recommender Systems Challenge 2017*, page 6. ACM, 2017.

[8] G. Bonnin and D. Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):26, 2015.

[9] S. Chatzis, P. Christodoulou, and A. S. Andreou. Recurrent latent variable networks for session-based recommendation. *arXiv preprint arXiv:1706.04026*, 2017.

[10] A. Fischer and C. Igel. An introduction to restricted boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*, pages 14–36. Springer, 2012.

[11] M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM, 2010.

[12] K. Georgiev and P. Nakov. A non-iid framework for collaborative filtering with restricted boltzmann machines. In *International Conference on Machine Learning*, pages 1148–1156, 2013.

[13] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[14] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[15] B. Hidasi and A. Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. *arXiv preprint arXiv:1706.03847*, 2017.

[16] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[17] B. Hidasi and D. Tikk. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery*, 30(2):342–371, 2016.

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

[19] D. Jannach and M. Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 306–310. ACM, 2017.

[20] J. Jones, J. Schmitt, et al. A college degree is no guarantee. Technical report, Center for Economic and Policy Research (CEPR), 2014.

[21] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[22] K. Kenthapadi, B. Le, and G. Venkataraman. Personalized job recommendation system at linkedin: Practical challenges and lessons learned. In *Proc. of ACM RecSys'17*.

[23] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[25] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428. ACM, 2017.

[26] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara. Variational autoencoders for collaborative filtering. *arXiv preprint arXiv:1802.05814*, 2018.

[27] R. Liu, W. Rong, Y. Ouyang, and Z. Xiong. A hierarchical similarity based job recommendation service framework for university students. *Frontiers of Computer Science*, 11(5):912–922, Oct 2017.

[28] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[29] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM, 2006.

[30] S. K. Mishra and M. Reddy. A bottom-up approach to job recommendation system. In *Proceedings of the Recommender Systems Challenge*, page 4. ACM, 2016.

[31] D. Parra and S. Sahebi. Recommender systems : Sources of knowledge and evaluation metrics. In *Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis*, pages 149–175. Springer, 2013.

[32] P. Pu, L. Chen, and R. Hu. A user-centric evaluation framework for recommender systems. In *Proc. of ACM RecSys'11*.

[33] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137. ACM, 2017.

[34] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[35] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[36] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[37] G. Shani, D. Heckerman, and R. I. Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005.

[38] E. Smirnova and F. Vasile. Contextual sequence modeling for recommendation with recurrent neural networks. *arXiv preprint arXiv:1706.07684*, 2017.

[39] Y. Song, A. M. Elkahky, and X. He. Multi-rate deep learning for temporal recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 909–912. ACM, 2016.

[40] Y. K. Tan, X. Xu, and Y. Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22. ACM, 2016.

[41] B. Twardowski. Modelling contextual information in session-aware recommender systems with neural networks. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 273–276. ACM, 2016.

[42] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[43] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[44] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162. ACM, 2016.

[45] C. Zhang and X. Cheng. An ensemble method for job recommender systems. In *Proceedings of the Recommender Systems Challenge*, page 2. ACM, 2016.

[46] Y. C. Zhang, D. Ó. Séaghdha, D. Quercia, and T. Jambor. Auralist: introducing serendipity into music recommendation. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 13–22. ACM, 2012.

[47] Y. Zheng, B. Tang, W. Ding, and H. Zhou. A neural autoregressive approach to collaborative filtering. *arXiv preprint arXiv:1605.09477*, 2016.

[48] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.