Evaluation of Algorithms for Interaction-Sparse Recommendations: Neural Networks don't Always Win

Yasamin Klingler
yasamin.klingler@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

Claude Lehmann
claude.lehmann@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

João Pedro Monteiro
jp@veezoo.com
Veezoo AG
Zurich, Switzerland

Carlo Saladin
carlo@veezoo.com
Veezoo AG
Zurich, Switzerland

Abraham Bernstein
bernstein@ifi.uzh.ch
University of Zurich
Zurich, Switzerland

Kurt Stockinger
kurt.stockinger@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

ABSTRACT

In recent years, top-K recommender systems with implicit feedback data gained interest in many real-world business scenarios. In particular, neural networks have shown promising results on these tasks. However, while traditional recommender systems are built on datasets with frequent user interactions, insurance recommenders often have access to a very limited amount of user interactions, as people only buy a few insurance products.

In this paper, we shed new light on the problem of top-K recommendations for interaction-sparse recommender problems. In particular, we analyze six different recommender algorithms, namely a popularity-based baseline and compare it against two matrix factorization methods (SVD++, ALS), one neural network approach (JCA) and two combinations of neural network and factorization machine approaches (DeepFM, NeuFM). We evaluate these algorithms on six different interaction-sparse datasets and one dataset with a less sparse interaction pattern to elucidate the unique behavior of interaction-sparse datasets.

In our experimental evaluation based on real-world insurance data, we demonstrate that DeepFM shows the best performance followed by JCA and SVD++, which indicates that neural network approaches are the dominant technologies. However, for the remaining five datasets we observe a different pattern. Overall, the matrix factorization method SVD++ is the winner. Surprisingly, the simple popularity-based approach comes out second followed by the neural network approach JCA. In summary, our experimental evaluation for interaction-sparse datasets demonstrates that in general *matrix factorization methods outperform* neural network approaches. As a consequence, traditional well-established methods should be part of the portfolio of algorithms to solve real-world interaction-sparse recommender problems.

1 INTRODUCTION

Recommender Systems have gained enormous popularity: many of our online purchasing interactions are accompanied by the suggestion of additional products that we could buy. But not all domains are equally amenable to existing systems. The insurance domain, for example, poses three major challenges to recommender systems.

The first challenge is the *high sparsity of the interactions per user*. In a typical recommender system scenario, such as the movie domain, users have access to thousands of movies and many users may watch hundreds of movies over time. However, in an insurance setting most users buy only one or two products. In contrast to typical movie recommenders, where users with few interactions are removed from the input data [37], the data sparsity in the insurance domain forces us to learn from as little information as available and provide recommendations for users with few interactions because they form the majority of our users.

The second challenge that differentiates the insurance domain from other domains, where recommender systems are typically used, is an extreme *popularity bias* in the data. On the one hand, there are a few products bought by almost all users. E.g., every car owner has car insurance. On the other hand, there are many products only bought by very few users. This skews the data extremely toward more popular products.

Lastly, *the lack of prior knowledge about life events* makes the recommendation task more challenging. Customers' typically purchase an insurance product mainly due to a life event such as getting married, birth of a child, or moving to a new apartment. This information is not available to the agents unless the customer actively requests the product or there is a direct meeting with the customer.

These challenges make the recommendation problem harder than in traditional settings and may require different approaches or recommender system designs compared to traditional domains. Given that to the best of our knowledge they have not been widely studied, this paper addresses the following questions:

- *Which recommender system performs the best for insurance product recommendations?* Traditional recommender systems are typically evaluated against datasets with many users and many products. In this setting, autoencoders are considered as the state of the art. However, in settings with sparse-interaction users and implicit feedback data, are autoencoders still performing the best?
- *How does the revenue change for different methods?* When recommending insurance products, it is not only important to recommend relevant products with high precision but also to generate revenue from the recommended products. Which algorithm performs best in this scenario? Does optimizing for more relevant products result in a higher revenue?

In answering these questions, this paper makes the following contributions:

- We shed new light on the problem of *recommending top-K insurance products*. These kind of recommendations have to rely on implicit feedback with high skewness toward popular products and with sparse user interactions.
- We show that in the setting of insurance data, *DeepFM outperforms the existing state of the art method, joint collaborative autoencoder (JCA)* [39], followed by the matrix factorized method SVD++ [16].
- We present a *new version of the MovieLens 1M dataset*, generated by keeping only less than 6 interactions per user. Moreover, we enrich this dataset by adding movie prices extracted from a public API. Consequently, this dataset shows similar characteristics to our real-world insurance dataset.
- We show that for other interaction-sparse datasets *matrix factorization methods outperform* neural network approaches. Hence, the choice of the best recommender algorithm highly depends on the characteristics of the data as well as on the interaction pattern of the users with the data.
- In summary, *neural networks do not always win* and well-established traditional methods such as matrix-factorization approaches should be taken into account when selecting a mix of promising algorithms for recommender problems.

The rest of this paper is structured as follows. Section 2 describes existing approaches for implicit feedback product recommendation and insurance recommendation. In Section 3, we describe the insurance recommendation setting in more detail. In Section 4, we introduce the five approaches that we use for our evaluation. Section 5.1 presents the datasets and our experimental results (Section 6). Finally, we conclude and present future work in Section 7.

2 RELATED WORK

Early recommendation methods have mainly focused on problems where users provide *explicit feedback*. Explicit feedback is available in many popular datasets such as Netflix prize[1] or MovieLens[2] [11], where ratings on a scale of 1 to 5 are available. These ratings are represented as user-item pairs followed by the corresponding rating. There is a considerable amount of literature for rating prediction tasks on the MovieLens 10M benchmark [3–5, 8, 14, 18–20, 30, 41] and the Netflix prize. Additionally, Rendle [27] proposed feature-based factorization machines to extend the rating data with contextual information such as time or latest movie rated to improve the predictions. Among neural network-based models, AutoRec [30] is an autoencoder-based model that outperformed non-neural methods on the rating prediction task.

While previous work has extensively studied explicit feedback, the importance of *implicit feedback* has gradually increased. This is mainly due to the rise of interest in market place recommender systems. Implicit feedback is automatically collected from users while they interact with marketplaces and is the result of a user's behavior e.g. purchase history or clicks. Leveraging implicit feedback—which is also referred to as One-class [7, 22, 33] or positive-only feedback [6, 32]—is very relevant in real-world scenarios such as e-commerce systems or insurance platforms. Early work on recommender systems with implicit feedback uses a Factorization Machine (FM) with Bayesian Personalized Ranking (BPR) [28]. BPR uses the positive instances in the data (i.e., purchased) and samples negative instances from missing data (i.e., not purchased).

More recent developments of top-K recommender systems increasingly use neural network architectures. The two most commonly used methods are autoencoder and multi-layer perceptron architectures. Collaborative Denoising Autoencoder (CDAE) [36] is a neural-network-based collaborative filtering method. Zhu et al. [39] extended CDAE as joint collaborative autoencoder to simultaneously learn from a user-centric view and an item-centric view of the data. Standard matrix factorization uses the dot product to combine latent user and item features to estimate the user interactions. Neural Collaborative Filtering (NCF) [13] generalizes matrix factorization by replacing the dot product by a trainable multi-layer perceptron (MLP).

Several researchers have addressed insurance recommender systems. Preliminary work by Qazi et al. [24] used Bayesian networks, which were further extended in [25]. More recent studies [2, 17] tackled low frequency interactions in the insurance domain. Bi et al. [2] utilized cross-domain mechanisms to deal with cold start users in the insurance domain. Lesange et al. [17] used XGBoost to build an up-selling system for car insurances. However, none of these approaches present an in-depth analysis of the private insurance data in comparison to other public datasets.

3 CASE STUDY: INSURANCE RECOMMENDATION

While many of the characteristics of e-commerce and multimedia recommendation datasets (such as the MovieLens dataset) are similar to insurance recommendation dataset, there exist two main differences: a *lack of customer analytics* for traditional insurance companies and the *low number of direct interactions* between customers (users) and an insurance company's products (items). In e-commerce and multimedia, it is not unusual to have tens, hundreds, and sometimes even thousands of interactions through the browsing behavior of customers. For traditional insurance companies, interactions basically only exist when a new policy is sold, adjusted, or canceled.

These *points of contact are rare* and (predominantly for private customers) occur at key moments in their life: Moving the apartment, getting married, or when having children. This changes market structures to a setting that economists call "viscous demand" [26]. Business customers, on the other hand, typically own more policies than private customers, as a large group of people or companies have to be covered for a large number of eventualities. Hence, it is harder to pinpoint the particular key moment when to offer certain insurance policies for business customers.

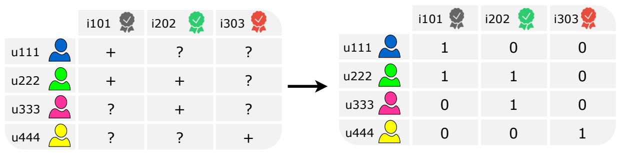
Furthermore, information about the customers is not only rare, but generally *out of date*. While it is relatively easy to track users on an e-commerce platform and gather information through linking accounts with various social media platforms or through analyzing their browsing behavior, it is much harder in the case of insurances. To address this gap, insurance companies are trying to move their customers to their own dedicated mobile applications, aiming to increase engagement and interact on a more regular basis.

[1] https://www.netflixprize.com/
[2] https://grouplens.org/datasets/movielens/

Additionally, insurance sales is of course more complex than just recommending products. An opportunity for selling insurances is the up-selling of an already existing policy. By offering additional coverages, or a more fitting policy entirely, insurance companies are able to increase customer retention and profit from free word-of-mouth marketing. This poses another interesting problem for recommender systems, where the recommendation is more structured with each product consisting of multiple coverages. However, a detailed discussion about up-selling is out of scope for this paper.

3.1 Data Characteristics

In insurance recommender systems, previous purchases of customers are the implicit feedback that reflects the customers' preferences. With this implicit data, we can assume that the product bought by the customer can be interpreted as positive feedback or an interest.



	i101	i202	i303
u111	+	?	?
u222	+	+	?
u333	?	+	?
u444	?	?	+

$\longrightarrow$

	i101	i202	i303
u111	1	0	0
u222	1	1	0
u333	0	1	0
u444	0	0	1

Figure 1: Implicit ratings in an insurance dataset where *u* refers to users and *i* to items.

Figure 1 represents the user-item matrix of a typical insurance dataset. The left side shows the purchases made by users, indicated by a plus sign. All other user-item pairs are marked with question marks. Question marks indicate either a missing or a negative rating. I.e., we have two different categories of missing information. The right side shows the user-item matrix of the same data where we assign the value 0 for both categories of missing information. If an item was bought, we assign the value 1. Because users in general only interact with a small subset of all available items, this user-item matrix is very sparse.

Another characteristic of insurance datasets is their dimensionality. While the number of users is typically large, i.e. several hundred thousands or millions of users, the number of products is typically relatively small, i.e., a few hundreds. Common benchmark datasets for recommendation tasks, in contrast, have thousands of products. Moreover, a large number of users typically buy only one or two products such as household or car insurance. Only a small number of users buy ten or more products. This creates a very sparse dataset, which is furthermore *very strongly dominated by the most popular products*, while the majority of products are in the long tail of rarely bought products. This is even more the case than in typical long-tail distributions. A detailed comparison of the insurance dataset with all other datasets used for our evaluation is given in Section 5.1.

Taking these data characteristics into account is important for the design of recommender systems. In particular, the designer of the recommender system should be cautious about a popularity bias in the system. Although recommending the most popular products may already achieve a reasonable result in the insurance recommendation setting, we expect our model to learn the long tail products as well, and recommend them where necessary.

3.2 Recommender System Design Goals

Our aim is to design a supporting system for sales representatives of an insurance company. This allows the representative to query potential products for a specific customer. While the recommender system does not have access to information discussed in a sales call, the representative can use his or her expertise to further select the most relevant products among the presented recommendations. On the one hand, this process makes the recommendation task more challenging, because we cannot get any immediate feedback from the end user of the system, in this case the customer. On the other hand, it is beneficial because the recommendations are evaluated by domain experts before they are conveyed to the customers. It is important to note that the goal is not to replace sales representatives, but to supplement their work with objective recommendations.

4 METHODOLOGY

In this section, we present five main approaches for the implicit top-K recommendation setting that we later use for our evaluation of interaction-sparse recommendations.

First, we present a simple popularity-based baseline for comparison. Second, we describe SVD++ as a matrix factorization method, which performs an optimized version of the Singular Value Decomposition (SVD). Third, we follow SVD++ with another matrix factorization method, Alternating Least Squares (ALS), a fast and efficient matrix factorization method. Fourth and fifth, we describe two methods — DeepFM and NCF — that combine matrix factorization with neural networks. Finally, we discuss Joint Collaborative Autoencoder (JCA), which is considered as the state-of-the-art recommender algorithm for implicit datasets [40].

We formally present N users as $U = \{u_1, ..., u_N\}$ and M items as $I = \{i_1, ..., i_M\}$. We refer to the one-hot encoded user features of a user U as UF and the one-hot encoded item features of the item I as IF. In our case the actual users of the recommender system are the sales representatives of the insurance company, while the recommender system's data describes the insured private or corporate customers. We refer to *user* and *customer* interchangeably throughout this paper and explicitly refer to the *sales representative* when needed.

The interaction between users and items is defined by a purchase history, which we define as the set $S \subseteq U \times I$. This set can be encoded as a matrix in $R^{N \times M}$ where an element s_{nm} is 1 iff $(u_n, i_m) \in S$ (i.e., user u_n purchased item i_m) and is otherwise 0. This means the cases where no rating is available or the user is not interested in that particular item are both encoded in the same way and thus the system is unable to differentiate between the two.

4.1 Popularity-Based Baseline

To compare the performance of our algorithms against a simple baseline, we use the *popularity-based* recommendation as a baseline. This non-personalized approach recommends the most popular items to every user under the condition that the user does not already have the product. We define the popularity of any given product by the number of occurrences in the purchase or rating history of the given dataset.

4.2 Singular Value Decomposition

The singular value decomposition (SVD) is a very useful transformation in linear algebra to dissect a matrix into a product of three matrices, each with useful mathematical properties. The SVD [31] (and also the extension SVD++ [16]) methods make use of this idea of decomposing the user-item matrix into two matrices that describe the factors influencing just the users and just items, respectively. SVD is generally used in explicit datasets, whereas SVD++ also incorporates the implicit feedback.

$$\widehat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) \tag{1}$$

Equation 1 shows how recommendations are generated. $\widehat{r}_{ui}$ is the predicted rating for user u and item i. It is calculated by adding the baseline estimate b_{ui} (a sum of the mean rating, the user bias and item bias) to the implicit factor inside the large parentheses multiplied by q_i^T, the item factor for item i. p_u is a vector of user factors of the explicit ratings, while the sum $|N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$ represents the implicit ratings. $N(u)$ is the set of all items with implicit feedback from user u. y_i is another item factor for item i. When using purely implicit feedback, negative sampling should be used for the explicit aspects of SVD++ to function. For a detailed description on SVDs, we refer the interested reader to [9].

4.3 Alternating Least Squares

Alternating Least Squares (ALS) [12] follows the same matrix factorization route as SVD++. The user-item matrix R is decomposed into two matrices U and V, such that $R \approx U^T V$. Individual rating predictions r_{ui} are generated by multiplying the user's component vector of U with

$$\min_{U,V} \sum_{\{i,j | r_{i,j} \neq 0\}} (r_{i,j} - u_i^T v_j)^2 + \lambda \left(\sum_i n_{u_i} ||u_i||^2 + \sum_j n_{v_j} ||v_j||^2 \right) \tag{2}$$

Equation 2 shows the function to be minimized in generating U and V, taking the difference between the known rating $r_{i,j}$ and the predicted rating $u_i^T v_j$. The regularization term is parametrized by λ. n_{u_i} denotes the number of items user i interacted with and n_{v_j} is the number of users that have interacted with item j.

While gradient descent could be used to solve the Equation 2, research has shown that it is inefficient and slow. Instead, the matrices U and V are optimized in an alternating process, hence the name. In each update step, one matrix is fixed at its current value and the other is optimized, then the roles are reversed. This has been shown to transform the objective into a convex function.

4.4 DeepFM

DeepFM [10] is a factorization machine architecture, originally introduced for click-through rate prediction. The goal of DeepFM is to bring together the successful architectures of factorization machines with deep neural networks. Figure 2 shows an overview of the DeepFM architecture. The factorization machine (FM) component on the left half of the figure represents a typical factorization machine. The deep component on the right half uses a feed-forward architecture to train higher order feature interactions. A prediction is calculated by applying the sigmoid function to the sum of both components' output.

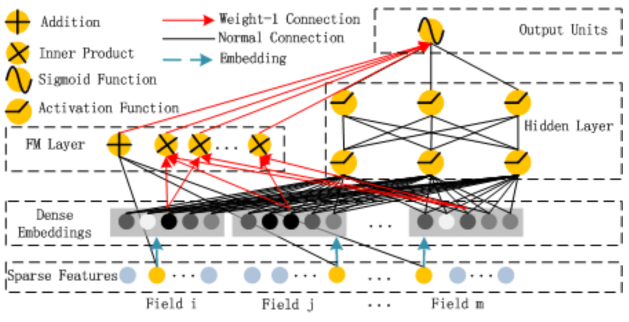


Figure 2: Illustration of the DeepFM architecture.

4.5 Neural Collaborative Filtering

Neural Collaborative Filtering (NCF) [13] is a framework, which models interactions in multiple layers. The feature vectors of both users and items are given to an embedding layer, which turns the one-hot encoded identification vectors into the latent user and item vectors. These latent vectors are passed through multiple layers where latent structures in the interactions are to be learned.

$$\widehat{r}_{ui} = f(u, v) \tag{3}$$

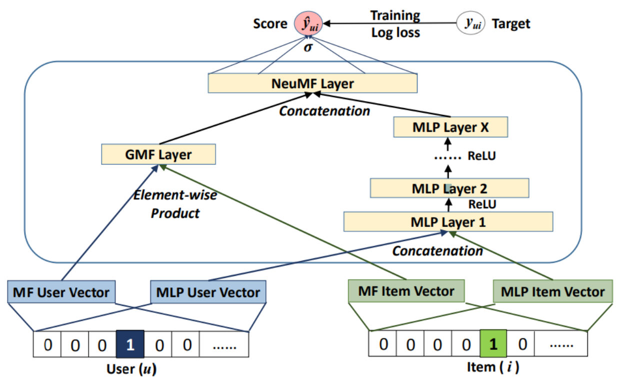


Figure 3: Illustration of the neural matrix factorization architecture (NeuMF) as an implementation of NCF.

In classical matrix factorization, a prediction $\widehat{r}_{ui}$ is generated as shown in Equation 3, where f represents the dot product and the latent user and item vectors are u and v, respectively. The authors show one implementation of NCF (termed as generalized matrix factorization, GMF) that, when using a linear kernel, their NCF framework can be understood as a special case of matrix factorization. Another implementation uses the nonlinearity of a multi-layer perceptron (MLP) as a general function approximator to learn the similarity function f instead of applying the dot product. Finally, they introduce a fusion of both GMF and MLP implementations of their framework as neural matrix factorization (NeuMF). Figure 3 shows the GMF component on the left and the MLP component on the right side. Unlike in DeepFM, both components learn their individual embedding vectors for flexibility and act independently of each other. Only in the final NeuMF layer are the components concatenated to produce the predicted rating. For our experiments, we will use the NeuMF implementation.

4.6 Joint Collaborative Autoencoder

4.6.1 Base model. An autoencoder (AE) is a class of unsupervised neural networks that attempts to reconstruct the input data in the output layer while learning a lower-dimensional feature representation of the unlabeled data. The autoencoder architecture that we describe in this section is called *Joint Collaborative Autoencoder* (JCA) that provides top-K recommendation [39].

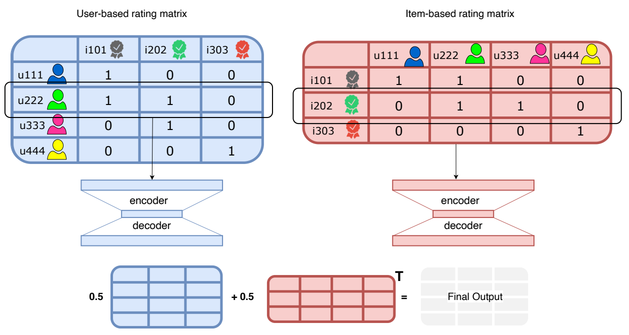


Figure 4: Illustration of a joint collaborative autoencoder. Left: User-based rating matrix. Right: Item-based rating matrix. Finally, user-based and item-based matrices are combined for calculating recommendations.

The architecture of JCA consists of two encoder-decoder neural networks (see Figure 4) with one hidden layer of 160 neurons in both encoder-decoder networks (the same configuration as used by the original authors). The neural network on the left side of Figure 4 receives the *user-based* rating matrix R as input, where the rows of the matrix correspond to the users and the columns correspond to the items. The neural network on the right side receives the *item-based* rating matrix R^T as input. This means that the former neural network will handle each user separately, while the latter will handle each item separately. In order to calculate the rating matrix that is used for recommendations, the output of both neural networks is averaged.

The JCA model is formulated as follows:

$$\hat{R} = \frac{1}{2}[\sigma(\sigma(RV^U + b_1^U)W^U + b_2^U) + \sigma(\sigma((R^T V^I + b_1^I)W^I + b_2^I)^T)] \tag{4}$$

where $\hat{R}$ is the predicted rating matrix and R is the input rating matrix. V^U and V^I are the model weights of the user and item component in the first layer. W^U and W^I are the model weights of the user and item component in the second layer. Moreover, b_1^U and b_2^U are the biases of the user component in the first and second layers, respectively. b_1^I and b_2^I are the biases of the item components in the first and second layers. Finally, σ refers to the sigmoid activation function.

The model minimizes a pairwise *hinge loss* defined in Equation 5.

$$\mathcal{L}(\theta) = \sum_{u \in U} \sum_{\substack{i \in S_u^+ \\ i' \in S_u^-}} max(0, \hat{r}_{u,i'} - \hat{r}_{u,i} + d) + \frac{\lambda}{2}||\theta||_F^2 \tag{5}$$

Here, $\hat{r}_{u,i}$ refers to the predicted ratings for user u and item i. The item i ranges over items with a positive feedback, $S_u^+ = \{i \in I \mid (u,i) \in S\}$, while the item i' ranges over items without positive feedback, $S_u^- = \{i \in I \mid (u,i) \notin S\}$. λ and θ present the L2-norm regularization term and all the model parameters, respectively. Finally, d is the allowed margin between positive and negative samples and $||.||_F$ is the Frobenius norm.

5 EXPERIMENTS

In this section we describe the datasets and our experimental setup. Note that to enable the reproducibility of our experiments, our code is available on Github[3]. All datasets with the exception of the proprietary insurance dataset are publicly available.

5.1 Datasets

For our experiments we use a private insurance dataset, the publicly available MovieLens dataset, the Youchoose dataset from the RecSys Challenge 2015, and the publicly available Retailrocket dataset. Their characteristics are as follows:

Insurance dataset. The core dataset of this paper is our insurance dataset (see Table 1), whose challenges we discussed previously in Section 3. The dataset contains several hundred thousand users, a few hundred items, and about 1 million user interactions resulting in a density below 1% (i.e. less than 1% of possible interactions appear in the dataset). A unique characteristic of the insurance dataset is the number of interactions per user and per item. The number of interactions per user is never more than 20, and users on average only interact with 1 to 3 items, with most users having only a single item. Also, the number of interactions per item ranges between a handful to a few hundred thousand. I.e., the most popular item is bought by a few hundred thousand users while the least popular items are bought by only a handful of users.

The user data in this dataset contains the policies bought by the users as well as the users' demographic features. These demographic features are age range, gender, marital status, whether the user is a company or a private customer, and the industry in which the user is active. For privacy reasons, we cannot give more details about the dataset.

MovieLens1M. We use the popular MovieLens1M dataset [11]. We choose this dataset over other versions of the MovieLens dataset and E-commerce datasets because the number of interactions and the number of products are more similar to our insurance dataset. Since the original version of the dataset does not contain any price value for the movies, we used a public API to enrich the dataset with movie prices. The enriched movie prices range from 2$ to 20$ and are approximately normally distributed around the 10$.

Since we want to solve an *implicit feedback* task, we consider the data points with ratings ≥ 4 as positive feedback. Data points with a rating below 4 are discarded. Hence, we transform the dataset into one with implicit feedback, making it impossible to differentiate between negative feedback and missing feedback – which is also an important characteristic of the insurance dataset. Moreover, this approach is also used by the authors in [15, 38, 39].

Recent related work focused on recommending items that have been rated by at least 6 different users, discarding all users who rated less than 6 items [39]. However, in order to reconstruct the insurance setting, where people buy typically few insurance policies, with the MovieLens dataset, we analyze *two different subsets* of the MovieLens datasets. *MovieLens1M-Min6* contains users that have rated at least 6 movies and movies that were rated by at least 6 users – as done in previous work. It contains

[3]https://github.com/edualc/InsuranceRecommender

Table 1: General statistics of the different datasets. Skewness is measured as the Fisher-Pearson coefficient.

Dataset	User Selection	# Users	# Items	# Interactions	Density [%]	Skewness	User/Item Ratio
Insurance	All	100k-1,000k	100-1,000	~1,000,000	<1.00	~10.00	~1000 : 1
MovieLens1M-Max5	≤ 5 interactions	11	38	39	10.26	5.91	0,29 : 1
MovieLens1M-Max5-New	≤ 5 interactions (newest)	6,038	2,771	30,179	0.18	3.61	2,18 : 1
MovieLens1M-Max5-Old	≤ 5 interactions (oldest)	6,038	2,493	30,179	0.20	9.92	2,42 : 1
MovieLens1M-Min6	≥ 6 interactions	6,027	3,062	574,026	3.11	3.65	1,97 : 1
Retailrocket	All	11,719	12,025	21,270	0.02	19.97	0,97 : 1
Yoochoose	All	509,696	19,949	1,049,817	0.01	17.75	25,55 : 1
Yoochoose-Small	5% of all interactions	49,670	7,369	52,491	0.01	11.96	6,74 : 1

Table 2: Interaction statistics for the different datasets.

		Interactions p. User			Interactions p. Item			Cold Start (10-fold CV)	
Dataset	User Selection	Min	Avg	Max	Min	Avg	Max	Users [%]	Items [%]
Insurance	All	1	1-3	<20	1-10	1-10k	100k-1,000k	~50.00	<1.00
MovieLens1M-Max5-Old	≤ 5 interactions (oldest)	1	4.99	5	1	12.11	831	0.00	5.28
MovieLens1M-Min6	≥ 6 interactions	6	95.24	1,415	6	187.47	2,853	0.00	0.00
Retailrocket	All	1	1.82	532	1	1.77	129	61.94	45.58
Yoochoose	All	1	2.06	53	1	52.63	12,440	28.91	5.09
Yoochoose-Small	5% of all interactions	1	1.06	5	1	7.14	635	90.42	12.89

6,027 users and 3,062 items with a total of 574,026 interactions resulting in a density of 3.11. As presented in Table 2, one user interacts on average with 95 items and the maximum number of rated movies by one user is 1,415.

The majority of users in the MovieLens dataset consume 6 or more items and there is only a very small number of users with 5 or fewer items – 11 users to be precise. Specifically, the *MovieLens1M-Max5* dataset contains users that have rated at most 5 movies (see Table 2) resulting in only 39 interactions. As this is not a sufficient amount of data to sensibly train a model. Hence, we instead selected *up to* 5 interactions from *all* users, choosing either the newest or oldest 5 interactions. These two variants of the MovieLens1M-Max5 dataset are shown with the suffixes `-New` (as *MovieLens1M-Max5-New*) and `-Old` (as *MovieLens1M-Max5-Old*), respectively.

For our experiments, we selected the MovieLens1M-Max5-Old and MovieLens1M-Min6. We believe MovieLens1M-Max5-Old reflects the interaction-sparse feature of the insurance dataset quite well.

All MovieLens-derived datasets have user features such as age range, gender, and occupation.

Retailrocket. We also use Retailrocket [29], another publicly available e-commerce dataset. It contains many different types of interactions, named as *view*, *addtocart* and *transaction*. We chose to only include the *transaction* interaction, as these signals represent a stronger interest than viewing an item. Furthermore, the transactions more closely resemble our real-world insurance dataset. Tables 1 and 2 show the detailed characteristics of the dataset. Retailrocket contains the fewest number of interactions across all datasets and is the most skewed. While users have an average of 1.82 interactions, the most active user has 532 interactions (2.5% of the whole dataset). Furthermore, the total number of users and items is roughly equal, at 11,719 and 12,025, respectively.

Yoochoose. This dataset was used in the RecSys Challenge 2015 [1] and contains implicit data about sessions in an e-commerce webshop. Data points are collected for individual clicks, as well as purchases. Unlike other datasets, interactions are grouped by session and not by user. Users will have multiple sessions over time, but only the session ids are available for identification. No additional user or item features are present.

As Table 2 shows, the Yoochoose dataset contains by far the largest number of items, is very sparsely populated and highly skewed. With an average of 2.06 interactions per user and 19,949 items to choose from, it is very challenging to generate predictions. Moreover, the total number of users strongly dominates the number of items at a ratio of 25,55:1 with over half a million of users. Note that this dataset does not contain any demographic features associated with sessions.

Yoochoose-Small. Given the disproportionally large number of interactions of the Youchoose dataset compared to the MovieLens and Retailrocket, we also ran the experiments on a smaller subset containing only 5% of the interactions. It was derived by randomly subsampling the interactions. As a result the number of Youchoose interactions are comparable with the insurance and the Youchoose-small interactions with the other datasets.

Comparison of the insurance dataset and MovieLens1M. Figure 5 shows the distribution of item interactions both for the insurance dataset as well as for the full MovieLens1M dataset. We can observe that the distribution of the insurance dataset is significantly more skewed than the distribution of the MovieLens1M dataset. When calculating the Fisher-Pearson coefficient [23, 42] of skewness, the MovieLens1M dataset has a coefficient of 3.65, while the insurance dataset's coefficient is three times its size, namely 10.03 (see Table 1 for the coefficients of all datasets). This coefficient measures the difference between a given distribution and the normal distribution. In a normally distributed dataset, this coefficient is zero.

While we are unable to give concrete numbers for how often each product was bought for the insurance dataset due to privacy concerns, we can see that the majority of users in the insurance dataset have bought a small number of very popular products. On the other hand, a large number of insurance products are located in the long tail and thus are bought by only a few users. We also note that the skewness of the MovieLens1M-Max5-Old dataset is very similar to the skewness of the insurance datasets. Hence, we use the MovieLens1M-Max5-Old dataset in our experiments to study interaction-sparse recommendations.

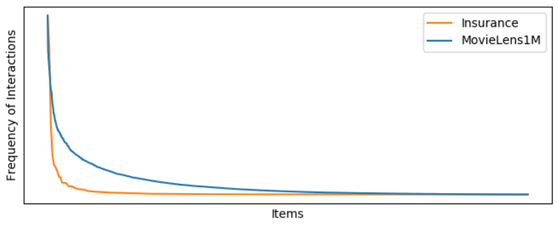


Figure 5: Distribution of item interactions for the insurance and MovieLens1M datasets. Axis scale information is removed for privacy reasons.

5.2 Training and Test Data Generation

The input for our algorithms is based on the user-item-matrix generated from all the data points in the respective dataset, where each row represents the sales history of a user and each column represents the users that bought a given product. We use 10% of our data as the test set for evaluation, whereas the remaining 90% of data is used to train the different algorithms used in this paper. The train and test datasets are generated over a 10-fold cross validation for each of the datasets.

5.3 Evaluation

5.3.1 Evaluation Metrics. We use three ranking-based metrics for evaluating our algorithms: F1@K, Normalized Discounted Cumulative Gain or NDCG@K [34], and Revenue@K. Specifically, we first take the top-K recommendations as well as the top-K ground truth values for each individual user. Next, we calculate the *metrics*@K for each individual user, where *metrics* is one of the evaluation scores and @K indicates that it is calculated with the top-K results. Finally, we average the metrics among the users.

The *Discounted Cumulative Gain at K* (DCG@K) is defined as:

$$DCG@K = \sum_{k=1}^{K} \frac{2^{\mathbb{I}[r(k) \in I_{GT}]} - 1}{\log_2(k+1)} \tag{6}$$

where $r(k)$ is the recommended product at rank k, I_{GT} is the item in the ground truth and $\mathbb{I}[.]$ is the indicator function.

We can now define *Normalized Discounted Cumulative Gain at K* (NDCG@K) as:

$$NDCG@K = \frac{DCG}{IDCG} \tag{7}$$

where the *DCG* is calculated for the recommended products and *IDCG* is the ideal DCG, calculated from the ground truth.

Finally, the *Revenue at K* is defined as:

$$Revenue@K = \sum_{n \in N} \sum_{\substack{r \in recom@K(n) \wedge \\ r \in GroundTruth(n)}} Price(r) \tag{8}$$

with *Price* referring to the price of the product at rank k.

5.3.2 Hyper-Parameter Optimization. For each of the methods, we tuned the hyper-parameters using a subset of the training data. We applied the algorithms for 20 iterations to find a suitable set of parameters, optimizing for the NDCG@1. Finally, we applied the selected parameters to each algorithm and reported the results after a fixed number of iterations suitable for each method and dataset.

The *popularity baseline* is without hyperparameters. For the remaining methods, we performed a grid search for various parameters such as batch size, learning rate and regularization parameters. The default parameters used can be taken from the Github repository[4]. Deviations from the default parameters are listed below. Both SVD++ and ALS use 256 factors on the insurance dataset and both Yoochoose and Yoochoose-Small, 64 on Retailrocket and 16 on the MovieLens datasets. In addition, SVD++ uses 0.001 as its regularization parameter for all datasets. DeepFM's embedding sizes are 32 for Insurance, Yoochoose and Yoochoose-Small, 16 for Retailrocket and 8 for both MovieLens datasets. The learning rate was set to $1e-4$ for Yoochoose and Yoochoose-Small, $3e-4$ otherwise. NeuMF's embedding sizes are 256 for Yoochoose, 64 for Retailrocket and 16 for all other datasets. The learning rates used for JCA experiments are $5e-5$ for the insurance dataset, $1e-2$ for MovieLens1M-Min6, $1e-3$ for both MoviveLens1M-Max5-Old and Retailrocket and finally, $1e-4$ for Yoochoose-small dataset. We used $1e-3$ as regularization parameter and 160 neurons in the hidden layer. Lastly, the batch size for both Movielens datasets, as well as Yoochoose-small is $8,192$, $1,500$ for the insurance dataset, and the sizes of the full dataset for Retailrocket.

5.3.3 Statistical Significance Test. After evaluating our results on the explained metrics for 10 fold cross-validation, we used the non-parametric *Wilcoxon Signed Rank Test* [35] to verify if the differences between the results are statistically significant. We perform the tests for each of the reported metrics for all the algorithms and datasets used in this paper.

5.3.4 Deployment in Veezoo. The experiments in this paper were conducted in isolation, that means outside of Veezoo's code base. However, these models have been deployed in Veezoo's platform[5] since 2020. Sales representatives of insurance companies are able to ask the system and generate recommendations. In combination with other machine learning approaches, Veezoo was able to measure a 20% increase in premium and a 94% decrease in time-to-answer for their insurance customers.

6 RESULTS AND DISCUSSIONS

In this section, we analyze the performance of the popularity-baseline method, SVD++, ALS, DeepFM, NeuMF and JCA for top-K recommendations on implicit feedback. For all algorithms we report F1@K, NDCG@K, and Revenue@K. We evaluate the methods initially on the *insurance dataset*. Afterwards we present the results on the *MovieLens1M-Min6* dataset, as previously done by Zhu et al. [39]. Next, we generalize the characteristics of the insurance dataset by using the *MovieLens1M-Max5-Old* dataset. For additional insights, we further evaluate all algorithms on the *Retailrocket*, *Yoochoose-Small* and *Yoochoose* datasets.

[4]https://github.com/edualc/InsuranceRecommender
[5]https://www.veezoo.com/insurance/

Table 3: Performance of recommender methods on *insurance dataset* (average of 10 runs).

	@1			@2			@3			@4			@5		
Method	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue
Popularity	*0.3233	$^{\times}$0.3306	$^{\bullet}26.05M$	$^{\bullet}$0.3300	$^{\bullet}$0.4242	$^{\bullet}52.36M$	$^{\bullet}$0.2722	$^{\bullet}$0.4468	$^{\bullet}57.75M$	$^{\bullet}$0.2432	$^{\bullet}$0.4733	$^{\bullet}64.64M$	$^{\bullet}$0.2170	$^{\bullet}$0.4892	$^{\bullet}69.09M$
SVD++	$^{\times}$0.3329	$^{\times}$0.3405	$^{\times}30.22M$	$^{\bullet}$0.3362	$^{+}$0.4334	**56.61M**	$^{\bullet}$0.2822	$^{\bullet}$0.4609	$^{\bullet}66.54M$	$^{\bullet}$0.2413	$^{\bullet}$0.4764	$^{\bullet}71.01M$	$^{\bullet}$0.2166	$^{\bullet}$0.4936	$^{\bullet}79.84M$
ALS	$^{\bullet}$0.1880	$^{\bullet}$0.1924	$^{\bullet}23.33M$	$^{\bullet}$0.1647	$^{\bullet}$0.2210	$^{\bullet}32.52M$	$^{\bullet}$0.1324	$^{\bullet}$0.2288	$^{\bullet}35.50M$	$^{\bullet}$0.1115	$^{\bullet}$0.2342	$^{\bullet}37.92M$	$^{\bullet}$0.0965	$^{\bullet}$0.2380	$^{\bullet}41.16M$
DeepFM	**0.3370**	**0.3438**	**30.95M**	**0.3500**	**0.4482**	$^{\times}56.05M$	**0.2953**	**0.4788**	$^{+}64.30M$	**0.2603**	**0.5035**	$^{\bullet}71.90M$	**0.2352**	**0.5240**	$^{\bullet}78.20M$
NeuMF	$^{\bullet}$0.2995	$^{\bullet}$0.3067	$^{+}26.61M$	$^{\bullet}$0.3147	$^{\bullet}$0.4011	$^{\bullet}52.03M$	$^{\bullet}$0.2707	$^{\bullet}$0.4337	$^{\bullet}62.96M$	$^{\bullet}$0.2420	$^{\bullet}$0.4601	$^{\bullet}73.28M$	$^{\bullet}$0.2164	$^{\bullet}$0.4765	$^{\bullet}80.56M$
JCA	$^{\times}$0.3246	$^{\times}$0.3320	$^{\bullet}26.61M$	$^{\bullet}$0.3346	$^{\times}$0.4289	$^{+}55.60M$	$^{\bullet}$0.2826	$^{\bullet}$0.4581	**69.59M**	$^{\times}$0.2543	$^{+}$0.4873	**81.81M**	$^{\bullet}$0.2214	$^{\bullet}$0.4976	**86.79M**

The results of the best performing method for every metric (column) is highlighted with **bold** font. We compare the results statistically with the best performing using the Wilcoxian Signed Rank test to ascertain that the difference is statistically significant, where * indicates significance at $p<0.1$, $^{+}$ at $p<0.05$, and $^{\bullet}$ at $p<0.01$ as well as $\times$ indicates not significant.

Table 4: Performance of recommender methods on *MovieLens1M-Max5-Old* with users having ≤ 5 products chosen from their oldest ratings.

	@1			@2			@3			@4			@5		
Method	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue
Popularity	**0.0324**	**0.0359**	$^{\bullet}$**1,623**	**0.0400**	**0.0469**	$^{\bullet}$**2,232**	**0.0415**	**0.0567**	$^{\times}$**3,111**	**0.0428**	**0.0656**	$^{\times}$3, 550	**0.0417**	**0.0718**	$^{\bullet}$4, 121
SVD++	$^{\times}$0.0305	$^{\times}$0.0337	$^{\bullet}$1, 243	$^{\times}$0.0383	$^{\times}$0.0448	$^{\bullet}$2, 161	$^{\times}$0.0408	$^{\times}$0.0551	$^{\bullet}$2, 938	$^{\times}$0.0417	$^{\times}$0.0637	$^{\times}$**3,653**	$^{\times}$0.0420	$^{\times}$0.0710	$^{\times}$**4,293**
ALS	$^{\bullet}$0.0224	$^{\bullet}$0.0248	$^{\bullet}$580	$^{\bullet}$0.0248	$^{\bullet}$0.0300	$^{\bullet}$970	$^{\bullet}$0.0256	$^{\bullet}$0.0359	$^{\bullet}$1, 331	$^{\bullet}$0.0255	$^{\bullet}$0.0406	$^{\bullet}$1, 674	$^{\bullet}$0.0255	$^{\bullet}$0.0449	$^{\bullet}$2, 031
DeepFM	$^{\bullet}$0.0202	$^{\bullet}$0.0243	$^{\bullet}$840	$^{\bullet}$0.0256	$^{\bullet}$0.0284	$^{\bullet}$1, 541	$^{\bullet}$0.0269	$^{\bullet}$0.0359	$^{\bullet}$2, 078	$^{\bullet}$0.0266	$^{\bullet}$0.0425	$^{\bullet}$2, 595	$^{\bullet}$0.0260	$^{\bullet}$0.0476	$^{\bullet}$3, 019
NeuMF	$^{\bullet}$0.0171	$^{\bullet}$0.0202	$^{\bullet}$531	$^{\bullet}$0.0213	$^{\bullet}$0.0236	$^{\bullet}$1, 092	$^{\bullet}$0.0244	$^{\bullet}$0.0317	$^{\bullet}$1, 755	$^{\bullet}$0.0248	$^{\bullet}$0.0387	$^{\bullet}$2, 142	$^{\bullet}$0.0255	$^{\bullet}$0.0440	$^{\bullet}$2, 540
JCA	$^{\bullet}$0.0242	$^{\bullet}$0.0266	$^{\bullet}$868	$^{\bullet}$0.0307	$^{\bullet}$0.0358	$^{\bullet}$1, 597	$^{\bullet}$0.0324	$^{\bullet}$0.0438	$^{\bullet}$2, 224	$^{\bullet}$0.0329	$^{\bullet}$0.0502	$^{\bullet}$2, 776	$^{\bullet}$0.0327	$^{\bullet}$0.0556	$^{\bullet}$3, 279

$^{*}p<0.1$; $^{+}p<0.05$; $^{\bullet}p<0.01$; $\times$ not significant. The p-values are reported from the comparison of the winner algorithm with all other algorithms.

Table 5: Performance of recommender methods on *MovieLens1M-Min6* with users having ≥ 6 products.

	@1			@2			@3			@4			@5		
Method	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue
Popularity	$^{\bullet}$0.0253	$^{\bullet}$0.1147	$^{\bullet}$7, 799	$^{\bullet}$0.0428	$^{\bullet}$0.1125	$^{\bullet}$16, 803	$^{\bullet}$0.0509	$^{\bullet}$0.1059	$^{\bullet}$22, 188	$^{\bullet}$0.0588	$^{\bullet}$0.1035	$^{\bullet}$27, 236	$^{\bullet}$0.0654	$^{\bullet}$0.1024	$^{\bullet}$32, 049
SVD++	$^{\bullet}$0.0253	$^{\bullet}$0.1153	$^{\bullet}$7, 837	$^{\bullet}$0.0429	$^{\bullet}$0.1128	$^{\bullet}$16, 756	$^{\bullet}$0.0510	$^{\bullet}$0.1063	$^{\bullet}$22, 145	$^{\bullet}$0.0590	$^{\bullet}$0.1040	$^{\bullet}$27, 266	$^{\bullet}$0.0656	$^{\bullet}$0.1029	$^{\bullet}$32, 220
ALS	**0.0632**	$^{\bullet}$0.2625	$^{\bullet}$17, 492	**0.0971**	$^{\bullet}$0.2401	$^{\bullet}$30, 299	**0.1188**	$^{\bullet}$0.2286	$^{\bullet}$41, 304	**0.1334**	$^{\bullet}$0.2217	$^{\bullet}$50, 942	$^{\bullet}$0.1434	$^{\bullet}$0.2174	$^{\bullet}$59, 505
DeepFM	$^{\bullet}$0.0243	$^{\bullet}$0.1611	$^{\bullet}$10, 677	$^{\bullet}$0.0465	$^{\bullet}$0.1449	$^{\bullet}$19, 084	$^{\bullet}$0.0615	$^{\bullet}$0.1380	$^{\bullet}$26, 290	$^{\bullet}$0.0726	$^{\bullet}$0.1336	$^{\bullet}$32, 771	$^{\bullet}$0.0811	$^{\bullet}$0.1317	$^{\bullet}$38, 878
NeuMF	$^{\bullet}$0.0262	$^{\bullet}$0.1675	$^{\bullet}$11, 215	$^{\bullet}$0.0493	$^{\bullet}$0.1507	$^{\bullet}$19, 927	$^{\bullet}$0.0661	0.1438	27, 613	$^{\bullet}$0.0774	$^{\bullet}$0.1398	$^{\bullet}$34, 392	$^{\bullet}$0.0856	$^{\bullet}$0.1372	$^{\bullet}$40, 503
JCA	$^{\bullet}$0.0508	**0.3414**	**23,012**	$^{\bullet}$0.0857	**0.3219**	**42,354**	$^{\bullet}$0.1123	**0.3088**	**59,596**	$^{\times}$0.1326	**0.2997**	**75,259**	**0.1487**	**0.2933**	**89,892**

$^{*}p<0.1$; $^{+}p<0.05$; $^{\bullet}p<0.01$; $\times$ not significant. The p-values are reported from the comparison of the winner algorithm with all other algorithms.

6.1 Evaluation of Metrics F1@K, NDCG@K, and Revenue@K

Table 3 presents the performance of the six methods on the *insurance dataset*. The results are reported for top-k, where $k \in \{1, ..., 5\}$. The best method on the insurance dataset is DeepFM, closely followed by JCA, SVD++ and the popularity baseline. Note that DeepFM is only outclassed in performance by SVD++ in terms of Revenue@2 and by JCA in Revenue@3 and higher. The popularity-based method produces competitive predictions, about 5% behind DeepFM's F1 and NDCG scores. It is able to exploit the large number of cold start users for a low number of products in a heavily skewed dataset. NeuMF is even further behind at 10 to 15% compared to DeepFM. However, ALS struggles to reach even half the performance of DeepFM, indicating that it is unable to utilize the popularity bias of this dataset.

The evaluation on the *MovieLens1M-Max5-Old* dataset generalizes the insurance recommendation results. It can be seen from Table 4 that in the sparse-interaction setting where, on average, users have 5 interactions, the popularity-based method and SVD++ perform best. The significance tests show that these two methods have almost identical performance. JCA's performance is about 25% behind SVD++ and the popularity-based method in terms of F1-score and revenue. ALS, DeepFM and NeuMF fall within the same significance range at about 40% behind. However, in this triplet of methods, DeepFM consistently outperforms its peers in terms of revenue and matches the revenue of JCA.

Table 5 shows the results for the *MovieLens1M-Min6* dataset. In contrast to the insurance dataset, where users had few products, the evaluation is carried out on users with 6 or more interactions. In this particular setting, JCA achieves the best result for a majority of all reported metrics. We also note that ALS performs best for F1@1 up to F1@4. While ALS is lower in terms of precision compared to JCA, it recommends a more diverse set of items and achieves a comparatively much higher recall. The popularity-based method and SVD++, the top methods on the previous iteration of MovieLens, perform here on the same level as DeepFM and NeuMF at 40% of ALS and JCA's F1-score. For the NDCG however, they perform even below DeepFM and NeuMF at 30%.

As Table 5 demonstrates, the more products we recommend, the easier it becomes to suggest a correct product and to achieve a higher F1-score. One reason is the high number of products in the ground truth of the users. For instance, in Table 2 we can see that the average number of interactions per user is 95.24. However, correctly ranking recommendations by relevance is a harder problem and NDCG decreases even for the strongly performing JCA as the number of recommendations increases.

On the *Retailrocket* dataset, no pricing information is available. Thus, we are unable to provide results for Revenue@k. The overall performance of all methods hovers just below 1% for F1@k and NDCG@k (see Table 6). We also notice that for F1@1 and NDCG@1 ALS performs best. For higher values of k, however, the popularity-based method shows the best results, though these are not strictly significant results when comparing the different methods. Among the neural network based methods, JCA is able to generate comparatively competitive predictions, but both

Table 6: Performance of recommender methods on *Retailrocket*

	@1			@2			@3			@4			@5		
Method	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue
Popularity	$^{\times}$0.0077	$^{\times}$0.0084	-	**0.0083**	**0.0104**	-	**0.0079**	**0.0118**	-	**0.0071**	**0.0124**	-	$^{\times}$**0.0063**	**0.0128**	-
SVD++	$^{+}$0.0072	$^{+}$0.0079	-	$^{\times}$0.0080	$^{\times}$0.0099	-	$^{+}$0.0071	$^{+}$0.0109	-	$^{\times}$0.0067	$^{\times}$0.0118	-	$^{\times}$**0.0063**	$^{\times}$0.0126	-
ALS	**0.0091**	**0.0100**	-	$^{\times}$0.0076	$^{\times}$0.0102	-	$^{\times}$0.0065	$^{\times}$0.0107	-	$^{\bullet}$0.0055	$^{\times}$0.0109	-	$^{\bullet}$0.0046	$^{\bullet}$0.0110	-
DeepFM	$^{\bullet}$0.0012	$^{\bullet}$0.0012	-	$^{\bullet}$0.0006	$^{\bullet}$0.0011	-	$^{\bullet}$0.0001	$^{\bullet}$0.0012	-	$^{\bullet}$0.0000	$^{\bullet}$0.0013	-	$^{\bullet}$0.0000	$^{\bullet}$0.0013	-
NeuMF	$^{\bullet}$0.0033	$^{\bullet}$0.0034	-	$^{\bullet}$0.0005	$^{\bullet}$0.0029	-	$^{\bullet}$0.0001	$^{\bullet}$0.0029	-	$^{\bullet}$0.0000	$^{\bullet}$0.0030	-	$^{\bullet}$0.0000	$^{\bullet}$0.0030	-
JCA	$^{\times}$0.0071	$^{\times}$0.0077	-	$^{+}$0.0068	$^{\times}$0.0089	-	$^{\bullet}$0.0063	$^{+}$0.0097	-	$^{+}$0.0057	$^{\times}$0.0102	-	$^{\bullet}$0.0052	$^{+}$0.0106	-

$^{*}p<0.1$; $^{+}p<0.05$; $^{\bullet}p<0.01$; $\times$ not significant. The p-values are reported from the comparison of the winner algorithm with all other algorithms.

Table 7: Performance of recommender methods on *Yoochoose-Small*

	@1			@2			@3			@4			@5		
Method	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue
Popularity	$^{\bullet}$0.0121	$^{\bullet}$0.0121	$^{+}$0	$^{+}$0.0129	$^{\bullet}$0.0166	$^{\bullet}$1, 956	**0.0119**	$^{\times}$0.0189	$^{+}$29, 329	**0.0110**	$^{\times}$0.0206	$^{\bullet}$29, 747	*0.0101	$^{\bullet}$0.0217	$^{\bullet}$46, 858
SVD++	**0.0130**	**0.0130**	$^{+}$244	**0.0132**	**0.0171**	$^{\bullet}$3, 909	**0.0119**	**0.0192**	$^{\bullet}$26, 189	$^{\times}$0.0109	**0.0208**	$^{\bullet}$32, 611	**0.0104**	**0.0223**	$^{\bullet}$57, 806
ALS	$^{\bullet}$0.0018	$^{\bullet}$0.0018	$^{\times}$10, 890	$^{\bullet}$0.0019	$^{\bullet}$0.0024	$^{\bullet}$15, 505	$^{\bullet}$0.0016	$^{\bullet}$0.0026	$^{\bullet}$16, 906	$^{\bullet}$0.0016	$^{\bullet}$0.0030	$^{\bullet}$20, 498	$^{\bullet}$0.0014	$^{\bullet}$0.0032	$^{\bullet}$21, 151
DeepFM	$^{\bullet}$0.0071	$^{\bullet}$0.0071	*4, 038	$^{\bullet}$0.0040	$^{\bullet}$0.0075	$^{+}$10, 518	$^{\bullet}$0.0024	$^{\bullet}$0.0086	$^{+}$19, 773	$^{\bullet}$0.0008	$^{\bullet}$0.0087	$^{\times}$39, 462	$^{\bullet}$0.0004	$^{\bullet}$0.0088	*45, 012
NeuMF	$^{\bullet}$0.0063	$^{\bullet}$0.0063	$^{\times}$4, 394	$^{\bullet}$0.0058	$^{\bullet}$0.0081	*18, 322	$^{\bullet}$0.0029	$^{\bullet}$0.0092	$^{\times}$30, 925	$^{\bullet}$0.0011	$^{\bullet}$0.0094	$^{\times}$43, 417	$^{\bullet}$0.0005	$^{\bullet}$0.0097	$^{\times}$54, 709
JCA	$^{\bullet}$0.0048	$^{\bullet}$0.0048	**13,268**	$^{\bullet}$0.0054	$^{\bullet}$0.0069	**36,138**	$^{\bullet}$0.0054	$^{\bullet}$0.0082	**51,928**	$^{\bullet}$0.0054	$^{\bullet}$0.0093	**65,938**	$^{\bullet}$0.0053	$^{\bullet}$0.0102	**81,132**

$^{*}p<0.1$; $^{+}p<0.05$; $^{\bullet}p<0.01$; $\times$ not significant. The p-values are reported from the comparison of the winner algorithm with all other algorithms.

Table 8: Performance of recommender methods on *Yoochoose*

	@1			@2			@3			@4			@5		
Method	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue	F1	NDCG	Revenue
Popularity	$^{\bullet}$0.0132	$^{\bullet}$0.0132	$^{\bullet}$85	$^{\bullet}$0.0138	$^{\bullet}$0.0177	$^{\bullet}$46, 364	$^{\bullet}$0.0139	$^{\bullet}$0.0210	$^{\bullet}$601, 826	$^{\bullet}$0.0120	$^{\bullet}$0.0219	$^{\bullet}$612, 250	$^{\bullet}$0.0112	$^{\bullet}$0.0232	$^{\bullet}$1, 159, 084
SVD++	$^{\bullet}$0.0132	$^{\bullet}$0.0132	$^{\bullet}$85	$^{\bullet}$0.0138	$^{\bullet}$0.0177	$^{\bullet}$46, 364	$^{\bullet}$0.0139	$^{\bullet}$0.0210	$^{\bullet}$601, 826	$^{\bullet}$0.0120	$^{\bullet}$0.0219	$^{\bullet}$612, 127	$^{\bullet}$0.0112	$^{\bullet}$0.0233	$^{\bullet}$1, 159, 502
ALS	**0.1089**	**0.1143**	**9,076,133**	**0.1079**	**0.1375**	**12,852,633**	**0.0978**	**0.1519**	**15,030,259**	**0.0874**	**0.1606**	**16,474,896**	**0.0782**	**0.1660**	**17,411,197**
DeepFM	$^{\bullet}$0.0134	$^{\bullet}$0.0136	$^{\bullet}$132, 492	$^{\bullet}$0.0076	$^{\bullet}$0.0138	$^{\bullet}$373, 954	$^{\bullet}$0.0038	$^{\bullet}$0.0140	$^{\bullet}$539, 194	$^{\bullet}$0.0015	$^{\bullet}$0.0141	$^{\bullet}$714, 781	$^{\bullet}$0.0006	$^{\bullet}$0.0143	$^{\bullet}$940, 544
NeuMF	$^{\bullet}$0.0127	$^{\bullet}$0.0128	$^{\bullet}$7, 063	$^{\bullet}$0.0101	$^{\bullet}$0.0145	$^{\bullet}$216, 167	$^{\bullet}$0.0061	$^{\bullet}$0.0151	$^{\bullet}$421, 229	$^{\bullet}$0.0036	$^{\bullet}$0.0156	$^{\bullet}$868, 317	$^{\bullet}$0.0020	$^{\bullet}$0.0159	$^{\bullet}$1, 012
JCA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$^{*}p<0.1$; $^{+}p<0.05$; $^{\bullet}p<0.01$; $\times$ not significant. The p-values are reported from the comparison of the winner algorithm with all other algorithms.

DeepFM and NeuMF perform significantly worse than all other methods, with F1-scores of almost zero for F1@3 and higher. DeepFM's F1-scores and NDCG are even an order of magnitude below the popularity-based method. The reason for the low performance of all algorithms is due to the extreme sparsity and skewness of this dataset (revisit the dataset statistics shown in Table 1 for details). Under these "extreme conditions" it is hard for the algorithms to learn a specific pattern.

Let us now analyze the behavior of the algorithms for the datasets *Yoochoose-Small* and *Yoochoose* (see Tables 7 and 8). The smaller dataset *Yoochoose-Small* has been chosen to measure the performance of JCA, since it was impossible to run JCA on the full Yoochoose dataset. Due to the selection of just 5% of interactions in Yoochoose-Small, the percentage of cold start users increases by over a factor of 3 from 28.91% to 90.42%. Similar to MovieLens1M-Max5 dataset, both the popularity-based method and SVD++ are outperforming all other methods in terms of F1-score and NDCG across all values of k. However, the large number of cold start users make this dataset difficult to solve. Furthermore, users that do have an existing interaction history have only a low number of products, at most five. In these harsh conditions, primarily relying on the popularity bias looks to be the only learnable pattern in the dataset.

The full Yoochoose dataset poses a difficult challenge, since it features the largest number of items (i.e., 19, 949) across all datasets. Predicting the top 1 to 5 out of almost 20 thousand items becomes truly challenging. With over half a million of users, and just over a million of interactions, the dataset — while heavily skewed — is also the most sparsely populated one. SVD++ and the popularity-based method reach similar performance metrics compared to the Yoochoose-Small dataset above the 1% mark. However, DeepFM and NeuMF are now able to match their performance for $k = 1$, but not for higher values of k. Unlike for the Yoochoose-Small dataset, ALS is able to learn a pattern and reaches 10.89% F1@1, steadily decreasing to 7.82% F1@5.

In summary, the datasets *Retailrocket* and *Yoochoose* are good examples for stress testing recommender algorithms. They also demonstrate that simpler methods seem to have more robust and scalable implementations than the neural network approaches.

6.2 Summary of the results

A summary of the overall recommender performance in terms of mean F1-score, NDCG and revenue across all values of $k \in [1, 5]$, is given in Table 9. Methods are ranked from 1 through 5, with 1 giving the best performance on the particular dataset. While this table lacks the relative differences in performance, it allows to quickly see the condensed overall result from the previous result tables.

Table 9: Overall recommender performance ranking.

Dataset	Popularity	SVD++	ALS	DeepFM	NeuMF	JCA
Insurance	4	3	6	**1**	5	2
MovieLens1M-Max5	**1**†	**1**†	5†	4	5†	3
MovieLens1M-Min6	5†	5†	2	4	3	**1**
Retailrocket	**1**†	**1**†	**1**†	6	5	4
Yoochoose-Small	**1**†	**1**†	6	4†	4†	3
Yoochoose	2†	2†	**1**	4†	4†	-
Average Rank	2.33	**2.17**	3.50	3.83	4.33	3.17*

† This rank is given to two or more methods, as their performance is within one standard deviation. See the individual dataset performance tables for reference.
* JCA was unable to be trained in reasonable time on Yoochoose and thus could not be ranked on the full Yoochoose dataset. The average rank was calculated counting its performance on Yoochoose as rank 6.

The popularity-based method performs surprisingly well, given how naïve and computationally efficient predicting the popularity bias is. SVD++ always performs similar to the popularity-based method (as indicated by the matching † symbols in Table 9 for *Popularity* and *SVD++*), suggesting that this method heavily relies on learning the popularity bias in a dataset. This begs the question, if the training of a SVD++ model is even necessary, given how a similar performance can be reached by the popularity-based method. ALS looks to be a method that should be in every recommender system toolbox. While not the best method over all datasets, its ability to conquer a difficult dataset like Yoochoose makes it potent. In addition, ALS is a very computationally efficient method, as we will show later in Section 6.3. NeuMF and DeepFM are conceptually similar methods, and it is not surprising that their performance is generally similar. However, the only dataset where they shine (the insurance dataset), is a dataset where almost all other methods perform well already (with the exception of ALS). JCA as the only pure neural network method has competitive results on 3 out of 6 datasets. However, feeding the full user-item matrix through the JCA network during training has a risk of memory errors, especially if both user- and item dimensions grow large.

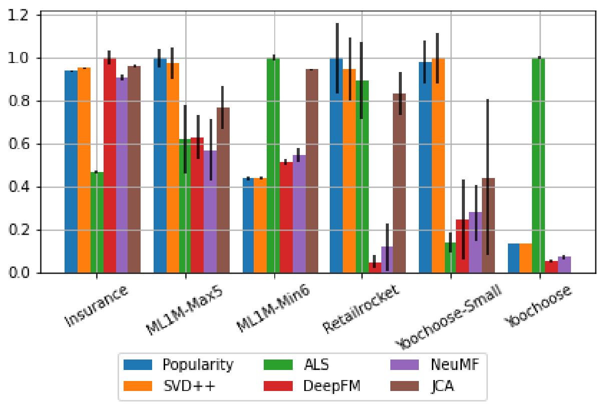


Figure 6: Comparison of the average F1-score across all methods and datasets, scaled to the maximum per dataset. Error bars indicate one standard deviation.

Figure 6 gives a summary of all results in terms of F1-score. Each bar represents the mean of all F1@1 through F1@5 measurements and the error bar is one standard deviation. All methods except ALS achieve similar F1-scores on the insurance dataset. Hence, other measures (such as revenue) need to be considered when choosing the best method. In this dataset, the popularity bias has to be learned. Table 1 in Section 5.1 shows that the number of users far outweighs the number of items. However, methods need to go beyond the popularity bias and be able to predict items that are rarely interacted with. ALS is the only method that shows to be struggling with this dataset, perhaps because it has difficulties dealing with outliers.

In the MovieLens-Max5 dataset, the goal is to reproduce the characteristics of the insurance dataset. There remain a few differences, especially in the ratio between users and items and the percentage of cold start users. We believe that the overall performance of all methods dropped by an order of magnitude compared to the insurance dataset, due to the lower number of interactions per item. The MovieLens-Max5 dataset has a similar skewness, but far fewer interactions have to be spread across far more items. Compared to the methods' performance on the insurance dataset, the relative spread remains identical except for the neural network based methods (DeepFM, NeuMF and JCA). This behavior suggests that not enough interactions were present to capture the patterns in the underlying interaction history.

The MovieLens1M-Min6 dataset portrays the performance on a typical non interaction-sparse dataset. Unlike with the other datasets, here the methods are able to exploit much larger patterns, as users on average interact with more than just a handful of items. The popularity-based method and SVD++ achieve the lowest results, as it becomes much more important to generate predictions that relate to pre-existing interactions. Because this dataset is much more densely populated than all other datasets, both ALS and JCA are able to exploit the additional number of interactions per user and item better than their counterparts. ALS in particular has to deal with a lot fewer outliers, since every user and item have at least six interactions.

Retailrocket is a challenging dataset because of the low user to item ratio, low density and highest skewness out of all our tested datasets. All methods perform quite poorly on this dataset, with F1-scores and NDCG values below 1%. With just 1.82 and 1.77 interactions per user and item, respectively, the challenge of predicting the correct items out of 12 thousand items is arduous. In addition, almost half (45.58%) of all items are never seen during training, eliminating a large number of items as potential predictions. DeepFM and NeuMF in particular struggle with this dataset — significantly more than the other approaches. The fusion of matrix factorization and neural networks might lead to worse predictions, as the individual components produce conflicting signals. All other methods exploit at least the popularity bias in the dataset.

The Yoochoose-Small dataset — while partially matching statistics of the insurance dataset — shows the same F1-score pattern as in the insurance case with ALS trailing far behind the other methods. However, while JCA was able to closely match the F1-scores of SVD++ and the popularity-based method, it is unable to repeat this on Yoochoose-Small. One of the main differences between the two datasets is the number of cold start users. Where the insurance dataset features about 50%, Yoochoose-Small has over 90% cold start users, suggesting that JCA struggles when users are new. Similarly, NeuMF and DeepFM also struggle with the increase in cold start users and far fewer interactions per item. These methods tend to perform better, if there are fewer numbers of items in relation to the number of users.

Finally, for the Yoochoose dataset results of JCA are not available. ALS clearly performs best with an order of magnitude lead over all other methods. This suggests that ALS is able to extract a pattern which is disconnected from the popularity bias that SVD++ and the popularity-based method are primarily exploiting. Another explanation could be that the interaction patterns of Yoochoose were broken apart by subsampling 5% of all interactions for Yoochoose-Small — and ALS is the only method to model the interaction pattern in such a hostile dataset. When looking at the absolute F1-scores and NDCG values, the other methods remain in the same orders of magnitudes as with Yoochoose-Small, largely performing very similarly.

Figure 7 gives a summary of all results in terms of revenue. Each bar represents the mean of all Revenue@1 through Revenue@5 measurements (error bars indicate one standard deviation). Since Retailrocket does not contain pricing information, it is omitted.

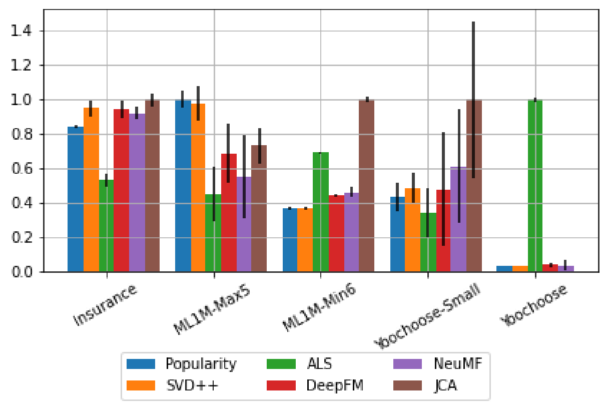


Figure 7: Comparison of the average revenue across all methods and datasets, scaled to the maximum per dataset. Error bars indicate one standard deviation.

On the insurance dataset, both the popularity-based method and SVD++ achieve less revenue than the F1-score would suggest. The Yoochoose-Small dataset demonstrates the same pattern even more pronounced — both the popularity-based method and SVD++ closely match the F1-score but in terms of revenue are only able to slightly surpass the revenue generated by ALS.

6.3 Training Time

Finally, we evaluate the run time of all six methods on all six datasets. We measure the mean training time per epoch. Note that this information is often omitted in research papers. However, when applying recommender systems in practice as in our use case, the training time is crucial — in particular when neural networks need to be retrained periodically. All experiments are executed on an NVIDIA TITAN Xp GPU.

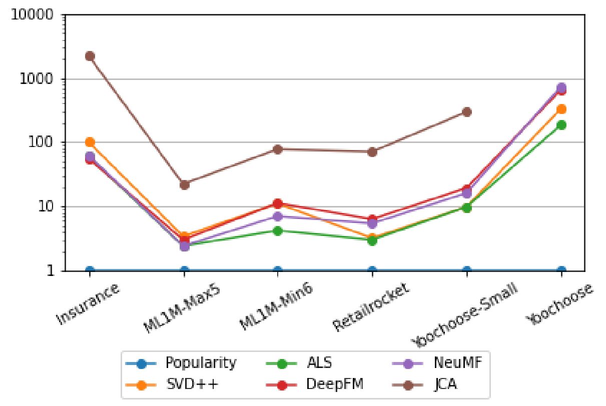


Figure 8: The mean training time per epoch in seconds.

Figure 8 shows the mean training time for one epoch across all datasets. The y-axis is in seconds and markers are placed at increasing powers of 10 (log-scale). Since the popularity-based method only needs to extract the frequency of items in the dataset, it was added to Figure 8 with an "honorary" 1 second training time. In the remainder of this section, we will focus our comparison only on those methods which need to be trained.

All trained methods slow down with the number of interactions, users and items. However, the SVD++, ALS, DeepFM and NeuMF manage to be trained faster than JCA by an order of magnitude or more. Moreover, JCA could not be trained on the Yoochoose dataset due to memory issues. Finally, these results also demonstrate that the simple popularity-based method not only shows high F1-scores but also has the added benefit of not needing any computationally expensive training — knowing the item frequencies suffices.

7 CONCLUSION & FUTURE WORK

This investigation set out to analyze the possibility of using recommender systems in the special insurance setting. This special, revenue-driven interaction-sparse setting poses new challenges to recommender systems that—to the best of our knowledge—were not yet explored in detail.

The main lessons learned of our systematic experimental evaluation are as follows:

- In the *sparse setting* of the insurance data with *medium skewness*, DeepFM showed better performance than JCA, the state of the art neural network approach for recommender systems. However, DeepFM is not very robust in other datasets and should be used carefully. Moreover, JCA is a very memory-intensive algorithm and needs implementation improvements to work well for extremely large combinations of users and items.
- Matrix factorization methods showed better performance than neural network methods for datasets with *higher skewness* than the insurance dataset and a *high ratio of cold start users*. Especially SVD++ is generally very robust across different skewness levels and cold start ratios.
- The popularity-based method showed surprisingly good performance across all datasets except for one.
- While a large part of the research community primarily focuses on the sparsity and skewness metrics, the number of interactions per user and item, respectively, is another important metric to take into consideration.
- In summary, there is no clear winner among the algorithms for the different dataset characteristics. Hence, in a real-world scenario with interaction-sparse characteristics one should *use a portfolio of algorithms consisting of matrix factorization and neural network methods*. Moreover, the *popularity-based approach* should always be part of the portfolio due to its good performance and easy interpretability. The latter is of particular importance for sales representatives who need to justify their recommendations to both their customers and superiors.

As part of future work, we will study more complex revenue-optimized methods such as multi-objective optimization [21]. In addition to revenue, an interesting optimization objective is fairness. In the context of the insurance domain, a typical question could be if a customer is over- or under-ensured and hence pays a fair price with respect to the insurance coverage.

Moreover, the findings provided here indicate that we can possibly choose an optimal recommendation algorithm based on data properties (in our case the skewness of R indicates whether to choose a neural network method or a matrix factorization method). This echos ideas from machine learning on algorithm selection and may inform Auto-ML approaches for recommendations. As such, we believe that this work paves the way for finding optimal recommendation algorithms for a given dataset based on data properties.

REFERENCES

[1] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. 2015. RecSys Challenge 2015 and the YOOCHOOSE Dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*. 357–358.

[2] Ye Bi, Liqiang Song, Mengqiu Yao, Zhenyu Wu, Jianming Wang, and Jing Xiao. 2020. DCDIR: A Deep Cross-Domain Recommendation System for Cold Start Users in Insurance Domain. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1661–1664.

[3] Chao Chen, Dongsheng Li, Qin Lv, Junchi Yan, Stephen M Chu, and Li Shang. 2016. MPMA: Mixture Probabilistic Matrix Approximation for Collaborative Filtering. In *IJCAI*. 1382–1388.

[4] Chao Chen, Dongsheng Li, Qin Lv, Junchi Yan, Li Shang, and Stephen M Chu. 2017. GLOMA: Embedding Global Information in Local Matrix Approximation Models for Collaborative Filtering. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[5] Chao Chen, Dongsheng Li, Yingying Zhao, Qin Lv, and Li Shang. 2015. WEMAREC: Accurate and Scalable Recommendation through Weighted and Ensemble Matrix Approximation. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 303–312.

[6] Jiawei Chen, Can Wang, Sheng Zhou, Qihao Shi, Jingbang Chen, Yan Feng, and Chun Chen. 2020. Fast Adaptively Weighted Matrix Factorization for Recommendation with Implicit Feedback. In *AAAI*. 3470–3477.

[7] Xiancong Chen, Lin Li, Weike Pan, and Zhong Ming. 2020. A Survey on Heterogeneous One-class Collaborative Filtering. *ACM Transactions on Information Systems (TOIS)* 38, 4 (2020), 1–54.

[8] J. Mary F. Strub, R. Gaudel. 2016. Hybrid Recommender System Based on Autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (Boston, MA, USA) *(DLRS 2016)*. Association for Computing Machinery, New York, NY, USA, 11–16. https://doi.org/10.1145/2988450.2988456

[9] Gene H Golub and Christian Reinsch. 1971. Singular Value Decomposition and Least Squares Solutions. In *Linear algebra*. Springer, 134–151.

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *arXiv preprint arXiv:1703.04247* (2017).

[11] F Maxwell Harper and Joseph A Konstan. 2015. The MovieLens Datasets: History and Context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[12] Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. *The Journal of Machine Learning Research* 16, 1 (2015), 3367–3402.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[14] G. Lebanon J. Lee, S. Kim and Y. Signer. 2013. Local Low-Rank Matrix Approximation. In *In Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*. PMLR, Atlanta, Georgia, USA, 82–90.

[15] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored Item Similarity Models for Top-N Recommender Systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 659–667. https://doi.org/10.1145/2487575.2487589

[16] Yehuda Koren. 2008. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 426–434.

[17] Laurent Lesage, Madalina Deaconu, Antoine Lejay, Jorge Augusto Meira, Geoffrey Nichil, et al. 2020. A Recommendation System For Car Insurance. *European Actuarial Journal* 10, 2 (2020), 377–398.

[18] Dongsheng Li, Chao Chen, Wei Liu, Tun Lu, Ning Gu, and Stephen Chu. 2017. Mixture-Rank Matrix Approximation for Collaborative Filtering. *Advances in Neural Information Processing Systems* 30 (2017).

[19] Dongsheng Li, Chao Chen, Qin Lv, Hansu Gu, Tun Lu, Li Shang, Ning Gu, and Stephen M Chu. 2018. AdaError: An Adaptive Learning Rate Method for Matrix Approximation-based Collaborative Filtering. In *Proceedings of the 2018 World Wide Web Conference*. 741–751.

[20] Dongsheng Li, Chao Chen, Qin Lv, Junchi Yan, Li Shang, and Stephen Chu. 2016. Low-Rank Matrix Approximation with Stability. In *International Conference on Machine Learning*. PMLR, 295–303.

[21] Nikola Milojkovic, Diego Antognini, Giancarlo Bergamin, Boi Faltings, and Claudiu Musat. 2019. Multi-Gradient Descent for Multi-Objective Recommender Systems. *arXiv preprint arXiv:2001.00846* (2019).

[22] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-Class Collaborative Filtering. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 502–511.

[23] Karl Pearson. 1895. Contributions to the Mathematical Theory of Evolution. *Philosophical Transactions of the Royal Society of London.(A.)* 186 (1895), 343–414.

[24] Maleeha Qazi, Glenn M Fung, Katie J Meissner, and Eduardo R Fontes. 2017. An Insurance Recommendation System Using Bayesian Networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 274–278.

[25] Maleeha Qazi, Kaya Tollas, Teja Kanchinadam, Joseph Bockhorst, and Glenn Fung. 2020. Designing and Deploying Insurance Recommender Systems using Machine Learning. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2020), e1363.

[26] Roy Radner. 2003. Viscous Demand. *Journal of Economic Theory* 112, 2 (2003), 189–231. https://doi.org/10.1016/S0022-0531(03)00115-7

[27] S. Rendle. 2012. Factorization Machines with LibFM. *ACM Trans. Intell. Syst. Technol.* 3, 3, Article 57 (May 2012), 22 pages. https://doi.org/10.1145/2168752.2168771

[28] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian Personalized Ranking from Implicit Feedback. *arXiv preprint arXiv:1205.2618* (2012).

[29] Retailrocket. 2017. Retailrocket Recommender System Dataset. Retrieved September 28th, 2021 from https://www.kaggle.com/retailrocket/ecommerce-dataset/

[30] S. Sanner S. Sedhain, A. K. Menon and L. Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15 Companion)*. Association for Computing Machinery, New York, NY, USA, 111–112. https://doi.org/10.1145/2740908.2742726

[31] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. *Application of Dimensionality Reduction in Recommender System – A Case Study*. Technical Report. Minnesota Univ Minneapolis Dept of Computer Science.

[32] Koen Verstrepen, Kanishka Bhaduriy, Boris Cule, and Bart Goethals. 2017. Collaborative Filtering for Binary, Positive-only Data. *ACM SIGKDD Explorations Newsletter* 19, 1 (2017), 1–21.

[33] L. Chen W. Pan. 2013. CoFiSet: Collaborative Filtering via Learning Pairwise Preferences over Item-sets. In *Proceedings of the 2013 SIAM international conference on data mining*. SIAM, 180–188.

[34] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A Theoretical Analysis of NDCG Type Ranking Measures. In *Conference on Learning Theory*. PMLR, 25–54.

[35] RF Woolson. 2007. Wilcoxon Signed-Rank Test. *Wiley encyclopedia of clinical trials* (2007), 1–3.

[36] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 153–162.

[37] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. 2019. Deep Item-Based Collaborative Filtering for Top-N Recommendation. *ACM Trans. Inf. Syst.* 37, 3, Article 33 (April 2019), 25 pages. https://doi.org/10.1145/3314578

[38] Shuang-Hong Yang, Bo Long, Alexander J. Smola, Hongyuan Zha, and Zhaohui Zheng. 2011. Collaborative Competitive Filtering: Learning Recommender Using Context of User Choice. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Beijing, China) *(SIGIR '11)*. Association for Computing Machinery, New York, NY, USA, 295–304. https://doi.org/10.1145/2009916.2009959

[39] J. Wang Z. Zhu and J. Caverlee. 2019. Improving Top-K Recommendation via Joint Collaborative Autoencoders. In *In The World Wide Web Conference*. ACM, 3483–3482.

[40] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.

[41] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A Neural Autoregressive Approach to Collaborative Filtering. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 764–773. http://proceedings.mlr.press/v48/zheng16.html

[42] Daniel Zwillinger and Stephen Kokoska. 1999. *CRC Standard Probability and Statistics Tables and Formulae*. Crc Press.