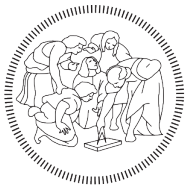


POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

AN ASSESSMENT OF REPRODUCIBILITY AND METHODOLOGICAL ISSUES IN NEURAL RECOMMENDER SYSTEMS RESEARCH

Doctoral Dissertation of:
Maurizio Ferrari Dacrema

Supervisor:
Prof. Paolo Cremonesi

Tutor:
Prof. Francesco Amigoni

The Chair of the Doctoral Program:
Prof. Barbara Pernici

2020 – Cycle XXXII

Abstract

The design of algorithms that generate personalized ranked item lists is a central topic of research in the field of recommender systems. In recent years, in particular, the interest of the research community has moved towards neural approaches based on deep learning, which have become dominant in the literature. Since each of those publications claims substantial progress over the state-of-the-art, it seems logical to expect the research field to be on a steady trajectory of increased effectiveness. However, several studies indicated the existence of certain problems in today's research practice, e.g., with respect to the choice and optimization of the baselines used for comparison or to the design of the experimental protocol itself, raising questions about the published claims. In order to assess the level of progress, reproducibility and the existence of issues in the current recommender systems research practice, this thesis attempts to reproduce recent results in the area of neural recommendation approaches based on collaborative filtering. The analysis in particular focuses on articles published at high level scientific conferences between 2015 and 2018. The results is that out of 24 articles, only 12 can be reproduced and only 1 shows to be consistently competitive against simple methods, e.g., based on the nearest-neighbor heuristics or linear machine learning. In our analysis, we discuss this surprising result and trace it back to several common issues in today's research practice, which, despite the many papers that are published on the topic, have apparently led the recommender system field, for the task considered in our analysis, to a certain level of stagnation.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In the era of exponential information growth, personalized recommendations have become an important part of many online services we use today.

Among the earliest approaches to perform personalized recommendations are *neighborhood-based* algorithms, developed in the mid 1990s [98] and early 2000s [104]. Those algorithms are based on a similarity between items or users computed via heuristics, several of which are available and have either been developed for specific recommendation tasks [2] or have been borrowed from other domains [39, 113].

A very important milestone for the recommender systems community was the Netflix Prize [11], which saw *matrix factorization* algorithms emerge as a new and very competitive family of models [61, 66, 70, 96]. All matrix factorization algorithms represent users and items in a low dimensional latent space and the interactions between them is modeled by a dot product of their embeddings. As opposed to the previous methods, matrix factorization models rely mostly, but not exclusively, on machine learning.

Following the increasing popularity and success of deep learning in several application domains, e.g., computer vision and text processing, nu-

merous neural approaches for collaborative filtering have been proposed in recent years. For the most part, researchers try to apply to the recommendation problem neural architectures and techniques that were proven successful in other application areas, with the necessary adaptations [29, 132].

Despite several years of apparent continuous progress, evidence is starting to mount that at least a portion of the algorithms claimed to outperform the state-of-the-art, in fact, fail to do so. In those cases, the original claim was due to a poor evaluation procedure.

More than a decade ago Armstrong et al. [4] observed that there was no evidence of improvement for certain information filtering tasks over the previous decade. They observed that the tendency to report weak baselines could easily give the impression of continuous incremental improvements, even though none existed. Their findings have been recently confirmed and further discussed by other studies [65]. In particular, in 2019 Yang et al. [127] too observed how in an evaluation study little evidence emerged that new complex neural methods for information retrieval were in fact competitive against simple long known baselines. Further to this point, Lin [74] observed how common it is to find new neural ranking papers reporting experimental evaluations on inadequately tuned baselines.

For recommender systems, only few of such evaluation studies have been published [4, 65, 102]. For session-based recommendations, the empirical analyses in [80] showed that in some cases very simple algorithms can outperform recent more complex neural methods.

Another recent analysis for rating prediction tasks authored by Rendle et al. in [97] analyzed articles published between 2015 and 2019 finding that the results reported in those articles actually did not outperform much older methods, when those were properly optimized. Rendle et al. traced those results to both the tendency to report poorly optimized baselines and to the lack of widely used standardized benchmarks. This finding confirms previous ones [65] observing how, despite several of such benchmarks being available [102], their use remains limited.

Due to this mounting evidence, we were left wondering to what extent recent complex neural methods were actually able to improve the quality of another recommendation task, *top-k* recommendation based on user-item rating matrices.

To that purpose, we conducted an extensive evaluation of the neural algorithms published in high-level conferences which are common venues for recommender systems research. The study included two parts. The first part is an analysis of the reproducibility of published results, an essential part of the scientific process. The second part is an evaluation of new complex

neural models when compared against simple and long known baselines, which often are not reported anymore.

1.2 Research Contributions

In this section we detail the research contributions of this study, which are primarily related to a critical analysis of current research practices in the recommender systems field.

Research Contribution 1: Assessment of reproducibility of published research on top-k recommendation with neural approaches. A preliminary step for the subsequent evaluation study is to be able to reproduce the results reported in published research. To this end, we reported a detailed list of all the articles we analyze, for each one indicating the publicly available source code and stating whether we could reproduce the published result or not. If reproducing the results was not possible, we provide a motivation. We also attempted to communicate with the authors of the articles if we encountered issues and report the results of this interactions.

Research Contribution 2: Assessment of the competitiveness of neural algorithms for top-k recommendation against simple non-neural baselines. As mentioned in the Motivation, several previous articles for various information retrieval and recommender systems tasks observed that new complex methods were never competitive against older and much simpler baselines. In order to measure what level of progress has been achieved thanks to those methods, we design a thorough evaluation study. In this study we compare all reproducible neural algorithms against an ample set of simple baselines belonging to different families of models, ensuring their proper optimization.

Research Contribution 3: Assessment on the principal causes for the lack of reproducibility and ways for improvement. From our reproducibility analysis in *Research Contribution 1* we could observe only a limited number of algorithms to be reproducible. We therefore report and discuss the issues we encountered and compare them with the ones observed by other reproducibility studies in other fields of machine learning. We also discuss possible directions for improvement.

Research Contribution 4: Assessment of the robustness of current methodological practices and ways for improvement. From our evaluation in *Research Contribution 2* we could observe all but one algorithm was consistently competitive against simple and well-known baselines. In order to explain the reason for this remarkable result, we provide a detailed

analysis of the evaluation protocols each algorithm adopted and point out at several possible limitations of the current experimental practice. Among the most important observations, the apparent arbitrariness of the experimental design as well as the often inadequate selection and optimization of baselines. Furthermore, several errors in the evaluation protocols and in the implementation of the original articles were observed. This analysis not only uncovers severe shortfalls in the current experimental practice regarding individual articles, it also shows how the effects spread in the future as weak algorithms become the state-of-the-art and propagate to later ones.

1.3 List of Publications

1.3.1 Invited Conference Paper

[47] *Methodological Issues in Recommender Systems Research*, **Maurizio Ferrari Dacrema**, Paolo Cremonesi, Dietmar Jannach, In *Proceedings of the 29th International Joint Conference on Artificial Intelligence* (IJCAI 2020) (Extended Abstract)

1.3.2 Journal Publication

[38] *Movie Genome: Alleviating New Item Cold Start in Video Recommendation*, Yashar Deldjoo, **Maurizio Ferrari Dacrema**, Mihai Gabriel Constantin, Hamid Eghbal-zadeh, Stefano Cereda, Markus Schedl, Bogdan Ionescu, Paolo Cremonesi, *User Modeling and User-Adapted Interaction* (UMUAI 2019)

[35] *A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research*, **Maurizio Ferrari Dacrema**, Simone Boglio, Paolo Cremonesi, Dietmar Jannach, *ACM Transactions on Information Systems* (ACM TOIS 2019) (Under review)

1.3.3 Book Chapter

[30] *Cross-domain recommender systems*, Paolo Cremonesi, **Maurizio Ferrari Dacrema**, Shlomo Berkovsky, Iván Cantador and Ignacio Fernández-Tobías, Chapter In Recommender systems handbook. *Springer*, 2020.

[31] *User Preference Sources: Explicit vs. Implicit Feedback*, Paolo Cremonesi, Franca Garzotto and **Maurizio Ferrari Dacrema**, Chapter In Collaborative Recommendations: Algorithms, Practical Challenges and Applications. *World Scientific Publishing Company*, 2019.

1.3.4 Conference Paper

[46] *Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches*, **Maurizio Ferrari Dacrema**, Paolo Cremonesi, Dietmar Jannach, In *Proceedings of the 13th ACM Conference on Recommender Systems* (RecSys 2019) (Long paper) **Best Long Paper Award**

[49] *Critically Examining the Claimed Value of Convolutions over User-Item Embedding Maps for Recommender Systems*, **Maurizio Ferrari Dacrema**, Federico Parroni, Paolo Cremonesi, Dietmar Jannach, In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management* (CIKM 2020) (Long paper)

[91] *ContentWise Impressions: An industrial dataset with impressions included*, Fernando Benjamín Pérez Maurera, **Maurizio Ferrari Dacrema**, Lorenzo Saule, Mario Scriminaci, Paolo Cremonesi, In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management* (CIKM 2020) (Long paper)

[14] *Estimating Confidence of Individual User Predictions in Item-based Recommender Systems*, Cesare Bernardis, **Maurizio Ferrari Dacrema**, Paolo Cremonesi, In *Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization* (UMAP 2019) (Long paper)

[13] *A novel graph-based model for hybrid recommendations in cold-start scenarios*, Cesare Bernardis, **Maurizio Ferrari Dacrema**, Paolo Cremonesi, In *Proceedings of the 12th ACM Conference on Recommender Systems* (RecSys 2018) (Poster)

[45] *Eigenvalue analogy for confidence estimation in item-based recommender systems*, **Maurizio Ferrari Dacrema**, Paolo Cremonesi, In *Proceedings of the 12th ACM Conference on Recommender Systems* (RecSys 2018) (Poster)

1.3.5 Workshop Paper

[48] *Deriving item features relevance from collaborative domain knowledge*, **Maurizio Ferrari Dacrema**, Alberto Gasparin, Paolo Cremonesi, In *Proceedings of the Workshop on Knowledge-aware and Conversational Recommender Systems 2018* (RecSys 2018)

[3] *Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario*, Sebastiano Antenucci, Simone Boglio, Emanuele Chioso, Ervin Dervishaj, Shuwen Kang, Tommaso Scarlatti, **Maurizio Ferrari Dacrema**, In *Proceedings of the ACM Recommender Systems Challenge 2018* (RecSys 2018)

[28] *Towards Evaluating User Profiling Methods Based on Explicit Ratings on Item Features*, Luca Luciano Costanzo, Yashar Deldjoo, **Maurizio Ferrari Dacrema**, Markus Schedl, Paolo Cremonesi, *Proceedings of Joint Workshop on Interfaces and Human Decision Making for Recommender Systems (IntRS 2019) at the 13th ACM Conference on Recommender Systems* (RecSys 2019)

[36] *Leveraging laziness, Browsing-Pattern Aware Stacked Models for Sequential Accommodation Learning to Rank*, Edoardo D'Amico, Giovanni Gabbolini, Daniele Montesi, Matteo Moreschini, Federico Parroni, Federico Piccinini, Alberto Rossettini, Alessio Russo, Cesare Bernardis, **Maurizio Ferrari Dacrema**, In *Proceedings of the ACM Recommender Systems Challenge 2019* (RecSys 2019)

[44] *Multi-Objective Blended Ensemble For Highly Imbalanced Sequence Aware Tweet Engagement Prediction*, Nicolò Felicioni, Andrea Donati, Luca Conterio, Luca Bartoccioni, Davide Yi Xian Hu, Cesare Bernardis, **Maurizio Ferrari Dacrema**, In *Proceedings of the ACM Recommender Systems Challenge 2020* (RecSys 2020)

1.4 Structure

- **Chapter 1** introduces this thesis and describes its motivations and research contributions, as well as list the publications related directly or indirectly with the work carried out for the writing of this thesis.
- **Chapter 2** introduces the main concepts for neural recommender systems and the architectures analyzed in this thesis, as well as the issue of reproducibility of published research.
- **Chapter 3** details the criteria adopted for the reproducibility analysis, therefore how were the articles surveyed and which is the adopted definition of reproducibility.
- **Chapter 4** details the evaluation protocol used by the comparative analysis and describes all baseline algorithms used, the optimization

procedure and the metrics that are reported in the evaluation. The evaluation framework used for the experiments is also described.

- **Chapter 5** reports the detailed results for each reproducible algorithm, providing a brief overview of the algorithm, the specific evaluation protocol adopted in the original paper, the results when comparing its recommendation quality with the baselines and an analysis of the methodological issues we could observe in the paper or in the available implementation.
- **Chapter 6** reports the analysis of a subset of algorithms which made specific claims regarding the modeling capacity of Convolutional Neural Networks. The chapter reports the results of a specifically designed study to verify those claims.
- **Chapter 7** provides an overview of the results and a detailed discussion along several dimensions both related to the reproducibility of the results and to the methodological issues that could be observed.
- **Chapter 8** presents the conclusions.
- **Appendix A** reports the detailed list of all hyperparameters for the baseline algorithms, their range and distribution.
- **Appendix B** describes a side contribution of this thesis demonstrating the equivalence of a group of beyond-accuracy metrics.
- **Appendix C** reports the full results for all analyzed algorithms of which only the most significant ones are reported in the main text.

CHAPTER 2

Preliminaries

In this Chapter we will first provide a brief overview of the architectures used in neural recommender systems we analyze in this study. Secondly, we will provide an overview on reproducibility for different fields of computer science and the definition of reproducibility we apply in this study.

2.1 Neural Recommender Systems

In recent years a multitude of deep learning architectures have been developed for several tasks, e.g., computer vision, natural language processing, audio recognition. For the purpose of this study we will briefly describe the architectures that have been used in the recommender systems field, in particular those employed by the algorithms we analyze [132]:

- Multi-layer perceptron
- Autoencoder
- Convolutional Neural Network

2.1.1 Multi-layer Perceptron

A Multi-layer perceptron (MLP) is a type of feed-forward neural network which consists of at least three layers: an input layer, one or more hidden layers and an output layer. Each layer has a non-linear activation function and is fully connected. MLPs are known to be *universal approximators* under certain assumptions [34,59]. In the recommender system field, MLPs are often used to learn embeddings, projecting the data into a latent space in a similar way as traditional matrix factorization algorithms do [57, 125].

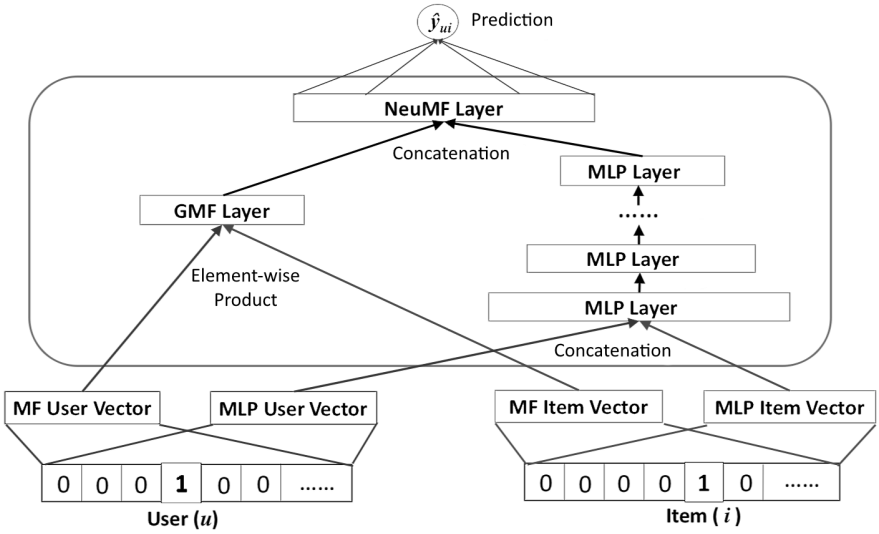
To illustrate an example of MLP applied to a recommender systems problem we can refer to perhaps the most cited among the articles we analyze, Neural Collaborative Filtering (NCF) [57]. Figure 2.1 shows the overall architecture which is constituted by two components: Generalized Matrix Factorization (GMF) and a MLP. GMF is used to generalize matrix factorization and simply computes the element-wise product of the user and item embeddings, which is later fed to a fully connected layer allowing to generalizes the traditionally used dot product. The MLP component instead concatenates user and item embeddings using them as input for a MLP neural network. The MLP network allows to model not only non-linear behavior but also the interactions between embedding dimensions which in traditional matrix factorization algorithms are not modeled. The network structure follows the common strategy of a pyramid pattern, where the input layer is the widest and every successive layer becomes smaller by a certain coefficient. In case of NCF the number of neurons is halved at every layer.

In the end, the final layers of GMF and MLP are concatenated and fed to a last dense layer which computes the final prediction. This allows the two parts to learn separate embedding sizes. The NCF framework allows significant flexibility (e.g., choice of network structure, activation function, embedding size) and has been further developed and specialized by several other articles.

2.1.2 Autoencoder

An Autoencoder (AE) is a type of feed-forward neural network which is used to learn an encoding of the data, i.e., to perform dimensionality reduction. Typically an AE has an encoding network, which may be composed by one or more layers (i.e., is an MLP), and a decoding network. The two are connected by an intermediate layer which acts as a bottleneck and will represent the data encoding. An AE learns this representation in an unsupervised way, the same data is used as both input and output, in order

Figure 2.1: *Architecture of the Neural Matrix Factorization algorithm, as described in the original article [57]*



for the AE to learn to encode it in the embedding and reconstruct from it, optimizing as loss the reconstruction error.

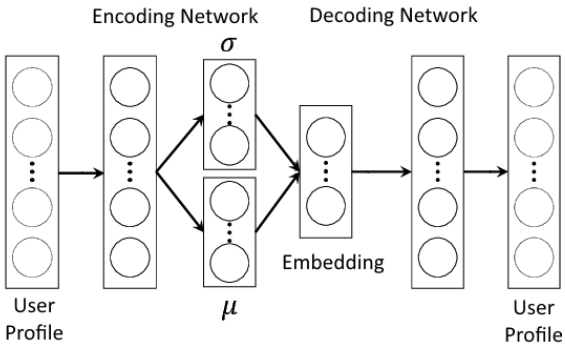
Several variants of AEs exist, the most frequently found in the recommender systems algorithms we analyze are denoising and variational [72, 73]:

Denoising Autoencoder (DAE): The purpose of a DAE is to learn a more robust representation of the data. This is done by corrupting the input adding a certain amount of random noise, while the output data is not altered. The AE is therefore trained to reconstruct the original non-corrupted data starting from a corrupted version. An example of DAE in the algorithms we analyze is Collaborative Denoising Autoencoder [122], the added noise is sampled from a Gaussian distribution.

Variational Autoencoder (VAE): The purpose of a VAE is to ensure the learned encoding space is continuous. This is done by forcing the autoencoder to learn a latent representation that follows a Gaussian distribution. From Figure 2.2 we can see that in addition to the encoding layer two other layers, with the same dimensionality, are introduced, one representing the averages and the other the standard deviations of the Gaussian distributions. In AEs the encoding space will have am-

ple regions the decoder has never observed during training, since the input data will be associated with only one encoding, which can limit the AE robustness to unseen data. VAEs, on the other hand, ensure the input data will have slightly different encodings if presented to the network multiple times, therefore the decoder will learn to decode a wider range of encodings. This allows to use the decoder as a generative model by providing variations of the latent features. In order to control the trade-off between learning a good encoding to reconstruct the input data and an encoding which follows the unit Gaussian distribution, the loss function is composed by two terms. The first is the usual reconstruction error, the second is the Kullback-Leibler divergence, which measures how closely the latent feature representation resembles a Gaussian distribution. The most competitive algorithm in our analysis, Mult-VAE, belongs to this category [73].

Figure 2.2: *Architecture of the Collaborative Variational Autoencoder algorithm, as described in the original article [72]*



When applied to the recommender system field, for the algorithms analyze in this paper, the model is trained to learn to encode and decode the whole user profile. Consider Figure 2.2, which represents a VAE. In that example, both the input and output layer will have a number of neurons which is equal to the total number of items. For recommender systems the AE can be trained using both implicit and explicit (i.e., rating) interactions. Due to the high sparsity of the interaction data the prevalence of zero values (i.e., missing interactions) can make the training difficult, among the proposed mitigation strategies is to sample only some of the negative items [122].

2.1.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of multilayer feed-forward neural network that was developed for certain types of inputs, i.e., images, that exhibit a semantically relevant topology [130]. In his foundational paper Le Cun et. al [68] stated that:

> A deficiency of fully-connected architectures is that the topology of the input is entirely ignored. The input variables can be presented in any (fixed) order without affecting the outcome of the training. On the contrary, images (or time-frequency representations of speech) have a strong 2D local structure: variables (or pixels) that are spatially or temporally nearby are highly correlated. Local correlations are the reasons for the well-known advantages of extracting and combining *local* features before recognizing spatial or temporal objects, because configurations of neighboring variables can be classified into a small number of categories (e.g. edges, corners...). *Convolutional Networks* force the extraction of local features by restricting the receptive fields of hidden units to be local.

This important concept was further stressed by the paper describing the widely known *AlexNet* for image classification [67], stating that CNNs are based on a very important assumption regarding the nature of the processed input data (i.e., the images), the *locality* of pixel dependencies. With data exhibiting these properties, local features (e.g., lines, corners) emerge from their respective immediate surroundings, regardless of their absolute position in the data. The importance of translation invariance and feature locality for CNNs has been widely discussed, both in terms of *spatial locality* [69, 77, 112, 128], and *time locality* for sequence modeling [5].

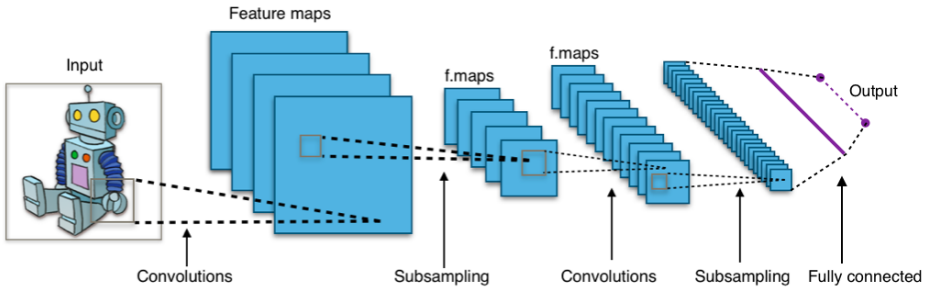
The architecture of a CNN usually consists of a convolution layer and a subsampling layer. Figure 2.3 shows an example of CNN; we can see that multiple blocks of convolution and subsampling layers are concatenated.

The convolution layer uses a kernel, with certain weights, which is moved across the two dimensional feature matrix (i.e., the image). By sharing the kernel weight's across the image a CNN can have fewer parameters than a fully connected neural network and can leverage the spatial locality and location invariance of features. The subsampling layer (e.g., max pooling[1]) reduces the dimensionality of the data and allows successive convolution layers a broader field of view. Successive convolutional layers

[1]The use of max-pooling to reduce the dimensionality of the data has been recently criticized, though [100]

are able to interpret increasingly complex patterns by further aggregating lower-level features. Again it was stressed by [68] that after a set of features has been detected, only the reciprocal relative positions are relevant, not the absolute ones.

Figure 2.3: *Architecture of a Convolutional Neural Network*



The property of feature locality implicitly depends on a definition of *proximity* between points. If we consider image data, pixels that are close in the image will be perceived as close by an observer and are therefore meaningful for the reconstruction of more complex patterns, therefore the proximity in images is defined in spacial terms according to the 2D structure of the image itself. Depending on the use case, different definitions of proximity may be used, e.g., for non-Euclidean data like social networks or knowledge graphs [37].

Following its success in computer vision tasks, some articles propose to use CNNs for collaborative filtering as well. Most of the existing approaches used for the traditional *top-k* recommendation problem can be grouped into three categories [132]:

Feature extraction: In this case, a CNN is used to extract features from heterogeneous data sources, e.g., images, video, audio, which are then used in another recommendation model [114].

Pretrained embeddings: In such approaches, a CNN is applied on user or item embeddings that were pretrained by another model [56, 123].

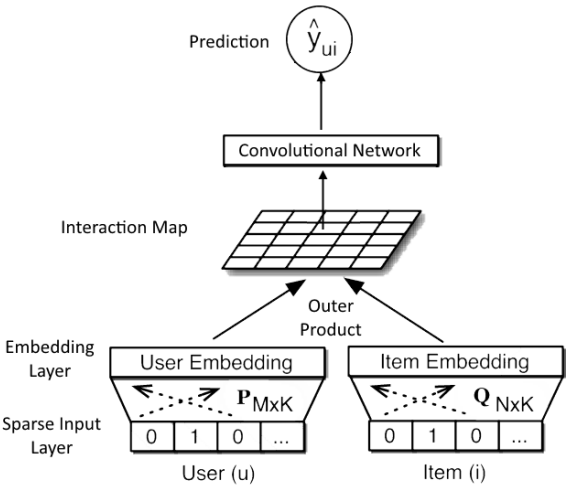
Learnable embeddings: Also in this case the CNN is applied on user or item embeddings. Differently from the previous case, the embeddings are an integral part of the model and are trained along with the CNN (i.e., they are not pre-trained with another approach) [64, 131].

In this thesis we only analyze algorithms belonging to the last two cases, when CNNs are applied on embeddings. CNNs have also been applied to

session-based recommendations [129], and to sequential recommendations [126].

As an example of CNN applied to the traditional *top-k* recommendation problem we briefly describe one of the algorithms we analyze in this study. Figure 2.4 shows the architecture of the ConvNCF algorithm [56].

Figure 2.4: *Architecture of the Convolutional Neural Collaborative Filtering, as described in the original article [56]*



We can see that the network takes as input the one-hot encoding of the user and item and learns to predict the rating associated to that interaction. Via two fully connected layers, the one-hot encoding of users and items is transformed into embeddings. Those embedding vectors are then multiplied via an outer product resulting in a square matrix which is called *interaction map*. It is on that interaction map that the CNN is applied. While in the original articles it is argued that this interaction map is analogous to an image and therefore the use of CNN is justified, this claim is not demonstrated or discussed in detail. In Chapter 6 we will discuss the claimed results and demonstrate they are not correct because they are based on a flawed methodology.

2.2 Reproducibility Crisis

The independent reproducibility of experimental results has been for centuries a cornerstone of the scientific development. Its importance is such that as far back as 1637 Descartes, in *Discourse on the Method of Rightly Conducting One's Reason and of Seeking Truth in the Sciences*, described it

as an essential component of the scientific method. Usually, only after several different researchers have been able to independently reproduce and verify the experiments and conclusions reported in an article, that becomes an accepted scientific fact.

In the last decade evidence started to mount that the scientific world faces a significant *reproducibility crisis* spanning widely across several research communities, e.g., psychology, medicine, chemistry, as well as several fields of computer science. In cancer biology a landmark study published in 2012 found as little as 11% of the reported results could be reproduced [8]. Some fields, like psychology, are dealing with this problem by shifting their community culture to favor replication, and computer science is slowly starting to do the same [62].

While warnings had been published many years ago [4], only in recent years the computer science research community is starting to grapple with the extent and implications of the problem. A significant quote from 2008 [40]:

> The prevalence of very relaxed attitudes about communicating experimental details and validating results is causing a large and growing credibility gap. It's impossible to verify most of the results that computational scientists present at conferences and in papers.

Due to some ambiguities in the use of the terms reproducibility and replicability [54], which we found are sometimes swapped, we first report the definition that we will use in this study, as provided by the American Statistical Association [19][2]:

Reproducibility The numerical findings reported in the study can be reproduced by using the original artifacts (i.e., data and source code).

Replicability The numerical findings reported in the study can be obtained by repeating the entire study independently, following the described methodology, without using any of the original artifacts.

One of the issues to overcome in the field of computer science is that while researchers agree the reproducibility of empirical results is important, what reproducibility means is not agreed upon [54]. Establishing reproducibility in computer science requires to address several important peculiarities of this field that will have an impact on the reproducibility of

[2]Recent SIGIR guidelines swap the definitions of reproducibility and replicability: `https://sigir.org/wp-content/uploads/2018/07/p004.pdf`, see also ACM Artifact Review and Badging: `https://www.acm.org/publications/policies/artifact-review-badging`.

the experiments. It is known, for example, that several AI algorithms are very sensitive even to hyperparameters that do not appear to be the most important ones or may even produce substantially different results according to the random number generator or initial seed [62]. Furthermore, it can easily happen that even with identical settings and software, random seed included, the algorithm will not produce the same results on a different machine, e.g., due to differences in the hardware impacting the model training process.

In recent years several studies have tried to assess the level of reproducibility in computer science and machine learning research. Gundersen et. al in [54] attempted to quantify the level of reproducibility for empirical research in AI by defining and reporting six reproducibility metrics for a total of 400 research papers accepted at two different conference series between 2013 and 2016. Their findings are that AI research is not sufficiently documented to allow the reproducibility of published results, but also that the documentation practices have been improving with time. A strong point made in the article is that the inability to reproduce published results impacts the trustworthiness of science. In order to ensure the trustworthiness of AI and machine learning, publications steps must be taken to improve its currently low reproducibility. A follow-up study [53] compared articles written by academics and industry practitioners showing that industry papers tend to show worse results in terms of the quality of the documentation but no overall statistically significant difference in reproducibility could be observed.

Similar results have been reported in several other studies. Collberg et. al [26] examined 600 articles published in ACM conferences and journals between 2011 and 2013, for each attempting to locate a publicly available implementation or to implement the algorithm themselves, emailing the authors if their attempts were unsuccessful. In their study Collberg et. al did not attempt to verify the published results based on the provided implementation. Results show that only 42% of the papers could be reproduced and only 60% of emails received a reply.

Vandewalle et. al [115] studied all 134 papers that were published in the 2004 edition of *IEEE Transactions on Image Processing*. In their study two or three reviewers per paper were asked to check its reproducibility answering to a short list of questions mainly focused on the availability of the artifacts and of details on the experimental setting, hyperparameters and pseudocode. In their study therefore they did not try to reimplement the algorithms. Vandewalle et. al could observe that in more than 80% of cases the algorithm and the data were described with sufficient detail, however in

in only 33% of cases the data was publicly available and in less than 10% of cases the source code was as well. The reason for the greater availability of data with respect to the source code was the use of some standard image datasets. Among the issues limiting the availability of source code and data is the short lifetime of web pages, since when a researcher retires, moves to another institution, or when the institution or lab websites are updated or redesigned, the link reported in old articles may become invalid. Among possible solutions are proposed the use of DOIs or institutional repositories.

The limited availability of source code for computational science was further investigated by Stodden [107], who submitted a survey to authors of articles published between 2008 and 2010 in the *Neural Information Processing Systems* conference series. While most of the authors stated they were willing to share the original artifacts, only 30% in the end did publicly share the source code. The most frequent reasons for not sharing the artifacts were: the time required to prepare and document the publicly available version, the need to provide assistance to researchers attempting to use the source code, the possible use of the public implementation without citation.

In a 2019 study, Raff [94] attempted to assess the correlation of reproducibility with other factors, e.g., year of publication, number of equations, availability of pseudocode, number of authors. He found the year of publication does not significantly affect reproducibility so, if a crisis exists, it has been going on for decades. An interesting finding is that Raff actually found the number of equations per page to be negatively correlated with the paper reproducibility. The authors do not provide a definitive explanation for the phenomena but suggest two possibilities, the higher number of equations may make the paper less readable and impede its reproducibility, or it may just be a manifestation of the higher complexity of certain methods which are more difficult to use.

While most of the previous studies have focused on the pure reproducibility of the results given the provided artifacts or on the replicability using publicly available datasets, another important dimension to consider is the methodological soundness of the evaluation protocol itself. A very recent study by Christodoulou et. al [24] compared the performance of logistic regression against machine learning methods for clinical prediction modeling. The authors argue that previous studies comparing clinical prediction models based on logistic regression and machine learning algorithms suffered from methodological issues. In particular the reporting of the methodology itself was incomplete and the validation procedure poor. In the majority of the studies methodological issues were found e.g., optimization of the hyperparameters on the whole dataset (i.e., therefore in-

cluding test data) or incorrect use for resampling on the data for validation.

An older study by Hand [55] for supervised classification, published in 2006, argued that numerous articles proposing complex algorithms that are shown to outperform simpler ones often do not take into account aspects of the real scenarios (i.e., the data used for training may not have the same distribution as the real one which the classifier will be used on, the classes definition may involve a level of uncertainty or arbitrariness) resulting in a spurious evaluation which may overestimate the new model performance. It is argued that those complex methods often yield to comparative performance with respect to simpler ones and that this amplifies the impact of uncertainty on the results, something which is typically not taken into account, producing results which are not reliable or realistic.

More specifically within the machine learning community Sculley et al. [105] observed how the current research culture poses too much emphasis on demonstrating a newly proposed method outperforms some baselines but does not pay as much attention on the empirical rigor of the evaluation procedure. Due to this, as several previous studies point out, gains of new and more complex methods can be due to factors other than their better modeling capacity, e.g., better hyperparameter optimization. Their findings are cited by Lin [74] who observes similar issues in the Information Retrieval domain, arguing that papers claiming to improve over the state-of-the-art do so by comparing the new approach against weak or insufficiently optimized baselines. In a later article Lin [75], although recognizing the significant progress achieved by neural methods in language modeling tasks, still stressed the issue of inadequately tuned baselines. More than a decade ago this very same observation was made in the field of Information Retrieval by Armstrong et al. in 2009 [4], who observed how the selection of weak baselines could create over time an illusion of continuous incremental improvement. In particular, Armstrong reached a very strong conclusion:

> There is, in short, no evidence that ad-hoc retrieval technology has improved during the past decade or more.

It is remarkable that ten years later Yang et al. [127] reached the very same conclusion stating:

> We do not find evidence of an upward trend in effectiveness over time. In fact, the best reported results are from a decade ago and no recent neural approach comes close.

The tendency to rely on weak evaluation protocols will create the illusion of continuous progress. Similar findings have also been reported for

the recommender systems field. For the task of session-based recommendation it could be shown that a simple neighborhood-based is able to compete if not outperform some recent neural algorithms [79, 80].

Several examples of this can also be found in other scientific communities, for example Makridakis et al. [81] recently showed that for the task of time series forecasting old statistical methods were able to outperform much more recent machine learning ones.

For the task of rating prediction Rendle et al. [97] too observed the reported results for the baselines on a widely known dataset are often suboptimal and that a carefully tuned baseline method is able to outperform all newly proposed algorithms from 2015 to 2019. Rendle et al. observe these results are particularly worrisome because the dataset is widely used in the community. If it was possible to report suboptimal baselines for a widely known dataset, other evaluations executed on less known datasets are even more prone to unreliable results. These results are attributed to the fact that in the community the difficulty of running baselines is mostly ignored and the community lacks common benchmarks.

While a poor experimental methodology is not strictly speaking a reproducibility issue when the results of the paper can be reproduced, it is still a sign that the research field is unaccustomed to the independent verification of published research and evaluation protocols. It would therefore seem that the Information Retrieval field is in the same position as the Artificial Intelligence field as pointed out by Raff [94], if a crisis exists, it has been going on for decades.

CHAPTER 3

Identifying relevant and reproducible articles

In this Chapter we describe the criteria followed to select the articles analyzed in this study as well as the specific requirements for reproducibility.

3.1 Paper Selection Criteria

The articles analyzed in this thesis were selected following a systematic approach. We define two categories of articles: *relevant*, if they are within the scope of this study, and *reproducible*, if we were able to reproduce the experimental setting and run the experiments.[1] Note that, in order to provide an evaluation as conservative as possible, we adopt a rather strict definition of reproducibility as will be described later.

More precisely, we consider an article to be *relevant* if it meets the following constraints:

[1] This study combines aspects of reproducibility (we rely on the original source code and data) and of replicability (if a data split is not available we generate a new one, if the original evaluation protocol contains an error or information leakage we correct it).

- The paper has been published between 2015 and 2018[2] in the proceedings of one of the following conferences: SIGIR, KDD, TheWebConf (WWW), IJCAI, WSDM, RecSys. We do not include workshop papers. All these conferences are common venues for recommender systems research and are, except RecSys, classified as A* (i.e., *flagship conference, a leading venue in a discipline area*) in the Australian CORE ranking system[3]. RecSys is instead classified as B (i.e., *good conference, and well regarded in a discipline area*), and was included because it is entirely devoted to recommender systems research. The articles have been selected manually from the conference proceedings.
- The paper is proposing a new deep learning algorithm for the classic *top-k* recommendation task. We did not consider articles targeting other scenarios like group recommendation as well as articles using deep learning to perform feature extraction from non structured data like text, audio or images. Furthermore, the experimental evaluation should include at least one classification accuracy or ranking accuracy metric. We did not consider articles evaluated only with rating prediction metrics.

Among the *relevant* articles, we consider an article to be *reproducible* if we can reproduce the experimental setup, a precondition to run the experiments. The following requirements should be met:

- The source code written by the original authors is available, requiring at most minimal modifications in order to execute correctly, which means it should include the model itself, its training and a way to compute the predictions. When applying modification ourselves the core algorithm was never altered. While it is expected that a paper should contain all necessary details to re-implement the proposed model, many details that can have an impact on the final results may not be described explicitly due to space reasons (e.g., initialization, regularization, adaptive gradients, train batches)[4]. As in Collberg et. al [26] we did not include articles having provided only a skeleton version of the source code or articles whose source code did not execute correctly or was missing dependencies we could not resolve. Furthermore, we did not include articles for which only a third-party implementation was available.

[2]The analysis did not extend before 2015 because the number of relevant articles, in the analyzed conferences, decreased rapidly and amounted to only two. The analysis did not extend after 2018 as it was the most recent year for which all conferences had already taken place.

[3]http://www.core.edu.au/conference-portal

[4]Similar observations were put forward, for example, in [102].

- At least one of the datasets used for the evaluation in the original paper is publicly available. It should be freely accessible either the training-test splits used by the original authors or the article should contain sufficient detail to reproduce the preprocessing and data split procedure. We did not include articles using datasets that were crawled on websites, since, although we could have crawled the data again ourselves we could not guarantee the data had remained the same. We also did not include articles evaluated only on datasets that could not be made public due to non disclosure agreements or copyright issues.

If the constraints we have defined for a paper to be *reproducible* were not met, we contacted all the authors by email. We waited a maximum of 30 days for a reply. If we did not receive a reply within that time or we did receive a reply but we could not solve the issue, the paper was considered *not reproducible*.

3.2 Relevant and Reproducible Papers List

Following the described methodology, we were able to identify 24 articles, out of which 12 could be reproduced. Table 3.1 reports the statistics per conference. Bearing in mind that the number of papers is too small for reliable statistics, we still can see differences across conferences. In terms of the ratio of reproducible articles over relevant articles, KDD and IJCAI have the best positions. In terms of the number of relevant articles IJCAI and RecSys are at the top of the list, however, while IJCAI has a high reproducibility ratio, RecSys has the second lowest, with one reproducible paper only. We could observe an increase in the quota of reproducible articles in more recent years, this both related to the increased tendency to provide the source code publicly, usually via GitHub, and partially to the higher probability of receiving a reply from the authors.

Conference	Rep. ratio	Reproducible setup	Not Reproducible
KDD	3/4 (75%)	[60], [72], [119]	[111]
IJCAI	5/7 (71%)	[56], [133], [131], [23], [125]	[86], [124]
WWW	2/4 (50%)	[57], [73]	[110], [43]
SIGIR	1/3 (30%)	[42]	[83], [21]
RecSys	1/5 (20%)	[134]	[108], [16], [101], [116]
WSDM	0/1 (0%)		[122]
Total	12/24 (50%)		

Table 3.1: *Statistics of relevant and reproducible papers on deep learning algorithms for* top-k *recommendation per conference series from 2015 to 2018.*

We provide the detailed list of each relevant paper, divided per conference and ordered by year, in Table 3.2 (SIGIR, WWW), 3.3 (RecSys, KDD) and 3.4 (WSDM, IJCAI). If the article has been classified as reproducible, a link to the GitHub repository with the source code is provided, when available. If the article is not reproducible, the motivation is provided. The increasing popularity of deep learning is clearly visible observing that the number of relevant papers published in 2018 is greater than the number of those published in the previous three years combined, between 2015 and 2017.

	SIGIR	
2018	**Collaborative Memory Network for Recommendation Systems** Reproducible: Yes[a]	[42]
2018	**A Contextual Attention Recurrent Architecture for Context-Aware Venue Recommendation** Reproducible: No. Motivation: The source code is publicly available[b] but was missing the preprocessing code and contains a bug preventing it to execute correctly. We received no reply.	[83]
2017	**Attentive Collaborative Filtering: Multimedia Recommendation with Item and Component-Level Attention** Reproducible: No. Motivation: The source code is publicly available[c] but the data is not due to copyright reasons.	[21]
	WWW	
2018	**Variational Autoencoders for Collaborative Filtering** Reproducible: Yes[d]	[73]
2018	**Latent Relational Metric Learning via Memory-based Attention for Collaborative Ranking** Reproducible: No Motivation: The source code is publicly available[e] but contains only a skeleton of the model and no documentation. We received no reply.	[110]
2017	**Neural collaborative filtering** Reproducible: Yes[f]	[57]
2015	**A Multi-View Deep Learning Approach for User Modeling in Recommendation Systems** Reproducible: No. Motivation: The source code is not available. We received no reply.	[43]

[a] https://github.com/tebesu/CollaborativeMemoryNetwork
[b] https://github.com/feay1234/CARA
[c] https://github.com/ChenJingyuan91/ACF
[d] https://github.com/dawenl/vae_cf
[e] https://github.com/vanzytay/WWW2018_LRML
[f] https://github.com/hexiangnan/neural_collaborative_filtering

Table 3.2: *Relevant papers for SIGIR and WWW conferences.*

	RecSys	
2018	**Spectral Collaborative Filtering** Reproducible: Yes.[a]	[134]
2018	**Recurrent Knowledge Graph Embedding for Effective Recommendation** Reproducible: No. Motivation: The source code is not available. We received no reply.	[108]
2018	**RecGAN: Recurrent Generative Adversarial Networks for Recommendation Systems** Reproducible: No. Motivation: The source code is not available. We were informed that the source code could not be shared.	[16]
2018	**Attentive Neural Architecture Incorporating Song Features for Music Recommendation** Reproducible: No. Motivation: The source code is not available. We were informed that the source code and data have been deleted.	[101]
2016	**Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation** Reproducible: No. Motivation: The source code is not available. We received no reply.	[116]
	KDD	
2018	**Leverage Meta-path based Context for Top-N Recommendation with a Neural Co-Attention Model** Reproducible: Yes[b][c]	[60]
2018	**Multi-Pointer Co-Attention Networks for Recommendation** Reproducible: No Motivation: The source code is publicly available[d] but could not be used due to missing dependencies. We were informed that a working version of the source code could not be provided.	[111]
2017	**Collaborative Variational Autoencoder for Recommender Systems** Reproducible: Yes[e]	[72]
2015	**Collaborative Deep Learning for Recommender Systems** Reproducible: Yes[f]	[119]

[a] https://github.com/lzheng21/SpectralCF
[b] https://github.com/librahu/MCRec
[c] https://github.com/librahu/Dataset-In-Papers
[d] https://github.com/vanzytay/KDD2018_MPCN
[e] https://github.com/eelxpeng/CollaborativeVAE
[f] https://github.com/js05212/CDL

Table 3.3: *Relevant papers for RecSys and KDD conferences.*

	WSDM	
2016	**Collaborative Denoising Auto-Encoders for Top-N Recommender Systems** Reproducible: No. Motivation: The source code is publicly available[a] but could not be used due to lack of documentation. We received no reply.	[122]
	IJCAI	
2018	**Outer Product-based Neural Collaborative Filtering** Reproducible: Yes[b]	[56]
2018	**NeuRec: On Nonlinear Transformation for Personalized Ranking** Reproducible: Yes[c]	[133]
2018	**CoupledCF: Learning Explicit and Implicit User-item Couplings in Recommendation for Deep Collaborative Filtering** Reproducible: Yes[d]	[131]
2018	**DELF: A Dual-Embedding based Deep Latent Factor Model for Recommendation** Reproducible: Yes[e]	[23]
2018	**NPE: Neural Personalized Embedding for Collaborative Filtering** Reproducible: No. Motivation: The source code is not available. We were informed that the source code could not be provided at that time.	[86]
2017	**Deep Matrix Factorization Models for Recommender Systems** Reproducible: Yes[f]	[125]
2017	**Tag-Aware Personalized Recommendation Using a Hybrid Deep Model** Reproducible: No. Motivation: The source code is not available. We received no reply.	[124]

[a] `https://github.com/jasonyaw/CDAE`
[b] `https://github.com/duxy-me/ConvNCF`
[c] `https://github.com/cheungdaven/NeuRec`
[d] The source code was not publicly available but the authors shared it with us upon request.
[e] The source code was not publicly available but the authors shared it with us upon request.
[f] The source code was not publicly available but the authors shared it with us upon request.

Table 3.4: *Relevant papers for WSDM and IJCAI conferences.*

CHAPTER 4

Evaluation protocol and baseline algorithms

In this Chapter we describe the experimental protocol used in the evaluation of the reproducible neural algorithms as well as all baseline algorithms and their hyperparameter optimization step. We further detail all classification and ranking accuracy metrics as well as the beyond-accuracy metrics we report.

4.1 Experimental Setup

For all the *reproducible* articles we compare the proposed neural model to a set of simple baselines in order to assess whether it is competitive against them or not in specific evaluation protocols. This comparison allows us to quantify the progress that complex neural models allowed to make versus simpler and older techniques. The experimental procedure is as follows:

- We build a wrapper around the original implementation of the neural algorithm. We keep the original code to build the model, training it and computing the item scores (i.e., predictions). The item scores are then passed to our own evaluation code which will sort them to

compute the final recommendation list and then the various evaluation metrics, in such a way to guarantee all algorithms are evaluated in exactly the same way. This allows us to have full control over the experimental conditions and ensure that differences in the recommendation quality are imputable to the recommender model alone and not to other factors like how the metrics are computed, what sorting algorithm is used etc.[1]

- We use the original training-test split, if available. If not, we get the original version of the dataset and apply the preprocessing procedure and split as described in the original paper. We further split the training data in order to obtain validation data to be used for the hyperparameter tuning of the baselines and the selection of the number of training epochs. If the original paper describes how the validation data is split, we apply that procedure, otherwise we apply the same split used to create the test data.

- For each dataset we run a Bayesian hyperparameter tuning of our baselines, selecting the hyperparameters optimizing the recommendation quality on the validation data. In order to select the evaluation metric and cutoff to optimize, we used the information reported in the original paper, if available. If not, we used the metric optimized in the original source code (i.e., the one used by the original authors to select the number of training epochs of their algorithm). If none of them is available, we select one of the ranking metrics reported in the original paper and as cutoff we select an intermediate value between the lowest and highest reported in the original results. Once the optimal hyperparameters are selected, each baseline is trained using them on the full training data (i.e., the union of the smaller train and validation data). The model obtained in this way is then evaluated on the test data under the original experimental conditions and methodology.

- The neural algorithm is trained on the training data using the same hyperparameters as reported in the original article. If a hyperparameter value is not described in the article we use the value reported in the original source code but ask the authors for confirmation. Using the original hyperparameters is appropriate in this case because we are evaluating the model in exactly the same experimental conditions. The

[1] Although the details of the training (e.g., adaptive gradient, batches, regularization etc) and evaluation protocol are often overlooked in the articles, possibly due to space reasons, it is known that they can have a significant impact on the numerical results. For example in [103] it was shown that even under the same evaluation protocol the same algorithm, in different evaluation frameworks, produces quite different results.

number of epochs and the stopping criteria are usually not reported in the articles, due to this we always select the number of epochs via early stopping on the same validation data used to optimize the baselines hyperparameters (see Section 4.4.1). Once the number of epochs has been selected, the neural model is trained again on the full training data for that number of epochs and then evaluated on the test data.

4.2 Baseline Algorithms

Since the very first recommender systems related paper published in the mid-nineties [98], a multitude of different algorithms have been proposed as well as strategies to improve them. In order for our experimental evaluation to provide a broad picture, we selected several simple algorithms of different families, some developed more than a decade ago, others more recently. An overview of all the baselines we use is given in Table 4.1 and the relative hyperparameters, their ranges and distributions are reported in Appendix A.

4.2.1 Popularity-Based Ranking

Recommending the most popular items to everyone, without any personalization, is a common strategy in practice and in certain scenarios proves to be quite effective. The method **TopPopular** implements this recommendation approach. The popularity of an item is defined as the number of interactions (i.e., explicit ratings or implicit interactions) the item has received in the given dataset. The items are recommended in order of decreasing popularity. While this algorithm always ranks the items in the same order, in all our experiments items already seen by the user will not be recommended again to them, so different users may still be provided with different recommendation lists

4.2.2 Nearest-Neighbor Methods

Nearest-neighbor techniques were among the very first collaborative filtering models to be developed [76], dating back to the early GroupLens system in 1994 [98]. This family of methods is based on a *similarity* between users or items computed with a heuristic. The general idea is as follows:

- **User-based (UserKNN)** users that behaved similarly in the past, interacting with the same items, are similar.

Family	*Method and Description*
Non-personalised	**TopPopular** Recommends the most popular items to everyone [33]
Nearest-Neighbor	**UserKNN** User-based k-nearest neighbors [98]
	ItemKNN Item-based k-nearest neighbors [104]
Graph-based	**P$^3\alpha$** A graph-based method based on random walks [27]
	RP$^3\beta$ An extension of P$^3\alpha$ [90]
Content-Based and Hybrid	**ItemKNN-CBF** ItemKNN with content-based similarity [78]
	ItemKNN-CFCBF A simple item-based hybrid CBF/CF approach [85]
	UserKNN-CBF UserKNN with content-based similarity
	UserKNN-CFCBF A simple user-based hybrid CBF/CF approach
Non-Neural Machine Learning Item-based	**SLIM ElasticNet** A scalable linear model [71, 87]
	SLIM BPR A variation of SLIM optimizing ranking [6]
	EASE^R A recent linear model, similar to auto-encoders [106]
Non-Neural Machine Learning Matrix Factorization	**iALS** Matrix factorization for implicit feedback data [61]
	PureSVD A basic matrix factorization method [33]
	NFM A basic non-negative matrix factorization method [25]
	FunkSVD Matrix factorization for rating prediction [70]
	MF BPR Matrix factorization optimized for ranking [96]

Table 4.1: *Overview of Baseline Methods*

- **Item-based (ItemKNN)** items that received interactions from the same users in the past are similar.

Conventionally, the similarities are represented as a square similarity

matrix S in which only the k most similar items or users are included. In the traditional formulation, the dot product between the user profiles and the similarity matrix is used to compute the model's predicted ratings (or more generally item scores). More precisely, the score of an item i for user u sill be computed as follows. In case of an ItemKNN model:

$$\hat{R}_{u,i} = \sum_{j \in KNN(i)} R_{u,j} \cdot S_{i,j}$$

In case of a UserKNN model:

$$\hat{R}_{u,i} = \sum_{t \in KNN(u)} R_{t,i} \cdot S_{u,t}$$

Where $KNN(\cdot)$ is a function that returns the most similar entities (i.e., items or users) to the one provided as input.

Despite the simple linear nature of this family of methods, several hyperparameters can be selected and variants applied [18]. In this work we will consider some of this variants and define a set of hyperparameters for both UserKNN and ItemKNN. The full list of hyperparameters we optimized in our experiments and their ranges are reported in Appendix A.

- *Neighborhood Size*: This is one of the most important parameters and determines how many neighbors are considered for prediction. If the number of neighbors is too small the model will have limited modeling capacity and only look for the very most similar users or items, on the other hand a high number of neighbors may improve the recommendation quality to an extent while adding increasing noise.

- *Similarity Heuristic*: This is another crucial parameter. There are a great many different heuristics that have been proposed as ways to measure the similarity. In our experiments we selected: Jaccard coefficient [93], Cosine [104], Asymmetric Cosine [2], Dice-Sørensen [39] and Tversky [113] similarities which will be described shortly. Some of these similarity measures also have their own parameters, as reported in Appendix A, which we optimized as well.

- *Shrinkage*: A commonly used strategy to increase the support of the computed similarity is the addition of a *shrink term* (i.e., a constant term) to the denominator of the similarity heuristic. This lowers the similarity between items or users that have only very few interactions, as proposed in [9]. The shrinkage is applied to all similarities.

- *Feature Weighting*: Applying feature weighting to ratings was proposed years ago in [120]. Feature weighting is more traditionally applied on features rather than collaborative data, however it can be applied both to items and users, allowing to weight the importance of them to compute the final model. In our experiments we tested three configurations: no weighting, TF-IDF or BM25. While several other weighting strategies exist, those are among the most commonly used in recommender systems.

- *Normalization*: Similarity heuristics have a denominator which is used to normalize the value obtained by the dot product, this setting determines if we should use the denominator or not. Only some of the similarity measures have this parameter.

Similarity Heuristic

In this Section all similarity heuristics will be described for collaborative data, but the same heuristics are applied seamlessly to feature data. Vectors $\vec{r}_i, \vec{r}_j \in \mathbb{R}^{|U|}$ represent the ratings of a user for items i and j, respectively, and $|U|$ is the number of users. In case of CBF recommender vectors $\vec{r}_i, \vec{r}_j \in \mathbb{R}^{|F|}$ would describe the features of items i and j, respectively, and $|F|$ is the number of features. Parameter h is the *shrink term*.

Cosine: The most frequently used similarity metric [104] is computed as the normalized dot product of the vectors.

$$s_{ij} = \frac{\vec{r}_i \cdot \vec{r}_j}{\|\vec{r}_i\|\|\vec{r}_j\| + h}$$

Asymmetric Cosine: This parameterized variant of the cosine similarity was described in [2], by the winning team of the Million Songs Dataset (MSD) challenge.[2] The paper shows that the Cosine similarity can be represented as a product of the square roots of two conditional probabilities, therefore attributing to the asymmetric cosine a probabilistic interpretation.

$$s_{ij} = \frac{\vec{r}_i \cdot \vec{r}_j}{\|\vec{r}_i\|^{\alpha}\|\vec{r}_j\|^{1-\alpha} + h}$$

Dice - Sørensen: Is a similarity measure defined on sets, therefore on boolean vectors, described in [39]. Given two sets or vectors the respective

[2] https://www.kaggle.com/c/msdchallenge

Dice similarity can be computed as a linear function of Jaccard similarity and vice versa. The Dice coefficient does not satisfy the triangle inequality, it can be considered a semimetric version of the Jaccard index. Equation 4.1 represents the Dice similarity computed on sets, while Equation 4.2 on boolean vectors.

$$s_{AB} = 2\frac{A \cap B}{\|A\| + \|B\| + h} \tag{4.1}$$

$$s_{ij} = 2\frac{\vec{r}_i \cdot \vec{r}_j}{\|\vec{r}_i\| + \|\vec{r}_j\| + h} \tag{4.2}$$

Jaccard - Tanimoto: Is a similarity measure on sets, therefore defined on boolean vectors, described in [93]. The term Tanimoto is sometimes used to refer to its formulation on vectors, however the corresponding similarity is not a proper similarity since its distance does not preserve the triangle inequality. Equation 4.3 represents the Jaccard similarity computed on sets, while Equation 4.4 on boolean vectors.

$$s_{AB} = \frac{A \cap B}{\|A\| + \|B\| - \|A \cap B\| + h} \tag{4.3}$$

$$s_{ij} = \frac{\vec{r}_i \cdot \vec{r}_j}{\|\vec{r}_i\| + \|\vec{r}_j\| - \vec{r}_i \cdot \vec{r}_j + h} \tag{4.4}$$

Tversky: Is an asymmetric similarity measure on sets, generalising Dice-Sørensen and Tanimoto-Jaccard coefficients, described in [113]. If $\alpha = \beta = 1$ Tversky is equal to the Tanimoto coefficient, if $\alpha = \beta = 0.5$ Tversky is equal to the Dice coefficient.

$$s_{AB} = \frac{A \cap B}{\|A \cap B\| + \alpha\|A - B\| + \beta\|B - A\| + h}$$

Where $-$ denotes the *relative complement* of two sets, therefore $\|B - A\|$ means the number of elements in B that are not in A and can be represented as $\|B\| - \|A \cap B\|$. In case of boolean vectors the Tversky similarity can be calculated as follows:

$$s_{ij} = \frac{\vec{r}_i \cdot \vec{r}_j}{\alpha\|\vec{r}_i\| + \beta\|\vec{r}_j\| + (1 - \alpha - \beta) \cdot \vec{r}_i \cdot \vec{r}_j + h}$$

While all these similarity metrics share some common characteristics (i.e., they are all based upon the dot product of the vectors) they also have some important differences. Namely, Asymmetric Cosine allows to take into account the different probability that a profile will have something in common with another. Think about popular items, they will be associated to significantly more interactions than unpopular items, therefore they will have interactions in common with other items with higher probability. Unpopular items, on the other hand, will tend to have fewer interactions in common. If we compare the set based similarities we can see how Dice normalizes the dot product depending on how many interactions the two profiles cumulatively have, while Jaccard only considers the *unique* ones. Tversky further increases the complexity taking into account with different weights the features that are *not* in common.

4.2.3 Graph-based Methods

Nearest-neighbor models, in their traditional form, compute similarities between pairs of items or users, therefore they consider only direct neighborhoods. Graph-based models rely on a broader definition which takes into account non-direct neighbors as well. Graph-based recommender systems usually define the user-item interaction data as an undirected bipartite graph, where users and items that are associated to an interaction are connected, but no direct connection exists between users or between items. The recommendation process is defined via a random walk on this graph. A traditional KNN would only consider direct user-item edges, while a graph based model will also consider indirect connections following longer paths, for example user-item-user-item. It is worth noting that a graph based model can be represented as a similarity based recommender, as a KNN would, by defining the similarity between two items as the transition probability of a random walk starting from one to reach the other. This formulation, which is the one we adopt, has the effect of removing the need to simulate a random walk resulting in a deterministic model, and has the same computation complexity of a KNN model.

In our study, we consider two graph-based methods called $\mathrm{P}^3\alpha$ [27] and $\mathrm{RP}^3\beta$ [90]. Both methods tend to lead to very competitive recommendation quality at a low computational cost, comparable to the traditional KNN models, and have been published in top-tier venues. Despite this, surprisingly, both methods are very rarely used as baselines in the literature and have a limited number of citations.

- ***P³α***: This method was proposed by [27] and implements a two-step

random walk from users to items and vice-versa. The probabilities to jump from a user to an item are computed from the normalized ratings raised to the power of α. As previously mentioned, $P^3\alpha$ is equivalent to a KNN item-based algorithm, with the similarity matrix being computed as the dot product of the probability vectors. Due to this, this method hyperparameters also include the size of the neighborhood. In order to ensure that the similarity row, after the selection of the neighborhood represents a probability distribution, we normalize each row of the similarity matrix with its $l1$ norm.

- ***RP$^3\beta$***: This is an improved version of $P^3\alpha$, proposed in [90]. Observing that $P^3\alpha$ tends to exhibit a high popularity bias, $RP^3\beta$ ties to mitigate this imbalance by dividing the similarity by the popularity of the items raised to the power of β. Again, we subsequently select the item neighborhood and then normalize each row of the similarity matrix with its $l1$ norm. If β is 0, $RP^3\beta$ is equivalent to $P^3\alpha$. The hyperparameters of the algorithm are the size of the neighborhood and the values for α and β.

4.2.4 Content-based and Hybrid Methods

All of the previously described baselines are pure collaborative models, therefore they only rely on user-item interaction data. Some of the neural models we analyze in this paper include side information about items or users, which we will broadly refer to as features. Due to this we also include some content-based and hybrid baselines. Those baselines are KNNs and can be applied on feature data seamlessly as their pure collaborative counterparts, and they share the same hyperparameters.

- **ItemKNN-CBF**, **UserKNN-CBF**: A neighborhood-based content-based-filtering (CBF) approach. The item (or user) similarities are computed based on the items' (or users') content feature vectors (attributes) instead of the interaction vectors [78]. We tested the same set of similarity measures used for the collaborative KNN methods (Jaccard coefficient, Cosine, Asymmetric Cosine, Dice-Sørensen and Tversky similarity). The hyperparameters are the same as for the ItemKNN and UserKNN methods.

- **ItemKNN-CFCBF**, **UserKNN-CFCBF**: A simple hybrid algorithm based on item-item (or user-user) similarities and described in [85]. While in case of the pure collaborative versions the similarity is computed using only the interaction vectors and in the CBF version the

similarity is computed using only the feature vectors, in this hybrid version both vectors are used by concatenating them. The similarity is computed, therefore, using both interactions and features data. In case of item-based models the item profile is concatenated to the item features, in case of user-based models the user profile is concatenated with the user features. The hyperparameters and similarity measures are the same as for the ItemKNN and UserKNN, plus a parameter w that controls the relative importance of the content features with respect to the collaborative features. When w is 0, this algorithm is equivalent to the pure collaborative versions, either ItemKNN or UserKNN.

4.2.5 Non-Neural Machine Learning Item-based Approaches

The amount of machine learning models proposed for *top-k* recommendation tasks is vast and selecting what represents the state of the art can be difficult. In order to select suitable examples of algorithms that were considered state of the art before the widespread adoption of neural models we followed the same criteria we adopted for KNN and graph-based baselines and selected several basic models.

- **SLIM ElasticNet**: *Sparse Linear Models (SLIM)* is an item-based regression method for *top-k* recommendation tasks, proposed in [87]. SLIM achieves very competitive recommendation quality and has been used in numerous other papers. In our work, we use a more scalable variant called *ElasticNet* proposed in [71], which learns the item similarity matrix one item at a time (e.g. one column w at a time) by solving a regression problem in such a way that the interactions for the target item y are learned by using all other interactions as training data. Both SLIM and SLIM ElasticNet optimize rating prediction accuracy, therefore, although they allow to achieve competitive ranking performance, they are not natively designed for item ranking tasks. To implement SLIM ElasticNet we used a standard ElasticNet solver provided in the `scikit-learn` package for Python.[3] The hyperparameters of this method include the ratio of *l1* and *l2* regularization as well as a regularization magnitude coefficient.

- **SLIM BPR**: The algorithm learns an item-item similarity matrix similarly to SLIM ElasticNet, but it optimizes ranking accuracy via the

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html`

BPR loss function [6]. The BPR loss function was described in [96], its main idea is to optimize ranking accuracy via gradient ascent by drawing a triple: user, positive item and negative item. While SLIM ElasticNet optimizes rating prediction, SLIM BPR tries to maximize the difference between the score assigned for a user to an item they interacted with (positive item) and an item they did not interact with (negative item). The hyperparameters of this method include the number of neighbors as described in the Nearest-Neighbor Methods, the regularization coefficients and whether the learned similarity matrix should be symmetric or not.

- ***EASE***^{R}: Is a linear item-based model for implicit feedback data recently proposed in [106]. The article shows this method has similarities with auto-encoders which, under the condition defined as "embarrassingly shallow", the autoencoder becomes a linear model with a closed-form solution. EASE^R, as opposed to the previous techniques, does not require gradient descent or ascent and it exhibits very high recommendation accuracy with remarkably fast training time. The only hyperparameter is the choice of the regularization factor.

  Since EASE^R has been published in 2019 the papers covered by our study could not include it as a baseline. Nonetheless, we include it in our study to investigate the competitiveness of shallow auto-encoders as opposed to more complex and deep architectures.

4.2.6 Non-Neural Machine Learning Matrix Factorization Approaches

One of the most known families of recommender algorithms is *Matrix Factorization (MF)*. The application of matrix decomposition methods for collaborative filtering problems was investigated already in the early years of recommender systems [17], and became a de-facto standard after the highly successful FunkSVD method proposed during the Netflix prize competition (2006-2009). Matrix factorization algorithms have been widely studied and a multitude of them exist. In order to provide an overview that is as complete as possible, we sample a few basic models each with particular characteristics. Generally speaking, all the described matrix factorization algorithms share the same model structure. U is the set of users and $|U|$ their number, similarly I is the set of items and $|I|$ their number, $f \in F$ is the index of the latent factor whose total number is $|F|$; $W \in \mathbb{R}^{|U|x|F|}$ is a two-dimensional matrix containing the user factors and $H \in \mathbb{R}^{|I|x|F|}$ is a two-dimensional matrix containing the item factors. $R \in \mathbb{R}^{|U|x|I|}$ is the

rating matrix. Given the latent factor matrices, a prediction is usually computed as the dot product of the user's factor and the item's latent factors:

$$\hat{R}_{u,i} = \sum_{f}^{|F|} W_{u,f} \cdot H_{i,f}$$

- **iALS**: *Alternating Least Squares* was proposed in the seminal work by Hu et al. [61]. In iALS multiple implicit feedback signals are transformed into a confidence value c, which allows to model different types of feedback. An example of such scenario could be made in the video recommendation domain, if a user has opened the detailed information of a recommendation, if hey have watched the trailer, if they begun watching it and they stopped at the beginning, middle or just before the end. In case the available feedback is explicit ratings, the article suggests linear and logarithmic scaling functions, where α and ϵ are parameters of the model:

  $$c_{u,i} = \begin{cases} 1 + \alpha \cdot r_{u,i} \\ 1 + \alpha \cdot \log\left(1 + \frac{r_{u,i}}{\epsilon}\right) \end{cases}$$

  The article also proposed a particular optimization method (i.e., Alternating Least Squares) that allows to render the optimization problem convex, by iteratively fixing one latent factor matrix and updating the other. This strategy allows for iALS to scale very well on larger datasets. A number of hyperparameters can be tuned for the method, including the number of latent factors, the confidence scaling and the regularization factor.

- **PureSVD**: This method corresponds to a the SVD decomposition of the user-item interaction matrix and was proposed in [33]. PureSVD has several advantages when compared to other matrix factorization recommenders, namely it is a standard matrix decomposition technique therefore widely available in efficient mathematical libraries, it is a simple model and has very fast computation time. The disadvantage of PureSVD is that it tries to represent the original user-item interactions without distinguishing between existing interactions and zeros. Its effectiveness can therefore be limited if too many missing interactions are present and specific strategies may be put in place. To implement PureSVD, we used a standard SVD decomposition method provided in the `scikit-learn` package for Python.[4] The only hy-

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.utils.extmath.randomized_svd.html`

perparameter of this method is the number of latent factors.

- **NMF**: This method performs a *Non Negative Matrix Factorization*, which is described in [25]. As opposed to PureSVD, NFM guarantees all latent factors to be positive, at the cost of a higher computation time. NFM decomposition is not unique and has a clustering property, meaning it can be used to cluster items or users. We used a standard NMF decomposition method provided in the `scikit-learn` package for Python.[5] The hyperparameters of this method include the number of latent factors and the solver.

- **FunkSVD**: This is one of the most widely known matrix factorization algorithms, proposed by Simon Funk in his online article[6] during the Netflix Prize. This method optimizes rating prediction via MSE, is therefore not natively designed for item ranking but rather for rating prediction. In the original model only items that have been rated by the user are used for training, this means that the model will never be trained to distinguish between positive and negative items, exhibiting poor ranking performance. In order to ensure the suitability of FunkSVD for a *top-k* recommendation task we added a hyperparameter which ensures a certain quota of the samples used during training are randomly sampled among the unseen items and are associated with a rating of 0. This greatly improves its performance for ranking tasks. The embeddings of users and items are regularized with a Frobenius norm. Another hyperparameter controls whether the model should include the global bias, user bias and item bias. Other hyperparameters include the learning rate, the regularization coefficients, and the number of latent factors. Overall, the predicted rating is computed as follows:

$$\hat{R}_{ui} = \mu + b_u + b_i + \sum_{f}^{|F|} W_{u,f} \cdot H_{i,f}$$

  Where μ is the global bias, $b_u \in \mathbb{R}^{|U|}$ the user bias, $b_i \in \mathbb{R}^{|I|}$ the item bias.

- **MF BPR**: This widely known algorithm was proposed in Rendle et al. [96]. As opposed to FunkSVD, MF BPR is designed to optimize ranking accuracy via the BPR loss and is a widely used baseline in the articles we analyzed. This method, as opposed to FunkSVD, PureSVD

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html`

[6] `http://sifter.org/~simon/journal/20061211.html`

and NFM, has been explicitly designed for implicit interactions. Furthermore, as opposed to iALS, it is trained using gradient ascent. Hyperparameters of this method include the number of latent factor, the learning rates and the regularization coefficients.

4.3 Evaluation metrics

In order to assess the recommendation quality of the algorithms we evaluate in this study, we report two categories of metrics: accuracy metrics and beyond-accuracy metrics.

4.3.1 Accuracy Metrics

Accuracy metrics are usually computed for a recommendation list of a certain length, i.e., cutoff *k*. All the metrics reported are computed for each user independently and then averaged, so the equations reported in this section refer to a single user.

We define $rel(i)$ as a boolean vector having the same length as the recommendation list, its purpose is to model whether an item is *relevant*, i.e., is a correct recommendation, or not:

$$rel(i) = \begin{cases} 1 & \text{if the item in position } i \text{ is relevant} \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, consider rel_t as the total number of relevant items the user has in the test data.

- **Precision (Prec)**: This metric measures the quota of correct recommendations received by a user over the recommendation list length.

$$Prec@k = \frac{1}{k} \sum_{i=0}^{k} rel(i)$$

- **Recall (Rec)** This metric measures the quota of correct recommendations received by a user over the *true positives* in its test data.

$$Rec@k = \frac{1}{rel_t} \sum_{i=0}^{k} rel(i)$$

- **Hit Rate (HR):** This metric computes the number of correct recommendations a user received. If the data is split via leave-one-out its

values are between 0 and 1. If the split is random holdout its value may exceed 1. Hit rate is a non normalized version of Precision, if the recommendation list length is constant for all users the two are equivalent.

$$HR@k = \sum_{i=0}^{k} rel(i)$$

- **Normalized Discounted Cumulative Gain (NDCG):** As opposed to the previous ones, this metrics takes also into account the ranking of the correct recommendations. This metric was originally proposed to evaluate the effectiveness of information retrieval systems [63] and is among the most commonly used metrics in recommender systems research (see Section 7.2.1).

  Assuming that the recommendations for user u are sorted according to the predicted relevance values in decreasing order, DCG is defined as follows:

$$DCG@k = \sum_{i=0}^{k} \frac{2^{r(i)} - 1}{\log_2(i+1)}$$

  where $r(i)$ is the true relevance, as found in the test data, for the item ranked at position i. In case of datasets with explicit ratings, the relevance will be equal to the rating, in case of implicit interactions the relevance will be 1.

  The cumulative gain for each user is normalized by computing the ideal DCG for that same user, denoted as $IDCG$. While the DCG considers all items in the recommendation list, the $IDCG$ is computed assigning to each item its true relevance (i.e., the one in the test data) and therefore obtaining the best possible ranking. The NDCG is then computed as follows:

$$NDCG@k = \frac{DCG@k}{IDCG@k}$$

- **Mean average precision (MAP)**: Is another metric used to measure ranking quality, though is less common than NDCG (see Section 7.2.1). MAP is computed based on the average precision (AP) at increasingly longer recommendation lists, up to the cutoff value, of each user. The AP of each user is defined as follows:

$$AP = \frac{1}{min(k, rel_t)} \sum_{i=0}^{k} Prec@i \cdot rel(i)$$

Finally, given the AP of each user, MAP will be defined as the global average:

$$MAP = \frac{1}{|U|} \sum_{u \in |U|} AP_u$$

4.3.2 Beyond-accuracy Metrics

The core purpose of a recommender system is to assist the user in exploring a catalogue, this of course requires finding relevant recommendations but also to ensure the user is assisted in the discovery of new items. Beyond-accuracy metrics measure how well the recommender is diversifying its recommendations for different users. Broadly speaking, diversity metrics can be classified in two different categories: *individual diversity* and *aggregate diversity*. While individual diversity only measures what is perceived by the user and is computed on each separate recommendation list, aggregate diversity considers the system as a whole and is measured taking into account the recommendations provided to all users [1, 121]. We will define $rec(i)$ as the number of times item i has been recommended, and rec_t as the total number of recommendations (i.e., cutoff value times the number of test users).

In this study, we will focus on the following measures:

- **Item Coverage:** This aggregate diversity metric represents the quota of items that were recommended at least once.

$$coverage = \frac{1}{|I|} \sum_{i \in I} rec(i) > 0 \tag{4.5}$$

  where $|I|$ is the cardinality of the item set and $rec(i) > 0$ is 1 if the item has been recommended at least once, 0 otherwise. Recommender systems exhibiting low coverage will be able to recommend only a low number of items, which can be a significant issue as the recommender fails in one of its most important tasks and constrains user exploration.

- **Shannon Entropy:** This aggregate diversity metric measures the distributional dispersion of the recommendation frequency, taking into account not only whether an item was recommended, but also how often. This metric is the Shannon entropy of how frequently each item has been recommended.

$$\text{Shannon} = -\sum_{i \in I} \frac{rec(i)}{rec_t} \cdot \ln \frac{rec(i)}{rec_t}$$

- **Gini Diversity:** This aggregate diversity metric is too a function of how frequently an item is recommended. It is derived from the definition of the Gini Index, but has its range flipped in such a way that a higher diversity recommender, therefore with balanced frequencies, will have a low Gini Index but a high Gini Diversity. This formulation is aimed at easing the comparison with other diversity metrics sharing a common behavior. Note that the item frequency in this case needs to be sorted by decreasing values. Function $s(i) = j$ given item i will provide its index j in the original non-sorted data.

$$\text{Gini} = \sum_{i=1}^{|I|} \frac{2i - |I| - 1}{|I|} \cdot \frac{rec(s(i))}{rec_t}$$

- **Herfindahl Index (HHI):** This aggregate diversity metric originated from the economics sector and is a quadratic function of the item frequency. Due to its quadratic nature it is more sensitive to changes in the recommendation frequency of items that tend to be recommended often.

$$\text{HHI} = 1 - \frac{1}{rec_t^2} \sum_{i \in I} rec(i)^2$$

- **Mean Inter-List Diversity (MIL):** This diversity metric compares all user's recommendation lists and measures how different they are [135]. It is among the metrics which are computed on the actual recommendations received by each user rather than on the global item count. This diversity considers the uniqueness of different user's recommendation lists and has a value between 0 and 1. The less likely any two users have been recommended the same items, hence the more diverse the recommendations are, the closer MIL will be to 1. Note that MIL was originally called *Personalization*, however we will not use this name due to the fact that the highest value for this metric (i.e., 1) is obtained by a non personalized Random recommender.

$$h(ua, ub) = 1 - \frac{q(ua, ub)}{c} \tag{4.6}$$

Equation 4.6 represents the *inter-list distance* for two users *ua* and *ub*, where $q(ua, ub)$ is the number of common items in their recommendation lists. Equation 4.7 shows how MIL is computed, as an average

over all inter-list distances, excluding the diagonal.

$$MIL = \frac{1}{|U|^2 - |U|} \sum_{\substack{ua,ub \in U \\ ua \neq ub}} h(ua, ub) \tag{4.7}$$

Computing MIL requires to compute function $q(ua, ub)$ for all couples of users, which is quadratic in their number therefore being very computationally expensive for all but the smallest datasets.

As a side contribution of this evaluation study, in Appendix B we demonstrate that MIL diversity is actually a distributional diversity metric, equivalent to both Herfindahl and Hamming diversity. This result allows to clarify the meaning and limitations of MIL diversity and allow its computation in negligible time.

- **Hamming Distance:** This is another metric defined between users [121], applied by representing the user's recommendation lists L as a one-hot encoding L_H of $|I|$ elements. The Hamming distance is the number of positions in which the two lists are different. Since the Hamming distance can be computed from $q(ua, ub)$ as $H(ua, ub) = |I| - q(ua, ub)$ Hamming distance and MIL diversity are equivalent.

It is possible to see that Shannon Entropy is logarithmic, Gini Diversity is linear, given that the $rec(i)$ data must be ordered beforehand, and Herfindahl is quadratic. This means that all three metrics measure the same phenomena, how recommendations are spread across items, but are are sensitive to slightly different behaviors.

4.3.3 Computation Time Metrics

Although often overlooked, the computation time required to train a model and to compute recommendations is an important aspect to take into account when deploying a recommender system. In recent years recommender models have grown more and more complex and frequently remarkably slower to train and to generate recommendations. This aspect is hardly ever mentioned or measured in the original papers, despite being very important. In order to provide a comprehensive picture, in addition to accuracy and beyond-accuracy metrics, we also report measurements of the train and recommendation generation time of all algorithms we evaluate. In order to allow for a meaningful comparison, we ran all experiments

on a specific Amazon AWS instance.[7] The measurements we report are the following.

- **Training Time:** This metric reports the total time required to train the model.
- **Recommendation Time:** This metric measures the time required to generate all recommendation lists during the model evaluation.
- **Throughput [usr/s]:** This metric is computed from the Recommendation Time and shows how many recommendation lists the model is able to generate per second, i.e., to how many users is possible to provide recommendations each second.

4.4 Hyperparameter Tuning

We performed extensive hyperparameter optimization for all examined baselines on each of the datasets used for evaluation. For the investigated neural methods we relied on the optimal hyperparameters reported in the original papers in all but one case. This is appropriate since in our experiments we used the same datasets (if possible the same training-test split) and evaluation procedures as in the original papers. The only exception to this is the SpectralCF algorithm (see Section 5.12). For SpectralCF we discovered an issue in the provided dataset split which was probably the result of an erroneous splitting procedure. Since the characteristics of a correct data split were different, we could not use the same hyperparameters as reported in the original article and so we performed a new hyperparameter optimization, as will be discussed later.

The approach we adopted to search for the optimal hyperparameters is a Bayesian search [3, 36, 38, 50, 58], using the Scikit-Optimize[8] implementation. We did not use the more commonly found Grid search because it has several important limitations, among which the high computational cost and that its results are strongly affected by how the value ranges of continuous valued hyperparameters are discretized. Furthermore it has been known for several years that even a Random search is often preferable and more efficient than a Grid search [12]. A Bayesian search improves over a Random search by providing a trade-off between exploration and exploitation, allowing to explore the hyperparameter space in the first stages of the search and then fine tune the best solution found.

[7] The computation time refers to the total instance time for one AWS instance p3.2xlarge, with 8 vCPU, 30GB RAM, and one Tesla V100-SXM2-16GB GPU.

[8] `https://scikit-optimize.github.io/`

For our Bayesian search we considered 50 cases for each algorithm. The search is composed by three steps:

- First, 15 random samples from the distributions of the hyperparameters are drawn and evaluated. These will constitute the initial points used by the Bayesian Search.
- Then 35 other hyperparameter configurations are explored with the Bayesian search. The hyperparameter configuration achieving the best recommendation quality on the validation data will be selected.
- The model is trained with the optimal hyperparameters on the union of train and validation data and then evaluated on the test data.

4.4.1 Early Stopping Approach

Many machine learning models are trained for a *number of epochs* in which the model's performance on the test data varies, first increasing and then stabilizing or decreasing if the model begins to overfit, usually exhibiting some degree of variance. The number of epochs represents another important parameter to be determined as it can have a strong impact on the quality of the overall model.

Despite this, in the articles we have analyzed neither the number of epochs nor the stopping criteria are usually mentioned. The procedure used to select this parameter in the original articles is therefore not clear. While looking at the code shared or while discussing with the authors we frequently came across bad experimental practices, e.g., number of epochs selected using the test data or reporting the best recommendation quality for each metric regardless of the epoch.

In our experiments we always selected the number of epochs, both for the baselines and for the deep learning models, with early stopping on the validation data. Early stopping is a widely used technique to select the optimal number of train epochs and is available in many libraries like Keras.[9] The idea of early stopping is to periodically evaluate the model on the validation data, while the model is being trained, and stop the training when for a certain number of validation steps the model quality has not improved over the best solution found so far. Early stopping has the advantage of selecting the number of epochs with a transparent criterion, avoiding arbitrary manual optimization, and often results in shorter training times.

To implement early stopping, we use two independent copies of the current model. One is the model that is still being trained, the other is the

[9]For early stopping in Keras, see `https://keras.io/callbacks/#earlystopping`

model frozen at the epoch with the best recommendation quality found so far. If the trained model, after further epochs, exhibits better recommendation quality than the best one found so far, the best model is updated. Since the evaluation step is time consuming, we run five train epochs before each validation step. Moreover, we choose to stop the training if for 5 consecutive validation steps the recommendation quality of the current model is worse than the best one found so far.

4.5 Evaluation Framework

This thesis required extensive experimental evaluation. At the beginning of the project no evaluation framework provided the necessary flexibility to support the variety of baselines, neural algorithms as well as metrics and evaluation protocols required for this study. Significant effort was therefore devoted to the development of a suitable evaluation framework written in Python. The framework and the documentation needed to install it and replicate our experiments are accessible on Github[10] (see also Appendix C).

In this section the structure of the framework will be presented and described via UML diagrams and flow diagrams, for the most important functionalities. The framework contains the following packages:

Recommender Model: This package contains the classes required to build and fit all recommender models.

Evaluation: This package contains the classes required to perform an evaluation of a given Recommender Model.

Hyperparameter Tuning: This package contains the classes required to perform the hyperparameter tuning of a model.

The Recommender Model package contains all recommender models, each as a separate class, and the auxiliary code required for fitting it. Each model has its own class but models of the same family all inherit from an abstract class. Figure 4.1 shows the hierarchy of the similarity based and matrix factorization recommenders. Abstract methods are in italic. Figure 4.3 shows the hierarchy of all similarity based baselines, both item-based and user-based. Figure 4.2 shows the hierarchy of all matrix factorization baselines. The most important methods are:

- *recommend*: generates the recommendation list for a specific user.

[10] `https://github.com/MaurizioFD/RecSys2019_DeepLearning_Evaluation`

- *_compute_item_score*: computes the prediction or score for each item, according to the given model.
- *load_model*: loads a previously saved model
- *save_model*: saves the current model
- *fit*: given the training data, fits the current model, e.g., computes the neighborhood-based heuristic, computes the training data decomposition, performs gradient descent etc.

As it is possible to see, a specialized class for early stopping exists and all models requiring to select the number of epochs inherit from it, i.e., *IncrementalTrainingEarlyStopping*. This class provides some methods to perform the incremental training and evaluations required by early stopping as well as controlling the execution flow.

A high level execution flow for the early stopping is shown in Figure 4.4 with the sequence diagram of the train procedure for the SLIM BPR method. This algorithm has a Python interface and then uses a Cython module (i.e., C code) that performs the train process. The IncrementalTrainingEarlyStopping class controls the training procedure invoking the methods to train the recommender model one epoch at a time and then, when necessary, evaluate it and update the best model.

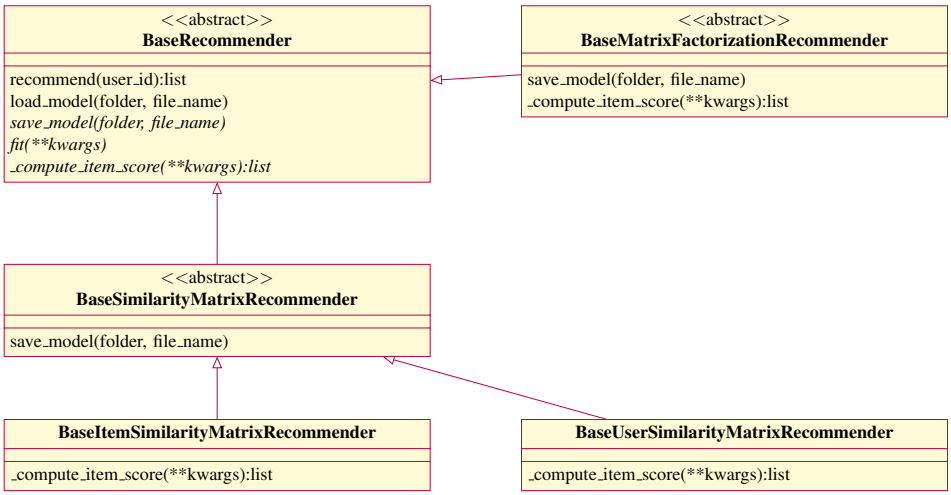


Figure 4.1: *Class hierarchy for Similarity and matrix factorization recommender models. The Similarity matrix recommenders have two sub-classes dependin on whether they are item-based or user-based.*

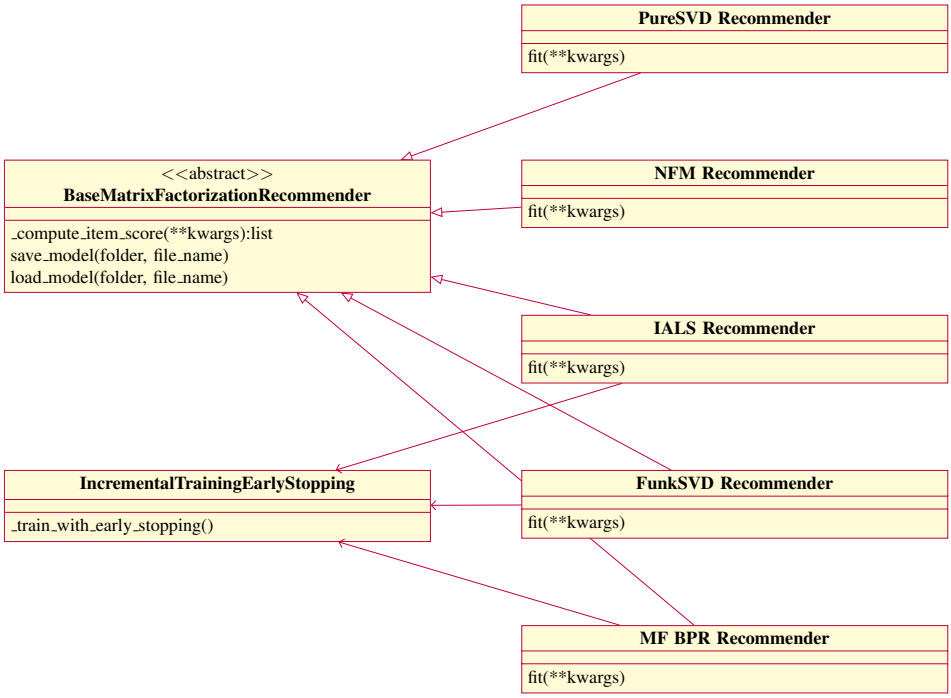


Figure 4.2: *Class hierarchy for matrix factorization recommender models used as baselines. The only algorithms supporting early stopping are iALS, FunkSVD and MF BPR.*

The Evaluation package contains all classes necessary to perform the evaluation and compute the metrics we report. Figure 4.5 shows a sequence diagram of the evaluation of a SLIM BPR model. The Evaluatior object calls the recommend function on the SLIM BPR model which goes all the way up to the main Recommender class. At this point, the scores for all items are computed, in this case by the BaseItemSimilarityRecommender class, of which SLIM BPR is a subclss. The item scores are returned to the Recommender class which ranks them and provides the recommendation list as output. Lastly, the Evaluator object computes the evaluation metrics.

The Hyperparameter Tuning package contains the class which performs the optimization with Bayesian Search using the `scikit-optimize` package for Python.[11] The Evaluation package is used to implement the objective function.

[11] `https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html`

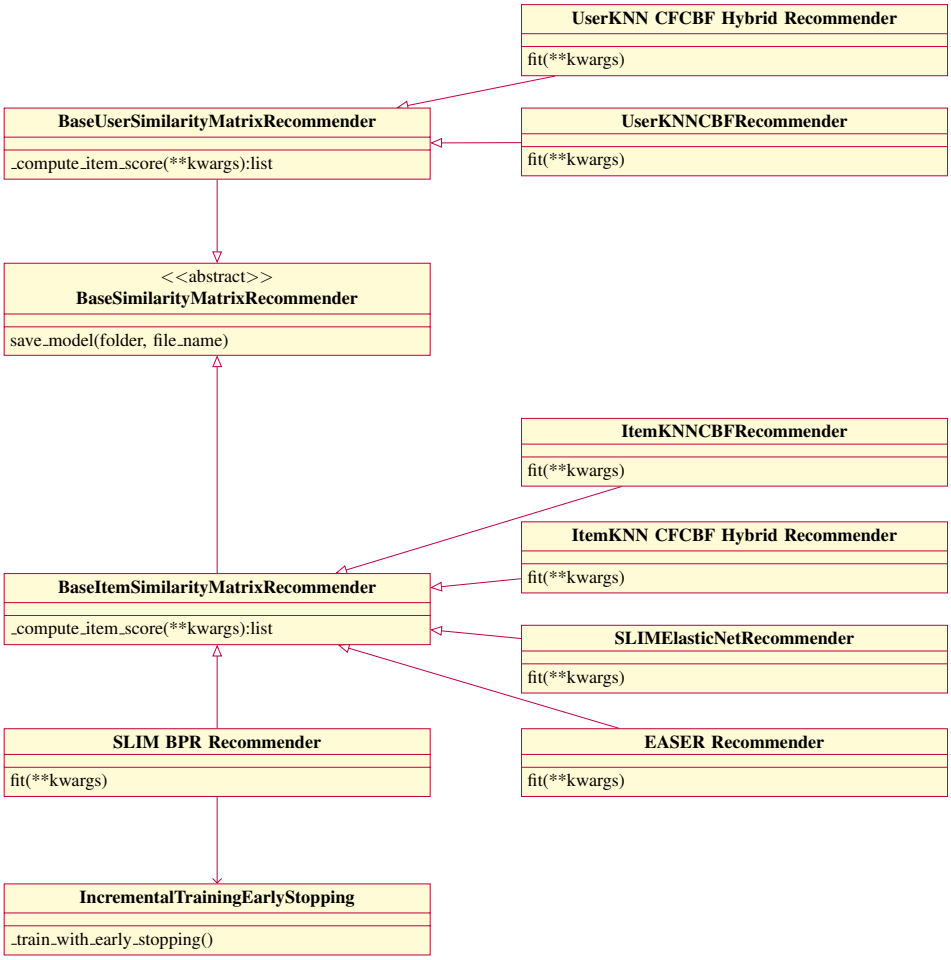


Figure 4.3: *Class hierarchy for Similarity recommender models used as baselines. All user-based models inherit from the BaseUserSimilarityMatrixRecommender class and all item-based models inherit from the BaseItemSimilarityMatrixRecommender class. The only algorithm supporting early stopping is SLIM BPR.*

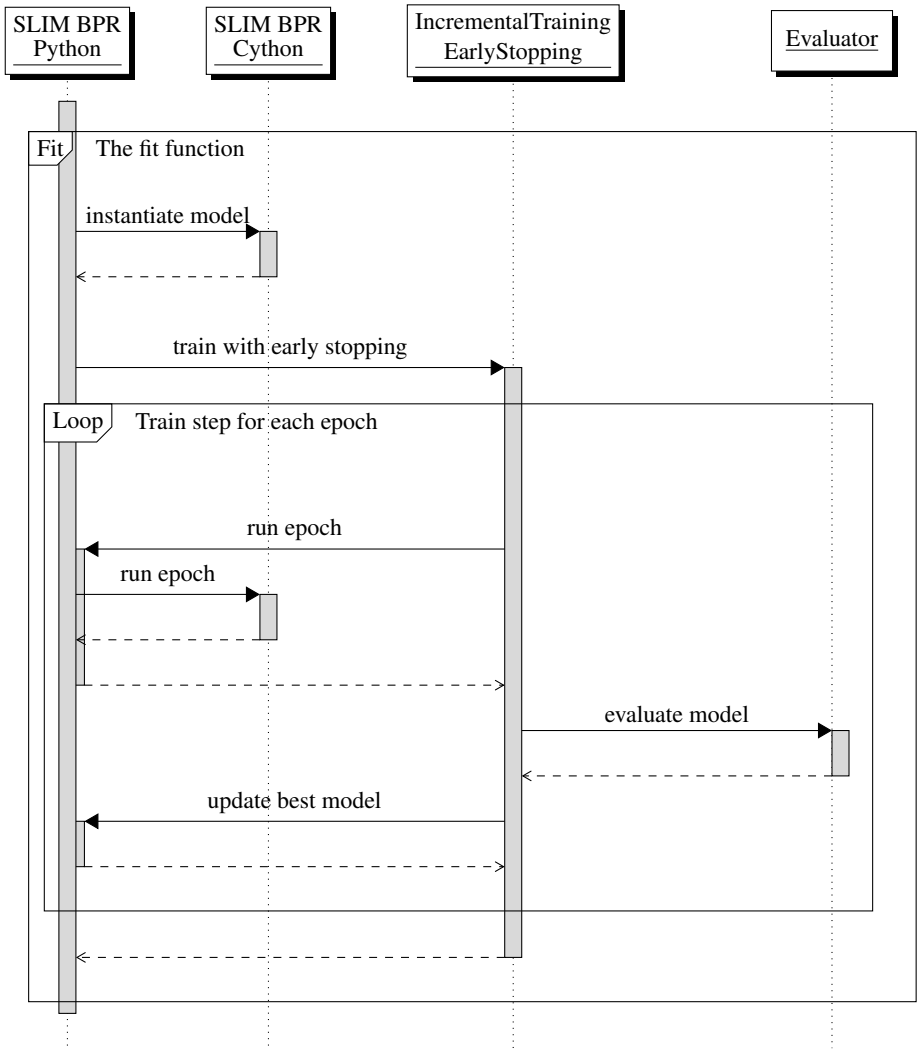


Figure 4.4: *Sequence diagram of the fit function in a SLIM BPR Recommender. The Fit function calls first the Cython implementation of the model, then early stopping is used to loop through the epochs. For each epoch, the model is fitted and evaluated.*

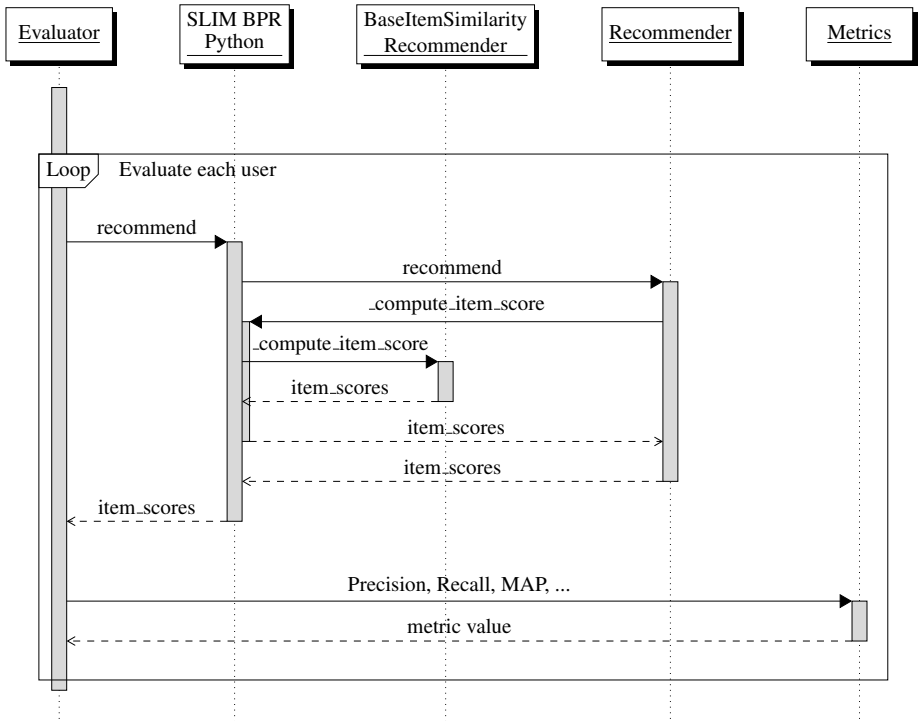


Figure 4.5: *Sequence diagram of the evaluation of a recommender model and of the computation of a recommendation list in a SLIM BPR Recommender. The recommend function calls first the BaseRecommender class, which calls the function to compute the item scores. The scores, in this case, are computed by a class SLIM BPR inherits by, BaseItemSimilarityRecommender.*

CHAPTER 5

Detailed results for reproducible articles

In this Chapter we will provide a detailed analysis of the reproducible algorithms reporting, for each of them: a description of the model and the evaluation protocol adopted in the original article, methodological issues we may have found, the results of our comparison with the new baselines both in terms of recommendation quality and computation time. The articles are ordered by year of publication starting from the oldest one. Due to the extensive number of results (i.e., 900 models evaluated, considering all algorithms and experimental procedures) in this Chapter we only report, for each analyzed algorithm, the most significant ones and summarize the others in the text. The complete results containing the recommendation quality on all datasets, the selected hyperparameters as well as training time and throughput, are reported in Appendix C.

5.1 Collaborative Deep Learning for Recommender Systems (CDL)

Collaborative Deep Learning for Recommender Systems (CDL) is the earliest method we analyze and was proposed in [119] at KDD '15. CDL is a probabilistic feed-forward model of a stacked denoising autoencoder, it

applies deep learning to learn a hybrid joint representation of both content and collaborative information.

5.1.1 Datasets and Evaluation

The evaluation in the original paper is based on three datasets:

CiteULike-a, CiteULike-t: The datasets have been collected in [117] and [118]. CiteULike allows users to create their own collections of articles, each of them has abstract, title, and tags. Both datasets are relatively small (135k and 205k interactions, respectively).

Netflix Prize: This dataset is based upon the Netflix Prize dataset, which contains movie ratings information, and is enriched by content information like the movie plot, crawled from IMDB. Since IMDB content information was not provided by the authors, we did not use this dataset in our experiments.

For each of these datasets, two sets of experiments were made in two sparsity configurations, described by a parameter P, which indicates how many interactions per user are left in the training set (with the rest going to the test set). The two sets of experiments correspond to P values of 1 and 10, which result to 5.5k and 55.5k training interactions, respectively. Note that with $P = 1$ there is only one training interaction per user in the training dataset.

Several baseline techniques were explored in the original article, among them a number of hybrid matrix factorization approaches, a content-based deep learning technique designed for music recommendation, as well as Collaborative Topic Regression, a combination of Latent Dirichlet Allocation on the content attributes and collaborative filtering on the interactions. For evaluation purposes, P interactions for each user were randomly sampled to be part of the training set as mentioned above. The average results of five evaluation runs are reported as well as the standard deviation (omitted from the tables but is mentioned in the text) which exhibits very small values. The authors report Recall for comparably long list lengths (50 to 300), and Mean Average Precision for list length 300.

5.1.2 Methodological considerations

From a methodological perspective, there is no indication of why comparably long list lengths were used for evaluation in the paper and why no measurements were reported for list lengths below 50, which is commonly the case in the literature (see Section 7.2.1, Table 7.5). Furthermore, only

non-neural machine learning baselines were reported and no neighborhood-based algorithms were present. Surprisingly, no pure content-based baselines were reported even though CDL is a hybrid model that uses content information. Lastly, it is not clear how the hyperparameters could have been tuned for the sparse data scenario, i.e., *P=1*, since any further split would result in cold users and that would not allow a meaningful optimization of the reported pure collaborative models, unless specific strategies are put in place.

5.1.3 Results and Discussion

The article shows this method to outperform all baselines on all measures. We could reproduce their results based on the provided code and dataset.[1] Information about the validation set size was not provided in the original paper. In order to optimize the baselines in our own evaluation, we created a validation set using 20% of the training set. In the evaluation scenario where *P=1*, due to the presence of only 1 training instance per user, any sampling would result in cold users. Therefore, in our own experiments, for this scenario the validation data is defined as a subset of the training data. In the evaluation scenario where *P=10*, training and validation data are disjoint.

Comparing CDL against our new baselines we can see that, in three out of four configurations (CiteULike-a with *P=10* and CiteUlike-t with *P=1* and *P=10*), CDL is consistently outperformed by our simple hybrid technique (ItemKNN CFCBF) and, in many cases, also by the pure content-based method (ItemKNN CBF). CDL is instead better than any of our baselines, by a large extent, only when removing all but one user interaction from the CiteULike-a dataset (with *P=1*). In the settings where *P=1*, pure collaborative filtering techniques are, as expected, not competitive.

Since the evaluation methodology is identical, we report exemplary results for CDL and *Collaborative Variational Autoencoder* in the following Section 5.2. Table 5.1 shows exemplary results for the CiteULike-a dataset for the setting *P=10*. In the table, we highlight in bold those entries where a baseline outperformed CDL. We can observe that, for shorter and much more typical list lengths, even the simplest collaborative filtering approaches outperform CDL. The iALS method based on matrix factorization for implicit feedback data was better with respect to CDL in all measurements and cutoff lengths. Finally, the best results were achieved with the

[1] In the source code the authors provided it is reported that the original evaluation contained an error such that the absolute values of the evaluation metrics was higher than the correct one, although the relative performance ordering of the algorithms remained unaltered. Once this error is fixed we can reproduce their results.

pure content-based method that uses only item features to recommend similar items (ItenKNN CBF).

In terms of computation time, CDL can generate a similar number of recommendation lists per second with respect to the baselines, however, it has a substantially longer training time requiring more than one hour while all baselines train in less than 10 minutes or a few seconds for the neighborhood-based models.

5.2 Collaborative Variational Autoencoder (CVAE)

The *Collaborative Variational Autoencoder* (CVAE) method was published in [72] at KDD '17. Like CDL, CVAE is a technique that leverages both content and collaborative information to build a hybrid recommendation model. CVAE learns a deep latent representations for content data in an unsupervised manner and also considers implicit relationships between items and users from both content and rating data. As opposed to previous works with denoising criteria, CVAE learns a latent distribution for content in the latent space instead of the observation space through an inference network.

5.2.1 Datasets and Evaluation

The evaluation in the original paper is based on two datasets:

CiteULike-a, CiteULike-t: The datasets have been collected in [117] and [118]. CiteULike allows users to create their own collections of articles, each of them has abstract, title, and tags. Both datasets are relatively small (135k and 205k interactions, respectively).

The CVAE method is evaluated using the same experimental procedure as CDL (see Section 5.1), i.e., the two datasets are used and different sparsity configurations are tested. Moreover, CVAE is evaluated using Recall at a series of long list lengths. As an additional baseline, the authors include the CDL [119] method described in the previous section. The hyperparameters for all baseline methods are stated to have been optimized using a validation set which is, however, not described.

5.2.2 Methodological considerations

From a methodological perspective, similar observation can be drawn for CVAE and CDL (see Section 5.1). In summary, the recommendation list length is uncommonly long and no motivation for this has been provided.

Similarly, despite leveraging content features, no pure content-based baselines were reported. Lastly, it is not clear how the hyperparameters could have been tuned for the sparse data scenario, i.e., *P=1*, since any further split would result in cold users and that would not allow a meaningful optimization of the reported pure collaborative models, unless specific strategies are put in place.

5.2.3 Results and Discussion

We could reproduce the results for CVAE.[2] Table 5.1 shows the results of our experiments for the CiteULike-a dataset with *P=10*, using the same evaluation measures and protocol as used in the original paper. The table reports the results for both CVAE and CDL (see Section 5.1).

Table 5.1: *Experimental results for the CVAE method for the CiteULike-a with* P=10*.*

	Recall					
	@50	@100	@150	@200	@250	@300
TopPopular	0.0040	0.0078	0.0103	0.0204	0.0230	0.0258
UserKNN CF jaccard	**0.0806**	0.1207	0.1480	0.1705	0.1887	0.2034
ItemKNN CF cosine	**0.0989**	0.1441	0.1752	0.1982	0.2156	0.2300
$P^3\alpha$	**0.0907**	0.1341	0.1636	0.1865	0.2055	0.2206
$RP^3\beta^3$	**0.0963**	0.1408	0.1692	0.1908	0.2090	0.2239
$EASE^R$	**0.0839**	0.1253	0.1546	0.1797	0.1988	0.2128
SLIM BPR	**0.0876**	0.1308	0.1583	0.1821	0.2005	0.2165
SLIM ElasticNet	**0.0869**	0.1281	0.1561	0.1789	0.1970	0.2115
MF BPR	0.0680	0.1011	0.1225	0.1402	0.1542	0.1663
MF FunkSVD	0.0483	0.0866	0.1157	0.1412	0.1636	0.1816
PureSVD	0.0715	0.1079	0.1313	0.1491	0.1636	0.1759
NMF	0.0628	0.1013	0.1285	0.1505	0.1679	0.1843
iALS	0.0779	0.1388	0.1834	0.2186	0.2472	0.2706
ItemKNN CBF cosine	**0.1989**	**0.2835**	**0.3402**	**0.3844**	**0.4193**	**0.4492**
ItemKNN CFCBF cosine	**0.1858**	**0.2816**	**0.3445**	**0.3930**	**0.4335**	**0.4642**
CVAE	0.0805	0.1569	0.2232	0.2760	0.3250	0.3687
CDL	0.0580	0.1108	0.1546	0.1946	0.2314	0.2640

We can see that a simple pure CBF baseline outperforms the more complex CVAE method on all measures, for this dataset, by a large margin. However, if we compare CVAE with the previously published CDL, we can see that there has been indeed an improvement from the very low recommendation quality of CDL. For the other dataset and sparsity configurations, our results are similar to what was reported in the previous section

[2] In the source code the authors provided it is reported that the original evaluation contained an error such that the absolute values of the evaluation metrics was higher than the correct one, although the relative performance ordering of the algorithms remained unaltered. Once this error is fixed we can reproduce their results.

[3] We report $RP^3\beta$ [90] for completeness although the DL algorithm we evaluate here predates its publication.

on CDL, i.e., that CVAE is only better than any of the baselines in the very sparse configuration with *P=1* for the CiteULike-a dataset.

Overall, the authors of CVAE could show an advance with respect to CDL, but our results indicate that CDL did not represent a strong baseline method. We will observe the following phenomenon several times, when a new neural method is proposed and claimed to improve on the state-of-the-art subsequent works will only focus on outperforming that neural method, without considering alternative baselines.

In terms of computation time statistics, by comparing CVAE with the baselines we can see it can generate a similar number of recommendation lists per second, however, it has a substantially longer training time requiring more than one hour while all baselines train in less than 10 minutes or a few seconds for the neighborhood-based models.

5.3 Neural Collaborative Filtering (NCF)

The *Neural network-based Collaborative Filtering* (NCF) framework was proposed in [57] at WWW '17. NCF rapidly gained significant popularity becoming a highly cited paper and being used as a baseline for most later neural recommendation approaches, as shown in Figure 7.1. The NCF framework generalizes matrix factorization by replacing the commonly used inner product with a neural architecture which will be able to model non-linearities in the embeddings. Different variants are considered in the paper:

Generalized Matrix Factorization (GMF): Is used to demonstrate that NCF can represent traditional matrix factorization models by applying a fixed element-wise product of the user and item latent features.

Multi-Layer Perceptron (MLP): Applies hidden layers on the concatenated user and item latent vectors, using a standard MLP to learn the interaction between latent features, being no longer constrained by the linear GMF.

Neural Matrix Factorization (NeuMF): Combines the linear and non-linear modes maintaining a great degree of flexibility, by allowing GMF and MLP to learn separate embeddings, and combine the two models by concatenating their last hidden layer. NeuMF is an ensemble of GMF and MLP.

In our evaluation, we will only consider Neural Matrix Factorization because the original paper showed it to be the best performing one.

5.3.1 Datasets and Evaluation

The evaluation in the original paper is based on two datasets:

MovieLens1M: Is a widely used movie rating dataset. Each user in the dataset has at least 20 ratings and each rating is binarized with a value of 1.

Pinterest: The dataset is gathered from the well known social network in which users may pin an image to their board [52]. Each interaction denotes whether the user has pinned the image to his own board. In the original paper it is stated that since the original dataset is big and very sparse only users with at least 20 interactions were selected.

The authors use a leave-last-out procedure to evaluate their method, the test data is built by selecting the last interaction of each user (according to its timestamp)[4]. The resulting data splits used in the experiments are publicly available. In order to evaluate the model performance, the positive item is ranked with 100 randomly sampled negative items. This procedure is justified stating that ranking all recommendable items is too time consuming, even though the datasets have modest sizes. Hit Rate and NDCG at list length 10 are used as performance metrics.

The original paper includes the following personalized baselines: matrix factorization with Bayesian Personalized Ranking (BPR matrix factorization), the eALS method from 2016 and the ItemKNN method. The original hyperparameter optimization is done on a validation set obtained by randomly selecting one interaction per user. It should be noted here that the validation set is sampled differently from the test set, the reason for this is not provided. For the ItemKNN method, the paper states that the only hyperparameter varied was the number of neighbors (i.e., no shrink term or normalization). According to the reported experiments, the NCF method, and in particular the NeuMF variant, outperformed all baselines on all dataset on all performance measures.

5.3.2 Methodological considerations

The analysis of the provided source code shows that the number of training epochs was chosen by maximizing the Hit Rate on the test data, which will result in information leakage. Since the number of epochs is a parameter

[4]It should be noted, however, that in MovieLens the timestamp refers to the moment when the rating has been added by the user, not when the user-item interaction occurred. Due to this, the ratings could be added at any time, with any order, and their timestamps do not represent the interaction pattern adopted by the user. The appropriateness of using said timestamps as a proxy for the user real interaction pattern is therefore doubtful.

like any other, it must be fixed before testing, e.g., through early stopping on a validation set. In our experiments, we therefore report the performance measure for the number of epochs that was considered optimal based on the validation set applying early stopping. Furthermore, by looking at the provided training-test-negative split, we could observe that for some users the positive test interaction also appears among the negative items, 156 for MovieLens1M and 559 for Pinterest. This is a minor methodological issue that has a marginal impact on the evaluation. For those users the positive item will be ranked with 99, and not 100, negative items resulting in a marginally simpler recommendation problem.

5.3.3 Results and Discussion

We could reproduce the results reported in the original paper. Table 5.2 and Table 5.3 report our results for both the MovieLens and Pinterest datasets. In the original paper the results at list length from 1 to 10 are reported as a plot, therefore in order to compare the results we report those at list length 1, 5 and 10.

On the well-known MovieLens dataset, see Table 5.2, NeuMF is competitive against the simple neighborhood-based baselines, however is outperformed by all but one non-neural machine learning methods. On the Pinterest dataset, see Table 5.3, NeuMF can only outperform PureSVD, which is not optimized for implicit feedback datasets. Most non-neural machine learning techniques are often either similar or better than NeuMF.[5]

If we compare the recommendation quality of non-neural machine learning methods to that of neighborhood-based techniques, on this experimental setup, we can see that while on Pinterest the two are equivalent, on MovieLens non-neural machine learning techniques show a clear advantage.

In terms of the computation cost, the time required by neighborhood-based algorithms is relatively small in both datasets requiring few seconds up to a minute, while non-neural machine learning algorithms require longer training times which are mostly in the range of 5-20 minutes for

[5] After the publication of our first results in [46], we were contacted by the authors of NeuMF who provided us with a new hyperparameter configuration for their method resulting in improved recommendation quality. Such configuration however included new hyperparameter values taken from broader hyperparameter ranges, and required other slight changes in the training procedure. While this new configuration indeed led to slightly improved results for NeuMF, the results of our analysis were confirmed. We would like to clarify that better configurations than those reported in the original papers may indeed exist for all neural methods investigated here, finding them, e.g., in the form of better hyperparameter ranges or alternative network structures, is however not the goal of our analysis. Instead, the goal of this analysis is to assess the reproducibility of existing works and to assess how challenging the reported evaluation really was by comparing the neural algorithm against new and optimized baselines.

[6] We report RP$^3\beta$ [90] for completeness although the DL algorithm we evaluate here predates its publication.

Table 5.2: *Experimental results for NCF (MovieLens 1M)*

	@1		@5		@10	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.1051	0.1051	0.3048	0.2064	0.4533	0.2542
UserKNN CF asymmetric	0.1921	0.1921	0.5070	0.3546	0.6768	0.4100
ItemKNN CF asymmetric	0.1843	0.1843	0.4906	0.3400	0.6627	0.3956
$P^3\alpha$	0.1791	0.1791	0.4846	0.3352	0.6460	0.3876
$RP^3\beta^6$	0.1836	0.1836	0.4935	0.3419	0.6758	0.4011
$EASE^R$	**0.2225**	**0.2225**	**0.5629**	**0.3986**	**0.7192**	**0.4494**
SLIM BPR	0.2013	0.2013	0.5320	0.3713	0.7002	0.4258
SLIM ElasticNet	**0.2207**	**0.2207**	**0.5576**	**0.3953**	**0.7162**	**0.4468**
MF BPR	0.1679	0.1679	0.4619	0.3186	0.6305	0.3730
MF FunkSVD	0.2008	0.2008	0.5202	0.3661	0.6844	0.4192
PureSVD	**0.2132**	**0.2132**	0.5339	0.3783	0.6937	0.4303
NMF	0.2056	0.2056	0.5171	0.3651	0.6844	0.4192
iALS	**0.2106**	**0.2106**	**0.5505**	**0.3862**	**0.7109**	**0.4382**
NCF (NeuMF variant)	0.2088	0.2088	0.5411	0.3803	0.7093	0.4349

Table 5.3: *Experimental results for NCF (Pinterest)*

	@1		@5		@10	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.0467	0.0467	0.1665	0.1064	0.2740	0.1409
UserKNN CF jaccard	**0.2898**	**0.2898**	0.7038	**0.5056**	0.8655	**0.5583**
ItemKNN CF asymmetric	**0.2903**	**0.2903**	**0.7117**	**0.5096**	0.8766	**0.5633**
$P^3\alpha$	**0.2853**	**0.2853**	0.7022	0.5024	0.8700	0.5571
$RP^3\beta^6$	**0.2966**	**0.2966**	**0.7151**	**0.5149**	**0.8796**	**0.5685**
$EASE^R$	**0.2909**	**0.2909**	0.7070	**0.5077**	0.8684	**0.5604**
SLIM BPR	**0.2983**	**0.2983**	**0.7117**	**0.5138**	0.8736	**0.5666**
SLIM ElasticNet	**0.2913**	**0.2913**	0.7059	**0.5072**	0.8679	**0.5601**
MF BPR	0.2655	0.2655	0.6858	0.4833	0.8651	0.5418
MF FunkSVD	0.2601	0.2601	0.6890	0.4820	0.8658	0.5398
PureSVD	0.2630	0.2630	0.6628	0.4706	0.8268	0.5241
NMF	0.2307	0.2307	0.6445	0.4434	0.8343	0.5052
iALS	**0.2811**	**0.2811**	**0.7144**	**0.5061**	0.8761	**0.5590**
NCF (NeuMF variant)	0.2801	0.2801	0.7101	0.5029	0.8777	0.5576

MovieLens and 15 minutes to 3 hours for Pinterest. In both cases we can see that NeuMF, even though it is trained on a high-end GPU, exhibits substantially longer training times, 4 hours for MovieLens and almost 2 days for Pinterest. Furthermore, the number of recommendation lists that can be generated per second is between 7 and 18 times lower.

Overall, NeuMF achieves either lower or comparable recommendation quality as other simpler or non-neural non-neural machine learning baselines, but at significantly greater computational cost. In principle, the archi-

tecture of NeuMF should make it more flexible than the dot-product based matrix factorization baselines here reported. However, a scenario where this is advantageous or translates to higher recommendation quality could not, in our evaluation, be identified.

5.4 Deep Matrix Factorization (DMF)

Deep Matrix Factorization Models (*DMF*) were proposed in [125] at IJCAI '17. As an input to their model, the authors first build a user-item matrix from explicit ratings and implicit feedback, which is then used by a deep *structure learning* architecture. One key aspect here is that a common low-dimensional space for representing users and items is used. Furthermore, the authors develop a new loss function based on cross entropy that considers both implicit feedback and explicit ratings. Using *cross entropy* as loss function allows to represent recommendations with implicit feedback as a binary classification problem. The proposed loss, referred to as *normalized cross entropy*, allows to incorporate the explicit ratings as well.

5.4.1 Datasets and Evaluation

The evaluation in the original paper is based on four datasets:

Amazon Movie, Amazon Music: Datasets publicly available collected from Amazon.com. As a preprocessing step only users with at least 20 interactions and items with at least 5 interactions are retained. This preprocessing step reduces the dataset sizes drastically. For Amazon Movies 80% of the interactions are removed, resulting in a dataset containing only 878k interactions for over 80k movies. For the Amazon Music dataset the preprocessing step removes more than 94% of the interactions, leaving the dataset with only 46k interactions for 18k items. Both datasets are therefore very sparse.

MovieLens100k, MovieLens1M: Both datasets refer to movie recommendations. Each user has at least 20 ratings.

All datasets contain explicit ratings in the range 1-5. The authors share the Amazon Music dataset after the preprocessing step. After analysis, however, we discovered the dataset could not be the result of the preprocessing as described in the paper (further details are reported below). In order to keep the results reported in this paper consistent across all datasets, we apply the preprocessing step as described in the original paper on all datasets, including Amazon Music. We do not report the results obtained

with the pre-processed Amazon Music dataset provided by the authors, but the results are consistent with our pre-processed version.

The training-test split is performed via *leave-last-out*, similarly to what done for NCF (see Section 5.3). The test data for each user is created by selecting the interaction with the latest timestamp. The positive interaction is then ranked together with 99 randomly sampled items that received no interactions from that user, i.e., negative items.[7] The Hit Rate and NDCG at list length 10 are used as metrics. The original training-test splits used in the experiments were not shared by the authors, therefore we created the data splits based on the publicly available datasets.

The original paper included NCF [57] (see Section 5.3), and the baselines reported in that article, i.e., eALS and ItemKNN. It should be noted that, although DMF used explicit and implicit feedback, the article used NCF with binarized feedback even though it can handle explicit ratings seamlessly.

To tune the hyperparameters a validation set was built from the training set by randomly sampling one interaction. The authors report that eALS and NCF were tuned as in the original papers. No details about neighborhood sizes or the used similarity functions are provided for the ItemKNN method.

5.4.2 Methodological considerations

As previously mentioned the authors share the Amazon Music dataset after the preprocessing step which, however, we discovered contains users with less than 20 interactions and items with less than 5 ratings. If the authors removed first the items with less than the desired number of interactions and then the users, the user profiles should all meet the constraints while the item profiles may not because removing users will reduce the number of interactions some items have. The same would apply the other way round. Furthermore, if the preprocessing step had been applied iteratively until convergence, as one would expect, both should meet the constraint. Since neither the users nor the items meet the minimal number of interactions constraint, it remains unclear how exactly the filtering was done. Concerning the methodological aspects, in the provided source code we found that the best Hit Rate and NDCG are obtained by selecting the maximum values those metrics reach, when evaluating the model on the test data, regardless of the epoch. Furthermore, the provided code shows that the authors sample different negative items to be used for testing in each training epoch.

[7] The paper reports that 100 negative items are used, as described for NCF. However, the source code provided by the authors uses 99 negative items. In our experiments we have used 99 negative items.

Table 5.4: *Experimental results for DMF for the MovieLens1M (left) and MovieLens100k (right) datasets.*

	HR@10	NDCG@10
TopPopular	0.4418	0.2475
UserKNN CF asymmetric	0.6324	0.3779
ItemKNN CF cosine	0.6347	0.3808
$P^3\alpha$	0.6097	0.3639
$RP^3\beta$[8]	0.6304	0.3726
$EASE^R$	0.6693	**0.4100**
SLIM BPR	0.6719	**0.4068**
SLIM ElasticNet	**0.6825**	**0.4209**
MF BPR	0.6323	0.3729
MF FunkSVD	0.6499	0.3912
PureSVD	0.6570	0.4015
NMF	0.6422	0.3862
iALS	**0.6947**	**0.4257**
DMF NCE	0.6266	0.3768
DMF BCE	0.6731	0.4033

	HR@10	NDCG@10
TopPopular	0.4145	0.2342
UserKNN CF asymmetric	0.5994	0.3492
ItemKNN CF tversky	0.6026	0.3506
$P^3\alpha$	0.5717	0.3421
$RP^3\beta$[9]	0.5685	0.3270
$EASE^R$	**0.6089**	0.3571
SLIM BPR	**0.6206**	0.3578
SLIM ElasticNet	**0.6238**	**0.3765**
MF BPR	0.5951	0.3365
MF FunkSVD	0.5707	0.3354
PureSVD	0.5877	0.3555
NMF	0.5855	0.3515
iALS	**0.6142**	**0.3691**
DMF NCE	0.5930	0.3410
DMF BCE	0.6026	0.3623

This is questionable as any evaluation will be performed under a different testing condition and would make the results *non comparable*.

In our evaluation we build a new data split following the information in the paper and report the results that were obtained after determining a suitable number of epochs on the validation set by applying eary-stopping and we use the same negative item set for all evaluations.

Lastly, in the original paper DMF is compared to NCF which is only trained with binarized data, even though this method can deal with explicit rating data as well.

5.4.3 Results and Discussion

We reproduced the experiments reported by the authors. The source code was not made available in the article but was provided to us upon request. We report the results of DMF with both loss functions evaluated in the paper, we could observe that, in our experiments, the normalized cross entropy (*NCE*) does not usually lead to accuracy improvements over the binary version (*BCE*).

In our evaluation, we could observe how DMF is not able to outperform the baselines in three out of four datasets. The results for MovieLens datasets are reported in Table 5.4. DMF shows improved quality over traditional nearest-neighbor baselines on the MovieLens datasets, but slightly worse than those obtained with the iALS and SLIM.

The detailed results for the Amazon Music and Movies datasets are

[9] We report $RP^3\beta$ [90] for completeness although the DL algorithm we evaluate here predates its publication.

Table 5.5: *Experimental results for DMF for the Amazon Music (left) and Amazon Movies (right) datasets. EASE^R and SLIM BPR results are missing for the Amazon Movies dataset because the code required too much memory.*

	HR@10	NDCG@10
TopPopular	0.5308	0.3037
UserKNN CF cosine	**0.6694**	0.4798
ItemKNN CF cosine	0.6647	**0.4880**
P$^3\alpha$	0.6588	**0.4823**
RP$^3\beta$[11]	**0.6754**	**0.4912**
EASE^R	0.6600	**0.4836**
SLIM BPR	**0.6694**	0.4720
SLIM ElasticNet	0.6469	0.4744
MF BPR	0.5367	0.3689
MF FunkSVD	0.5474	0.3870
PureSVD	0.5912	0.4190
NMF	0.6540	0.4486
iALS	0.6600	**0.4880**
DMF NCE	0.4799	0.3371
DMF BCE	0.6659	0.4815

	HR@10	NDCG@10
TopPopular	0.5794	0.3489
UserKNN CF cosine	0.7327	0.5132
ItemKNN CF asymmetric	0.6986	0.4914
P$^3\alpha$	0.6972	0.5028
RP$^3\beta$[12]	0.7107	0.5078
EASE^R	-	-
SLIM BPR	-	-
SLIM ElasticNet	0.6981	0.5005
MF BPR	0.6422	0.4161
MF FunkSVD	0.5972	0.4091
PureSVD	0.6021	0.4156
NMF	0.6252	0.4217
iALS	0.7352	0.5230
DMF NCE	0.6832	0.4677
DMF BCE	**0.7818**	**0.5417**

shown in Table 5.5.[10] While on MovieLens datasets machine learning algorithms tend to provide better recommendation quality then neighborhood-based models, for the Amazon Music dataset the opposite happens, since the simple UserKNN and RP$^3\beta$ methods work better here than any non-neural machine learning model. We can observe that for the Amazon Music dataset DMF is not competitive against the baselines, while for the Amazon Movies dataset DMF is actually much better than all other methods on both measures. In particular the gains in terms of the Hit Rate are substantial and much higher than the second best method iALS. The question of why DMF seems to perform so well on Amazon Movies and not on the other datasets is still open. We can observe however that Amazon Movies is a very sparse dataset, which, after preprocessing, has an average of two interactions per item. As discussed in Sections 5.1 and 5.2, both CDL and CVAE methods worked similarly well on a sparse configuration with only one training interaction per user.

The article argues that DMF combines implicit feedback and explicit feedback. However, DMF mainly relies on the explicit ratings contained in the dataset and fills all missing interactions with zeros, which are considered as implicit ratings. This constitutes a questionable interpretation of implicit ratings. Under such interpretation, even PureSVD, which approximates the rating matrix by performing an SVD decomposition and therefore treats all missing interactions as zeros, could be similarly defined a hybrid.

[10]The results for EASE^R are missing for the Amazon Movies dataset, as the author-provided Python implementation of the method needed too much memory.

A last significant observation that can be made regarding DMF is its very high computation cost. For Amazon Music most baselines complete their training in seconds or 2-3 minutes, while DMF requires up to 36 minutes. For MovieLens1M most baselines complete their training in seconds or a few minutes, while DMF requires between 4 and 7 days. On both other datasets the results are consistent, with DMF requiring up to 6 hours of training for MovieLens100k and 20 hours on Amazon Movie. Remember that training times for the baselines are computed on a CPU while those of the deep learning models on a high-end GPU. The number of recommendations that the algorithm can provide per second is also very small when compared to all baselines.

5.5 Variational Autoencoders for Collaborative Filtering (Mult-VAE)

Variational Autoencoders for Collaborative Filtering was proposed in [73] at WWW '18. Mult-VAE is a collaborative filtering algorithm based on implicit feedback, which uses variational autoencoders. The paper introduces a generative model with multinomial likelihood, a different regularization parameter for the learning objective and uses Bayesian inference for parameter estimation. The authors furthermore show that there is an efficient way to tune the parameter using annealing.

5.5.1 Datasets and Evaluation

The evaluation in the original paper is based on three datasets:

MovieLens20M: A common dataset for movie recommendations. The ratings are binarized by keeping only ratings equal or greater than 4. Users with less than 5 interactions are then removed.

Netflix Prize: This is the user-movie ratings data from the Netflix Prize.[13] The ratings are binarized by keeping only ratings equal or greater than 4 and users with less than 5 interactions are removed.

Million Song Dataset: This is a dataset of user-song play counts released as part of the Million Song Dataset [15]. The playcounts are binarized. Users with less than 20 songs and songs that were listened by less than 200 users are removed.

[13] http://www.netflixprize.com/

After preprocessing, the datasets are still relatively large, having between 10 and almost 57 millions interactions. We only report the experiments for the MovieLens and the Netflix Prize datasets, since the paper did not contain sufficient information to guarantee we would use the Million Song Dataset in the exact same way as the authors.

The original paper reports as baselines four machine learning models, iALS, SLIM, NCF [57] (see Section 5.3) and the Collaborative Denoising Autoencoder (CDAE) method proposed in [122] in 2016. The data is split in training-validation-test by holding out a certain number of user profiles. For example, for the MovieLens20M dataset (136k users overall), 10k users are removed for validation and 10k users are removed for testing. For each held out user profile, 80% of the interactions are used as user profile, and the remaining 20% are used as ground truth to measure the performance metrics. When evaluating the model, the held out user profiles are fed to Mult-VAE which then builds a representation for those users, and only afterwards the recommendations are computed. The authors motivate this procedure by stating it provides stronger generalizability.

The original article mentions all models are optimized for NDCG@100 on the validation set. Performance results for Recall@20 and Recall@50 are reported as well.

This evaluation methodology required a slight modification to the baselines we report. In order to use matrix factorization baselines on cold users we built the cold users' latent factors based upon both their user profile and the latent factors of the warm items. In particular, we added a hyperparameter to the matrix factorization models to select how those cold user's latent factors are estimated, either via an item based similarity or an item embeddings average, see [33]. In the first case an ItemKNN model is created by defining the similarity matrix as the dot product of the items' latent factor matrix by its transpose. In the second case the latent factors of a user are the product of the user profile and the items' latent factors, resulting in the average of the embeddings of the items the user interacted with. Usually, generating an ItemKNN similarity matrix proved to be the most effective solution.

5.5.2 Methodological considerations

The analysis of the article and the publicly provided materials did not yield to the discovery of methodological issues for Mult-VAE. Nonetheless, in the original paper NDCG@100, Recall@20 and Recall@50 are used, however no reason is provided on this unusual selection of different metrics at

different cutoffs and, for example, why Recall@100 or NDCG@20 were not reported. To obtain a more comprehensive picture, we report additional measurements at the corresponding but missing cutoff values: Recall@100, NDCG@20 and NDCG@50. As a minor observation, a standard deviation is mentioned in the results of the original paper, but the number of random data splits is not described.

5.5.3 Results and Discussion

The source code, the data splits and information about the seed for the random number generator are publicly available, based on those we could reproduce the results reported in the paper. Table 5.6 shows our results for the MovieLens20M dataset and Table 5.7 those for the Netflix Prize dataset.

Table 5.6: *Results for Mult-VAE for the MovieLens20M dataset. UserKNN could not be applied because of the evaluation protocol (hold-out of users).*

	@20		@50		@100	
	REC	NDCG	REC	NDCG	REC	NDCG
TopPopular	0.1441	0.1201	0.2320	0.1569	0.3296	0.1901
UserKNN	-	-	-	-	-	-
ItemKNN CF asymmetric	0.2937	0.2444	0.4486	0.3087	0.5709	0.3527
$P^3\alpha$	0.2620	0.2168	0.4047	0.2742	0.5287	0.3182
$RP^3\beta$	0.3006	0.2501	0.4540	0.3133	0.5797	0.3583
$EASE^R$	0.3530	**0.3074**	0.5147	**0.3755**	0.6353	**0.4196**
SLIM BPR	0.3206	0.2646	0.4783	0.3291	0.6030	0.3731
SLIM ElasticNet	0.3356	0.2920	0.4893	0.3576	0.6110	0.4017
MF BPR	0.2379	0.1888	0.3867	0.2481	0.5100	0.2908
MF FunkSVD	0.2765	0.2254	0.4243	0.2864	0.5576	0.3331
PureSVD	0.2935	0.2514	0.4371	0.3117	0.5544	0.3538
NMF	0.2269	0.1960	0.3533	0.2480	0.4664	0.2875
iALS	0.2968	0.2496	0.4406	0.3090	0.5631	0.3521
Mult-VAE	**0.3541**	0.2988	**0.5222**	0.3690	**0.6517**	0.4158

For the MovieLens dataset, we observed a positive result and could confirm the claims made by the original article of Mult-VAE. On all measurements, both the original and the additional ones, Mult-VAE leads to better performance results than all baseline methods available at the time of the algorithm publication (therefore excluding $EASE^R$). SLIM is the second best method in this evaluation, with performance results that are around 1% to 2% lower in terms of the NDCG.

For the Netflix Prize dataset, the claims of the original article could not be confirmed to the full extent. In terms of NDCG, which is the optimization criterion, SLIM outperforms Mult-VAE at all list lengths. Mult-VAE

Table 5.7: *Results for Mult-VAE for the Netflix Prize dataset. UserKNN could not be applied because of the evaluation protocol (hold-out of users).*

	@20		@50		@100	
	REC	NDCG	REC	NDCG	REC	NDCG
TopPopular	0.0786	0.0762	0.1643	0.1159	0.2717	0.1570
UserKNN	-	-	-	-	-	-
ItemKNN CF cosine	0.2091	0.1970	0.3387	0.2592	0.4598	0.3092
$P^3\alpha$	0.1960	0.1759	0.3325	0.2412	0.4633	0.2962
$RP^3\beta$	0.2210	0.2053	0.3633	0.2739	0.4932	0.3281
$EASE^R$	**0.2681**	**0.2591**	**0.4170**	**0.3334**	**0.5471**	**0.3890**
SLIM BPR	0.2394	0.2219	0.3767	0.2886	0.5004	0.3403
SLIM ElasticNet	0.2555	**0.2479**	0.4002	**0.3203**	0.5299	**0.3752**
MF BPR	0.1572	0.1403	0.2748	0.1952	0.3952	0.2431
MF FunkSVD	0.2300	0.2130	0.3609	0.2758	0.4803	0.3250
PureSVD	0.2271	0.2184	0.3593	0.2840	0.4784	0.3342
NMF	0.1844	0.1802	0.3035	0.2385	0.4172	0.2856
iALS	0.1956	0.1839	0.3138	0.2410	0.4216	0.2862
Mult-VAE	0.2615	0.2423	0.4127	0.3167	0.5456	0.3730

is however better in terms of Recall.

Overall, with Mult-VAE a method was found which was easy to reproduce, thanks to all needed material being made by the authors publicly available. Furthermore, as our results indicated, the method consistently outperformed existing methods at least on one well-known and comparably large dataset. We could also confirm that $EASE^R$ leads to improvements over Mult-VAE in most cases, supporting the claim that shallow models are a competitive solution.

Considering the computation cost, we can observe that the training time for Mult-VAE is comparable to that of several non-neural baselines on both datasets. In particular, on the Netflix Prize dataset Mult-VAE is much faster than the competitive SLIM. The number of recommendations per second is also comparable. It should be noted that all deep learning algorithms we analyze are trained and evaluated on a high-end GPU, while all baselines on a CPU. This means that Mult-VAE achieves this timings on a higher performance device with respect to the other baselines. Still, based on our experiments, the low training time of Mult-VAE when compared to the best performing baselines constitutes a very rare occurrence

5.6 NeuRec: On Nonlinear Transformation for Personalized Ranking

NeuRec [133] was presented at IJCAI '18. Its underlying idea is to learn user-item relationships from implicit feedback and then combine latent factor models with neural networks in order to capture both linear and nonlinear dependencies in the data. A multi-layered network is used to map user-item interactions into a low-dimensional space. Recommendations are then generated by computing the inner product of item and user latent factors, as in a traditional matrix factorization algorithm. Two variants are proposed: *INeuRec* (item-based) and *UNeuRec* (user-based).

5.6.1 Datasets and Evaluation

The evaluation in the original paper is based on four datasets:

MovieLens1M: Is a well know movie dataset collected by GroupLens, each user has at least 20 ratings.

MovieLens Hetrec: Is an extension of MovieLens1M released in 2011 where ratings are linked with the corresponding IMDB information.

FilmTrust: Is a crawled dataset from a movie sharing and rating website[14].

Frappe: Is an Android application recommendation dataset. In case of multiple interactions for the same user-item, only the earliest is considered. The original dataset contains around 100k interactions which are reduced to 20k after preprocessing.

For all datasets, the interactions or ratings are binarized and set to a value of 1. The original data splits were not provided by the authors, but could be reproduced based on the description provided in the paper.

The evaluation is performed using five random training-test splits (80% - 20%) and reporting the average results and the standard deviation. As performance metrics, the authors use Precision and Recall at list lengths 5 and 10, as well as MAP, MRR, and the NDCG, at list length 10.

Among the baselines, the authors consider SLIM, BPR matrix factorization, NeuMF and the GMF model, which is part of NCF [57] (see Section 5.3). The original paper does not provide information regarding the hyperparameter tuning of the baselines, except for GMF and NeuMF, which are said to use the *default* configuration that was used in that article.

[14] `https://www.librec.net/datasets/filmtrust.zip`

5.6.2 Methodological considerations

The source code is publicly available but, despite this and the availability of the detailed hyperparameter configurations, we could *not* obtain the results reported in the original paper for the NeuRec method. We contacted the authors on this issue but we were not able to define an experimental pipeline, from data preprocessing to hyperparameter optimization, that allowed us to obtain results comparable to what reported in the original paper. Therefore, the reason for this discrepancy could not be clarified. However, for the non-personalized TopPopular baseline we were able to obtain results consistent with the original paper for all but one dataset, Frappe. This suggests the data splitting is consistent to what done in the original paper and the issue lies with the NeuRec model, either in the hyperparameter setting or the epoch selection criteria.

Hyperparameters for NeuRec were determined through grid search for each dataset and are reported in the paper in detail. This constitutes a good experimental practice as it means the model has been optimized for each dataset. In stark contrast to this good practice adopted to optimize NeuRec, as previously mentioned, the paper states that for GMF and NeuMF the hyperparameters reported in the original article have been used. This constitutes a bad experimental practice since both the evaluation protocol and most of the datasets used in this experiments are different from those that had been used to tune GMF and NeuMF. The results reported for these methods are therefore prone to be suboptimal and cannot be claimed to represent the best performance those algorithms can reach.

As opposed to the other parameters, the number of training epochs and the stopping criteria are not reported in the paper. According to an exchange of emails with the authors, the training was done for a large number of epochs and the best performance values on the test set were reported, which appears to be a common practice even though it causes information leakage. In our experiments, as previously described, we selected the number of epochs via early stopping on a validation split.

5.6.3 Results and Discussion

The source code for NeuRec is publicly available and executable but, as we previously mentioned, we could not reproduce the results obtained in the original paper. The outcome of our evaluation is that NeuRec is outperformed on any dataset and almost on any measure by at least one, but usually several, of the baselines in our comparison.

Due to the number of datasets, cutoffs and metrics, the full results are

extensive. Since the conclusions are consistent for all datasets, in Table 5.8 we only provide the results for two datasets, MovieLens1M and FilmTrust and for list lengths 5 and 10. All other results can again be found in Appendix C.

Looking at the results, we observe that on both datasets even the simplest baselines are better than NeuRec. For MovieLens1M the performance of the best baselines is better by a large margin while for FilmTrust the difference is less marked. For the HetRec dataset, which is not shown in detail here, the result is the same. Finally, for the small and rarely used Frappe dataset, NeuRec leads to the best results for Precision@5, but is outperformed, e.g., by RP$^3\beta$ on all other measures. The Frappe dataset is very small and tends to show very unstable results when split randomly. In particular, compared to the results reported in the original paper, our TopPopular algorithm exhibits results that are four times higher; also NeuRec's values are two times higher. Due to this we do not consider Frappe as a reliable dataset on which to base our conclusions in this experimental setting.

Overall, for both datasets, NeuRec often achieves similar recommendation quality as a TopPopular recommender. In order to assess whether NeuRec was indeed able to personalize its recommendations, for this experiment we also include beyond-accuracy metrics. From Table 5.9 we can see that for both MovieLens and FilmTrust datasets UNeuRec and the non-personalized TopPopular algorithm exhibit very similar diversity across all measurements. In particular for Item Coverage, both are able to recommend only less than 2%-3% of the available items. By looking at the low Mean Inter-List Diversity (MIL) we can clearly see that the generated recommendation lists for any two users will tend to have a significant number of items in common. We can say therefore that UNeuRec behaves like a non-personalized recommender algorithm, in this experimental scenario. The item-based variant, INeuRec exhibits both better recommendation quality and diversity than the user-based one. For the MovieLens dataset, INeuRec has a comparable diversity to the other better performing baselines. For the FilmTrust dataset however, INeuRec falls significantly behind with an Item Coverage of only 7%, compared to 28% of the better performing RP$^3\beta$. Overall, NeuRec seems to be able to provide only limited personalization when compared with our simple baselines.

In terms of computational cost, NeuRec has a higher training time than all baselines. On FilmTrust, most baselines are able to complete their training in less than a minute, while NeuRec requires between 3 and 7 minutes. The difference is more notable for MovieLens1M where, when all baselines can be trained in less than 10 minutes, NeuRec requires between 16 and 19

Table 5.8: *Experimental results for the selected MovieLens1M and FilmTrust datasets.*

	MovieLens1M									
			@5					@10		
	PREC	REC	MAP	NDCG	MRR	PREC	REC	MAP	NDCG	MRR
TopPopular	0.2105	0.0402	0.1531	0.0689	0.3621	0.1832	0.0685	0.1168	0.0939	0.3793
UserKNN CF asymmetric	**0.4212**	**0.1065**	**0.3441**	**0.1674**	**0.6399**	**0.3617**	**0.1726**	**0.2774**	**0.2230**	**0.6509**
ItemKNN CF asymmetric	**0.3995**	**0.0984**	**0.3244**	**0.1563**	**0.6179**	**0.3452**	**0.1590**	**0.2618**	**0.2084**	**0.6293**
$P^3\alpha$	**0.4041**	**0.1007**	**0.3286**	**0.1596**	**0.6250**	**0.3456**	**0.1627**	**0.2627**	**0.2121**	**0.6362**
$RP^3\beta$	**0.4080**	**0.1007**	**0.3325**	**0.1602**	**0.6260**	**0.3508**	**0.1639**	**0.2676**	**0.2137**	**0.6374**
$EASE^R$	**0.4488**	**0.1134**	**0.3717**	**0.1779**	**0.6620**	**0.3857**	**0.1820**	**0.3035**	**0.2364**	**0.6717**
SLIM BPR	**0.3964**	**0.1034**	**0.3161**	**0.1606**	**0.6222**	**0.3358**	**0.1663**	**0.2494**	**0.2128**	**0.6335**
SLIM ElasticNet	**0.4437**	**0.1106**	**0.3692**	**0.1749**	**0.6578**	**0.3813**	**0.1770**	**0.3003**	**0.2321**	**0.6679**
MF BPR	**0.3576**	**0.0830**	**0.2812**	**0.1340**	**0.5628**	**0.3073**	**0.1384**	**0.2217**	**0.1807**	**0.5768**
MF FunkSVD	**0.3936**	**0.0927**	**0.3154**	**0.1479**	**0.6000**	**0.3458**	**0.1555**	**0.2572**	**0.2014**	**0.6125**
PureSVD	**0.4123**	**0.0987**	**0.3371**	**0.1586**	**0.6266**	**0.3575**	**0.1624**	**0.2722**	**0.2132**	**0.6380**
NMF	**0.3811**	**0.0891**	**0.3017**	**0.1430**	**0.5817**	**0.3338**	**0.1499**	**0.2442**	**0.1948**	**0.5947**
iALS	**0.4164**	**0.1036**	**0.3373**	**0.1635**	**0.6327**	**0.3628**	**0.1702**	**0.2743**	**0.2200**	**0.6443**
INeuRec	0.3280	0.0663	0.2554	0.1110	0.5003	0.2839	0.1094	0.2027	0.1500	0.5129
UNeuRec	0.2098	0.0395	0.1560	0.0684	0.3663	0.1856	0.0688	0.1199	0.0944	0.3852
	FilmTrust									
			@5					@10		
	PREC	REC	MAP	NDCG	MRR	PREC	REC	MAP	NDCG	MRR
TopPopular	0.4200	**0.4126**	0.4393	**0.4203**	0.6145	0.3471	**0.6351**	**0.4597**	**0.5450**	0.6273
UserKNN CF jaccard	**0.4354**	**0.4400**	**0.4713**	**0.4510**	**0.6492**	**0.3560**	**0.6455**	**0.4907**	**0.5712**	**0.6579**
ItemKNN CF asymmetric	**0.4286**	**0.4238**	**0.4564**	**0.4360**	**0.6348**	**0.3490**	**0.6303**	**0.4734**	**0.5548**	**0.6447**
$P^3\alpha$	**0.4240**	**0.4199**	**0.4526**	**0.4321**	**0.6351**	**0.3500**	**0.6343**	**0.4719**	**0.5550**	**0.6467**
$RP^3\beta$	**0.4373**	**0.4365**	**0.4709**	**0.4492**	**0.6537**	**0.3575**	**0.6436**	**0.4880**	**0.5701**	**0.6631**
$EASE^R$ $EASE^R$	**0.4458**	**0.4498**	**0.4825**	**0.4604**	**0.6620**	**0.3597**	**0.6590**	**0.4990**	**0.5805**	**0.6717**
SLIM BPR	**0.4327**	**0.4351**	**0.4643**	**0.4455**	**0.6465**	**0.3522**	**0.6399**	**0.4825**	**0.5647**	**0.6549**
SLIM ElasticNet	**0.4418**	**0.4417**	**0.4803**	**0.4572**	**0.6600**	**0.3583**	**0.6566**	**0.4983**	**0.5796**	**0.6708**
MF BPR	0.4115	0.4047	0.4309	0.4114	0.5979	0.3433	0.6156	0.4519	0.5330	0.6088
MF FunkSVD	0.4112	0.4004	0.4148	0.3972	0.5781	0.3452	0.6265	0.4378	0.5245	0.5917
PureSVD	**0.4292**	**0.4255**	**0.4563**	**0.4366**	**0.6366**	**0.3478**	0.6255	**0.4724**	**0.5526**	**0.6453**
NMF	0.2721	0.2407	0.2769	0.2584	0.4131	0.1983	0.3332	0.2443	0.3123	0.4234
iALS	0.4038	0.3855	0.4240	0.4028	0.6021	0.3342	0.5920	0.4400	0.5201	0.6137
INeuRec	0.4221	0.4089	0.4398	0.4196	0.6151	0.3466	0.6187	0.4577	0.5398	0.6261
UNeuRec	0.4174	0.4062	0.4384	0.4181	0.6157	0.3472	0.6291	0.4596	0.5436	0.6286

hours. Despite this significantly higher computational cost, NeuRec falls significantly below all simple baselines on MovieLens. We can also see that all algorithms, both baselines and deep learning ones, are able to generate the same number of recommendations per second. It should be noted that NeuRec generates a latent factor representation of items and users and the predictions are computed using the dot product of those embeddings. Therefore, NeuRec is a matrix factorization algorithm and at the evaluation phase it shares the same code used by all matrix factorization baselines, explaining the similar number of recommendations per second.

Table 5.9: *Experimental results for beyond-accuracy metrics for NeuRec on the selected MovieLens1M and FilmTrust datasets at cutoff 10.*

	MovieLens1M				
	Div. MIL	Div. HHI	Cov. Item	Div. Gini	Div. Shannon
TopPopular	0.4946	0.9495	0.0180	0.0050	4.5858
UserKNN CF asymmetric	0.9288	0.9929	**0.2602**	0.0511	7.9644
ItemKNN CF asymmetric	0.9283	0.9928	**0.2664**	0.0500	7.9312
$P^3\alpha$	0.9080	0.9908	0.1741	0.0371	7.5290
$RP^3\beta$	0.9121	0.9912	0.1927	0.0426	7.6928
$EASE^R$	**0.9398**	**0.9940**	**0.2486**	**0.0582**	**8.1688**
SLIM BPR	0.8748	0.9875	0.1914	0.0313	7.2022
SLIM ElasticNet	0.9296	0.9929	0.2354	0.0508	7.9582
MF BPR	0.9148	0.9915	**0.2589**	0.0483	7.8170
MF FunkSVD	**0.9596**	**0.9959**	0.2259	**0.0731**	**8.5521**
PureSVD	**0.9422**	**0.9942**	0.1968	**0.0523**	**8.0678**
NMF	**0.9589**	**0.9959**	**0.2607**	**0.0792**	**8.6403**
iALS	**0.9535**	**0.9953**	**0.2463**	**0.0682**	**8.4346**
INeuRec	0.9377	0.9938	0.2414	0.0515	8.0217
UNeuRec	0.5048	0.9505	0.0185	0.0052	4.6377
	FilmTrust				
	Div. MIL	Div. HHI	Cov. Item	Div. Gini	Div. Shannon
TopPopular	0.6318	0.9631	0.0275	0.0129	5.0890
UserKNN CF jaccard	**0.7332**	**0.9733**	**0.1318**	**0.0199**	**5.7041**
ItemKNN CF asymmetric	0.6646	0.9664	0.0526	0.0147	5.2798
$P^3\alpha$	**0.7020**	**0.9701**	**0.1241**	**0.0178**	**5.5226**
$RP^3\beta$	**0.7148**	**0.9714**	**0.2849**	**0.0292**	**5.8254**
$EASE^R$	**0.7303**	**0.9730**	**0.0966**	**0.0184**	**5.6183**
SLIM BPR	**0.6962**	**0.9696**	**0.1053**	**0.0169**	**5.4571**
SLIM ElasticNet	**0.7418**	**0.9741**	**0.0980**	**0.0188**	**5.6488**
MF BPR	**0.6944**	**0.9694**	**0.1719**	**0.0192**	**5.5164**
MF FunkSVD	0.6467	0.9646	0.0328	0.0139	5.1954
PureSVD	**0.7665**	**0.9766**	0.0502	**0.0196**	**5.7145**
NMF	**0.9242**	**0.9923**	**0.4389**	**0.1234**	**8.1504**
iALS	**0.7856**	**0.9785**	0.0478	**0.0206**	**5.7426**
INeuRec	0.6654	0.9665	0.0715	0.0149	5.3145
UNeuRec	0.6348	0.9634	0.0285	0.0131	5.1080

5.7 CoupledCF: Learning Explicit and Implicit User-item Couplings

CoupledCF was proposed in [131] at IJCAI '18. CoupledCF is a hybrid content and collaborative filtering recommender which leverages side information, e.g., user demographics, item features. Its core observation is that in real-world datasets users and items are not independent and identically distributed. The proposed method aims to learn implicit and explicit couplings between users and items which should allow to leverage side information more effectively. This is achieved using a complex architecture

composed by a CNN to learn the couplings based on the side information and a deep CF model that considers explicit and implicit interactions between users and items.

The authors propose two variants of their model:

Deep Collaborative Filtering (DeepCF): This variant concatenates user and item latent factors into a multi-layered fully-connected neural network to learn the implicit user-item interactions.

Coupling learning for collaborative filtering (CoupledCF): This variant uses a CNN to learn user-item couplings. It consists of two components: a local CoupledCF (lCoupledCF) which models the explicit user-item couplings using a CNN aimed at capturing local user-item interaction, and a global CoupledCF (gCoupledCF) which combines the local CoupledCF output with a global user-item embedding product. The local and global model are combined in the final CoupledCF model.

5.7.1 Datasets and Evaluation

The evaluation in the original paper is based on two datasets:

MovieLens1M: Is a well known movie rating dataset which contains user side information (i.e., gender, occupation and age) and item side information (i.e., genre). Each user has at least 20 ratings. All ratings are binarized and transformed to 1.

Tafeng: Is a dataset that contains grocery store transactions and side information about items (i.e., asset code, price) and users (i.e., age, region). This dataset has about 750k transactions (i.e., less than the MovieLens1M dataset), but is much more sparse as it contains many more users and items.

The original source code was not publicly available at the time this analysis was made. However, the authors provided us with the source code and the training-test splits, including the sampled test negative items they had used during the evaluation. The source code has been made publicly available by the original authors at the time of writing.[15]

The training-test split is performed via leave-one-out, creating the test split by sampling for each user one random interaction. The recommendation accuracy is computed by ranking the positive item against 99 items

[15] https://github.com/zhqgui/CoupledCF

(negative) the user did not interact with. The 100 items are then ranked by the algorithm and the Hit Rate and the NDCG are used to evaluate the performance. Cutoff list lengths between 1 and 10 were considered.

The article mentions the hyperparameters of the proposed model were systematically fine-tuned but no information regarding the hyperparameter tuning for the baselines is provided. Among the reported baselines are NCF [57] (see Section 5.3), and Google's Wide&Deep method [22].

5.7.2 Methodological considerations

Since CoupledCF relies on side information, we have included item-based and user-based content techniques among the baselines as well as the content-collaborative hybrid KNN baselines. Although we have been provided with the source code and the data split by the original authors, we could not fully reproduce the results reported in the original paper for CoupledCF, we could fully reproduce the results only for DeepCF.

By analyzing the provided data split we could observe there seems to be an issue regarding how the data was split. For both Movielens and Tafeng the negative item data contains duplicates leading to many users (72% for Movielens, 28% for Tafeng) having less of the desired 99 negative items, e.g., some have only 93. For the Tafeng dataset, cumulatively, almost 3,000 negative items (0.1% of the total number of negative items) also appeared as training items or test items for that same user. If the number of unique negative items is not constant, different users will be evaluated under *slightly* different conditions, in particular, the recommendation problem becomes easier as the number of negatives decreases due to the reduction in the pool of alternatives the recommender has to choose from.

We also found that 8% of the users in the Tafeng dataset have inconsistent test data, being either associated to a test item but no negative items or vice versa. If a user has no test data this means that the user will be excluded from the evaluation as any model will always have zero success, similarly, if a user has no negative items, any recommender will always be able to suggest the only item available, which is the test item, and achieve a perfect recommendation score.

This problem is present in the original data split and therefore all models will be evaluated under the same conditions. However, if the users having inconsistent test data have a different behavior with respect to the others, then the relative ordering of the algorithms in terms of their recommendation quality could change. This inconsistency of the training-test splits with respect to what described in the article suggests an erroneous splitting

procedure.

Looking at the methodological aspects, we could again observe that the baselines were not properly optimized and default hyperparameters were used. Furthermore, we could observe that the original source code reports the maximum value each metric reaches, when evaluated on the test data, regardless of the epoch. In our experiments, again, we report the values of the metrics after the optimal number of epochs is chosen with early stopping on the validation set.

Now, we consider the claims made in the article more broadly. The original article states that using a CNN on the outer product of the embeddings, the way CoupledCF does, allows to learn couplings between them. This statement is justified by claiming that an outer product of embeddings is analogous to an image and therefore the convolution operation can be applied. This, we argue is not sufficient, because it merely compares the shape of these two rather than the properties the convolution operation leverages. Furthermore, the article states that CoupledCF is able to effectively learn the user-item couplings because it outperformed other methods which did not use them. Again, this we argue is not sufficient since when comparing models with very different architectures many factors, other than what is claimed, could have had an impact on the results. We found other instances of the convolution operation applied on the outer product of embeddings without proper justification. We further discuss these claims and the weaknesses the evaluation protocol had in Chapter 6.

5.7.3 Results and Discussion

Although we have been provided with the source code and the data split by the original authors, we could not fully reproduce the results reported in the original paper, as mentioned above. We consider in our experiments both versions of the model: DeepCF and CoupledCF.

The results are shown in Table 5.10 (MovieLens) and Table 5.11 (Tafeng).

For the MovieLens datasets CoupledCF is competitive against the simple neighborhood-based methods and the hybrids, except for very short list lengths, confirming previous observations that machine learning models tend to perform well on MovieLens. However, other non-neural methods like iALS and EASE^R are consistently better than CoupledCF. Furthermore the difference between CoupledCF and DeepCF is rather small, this is in contrast to the original article which reported CoupledCF was able to achieve substantial improvements over DeepCF.

For the Tafeng dataset, all collaborative algorithms, both simple neighborhood-

Table 5.10: *Experimental results for CoupledCF for the MovieLens1M dataset.*

	@1		@5		@10	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.1593	0.1593	0.4217	0.2936	0.5813	0.3451
UserKNN CF asymmetric	0.3546	0.3546	0.6914	0.5343	0.8114	0.5735
ItemKNN CF cosine	0.3305	0.3305	0.6682	0.5080	0.7940	0.5488
$P^3\alpha$	0.3316	0.3316	0.6543	0.5031	0.7687	0.5402
$RP^3\beta$	0.3464	0.3464	0.6743	0.5198	0.7959	0.5591
$EASE^R$	**0.4003**	**0.4003**	**0.7258**	**0.5738**	**0.8343**	**0.6093**
SLIM BPR	0.3515	0.3515	0.6843	0.5281	0.7983	0.5651
SLIM ElasticNet	**0.3906**	**0.3906**	**0.7116**	**0.5625**	**0.8315**	**0.6014**
MF BPR	0.3151	0.3151	0.6550	0.4945	0.7838	0.5365
MF FunkSVD	**0.3646**	**0.3646**	0.7017	**0.5434**	0.8151	**0.5802**
PureSVD	**0.3735**	**0.3735**	**0.7088**	**0.5522**	0.8132	**0.5861**
NMF	0.3508	0.3508	0.6879	0.5291	0.7995	0.5656
iALS	**0.3816**	**0.3816**	**0.7121**	**0.5581**	0.8200	**0.5933**
ItemKNN CBF asymmetric	0.0884	0.0884	0.2586	0.1752	0.3780	0.2137
UserKNN CBF tversky	0.1714	0.1714	0.4427	0.3108	0.6065	0.3636
ItemKNN CFCBF cosine	0.3328	0.3328	0.6694	0.5107	0.7985	0.5526
UserKNN CFCBF dice	**0.3555**	**0.3555**	0.6869	0.5328	0.8008	0.5698
DeepCF	0.3550	0.3550	0.7017	0.5388	0.8272	0.5794
CoupledCF	0.3522	0.3522	0.7018	0.5374	0.8247	0.5775

Table 5.11: *Experimental results for CoupledCF for the Tafeng dataset.*

	@1		@5		@10	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	**0.2654**	**0.2654**	0.5194	0.3965	0.6549	0.4402
UserKNN CF cosine	**0.3215**	**0.3215**	**0.5412**	**0.4369**	0.6415	**0.4693**
ItemKNN CF asymmetric	**0.3322**	**0.3322**	**0.5445**	**0.4442**	0.6356	**0.4736**
$P^3\alpha$	**0.3245**	**0.3245**	**0.5503**	**0.4437**	0.6404	**0.4730**
$RP^3\beta$	**0.3202**	**0.3202**	**0.5525**	**0.4424**	0.6470	**0.4732**
$EASE^R$	**0.3272**	**0.3272**	**0.5452**	**0.4417**	0.6435	**0.4736**
SLIM BPR	**0.3171**	**0.3171**	**0.5454**	**0.4368**	0.6457	**0.4693**
SLIM ElasticNet	**0.3233**	**0.3233**	**0.5438**	**0.4389**	0.6476	**0.4726**
MF BPR	0.2556	0.2556	0.5017	0.3827	0.6315	0.4247
MF FunkSVD	**0.2676**	**0.2676**	0.5196	0.3980	0.6541	0.4414
PureSVD	0.2462	0.2462	0.4889	0.3714	0.6260	0.4156
NMF	0.2556	0.2556	0.4761	0.3706	0.5765	0.4031
iALS	**0.2920**	**0.2920**	0.5219	**0.4126**	0.6293	**0.4473**
ItemKNN CBF asymmetric	0.0589	0.0589	0.0958	0.0769	0.1467	0.0931
UserKNN CBF asymmetric	0.2464	0.2464	0.4654	0.3600	0.5798	0.3970
ItemKNN CFCBF asymmetric	**0.3331**	**0.3331**	**0.5434**	**0.4442**	0.6314	**0.4727**
UserKNN CFCBF asymmetric	**0.3424**	**0.3424**	**0.5882**	**0.4713**	**0.6937**	**0.5055**
DeepCF	0.2647	0.2647	0.5244	0.3995	0.6583	0.4428
CoupledCF	0.2641	0.2641	0.5175	0.3948	0.6499	0.4377

based methods and non-neural machine learning models outperform CoupledCF by a significant margin. Only the pure content-based baselines do not reach the performance level of CoupledCF. On Tafeng we could observe the recommendation quality of CoupledCF is at the level of the TopPopular baseline. The simpler DeepCF method also leads to better accuracy results than the CoupledCF variant.

For the Tafeng datasets, CoupledCF achieves similar recommendation quality as a TopPopular recommender. In order to assess whether CoupledCF was indeed able to personalize its recommendations, for this experiment we also include beyond-accuracy metrics. As can be seen from Table 5.12, CoupledCF exhibits better diversity than the non-personalized TopPopular baseline. In particular, CoupledCF has a much better Item Coverage and a higher Gini Diversity, meaning that the frequency at which each item is recommended is more balanced than that of the TopPopular model, indicating a lower popularity bias. Despite achieving similar recommendation quality as the TopPopular baseline, in this experiment, CoupledCF still exhibits better capability to differentiate the recommendation lists, although without being able to provide a higher recommendation accuracy. This is, to some extent, a positive finding if we consider that in another case where the neural algorithm had similar recommendation quality as the TopPopular baseline it also had the same low diversity, i.e., it was not able to personalize (see Section 5.6).

In terms of computational cost, on MovieLens most neighborhood-based baselines have a training time of a few seconds and non-neural machine learning models of a few minutes, while CoupledCF requires almost 4 hours. For Tafeng the difference is less marked with nearest-neighbour baselines requiring around 10-20 seconds, most non-neural machine learning models requiring 30 to 40 minutes and CoupledCF slightly more than 1 hour.

5.8 DELF: A Dual-Embedding based Deep Latent Factor Model for Recommendation

The *Dual-Embedding based Deep Latent Factor Model* was presented in [23] at IJCAI '18. DELF was designed for *top-k* recommendation tasks with implicit feedback data with the goal to generalize both N-Singular Value Decomposition (NSVD) [89] and Neural Collaborative Filtering (NCF) [57]. In NSVD a user embedding is determined using the embeddings of all the items they interacted with. An issue with NSVD is that if two users have interacted with the same set of items but provided completely differ-

Table 5.12: *Experimental results for CoupledCF on beyond-accuracy metrics for the selected Tafeng dataset at cutoff 5.*

	Tafeng				
	Div. MIL	Div. HHI	Cov. Item	Div. Gini	Div. Shannon
TopPopular	0.9965	0.9992	0.1360	0.0535	10.6959
UserKNN CF cosine	**0.9978**	**0.9995**	**0.5041**	**0.1268**	**11.8704**
ItemKNN CF cosine	**0.9981**	**0.9996**	**0.6799**	**0.1898**	**12.3269**
$P^3\alpha$	**0.9978**	**0.9995**	**0.5740**	**0.1369**	**11.9159**
$RP^3\beta$	**0.9990**	**0.9998**	**0.9148**	**0.3974**	**13.4528**
$EASE^R$	**0.9982**	**0.9996**	**0.5924**	**0.1516**	**12.1238**
SLIM BPR	**0.9977**	**0.9995**	**0.5831**	**0.1374**	**11.8920**
SLIM ElasticNet	**0.9977**	**0.9995**	**0.4654**	**0.1116**	**11.7116**
MF BPR	0.9970	0.9993	0.2171	0.0661	11.0286
MF FunkSVD	0.9965	0.9992	0.1392	0.0537	10.7046
PureSVD	0.9966	0.9993	0.1433	0.0539	10.7071
NMF	**0.9982**	**0.9996**	**0.3697**	**0.1145**	**11.8346**
iALS	**0.9985**	**0.9997**	**0.3529**	**0.1401**	**12.1119**
ItemKNN CBF asymmetric	**0.9996**	**0.9999**	**0.9018**	**0.4758**	**13.8621**
UserKNN CBF asymmetric	**0.9977**	**0.9995**	**0.5346**	**0.1265**	**11.8308**
ItemKNN CFCBF asymmetric	**0.9982**	**0.9996**	**0.6811**	**0.2015**	**12.4229**
UserKNN CFCBF asymmetric	**0.9977**	**0.9995**	**0.4444**	**0.1161**	**11.7731**
DeepCF	0.9971	0.9994	0.2314	0.0724	11.1734
CoupledCF	0.9969	0.9993	0.1859	0.0627	10.9586

ent ratings, the difference cannot be modeled and the users will be indistinguishable. NCF is a widely used framework for collaborative filtering (see Section 5.3) in which users and items are represented via embeddings learned with a multi-layer perceptron (MLP)

DELF learns *dual* embeddings to capture interactions in the data. In particular, instead of using the common user embedding, the authors propose to learn an additional item-based embedding to represent the users and a user-based embedding to represent the items. The embeddings are then combined to model non-linear interactions between users and items within a deep learning architecture. Through this approach the authors generalize ideas of NSVD and Neural Collaborative Filtering (NCF). Two variants of the approach were investigated applying different fusion functions to merge the embeddings representation:

DELF-MLP: Applies a Multi Layer Perceptron on the concatenation of all embeddings representations.

DELF-EF: Applies an empirical approach that assigns different weights to the different embeddings according to the number of interactions of

the user and the item.

5.8.1 Datasets and Evaluation

The evaluation in the original paper is based on two datasets:

Amazon Music: Amazon dataset about Digital Music[16], all users with less than 20 interactions are removed. Through this preprocessing, 90% of the interactions were removed, resulting in only 76k interactions for 41k items. Each item is associated, on average, to 1.8 interactions. The users are 1.8k, each with an average of 40 interactions.

MovieLens1M: The well know movie dataset, each user has at least 20 ratings.

In both cases all explicit ratings were binarized and transformed to 1.

DELF was evaluated under a similar procedure as NFC (see Section 5.3). The test data was sampled via leave-last-out, by selecting for each user the interaction with the latest timestamp. Given this training-test split methodology, the training data for Amazon Music contains, on average, 1.7 interactions per item. The test item is then ranked together with 99 randomly sampled negative items, and the Hit Rate and the NDCG at a cut-off length of 10 were used as performance measures. The original article includes several baselines among which BPR matrix factorization, iALS, DMF (see Section 5.4), and two variants of the NCF model [57] (see Section 5.3).

The article states that hyperparameters for the proposed model were systematically optimized on a validation set built with the second most recent interaction.

5.8.2 Methodological considerations

By analyzing the source code we found, like in other works discussed here, the number of epochs was selected by optimizing the result on the test data, which will cause information leakage. Furthermore, there is no mention in the original article concerning the hyperparameter tuning of the baselines.

A particular issue we observed is in the Amazon Music dataset. As previously mentioned the preprocessing reduces substantially the size of the dataset to fewer than 2k users, and to only fewer than 2 interactions per item. Due to this, when the training-test split is performed according to the timestamp, 52% of the items in the test data are cold (i.e., items that never

[16] `http://jmcauley.ucsd.edu/data/amazon/`

	Max pop	Avg pop	Gini Index	Kendall Tau	Shannon
	Amazon Music				
Full data	78.00	1.83	0.42	1.00	14.51
Training data	75.00	1.79	0.43	0.94	14.48
Test data	6.00	0.04	**0.96**	**0.15**	**10.72**
	MovieLens1M				
Full data	3428.00	257.10	0.65	1.00	10.81
Training data	3407.00	255.55	0.65	1.00	10.81
Test data	50.00	1.56	**0.75**	**0.56**	10.26

Table 5.13: *The statistics compare the popularity bias of the two datasets used to evaluate DELF, Amazon Music and MovieLens1M. Amazon Music has a test distribution very different from the train distribution.*

appear in the training data). This is particularly surprising considering that DELF is a pure collaborative algorithm that will, therefore, not be able to build a representation for any of those cold items. This means that the real modeling capacity of both DELF and of all other collaborative baselines will be evaluated, in practice, only for the 900 users whose test data contains a warm item, i.e., an item that appeared at least once in the training data.

Further statistics on the popularity bias of the two datasets used for the evaluation are reported in Table 5.13. We can see that the Amazon Music dataset has a test data with a very different distribution than the training data. In particular, the Gini Index of the test data (0.96) shows a substantially more pronounced popularity bias than in the training data (0.43). The table also reports the Kendall Tau metric, which counts the number of pairwise agreements between two ranking lists, the lists are the item indices ranked according to their popularity in the given data split (i.e., train, test). Its result is the percentage of item pairs whose ordering is consistent between the two splits, i.e., one item is more popular than the other in both training and test. We can see that the Kendall Tau of the training data when compared to the full dataset is almost 1 and that of the test data is 0.15, which means the popularity distribution of the items has been strongly altered. If we consider the MovieLens1M dataset, we can still see a difference in the statistics of the data between training and test, but to a much more limited extent.

This suggests that the combination of preprocessing and splitting protocol adopted for Amazon Music is resulting in a drastically altered data which likely does not represent the intended scenario.

5.8.3 Results and Discussion

We were able to reproduce the results reported in the original paper. In order to tune the baselines and perform early stopping we used a validation set created in the same way as reported by the authors. NDCG@10 was used as an optimization criterion.

According to our results, DELF for both the MovieLens1M dataset (see Table 5.15) and the Amazon Music dataset (see Table 5.14) was never the best-performing model in any measurement. On MovieLens, DELF was consistently outperformed by most non-neural machine learning algorithms, while not consistently by neighborhood-based algorithms. This again confirms that on MovieLens machine learning algorithms tend to perform better than neighborhood-based ones. On Amazon Music, on the other hand, DELF was outperformed by both neighborhood-based and non-neural machine learning algorithms.

Table 5.14: *Experimental results for DELF for the Amazon Music dataset. EASE^R results are missing because the code required too much memory on these datasets.*

	@5		@10		@20	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.2474	0.1730	0.3041	0.1913	0.3738	0.2090
UserKNN CF cosine	**0.3150**	**0.2495**	**0.3471**	**0.2600**	0.3738	**0.2668**
ItemKNN CF asymmetric	**0.3090**	**0.2506**	**0.3401**	**0.2609**	0.3717	**0.2689**
P$^3\alpha$	**0.3074**	**0.2465**	**0.3373**	**0.2564**	0.3689	**0.2644**
RP$^3\beta$	**0.3046**	**0.2434**	**0.3379**	**0.2543**	0.3651	**0.2611**
EASE^R	-	-	-	-	-	-
SLIM BPR	**0.3008**	**0.2380**	**0.3390**	**0.2504**	0.3673	**0.2576**
SLIM ElasticNet	**0.3101**	**0.2526**	**0.3411**	**0.2625**	0.3711	**0.2701**
MF BPR	0.2360	0.1888	0.2687	0.1995	0.3095	0.2099
MF FunkSVD	0.2545	0.2035	0.2899	0.2150	0.3292	0.2248
PureSVD	0.2627	0.2141	0.3084	0.2290	0.3542	0.2406
NMF	**0.2910**	**0.2294**	**0.3482**	**0.2480**	**0.4038**	**0.2621**
iALS	**0.3319**	**0.2604**	**0.3706**	**0.2729**	**0.4109**	**0.2831**
DELF MLP	0.2905	0.2239	0.3275	0.2361	0.3787	0.2489
DELF EF	0.2883	0.2224	0.3313	0.2364	0.3831	0.2496

In terms of computational cost, we can see that DELF requires significantly more training time than all other baselines. On Amazon Music all baselines complete their training in few seconds for the neighborhood-based models to 20 minutes for the non-neural machine learning ones, while DELF requires more than 3 hours. Significantly, on this dataset, the number of recommendation lists that can be generated per second is remarkably lower than all be baselines, 3 against 500. For MovieLens the observations that can be made are similar, except that the recommendation

Table 5.15: *Experimental results for DELF on the MovieLens dataset.*

	@5		@10		@20	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.3302	0.2229	0.4696	0.2674	0.6577	0.3148
UserKNN CF asymmetric	**0.5205**	**0.3635**	**0.6852**	**0.4168**	0.8329	**0.4542**
ItemKNN CF cosine	0.4936	0.3426	0.6677	0.3989	0.8243	0.4387
$P^3\alpha$	0.4945	0.3438	0.6574	0.3965	0.7952	0.4313
$RP^3\beta$	0.5138	0.3559	0.6809	0.4102	0.8276	0.4475
$EASE^R$	**0.5716**	**0.4064**	**0.7258**	**0.4566**	**0.8516**	**0.4887**
SLIM BPR	**0.5380**	**0.3742**	**0.7077**	**0.4292**	**0.8452**	**0.4640**
SLIM ElasticNet	**0.5706**	**0.4038**	**0.7306**	**0.4557**	**0.8586**	**0.4882**
MF BPR	0.4844	0.3310	0.6595	0.3877	0.8152	0.4275
MF FunkSVD	**0.5312**	**0.3708**	**0.6948**	**0.4239**	0.8245	**0.4569**
PureSVD	**0.5513**	**0.3891**	**0.7021**	**0.4382**	0.8303	**0.4708**
NMF	**0.5339**	**0.3746**	**0.6965**	**0.4272**	**0.8385**	**0.4635**
iALS	**0.5643**	**0.3975**	**0.7228**	**0.4489**	**0.8354**	**0.4776**
DELF MLP	0.5168	0.3587	0.6809	0.4119	0.8342	0.4508
DELF EF	0.4805	0.3305	0.6504	0.3852	0.8043	0.4243

lists generated per second, although still much less than the baselines, exhibit a lower difference, 11 against 500.

5.9 Outer Product-based Neural Collaborative Filtering (ConvNCF)

The *Convolutional Neural Collaborative Filtering* (ConvNCF) method was presented in [56] at IJCAI '18. Its purpose is to overcome a limitation of several current recommendation models that combine user and item embeddings via a concatenation or a dot product, therefore assuming the embedding dimensions are independent. ConvNCF attempts to explicitly model the pairwise correlations between the embedding dimensions. This is done in three steps. First a traditional matrix factorization BPR model is trained on the data. Secondly, the pretrained embeddings are used to build an *interaction map*, via an outer product. The interaction map is said to be more expressive than the simple concatenation of embeddings or the element-wise product. Lastly the interaction map, which is said to be analogous to an image, is used to train a convolutional neural network whose purpose is to learn to model the embeddings interactions.

5.9.1 Datasets and Evaluation

The evaluation in the original paper is based on two datasets:

Gowalla: Is a check-in dataset from Gowalla[17], a location-based social network. The data is filtered to remove all users with less than 2 interactions and items with less than 10 interactions. The pre-processed dataset contains 69k interactions.

Yelp: Is the Yelp Challenge[18] dataset for user ratings on business, the authors remove users and items with less than 10 interactions.

In both datasets all interactions are considered as implicit feedback with value 1. Both datasets contain multiple interactions at different timestamps for the same user-item pair. These interactions are merged by keeping only the earliest for each user-item pair.[19] Both filtered datasets with their training-test splits have been provided by the authors.

The test data is split by a leave-last-out protocol, selecting for each user the interaction with the latest timestamp. The positive item is then ranked against 999 randomly selected negative items. The Hit Rate and the NDCG at different list lengths are used as evaluation measures. The hyperparameter optimization of both baselines and the proposed method is done via a validation set sampled by holding out randomly one interaction per user, it is therefore built differently from the test split.

5.9.2 Methodological considerations

The analysis of this article, as well as of the provided source code and data split led to the discovery of several methodological issues.

By analyzing the source code we could see that the number of epochs was chosen by optimizing the recommendation quality on the test data, causing information leakage.

Furthermore, the article states the embedding size was set to the constant value of 64 for all matrix factorization baselines. This procedure is justified by claiming this allows for a "*fair comparison*", however we argue it achieves the opposite effects. The embedding size is a hyperparameter of all matrix factorization models that should be tuned for each dataset and for each algorithm. Different algorithms will have different objective functions and training procedures that will likely require different values for it. Constraining the hyperparameter space in this way will result in the reported baselines to achieve suboptimal recommendation quality. Any experimental evaluation performed based on their results will therefore be meaningless.

[17] `https://snap.stanford.edu/data/loc-gowalla.html`
[18] `https://github.com/hexiangnan/sigir16-eals/raw/master/data/yelp.rating`
[19] For the Gowalla dataset the authors use the first interaction appearing in the file as the *earlier* interaction.

We also observed an issue with the publicly available test data splits. The set of negative test items contained duplicates and partially overlapped with the training data. The number of unique negative items is different across users, with the lowest being 961 for Yelp and 976 for Gowalla. This results in different users being evaluated under slightly different conditions and the recommendation problem becoming easier as the number of negatives decreases. We also report that at the time of our experiments the published version of the algorithm contained a bug that caused information leakage from the test data. The only items that could be selected as negatives were those unobserved in the training data that also did not appear as test items. Items unobserved in the training data should all be potential negative items to be sampled during training, regardless of the test data, otherwise test items will be advantaged over other items. This bug was later fixed on the public Github repository. In our experiments, during the training we only relied upon training data and never used any information from the test data to alter the training process.

The original article stated that using a CNN on the outer product of the embedding, the way ConvNCF does, allows to model the interactions between embeddings. This statement is justified by claiming that an outer product of embeddings is analogous to an image and therefore the convolution operation can be applied. This we argue is not sufficient, because it merely compares the shape of the two rather than the properties the convolution operation leverages. Furthermore, the article stats that ConvNCF is able to effectively learn the user-item couplings because it outperforms other methods which do not use them. Again, this we argue is not sufficient since when comparing models with very different architectures many factors, other than what is claimed, could have had an impact on the results. We found other instances of the convolution operation applied on the outer product of embeddings without proper justification. We further discuss this claims and the weaknesses the evaluation protocol had in Chapter 6.

5.9.3 Results and Discussion

The source code and the datasets are publicly available and we could reproduce the results reported in the original paper. The results for the Yelp dataset are shown in Table 5.16. Those for the larger Gowalla dataset are given in Table 5.17.

For the Yelp dataset, ConvNCF is consistently outperformed by the traditional neighborhood-based methods, RP$^3\beta$, and SLIM. The other baselines outperform ConvNCF as well in most cases. For the Gowalla case,

Table 5.16: *Experimental results for ConvNCF for the Yelp dataset. SLIM BPR results are missing because the code required too much memory on this dataset.*

	@5		@10		@20	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.0817	0.0538	0.1200	0.0661	0.1751	0.0799
UserKNN CF asymmetric	**0.2131**	**0.1400**	**0.3209**	**0.1747**	**0.4482**	**0.2068**
ItemKNN CF cosine	**0.2521**	**0.1686**	**0.3669**	**0.2056**	**0.4974**	**0.2385**
$P^3\alpha$	**0.2146**	**0.1395**	**0.3211**	**0.1737**	0.4442	**0.2049**
$RP^3\beta$	**0.2202**	**0.1431**	**0.3323**	**0.1793**	**0.4667**	**0.2132**
$EASE^R$	**0.2349**	**0.1557**	**0.3419**	**0.1902**	**0.4617**	**0.2205**
SLIM BPR	-	-	-	-	-	-
SLIM ElasticNet	**0.2330**	**0.1535**	**0.3475**	**0.1904**	**0.4799**	**0.2238**
MF BPR	0.1557	0.1024	0.2421	0.1302	0.3599	0.1598
MF FunkSVD	0.1728	0.1121	0.2621	0.1409	0.3727	0.1688
PureSVD	**0.2011**	**0.1307**	0.3002	**0.1626**	0.4238	0.1938
NMF	0.1817	0.1172	0.2824	0.1496	0.4090	0.1815
iALS	**0.2048**	**0.1348**	**0.3080**	**0.1680**	0.4319	**0.1993**
ConvNCF	0.1947	0.1250	0.3059	0.1608	0.4446	0.1957

Table 5.17: *Experimental results for ConvNCF for the Gowalla dataset. $EASE^R$ and SLIM BPR results are missing because the code required too much memory on this dataset.*

	@5		@10		@20	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.2188	0.1652	0.2910	0.1884	0.3803	0.2110
UserKNN CF cosine	**0.7131**	**0.5879**	**0.7939**	**0.6142**	0.8532	**0.6293**
ItemKNN CF tversky	**0.7047**	**0.5864**	0.7790	**0.6105**	0.8331	**0.6244**
$P^3\alpha$	**0.6926**	**0.5703**	0.7674	**0.5948**	0.8158	**0.6071**
$RP^3\beta$	**0.6836**	**0.5525**	0.7723	**0.5814**	0.8361	**0.5976**
$EASE^R$	-	-	-	-	-	-
SLIM BPR	-	-	-	-	-	-
SLIM ElasticNet	0.6365	**0.5284**	0.7083	0.5517	0.7608	0.5651
MF BPR	0.6376	0.4996	0.7416	0.5334	0.8234	0.5542
MF FunkSVD	0.6029	0.4592	0.7216	0.4979	0.8082	0.5199
PureSVD	0.5653	0.4482	0.6627	0.4798	0.7393	0.4993
NMF	0.5856	0.4607	0.6842	0.4927	0.7674	0.5138
iALS	0.6460	0.5081	0.7554	0.5436	0.8356	0.5641
ConvNCF	0.6702	0.5233	0.7799	0.5590	**0.8623**	0.5799

ConvNCF is slightly more competitive. In all but one measurement, however, it is outperformed by the UserKNN method. Interestingly, on this dataset the neighborhood-based baselines provide better recommendation quality than the simple non-neural machine learning methods.

In terms of computational complexity, on Gowalla most neighborhood-based baselines train in less than 1 minute and most non-neural machine

learning baselines train 1 hour with only one requiring 5 hours, while ConvNCF requires more than 12 hours. The number of recommendation lists generated per second is lower than most methods but is higher than PureSVD and NMF, this can be attributed to the very high number of latent factors they have. On Yelp the slowest baseline requires 1 hour of training while ConvNCF requires 3 hours and generates less recommendations per second than other baselines.

5.10 Leveraging Meta-path based Context (MCRec)

Meta-path based Context for top-n recommendations was proposed in [60] at KDD '18. MCRec is a hybrid method that uses side information of the items to build a *heterogeneous information network* (HIN). A HIN may consist of multiple types of nodes and links, allowing great flexibility in modeling the available auxiliary information, for this reason they have been proposed as a general information modeling method. Meta-paths are fixed sequences of node types used to guide random walks, and are widely used to extract structural features that capture relevant semantics which can be used in recommendations. MCRec also includes a priority-based sampling technique to identify the more informative paths as well as and a novel co-attention mechanism to improve the representations of meta-path based context, users and items. Several variants of MCRec are reported:

MCRec$_{rand}$: It employs the random meta-path guided sampling strategy for path generation.

MCRec$_{avg}$: It employs the naive context embedding strategy for meta-paths.

MCRec$_{mp}$: It only reserves the attention components for meta-paths and removes the attention component for users and items.

MCRec: It is the complete MCRec model.

5.10.1 Datasets and Evaluation

The evaluation in the original paper is based on three datasets:

MovieLens100K: Is a widely used movie rating dataset, it includes side information like the genre of the movie.

LastFM: Contains listening records of users, it includes as side information the artists of the tracks.

YelpBusiness: The dataset is gathered from Yelp, a search service powered by crowd-sourced reviews about local businesses, it includes as side information the city and category of the business.

All datasets are transformed into implicit data by setting all interactions values to 1.

MCRec is evaluated with a test data built by a random holdout of 20% interactions, while the validation data contains 10%. For each positive item in the test set, 50 negative test interactions are randomly selected, that will be ranked together with the positive items. Precision, Recall, and the NDCG are then used as evaluation measures at a cutoff length of 10 and the averaged results of ten random splitting is reported.

As baselines, the authors use ItemKNN, two collaborative MF methods (one based on BPR loss, the other based on cross entropy loss), two hybrid MF methods, two methods designed for metadata networks, and a number of variants of their own method. It is stated that for the MF and the NeuMF method, hyperparameters and configuration are taken from the original papers. Other baselines are reported to be systematically optimized on the validation set.

5.10.2 Methodological considerations

The first methodological issue we found is the lack of motivation for the evaluation protocol used by MCRec, which is very uncommon in two ways. First, ranking positive items with negative items is a rather common evaluation procedure, but that is predominantly associated to a leave-one-out test split, rather than a random holdout (see Section 7.2.1, Table 7.3). MCRec is the only instance we could observe where the negative sampling is combined with a random holdout. Secondly, the number of negative items is uncommonly low, being 50, when compared to the usual 1000 or 100. While the selection of a specifically crafted evaluation protocol is certainly not an issue per-se, as we will discuss in Chapter 7 the lack of justification conveys a sense of arbitrariness. Further in the original article, it is stated that the results are the average of ten random splitting of the data, but no standard deviation is reported, which is *crucial* to correctly interpret the results.

By analyzing the source code we could observe that the NDCG measure was implemented in a non-standard way. The "ideal" NDCG was computed not by using all test items, but rather only based on the successfully recommended items. This means that the Ideal NDCG is lower than it should and the values reported in the paper are much higher than the ones obtained by us using a standard implementation of NDCG. Furthermore, we could

observe that the original source code reports the maximum value each metric reaches, when evaluated on the test data, regardless of the epoch. In our experiments, again, we report the values of the metrics after the optimal number of epochs is chosen with early stopping on the validation set. Lastly, for the MF and the NeuMF methods it is stated that hyperparameters and configurations are taken from the original papers, which means those baselines were used in a suboptimal configuration and the results obtained cannot be considered a reliable representation of the baseline performance.

5.10.3 Results and Discussion

The publicly available source code is specifically tailored for the MovieLens dataset because the meta-path information is hard-coded, therefore it cannot be used for the other datasets and no specific implementation for those is available. Furthermore, at the time of writing, the code for preprocessing the Yelp and Last.fm was not available. Due to this we only perform our evaluation for the MovieLens dataset.[20] The results are reported in Table 5.18. We can see that MCRec is outperformed both by the traditional neighborhood-based methods, the more complex non-neural machine learning methods and one of our simple hybrids. Our pure content-based baseline led to generally weak results, lower than the non-personalized TopPopular, and was also outperformed by MCRec.

Table 5.18: *Experimental results for MCRec for the MovieLens dataset.*

	PREC@10	REC@10	NDCG@10
TopPopular	0.1907	0.1180	0.1361
UserKNN CF dice	**0.3442**	**0.2237**	**0.2692**
ItemKNN CF asymmetric	**0.3320**	**0.2171**	**0.2601**
$P^3\alpha$	**0.3305**	0.2081	**0.2554**
$RP^3\beta$	**0.3435**	**0.2191**	**0.2588**
$EASE^R$	**0.3739**	**0.2430**	**0.2905**
SLIM BPR	**0.3127**	0.2040	0.2460
SLIM ElasticNet	**0.3770**	**0.2441**	**0.2957**
MF BPR	0.2816	0.1860	0.2195
MF FunkSVD	**0.3442**	**0.2203**	**0.2642**
PureSVD	**0.3545**	**0.2247**	**0.2719**
NMF	**0.3350**	**0.2139**	**0.2585**
iALS	**0.3596**	**0.2283**	**0.2759**
ItemKNN CBF cosine	0.0455	0.0185	0.0254
ItemKNN CFCBF cosine	**0.3398**	**0.2239**	**0.2646**
MCRec	0.3110	0.2113	0.2466

[20]The preprocessing code for MovieLens was made available later by the authors on their GitHub repository.

By analyzing the computational cost of MCRec we can observe it is substantially slower than all baselines. All neighborhood-based baselines are able to complete their training in fractions of a second, while non-neural machine learning baselines require less than 3 minutes. MCRec instead takes more than 2 hours. Furthermore MCRec is between 100 and 200 times slower in generating recommendation lists.

5.11 Collaborative Memory Network for Recommendation System (CMN)

Collaborative Memory Networks (CMN) was proposed in [42] at SIGIR '18. CMN's main idea is to unify two families of collaborative filtering methods, latent factor models and neighborhood models, using a non-linear architecture. The neighborhood component is built by fusing a memory component and a neural attention mechanism, the associative addressing scheme of the memory module acts as a nearest neighborhood model identifying similar users. The attention mechanism learns an adaptive nonlinear weighting of the user's neighborhood based on the specific user and item. The output module exploits nonlinear interactions between the adaptive neighborhood state jointly with the user and item memories to derive the recommendation. Stacking multiple memory modules together (*hops*) allows to build deeper architectures capturing increasingly more complex user-item relations. CMN is associated to different variants according to the number of hops, from 1 to 3.

5.11.1 Datasets and Evaluation

The evaluation in the original paper is based on three datasets:

Epinions: Is an online service to share product feedback and reviews.

CiteULike-a: Is a dataset collected from CiteULike, which is an online service providing users with a digital catalog to save and share academic papers.

Pinterest: Refers to the well known social network which allows users to save or pin an image to their board.

All interactions are binarized and associated to the value of 1. The training-test splits for CiteULike and Pinterest are provided by the authors.

The test data is built via leave-one-out, sampling a random interaction per each user. If the user rated only one item, this interaction is kept in

the training set. To evaluate the algorithm, for each user 100 unobserved (negative) items are sampled and ranked together with the positive item. The Hit Rate and the NDCG at a cutoff length of 10 are used as evaluation metrics.

As baselines, the authors include both simple, non-neural, and neural methods in their experiments: ItemKNN, BPR matrix factorization, SVD++, two variants of NCF [57] (see Section 5.3), and the Collaborative Denoising Auto Encoder. Hyperparameters are tuned on a validation set.

5.11.2 Methodological considerations

It is possible to see from the original paper that details for the hyperparameter optimization process are provided for the proposed model, but not for the baselines. The number of training epochs is not mentioned in the paper and in the source code it is set to a fixed value, therefore the epochs selection criteria adopted in the original experiments is not transparent. In our evaluation, we applied early stopping as per the described methodology.

5.11.3 Results and Discussion

We could reproduce the results of the paper using the data splits provided by the authors for two datasets, CiteULike and Pinterest. We created the split for Epinions using the information provided in the paper. However, in that case, we could not reproduce the original results. For the CMN method, we report the results for the *CMN-3* variant, which led to the best results.

We report the results for Pinterest dataset in Table 5.19 and for the Epinions dataset in Table 5.20.

For the Pinterest dataset is possible to observe that CMN leads to equivalent results with respect to some baselines but is always slightly outperformed by both neighborhood-based and non-neural machine learning baselines. On the CiteULike dataset, the main observations are the same. On this smaller dataset, however, the performance of CMN is often much lower than the one achieved by neighborhood-based methods and non-neural machine learning techniques.

For the Epinions dataset, whose results are reported in Table 5.19, CMN is indeed able to outperform all personalized baselines. However, the non personalized TopPopular method consistently led to the best values across all measurements by a huge margin. In order to explain this phenomena, we compute the Gini Index (not the Gini Diversity) of the popularity of

Table 5.19: *Experimental results for CMN for the Pinterest dataset.*

	@5		@10	
	HR	NDCG	HR	NDCG
TopPopular	0.1665	0.1064	0.2740	0.1409
UserKNN CF asymmetric	0.7005	**0.5037**	0.8630	**0.5567**
ItemKNN CF cosine	**0.7132**	**0.5116**	**0.8781**	**0.5653**
$P^3\alpha$	0.6990	**0.5034**	0.8596	**0.5559**
$RP^3\beta$	**0.7147**	**0.5150**	**0.8772**	**0.5680**
$EASE^R$	**0.7072**	**0.5129**	0.8567	**0.5617**
SLIM BPR	**0.7120**	**0.5151**	**0.8733**	**0.5678**
SLIM ElasticNet	**0.7084**	**0.5107**	**0.8683**	**0.5628**
MF BPR	0.6924	0.4886	**0.8694**	0.5463
MF FunkSVD	**0.7088**	**0.5037**	**0.8686**	**0.5559**
PureSVD	0.6619	0.4721	0.8146	0.5219
NMF	0.6550	0.4618	0.8287	0.5183
iALS	**0.7219**	**0.5175**	**0.8677**	**0.5652**
CMN	0.7013	0.5005	0.8674	0.5547

each item in all datasets, the higher the Gini Index the more unbalanced the distribution will be and therefore the higher the popularity bias. For the CiteULike dataset the Gini Index is 0.37, for Pinterest is 0.45 and for Epinions is 0.69. This means that the datasets have an item distribution which is increasingly skewed to the extent that, for Epinions, personalized recommendations are very difficult to make. It can be observed how the competitiveness of CMN against the simple personalized baselines is positively correlated with the strength of the popularity bias of the dataset. On the least unbalanced dataset, CiteULike, CMN performs poorly while on the most unbalanced dataset, Epinions, CMN is able to outperform all personalized baselines. Considering diversity metrics, on Epinions CMN has an item coverage of only 55%, compared with 78% of most personalized baselines and 14% of the best performing non-personalized baseline, while on CiteULike and Pinterest the Item Coverage of CMN is slightly higher than most baselines, while the Gini Diversity is comparable. Overall, CMN appears to be highly affected by the popularity bias of the dataset.

In terms of computation time on the Epinions dataset, we can see that most neighborhood-based baselines have a training time in the range of a few minutes, non-neural machine learning baselines require under 3 hours while CMN requires 9 hours. SLIM BPR in this case is the slowest algorithm, this is explained by the fact that the similarity matrix would have been too big to fit into memory and therefore it was modeled as a sparse data structure, which considerably slows the training process.

Table 5.20: *Experimental results for CMN for the Epinions dataset. EASE^R results are missing because the code required too much memory on these datasets.*

	Epinions			
	@5		@10	
	HR	NDCG	HR	NDCG
TopPopular	**0.5492**	**0.4204**	**0.6672**	**0.4587**
UserKNN CF asymmetric	0.4294	0.3642	0.4767	0.3795
ItemKNN CF cosine	0.4309	0.3584	0.4854	0.3760
P$^3\alpha$	0.4008	0.3411	0.4389	0.3533
RP$^3\beta$	0.3928	0.3329	0.4341	0.3462
EASE^R	-	-	-	-
SLIM BPR	0.3988	0.3393	0.4422	0.3533
SLIM ElasticNet	0.4133	0.3471	0.4667	0.3643
MF BPR	0.4668	0.3662	**0.5594**	0.3962
MF FunkSVD	**0.5427**	**0.4196**	**0.6567**	**0.4566**
PureSVD	0.4073	0.3069	0.5045	0.3384
NMF	0.4055	0.3218	0.4951	0.3508
iALS	0.0519	0.0316	0.1003	0.0470
CMN	0.4699	0.3781	0.5399	0.4008

5.12 Spectral Collaborative Filtering (SpectralCF)

Spectral Collaborative Filtering was proposed by [134] at RecSys '18. SpectralCF represents users and items as nodes in a bipartite graph. The novelty of this method is a convolution approach which operates on the *spectral domain*, this it is claimed allows to consider both proximity and connectivity information in the graph, which is assumed to be particularly helpful for cold-start problems. SpectralCF exploits the eigenvalues of the laplacian matrix of the graph and stacks multiple convolution layers.

5.12.1 Datasets and Evaluation

The evaluation in the original paper is based on three datasets:

MovieLens1M: Is a widely used movie rating dataset with values in range 1-2 and all users have at least 20 interactions.

HetRec: The dataset has been released by the Second International Workshop on Information Heterogeneity and Fusion in Recommender Systems[21]. It is based upon MovieLens10M.

Amazon Instant Video: The dataset consist of ratings for the *Amazon.com* video platform[22].

[21] http://ir.ii.uam.es/hetrec2011/

[22] http://jmcauley.ucsd.edu/data/amazon/

As preprocessing all ratings lower than 5 are removed and the remaining ones binarized with a value of 1, then users associated to less than 5 interactions are removed. After preprocessing, the MovieLens1M dataset is reduced to one fifth of the original size (226k interactions). The other datasets are even smaller (71k and 22k interactions, respectively). The data split for the MovieLens1M dataset is provided online by the authors.

In the original paper SpectralCF is evaluated under two different scenarios: a regular one and a cold-start setup. For the main scenario the test data is built by sampling randomly 20% of the interactions of each user. The evaluation is performed 5 times and the average of the results as well as the standard deviation are reported. Recall and MAP at different list lengths are used as metrics for this scenario.

In the cold-start scenario, the training set is built with different degrees of sparsity by varying the number P of interactions associated with each user, where P is varied from one to five. The remaining items associated with each users are used as test set. Recall@20 and MAP@20 are used in this scenario for evaluation.

As baselines, the authors consider ItemKNN, BPR matrix factorization, iALS, NCF [57] (see Section 5.3) and two graph-based methods, GNMF and GCMC, originally designed for explicit feedback datasets. The hyperparameters are tuned on a validation set and the parameter ranges are reported in the paper.

5.12.2 Methodological considerations

Looking at the original article we could observe some common issues, e.g., for NCF was used the configuration reported in the original paper, only one set of hyperparameters for SpectralCF is reported rather than one for each dataset.

By analyzing the provided data splits for MovieLens, we observed another very significant problem. Our experiments show that SpectralCF has very strong recommendation quality on the training-test split of MovieLens provided by the authors, but has very weak performance on the HetRec and Amazon Video datasets. This led us to further investigate the provided data splits. While we confirmed that the 80/20 interaction quota was met, the items in the test data had a very different popularity distribution with respect to the training data. This led us to conclude the split provided by the original authors was unlikely the result of the procedure described in the paper.

In order to illustrate the data splitting problem, we compare the popular-

ity distribution of the items in the training and the test split in the provided data, see Figure 5.1b. The figure plots the number of interactions each item received (i.e., its popularity) in both training and test data, normalized by setting to 1 the most popular item. The items are ranked in decreasing popularity according to the training data.

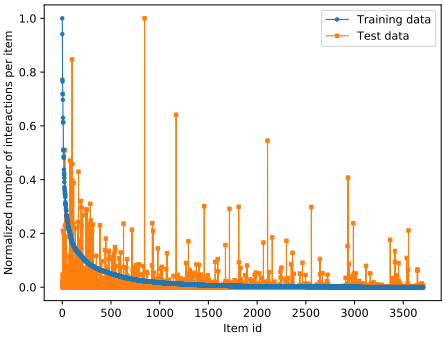
A truly randomized split will preserve the distribution of the data and therefore should produce a test popularity distribution that closely mirrors the training distribution. However, the figure shows the split provided by the authors exhibits a very different behavior with several items having low popularity in the training data appearing surprisingly often in the test data, while highly popular training items appear rarely present in the test data. Figure 5.1b shows the popularity distributions of our random split, which are almost identical between training and test sets.

Besides the visual analysis, we also computed numerical statistics on the item popularity values computed in the various splits, i.e., Gini Index, Shannon entropy and Kendall Tau, see Table 5.21. In a true random split, the Gini Index should be close to the value obtained on the non-split data (in this case 0.78) for both the training and test splits. However, the Gini Index of the test split provided by the authors is much higher (0.92). This indicates that the provided test split has a much higher popularity bias than it should. A similar consideration applies for the Shannon entropy: the entropy of the original dataset is close to 10 and is similar to the entropy of our random training-test split. However, the provided test split has a lower entropy (8.5). Lastly we report the Kendall Tau metric, which counts the number of pairwise agreements between two ranking lists, the lists are the item indices ranked according to their popularity in the given data split (i.e., train, test). Its result is the percentage of item pairs whose ordering is consistent between the two splits, i.e., one item is more popular than the other in both training and test. In our training-test split the Kendall Tau of the training data when compared to the full dataset is almost 1 and that of the test data is 0.85, while Kendall Tau of the publicly available test data is only 0.44 which means the majority of items have different relative rankings in their popularity between training and test data. Overall, these metrics highlight how inconsistent is the original test data with respect to the original training data, strongly pointing to errors in the splitting procedure adopted in the original article.

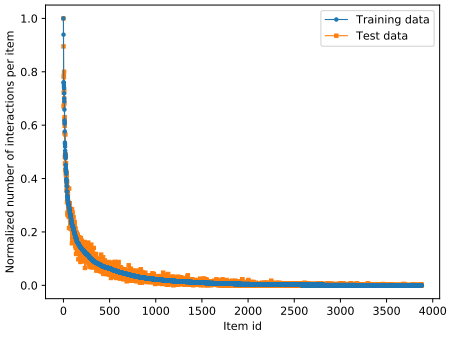
Due to the anomalous nature of the test data, we created a new data split ourselves following the procedure described in the paper. Since we had to create a new training-test split with different properties than what provided by the authors, we could not use the hyperparameters provided by the orig-

	Max pop	Avg pop	Gini Index	Kendall Tau	Shannon
	MovieLens original split				
Full data	1963.00	61.08	0.78	1.00	9.99
Training data	1936.00	48.37	0.79	0.87	9.89
Test data	1361.00	12.71	0.92	0.44	8.49
	MovieLens our split				
Full data	1963.00	58.08	0.79	1.00	9.99
Training data	1575.00	46.44	0.79	0.97	9.99
Test data	388.00	11.64	0.80	0.85	9.93

Table 5.21: *Statistics for the popularity bias of the two splits of MovieLens1M, one generated by us according to what described in the paper and the other provided by the authors. See Figure 5.1.*



(a) *Training and test splits provided by the original authors.*



(b) *Training and test splits generated by us following what reported in the original paper.*

Figure 5.1: *Normalized popularity distributions of the training and test splits for SpectralCF, the value 1 corresponds to the most popular item in that split. For a random split, as can be seen in Figure 5.1b, the normalized values of both splits are, on average, similar. In the split provided by the original authors, however, as can be seen in Figure 5.1a, training and test data have quite different distributions.*

inal paper for SpectralCF, therefore we ran a hyperparameter tuning step. We applied the same tuning algorithm and protocol as per all baseline algorithms, the hyperparameters we tuned are reported in Table 5.22. In order to ensure the hyperparameter range and distribution we selected was adequate, we compared the results of our optimization on the publicly available data split, and we could obtain results slightly better than the ones reported in the original article, as reported in Table 5.23. This result confirms that our hyperparameter optimization strategy for SpectralCF is effective and that we are not inadvertently penalizing the algorithm.

Table 5.22: *Hyperparameter values for SpectralCF*

Algorithm	Hyperparameter	Range	Type	Distribution
SpectralCF	batch size	128, 256, 512, 1024, 2048	Categorical	
	embedding size	4, 8, 16, 32	Categorical	
	k	1 - 6	Integer	uniform
	decay	10^{-5} - 10^{-1}	Real	log-uniform
	learning rate	10^{-5} - 10^{-2}	Real	log-uniform

Another issue we discovered with SpectralCF concerns the computation of the eigenvalues, which sometimes are complex. How complex eigenvalues should be handled is not discussed in the paper and by looking at the source code we can see that the complex part is simply discarded (which results in warnings from the mathematical software library). We argue that since complex eigenvalues can occur for the computations required by SpectralCF, how to handle those cases should have been discussed and motivated.

Lastly, during our experiments we noticed SpetralCF exhibits some numerical instability, as sometimes the loss function gets to infinite or to *not-a-number*, either at the very first epoch or during training. In those cases the training of the model must be stopped and resumed from scratch.

5.12.3 Results and Discussion

The source code and the data split for MovieLens are available online. We were able to reproduce the results obtained by the authors based on the provided code and following the information in the paper. However, as previously discussed, the data split provided by the authors is probably the result of an erroneous splitting procedure.

The results obtained for the MovieLens1M dataset are shown in Table 5.23. The results show that SpectralCF is outperformed by all baselines in our comparison often exhibiting a recommendation quality equal to the

TopPopular method. The observations for the other datasets are similar, the full results are available Appendix C.

Table 5.23: *Experimental results for SpectralCF for the MovieLens dataset. SpectralCF is reported both using the hyperparameters as reported in the original article (original) and the ones we found ourselves (ours) with a Bayesian Search as reported in Table 5.22.*

	@20		@40		@60		@80		@100	
	REC	MAP	REC	MAP	REC	MAP	REC	MAP	REC	MAP
TopPopular	**0.1892**	**0.0584**	**0.2788**	**0.0636**	**0.3356**	**0.0666**	**0.3834**	**0.0687**	0.4226	**0.0702**
UserKNN CF jaccard	**0.3001**	**0.1201**	**0.4134**	**0.1285**	**0.4901**	**0.1335**	**0.5457**	**0.1367**	**0.5884**	**0.1388**
ItemKNN CF asymmetric	**0.2876**	**0.1134**	**0.4000**	**0.1213**	**0.4768**	**0.1263**	**0.5367**	**0.1295**	**0.5820**	**0.1317**
$P^3\alpha$	**0.2939**	**0.1141**	**0.4150**	**0.1233**	**0.4900**	**0.1285**	**0.5463**	**0.1318**	**0.5903**	**0.1342**
$RP^3\beta$	**0.2737**	**0.1044**	**0.3879**	**0.1124**	**0.4664**	**0.1173**	**0.5234**	**0.1206**	**0.5726**	**0.1230**
$EASE^R$	**0.2907**	**0.1215**	**0.3622**	**0.1243**	**0.3994**	**0.1251**	**0.4238**	**0.1255**	**0.4467**	**0.1258**
SLIM BPR	**0.2886**	**0.1086**	**0.4048**	**0.1170**	**0.4813**	**0.1219**	**0.5362**	**0.1249**	**0.5782**	**0.1269**
SLIM ElasticNet	**0.3069**	**0.1265**	**0.4246**	**0.1356**	**0.5010**	**0.1410**	**0.5564**	**0.1443**	**0.6001**	**0.1466**
MF BPR	**0.2616**	**0.0956**	**0.3662**	**0.1028**	**0.4377**	**0.1071**	**0.4890**	**0.1097**	**0.5307**	**0.1116**
MF FunkSVD	**0.2684**	**0.0875**	**0.3890**	**0.0963**	**0.4663**	**0.1015**	**0.5252**	**0.1049**	**0.5720**	**0.1072**
PureSVD	**0.2595**	**0.1008**	**0.3638**	**0.1083**	**0.4378**	**0.1131**	**0.4913**	**0.1161**	**0.5347**	**0.1182**
NMF	**0.2384**	**0.0908**	**0.3351**	**0.0972**	**0.4032**	**0.1014**	**0.4568**	**0.1041**	**0.4981**	**0.1060**
iALS	**0.3033**	**0.1183**	**0.4201**	**0.1273**	**0.4933**	**0.1326**	**0.5493**	**0.1360**	**0.5925**	**0.1383**
SpectralCF (ours)	0.1813	0.0533	0.2643	0.0581	0.3274	0.0613	0.3823	0.0635	0.4261	0.0651
SpectralCF (original)	0.1785	0.0540	0.2590	0.0586	0.3232	0.0614	0.3689	0.0632	0.4101	0.0646

Table 5.24: *Experimental results for beyond-accuracy metrics for SpectralCF on the selected MovieLens1M dataset at cutoff 50.*

	MovieLens1M				
	Div. MIL	Div. HHI	Cov. Item	Div. Gini	Div. Shannon
TopPopular	0.1855	0.9837	0.0322	0.0150	6.0329
UserKNN CF jaccard	**0.6718**	**0.9934**	**0.3830**	**0.0484**	**7.9176**
ItemKNN CF asymmetric	**0.6214**	**0.9924**	**0.5621**	**0.0452**	**7.7170**
$P^3\alpha$	**0.6824**	**0.9936**	**0.3449**	**0.0476**	**7.9227**
$RP^3\beta$	**0.6343**	**0.9927**	**0.6430**	**0.0492**	**7.7865**
$EASE^R$	**0.7258**	**0.9945**	**0.2777**	**0.0519**	**8.0749**
SLIM BPR	**0.6465**	**0.9929**	**0.3220**	**0.0424**	**7.7517**
SLIM ElasticNet	**0.7481**	**0.9950**	**0.3290**	**0.0599**	**8.2649**
MF BPR	**0.6152**	**0.9923**	**0.4088**	**0.0425**	**7.6973**
MF FunkSVD	**0.8022**	**0.9960**	**0.3006**	**0.0878**	**8.7693**
PureSVD	**0.7689**	**0.9954**	**0.1383**	**0.0520**	**8.0705**
NMF	**0.7306**	**0.9946**	**0.1069**	**0.0447**	**7.8312**
iALS	**0.8324**	**0.9966**	**0.2326**	**0.0772**	**8.6570**
SpectralCF (ours)	0.2217	0.9844	0.0425	0.0155	6.1331

Since, based on our results, SpectralCF achieves similar recommendation quality as a TopPopular recommender, we wanted assess whether SpectralCF was at least able to personalize its recommendations (i.e., providing, to an extent, different recommendation lists to different users), for this experiment we also include beyond-accuracy metrics. From Table 5.24

we can see how SpectralCF exhibits a diversity comparable with the non-personalized TopPopular algorithm on all metrics. SpectralCF only explores 4% of the available items and, according to MIL, given any two users most of their recommendation lists will be identical. The Gini Diversity of SpectralCF is too equivalent to a TopPopular, meaning that the algorithm recommends few items very frequently. Overall, in this experimental scenario, it appears that SpectralCF simply behaves like a non-personalized model.

In terms of computation time , the training time of all neighborhood-based baselines is complete in slightly more than 3 seconds, while the non-neural machine learning baselines require at most 6 minutes. SpectralCF, on the other hand, requires more than 40 minutes. Furthermore the algorithm has high memory requirements, the eigenvector matrix may easily exceed 1GB for MovieLens1M.

CHAPTER 6

The Claimed Value of Convolutions over User-Item Embedding Maps

In the previous chapters the analyzed articles have been compared with simple but well optimized baselines, challenging their claimed state-of-the-art performance. In this Chapter, we provide a different type analysis which instead questions the assumption some of them rely upon. One specific idea, recently put forward by several researchers, is to consider potential correlations between the latent factors (embeddings) by applying convolutions over the user-item interaction map. However, such interaction maps do not share the properties of images where Convolutional Neural Networks (CNNs) are particularly useful. In this Chapter, we show both through analytical considerations and empirical evaluations that the claimed gains reported in the recent literature cannot be attributed to the ability of CNNs to model embedding correlations, as argued in the original papers. On a more general level, we point to methodological issues in how claims are demonstrated within the research field.

6.1 Background

As we could observe in our review of the current state-of-the-art of neural recommender approaches, researchers often try to use certain architectural components that were proven successful in other application areas of neural networks and apply them to the recommendation problem. Examples are attention layers, autoencoders and convolution layers [73, 132]. In particular, Convolutional Neural Networks (CNNs), have been successfully applied to different recommendation-related tasks, including image or text feature extraction for content-based models, sequential recommendations [109] or collaborative filtering [132].

In some recent works, different proposals were put forward to combine the proven power of low dimensional approximation models (e.g., latent factor models) with CNNs. In particular, one underlying idea of the proposals by [41, 56, 123, 131] is to use CNNs to model and leverage correlations in the latent factors (embeddings) space. All these papers claim to have obtained significant gains in accuracy by applying convolutions over user-item interaction maps derived from the outer-product of user-item embeddings.

Some of these articles argue that the interaction maps can be viewed as analogous to images, an application area in which CNNs are very effective. However, user-item interaction maps, as produced by common latent space approaches, do not share the properties of images. For most common latent space models, there is no natural order of the elements in the latent vectors, the dimensions are independent, and the correlation between the latent dimensions is not modeled. Therefore, it is more than surprising that the above-mentioned CNNs approaches were able to benefit from detecting correlations between the dimensions.

In this Chapter, we therefore critically examine the progress claimed in these papers, based on both theoretical considerations and alternative experimental evaluations. This study is different to what previously done in Chapter 5, in that we do not simply compare the neural approaches against other baselines, but we critically examine whether the claims made in the original paper were indeed supported. To this end, we also report the results of ablation studies that were not present in the original papers. These results indicate that the proposed CNN-based models, besides not being competitive against simple baselines, do *not* capture correlations between the embedding dimensions as claimed in the original papers. Rather, they merely act as a non-linear function of their element-wise product and the removal of the embedding correlations leads to no significant effects.

Furthermore, our work indicates that sometimes the experimental evaluations are not suited to demonstrate the articles' claims regarding which part of a complex architecture actually contributes to an observed performance gain.

6.2 Principles and Assumptions of CNNs

As described in Chapter 2, a convolutional neural network is a multilayer feed-forward neural network that was originally developed to address image recognition problems [130]. Unlike other types of neural networks, CNNs were therefore designed for certain types of inputs, i.e., images, which have a specific topology. With data exhibiting these properties, local features (e.g., lines, corners) emerge from their respective immediate surroundings, regardless of their absolute position in the data.

Convolution is usually applied on a *local area* (i.e., the kernel size) of a two-dimensional map of a certain size. Identifying the *local area* on which to apply the convolution requires a definition of *proximity* between points. Depending on the use case, different definitions of proximity may be used. In case of images, the proximity is defined in spatial terms, i.e., pixels that are close in the image will be perceived as close by an observer and are therefore meaningful for the reconstruction of more complex patterns. Other definitions of proximity have also been developed for non-Euclidean data like social networks or knowledge graphs [37].

6.2.1 CNNs on Embedding correlations

As previously described in Chapter 2, a growing number of papers aim to use CNNs for collaborative filtering tasks. In this study we will focus on algorithms applying CNNs on embeddings and only using a user-item rating matrix as an input.

If we compare images or graph data to embeddings, in the specific form of latent factors derived from a user-item rating matrix, there is a key difference. For images or graphs, there is a "natural" way of defining the local area, e.g., based on the distance in pixels or the number of hops in the graph. The corresponding proximity measure has a strong relation with the data semantics.

However, the papers we analyzed, i.e., [41, 56, 123, 131], do not clearly provide a definition of locality for the embeddings, nor do they describe the semantic topology of the input data. In the ConvNCF approach [56] (see Section 5.9), for example, the input to the CNN is a user-item interaction map that is created by computing the outer product of embeddings

pretrained using matrix factorization. While the articles argue that this map is analogous to an image and therefore the use of CNN is justified, it is not demonstrated or discussed in detail what the resulting map topology should represent and whether it possesses typical image properties (e.g., spatial locality and translation invariance). Technically, the interaction map created by the outer product of the embeddings in the described approach contains two components: (i) the main diagonal that represents the element-wise product between two embedding vectors and (ii) the off-diagonal elements that capture embeddings correlations. Unfortunately, none of the analyzed papers measured and compared the contribution of the two components to the proposed CNN model. In all papers it is claimed claim that the CNN is able to model the correlations between embeddings based on the fact that CNN models outperform other models that are not using convolutions. This, we argue, is not sufficient. Comparing models with vastly different structures means that a multitude of factors may have an impact on the results. Moreover, as we have shown multiple times in Chapter 5, the baselines reported are very often weak or inadequately optimized, therefore the evaluation protocol does not allow to draw meaningful conclusions. Ultimately, it is not entirely clear from the provided experiments how much the correlations between embeddings, as modeled by the CNN, actually contributed to the observed performance gains.

6.3 Overview of Analyzed Approaches

In this Chapter we examine three recent neural approaches that use convolutions over embeddings derived from a user-item rating matrix. For all approaches the source code was shared by the authors. We identified an additional relevant work [64], which however did not have a reproducible experimental setup as per our definition (see Section 3).

6.3.1 Convolutional Neural Collaborative Filtering

Convolutional Neural Collaborative Filtering (*ConvNCF*) was proposed in [56] (see the analysis in Section 5.9). The ConvNCF model is trained in two steps. First, a matrix factorization model is fitted on the data. Then, for each user-item interaction, the outer product of their embedding is computed, resulting in a two-dimensional *interaction map* on which the CNN is applied.

Following the original notation, let u be a user and i an item, $P \in \mathbb{R}^{M \times K}$ and $Q \in \mathbb{R}^{N \times K}$ the embedding matrix of users and items, respectively; K the embedding size, M the number of users and N the number of items.

Lastly, let $p_u, q_i \in \mathbb{R}^K$ be their respective embeddings. Based on these embeddings, the interaction map $E \in \mathbb{R}^{K \times K}$ is obtained by computing their outer product as follows:

$$E = p_u \oplus q_i = p_u q_i^T$$
$$e_{x,y} = p_{u,x} \cdot q_{i,y} \tag{6.1}$$

In the original paper, pretraining is performed with a MF BPR model [96] and, a subsequent paper extends the pretraining to FISM and SVD++ [41]. As mentioned before, the interaction map is said to be analogous to an image, but there is no deeper discussion of this claim, and the CNN is said to model embedding correlations but no direct measurement of their contribution is provided.

In our previous analysis (see Section 5.9) we could observe several methodological issues in the evaluation and that ConvNCF is never competitive against simple baselines when evaluated as in the original paper.

6.3.2 Convolutional Factorization Machines

Convolutional Factorization Machines (*CFM*) were proposed in [123] as a context-aware model able to overcome the limited modeling capacity of factorization machines [95], which are constrained by a linear representation of feature interactions. Similarly to ConvNCF, CFM applies a convolution operation on the outer product of the embeddings. In the CFM approach, however, the embeddings are obtained from a Factorization Machine via a self-attention layer, which reduces the model's dimensionality. The outer product of the embeddings is computed independently for each context and all interaction maps are stacked on top of each other, forming an interaction cube, on which 3D convolution is applied. If C is the number of contextual features, the interaction maps in the memory cube will be $C(C-1)/2$.

Although the scenario of application is different from ConvNCF and the embeddings are obtained from a different pretraining step, CFM has the same theoretical problems as ConvNCF in that the outer product is treated as if it were an image, without ever demonstrating that this assumption holds. The ability of CFM to model embeddings interaction is again claimed based on it outperforming baselines with quite different architectures, including ConvNCF.

As opposed to ConvNCF and CoupledCF, we did not report the evaluation of CFM in Chapter 5. This is because the article proposing CFM was published in 2019 and therefore does not meet the criteria for being in-

cluded in the previous evaluation study. Nonetheless, in Table 6.1 we report the comparison of CFM against the same baselines we used and with the same evaluation protocol in the other evaluations (see Chapter 4). Based on these results, we can see that CFM is never competitive against simple baselines, by a very large margin, exhibiting on average half of the recommendation quality of both neighborhood-based and non-neural machine learning baseline algorithms. In the original article CFM was compared against TopPopular, a Factorization Machine, NeuMF [57] (see Section 5.3) and a Deep Factorization Machine. In terms of computational time, neighborhood-based baselines complete their training in less than 2 seconds, non-neural machine learning algorithms in between 5 and 44 minutes on a CPU, while CFM requires more than 14 hours on a high-end GPU.

Table 6.1: *Experimental results for CFM for the Last.fm dataset.*

	Last.fm					
	@5		@10		@20	
	HR	NDCG	HR	NDCG	HR	NDCG
TopPopular	0.0016	0.0009	0.0023	0.0011	0.0033	0.0014
UserKNN CF cosine	**0.5964**	**0.4527**	**0.6715**	**0.4773**	**0.7032**	**0.4855**
ItemKNN CF cosine	**0.5975**	**0.4425**	**0.6776**	**0.4689**	**0.7070**	**0.4764**
$P^3\alpha$	**0.6327**	**0.4929**	**0.6744**	**0.5066**	**0.7014**	**0.5135**
$RP^3\beta$	**0.5896**	**0.4458**	**0.6756**	**0.4739**	**0.7071**	**0.4821**
$EASE^R$	**0.6496**	**0.5057**	**0.6884**	**0.5183**	**0.7037**	**0.5223**
SLIM BPR	**0.6569**	**0.5260**	**0.6893**	**0.5365**	**0.7122**	**0.5424**
SLIM ElasticNet	**0.6674**	**0.5169**	**0.6972**	**0.5267**	**0.7102**	**0.5300**
MF BPR	**0.5972**	**0.4432**	**0.6690**	**0.4669**	**0.6974**	**0.4742**
MF FunkSVD	**0.6214**	**0.4702**	**0.6789**	**0.4891**	**0.7024**	**0.4950**
PureSVD	**0.4026**	**0.3117**	**0.4891**	**0.3397**	**0.5652**	**0.3590**
NMF	**0.2971**	**0.2425**	**0.3879**	**0.2716**	**0.5017**	**0.3005**
iALS	**0.6110**	**0.4811**	**0.6735**	**0.5017**	**0.7033**	**0.5093**
CFM	0.2241	0.1485	0.3338	0.1839	0.4661	0.2173

6.3.3 Coupled Collaborative Filtering

Coupled Collaborative Filtering (*CoupledCF*) was proposed in [131] (see the analysis in Section 5.7) as a method to learn implicit and explicit couplings between users and items, taking advantage of them not being independent, to leverage side information more effectively (e.g., user demographics, item features). The model is composed by two cooperating architectures, one is a deep collaborative filtering model which only uses user-item interactions, while the other is a CNN on user and item embeddings whose aim is to learn the couplings.

CoupledCF is different from the other two methods examined here in that the embeddings are not pretrained but are parameters of the model to be learned. The original article shows experimentally that the quality of the model is improved by adding the CNN and this is claimed to demonstrate that the model is effectively learning the couplings. Again, as previously mentioned, the paper does not distinguish between the contribution of the the embeddings correlation and the element-wise product.

In our previous analysis (see Section 5.7) we could observe several methodological issues in the evaluation and that CoupledCF is never competitive against simple baselines when evaluated as in the original paper.

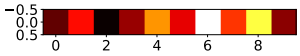


Figure 6.1: *Vector u*

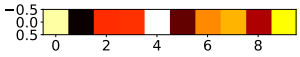


Figure 6.2: *Vector v*

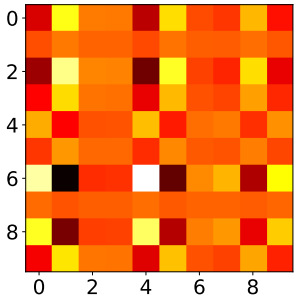


Figure 6.3: *Interaction map*

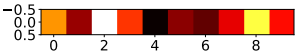


Figure 6.4: *Vector u*

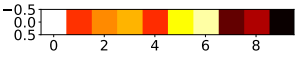


Figure 6.5: *Vector v*

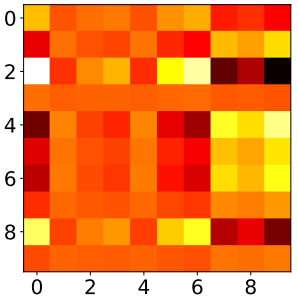


Figure 6.6: *Interaction map*

Figure 6.7: *Effects of permutating the columns of vectors u and v on their resulting outer product (the interaction map). The same vectors under different permutations will generate interaction maps with very different 2D structure.*

6.4 Analysis

One fundamental assumption of the analyzed papers is that the interaction map computed via an outer product is analogous to an image (i.e., exhibits spatial locality and translation invariance). In this section, we demonstrate why this is not the case. We will first discuss this aspect theoretically and

then present the results of two empirical studies. In the first study we show that changing the input topology (i.e., the ordering of the latent factors) does not have an impact on accuracy. If the interaction map had local features a significant drop in accuracy would be expected. The second study consists of two ablation analyses showing that the correlations between embeddings do not provide a statistically significant contribution to the accuracy of the CNN models.

6.4.1 Theoretical Considerations

As previously stated, CNNs were developed to model data that exhibits feature locality and a strong topology. To assess whether CNNs are an appropriate tool to be used on interaction maps, we first have to assess which is the meaning of the topology in an interaction map and why two points are within or outside each others' local area.

	NDCG	HR
MF BPR	0.1576 ± 0.0000	0.2966 ± 0.0000
ConvNCF	0.1623 ± 0.0008	0.3052 ± 0.0019
FM	0.1230 ± 0.0000	0.2234 ± 0.0000
CFM	0.1730 ± 0.0398	0.3155 ± 0.0724

Table 6.2: *Averaged performance results (and standard deviations) obtained for 20 permutations of the interaction maps.*

Consider three cells of an interaction map E, created via Equation 6.1, with coordinates (x, y), $(x, y + 1)$, $(x, y + 4)$. If we consider point (x, y), with a kernel size of 2, $(x, y+1)$ will be in its local area while $(x, y+4)$ will not. But what is the relation between y, $y + 1$ and $y + 4$? If the embeddings are created with a typical latent factor model (e.g, matrix factorization or factorization machines), as done in two of the analyzed papers, the answer is that the latent factors y and $y + 1$ are direct neighbors in the embedding vector, while y and $y + 4$ are not.

The question now is whether the position of the latent factors has a specific meaning. If we look at typical matrix factorization algorithms, such as MF BPR or iALS, we can see that a prediction is computed as follows [96].

$$\hat{r}_{ui} = p_u^T \cdot q_i = \sum_k^K p_{u,k} \cdot q_{i,k} \tag{6.2}$$

Here, $\hat{r}_{ui}$ is the predicted relevance for user u on item i, p_u and q_i are the user and item embedding vectors, and k is the latent factor index.

From Equation 6.2, we see that the ordering of the latent factors has no impact on the prediction, and the model only requires a biunivocal correspondence between the columns of P and Q, regardless of their relative ordering. Such a lack of a *natural ordering* of the latent factors is common to many matrix factorization algorithms including MF BPR, AsySVD, and iALS [61, 96]. Only for some techniques, like PureSVD [33], the latent factors are ordered according to the decreasing singular value they are associated with. Due to this lack of a natural ordering, the specific arrangement of the latent factors is mainly a contingent property and a multitude of equivalent models can be learnt from the same data due to the stochastic nature of the training process. Each permutation of the ordering of the factors leads to an equivalent matrix factorization model but to a different interaction map, which will exhibit different local features. Consider two randomly created vectors u and v of length 10 (Figures 6.1 and 6.2) and two permutations of the same vectors (Figures 6.4 and 6.5), where darker cells indicate higher values. It can be easily seen that the interaction map of the original vectors (Figures 6.3) and the one of the permuted vectors (Figures 6.6) do not have any identifiable pattern in common. This lack of a natural ordering provides evidence that the interaction map is not analogous to an image because it does not exhibit spacial locality (i.e., meaningful local features). This means that the main claim justifying the use of CNNs in the original articles is unsound.

6.4.2 Experiment Configurations

In our two studies, we used the same experimental designs as in the original papers. In particular, we used the original code, data, data splits, as well as hyperparameters that were provided by the authors (see Chapter 4). To determine the number of epochs, which is not usually reported, we apply early stopping.

ConvNCF was evaluated on Yelp dataset. The algorithm code and the data split for the Yelp data was published by the authors. MF BPR was used to pretrain the latent factors.

CFM was tested on Last.fm dataset. The code and the preprocessed data split for Last.fm were provided by the authors. The latent factors were pretrained using a Factorization Machine.

CoupledCF was tested on the MovieLens1M dataset, which also contains side information about users and items. CoupledCF, as stated above, does not use pretrained models but learnable embeddings.

In all original papers the authors use a leave-one-out evaluation procedure. In two cases a number of randomly sampled negative items (e.g., 99 for CoupledCF) were ranked with the true positive. The Hit Rate (HR) and the NDCG are used as evaluation measures in all papers, using different cutoff lengths. Due to the leave-one-out procedure, other metrics like Recall, Precision and F1 are linearly correlated to HR. In our evaluation we applied, for each algorithm, its specific evaluation setting as described in the original paper.

6.4.3 Varying the Input Topology

In our first experiment, we varied the topology of the inputs (i.e., the order of the latent factors) which are fed to the CNN. Given our theoretical considerations from Section 6.4.1, altering the topology should have virtually no impact on the model quality because the topology of the interaction map does not provide any relevant information. To empirically validate this consideration, we designed the following experiment for the approaches that use pretrained embeddings (ConvNCF and CFM).

First, we pre-train the embeddings as done in the original articles, i.e. using either MF BPR or factorization machines. We then create 20 equivalent pre-trained models by randomly permutating the positions of the latent factors. Each permutation will lead to different interaction maps (see Figure 6.7). For each of these permutations, a convolution model is trained, and the quality of each resulting model is evaluated based on the measures used in the original papers (HR and NDCG) at cutoff 10.

The results of this experiments are reported in Table 6.2. We report both (i) the results for the model without the CNN layer (MF BPR and factorization machines) and (ii) the results for the full model (ConvNCF and CFM). For each algorithm, we report the averaged accuracy and the standard deviation resulting from the 20 permutations. The following observations can be made.

- For the "plain" MF BPR and factorization machines models, the standard deviation is zero, as expected by definition from Equation 6.2. However, also for the CNN-based models, the standard deviation is almost zero. This confirms that changing the input topology has no relevant impact on the results, indicating that the order of the latent factors, as expected, does not matter and no local features can exist in the interaction map. Note that statistical tests like the t-test cannot be applied when the variance is zero.

- The CNN models show improved accuracy over the pretraining models, when both are optimized as in the original paper. For the ConvNCF, the gains are tiny, i.e., we could not reproduce in our experiments that the CNN adds much value. For the CFM model, which applies a more complex preprocessing step, improvements over the plain Factorization Machine model *could* be reproduced. These gains, however, cannot be attributed to the fact that the interaction maps can be considered as images (given the irrelevance of the ordering of the "pixels"). To put these observed gains into perspective, we should stress that, as demonstrated in Sections 5.9 and Table 6.1, even despite this improvement, neither ConvNCF nor CFM are competitive against simple baselines.

Algorithm	Mode	Study 1		Study 2	
		NDCG	HR	NDCG	HR
ConvNCF	full	0.1623 ± 0.0008	0.3052 ± 0.0019	0.1623 ± 0.0008	0.3052 ± 0.0019
ConvNCF	element-wise	0.1622 ± 0.0008	0.3051 ± 0.0016	0.1632 ± 0.0012	0.3068 ± 0.0015
ConvNCF	correlations	0.0193 ± 0.0076	0.0403 ± 0.0150	0.1522 ± 0.0013	0.2900 ± 0.0020
CFM	full	0.1730 ± 0.0398	0.3155 ± 0.0724	0.1730 ± 0.0398	0.3155 ± 0.0724
CFM	element-wise	0.1730 ± 0.0398	0.3155 ± 0.0724	0.1805 ± 0.0034	0.3292 ± 0.0062
CFM	correlations	0.0015 ± 0.0003	0.0032 ± 0.0008	0.0011 ± 0.0001	0.0019 ± 0.0002
CoupledCF	full	0.5272 ± 0.0491	0.7865 ± 0.0470	0.5272 ± 0.0491	0.7865 ± 0.0470
CoupledCF	element-wise	0.5404 ± 0.0631	0.7744 ± 0.0994	$\mathbf{0.5763 \pm 0.0059}$	$\mathbf{0.8243 \pm 0.0071}$
CoupledCF	correlations	0.5137 ± 0.0903	0.7822 ± 0.0659	0.5503 ± 0.0343	0.7978 ± 0.0391

Table 6.3: *Results of* Study 1*, evaluating the contribution of each component of the interaction map to a model trained on the full map and* Study 2*, evaluating the contribution of training on different parts of the interaction map. Significant improvements over the full map are in bold.*

6.4.4 Ablation Studies

The results shown in Table 6.2 indicate that, despite not being competitive against several simple baselines, the CNN layer, at least in one of the cases, has a positive effect on the overall performance. According to our theoretical considerations, these gains cannot stem from leveraging correlations in the embeddings as claimed in the papers, but might be merely the result of adding a neural network layer to the pretraining model, which acts as a universal approximator.

Study 1 In order to measure how much the CNN model has learned to represent the embeddings correlation, we designed a novel type of ablation study, which was not part of the original papers. We started from the models previously trained on the full interaction map, but we computed the recommendations using only certain interaction map components.[1]

Remember that the correlations in the embeddings are represented by the elements of the interaction map (Equation 6.1) that are not on the main diagonal. The following configurations were tested:

- *full* : This corresponds to the original setting.
- *element-wise*: Only the element-wise products (i.e., main diagonal) are considered.

$$e_{x,y} = \begin{cases} p_{u,x} \cdot q_{i,y} & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases}$$

- *correlations*: Only the embeddings correlations (i.e., off-diagonal) are used.

$$e_{x,y} = \begin{cases} p_{u,x} \cdot q_{i,y} & \text{if } x \neq y \\ 0 & \text{otherwise.} \end{cases}$$

The results of this ablation study are reported in Table 6.3. They show that there is no statistical difference between the models evaluated by computing the recommendations using the full interaction map and when only using the element-wise product (diagonal elements). The statistical significance of the difference between the observed results with α=0.05 was verified using a paired t-test if the values were normally distributed; and a Wilcoxon signed-rank test otherwise. To assess if the values of the metrics are normally distributed we used both Shapiro-Wilk and Kolmogorov-Smirnov tests [94].

In other words, all models have learned that the off-diagonal correlation elements *are not contributing anything to the overall performance*. Interestingly, when the recommendations are computed using only the embeddings correlation, the observed results for ConvNCF and CFM are lower by at least an order of magnitude. For CoupledCF, the accuracy obtained with the embeddings correlation is similar to the full interaction map so, although the model has learned to use the embeddings correlation, this does not improve the model quality. Overall, the results clearly indicate that the convolutional models are not learning to represent the embeddings correlation when those are pre-trained (i.e., ConvNCF and CFM), and are not

[1]For CoupledCF, which does not use pretrained embeddings, we trained and evaluated the model on 20 random training-test splits.

befitting from them even when they are learnable (CoupledCF), which is in direct contradiction to what stated in the original articles. Furthermore, it should be considered that since a similar ablation study was not present in the original papers, the contribution of the embeddings correlation to the convolution model was never directly measured.

Study 2 In *Study 1*, we observed that training the model on the full interaction map resulted in the embeddings correlation not contributing to improve performance over the simple element-wise product. The models either did not benefit from the added parameters (CoupledCF) or actively learned to ignore the correlations (ConvNCF and CFM).

While in *Study 1* we trained the model on the full interaction map, in *Study 2*, we isolate the different components of the interaction map at an earlier stage, during *the training phase*, i.e., we do not train the network on the full map as in *Study 1*.

In this new experiment, different models are therefore trained from scratch, using only the interaction map component associated with a given configuration (i.e., *full map*, *element-wise*, *correlations*). As a result, a model trained only on the element-wise product will never observe embeddings correlation and vice versa. This should allow to have a clearer idea of how much of each component the convolution algorithms can actually learn to model when the other is not present.

The results of *Study 2* are also reported in Table 6.3. Since the training data (i.e., interaction map) fed to the CNN in *Study 2* and *Study 1* are different, it is expected that the absolute values of the measurements are different. However, we can again observe that the results obtained when training the convolution model on the full interaction map and on the element-wise product are not different to a statistically significant extent for ConvNCF and CFM. The convolution operation is therefore not leveraging the embedding correlations in any effective way. As a result, these correlations can be discarded entirely during the training phase without degrading the model performance. Remarkably, for CoupledCF we can observe that training the model on the element-wise product alone results in significantly better results then using the full map. It should be noted here that CoupledCF does not use pretrained embeddings but learns them during the train process. This makes CoupledCF complementary with respect to ConvNCF and CFM. While the original CoupledCF paper stated that the model was able to leverage the embeddings correlations, our results suggest that the additional parameter space introduces noise actively harming the model quality.

Interestingly for ConvNCF and CoupledCF, different from *Study 1*, we

can see that training the convolution on the embeddings correlation elements alone now yields results that are very close to those obtained when using the full interaction map. This suggests that the pretrained embeddings in ConvNCF do indeed carry some information that can, to an extent, be modeled. Similarly, CoupledCF can learn some correlations although with a rather high standard deviation. However in none of those cases the CNN models are, as previously demonstrated in *Study 1*, able to leverage correlations to improve the accuracy obtained on the element-wise product alone. There can be different reasons for this. First, it might be that the convolution on the full map was only able to model information that was redundant and already captured by the element-wise model. An alternative explanation is that the CNN model did not succeed in hybridizing these two pieces of information in a synergistic way, i.e., it only learned to select the best-performing or the less noisy one.

6.5 Summary

In this Chapter, we have analyzed recently published proposals of using CNNs on the interaction maps obtained from user and item embeddings. We have argued that the original articles lacked a proper discussion on two crucial claims they made: the analogy of embeddings and images, and the ability of CNNs to model embeddings correlations. Our work has shown both through theoretical considerations and through empirically studies that embeddings do not share the topological properties of images. The use of CNNs is therefore not well justified. We have also shown that CNNs, as opposed to what claimed in the original articles, are insensitive to the embeddings correlation and fail to improve over a model only using the element-wise product. While we do not argue that convolution cannot be applied on embeddings, we stress that a deeper understating and theoretical analyses of the semantics that new approaches are claimed to leverage are essential to obtain reliable progress in this field. Similarly, claims regarding the improved modeling capacity of an algorithm cannot be based simply upon its ability to outperform a set of baseline algorithms and datasets whose choice is not well justified. Again, for all the CNN models we could find the original comparison included only poorly optimized baselines. Instead, these aspects should be directly verified via specifically designed experiments.

CHAPTER 7

Result overview and discussion

In this Chapter we will provide an overview of the detailed results reported in Chapter 5 as well as a discussion on their implications along three main dimensions: reproducibility of published research, methodological aspects of the evaluation protocol and scalability. Lastly we discuss some limitations of this study. Overall the experimental evaluation reported in this study required significant computational effort. Considering all baselines and reproducible deep learning algorithms, due to the number of different datasets and preprocessing procedures, we report the recommendation quality of more than 900 models. When taking into account the hyperparameter tuning procedure, 41,000 models were fitted, corresponding to a total computation time of 253 days for a specific Amazon AWS instance.[1]

7.1 Reproducibility

As was previously discussed in Chapter 3, only 50% of the relevant algorithms could be reproduced according to our specific definition of reproducibility. We will now provide a discussion along two main dimensions

[1] The computation time refers to the total instance time for one AWS instance p3.2xlarge, with 8 vCPU, 30GB RAM, and one Tesla V100-SXM2-16GB GPU

which had a strong impact on this result: the availability of the original implementation and the reachability of the original authors to answer questions.

7.1.1 Artifacts not available or not working

We will now provide a brief discussion on the availability of the artifacts (i.e., source code and data) for the non reproducible articles. We can divide them in three groups:

- **Source code available:** 1
- **Source code available but not working:** 4
- **Source code not available:** 7

Only one of the non reproducible articles has a publicly available and executable source code, the data however is not available due to, as stated by the authors, Copyright reasons.

For the algorithms where the source code was available but could not be used, the main issues were: lack of documentation on how to compile it, missing dependencies, missing training loop and no detail on the data structures required. We should mention here that those issues were such that we could not find a way to address them ourselves without extensive reverse-engineering of the available source code. In some cases the Github repository listed issues similar to the ones that prevented us from using the source code that went back even 3 years at the time of our analysis, but were left unanswered. It is important here to mention that providing the source code for a newly published method should be done in a way that allows other researchers and practitioners to use it effectively even after a few years. Sharing the code without any documentation on how to use it and precise information on the dependencies, in a very short amount of time becomes equivalent to not providing anything at all. This is in part due to the rapid succession of versions for many commonly used deep learning frameworks which are now being updated and replaced with incredible speed. Consider, for example, Keras. The version 1.0.0 of Keras was released on April 2016. Less than a year later, in March 2017, Keras 2.0.0 was released, including several changes making it incompatible with Keras 1. Another year later the version was 2.1.5. Each of these many updates runs the risk to alter the behavior of the original source code, e.g., introducing slight functionality changes, maybe the original code did not use the library as per specifications relying on deprecated functions and the new version removes them,

or a newly introduced bug. As another example consider *numpy*, an omnipresent numerical library for Python. When version 1.16 was released it included a slight change in the internal hierarchy of the classes, which did not alter the interface and functionality. However, any numpy data (i.e., data split, recommender model) that was saved using the common *pickle* library, which serializes the entire python object, became unreadable because the numpy class hierarchy was different. This is an issue we encountered ourselves and caused significant disruption to our experimental activities since due to the intricate dependencies several libraries have with numpy it proved to be extremely difficult to obtain a consistent environment using version 1.15. Even after a couple of years it may become very challenging to reconstruct a consistent and correct environment in which to execute past algorithms, if the original documentation and dependencies were not provided with sufficient detail. Remarkably we found instances where the publicly available source code could not be executed because it contained errors, missed functions or library dependencies that we could not resolve. In all those cases the public repository was probably created by selecting from another project (i.e., the one used for the original experiments) but then was never tested. In all these cases we contacted the authors for assistance but we only received one reply, where the authors stated they could not share a more complete version of the source code at that time.

Lastly for a third of the relevant articles we could not find a publicly available implementation and could not obtain one from the original authors. In half of those cases the original authors did not reply to our request. The algorithms in question were published one per year from 2015 to 2018. In the other half of cases we received a reply from the authors stating the code could not be shared. In two of those cased the authors did not provide any motivation for this, while in the other two the source could not be shared due to a *non-disclosure-agreement* and the source code had been deleted. These findings are similar to those of Stodden [107]. While we can understand that 4 years after the publication of an article the original materials may not be available anymore, this is much less understandable for an algorithm published only one year prior, in 2018.

It is clear that providing a publicly available implementation of an algorithm including at least minimal documentation requires extra work which, due to the current incentive structure for publication, may not be perceived as necessary, as was also pointed out by Collberg et. al [26]. Researchers may be encouraged to provide a public implementation which may even be a requirement during the review process (e.g., as in KDD), but seldom would this code be actually tested and executed by the reviewers, much

less analyzed for issues or information leakage. Even if the reviewers did indeed try to use the provided implementation, it is difficult to imagine a reasonably well-written paper be rejected only because the reviewers were not able to use the provided source code. It must be remembered that the publicly provided implementation should be treated as a proof of concept and not as a production-grade highly engineered solution. Therefore we cannot ask researchers to spend the significant amount of time that would be necessary to ensure their implementation works transparently across a multitude of systems. However, as a way to mitigate this, the use of virtualization could be helpful, allowing to create a controlled environment everyone could use. This would also enable a reviewer to check the provided materials more easily, but probably only in terms of whether the source code executes. Several obstacles would still be present, e.g., long run times, need for specialized hardware or high end GPUs. Moreover, some other problems will remain undiscovered as they would require an in-depth analysis of the provided materials, e.g., errors in the data processing, accidental information leakage, poorly optimized baseline algorithms. It is probably unsustainable to ask reviewers for such analysis. Therefore, addressing those further problems may be only possible if evaluation and reproducibility studies like ours were to become more common in our research community.

7.1.2 Difficulty to contact the authors

Another important dimension to consider in order to understand the low number of reproducible papers is the reachability of the original authors to answer questions.

We sent a number of emails to the authors of most of the articles which are discussed in this paper. In Table 7.1 we report an overview of those interactions along two dimensions: the columns refer to the reason for our request and the rows to how successful the interaction was. The column *Reproducibility issue* refers to issues preventing the article from being considered reproducible as per the definition we have adopted for this study (i.e., missing or incomplete artifacts). The column *Methodology question* refers instead to questions arising from the analysis of the materials provided by the authors (e.g., the data splits characteristics, epoch selection criteria). Now we will discuss both those cases in detail.

Concerning the *Reproducibility issue* questions we sent 15 emails in total, while for the 9 remaining articles we could immediately reproduce the results and did not require the original author's assistance. In case the au-

thors replied to our request and the issue could be solved the algorithm is classified as reproducible. Those cases all concerned algorithms for which the source code was not publicly available but was kindly provided to us upon our request. The algorithms in question were published in 2017 and 2018.

In the second case, the authors did provide a reply but that did *not* allow us to address the issue. In 4 of those cases we requested the source code while in 1 of those we requested the private data. Lastly, in almost half of the cases we received no reply, despite sending our email to all the authors (often several) of the article. It should be noted that in two further cases we received a reply several months later, only after the paper reporting our results had been published or the preprint made publicly available. We did not include these replies in the current analysis for three reasons. Firstly, it violated our own methodology which sets a waiting time of 30 days. Secondly, while we believe in the importance of being flexible and understandable, we received those replies only after this study was already complete. Lastly, the publication of the reproducibility study would of course change the incentive the original authors have to engage in a conversation with us.

Concerning the *Methodology questions* we sent 8 emails, it is significant to observe that we could not answer any of them (e.g., data splitting procedure, epoch selection criteria). The only replies we received did not help us in addressing the issue or clarifying our doubts. In some cases we were able to engage in a constructive conversation with the authors while in others we received short replies but we could not progress further. Remarkably, in 5 out of 8 cases we received no reply.

	Reproducibility issue	**Methodology question**	**Total**	**%**
Reply received, issue addressed	3	0	3	13%
Reply received, issue *not* addressed	5	3	8	35%
Reply *not* received	7	5	12	52%
No email sent	9	4		
Total	24	12	23	100%

Table 7.1: *Overview of our attempts to contact the authors of the original articles for two different types of requests, indicating whether we received a reply and, if so, whether we were able to successfully address with the authors the subject of our request. In summary we could address the issue in only 13% of cases and we received a reply at all in only 48% of cases.*

To summarize, we tried to contact the authors of 23 out of the 24 total articles we analyze in this study. It is remarkable to observe that in only 11

(48%) cases we received a reply at all and that in as little as 3 (13%) cases we could successfully address the question that was asked. These results are comparable in terms of the number of replies received but actually *far worse* in terms of their helpfulness then those reported by Collberg et. al [26] and Raff [94]. In our experience therefore, asking the authors for clarification *was not a reliable and effective way to gather information* mostly due to the low tendency the authors have in replying to our emails. While in some cases a few of the email addresses reported in the article were not active anymore, all articles had at least one that was still active. Aside from that we cannot know for certain which are the reasons for this lack of replies, but we can imagine several phenomena may contribute. For example, the implementation and experiments may have been done by a Ph.D. student, which, by the time the article is published and becomes known, may have graduated and concluded his/her academic career and have no interest in replying. Moreover, the increasing pressure to publish may contribute to a lack of interest in looking back to spend time discussing two or three years old articles, which may not have received many citations or may not be an active research topic anymore. Lastly, it is certainly possible that after a few years the original source code and data may not be available anymore if they were not archived properly and the memory of how certain things were done, when they were not described in the paper, could fade.

It is of course not possible to improve the reply rate of other researchers, but it is indeed possible, as a community, to take steps that will reduce the need for such requests in the first place. This boils down again to the improvement of the reproducibility of published research. As a way to overcome the issue we can only encourage the authors to provide as much relevant information as possible on the experimental protocol, possibly via public repositories, to ensure that the relevant knowledge on their work is not lost with time. It is especially in the interest of the research community to preserve track of its efforts for the benefit of everyone.

7.2 Methodological Issues

In Chapter 5 we have provided a detailed analysis of the reproducible articles and listed both the evaluation protocol as well as the methodological issues we found, see Table 7.2. In this section we will try to categorize those issues an provide further discussion on them.

Methodological issue	Paper
Stating the use of default hyperparameters for baselines	[125] [133] [131] [60]
Provided data splits inconsistent with the description	[57] [125] [131] [56] [134]
Number of epochs selected using test data	[57] [133] [131] [23] [56] [60]
Epoch selection criteria not described	[119] [72] [42]
Only one set of hyperparameters selected for multiple datasets	[119] [72] [134]
Others	[131] [56] [60]

Table 7.2: *Summary of the methodological issues we found for the reproducible articles*

7.2.1 Arbitrary experimental design

In our analysis we encountered a wide variety of experimental designs in terms of datasets, preprocessing, training-test splits, evaluation metrics and cutoffs. In this section we report a summary of the evaluation methodology used in the reproducible papers. Table 7.3 reports for each paper the datasets and training-test split used for the evaluation. It is possible to see that most articles use unique evaluation protocols and there are never more than two articles using the same one.

In terms of training-test split leave-one-out is used by 6 articles, of which half select a random interaction and half the last interaction. Holdout is used by 3 articles with the same 80/20 % split. The remaining papers use less common splitting methodologies like keeping a fixed number of interactions as training data or a cold user evaluation. We can also see that 7 articles rank the positive item with a certain number of negative items, sometimes 99, 100, 999 or 50 per each positive. In terms of the datasets, Table 7.4 lists all datasets with the papers using them. We can count 18 different datasets, with 10 of them being used only once and 6 of them being used only twice. On average, each dataset is therefore used by only 1.5 articles.

A further dimension to consider for the datasets is the preprocessing applied. A common preprocessing step involves the removal of items or users with less than a certain number of interactions: 2, 5, 10, 20. Another common preprocessing is binarizing explicit ratings, sometimes all of them, otherwise only if they exceed a given threshold. In some cases both preprocessings are combined resulting in a reduction of the size of the dataset by more than 80% or even 90% (e.g., see Sections 5.4 and 5.8) altering significantly the used dataset to the extent that it may have completely different characteristics. These aggressive preprocessings are almost never motivated besides for generic statements regarding the data being *too sparse*. The criteria used to decide whether a dataset is too sparse is, however, never stated. Furthermore, as is discussed in Section 7.3, Table

7.7, it is easy to find other articles using datasets whose sparsity is orders of magnitude higher. It certainly stands to question whether articles applying such drastic alterations to the data could claim, as some do, to be evaluated with *real-world datasets*.

It is worth here to refer again to the findings of Rendle et al. [97], where it was observed the reported results for the baselines on a widely known dataset were often suboptimal. If it was possible to report suboptimal baselines, without anyone noticing, for a widely known dataset, other evaluations executed on less known or aggressively preprocessed datasets are even more prone to unreliable results going undetected.

Paper	Datasets	Split
CMN	Epinions, CiteULike-a, Pinterest	leave-one-out, 100 negatives
SpectralCF	MovieLens1M, HetRec, AmazonInstantVideo	holdout 80/20
MCRec	MovieLens100K, LastFM, Yelp	holdout 80/20, 50 negative per positive
CVAE	CiteULike-a, CiteULike-t	train 1 or 10 interactions
CDL	CiteULike-a, CiteULike-t, Netflix Prize	train 1 or 10 interactions
NeuMF	MovieLens1M, Pinterest	leave-one-out, 100 negatives
Mult-VAE	MovieLens20M, Netflix Prize, MSD	cold user, holdout 80/20 profile
ConvNCF	Gowalla, Yelp	leave-last-out, 999 negatives
NeuRec	MovieLens1M, MovieLens Hetrec, FilmTrust, Frappe	holdout 80/20
DMF	Amazon Music, Amazon Movie, MovieLens100K	leave-last-out, 99 negatives
CoupledCF	MovieLens1M, Tafeng	leave-one-out, 99 negatives
DELF	Amazon Music, MovieLens1M	leave-last-out, 99 negatives

Table 7.3: *Datasets and training-test split used by reproducible articles*

Dataset	Paper	Dataset	Paper
MovieLens1M	[134] [57] [133] [131] [23]	Amazon Movie	[125]
CiteULike-a	[42] [72] [119]	Epinions	[42]
Yelp	[134] [60] [56]	FilmTrust	[133]
LastFM	[134] [60]	Frappe	[133]
MovieLens100K	[60] [125]	Gowalla	[56]
Netflix Prize	[119] [73]	MovieLens Hetrec	[133]
Amazon Music	[125] [23]	MovieLens20M	[73]
CiteULike-t	[72] [119]	MSD	[73]
Pinterest	[42] [57]	Tafeng	[131]

Table 7.4: *Datasets used by reproducible papers sorted from the most frequent to the least frequent*

Table 7.5 lists the metrics and recommendation list length (cutoffs) used for the evaluation. Again, as for datasets and data split, we can see numerous variations. Overall 6 metrics are reported. The most used metric is NDCG with 9 occurrences, then Recall and HR both at 6. In terms of cutoffs, most articles use relatively short lists, between 1 and 20 elements, two articles report up to a 100 elements and only two go beyond that.

This great heterogeneity of evaluation protocols is the product of the

great freedom researchers have. The only common theme appears to be the comparison of the newly proposed model against a selection of baselines, datasets, data split protocols, metrics, cutoffs and, remarkably, the absence of any justification for that choice. Furthermore, we can see that some algorithms do exhibit a more *unusual* evaluation protocol than others, for example by keeping only a fixed number of interactions in the training data or by fusing a random holdout with the negative item evaluations and selecting the number of negatives as a function of the number of test items. Again, even in those cases, no detailed justification is provided for why that specifically crafted evaluation protocol was adopted.

It is expected that different articles may focus on different scenarios and therefore will rely on different evaluation protocols. The problem arises due to the tendency of using different evaluation protocols even for algorithms that are, according to the description reported in the article, competing in the same scenario. In all those cases, no reason is provided on why the evaluation protocol is changed. There is no clear explanation for this phenomena, which may again be due to the contribution of several factors. It is possible authors are simply relying on previously written evaluation code which is then re-used for later articles. It could also be that the reported evaluation protocol may be the one allowing to show the strongest results for the newly proposed algorithm. Those are, in our view, not issues per-se. Furthermore, it is expected that different algorithms may exhibit different relative performance under different evaluation protocols.

The problem arises because most articles propose new algorithms claiming their ability to outperform the state-of-the art in a very general sense and, in what appears a contradiction, almost inevitably support this claim evaluating the model in only a very specific condition. This not only relies on extraordinary assumptions of generalizability for the reported evaluation results, but also has the effect of creating a level of opacity on which is the scenario of *successful applicability* of those new methods, an issue raised by Christodoulou et. al [24]

As a way to overcome this issues, we can suggest two possible directions: justify the experimental protocol used and strengthen the generalizability claims by reporting results on multiple scenarios as well as negative results.

For the first point, justifying the experimental protocol requires to clarify the scenario the algorithm has been developed for. This begins with the choice of the dataset, as different domains will exhibit different behaviors. Then, the choice of recommendation list length and metrics. For example, in a scenario requiring to provide a short list of recommendations which

must contain at least a relevant item, one would choose a short list length and a metric like precision. In a scenario where we want to maximize the number of retrieved relevant items the evaluation might require longer recommendation lists and use recall as metric. If the scenario is such that the user will tend to look and explore the whole recommendation list the ranking of the items may have limited importance, in other cases instead the users will tend to only look at the very first positions and therefore ranking metrics like NDCG or MAP become important.

For the second point, in order to strengthen the generalizability claims, the evaluation should be done in more varied conditions. This again begins with the dataset, since one cannot claim a generalizable result if the evaluation is only done on a specific domain like movie ratings (i.e., MovieLens, Netflix) or with datasets which are just variations of the same data (i.e., CiteULike-a and CiteULike-t). Furthermore, the presence of *negative results*, i.e., scenarios where the newly proposed algorithm is not competitive against the state-of-the-art, would be of significant interest. Negative results, associated with an analysis of how the successful and non-successful scenarios are different would allow for a more in-depth understanding of the characteristics of the new algorithm and its potential as well as limitations. This would shed further light on the current state-of-the-art and allow the community as a whole to have a clearer picture of which are the scenarios where we are strongest and those where we are lagging behind and may need further efforts.

Paper	Metrics	Cutoffs
[42]	HR, NDCG	5, 10
[134]	Recall, MAP	20 - 100, step 10
[60]	Precision, Recall, NDCG	10
[72]	Recall	50 - 300, step 50
[119]	Recall	50 - 300, step 50
[57]	HR, NDCG	1, 5, 10
[73]	Recall, NDCG	20, 50, 100
[56]	HR, NDCG	5, 10, 20
[133]	Precision, Recall, MAP, NDCG, MRR	5, 10
[125]	HR, NDCG	10
[131]	HR, NDCG	1-10
[23]	HR, NDCG	10

Table 7.5: *Evaluation metrics and cutoffs used by reproducible articles*

7.2.2 Selection and propagation of weak baselines

In our analysis we could observe two very important problems related to the baselines the new algorithm is compared against: the choice of baseline al-

gorithms combined with a lack of optimization. These problems, although may seem confined to the singe article, have also an impact in later research papers as weak baselines tend to propagate.

Regarding the choice of baselines we could observe the disappearance of simple neighborhood-based and non-neural machine learning baselines, with them being replaced by much newer neural models. Simple baselines do appear sometimes, in the form of an ItemKNN or BPR matrix factorization, but usually their results as reported in the original paper are substantially worse than the ones we could obtain in our experiments.

The lack of simple baselines is especially problematic because, as we could observe, there is no clear winner between neighborhood-based, non-neural machine learning and neural algorithms for all datasets. For example we could observe that in some datasets neighborhood-based methods work better than non-neural machine learning while in others the opposite is true (see Section 5.3). This creates obvious problems if a neural method is compared only against non-neural machine learning baselines in a scenario where those are not competitive against neighborhood-based algorithms. Further to this point, we could even observe one case where the dataset exhibited such a high popularity bias that a non-personalized baseline was better than all other algorithms (see Section 5.11).

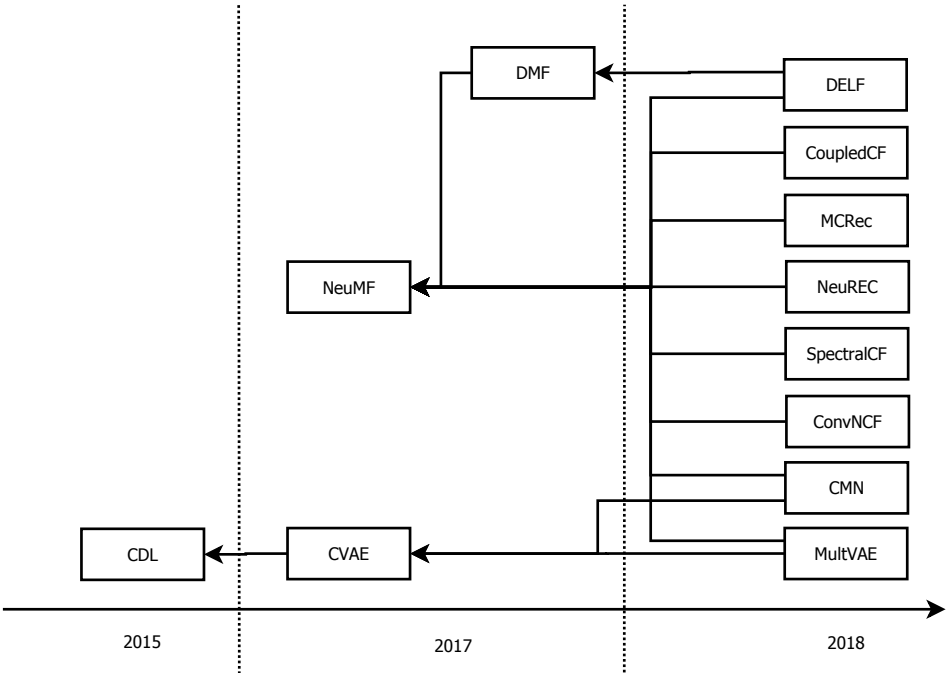
Another critical finding is that, often, those baselines are not optimized properly. In a number of cases, we could read that the hyperparameters or many neural baselines were set as in the original article proposing it. This was done despite the evaluation protocol and datasets used were different. Any algorithm which is not optimized for a specific experimental configuration will not exhibit its best recommendation quality, of course, and therefore any comparison made using that result is inconclusive. It is not sufficient to pick an algorithm and use it, it is essential to optimize it in order to obtain reliable results. Any comparison made without optimizing all baselines is meaningless. Similarly, any comparison made using a model which is missing important parts is again meaningless, for example using ItemKNN without shrinkage is to be considered as a bad practice. Unfortunately very few papers provide details on how at least some of the baseline algorithms have been optimized. While we can say for certainty that the comparison is meaningless when the paper states that default hyperparameters have been used, we are left wondering about those who do not state anything at all. The reasons why the baselines tend to be not optimized can be several. Fist, it can be complicated to acquire a working implementation of new algorithms, as we have discovered ourselves. The publicly available materials never include any code to perform the hyper-

parameter tuning and, therefore there is no indication of how it should be done in terms of the range and sensitivity the algorithm exhibits with respect to each hyperparameter. Finding suitable ranges and distributions as we did for the baselines may require significant experimentation (i.e., if the optimal value for a hyperparameter is at one end of the range, the range should be extended and the tuning executed again). This is significantly aggravated by the enormous computational cost most neural models have. According to our experiments, systematic hyperparameter tuning for one single neural baseline can take several days or even weeks, depending on the dataset size, even when using modern GPUs.

The combination of those problems creates a cascade effect that propagates further as new articles are published. In Figure 7.1 all reproducible algorithms analyzed in this study are reported and connected whenever the latter article uses a previous one as baseline. As we have previously mentioned, there is a tendency in the community to disregard simple methods in favor of more complex ones. If the evaluation of older neural models is not done properly, those non competitive models will appear to be state-of-the-art and will be used as baselines in later articles. If those later articles do not report and optimize properly simple baselines, weak complex baselines will propagate and result in less challenging baselines for new algorithms to outperform. In our study we found that CDL was not a strong algorithm in many cases, but it nonetheless was used as baseline by CDAE which again was not a strong baseline, that was used by both Mult-VAE and CMN. The same applies for NeuMF, which is on par or slightly below the simple baselines we report, and is then used as baseline by all articles published in 2018. Any article using NeuMF in its *default* configuration would be reporting an even lower recommendation quality, therefore much easier to outperform.

As a way forward for these issues, several suggestions can be made. It is indispensable to keep as baselines some simple neighborhood-based and non-neural machine learning models, as well as at least a non-personalized baseline. This is both because they provide, as we could demonstrate, very good recommendation quality, and are therefore still state-of-the-art algorithms, but also because they are much faster and easier to optimize than complex methods. Due to this, they will always constitute a more practical and reliable *minimal set* of baselines, on top of which more complex models can certainly be added. Furthermore, both researchers and reviewers should ensure all models are properly optimized for the given experimental scenario, and that some details on how the hyperparameters have been tuned is available in the paper. We stress again that, if the articles mentions default

Figure 7.1: *Overview of Neural Methods, arrows indicate when a newer method used another one as baseline in the experiments.*



hyperparameters have been used, that should constitute a strong warning sign that the evaluation reported in the paper was not done properly. As mentioned before, ensuring proper optimization of the neural baselines is challenging both due to the lack of information on how to do that and due to the high computational cost. As a way to mitigate this phenomena, having a publicly available predefined training-test split for various datasets with associated baselines and results would avoid the need for every researcher to run the optimization from scratch. Such public repository would be useful to ensure full comparability between the results of newly proposed methods and could be incremented by each new article published using it.

7.2.3 Errors and information leakage

As we have summarized in a subsection for each of the algorithms we analyzed in Chapter 5, a multitude of bad experimental practices or errors in the evaluation protocol were found.

Table 7.3 summarizes the methodological issues we discovered and that are described in detail in each algorithm's result section. As we have previously described, we can see that four articles explicitly state that they

use default hyperparameters for the baselines. Furthermore, three articles adopt an uncommon evaluation protocol without sufficient justification and discussion.

Specifically referring to errors in the evaluation procedure, we can see that almost half of the articles, five, have made data splits publicly available that are inconsistent with the description provided in the paper. In some cases, this inconsistency is found at the preprocessing stage, as the data contains users and items with less than the minimum number of interactions reported in the article. In other cases the inconsistency is found in the training-test split. Rather frequently, the set of negative items to be used in the evaluation contains duplicates or overlaps with the train or test data, in such a way that the items there present are not real negative items. Lastly, in a remarkable case, the distribution of the test data proved to be different from the training data an likely the result of an erroneous splitting procedure.

Another common issue we found is the lack of a proper way of selecting the number of epochs the newly proposed algorithm will need to be trained for. The cases we observed are tree:

- The source code contains the training loop in which the model is also evaluated. The *model is evaluated on the test set* at each epoch and that result is used to select the optimal number of epochs. This causes information leakage from the test data obfuscating the real generalisability of the original model and inflating its performance when compared to other models whose training does not contain information leakage.

- Similarly to the previous case, the model is evaluated on the test data at each epoch. However, even more remarkably, the reported results are not associated to a specific epoch. Rather, for each metric the best value is reported. This not only means that there is information leakage from the test data, but also that *the results reported are inconsistent* as each metric may reach its optimal value at a different stage in the training process.

- Lastly, when none of the previous issues could be observed, in all but few of the remaining articles *we could not discern any criteria* for the selection of the number of epochs. In the few papers motivating the choice of the number of epochs, a plot reporting the variation of the training and validation loss was usually reported. In the other articles, the number of epochs (ranging from 30 to 1000) was not motivated

or even mentioned if not for being present in the source code. This too, we argue, is a bad practice for the following two reasons: first, it is not a transparent stopping criteria and may hide the selection of such fixed number of epochs using the test data as well as other issues; second, it does not provide any information to other researchers regarding the training behavior of the algorithm, like how quick its convergence is and how variable its recommendation quality is during training. Clearly different algorithms will exhibit different behaviors, this is confirmed by the fact that the number of training epochs we found in the source code ranges from 30 to 1000 depending on the algorithm.

Another issue we encountered, although to a lesser extent, is the reporting of only one set of hyperparameters for the proposed neural model even though the evaluation is reported on multiple datasets. As we have discussed previously when talking about baselines, any comparison in which the algorithm has not been optimized for the specific dataset is mostly meaningless. As an additional problem, none of those articles stated on which datasets those hyperparameters were optimized so a reader and a reviewer would not know how were those values obtained and which of the experiments reported in the article is unreliable. The only way to address this issue is to make sure that all algorithms, both the baselines and the newly proposed one, are optimized fairly and consistently for all experiments.

In three cases the metrics implemented in the articles were incorrect and the results reported in the original paper were therefore not correct. Only one of such cases is present in Table 7.2, since in the other two the authors later published a correct implementation and the corresponding results on the public repository, showing that the relative performance of the models was not changed.

The last issue we encountered is in the justification and proper demonstration of the claims made in the article. We refer to the use of CNNs to learn embeddings correlations. In those cases, the appropriateness to use a technique developed for other types of data was not unequivocally demonstrated. Furthermore, the claim that CNN allows to learn embedding correlations was not supported by a specifically designed evaluation protocol, but rather by a simple comparison against other baselines. As previously described and demonstrated, the comparison was meaningless as plenty of other factors would have played a role in the results and, in the end, we could demonstrate that the CNNs are not able to model the embeddings correlations in interaction maps.

7.3 Scalability

In this section we provide an overview of the computation time of all the algorithms we analyze, and compare it with the baseline models. As described in Section 4.3.3, we report the training time of the model, the time required to compute all the recommendation lists during the evaluation as well as the throughput, i.e., the number of recommendation lists that can be generated per second.

In order to read these results correctly, we first provide some further details on the measurements and their limitations.

All measurements refer to a specific Amazon AWS instance.[2] All neural algorithms were executed on the GPU, with the only exception of CDL (see Section 5.1), while all baseline algorithms were executed only on the CPU.

In terms of the training time, the implementations of the baseline algorithms vary in terms of efficiency. Some use standard solvers (PureSVD, NMF, SLIM ElasticNet), others are written in Cython[3], i.e., in C, and compiled (KNNs, MF BPR, FunkSVD, SLIM BPR), others are written in plain Python with vectorised operations ($P^3\alpha$, $RP^3\beta$, iALS, $EASE^R$), some are single core, others take advantage of multithreading. Furthermore, the measurements will also be affected by the stochastic nature of the Bayesian Search which could select different hyperparameters for the same algorithm (e.g., number of latent factors or neighbors) and dataset or by the preprocessing of the dataset altering its size significantly. Similarly the deep learning models are implemented in Tensorflow or Keras and with varying degrees of efficiency. Due to this heterogeneity the computational time measurements should not be taken as exact measurements but rather as a qualitative comparison.

Regarding the recommendation time and throughput, it should be noted that the computation of the top-k recommendation list requires two steps, first computing a score for each item and then ranking the items. The first step, computing the scores of the items, is done for each algorithm independently and mostly in different ways. The second step, ranking the items given their score, is instead done by a specific function of our recommendation framework (see Section 4.5), therefore this step will be common to all algorithms. The computation cost of ranking the items is, on average, constant. Due to this it will dominate the total recommendation time for algorithms generating the scores very quickly. This is the reason why,

[2] The computation time refers to the total instance time for one AWS instance p3.2xlarge, with 8 vCPU, 30GB RAM, and one Tesla V100-SXM2-16GB GPU.

[3] `https://cython.org/`

for example, the non-personalized TopPopular and the neighborhood-based models often have very similar throughput. When comparing algorithms from different papers, it should also be taken into account that, in some of those, the algorithm computes the scores for all items, while in others, only the scores of a subset of items (i.e., negatives and test items).

Table 7.6: *Overview of train and recommendation time for the analyzed neural algorithms compared with the best performing baseline method.*

	Dataset	Algorithm	Training time	Recommend. Time	Throughput [usr/s]
CMN	Pinterest	$RP^3\beta$	9.23 [sec]	1.50 [min]	608
		CMN	7.81 [hour]	5.90 [min]	156
SpectralCF	MovieLens1M	SLIM ElasticNet	1.15 [min]	34.91 [sec]	169
		SpectralCF	37.40 [min]	21.55 [sec]	280
MCRec	MovieLens100K	SLIM ElasticNet	11.45 [sec]	1.09 [sec]	862
		MCRec	2.36 [hour]	2.76 [min]	6
CVAE	CiteULike-a P=10	ItemKNN CBF	8.61 [sec]	14.66 [sec]	347
		CVAE	1.27 [hour]	21.33 [sec]	241
CDL	CiteULike-a P=10	ItemKNN CBF	8.61 [sec]	14.66 [sec]	347
		CDL	1.51 [hour]	15.26 [sec]	337
NeuMF	Pinterest	$RP^3\beta$	17.95 [sec]	7.22 [min]	126
		NeuMF	1.94 [day]	1.94 [hour]	8
Mult-VAE	Netflix Prize	SLIM ElasticNet	8.22 [hour]	1.16 [min]	580
		Mult VAE	1.26 [hour]	1.36 [min]	491
ConvNCF	Gowalla	UserKNN CF	24.25 [sec]	2.68 [min]	322
		ConvNCF	12.43 [hour]	3.90 [min]	232
NeuRec	MovieLens HetRec	SLIM ElasticNet	5.17 [min]	42.49 [sec]	49
		INeuRec	13.93 [hour]	35.26 [sec]	60
		UNeuRec	20.71 [hour]	39.53 [sec]	53
DMF	MovieLens1M	SLIM ElasticNet	3.41 [min]	14.18 [sec]	425
		DMF NCE	7.77 [day]	9.15 [min]	11
		DMF BCE	4.07 [day]	8.05 [min]	12
CoupledCF	MovieLens1M	SLIM ElasticNet	2.86 [min]	14.69 [sec]	402
		DeepCF	1.08 [hour]	25.05 [sec]	241
		CoupledCF	3.90 [hour]	34.18 [sec]	177
DELF	MovieLens1M	SLIM ElasticNet	19.23 [min]	5.19 [sec]	354
		DELF MLP	3.45 [hour]	9.22 [min]	11
		DELF EF	6.69 [hour]	9.18 [min]	11

In Table 7.6 we compare the training time and throughput for all neural algorithms with the best performing baseline on the dataset with the highest number of interactions, the statistics of the dataset are reported in Table 7.7. The baseline is selected as the one with the highest recommendation quality on the most metrics-cutoff measurements on that dataset. In case of ties, the faster baseline is selected. The complete measurements for all algorithms

Table 7.7: *Overview of the size and density of the datasets reported in Table 7.6.*

Algorithm	Dataset	Interactions	Items	Users	Density
CMN	Pinterest	1.5 M	9.9 K	55.1 K	$2.7 \cdot 10^{-3}$
SpectralCF	MovieLens1M	226 K	3.7 K	6.0 K	$1.0 \cdot 10^{-2}$
MCRec	MovieLens100K	100 K	1.6 K	0.9 K	$6.3 \cdot 10^{-2}$
CVAE	CiteULike-a P=10	55.5 K	16.9 K	5.5 K	$5.9 \cdot 10^{-4}$
CDL	CiteULike-a P=10	55.5 K	16.9 K	5.5 K	$5.9 \cdot 10^{-4}$
NeuMF	Pinterest	1.5 M	9.9 K	55.1 K	$2.7 \cdot 10^{-3}$
Mult-VAE	Netflix Prize	56.9 M	17 K	463 K	$7.2 \cdot 10^{-3}$
ConvNCF	Gowalla	1.2 M	52.4 K	54.1 K	$4.4 \cdot 10^{-4}$
NeuRec	MovieLens HetRec	855 K	10.1 K	2.1 K	$4.0 \cdot 10^{-2}$
DMF	MovieLens1M	1 M	3.7 K	6.0 K	$4.4 \cdot 10^{-2}$
CoupledCF	MovieLens1M	1 M	3.7 K	6.0 K	$4.4 \cdot 10^{-2}$
DELF	MovieLens1M	1 M	3.7 K	6.0 K	$4.4 \cdot 10^{-2}$

and datasets are available in Appendix C.

From Table 7.6 we can see that a clear trend emerges. The training time of the baseline algorithms is usually very short, in all but one case can be measured in seconds or minutes, while the the training time of the neural algorithms is substantially longer, measuring usually more than one hour up to several days. In 7 cases the baseline algorithms have a training time below 2 minutes, while the neural algorithms have a training time of between 37 minutes and almost 2 days. We can see that 11 baselines have a training time of less than 20 minutes while, in those same cases, the neural algorithms have a training time which is frequently of several hours up to an impressive 7 days.

The only case where the baseline is slower than the neural algorithm is Mult-VAE (see Section 5.5), which is also the only algorithm we found to be competitive against our baselines.

A similar trend, although less marked, can be observed for the Recommendation time and throughput, where in 6 out of 12 cases the neural algorithm has a throughput of less than half that of the baseline algorithm. In 4 further cases we can see that the neural model is able to generate only around ten or less recommendation lists per second, being almost two orders of magnitude slower than the baseline.

Another important dimension to consider in terms of scalability is the size of the datasets used for the evaluation. In Table 7.7 we list the statistics of the datasets given the specific preprocessing applied (i.e., MovieLens1M may have less than 1 M interactions if ratings lower than a threshold are removed). As we can see most datasets are rather small, in 5 cases the dataset has less than 1 million interactions, in 6 cases between 1 and 1.5 millions of interactions, only one dataset has more than 50 millions. It is remarkable to

observe how more than ten years after the Netflix Prize with its dataset of 100 million interactions, the current research practice is still to use datasets with less than 1% of that interactions. This is also despite the availability of several other datasets of various sizes and characteristics [10]. In terms of the number of items, most datasets only have a few thousands, with only 3 datasets having more than 15 thousand items. Similar observations can be made for the number of users, in 8 cases equal or less than 6 thousands and only in 4 cases higher than 50 thousands. Their density varies widely from the $6.3 \cdot 10^{-2}$ of MovieLens 100K to the $4.4 \cdot 10^{-4}$ of Gowalla. Yet, despite this rather (sometimes very) small datasets, the computation time of complex models is still commonly measured in tens of hours up to days.

It is expected that more complex models will exhibit higher computation cost and that improvements on recommendation accuracy will follow a law of diminishing returns, becoming harder and harder as the accuracy improves. Still, the computation cost of some of the algorithms we analyze is extremely high and, despite this, most of them are not able to compete with baselines requiring only a fraction of that training time.

Limited scalability of a proposed model is not a methodological issue per-se, however, it has important consequences. First, it will make deploying the proposed algorithm in a realistic environment extremely difficult, as the cost could be prohibitive. A second, less obvious, but extremely important consequence is that the enormous costs for tuning the algorithms can lead researchers to skip the optimization of the neural algorithms they use as baselines. As reported in Section 7.2.2, there is a tendency to take the hyperparameter settings from previous articles (i.e., the default hyperparameters), even when those were determined for different evaluation setups. This contributes greatly to the frequent occurrence of weak or inadequately optimized baselines.

While the computation cost of neural methods is often overlooked in the recommender systems field, it is a topic of great importance with immediate repercussions on the practical usability of the research results. The high economic cost of running the algorithms can be a hindrance limiting the pool of researchers to only those who can afford the expensive computational infrastructure required. The potential for improvement is significant, as an example a recent paper [20] demonstrated that, with certain optimizations, the computation time of a complex algorithm on a CPU could be reduced by an order of magnitude.

7.4 Limitations

In this section we describe two possible limitations of the experimental protocol adopted in this thesis.

Hyperparameters of the neural algorithms taken from the original articles. The first limitation of this thesis is the use, for the analyzed neural algorithms, of the hyperparameters as reported in the original articles. This step relies on the assumption that the neural algorithms were properly optimized by the original researchers. It does however not take into account two possible scenarios. The first, that we have observed several times, is that there were cases of information leakage in the original evaluation protocol, in particular in the selection of the number of train epochs. It is therefore possible that information leakage was also present during the hyperparameter optimization of the analyzed algorithm. In those cases, the recommendation quality of the neural method, when properly tuned on a validation set, could be even lower than what we observed in this study. On the other hand, it is also possible that the optimization of the neural model was not done to its full extent because, due to the weak baselines reported in the original articles, further optimization was not necessary to outperform them. If stronger baselines had been added in the original article, the optimization of the neural model itself may have been more thorough and, therefore, its end result more competitive.

The motivation of why such optimization was not done is due to a few orders of reasons. The first is that the hyperparameters said to have been optimized for the neural models are not consistent among articles. Probably the most important example is the architecture itself which in some cases is said to have been optimized, while in others no detail is provided. Defining a *fair* optimization protocol for all algorithms (i.e., architecture, activation function, regularization etc.) is not trivial. In our study a new optimization of the hyperparameters was only done for SpectralCF (see Section 5.12) due to an error in the originally provided training-test split. A second order of reasons is due to limitations in the original source code, which rarely provides the required flexibility to optimize the neural method. In some cases it can be seen that several architectural components are commented-out in the source code, suggesting the architecture optimization process was, to an extent, manual. Overcoming this would require significant implementation effort. The last, and probably most important, is that due to their huge computation cost a hyperparameter optimization for all neural models would be prohibitively long and resource intensive. Consider that the total GPU time needed for the current experiments is composed by a first train-

ing with early stopping then another on the union of training and validation data for the selected number of epochs. Based on this, we estimate the total GPU time required to optimize all neural algorithms, with the same protocol used for the baselines, would be in the range of 3 years, and could be as high as 15 years if 5-fold cross validation for significance testing is used.

Hyperparameter optimization with a fixed number of evaluations regardless of execution time. Another limitation of this thesis is that the hyperparameter optimization was done on a fixed number of 50 functional evaluations (i.e., model evaluations on the validation data), regardless of the time it required. This means that, for some baselines and neural algorithms the optimization could run for days, while for others it could be completed in minutes. As we have argued in Section 7.3, the computation time is an important dimension to be taken into account and has an important impact on the applicability of that algorithm. In order to improve the *scalability fairness* of the evaluation protocol, baselines as well as neural algorithms could be optimized not for a fixed number of hyperparameter configurations but rather for a fixed amount of time. This would give faster algorithms an advantage as they would be able to explore a significantly higher number of cases, and would impact negatively slower algorithms, which may be able to explore only a limited set of hyperparameters. The results would therefore mirror more closely a scenario where only a limited time or resource budget is available and allow to select the most *cost-effective* algorithm.

CHAPTER 8

Conclusions

In recent years a substantial number of new neural approaches for the recommendation problem have been proposed. Despite the ever present claim that those complex models are outperforming the state-of-the-art, evidence exists that complex models do not necessarily achieve better recommendation quality when compared with simple baselines [127]. Furthermore, a tendency to report weak or inadequately optimized baselines in the evaluations has been observed for other information ranking tasks [74, 80, 97], which could give the impression of continuous incremental improvement, even though none may exist.

In this study, we have reported an analysis of the reproducibility of neural algorithms published in recent years at high level conferences as well as a comparative evaluation of those new complex models against an ample set of simple but well optimized baselines.

The results of our study reveal that, in the great majority of cases, neural algorithms can be outperformed by baselines developed a decade or more ago, i.e., neighborhood-based and non-neural machine learning algorithms. Despite the vast number of new neural algorithms that are published for the top-k recommendation task, virtually no progress appears to have been made in the last years.

This surprising result can be explained by several issues we could observe in how claims are demonstrated within the research community. We could identify a tendency to rely on unique evaluation protocols which differ even between articles appearing to target the same scenario, making comparisons opaque or impossible. We also found numerous errors in the evaluation protocols themselves. It is common to report weak or inadequately optimized baselines, if those are tuned at all, or to use *default hyperparameters*, and comparing them with a much more optimized neural method. This results in the false appearance of the proposed model outperforming the state of the art. Even using test data during the training of the model to select the number of epochs, causing information leakage, appears to be common practice.

These issues are however neither new nor specifically tied to the domain of recommender systems or to neural approaches in particular. Most of those issues could be addressed with increased awareness of the community and improved reproducibility of published research. To this end, independent evaluation and reproducibility studies published by different research groups should be encouraged. This would put the recommender systems research community on the same trajectory as the information retrieval research community, in part probably thanks to the surprising results from [4, 74, 127].

There are however a few last observations that are, we believe, worthy of attention. Not a single reproducible paper we analyzed reported an online study, where the recommendations are shown to real users. This is yet another important limitation of today's research practice, which appears to forget that, in the end, we are not building offline tools but rather algorithms that will be used by a real person. Several research papers have already questioned whether small accuracy gains really translate into greater user satisfaction [7, 32, 51, 82, 99]. In this sense, the current focus on achieving the *best* recommendation quality, sometimes needing to use statistical significance tests to confirm a small improvement indeed exists, overshadows several other important dimensions. Those dimensions are embedded in the very nature of a recommender system, its deployment in a real case to be used by a real person.

Among the feedback we received after the publication of our first evaluation study [46], one comment said that if our community is too strict on how to conduct an evaluation, it will render us impermeable to new ideas and techniques. It is certainly true that it is not reasonable to expect any new and interesting idea to be immediately able to outperform other ones that have been perfected over years or decades. But the answer cannot be

to lower our experimental standards in order to allow them to pass through, hoping that they will become competitive in the future. As we have seen, for the specific scenario we evaluate in this study, if the upward pressure for growth lacks due to poor evaluation practices, so will the improvement in the algorithms we develop. That observation regarding our openness to new ideas instead raises indirectly a very important point. It is undeniable the very strong focus on demonstrating accuracy improvements over the state-of-the-art is currently (almost) a prerequisite for the acceptance of a paper. Often when a model is proposed the very first claim made in the paper is that it allows to outperform the state-of-the-art. Its other merits, e.g., greater explainability of recommendations, faster training time, robustness to noise or missing data and so on, are subordinate. While papers specifically targeted to those aspects can be found, they are certainly far less frequent. This is also common practice in several other fields of machine learning [84].

It is this over-reliance on accuracy alone when combined with inadequate evaluation procedures, we argue, that is hampering our ability to explore and adopt new techniques. Accuracy alone, as previously mentioned, is a strong simplification of the problem we try to solve, and it is easy to imagine use cases where *less accurate* algorithms can be acceptable if they bring other advantages. One of them is the computational cost and model complexity, which translates directly into greater engineering and maintenance cost in a real world service, to the point where the increased recommendation quality may not be worth it. A notable example comes from the Netflix Prize, as the company never fully integrated the winning algorithm in its systems partially due to the accuracy improvements being too small to justify the engineering effort and partially because, by the time the challenge ended, the business model of Netflix itself had evolved[1]. Another example of a desirable property could be the model being able to maintain good quality despite the *data drift* i.e., the continuous change in the available data, which is the natural environment when considering an operational recommender system but is invisible in the traditional offline experiments we rely upon. As a last example, we can cite the need for compliance with privacy laws, which may require the company to be able to remove at short notice the personal data of some of its users and to ensure that data does not persist hidden in its trained models. The 2020 RecSys Challenge[2], hosted by Twitter, updated the dataset *daily* to ensure

[1] https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429

[2] https://recsys-twitter.com/

compliance with the European General Data Protection Regulation[3].

In the end, all these problems: the inadequately optimized baselines creating the illusion of progress, the arbitrary experimental protocol making comparisons impossible, the errors in the evaluation itself and the restricted focus of published research all seem to contribute to a certain level of stagnation in the research field. Yet, as we have exemplified, there are innumerable options and directions we can explore. All of them should, nonetheless, be assessed with transparent, fair and thorough evaluation in the appropriate scenarios, so that we can maintain trust in our own experimental practices ensuring the progress is not only reliable, but also shared by the community as a whole.

8.1 Future Works

In this thesis we have presented a reproducibility and evaluation study for recent neural algorithms applied to the top-k recommendation problem. We will now highlight possible future research directions.

Provide guidelines on the baseline algorithms a well as optimization space. As stated in Chapter 7, the experiments required for this evaluation study have been extensive and allowed us to collect the results of more than 900 models evaluated on different datasets on several metrics (see Section 4.3) as well as the results of 41,000 hyperparameter configurations. This amount of data could be mined as done, for example, in [92], in order to analyze how sensitive various algorithms are to certain hyperparameters and provide guidelines for the relative range and distribution. The characteristics of the datasets could be taken into account as well. This would help to design a more efficient search space for the optimization phase.

Expanding the analysis to other recommendation problems and other algorithmic techniques. In this study we have focused on a very specific scenario, neural algorithms for top-k recommendations. As we have discussed several times (see Chapter 2) issues in the reproducibility of published research and methodological problems have been observed widely. Possible extensions of this study could go in the direction of other recommendation tasks (e.g., session-based, cross-domain) as well as on other model types (e.g., non-neural algorithms)

[3] https://gdpr.eu/

APPENDIX $\mathcal{A}$

Hyperparameter range and distribution for baseline algorithms

Appendix A. Hyperparameter range and distribution for baseline algorithms

Table A.1: *Hyperparameter values for our KNN and graph based baselines.*

Algorithm	Hyperparameter	Range	Type	Distribution
UserKNN, ItemKNN cosine	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	similarity	cosine	Categorical	
	normalize[a]	True, False	Categorical	
	feature weighting	none, TF-IDF, BM25	Categorical	
UserKNN, ItemKNN dice	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	similarity	dice	Categorical	
	normalize[a]	True, False	Categorical	
UserKNN, ItemKNN jaccard	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	similarity	jaccard	Categorical	
	normalize[a]	True, False	Categorical	
UserKNN, ItemKNN asymmetric	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	similarity	asymmetric	Categorical	
	normalize[a]	True	Categorical	
	asymmetric alpha	0 - 2	Real	uniform
	feature weighting	none, TF-IDF, BM25	Categorical	
UserKNN, ItemKNN tversky	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	similarity	tversky	Categorical	
	normalize[a]	True	Categorical	
	tversky alpha	0 - 2	Real	uniform
	tversky beta	0 - 2	Real	uniform
$P^3\alpha$	topK	5 - 1000	Integer	uniform
	alpha	0 - 2	Real	uniform
	normalize similarity[b]	True, False	Categorical	
$RP^3\beta$	topK	5 - 1000	Integer	uniform
	alpha	0 - 2	Real	uniform
	beta	0 - 2	Real	uniform
	normalize similarity[b]	True, False	Categorical	

[a] The *normalize* hyperparameter in KNNs refers to the use of the denominator when computing the similarity.

[b] The *normalize similarity* hyperparameter in P3alpha and RP3beta refers to applying L1 regularization on the rows of the similarity matrix

Table A.2: *Hyperparameter values for our machine learning baselines.*

Algorithm	Hyperparameter	Range	Type	Distribution
SLIM BPR	topK	5 - 1000	Integer	uniform
	epochs	1 - 1500	Integer	early stopping
	symmetric	True, False	Categorical	
	sgd mode	sgd, adam, adagrad	Categorical	
	lambda i	10^{-5} - 10^{-2}	Real	log-uniform
	lambda j	10^{-5} - 10^{-2}	Real	log-uniform
	learning rate	10^{-4} - 10^{-1}	Real	log-uniform
SLIM ElasticNet	topK	5 - 1000	Integer	uniform
	l1 ratio	10^{-5} - 10^{0}	Real	log-uniform
	alpha	10^{-3} - 10^{0}	Real	uniform
MF BPR	num factors	1 - 200[a]	Integer	uniform
	epochs	1 - 1500	Integer	early stopping
	sgd mode	sgd, adam, adagrad	Categorical	
	batch size	2^{0} - 2^{10}	Integer	log-uniform
	positive reg	10^{-5} - 10^{-2}	Real	log-uniform
	negative reg	10^{-5} - 10^{-2}	Real	log-uniform
	learning rate	10^{-4} - 10^{-1}	Real	log-uniform
MF FunkSVD	num factors	1 - 200[a]	Integer	uniform
	epochs	1 - 500[b]	Integer	early stopping
	use bias	True, False	Categorical	
	sgd mode	sgd, adam, adagrad	Categorical	
	batch size	2^{0} - 2^{10}	Integer	log-uniform
	item reg	10^{-5} - 10^{-2}	Real	log-uniform
	user reg	10^{-5} - 10^{-2}	Real	log-uniform
	learning rate	10^{-4} - 10^{-1}	Real	log-uniform
	negative quota[c]	0.00 - 0.50	Real	uniform
PureSVD	num factors	1 - 350	Integer	uniform
NMF	num factors	1 - 350	Integer	uniform
	solver	mult. update, coord. descent	Categorical	
	init type	nndsvda, random	Categorical	
	beta loss	kullback-leibler, frobenius	Categorical	
iALS	num factors	1 - 200[a]	Integer	uniform
	epochs	1 - 500[b]	Integer	early stopping
	confidence scaling	linear, log	Categorical	
	alpha	10^{-3} - $5 \cdot 10^{+1}$ [d]	Real	log-uniform
	epsilon	10^{-3} - 10^{+1} [d]	Real	log-uniform
	reg	10^{-5} - 10^{-2}	Real	log-uniform
EASE^R	l2 norm	10^{0} - 10^{+7}	Real	log-uniform

[a] The number of factors is lower than PureSVD or NFM due to the algorithm being slower.

[b] The number of epochs is lower than SLIM BPR or MF BPR due to the algorithm being slower.

[c] The *negative quota* is the percentage of samples chosen among items unobserved by the user, having a target rating of 0.

[d] The maximum value of this hyperparameter had been suggested in the article proposing the algorithm.

APPENDIX B

Equivalence of Hamming distance and Herfindahl index

In this Appendix we report a side result of this study considering the *mean inter list* diversity metric (MIL), proposed by [135]. MIL represents the average number of recommendations any pair of users has in common. We first show that MIL is equivalent to the average Hamming diversity in a set of strings having all equal length. Secondly, we demonstrate that both can be computed based solely on the number of times each item appears in any recommendation list (or string). Therefore both MIL and Hamming diversity are aggregate diversity metrics. We then show both are equivalent to another metric, the *Herfindahl index*. Lastly, we show how a reranking strategy, previously demonstrated only at an empirical level, optimizes MIL diversity.

B.1 Aggregate diversity metrics

B.1.1 Mathematical notation

We will adopt the following notation. The item set is I, the user set U and the respective cardinality $|I|$, $|U|$. The length of the recommendation list,

i.e. the cutoff, is c, while $rec(i)$ represents the number of times item i has been recommended across all users. The total number of recommendations is $rec_t = \sum_{i \in I} rec(i) = c \cdot |U|$.

B.1.2 Metrics

Among the most used aggregate diversity metrics are Item Coverage, Shannon Entropy, Gini Index and Herfindahl index [38, 88, 90, 135] (see Section 4). These metrics are all functions of the global number of times each item has been recommended.

$$\text{Shannon} = -\sum_{i \in I} \frac{rec(i)}{rec_t} \cdot \ln \frac{rec(i)}{rec_t}$$

$$\text{Gini} = \sum_{i=1}^{|I|} \frac{2i - |I| - 1}{|I|} \cdot \frac{rec(i)}{rec_t}$$

$$\text{Herfindahl} = 1 - \frac{1}{rec_t^2} \sum_{i \in I} rec(i)^2$$

It is possible to see that Shannon Entropy is logarithmic, Gini Index is linear, given that the $rec(i)$ data must be ordered beforehand, and Herfindahl is quadratic. This means that all three metrics measure the same phenomena, how recommendations are spread across items, but are are sensitive to slightly different behaviors. An analogy could be drawn between those aggregate diversity metrics and the prediction error measured by MSE or RMSE, and MAE. While they all measure the same quantity, the prediction error, MSE and RMSE will be much more sensitive to outliers and error variance than MAE, due to their quadratic nature.

B.1.3 Mean inter-list diversity

Mean inter-list diversity (MIL) was presented by [135][1]. It is among the metrics which are computed on the actual recommendations received by each user rather than on the global item count. This diversity considers the uniqueness of different user's recommendation lists and has a value between 0 and 1. The less likely any two users have been recommended the same items, hence the more diverse the recommendations are, the closer

[1] Note that MIL was originally called *Personalization*, however we will not use this name due to the fact that the highest value for this metric (i.e., 1) is obtained by a non personalized Random recommender.

MIL will be to 1.

$$h(ua, ub) = 1 - \frac{q(ua, ub)}{c} \tag{B.1}$$

Equation B.1 represents the *inter-list distance* for two users *ua* and *ub*, where $q(ua, ub)$ is the number of common items in their recommendation lists. Equation B.2 shows how MIL is computed, as an average over all inter-list distances, excluding the diagonal.

$$MIL = \frac{1}{|U|^2 - |U|} \sum_{\substack{(ua,ub)\in U \\ ua \neq ub}} h(ua, ub) \tag{B.2}$$

Computing MIL requires to compute function $q(ua, ub)$ for all couples of users, which is quadratic in their number therefore being very computationally expensive for all but the smallest datasets. Another issue with the formulation of this metric is that it is not entirely clear its relation with other diversity metrics. Clearly it is not an individual diversity metric, since it does not measure the diversity as perceived by a single user. However, its relation to aggregate diversity metrics is not apparent.

B.1.4 Hamming diversity

Hamming diversity is another metric defined between users [121], applied by representing the user's recommendation lists L as a one-hot encoding L_H of $|I|$ elements. The Hamming distance is the number of positions in which the two lists are different. Since the Hamming distance can be computed from $q(ua, ub)$ as $H(ua, ub) = |I| - q(ua, ub)$ Hamming and MIL diversity are equivalent.

B.2 MIL as aggregate diversity

We can now demonstrate that MIL, and therefore Hamming diversity, are in fact aggregate diversity metrics and can be computed based solely on the global item distribution. We assume, without loss of generality, that all users will receive a recommendation list of the same length c. Equations B.1 and B.2 can be re-written in order to isolate function q and highlight how MIL only requires the *global* common item count, $count_g$, rather than the specific *user-user* common item count. See Equations B.3 and B.4.

$$count_g = \sum_{\substack{(ua,ub)\in U \\ ua \neq ub}} q(ua, ub) \tag{B.3}$$

$$MIL = 1 - \frac{1}{|U|^2 - |U|} \cdot \frac{count_g}{c} \tag{B.4}$$

Function $q(\cdot)$ can now be decomposed as a summation of other functions $q_i(\cdot)$, each associated to a specific item i, that will have value 1 if both users have been recommended item i, 0 otherwise. See Equation B.5.

$$q(ua, ub) = \sum_{i \in |I|} q_i(ua, ub) \tag{B.5}$$

$$count_g = \sum_{i \in |I|} \sum_{\substack{(ua,ub) \in U \\ ua \neq ub}} q_i(ua, ub) \tag{B.6}$$

The purpose of function $q_i(\cdot)$ and of swapping the item and users summations, as done in Equation B.6, is to represent $count_g$ in terms of a combinatorial problem that can be easily solved. If we consider the set of all users that received item i in their recommendation list, the summation of $q_i(\cdot)$ across all pairs of users will be equal to the number of non-ordered pairs that can be defined from such set. Since the set contains $rec(i)$ users, the number of non-ordered pairs it allows is $rec(i) \cdot (rec(i) - 1)$.

The global common item count can be therefore represented in terms of the global item distribution, in the following way:

$$count_g = \sum_{i \in |I|} rec(i) \cdot (rec(i) - 1) = -|U| \cdot c + \sum_{i \in |I|} rec(i)^2 \tag{B.7}$$

Equation B.7 has a series of important consequences:

- MIL and Hamming diversity measure aggregate diversity, they do not depend on the specific user recommendation lists but only on the final item occurrence distribution. By using Equation B.7 and B.4 it is now possible to compute MIL in negligible computational time.
- MIL, Hamming diversity and Herfindahl index (HHI) are equivalent, being linear functions of the same quadratic summation of $rec(i)$. Due to their quadratic nature, they will have low sensitivity to items having low number of occurrences and high sensitivity to items being recommended often. This means that a change in the values of $rec(i)$ for infrequent items will cause much smaller variations in the metrics value, than a change involving frequent items.

Moreover, MIL and Hamming diversity are not sensitive to local diversity variance. Just as all the other aggregate diversity metrics, MIL is

not able to distinguish between a recommender yielding to an almost even diversity across all users and one exhibiting high variance for clusters of users. Another useful consequence of what previously demonstrated is that MIL and Hamming diversity of a RS can be steered at a system level. Due to the fact that MIL is a aggregate metric, it is possible to control the diversity of a RS by altering the average probability each item will appear in the recommendation lists, for example via a reranking step.

Value range

Although equivalent, MIL and Herfindahl index have very different value ranges. The RS with the minimal diversity is the one recommending to all users the same items (i.e. a Top Popular). In this case a number of items equal to the length of the recommendation list, c, will have $rec(i) = |U|$ while all the others 0, hence:

$$HHI_{min} = 1 - \frac{1}{c} \qquad MIL_{min} = 0$$

While MIL has a minimum value of 0, HHI has a minimal value of 0.80 for recommendation lists of length 5 and of 0.95 for lists of length 20.

The RS exhibiting maximal diversity is the one able to best balance recommendations across the whole item set, therefore a uniformly random RS. Each item will have an equal value[2] of $rec(i) = \frac{|U| \cdot c}{|I|}$. The resulting bounds are:

$$HHI_{max} = 1 - \frac{1}{|I|} \qquad MIL_{max} = \frac{|U|\,(|I| - c)}{|I|\,(|U| - 1)}$$

As opposed to the very different minimal values, it is possible to empirically show[3] that maximum values for both metrics are between 0.99 and 1.00.

B.2.1 Diversity enhancing reranking

Amongst many techniques that can be applied to improve diversity in recommender systems, [90] proposes *RP*$^3\beta$, an item-based collaborative recommendation algorithm which is based upon another algorithm, *P*$^3\alpha$. Both are simple graph-based algorithms implementing a random walk between users and items and, although being seldom used in literature, they provide very competitive recommendation quality even against the most re-

[2]Note that this is just an approximation not taking into account the fact that $rec(i) \in \mathbb{N}^{0+}$. If the number of items is greater then $|U| \cdot c$ the approximate average item count becomes a real value $rec(i) \in (0, 1)$. In practice a number of items equals to $|U| \cdot c$ would have $rec(i) = 1$ while the others $rec(i) = 0$. In this scenario no pair of users sharing any item could exist, therefore $count_{g-MAX} = 0$ and $InterL = 1$

[3]Computed on the following datasets: MovieLens 20M, BookCrossing, Epinions, NetflixPrize, Spotify Challenge 2018, 30Music, Xing Challenge 2016, Xing Challenge 2017

cent neural models (see Chapter 5). The article proposing RP$^3\beta$ states it was observed that P$^3\alpha$ is very influenced by popularity and has low diversity. Therefore, in order to improve its diversity, they propose a reranking procedure which divides the score of the item as computed by P$^3\alpha$ by its popularity, in order to penalize very popular items. The experimental evaluation of RP$^3\beta$ shows a substantially higher MIL[4] and Gini diversity than P$^3\alpha$. The paper provides an intuitive justification of this reranking procedure but not a mathematical one. Based upon Equation B.7 we can now state that their reranking approach is optimizing MIL diversity. Since P$^3\alpha$ is very sensitive to popularity, the item popularity constitutes a good approximation of the $rec(i)$ function, because popular items will be recommended often while unpopular ones much less frequently. A similar approach could be applied to other recommendation models, although for algorithms less prone to popularity bias the item popularity may not be a good approximation of the $rec(i)$ function and using it as penalizing factor may steer the reranking in the wrong direction.

B.3 Summary

In this appendix we have analyzed different aggregate diversity metrics. Thanks to a different formulation we have shown that MIL and Hamming diversity are equivalent to the Herfindahl index and therefore all are aggregate diversity metrics. This uncovers their previously unknown relationships and allows to compute them in negligible time, avoiding the computationally expensive step of calculating the common items between all pairs of users.

[4]The article reports MIL as *Pers*.

APPENDIX C

Detailed results for all the analyzed algorithms

The detailed results of all analyzed algorithms are available as an online article in the same Github repository that contains the source code for our experiments. The detailed results contain the full evaluation results for all baselines and datasets, the selected hyperparameters for all algorithms as well as the computation time and throughput. Both are accessible at the following URLs:

- Github repository [1]
- Document with the detailed results [2]

[1] https://github.com/MaurizioFD/RecSys2019_DeepLearning_Evaluation

[2] https://github.com/MaurizioFD/RecSys2019_DeepLearning_Evaluation/blob/master/DL_Evaluation_TOIS_Additional_material.pdf

Bibliography

[1] Gediminas Adomavicius and YoungOk Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):896–911, 2012.

[2] Fabio Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2013)*, pages 273–280. ACM, 2013.

[3] S. Antenucci, S. Boglio, E. Chioso, E. Dervishaj, Kang Shuwen, Tommaso Scarlatti, and Maurizio Ferrari Dacrema. Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys 2018)*, 2018. Source: `https://github.com/MaurizioFD/spotify-recsys-challenge`.

[4] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *Proceedings of the Conference on Information and Knowledge Management (CIKM 2009)*, pages 601–610, 2009.

[5] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv:1803.01271*, 2018.

[6] Krisztian Balog, Filip Radlinski, and Shushan Arakelyan. Transparent, scrutable and explainable user models for personalized recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 265–274, New York, NY, USA, 2019. ACM.

[7] Jöran Beel and Stefan Langer. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *Proceedings of the International Conference on Theory and Practice of Digital Libraries (TPDL 2015)*, pages 153–168, 2015.

[8] C Glenn Begley and Lee M Ellis. Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, 2012.

[9] Robert M Bell and Yehuda Koren. Improved neighborhood-based collaborative filtering. In *KDD Cup and Workshop at the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2007)*, pages 7–14, 2007.

[10] Alejandro Bellogín and Alan Said. Offline and online evaluation of recommendations. In Shlomo Berkovsky, Iván Cantador, and Domonkos Tikk, editors, *Collaborative Recommendations*, chapter 9, pages 295–328. WORLD SCIENTIFIC, 2018.

[11] J. Bennett and S. Lanning. The Netflix Prize. In *KDD Cup Workshop 2007*, 2007.

[12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[13] Cesare Bernardis, Maurizio Ferrari Dacrema, and Paolo Cremonesi. A novel graph-based model for hybrid recommendations in cold-start scenarios. *Proceedings of the Late-Breaking Results of the ACM Conference on Recommender Systems (RecSys 2018)*, 2018.

[14] Cesare Bernardis, Maurizio Ferrari Dacrema, and Paolo Cremonesi. Estimating confidence of individual user predictions in item-based recommender systems. In *Proceedings of the ACM Conference on User Modeling, Adaptation and Personalization (UMAP 2019)*, pages 149–156. ACM, 2019.

[15] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Ismir*, volume 2, page 10, 2011.

[16] Homanga Bharadhwaj, Homin Park, and Brian Y. Lim. Recgan: Recurrent generative adversarial networks for recommendation systems. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2018)*, pages 372–376, New York, NY, USA, 2018. ACM.

[17] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the International Conference on Machine Learning (ICML 1998)*, pages 46–54, 1998.

[18] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 1998)*, pages 43–52, 1998.

[19] Karl Broman, Mine Cetinkaya-Rundel, Amy Nussbaum, Christopher Paciorek, Roger Peng, Daniel Turek, and Hadley Wickham. Recommendations to funding agencies for supporting reproducible research. In *American Statistical Association*, volume 2, 2017.

[20] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *arXiv preprint arXiv:1903.03129*, 2019.

[21] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017)*, pages 335–344. ACM, 2017.

[22] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 7–10, New York, NY, USA, 2016. Association for Computing Machinery.

[23] Weiyu Cheng, Yanyan Shen, Yanmin Zhu, and Linpeng Huang. Delf: A dual-embedding based deep latent factor model for recommendation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 3329–3335, 2018.

[24] Evangelia Christodoulou, Jie Ma, Gary S Collins, Ewout W Steyerberg, Jan Y Verbakel, and Ben Van Calster. A systematic review shows no performance benefit of machine learning over logistic regression for clinical prediction models. *Journal of clinical epidemiology*, 2019.

[25] Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

[26] Christian Collberg and Todd A Proebsting. Repeatability in computer systems research. *Communications of the ACM*, 59(3):62–69, 2016.

[27] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. Random walks in recommender systems: exact computation and simulations. In *Proceedings of the International World Wide Web Conference (WWW 2014)*, pages 811–816, 2014.

[28] Luca Luciano Costanzo, Yashar Deldjoo, Maurizio Ferrari Dacrema, Markus Schedl, and Paolo Cremonesi. Towards evaluating user profiling methods based on explicit ratings on item features. *Proceedings of Joint Workshop on Interfaces and Human Decision Making for Recommender Systems (IntRS '19) at the ACM Conference on Recommender Systems (RecSys '19)*, 2019.

[29] Paul Covington, Jay Adams, and Emre Sargin. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the ACM Conference on Recommender systems (RecSys 2016)*, page 191–198, 2016.

[30] Paolo Cremonesi, Maurizio Ferrari Dacrema, Shlomo Berkovsky, Iván Cantador, and Ignacio Fernández-Tobías. *Cross-Domain Recommender Systems*. Springer US, Boston, MA, 2015.

[31] Paolo Cremonesi, Franca Garzotto, and Maurizio Ferrari Dacrema. User preference sources: explicit vs. implicit feedback. In *Collaborative Recommendations: Algorithms, Practical Challenges and Applications*, pages 233–252. World Scientific Publishing Company, 2019.

[32] Paolo Cremonesi, Franca Garzotto, and Roberto Turrin. Investigating the persuasion potential of recommender systems from a quality perspective: An empirical study. *Transactions on Interactive Intelligent Systems*, 2(2):1–41, 2012.

[33] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2010)*, pages 39–46, 2010.

[34] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[35] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems (TOIS)*, 2019.

[36] Edoardo D'Amico, Giovanni Gabbolini, Daniele Montesi, Matteo Moreschini, Federico Parroni, Federico Piccinini, Alberto Rossettini, Alessio Russo Introito, Cesare Bernardis, and Maurizio Ferrari Dacrema. Leveraging laziness, browsing-pattern aware stacked models for sequential accommodation learning to rank. In *Proceedings of the ACM Recommender Systems Challenge 2019 (RecSys 2019)*, page 7. ACM, 2019.

[37] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of Advances in Neural Information Processing Systems (NIPS 2016)*, pages 3844–3852, 2016.

[38] Yashar Deldjoo, Maurizio Ferrari Dacrema, Mihai Gabriel Constantin, Hamid Eghbal-zadeh, Stefano Cereda, Markus Schedl, Bogdan Ionescu, and Paolo Cremonesi. Movie genome: alleviating new item cold start in movie recommendation. *User Modeling and User-Adapted Interaction*, Feb 2019. Source: `https://github.com/MaurizioFD/CFeCBF`.

[39] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[40] David L Donoho, Arian Maleki, Inam Ur Rahman, Morteza Shahram, and Victoria Stodden. Reproducible research in computational harmonic analysis. *Computing in Science & Engineering*, 11(1):8–18, 2008.

[41] Xiaoyu Du, Xiangnan He, Fajie Yuan, Jinhui Tang, Zhiguang Qin, and Tat-Seng Chua. Modeling embedding dimension correlations via convolutional neural collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 2019.

[42] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative memory network for recommendation systems. *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2018)*, 2018.

[43] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the International World Wide Web Conference (WWW 2015)*, pages 278–288. International World Wide Web Conferences Steering Committee, 2015.

[44] Nicolò Felicioni, Andrea Donati, Luca Conterio, Luca Bartoccioni, Davide Yi Xian Hu, Cesare Bernardis, and Maurizio Ferrari Dacrema. Multi-objective blended ensemble for highly imbalanced sequence aware tweet engagement prediction. In *Proceedings of the ACM Recommender Systems Challenge 2020 (RecSys 2020)*, page 8. ACM, 2020.

[45] Maurizio Ferrari Dacrema and Paolo Cremonesi. Eigenvalue analogy for confidence estimation in item-based recommender systems. *Proceedings of the Late-Breaking Results of the ACM Conference on Recommender Systems (RecSys 2018)*, 2018.

[46] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. *Proceedings of the ACM Conference on Recommender Systems (RecSys 2019)*, 2019. Source: `https://github.com/MaurizioFD/RecSys2019_DeepLearning_Evaluation`.

[47] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Methodological issues in recommender systems research. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2029)*. ijcai.org, 2020.

[48] Maurizio Ferrari Dacrema, Alberto Gasparin, and Paolo Cremonesi. Deriving item features relevance from collaborative domain knowledge. *Proceedings of the Workshop on Knowledge-aware and Conversational Recommender Systems 2018 co-located with RecSys 2018*, pages 1–4, 2018.

[49] Maurizio Ferrari Dacrema, Federico Parroni, Paolo Cremonesi, and Dietmar Jannach. Critically examining the claimed value of convolutions over user-item embedding maps for recommender systems. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), October 19–23, 2020, Virtual Event, Ireland*, 2020.

[50] Antonino Freno, Martin Saveski, Rodolphe Jenatton, and Cédric Archambeau. One-pass ranking models for low-latency product recommendations. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2015)*, pages 1789–1798, New York, NY, USA, 2015. ACM.

[51] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. Offline and online evaluation of news recommender systems at Swissinfo.Ch. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2014)*, pages 169–176, 2014.

[52] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. Learning image and user features for recommendation in social networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4274–4282, 2015.

[53] Odd Erik Gundersen. Standing on the feet of giants—reproducibility in ai. *AI Magazine*, 40(4):9–23, 2019.

[54] Odd Erik Gundersen and Sigbjørn Kjensmo. State of the art: Reproducibility in artificial intelligence. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2018)*, 2018.

[55] David J Hand. Classifier technology and the illusion of progress. *Statistical science*, pages 1–14, 2006.

[56] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 2227–2233, 2018.

[57] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the International World Wide Web Conference (WWW 2017)*, pages 173–182, 2017.

[58] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems (NIPS 2014)*, pages 918–926. Curran Associates, Inc., 2014.

[59] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[60] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2018)*, pages 1531–1540. ACM, 2018.

[61] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2008)*, volume 8, pages 263–272. Citeseer, 2008.

[62] Matthew Hutson. Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726, 2018.

[63] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[64] Junyang Jiang, Deqing Yang, Yanghua Xiao, and Chenlu Shen. Convolutional gaussian embeddings for personalized recommendation with uncertainty. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 2642–2648, 2019.

[65] Sadegh Kharazmi, Falk Scholer, David Vallet, and Mark Sanderson. Examining additivity and weak baselines. *ACM Transactions on Information Systems (TOIS)*, 34(4), June 2016.

[66] Yehuda Koren and Robert Bell. *Advances in Collaborative Filtering*, pages 145–186. 2011.

[67] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS 2012)*, pages 1097–1105, 2012.

[68] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324, 1998.

[69] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the International Conference on Machine Learning (ICML 2009)*, pages 609–616, 2009.

[70] Lukas Lerche and Dietmar Jannach. Using graded implicit feedback for bayesian personalized ranking. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2014)*, pages 353–356, New York, NY, USA, 2014. ACM.

[71] Mark Levy and Kris Jack. Efficient top-n recommendation by linear regression. In *RecSys Large Scale Recommender Systems Workshop*, 2013.

[72] Xiaopeng Li and James She. Collaborative variational autoencoder for recommender systems. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2017)*, pages 305–314. ACM, 2017.

[73] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the International World Wide Web Conference (WWW 2018)*, pages 689–698. International World Wide Web Conferences Steering Committee, 2018.

[74] Jimmy Lin. The neural hype and comparisons against weak baselines. *SIGIR Forum*, 52(2):40–51, January 2019.

[75] Jimmy Lin. The neural hype, justified! a recantation. *SIGIR Forum*, 53(2):88–93, 2019.

[76] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[77] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2013)*, pages 3431–3440, 2015.

[78] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[79] Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *User-Modeling and User-Adapted Interaction*, 28(4–5):331–390, 2018.

[80] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2019)*, 2019.

[81] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3), 2018.

[82] Andrii Maksai, Florent Garcin, and Boi Faltings. Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2015)*, pages 179–186, 2015.

[83] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. A contextual attention recurrent architecture for context-aware venue recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2018)*, pages 555–564. ACM, 2018.

[84] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2006)*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[85] Bamshad Mobasher, Xin Jin, and Yanzan Zhou. Semantically enhanced collaborative filtering on the web. In Bettina Berendt, Andreas Hotho, Dunja Mladenič, Maarten van Someren, Myra Spiliopoulou, and Gerd Stumme, editors, *Web Mining: From Web to Semantic Web*, pages 57–76, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[86] ThaiBinh Nguyen and Atsuhiro Takasu. Npe: Neural personalized embedding for collaborative filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 1583–1589. AAAI Press, 2018.

[87] Xia Ning and George Karypis. SLIM: Sparse linear methods for top-n recommender systems. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2011)*, pages 497–506, 2011.

[88] Umberto Panniello, Alexander Tuzhilin, and Michele Gorgoglione. Comparing context-aware recommender systems in terms of accuracy and diversity. *User Modeling and User-Adapted Interaction*, 24(1-2):35–65, 2014.

[89] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings KDD Cup and Workshop*, pages 39–42, 2007.

[90] Bibek Paudel, Fabian Christoffel, Chris Newell, and Abraham Bernstein. Updatable, accurate, diverse, and scalable recommendations for interactive applications. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):1, 2017.

[91] Fernando Benjamín Pérez Maurera, Maurizio Ferrari Dacrema, Lorenzo Saule, Mario Scriminaci, and Paolo Cremonesi. Contentwise impressions: An industrial dataset with impressions included. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), October 19–23, 2020, Virtual Event, Ireland*, 2020.

[92] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

[93] Ali Mustafa Qamar, Éric Gaussier, Jean-Pierre Chevallet, and Joo-Hwee Lim. Similarity learning for nearest neighbor classification. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2008)*, pages 983–988, 2008.

[94] Edward Raff. A step toward quantifying independently reproducible machine learning research. In *Proceedings of Advances in Neural Information Processing Systems (NIPS 2019)*, pages 5486–5496, 2019.

[95] Steffen Rendle. Factorization machines. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2010)*, pages 995—-1000, 2010.

[96] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 2009)*, pages 452–461, 2009.

[97] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *CoRR*, abs/1905.01395, 2019.

[98] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW 1994)*, pages 175–186, 1994.

[99] Marco Rossetti, Fabio Stella, and Markus Zanker. Contrasting offline and online results when evaluating recommendation algorithms. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2016)*, pages 31–34, 2016.

[100] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Proceedings of Advances in Neural Information Processing Systems (NIPS 2017)*, pages 3856–3866, 2017.

[101] Noveen Sachdeva, Kartik Gupta, and Vikram Pudi. Attentive neural architecture incorporating song features for music recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2018)*, pages 417–421, New York, NY, USA, 2018. ACM.

[102] Alan Said and Alejandro Bellogín. Comparative recommender system evaluation: Benchmarking recommendation frameworks. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2014)*, page 129–136, New York, NY, USA, 2014. Association for Computing Machinery.

[103] Alan Said and Alejandro Bellogín. Rival: a toolkit to foster reproducibility in recommender system evaluation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2014)*, pages 371–372, 2014.

[104] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the International World Wide Web Conference (WWW 2001)*, pages 285–295, 2001.

[105] D Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner's curse? on pace, progress, and empirical rigor. 2018.

[106] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *Proceedings of the International World Wide Web Conference (WWW 2019)*, TheWebConf 2019, pages 3251–3257, 2019.

[107] Victoria Stodden. The scientific method in practice: Reproducibility in the computational sciences. 2010.

[108] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2018)*, pages 297–305, New York, NY, USA, 2018. ACM.

[109] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM 2018)*, pages 565–573, 2018.

[110] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the International World Wide Web Conference (WWW 2018)*, pages 729–739. International World Wide Web Conferences Steering Committee, 2018.

[111] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Multi-pointer co-attention networks for recommendation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2018)*, 2018.

[112] Srinivas C Turaga, Joseph F Murray, Viren Jain, Fabian Roth, Moritz Helmstaedter, Kevin Briggman, Winfried Denk, and H Sebastian Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, pages 511–538, 2010.

[113] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[114] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Proceedings of Advances in Neural Information Processing Systems (NIPS 2013)*, pages 2643–2651, 2013.

[115] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3):37–47, 2009.

[116] Flavian Vasile, Elena Smirnova, and Alexis Conneau. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2016)*, pages 225–232, New York, NY, USA, 2016. ACM.

[117] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011)*, pages 448–456. ACM, 2011.

[118] Hao Wang, Binyi Chen, and Wu-Jun Li. Collaborative topic regression with social regularization for tag recommendation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2719–2725, 2013.

[119] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2015)*, pages 1235–1244, 2015.

[120] Jun Wang, Stephen Robertson, Arjen P de Vries, and Marcel JT Reinders. Probabilistic relevance ranking for collaborative filtering. *Information Retrieval*, 11(6):477–497, 2008.

[121] Hao Wu, Xiaohui Cui, Jun He, Bo Li, and Yijian Pei. On improving aggregate recommendation diversity and novelty in folksonomy-based social systems. *Personal and Ubiquitous Computing*, 18(8):1855–1869, 2014.

[122] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM 2016)*, pages 153–162. ACM, 2016.

[123] Xin Xin, Bo Chen, Xiangnan He, Dong Wang, Yue Ding, and Joemon Jose. Cfm: Convolutional factorization machines for context-aware recommendation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 3926–3932, 2019.

[124] Zhenghua Xu, Thomas Lukasiewicz, Cheng Chen, Yishu Miao, and Xiangwu Meng. Tag-aware personalized recommendation using a hybrid deep model. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pages 3196–3202, 2017.

[125] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 3203–3209, 2017.

[126] An Yan, Shuo Cheng, Wang-Cheng Kang, Mengting Wan, and Julian McAuley. Cosrec: 2d convolutional neural networks for sequential recommendation. In *Proceedings of the Conference on Information and Knowledge Management (CIKM 2019)*, pages 2173–2176, 2019.

[127] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. Critically examining the neural hype: Weak baselines and the additivity of effectiveness gains from neural ranking models. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 1129–1132, 2019.

[128] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

[129] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM 2019)*, pages 582–590, 2019.

[130] Bangzuo Zhang, Haobo Zhang, Xiaoxin Sun, Guozhong Feng, and Chunguang He. Integrating an attention mechanism and convolution collaborative filtering for document context-aware rating prediction. *IEEE Access*, pages 3826–3835, 2018.

[131] Quangui Zhang, Longbing Cao, Chengzhang Zhu, Zhiqiang Li, and Jinguang Sun. Coupledcf: Learning explicit and implicit user-item couplings in recommendation for deep collaborative filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 3662–3668, 7 2018.

[132] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52, 2019.

[133] Shuai Zhang, Lina Yao, Aixin Sun, Sen Wang, Guodong Long, and Manqing Dong. Neurec: On nonlinear transformation for personalized ranking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 3669–3675, 2018.

[134] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S. Yu. Spectral collaborative filtering. In *Proceedings of the ACM Conference on Recommender Systems (RecSys 2018)*, pages 311–319, 2018.

[135] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.