Executive Summary of the Thesis

Bayesian Personalized Ranking sampling techniques

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Matteo Moreschini

Advisor: Prof. Paolo Cremonesi

Co-advisor: Cesare Bernardis

Academic year: 2020-2021

1. Introduction

Recommender Systems (RS) play a crucial role in a wide variety of domains. E-commerce websites as Amazon or Alibaba provide users with personalized suggestions for products that they may like, music streaming services like Spotify recommend to each user songs that are close to their taste and the same thing is done by Netflix and YouTube with visual content. Nowadays, most of the data that can be gathered in such systems is not *explicit*, meaning that the content providers are able to collect general information about whether a user interacted with an item on the platform but they have no possibility of directly measuring whether the interaction was a positive or negative one. In fact, even if there are few systems, such as Amazon or Netflix, that let users rate their content (generally in a predefined scale, like 1 to 5 stars on Amazon), only a small percentage of the total users of these platform are willing to rate the content, generally due to their laziness. Moreover, apart from those systems, most of the interactions is totally *implicit* and it is tracked automatically: clicks, views, add to carts or add to playlist etc. In such situations, named *implicit feedback* scenarios, it's often very difficult to be able to infer the user preference over both the rated and unrated items. In this setting, *Bayesian Personalized Ranking* (BPR) is a very popular and widely discussed criterion that, as opposed to its predecessors, treats the top-n recommendation task as a ranking problem. In the context of the BPR learning algorithm, *negative sampling* is the task of selecting negative instances that a user has not rated to learn the model parameters. Several works discussed how the negative sampling quality affects the model recommendation accuracy and, among them, recent ones focused their attention on weighting each sample differently according to the *difficulty* of a negative item.

In our thesis, we analyzed the state of the art negative sampling techniques to detect each strategy's benefits and drawbacks and we proposed an *item similarity based negative confidence* that can be used as a sample weighting strategy which takes into account the probability of the negative item of becoming positive in the future. The experimental results obtained on four popular research datasets show how our proposed technique is able to improve all the methods taken into consideration reducing the impact of the updates of negative samples that have a high chance of becoming positive in the near future.

2. BPR

As our work focuses on part of the Bayesian Personalized Ranking (BPR) learning algorithm, let's first give a brief overview of the BPR criterion. The criterion, proposed by Rendle et al. [3], is formulated for implicit feedback settings, where only positive feedback is observable and the non-observed content is a mixture of missing and negative data. Classical techniques deal with the top-n recommendation problem by predicting a user-specific score for each user-item pair, which should reflect the preference of the user for the item; moreover, the common machine learning approach is to adopt the *missing as negatives* assumption, so that in the training set (i) the positive label is assigned to the observed interactions while (ii) all the non-observed data have negative label. The problem with this approach is that the model is not able to distinguish between the two layers, *negative* and *missing*. To solve this issue, Rendle et al. proposed to create the training data D_S as a set of triples (u, i, j) where u is a user, (u, i) is an observed interaction and (u, j) is an unobserved one. The idea is to use D_S to learn the model parameters by *pairwise learning*, trying to optimize the ranking between item pairs for each user. This approach is totally different from the traditional ones because, instead of trying to score each single item for every user, it is directly optimized for ranking as its goal is to maximize the probability of correctly ranking the unseen items of a user. The definition of the BPR optimization criterion is derived from the Bayesian formulation of the recommendation task, as the maximization of the following posterior probability:

$$p\left(\Theta \mid >_u\right) \propto p\left(>_u \mid \Theta\right) p(\Theta) \tag{1}$$

where Θ represents the parameter vector of an arbitrary model class and $>_u$ is the desired latent personalized ranking for user u. Assuming that (i) users act independently from one another and (ii) the ordering of each pair is independent from the ordering of every other pair, Equation 1 can be rewritten as product of the densities:

$$\prod_{u \in U} p\left(>_u \mid \Theta\right) = \prod_{(u,i,j) \in D_S} p\left(i >_u j \mid \Theta\right) \tag{2}$$

In order to define a total ordering between all the pairs, we have to define a measure of the preference of a user over an item. In particular:

$$p\left(i >_u j \mid \Theta\right) := \sigma\left(\hat{x}_{uij}(\Theta)\right) \tag{3}$$

where σ is the sigmoid function and $\hat{x}_{uij}(\Theta)$ is a real valued function in the parameters Θ which models the user's preferences between the two items i and j. A very interesting point is that $\hat{x}_{uij}$ is a generic and totally arbitrary function, that can be a Matrix Factorization model rather than a kNN. We should note that, since D_S is composed by triplets, the pairwise preference $\hat{x}_{uij}$ is indeed the difference between the individual scores $\hat{x}_{ui}$ and $\hat{x}_{uj}$. Assuming as prior density $p(\Theta)$ a normal distribution with zero mean, we can finally formulate the BPR optimization criterion maximizing the logarithm of the posterior probability in Equation 1:

$$\texttt{BPR-OPT} := \sum_{(u,i,j) \in D_S} \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta \|\Theta\|^2 \tag{4}$$

In order to optimize the model parameters using the BPR criterion in Equation 4, Rendle et al. also proposed a stochastic gradient-descent algorithm [3], called `LEARN-BPR` (Algorithm 1).

Algorithm 1 `LEARN-BPR`

```
1: initialize $\Theta$
2: repeat
3:     draw $(u, i, j)$ from $D_S$
4:     $\Theta \leftarrow \Theta + \alpha \left( \frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta \right)$
5: until convergence
6: return $\hat{\Theta}$
```

The equation at line 4 of Algorithm 1 is exactly the stochastic gradient descent single update for the model parameters, obtained by computing the gradient of the `BPR-OPT` with respect to the model parameters.

As we can see from the pseudocode, at each iteration the `LEARN-BPR` procedure draws a sample from D_S; this task is known as *sampling*. More in details, in the original proposal Rendle et al. suggest a bootstrap sampling approach with replacement for the positive interactions and a uniform sampling for the negative items. This means that first a positive interaction (u, i)

is drawn from the observed data and then a negative one is chosen randomly among the list of non-observed pairs (u, j). The task of selecting the negative item j is commonly defined as *negative sampling*.

3. Uniform Sampling Issues

Although the original BPR paper did not put too much stress on the sampling aspect, the task of drawing informative negative samples was demonstrated to be vital for the learning process of the algorithm so that Rendle published a second paper that directly focuses on this [2]. Looking once again at the `LEARN-BPR` algorithm, we can observe that, at each iteration, the gradient $\frac{\partial}{\partial\Theta}\hat{x}_{uij}$ is multiplied by a factor that can be rewritten as:

$$\begin{aligned} \frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} &= 1 - \frac{1}{1+e^{-\hat{x}_{uij}}} \\ = 1 - \sigma\left(\hat{x}_{uij}\right) &= 1 - \sigma\left(\hat{x}_{ui} - \hat{x}_{uj}\right) \\ &= 1 - p\left(i >_u j\right) = \Delta_{uij} \end{aligned} \tag{5}$$

This quantity depends on how the model is able to discern between the positive item i and the negative one j for user u. In particular, Δ_{uij} represents the *probability of incorrectly ranking* the pair of items i and j, so it will be close to 0 if the model correctly assigns a larger score to item i and close to 1 if, on the contrary, j is incorrectly assigned a larger score than i. In other words, Δ_{uij} represents the *gradient magnitude*, which can be viewed as a measure of the influence that the sample (u, i, j) has in updating the model parameters Θ. The crucial aspect is that if Δ_{uij} is close to 0, then the model parameters are practically not affected by the update of sample (u, i, j), that is to say that the model isn't able to learn anything from (u, i, j) because the gradient vanishes. This analysis suggests that, in order for the model to be able to continue the learning process, the triplets (u, i, j) have to be *difficult* and *informative*, meaning that the model should be uncertain in the ranking of two items. Rendle et al. [2] argued that the originally proposed uniform negative sampler is not able to draw such negative instances. This relates to the fact that item popularity in RS usually follows the *long tail* model, with most of the interactions involving a small portion of the total items. Since (u, i) is a positive interaction, a uniform negative sampler that randomly extracts a negative item from the whole dataset is likely to draw non-popular negative instances j, which will generally have a much lower score with respect to the positive one. In this case, $\sigma\left(\hat{x}_{ui} - \hat{x}_{uj}\right)$ will be close to 1 and, consequently, the gradient magnitude will be close to 0, thus preventing the model from learning anything from the sample, since the pair (i, j) is too easy to rank.

4. Sampling Techniques

From the previous analysis, it should be clear that the negative sampling plays a very important role in the model training. Several techniques have been proposed to improve the sampling strategy; among them, we can identify the following classes:

- *Static - Dynamic*: this distinction is based on whether the sampling probability of a negative item is fixed or it changes at each iteration of the learning process. In the first case, the sampler is said to be *static*, in the other case it is called *dynamic*. Dynamic samplers usually define the sampling probability for an item at iteration k as a function of the model scores at that iteration (in this case, the sampler is also called *adaptive*).
- *Global - Context-dependent*: in *global* sampling techniques, the probability of an item to be sampled is independent from the current user and positive item in the triple while, on the contrary, *context-dependent* strategies employ the context in the sampling process, meaning that the sampling probability distribution of a negative instance is different for every user (or even for each different pair (u, i)).
- *Heuristics - Model Based*: we can name a sampler *heuristic* if it defines the negative sampling probability of the items using heuristics (e.g. the item popularity). Many recent sampling (that we call *model based*) methods instead delegate the job of selecting the negative items to external models; in particular, the most popular and discussed ones make use of GANs. Although these techniques seem particularly appealing, they pose several challenges in the computational complexity, they need a incredible amount of training samples and are often very difficult to tune thus it's tricky to really understand their performance.

5. Related Work

Among the existing techniques, many of them (especially model based ones) often perform some kind of approximation in order to reduce the computational complexity introduced by the technique itself, directly modifying the BPR loss. We did not take such methods into consideration in our work because we wanted to remain as general as possible in the analysis of the sampling methods that work strictly with the original formulation of the BPR and, moreover, that only make use of the user ratings; in other words, our study concentrates on *heuristic* and *single-feedback* sampling methods. After the publication of the BPR orginal paper [3], many works proposed slight variations of the loss function and different sampling strategies. In [2], the authors introduce a popularity oversampler (POP) that favors the selection of negative items with high popularity. In [5], Zhang et al. introduce a ranking-aware rejection strategy (DYN) that is dynamic and adaptive but, instead of computing all the model scores, it restricts the computation to a small group of negative candidates. More recently, [4] suggests a slight alternative of the popularity oversampler called imbalance rejection sampling (IMB) which, in order to sample a negative instance for the positive interaction (u, i), first randomly extracts n items from the unobserved and then loops over these items and selects the first instance with a larger popularity than the positive item i or, if there is no such instance, the item with maximum popularity. This sampler is also used to define the VINS sampler [4], that is dynamic and adaptive and, in addition to the sample, for each triplet (u, i, j) also returns a *sample weight* as a measure of the hardness of finding the violating negative. A similar concept of difficulty is also present in [1], where the authors weighed each sample using the model scores of the negative item normalized over all the scores, as a measure of the probability of the negative item of actually being negative. Since the score needs to be normalized, this strategy requires some approximations because, as for the AOBPR proposed in [2], it would be unfeasible to compute the scores for every single item at each step. The works which are closest to our thesis are [4] and [1], as they propose the concept of sample difficulty as weight that increases or decreases the importance of a sample. One issue that we encountered with these techniques is that they add an extra layer of computational complexity to the algorithm training since they are dynamic and this motivated us to implement a static strategy, in which the difficulty of a sample can be easily computed a priori, so that the weights are fixed and can be retrieved in constant time.

6. Negative Confidence

As already mentioned, in order for the learning process to not become stagnant, the sampler needs to draw *difficult* and *positive-like* negative instances to increase the gradient magnitude and avoid vanishing gradient situations. Although this is definitely a good point from a mathematical perspective, we want to argue that selecting the hardest negative item can decrease the model expressiveness. In fact, one common issue in RS is that the unobserved negative interactions are *exactly* the ones that need to be recommended in the future. In case of adaptive dynamic samplers, which are proven to be state of the art, these techniques at each iteration draw as negatives those item with larger score which, in other words, are exactly items that the model would recommend at that iteration. The problem is that, although those items appear as unobserved feedback for the user in the collected data, they can become future positives and presenting them to the model as negative examples makes the model decrease their scores, thus reducing the probability of those items of being then recommended to the user. This is why [1] introduced the idea of *negative confidence* as the probability of an item of being negative, which is a bit different w.r.t. [4] that defines the difficulty in a way that depends on the algorithm itself. Since we want to avoid introducing additional complexity, instead of using the model scores we define the negative confidence as a function of the collaborative *item-item similarity*. Despite the fact that various works propose to employ the collaborative similarity information inside Matrix Factorization models, to the best of our knowledge there is no such thing in the direct context of the BPR sampling, as most of the heuristics use either the popularity or the underlying model scores.

Let's again denote the sample as (u, i, j), with (u, i) being an observed pair and (u, j) an unobserved one. To measure the confidence of j of being an *actual negative* for u, we could use the item-item similarity between i and j with the following idea: if j is very similar to i, then u would probably like j in the future. Although this confidence would take into account the positive item, it does not consider the user's profile; in order to solve this, we can weight the item-item similarity using u's positive interactions so that the confidence of j of being negative for u is the sum of the similarities between j and the items the u liked in the past. Formally, we can define a personalized confidence score for every pair (u, j) as follows:

$$\overline{r_{uj}} = \sum_{i \in I} r_{ui} \cdot \mathrm{sim}(i, j) \tag{6}$$

where $\mathrm{sim}(i, j)$ is the item-item similarity between i and j. Since scores are relative, it is better for our purpose to formalize this idea with ranks. Our proposed *item similarity based negative confidence* for the pair (u, j) can then be defined as:

$$c(u, j) = \left(\frac{1}{\hat{\rho}_{uj}}\right)^{\alpha} \tag{7}$$

where $\hat{\rho}_{uj}$ is the rank of item j for the user u with respect to the item similarity weighted with u's profile and α is a hyperparamter to be tuned. It is important to highlight that the ranks are defined by sorting the similarity scores in ascending order, which means that a small rank (close to 1) indicates an item with low similarity, while a high one implies a large item similarity score. The role of α is to either smooth or increase the impact of the sample weights and it is dataset specific. We can compute $c(u, j)$ for each unobserved interaction and then use it directly as a sample weight in Equation 4:

$$\sum_{(u,i,j) \in D_S} c(u, j) \cdot \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_{\Theta} \|\Theta\|^2 \tag{8}$$

More in details, if the confidence is close to 1, according to the collaborative item similarity the negative instance is not likely to become a positive one; since the confidence is used as sample weight, the weight is close to 1 and the model update is practically unchanged w.r.t. the original formulation. If the confidence is close to 0 instead, it means that the negative item has a high chance of becoming a future positive for the user and, since we do not want to overly penalize its scores, the effect of this confidence weight is to strongly reduce the impact of this update in the model scores.

Summing up, the item similarity confidence strategy that we just defined has the following properties:

- it's *static*, thus it doesn't add extra complexity to the BPR learning process;
- it's *context-dependent*, in fact the weight are defined for every unobserved pair (u, j);
- it's able to employ the *collaborative item similarity* directly inside the BPR;
- it's *general*, since it only uses the rating matrix, so it can be applied to every implicit feedback scenario and, moreover, also to various other criteria.

7. Results and Discussion

Reproducibility We evaluated our proposed confidence strategy on four research datasets: BookCrossing, LastFMHetRec2011, Movielens1M and Yahoo Movies. As for the preprocessing, we removed *cold* items and users that have less than 5 interactions and we also applied an implicitization step to the explicit dataset BookCrossing, Yahoo Movies and Movielens1M, keeping as positive feedback only the interactions with rating greater than 7 for BookCrossing (that has 1-10 ratings) and greater than 3 for LastFMHetRec2011 and Movielens1M (originally 1-5 star ratings). All the sampling methods have been applied to the BPR algorithm combined with Matrix Factorization. For POP, we used the empirical distribution of the items. For IMB and VINS, we followed the suggestions of the authors [4], setting the group size to 5 for IMB and the max iterations to 64 for VINS. For DYN we tried different group sizes, namely 5, 10, 15 and 20, and we set all the β_k parameters of the distribution to 0 thus always selecting the item with maximum score since this can be computed in linear time $\mathcal{O}(n + 1)$. The implementation was carried out in Python and Cython and all the algorithms ran on a 32GB RAM Windows machine without the use of the GPU. As for parameters tuning, we fixed the learning rate and the batch size to 0.001 and 64 respectively and tried 50 configurations on each sampling.

Prediction Accuracy We measured the prediction accuracy of the algorithms using classification (precision and recall) and ranking (MAP and NDCG) metrics, for different cutoffs. Table 1 shows the recall scores with cutoff at 10; in the upper part of the table we reported the original sampling methods, while in the lower part the same algorithms are weighted with our proposed item similarity based confidence (samplings that start with W are weighted).

	BK	LFM	M1M	YM
UNI	0.064	0.150	0.089	0.238
POP	0.043	0.080	0.050	0.094
IMB	0.062	0.150	0.106	0.247
DYN	<u>0.066</u>	<u>0.153</u>	<u>0.109</u>	<u>0.257</u>
VINS	0.061	0.134	0.075	0.252
W-POP	0.054	0.099	0.053	0.156
W-IMB	0.072	0.163	0.108	0.252
W-DYN	**0.082**	**0.176**	**0.119**	**0.267**
Impr.	*24.2%*	*15.0%*	*9.1%*	*3.9%*

Table 1: *Recall@10* results on the four datasets.

As we can observe, our negative confidence is able to improve the recall of every algorithm on each of the four datasets and the same applies also on the other metrics (that we do not include for space reasons, see the thesis for the full results).

Future Positive Rate In addition to plain accuracy, we also evaluated and discussed the relative improvement that our strategy is able to bring. In particular, we determined that the sampling methods that obtain the largest advantages are the ones that have a high *future positive rate* (FPR), which can be defined as the number of sampled negative items during the training that appear as positives in the test set over total number of sampled items.

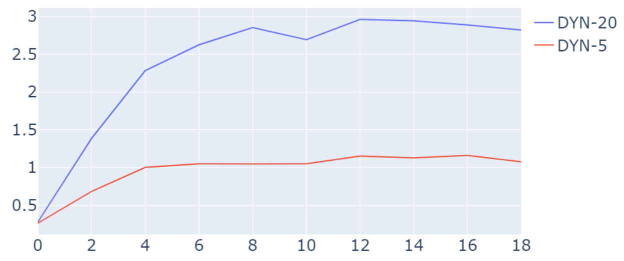


Figure 1: FPR per epoch on LastFm.

We examined the FPR for the different sampling strategies and, in particular, for different group sizes in the DYN samplers, noting that larger groups most benefit from our strategy because they are more prone to select future positives (as you can see in Figure 1, that shows how DYN with groups of 20 almost samples 3x the number of future positives selected by DYN with group size of 5).
We discovered that, as expected, the FPR is closely related to both the model accuracy and the improvement of our strategy, even if it is not enough for evaluating a sampling algorithm as sampling strategies like POP tend to have a high FPR but a low accuracy. In Table 2, we can see that standard non-weighted DYN-5 outperforms the DYN variations with larger group size; on the contrary, the trend is inverted once we apply our confidence weighting strategy that, limiting the penalization of the positive-like items, is able to bring the best advantages to the larger group sizes that have higher FPR.

	Standard	Weighted
DYN-5	**0.203**	0.209
DYN-10	<u>0.198</u>	<u>0.212</u>
DYN-15	0.191	**0.214**
DYN-20	0.180	0.209

Table 2: *NDCG@10* results on Movielens1M for different group sizes of DYN.

Convergence and Complexity Fixing the learning rate and batch size parameters, we measured the convergence speed using the number of epochs. As for the non-weighted strategies, the fastest are the ones that have a high FPR

(POP and DYN with large groups), while the slowest are IMB and UNI. The impact of our proposed confidence weighting is not too well defined, as it seems to slow the convergence on Movielens1M and LastFm while speeding it up a lot on BookCrossing, but one thing that we noticed is that the non-weighted versions (especially on DYN with large groups) tend to have a drop of accuracy right after convergence that does not verify in the case of the weighted strategies. As for the complexity, we already said that our proposed weighting technique is static thus the weights can be computed a priori and retrieved in constant time; moreover, we want to highlight the fact that even a static sampler like IMB, if combined with our strategy, is able to improve dynamic adaptive techniques like DYN that require the computation of the model scores at each iteration (as you can see on BookCrossing and LastFm in Table 1).

8. Conclusions

In our thesis we provided an in-depth analysis of the state of the art sampling techniques for the BPR learning algorithm and we proposed our item similarity based confidence weighting strategy to account in the loss the probability of a negative item of becoming positive. The experimental results show how our strategy is able to improve all the existing baselines reducing the impact of the updates on items that have high probability of being liked by the user in the future; moreover, with respect to the current alternatives, our technique does not introduce additional complexity as the weights can be computed a priori. The main contribution of our work, apart from the introduction of this strategy, should also be stimulating the use of external models not to directly sample the negative instances but rather to compute the sample weights in a similar manner to what we did, in order to reduce the penalization of the items that are likely to become future positives.

References

[1] Defu Lian, Qi Liu, and Enhong Chen. Personalized ranking with importance sampling. In *Proceedings of The Web Conference 2020*, pages 1093–1103, 2020.

[2] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 273–282, 2014.

[3] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[4] Lu Yu, Shichao Pei, Chuxu Zhang, Shangsong Liang, Xiao Bai, Nitesh Chawla, and Xiangliang Zhang. Addressing class-imbalance problem in personalized ranking. *arXiv preprint arXiv:2005.09272*, 2020.

[5] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 785–788, 2013.

POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Bayesian Personalized Ranking Sampling Techniques

ADVISOR:
Prof. Paolo Cremonesi
CO-ADVISOR:
Cesare Bernardis

CANDIDATE:
Matteo Moreschini

Academic Year 2020/2021

Abstract

Recommender Systems (RSs) provide users with personalized suggestions for products of different kinds. Collaborative Filtering (CF) methods leverage past user-item interactions to provide recommendations and, among these techniques, Matrix Factorization (MF) is a well known family which is able to achieve state of the art performances. In the context of Matrix Factorization and particularly in implicit feedback scenarios, Bayesian Personalized Ranking (BPR) is one of the criteria that can be adopted for learning the model weights. This criterion offers the key advantage of dealing with the problem of recommendation as a ranking task, thus maximizing the probability of correctly ranking the unseen items of a user. At each iteration of the BPR learning procedure, a sample composed by a user, a seen item (positive item) and an unseen item (negative item) is extracted in order to update the model weights; this process is known as sampling and it highly affects the model capabilities. Various sampling frameworks have been proposed with the aim of speeding up the computation and finding good negative candidates. Recent works focus on sample weighting strategies that try to take into account the difficulty of a sample. In this thesis we analyze the state of the art sampling techniques and introduce an item similarity based confidence which uses the item-item similarity to infer the negativeness of an item. The confidence is then used as a sample weight in order to decrease the impact of the model updates for those items that, according to the item-item similarity, have a high chance of become positive interactions in the future. The results of the experiments conducted on four popular research datasets show that our proposed confidence weighting strategy is able to achieve very good results both in accuracy and convergence speed without impacting the computational complexity.

Sommario

I Sistemi di Raccomandazione (RS) producono suggerimenti personalizzati per vari tipi di prodotti e contenuti. Gli algoritmi collaborativi (CF) sfruttano le interazioni passate tra utenti e oggetti per costruire le raccomandazioni relative a ciascun utente e, in particolare, tra queste tecniche si distinguono i modelli di fattorizzazione di matrici (MF). Tra questi modelli, specialmente in casi in cui la preferenza degli utenti non è esplicita ma deve essere inferita (feedback implicito), un criterio di ottimizzazione molto popolare è il *Bayesian Personalized Ranking* (BPR). Tale criterio offre il vantaggio di affrontare la questione della raccomandazione come un problema di ordinamento, cercando di massimizzare la probabilità di ordinare in modo corretto un oggetto con cui l'utente ha interagito rispetto ad uno con cui non ha interagito. Nel processo di ottimizzazione del criterio, ad ogni iterazione viene estratta una tripletta composta da un utente, un oggetto con cui l'utente ha interagito (oggetto positivo) e uno con cui non ha interagito (oggetto negativo); questo processo è noto come *sampling* e ha un forte impatto nel processo di allenamento del modello. Varie tecniche di sampling sono state proposte con l'obiettivo di velocizzare la convergenza dell'algoritmo BPR e di migliorare la qualità delle raccomandazione. I lavori più recenti in questo ambito sono incentrati sul concetto di peso della tripletta al fine di tenere in considerazione la difficoltà di un oggetto negativo. In questo lavoro di tesi analizziamo lo stato dell'arte identificando pregi e difetti delle attuali tecniche esistenti e, utilizzando la similarità collaborativa tra gli oggetti, proponiamo il concetto di confidenza di un negativo per ogni singolo utente come misura

della probabilità che possa in futuro diventare un positivo per quell'utente. Tale concetto di confidenza viene utilizzato per assegnare un peso a ciascuna tripletta al fine di diminuire l'impatto dell'aggiornamento del gradiente per gli oggetti negativi che hanno una bassa confidenza. I risultati sperimentali ottenuti su quattro dataset di ricerca dimostrano che la strategia proposta è in grado di migliorare le tecniche esistenti con un impatto minimo sulla complessità computazionale dell'algoritmo.

Contents

List of Figures

Introduction

In the last two decades, Recommender Systems (RS) have become one of the most successful and popular trends in the data mining and machine learning world, both from an industry perspective and from a research one. The need for RS arises from the incredibly fast growth of online content, from movie websites to music streaming services, from traveling agencies to e-commerce websites, and was recently strongly fueled by the increased availability of large datasets and by hardware innovation. In this scenario, providing a personalized aid turns out to be beneficial both for the content providers, that aim at maintaining users on their platform for as long as possible, and for the users, since good personalized recommendations can add another dimension to the user experience and let users discover new contents. In 2003 Amazon published their paper [39] about item collaborative filtering and a later report dated 2013 stated that their recommendation algorithm (such as the ”other customers also bought”) was able to generate 35% of the company's revenue. Netflix started using analytics for recommending DVDs to its users. Time has passed and DVDs have been replaced by streaming services; currently more than 80 percent of the shows people watch on Netflix are discovered through the platform's recommendation system, which is responsible of generating ca. one billion dollars every year. Spotify, YouTube, TikTok and other e-commerce leaders have made recommender systems a salient part of their websites with the aim of producing accurate personalized content for the users in order to keep them on their platform as long as possible.
Recommender System algorithms are classified on the type of information

they use to compute predictions and are typically distinguished in content based approaches and collaborative filtering. While the former methods use additional information about the items, the latter ones try to understand the similarities among the users and the items based only on past interactions. Collaborative filtering methods are considered to be state of the art and, among them, Matrix Factorization excels as a popular class of models for learning the collaborative similarities among users and items. In the context of Matrix Factorization, the Bayesian Personalized Ranking (BPR) is an optimization criterion that, differently from its alternatives, deals with the recommendation task as a ranking task. This makes it perfectly suitable for implicit feedback scenarios, which are situations where the data contains binary information about whether an interaction between a user and an item occurred or not, without further specifying if the interaction was positive or negative.
The main contribution of this thesis is a wide analysis on the existing negative sampling techniques that are used in the BPR learning algorithm and the introduction of a negative confidence sample weighting strategy based on item similarity.

The thesis is organized in four chapters:

- **State of the art**: starting with an overview of the Recommender Systems field, in this chapter we gradually dig deeper into the topics that most matter for our study. It contains an in-depth analysis of the Bayesian Personalized Ranking criterion in its original formulation, as well as an extensive review of the existing sampling strategies.
- **Models**: in this chapter we propose our concept of negative confidence based on item similarity, showing how it can be employed in the BPR loss and explaining the reasons for adopting our strategy over the existing alternatives.
- **Evaluation**: this chapter contains all the information regarding data and metrics that we used for evaluating the BPR sampling algorithms. It also includes the implementation details to reproduce our work.
- **Results**: in the final chapter we present the results of our research. We show that our proposed confidence is able to improve the existing samplings and we discuss some interesting facts that emerged from each dataset.

Chapter 1

State of the art

Recommender Systems (RS) are a collection of data science techniques and software engineering tools that aim at providing suggestions of *"items"* to *"users"*. Generally speaking, we can define the **users** as the main actors that are capable of taking some action in a certain environment, that could be an e-commerce website rather than a social network platform rather than Spotify. On the other hand, the **items** can be defined as the objects that the users interact with; this could be songs in case of Spotify or other users in case of a social network. From this definition, it should be clear that user and items are specific terms used in the RS area and can identify, without loss of generality, any two entities that have an interaction. The **interaction** is the real main character of Recommender Systems. Interaction is an extremely broad term, pretty much any complex system can be seen as one or multiple entities interacting, that means that the RS theory can be applied to several domains and datasets even if their story is very much related to social or marketing platforms.

1.1 Data Structures

In order to tackle a certain problem as a Recommender System one, we want to identify and define some common data elements:

- **Users**: a set of users U.
- **Items**: a set of items I, i.e. the entities the users interact with[1].
- **Ratings**: a set of ratings S, that are the scores assigned by users to the interactions with the items.
- **User features**: set of attributes of the users F_u.
- **Item features**: set of properties of the items F_i.

The research in RS stemmed from the *information retrieval* and *data mining* field; as a result of this, the data is commonly described using matrices and classical data structures:

- **R**: the *User Rating Matrix* is a $|U| \times |I|$ shaped matrix that captures the historical interactions between user and items. A generic cell of the matrix, denoted from now on as r_{ij}, contains the interaction of user i with item j; r_{ij} can be an explicit value in a certain predefined interval representing the preference of user i over the item j or simply a binary value (i.e. 1 if user i interacted with item 0 and a 0 otherwise). Since most of the time the number of items that a user interacted with is very small compared to the total number of items, R is usually an extensive and greatly *sparse* matrix.
- **ICM**: the *Item Content Matrix* is a $|I| \times |F_i|$ shaped matrix used to represent the properties of each item as a vector. The features are a predefined set of properties, highly dependent on the application domain, used to characterize each item. Each feature can take on different values: integer numbers, real numbers, booleans and so on. Since handling different types of data into a single matrix can lead to many difficulties (e.g. computing similarity matrices and products), the item features are usually preprocessed in order to remove non number-like features using different data mining methods, as one-hot encoding or binary encoding.

[1]Generally speaking, I and U are disjoint sets that represent different types of entities, but this is not always true. In a social network like Facebook, for example, users interact with other users, so in this case the set of items coincides with the set of users.

- **UCM**: the *User Content Matrix* is a $|U| \times |F_u|$ matrix that in each row i contains the vector of features of user i. The same considerations we made earlier about the different data types for the ICM can also apply for the UCM.

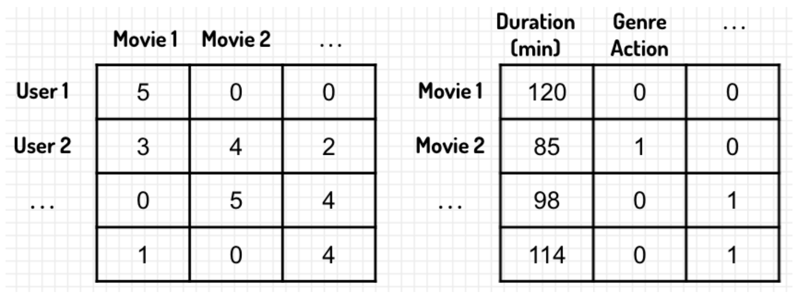
Let's now try to identify these data structures in a simplified real life case: a Netflix[2]-like movie recommendation system. Netflix is a popular video streaming service offering their subscribed users a great variety of visual content that spans from movies and TV series to documentaries and cartoons. Let's define what happens in our simplistic Netflix clone:

- each *user* subscribed to the service is able to choose between a vast selection of *movies*, *documentaries* etc.;
- after watching a *movie*, a *user* leaves a *score* between 0 and 5, with 0 representing the fact that the user totally disliked the movie and 5 the fact that the user really liked it;
- for each *user*, our Netflix clone is able to collect *some minimal information* such as the preferred language, the state where the user is located and the user's date of birth;
- for each *item*, our clone has a *list of features* characterizing the item, for example: the genre, the duration, the date when the content was released, the language etc.

Using the data structures that we just presented, a possible way to model this data is the following:

- **R**: the user rating matrix contains, for each user, the ratings of the items (aka videos) that the user watched, as an integer number between 1 and 5 representing how much user i liked item j or 0 if the user did not watch the item.
- **ICM**: there are different kinds of features for the movies; we can imagine that the genre is a string withing a possible list of genres (hence, a categorical feature), the duration can be expressed as number of minutes, the date could be a timestamp or a string as well. In order to

[2]Netflix and Recommender Systems have always been very close and the ability of their recommendation systems to suggest movies/TV series is one of the main reasons behind the company's fortune. Netflix also organized one of the most popular competitions of RS in 2006, which took almost three years to complete and had a 1 million dollars price.



	Movie 1	Movie 2	. . .
User 1	5	0	0
User 2	3	4	2
. . .	0	5	4
	1	0	4

	Duration (min)	Genre Action	. . .
Movie 1	120	0	0
Movie 2	85	1	0
. . .	98	0	1
	114	0	1

Figure 1.1.1: The dummy rating matrix R (on the left) and ICM (on the right) of the example movie recommendation system.

store this information within the ICM we could apply usual data mining techniques to adjust the data.

- **UCM**: the same reasoning applies for the UCM, as the language and the state could be categorical variables, while the data of birth could be a timestamp or a string, so in this case we also apply some common cleaning and preprocessing techniques.

At the end of this process we have something that looks like the data structures in Figure 1.1.1, which can be used to build our models for recommending new TV shows to the users.

1.2 Data Types

A requirement for the adoption of a RS is the availability of data, which generally describes the past interactions between users and items and identifies which kind of algorithms can be applied to the problem.

1.2.1 Explicit vs. Implicit Feedback

The first and most significant distinction is between an *explicit* and *implicit* dataset:

- **Explicit datasets**: in this case we rely on the user giving us explicit signals about their preferences, so the ratings represent the explicit feedback that users directly assigned to the items they interacted with. An example could be a 0 to 5 star score of Amazon reviews or a 0 to 5 score to a Tripadvisor restaurant. In this case, each cell of the rating matrix is a numeric value in a (usually) predefined scale.
- **Implicit datasets**: the explicit ratings are not available and are binary values (1 or 0) that represent whether a user interacted with an item or not. Implicit feedback indirectly reflects opinions by observing user behaviors such as the purchase history, browsing history, search patterns, clicks and even mouse movements [44, 34, 27].

Explicit dataset are more informative but less common, and they are generally less dense compared to implicit datasets. In the vast majority of real life situations the feedback that we are able to observe is implicit. The vast majority of the research in the field is focused on processing explicit feedback, probably thanks to the convenience and ease of use, in many practical situations, due to the reluctance of users to rate products[3] or limitations of the system that is unable to collect explicit feedback [27], almost every interaction that happens on the Web is a "click" action which is intrinsically not explicit; we can record that some not logged user clicked on some item on our platform but we have no clue on the reason why this happened neither if the interaction was a positive or a negative one. We can point out the fact that is always possible to transform an explicit dataset into an implicit one making some assumptions through a process called **implicitization**, but the reverse is generally not possible; for example, in an e-commerce website that can track the click actions of the user as well as the time that a user spends

[3]This is caused by the "laziness" of users that tend not to leave reviews. It's estimated only around 5% of customers typically provide a review or rating for products on Amazon.

on a page, one could use the time in order to infer the user preference and this approach could work but it's still not a precise nor explicit indication of the user's preference.

An important and remarkable difference between the two types of feedback is that in implicit data there is no way (at least having only the sole user ratings, without additional information) to *distinguish between a positive and a negative interaction*, in fact we only know if a user interacted with an item but we have no information about the user preference since there could be a lot of reasons why the user did not interact with a specific item (such as being unaware of the existence of the item, being unable to consume it due to its price, a limited item availability etc) [44]. This means that in an explicit dataset we can label the interactions as either *positive*, *negative* or *missing*, while with implicit dataset we only have *positive* and *missing* interactions, making it trickier for the recommender to understand the user preferences and make accurate predictions especially because, in general, as we already said only a very small portion of the user rating matrix contains non-zero entries and this creates a great imbalance between the positive class and negative/missing one (this kind of problem is also known as "class imbalance" problem and it's particularly clear in the case of one-class problems like implicit feedback).

Another crucial difference, which is related to what we just explained, is that this kind of data is intrinsically *noisy* [27], we can observe a positive, but there is no way to know if it was an actual positive or a bad one. This is what happens for example on Amazon when a user purchases a set of speakers that end up not working as they should, thus the user gives a bad review to the product; this type of information cannot be reconstructed with implicit feedback.

1.2.2 Item and User Features

Item features are the descriptive attributes of the items, denoting the item's content, while **user features** have the same meaning with respect to the users. The features can appear in various data types and forms; taking a song streaming service:

- *item features*: the attributes of the items/songs could be the duration in seconds, the author, the album, the song's text etc.;
- *user features*: possible user features could be the user's email, name,

preferred language and maybe a bio.

It's worth noting that, while item features are easy to collect as they depend on the system, user features are usually more rare and difficult to collect as most of the interactions that happen online do not require users to be logged in and, even if the user is logged in, it's not always possible to collect that much information due to privacy policies.

1.2.3 Additional Data

In addition to interactions and user/item features, there are other kinds of data that provide meaningful insights for improving the recommender's capabilities but are more specific and domain dependent.

1.2.3.1 Temporal Data

In many settings, the recommendations for an item might *evolve over time* and also the *preference of the user might change* as time passes by. For example, the recommendations for a movie may be very different at the time of release with respect to several years later, and also the taste of a user could change as the user becomes older. In such cases, it is extremely important to incorporate some form of **temporal knowledge** in the recommendation process. Even if so far we considered the rating matrix as the matrix the tracks the historical interactions between user and items without making any distinction between recent and old interactions, nowadays pretty much every kind of data is labeled with some timestamp or date and this additional knowledge can be incorporated in the recommendation algorithms.

1.2.3.2 Session Data

Sometimes, rather than observing the evolution over time, it's also important to understand the best way to recommend items in a certain short *time window*. The concept of **session** is strongly paired with the concept of browsing session, as the continuous period of time that a user spends browsing some platform until the user leaves that platform. In this setting, the information is sometimes collected as a list of historical sessions of the users, that contains the interactions between users and items in a specific time interval.

Session-based recommender systems [62, 49, 48] have recently emerged in such settings to try to investigate the user's preference in a more short-term and dynamic way with respect to classical RS, trying to capture and adapt to the evolving behaviour of the user in a short period of time.

1.2.3.3 Context Information

On top of tracking the interactions, it's sometimes possible to observe data about the environment, or **context**, in which the interaction happens; such contextual information can include time, location or social data. Let's take a movie service platform as an example once more, the context in which users decides to choose a certain movie can vary: were they alone? Were they with the family? Were they with friends? Was it morning, afternoon or evening? Were they eating while watching? Were they watching on a TV, a laptop or an iPad?

Context-aware (or context-based) recommender systems [2, 57] are domain specific RS that exploit this kind of additional data in order to make more accurate predictions based on the situation in which users interact with the items.

1.2.3.4 Social Data

Traditional recommender systems always ignore **social relationships** among users but, in real life, when we are asking our friends for book recommendations or when we browse on YouTube for reviews of a 34 inches monitor, we are actually requesting social recommendations. Social recommendation is a daily occurrence, and we always turn to our friends for recommendations. This fact is also influenced by the boom of Web 2.0 websites and applications, blogs, affiliate marketing websites and YouTube channels. Hence, in order to improve the suggestions and to provide more personalized recommendation results, recent RS models [17, 64] try to incorporate real social life and online social network information among users to improve the recommendation quality.

1.3 Models

The basic and standard models for recommender systems work with two of the data kinds that we explained:

- **user-item interactions**, i.e. the user rating matrix; such methods are referred to as collaborative filtering methods, which are techniques that rely on the historical user behavior, using only past **ratings**, regardless of the item or user features.

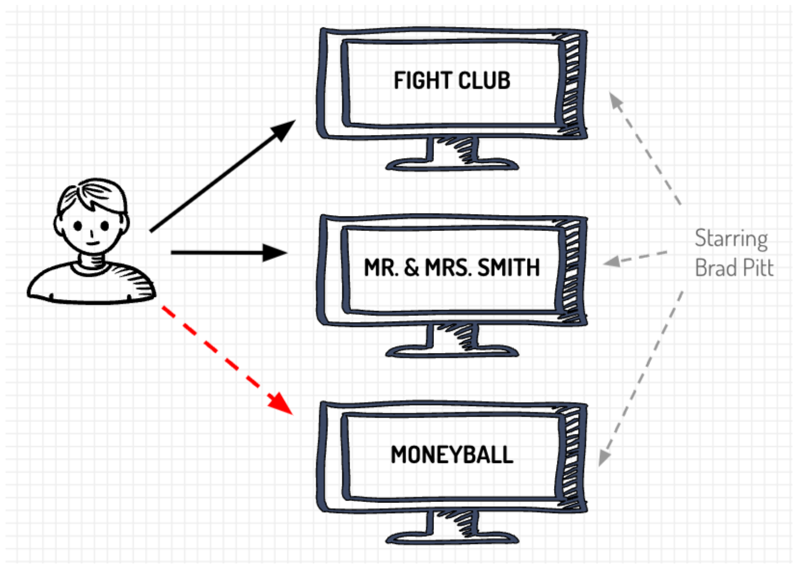


Figure 1.3.1: Simple content based approach for a movie recommender system.

- **item and/or user features**, i.e. the ICM and/or the UCM; these algorithms are called content based recommenders. Content-based recommender systems try to match users to items that are similar to what they have liked in the past. These approaches, for a specific user, don't try to find a correlation with other users but rely only on the user's past interactions to find common attributes among the items the user interacted with, so the focus is only on the target user's own ratings and the attributes of the items liked by the user, the ratings of the other users usually play no role in this scenario (Figure 1.3.1).

In addition to these models, more advanced techniques are hybrid/ensemble models of these approaches or domain specific families of recommender systems like the one the we briefly introduced earlier. In the following sections we discuss the collaborative filtering methods.

1.3.1 Collaborative Filtering

The idea behind Collaborative Filtering[4] is simple: since the observed ratings are often highly correlated, we can use the collaborative power of the ratings given by other like-minded users in order to infer the missing ratings for a specific user [27]. A big advantage that makes these methods very appealing is that, since they only take advantage of the rating matrix, they are domain

[4]The term was coined by the developers of Tapestry as a way to gather qualitative data. It was developed at Xerox PARC as a way to handle the large amounts of email and messages.

free and yet they can address aspects that are often elusive and hidden to content based approaches. These methods often use some kind of *similarity metric* between the items or between the users.
Collaborative filtering can be viewed as a special case of *missing matrix values analysis*, which studies the problem of finding the latent entries in a matrix. From a different perspective, collaborative filtering can also be interpreted as a generalization of the classification and regression tasks, considering the columns as features (that can be missing for certain rows) and performing the prediction phase in entry-wise fashion rather than row-wise fashion [3].
One noticeable aspect is that a matrix completion setting is inherently *transductive*, that means that the test instances are also included during the training process, since the rating matrix is a $|U| \times |I|$ matrix of all the users and items; this fact often makes it hard for the collaborative recommender to produce recommendations for test instances that are not available at the time of training. This does not happen in algorithms that are *inductive* such as naive Bayes.
One of the main challenges in developing collaborative filtering techniques is that the rating matrix is a greatly sparse matrix, most of the data (usually 97/98%) is composed by zeroes, i.e. missing values. Taking as example a music platform like Spotify, most users would have listened to a very small portion of the immense universe of available songs and this reflects in the rating matrix with a lot of missing/unspecified/unobserved data, making it difficult for the recommender to suggest items to a user with very few interactions and also to suggest to a user those items that have few interactions.

We can distinguish two families of collaborative filtering methods: **memory based** and **model based**. The first techniques often use some kind of similarity heuristic, thus we present some notions about similarities and then we illustrate the most popular memory based and model based algorithms.

1.4 Similarity Metrics

Recommender systems algorithms often require a definition of an appropriate similarity function to measure how much users or items differ from each other. For this purpose, usually users and items are represented by vectors so that data mining and information retrieval derived approaches studied for these type of structures can be adopted. In the following definitions, x and y will be considered to be representations of users or items lying in an n-dimensional space, without loss of generality.

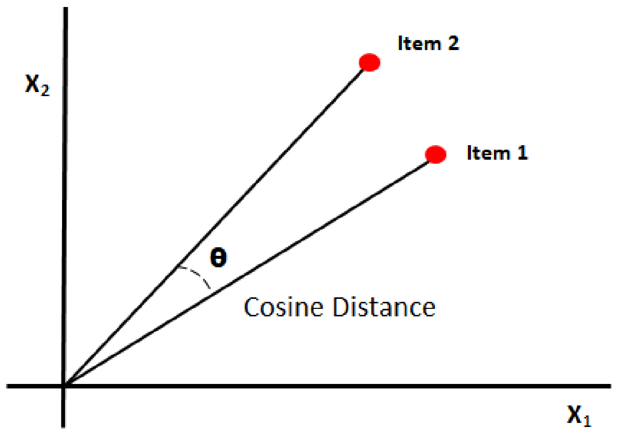


Figure 1.4.1: Cosine similarity graphical explanation.

1.4.1 Cosine Similarity

Cosine similarity is one of the most basic yet powerful similarity metrics. The idea is the following: measuring the cosine of the angle formed by the two input vectors (see figure 1.4.1), knowing that the cosine it's a real valued function that will be 1 if the two vectors share the same direction and verse, 0 if the two vectors are orthogonal and -1 if they share the same direction but have opposite verse.

Given two input vectors, x and y, the cosine similarity $cos(x,y)$ is defined as:

$$\cos(x,y) = \frac{x \cdot y}{||x|| \cdot ||y|| + H} \tag{1.1}$$

where H is the shrink coefficient (hyperparameter to be tuned) use to penalize the vectors with a small number of observed interactions (i.e. 1 in the vectors) and $||x||$ indicates the norm of the vector x.

1.4.2 Pearson Coefficient

The **Pearson correlation coefficient** (also known as Person's r) quantifies the linear correlation between two sets of data.

The formula for computing the coefficient is the following:

$$\text{Pearson}(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x}^2)}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y}^2)}} \tag{1.2}$$

1.4.3 Jaccard Coefficient

Jaccard similarity is a set similarity metric, measuring the percentage of attributes two sets X and Y share, i.e. generally the ratio of intersection over union:

$$\text{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{1.3}$$

Generally, if both X and Y are empty, the Jaccard coefficient is assumed to be 1. The Jaccard distance, which measures *dissimilarity* between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union.

1.5 Tanimoto Coefficient

Tanimoto reformulated the Jaccard coefficient to real valued vectors, under the name of Tanimoto coefficient or Extended Jaccard coefficient. The formula is the following:

$$\text{Tanimoto}(x, y) = \frac{x \cdot y}{||x||^2 + ||y||^2 - x \cdot y} \tag{1.4}$$

In case of binary vectors, the **Tanimoto coefficient** coincides with the Jaccard coefficient.

1.6 Memory Based Collaborative Filtering

In memory based techniques, the missing ratings for user-item couples are inferred on the basis of their neighbors using some sort of heuristic or similarity measure (see Appendix 1.4). These techniques *directly* use the given ratings in order to compute the predictions and consequently they are both easy to implement and to explain, and were among the first collaborative filtering algorithms.

1.6.1 Top Popular

Top Popular is a **non-personalized technique**, which means that recommends the same set of items to every user regardless of the user's profile. The top popular algorithm simply recommends the items to every user, ordering

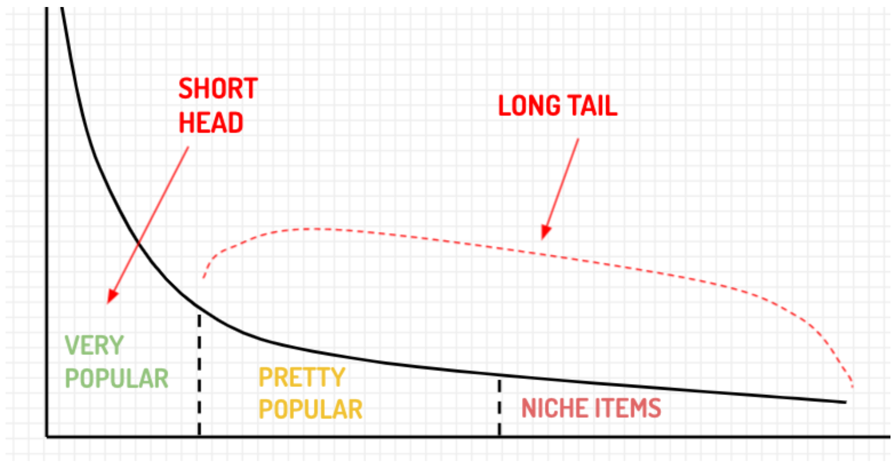


Figure 1.6.1: The long tail model.

them by their popularity (i.e. the number of interactions for each item)[5]. This heuristic is often used as a baseline model for comparing the performance of other approaches.
RS datasets tend to be very *popularity biased*, most of the interactions involve only a small portion of the dataset, and collaborative techniques often emphasizes the popular items in the so called "short head" over "long tail" items [46, 1] that are popular among small groups of users (Figure 1.6.1).

1.6.2 User k-Nearest Neighbors

User based collaborative filtering (UserKNN) is a neighborhood method that computes the **similarity between the rows** of the rating matrix, i.e. the similarity between the users' profiles. The idea behind this method is that if two users, u and v, have similar past user's profile, it's likely the items that are positive interactions for v and are still unobserved for u would represent good recommendations.
In this setting, the rating for the unobserved user-item pair (u, i) is computed using the ratings given by the users that are most similar to u. In order to do that, a similarity metric $sim(u, v)$ should be defined in order to quantify how much similar is the taste of the two users[6]. The predicted score, using the $|U| \times |U|$ shaped similarity matrix sim, is then estimated as:

[5]This can be easily computed as the sum of the item column in the rating matrix.
[6]Usually, some normalization technique like TF-IDF or BM25 is applied to the rating matrix before computing the rating matrix.

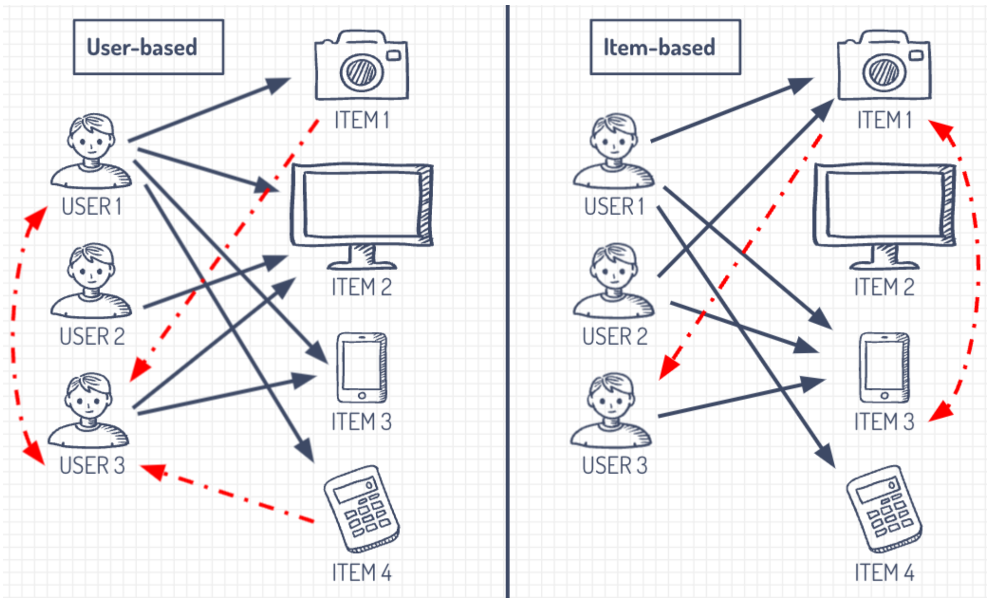


Figure 1.6.2: On the left, a user-based approach identifies the similarity between user 1 and user 3 and thus suggests to user 3 an item that user 1 rated. On the right, since item 1 and item 3 are similar and user 3 has interacted with item 3, item 1 is recommended to user 3.

$$\overline{r_{ui}} = \frac{\sum_{v \in U} r_{vi}\, \mathrm{sim}(u,v)}{\sum_{v \in U} |\mathrm{sim}(u,v)|} \tag{1.5}$$

1.6.3 Item k-Nearest Neighbors

Item based collaborative filtering (ItemKNN) computes the **similarity between items** implementing the idea of recommending to user u the items that are similar to the ones that are in u's profile. The similarity metric, in this case, is computed between the columns of the R and has a $|I| \times |I|$ shape. The rating for user u and item i will be predicted as:

$$\overline{r_{ui}} = \frac{\sum_{j \in I} r_{uj}\, \mathrm{sim}(i,j)}{\sum_{j \in I} |\mathrm{sim}(i,j)|} \tag{1.6}$$

The difference between item-based and user-based techniques are exampli-fied in figure 1.6.2.

1.6.4 Neighborhood-Based Graph Models

Another approach for dealing with the recommendation task from a different perspective consists of using the **graph theory**.

1.6.4.1 Graph Approach Intuition

Network analysis is a versatile tool in uncovering the principles of many complex systems, especially the ones in which the centrality of the information is in the relationship between the nodes rather than in the nodes themselves. This perfectly applies to RS. Many technological, physical and biological systems can be viewed as graphs, with nodes representing the individual entities and edges capturing the interactions between them. A RS such as the one we saw in the example in Section 1.1 could be defined, using the graph theory, as a *bipartite*, *directed* and *weighted* graph. Let's give some notions to better understand what this means:

- **Graph**: a graph (or, equivalently, network) G is an ordered pair of disjoint sets (V, E) where V is the set of nodes (or, equivalently, vertices) and E is the set of edges which is a subset of the cartesian product of the nodes $V \times V$.
- **Directed** graph: a directed graph is a graph in which the edge joining two nodes has a meaningful direction, that means that if $a, b \in V$ the relationship $a \rightarrow b$ (with the arrow representing the edge) has a different meaning with respect to $a \leftarrow b$ (and the two edges can be present simultaneously). On the other hand, if $a \rightarrow b$ and $a \leftarrow b$ aren't distinct, we label the graph as undirected.
- **Bipartite** graph: a graph $G(V, E)$ is said to be bipartite if there exists a partition (V_1, V_2) such that $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ and every edge connects exactly a node of V_1 and a node of V_2.
- **Weighted** graph: a weighted graph (sometimes referred to as edge-weighted graph) is a graph denoted by a triple $G(V, E, W)$ in which V is the set of nodes, E is the set of edges and W is the set of weights (numbers) assigned to each edge.

Let's take the previous example of a Netflix-like video streaming platform and try to model the data using the definitions that we just gave.

As previously stated, there are two main types of entities, users and TV shows, that will be the two types of nodes of our *bipartite graph* U (set of users) and I (set of movies, or items in the RS theory). The edges of the graph capture the relationships between its nodes and in this case we can state that users interact with movies when they see the movie, hence we can define a *HAS SEEN* relationship. In this model, an edge $u_1 \rightarrow i_1$ connecting $u_1 \in U$

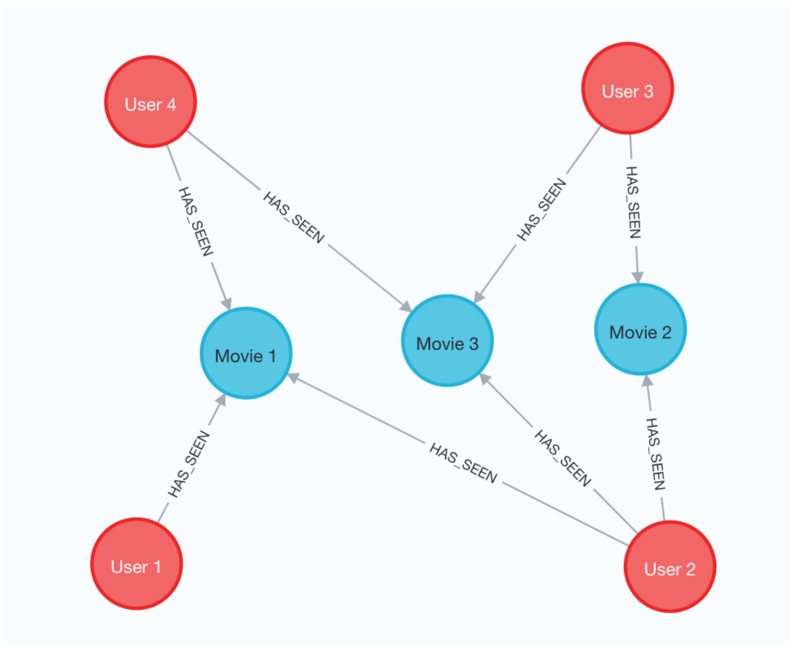


Figure 1.6.3: A possible graph representation of a simplistic movie recommendation engine, using Neo4j, a popular graph database engine.

to $i_1 \to I$ represent the fact the user u_1 saw the movie i_1. An example of the resulting graph using the former approach (with features incapsulated in nodes and edges) is in figure 1.6.3.

Our graph is *bipartite* with heterogeneous nodes and it's *directed*, since the edges are only possible from users to movies, but it's not weighted yet. In our Netflix clone, users are able to give a feedback on the movie they see, so we can store this information at *edge level* attaching a weight to each edge representing the preference of the user on that specific movie; the weight will be an integer number in range 1 to 5[7]. A classical way to store this data is by using the **adjacency matrix** A, which is the matricial representation of a graph with a row for each user node and a columns for each item node; the values of each cell, if the graph is unweighted, A_{ij} will be 0 if there is no edge between i and j or 1 otherwise (if the graph is weighted, the 1s are replaced with the rating given by i to j). So far we only considered interactions, but we also have access to user and items features. This kind of information can be incapsulated in our graph at *node level* or adding *feature nodes* that are

[7]Notice that in this case it makes no sense to have 0 weighted edges, since in the rating matrix a 0 means that user did not interacted with the item and this naturally reflects in the graph by not having an edge between that user and that item node.

connected to the specific node that has that property[8]. Once we have put our data in the graph structures, we can handle the problem as a **link prediction** one, whose goal is trying to predict the missing edges between users and items.

This graph approach is really helpful for visualization and became very popular in the last few years thanks to the great interest in graph machine learning models such as GraphSAGE [18, 61]. It is evident that the two approaches are completely interchangeable (the rating matrix in figure is exactly the weighted adjacency matrix of the graph in Figure 1.6.3) and are indeed two equivalent ways to deal with the same problem, but from two different perspectives.

1.6.4.2 Graph Collaborative Filtering

Graphs are a powerful abstraction that enable many algorithmic tools from the network domain providing a structural representation of the user-item interactions. Since the sparsity of the user rating matrix causes significant challenges in the computation of similarities, graph methods try to tackle the problem from a different perspective.

In the setting of user-based collaborative filtering, the neighborhood of a user is defined by the set of users that are encountered frequently in a *random walk* starting from that user (the same applies for an item-based setting); the expected frequency the random walks can be measured using one of the many algorithms like PageRank [43] or SimRank [30] that were born primarily to respond to the needs of Web-ranking applications and search engines.

The concept of **neighborhood** has been defined using the concept of random walk. A **random walk** process on a graph can be seen as a discrete "memorylessness" Markov chain, where an hypothetical walker starts on a node and at each time step it chooses one of its neighbours at random and moves to it without any dependence on the past actions and states. A Markov chain is characterized by a special matrix called **transition matrix** P, a non-negative real valued matrix in which every cell P_{ij} contains the probability of reaching the node j starting from i. The transition matrix P can be obtained by the (possibly weighted) adjacency matrix:

[8]Note that adding "Feature" nodes would make the graph tripartite. Feature nodes will also have their own adjacency matrix with users and items.

$$P_{ij} = \frac{A_{ij}}{\sum_{k \in V} A_{ij}} \tag{1.7}$$

There are several methods, such as P^3_α and RP^3_β, that use the transition matrix that we just defined in order to compute user/item collaborative filtering.

1.7 Model Based Collaborative Filtering

Model based methods differentiate from the memeory based techniques as they often utilize machine learning or data mining concepts in order to make predictions.
Model based recommenders offer a few advantages:

- **Space Efficiency**: kNN algorithms require computation that grows with both the number of users and the number of items. While memory based methods use the user's ratings directly to compute the predictions, model based techniques learn a model of the user rating.
- **Prediction Speed**: for what we just explained about space efficiency, differently from memory based approaches, these methods do not rely on the whole dataset every time to compute the predictions and this makes the recommendation process much faster.
- **Avoiding Overfitting**: due to their summarization and generalization approach, these methods often help in reducing the overfitting in the recommendations.
- **Synonymy**: in real life, different product/item names can refer to the similar objects. This latent correlation cannot be identified by kNN methods.

These methods include naive Bayes, regression and decision trees and latent factor models.

1.7.1 SLIM

Sparse Linear Method is an algorithm proposed by Ning and Karypis for top-N recommendations, based on the item-item regression [41]. This family of models is referred to as **sparse linear models** because they encourage sparsity in the regression coefficients with the use of regularization methods, learning a sparse aggregation coefficient matrix W, $|I| \times |I|$, using the rating matrix; more in detail, the W matrix is obtained as:

$$\underset{W}{\operatorname{argmin}} \frac{1}{2} \|R - R \odot W\|_F^2 + \frac{\beta}{2} \|W\|_f^2 + \lambda \|W\|_1 \tag{1.8}$$

While other methods assume to have mean-centered ratings, sparse linear models assume to have non-negative rating values, hence every value of W is positive, which makes SLIM more suitable for the implicit feedback setting. The predicted score for an unobserved pair (u, i) is then calculated as a sparse aggregation of items that have already been rated by u:

$$\overline{r_{ui}} = \sum_{j \in I} r_{uj} \mathrm{W}(j, i) \tag{1.9}$$

1.7.2 Matrix Factorization

Matrix factorization methods provide a neat way to leverage all row and column correlations at once to estimate the entire data matrix in a **lower dimension** [34]. The key idea in **dimensionality reduction** methods is that the reduced, rotated, and completely specified representation can be robustly estimated from an incomplete data matrix. Latent factor models try to explain the ratings by characterizing both items and users on an arbitrary number of **latent factors** inferred from the ratings patterns, exploiting the fact that significant portions of the rows and columns of data matrices are highly correlated. A high correspondence between item and user factors leads to a high predicted rating and, consequently, to a recommendation.

Matrix factorization models have gained popularity in recent years as they are able to combine a good scalability with a significant accuracy and are considered to be state of the art in large scale recommendation tasks because, differently from other collaborative methods, they also allow the incorporation of *additional data* such as implicit feedback, temporal knowledge, context information and confidence levels, simply adding new dimensions to the existing user-item rating matrix.

In its basic form, MF consists in representing users and items in a latent factor space of dimension f, using two matrices, namely $Q \in R^{|U| \times f}$ and $P \in R^{|I| \times f}$. The main challenge is computing the mapping of each item and user to factor vectors, once we have the mapping the ratings can be computed as follows (in matrix notation):

$$\overline{R} = \mathrm{P}\,\mathrm{Q}^T \tag{1.10}$$

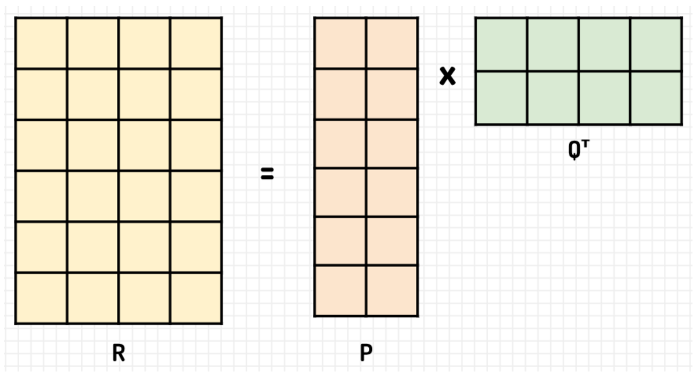


Figure 1.7.1: The basic matrix factorization approach.

Figure 1.7.1 shows a graphical representation of the matrix decomposition.

Researches proposed various methods for learning the P and Q latent matrices.

1.7.2.1 SVD

Singular Value Decomposition is a well-established information retrieval technique for identifying latent semantic factors matrix and thus producing low-rank approximations [34, 58]. In the collaborative filtering domain, SVD requires factoring the rating matrix (Figure 1.7.2):

$$\mathrm{SVD}(R) = U \times S \times V^T \tag{1.11}$$

U and V are orthogonal matrices, respectively called left and right singular matrices, of size $m \times r$ and $n \times r$, with r being the rank of the rating matrix. S is a $r \times r$ shaped diagonal matrix with all the singular values of the rating matrix as diagonal entries. U describes the relationships between users and latent factors, S represents the strength of each latent factor and V contains the similarities between latent factors and items. The matrices obtained by performing SVD are particularly appealing because SVD provides the best lower rank approximations of the rating matrix in terms of Frobenius norm [53].

This mathematical process often raises difficulties due to the high sparsity of ratings, since conventional SVD is undefined when the knowledge about the matrix is incomplete. E-commerce sites like Amazon have tremendous amount of users visits per day, and recommending products to these large number of customers in real-time requires the underlying recommendation

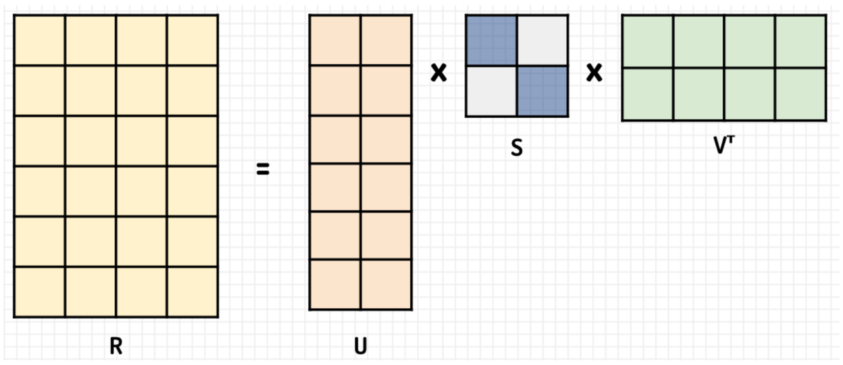


Figure 1.7.2: The SVD of the user rating matrix.

engine to be highly scalable. Recommendation systems usually divide the prediction process into two steps: offline and online. In case of SVD, the offline component requires more time compared to the correlation-based algorithm. Moreover, inaccurate imputation might distort the data and produce inefficient recommendations and SVD has been shown to be prone to overfitting. Several versions of SVD have been proposed in literature to overcome these drawbacks.

1.7.2.2 Funk SVD

This algorithm was proposed by Simon Funk as a special kind of SVD that loops through all the ratings in the training set. Funk SVD uses the following regularized loss function:

$$\sum_{(u,i)\in\kappa} (r_{ui} - q_i \cdot p_u)^2 + \lambda_q \|q_i\|^2 + \lambda_p \|p_u\|^2 \tag{1.12}$$

The idea is learning the latent factors in such a way that they can mimic the known ratings as closely as possible. Funk SVD popularized the stochastic gradient descent optimization for MF. Funk SVD proved to be prone to overfitting, i.e. to approximate just the known ratings but failing to provide good estimates for the unknown ones, so choosing an appropriate value for λ is crucial.

1.7.2.3 ALS

Alternating Least Squares optimization method basically iterates over the following steps until convergence is reached:

- Step 1: keeping P fixed, solve for each of the $|I|$ rows of Q by treating the problem as a least-squares regression problem, using only the observed ratings. The objective function to maximize is :

$$\sum_{u:r_{ui}>0} \left(r_{ui} - \sum_{s}^{f} P_{us} \cdot Q_{is} \right)^2 \tag{1.13}$$

- Step 2: the same thing is done for the items, keeping Q fixed and solving for each of the $|U|$ rows of Q by treating the problem as a least-squares regression problem, using only the observed ratings. The objective function is the counterpart of in Equation 1.13 with respect to items.

1.7.2.4 Bayesian Personalized Ranking

Rendle proposed **BPR-OPT** as a generic optimization criterion [51], derived from the Bayesian analysis of the task, that can be applied to a wide variety of algorithms, ranging from Matrix Factorization to adaptive k-nearest-neighbors. Since this algorithm is the main focus of this thesis, we will go more into more detail in the next section.

1.8 Bayesian Personalized Ranking

All the methods that we presented so far are designed for the top N item prediction task of personalized ranking, but actually *none of them is directly optimized for ranking*. The great innovation of the Bayesian Personalized Ranking (BPR) criterion is dealing with the problem of recommendation as a **ranking task**, thus maximizing the probability of correctly ranking the unseen items of a user.
The BPR criterion is formulated for **implicit feedback** scenarios, where the behaviour of the user is not explicit but needs to be inferred[9] as discussed in Section 1.2.1.

The usual approach for item recommenders is to predict a personalized score r_{ui} for each item, reflecting the preference of the user for the item. Items are then ranked by sorting them according to that predicted score. In implicit settings, the unobserved interactions are a mixture of negative and missing (i.e. possibly future positives) values. The way typical machine learning

[9]The criterion can be applied to explicit dataset thorough the process of implicitization previously discussed.

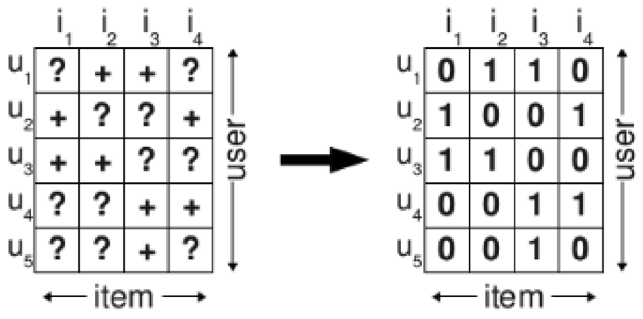


Figure 1.8.1: Missing as negatives assumption.

methods for item recommendation deal with the missing values issue is to *ignore* the hidden distinction between negative and missing values (Figure 1.8.1). The training set is created as the pairs (u, i) with positive label 1 and the rest of the dataset is considered negative. This way, the machine learning method is optimized to predict the value 1 for elements that are positive interactions in the training set and 0 for the rest. What this means is that a model with enough expressiveness that perfectly fits the training data becomes *unable to rank*, as the missing interactions that could possibly become positive in the future are presented as negative feedback. The only reason why this does not happen most of the time is that these methods heavily rely on *regularization techniques* to prevent overfitting[10] [51].

The approach proposed by Rendle is novel and different: rather than *scoring* single items, the model parameters are learned by *pairwise ranking*, where the goal is trying to *optimize the ranking between item pairs for each users*. This idea can be formulated as a real life example in a movie recommender system such as Netflix in which the user can decide which movie to watch within a certain set, expressing the preference of an item over another rather than an absolute scored preference. From the original dataset S, the construction of the training data D_S is based on the following **assumptions**:

- for every observed pair (u, i), the user prefers this item over all other non-observed items;
- for two items that a user viewed, we cannot infer any preference (since we are in an implicit setting);

[10]An overfitted model is a statistical model that contains more parameters than can be justified by the data and lacks of generalization abilities.

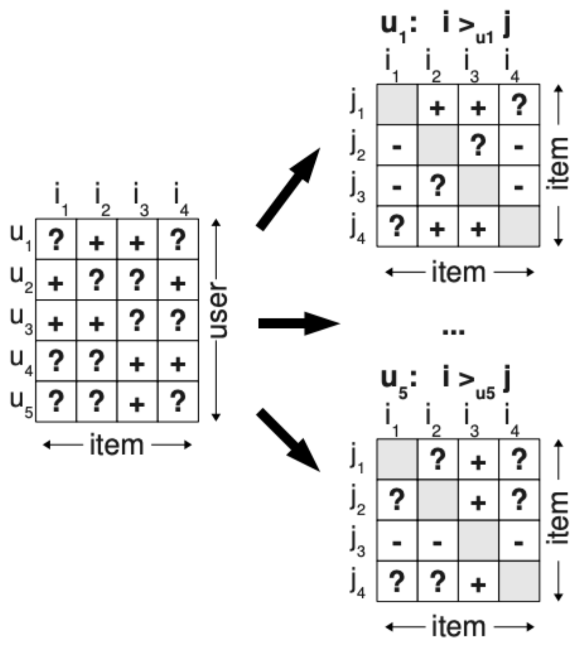


Figure 1.8.2: The BPR approach for constructing the training data.

- for two items that a user did not view, again we cannot infer any preference.

The result of this process is exemplified in figure 1.8.2. The training data can be formalized as:

$$D_S := \{(u,i,j) \mid i \in I_u^+ \wedge j \in I \backslash I_u^+\} \tag{1.14}$$

With I_u^+ being the set of items the user u interacted with. This approach has the big advantage of **distinguishing the negative layer from the missing layer**, since the missing values between two non-observed items are exactly the item pairs that have to be ranked in the future. Moreover, the dataset perfectly suits the ranking objective, D_S is a subset of the global preference of the user that needs to be inferred.

The Bayesian formulation of the personalized ranking task is maximizing the following posterior probability:

$$p(\Theta \mid >_u) \propto p(>_u \mid \Theta)\, p(\Theta) \tag{1.15}$$

where Θ represents the parameter vector of an arbitrary model class and $>_u$

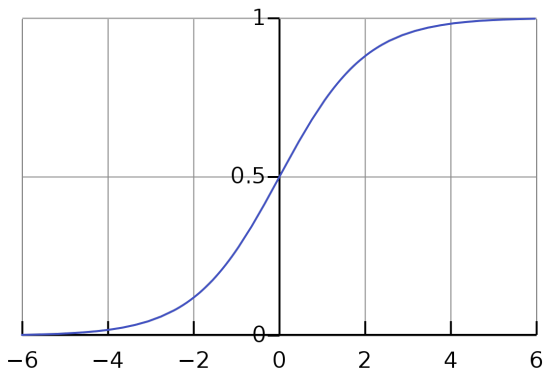


Figure 1.8.3: The sigmoid function.

is the desired latent personalized ranking for user u. In order for the ranking $>_u$ to be a total ordering, it needs to meet three sound properties:

- *totality*: for each couple of distinct items i and j, either $i >_u j$ or $j >_u i$;
- *anti-symmetry*: for each couple of items i and j, if $i >_u j$ and $j >_u i$ than i and j are the same item ($i = j$);
- *transitivity*: for each triplet of items i, j and k, if $i >_u j$ and $k >_u k$ than also $i >_u k$.

To achieve these properties, there are two main and **fundamental assumptions**:

1. all the users are expected to act *independently*;
2. the ordering of each pair of items (i, j) for a specific user is *independent* of the ordering of every other pair.

With these assumptions we can rewrite the previous probability as product of densities:

$$\prod_{u \in U} p\,(>_u \mid \Theta) = \prod_{(u,i,j) \in D_S} p\,(i >_u j \mid \Theta) \tag{1.16}$$

In order to meet the previous sound properties, we now need to define a **scoring function** representing the probability that user u prefers i over j:

$$p\,(i >_u j \mid \Theta) := \sigma\left(\hat{x}_{uij}(\Theta)\right) \tag{1.17}$$

Where:

- σ is the logistic sigmoid function (Figure 1.8.3):

$$\sigma(x) := \frac{1}{1 + e^{-x}} \tag{1.18}$$

- $\hat{x}_{uij}(\Theta)$ is a real valued function in the parameters Θ which models the user's preferences between the two items i and j. In practice, the BPR delegates to an external function the job of modeling this probability. This function is totally generic since, as we already said, the BPR is just an optimization criterion which can be applied to any underlying algorithm that's in charge of computing this preference. This function, together with the two previous assumptions about the independence of the interactions, assures that the personalized ranking to be learned meet the totality, anti-symmetry and transitivity properties (i.e. it is a total ordering) [51]. Since the dataset is composed by triples, the pairwise prediction $\hat{x}_{uij}$ can be expressed as the difference of the two single predictions:
$$\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj} \tag{1.19}$$

The **criterion** can now be finally formulated as:

$$\text{BPR-OPT} := \sum_{(u,i,j)\in D_S} \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta ||\Theta||^2 \tag{1.20}$$

where:

- i is a *positive* item for user u, $i \in I_u^+$;
- j is *not a positive* item for user u, $j \in I \backslash I_u^+$;
- λ_Θ are *regularization parameters* that depend on the model (e.g. Matrix Factorization).

Algorithm 1 The LEARN-BPR algorithm.

```
**procedure** LEARN-BPR($D_S$,$\Theta$)
    initialize $\Theta$
    **repeat**
        draw $(u,i,j)$ from $D_S$
        $\Theta \leftarrow \Theta + \alpha\left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial\Theta}\hat{x}_{uij} + \lambda_\Theta \cdot \Theta\right)$
    **until** convergence
    **return** $\hat{\Theta}$
**end procedure**
```

The LEARN-BPR algorithm is illustrated in Algorithm 1. The BPR criterion is differentiable, so gradient descent based algorithms [11] are the logical choice for maximization.

1.8.1 BPRMF

We already presented MF in Section 1.7.2, stating that SVD is in general the best approximation with respect to least-square [51, 34, 52], but for the ranking task the BPR criterion is a far better choice. As we just saw, the BPR criterion delegates to an external model the job of defining and computing the scoring function $\hat{x}_{uij}$, BPRMF is the matrix factorization model optimizing the criterion in Equation 1.20.

The prediction formula can be rewritten using the notation in the previous section and the one in Section 1.7.2:

$$\hat{x}_{ui} = \sum_{k=1}^{f} p_{uk} \cdot q_{ik} \tag{1.21}$$

The model parameters Θ are the weights of the matrices P and Q that represent the user and item features in the latent space. In the *LEARN-BPR* procedure in Algorithm 1, the only thing required is knowing the derivative:

$$\frac{\partial}{\partial\theta}\hat{x}_{uij} = \begin{cases} \left(q_{ik} - q_{jk}\right) & \text{if } \theta = p_{uk} \\ p_{uk} & \text{if } \theta = q_{ik} \\ -p_{uk} & \text{if } \theta = q_{jk} \\ 0 & \text{else} \end{cases} \tag{1.22}$$

With this in mind and using regularization constants for the users and items features, we can adopt MF as the underlying model for learning the *BPR* criterion.

[11] Gradient descent is a very popular optimization technique used to find a local minimum of a differentiable function.

1.9 BPR Sampling

It should be clear at this point that at each step the LEARN-BPR algorithm (Algorithm 1) draws a **triple** composed by:

- a **user** u;
- a **positive** item for the user i;
- a **negative** item j.

This process is known as **sampling**. Sampling is a widely discussed topic that concerns several tasks, spanning from Computer Vision to Natural Language Processing models, and it hardly affects the algorithm performance and convergence speed.

The original LEARN-BPR introduced a stochastic gradient descent algorithm that randomly chooses the triples from a uniform distribution with a bootstrap sampling strategy with replacement [51]. In recent years, more sophisticated techniques have been proposed that try to understand the best way to extract the **negative samples** (the j item in the triple (u, i, j)) in order to both reduce the convergence time and the overall complexity.

1.9.1 Uniform Sampling Issues

The problem with the proposed uniform sampler, that led Rendle and Freudenthaler to publish a paper directly focused on sampling [50], is that since samples are extracted from a uniform distribution, most of the triples extracted are usually *not very informative* and this fact dramatically slows the convergence and limits the recommendation quality. As we briefly discussed in Section 1.6.1, the item popularity in RS typically follows a tailed non uniform distribution, in which few items are very popular and the majority of the dataset have fewer interactions. Randomly sampling triplets from the dataset means that the sampler is showing to the algorithm triples that are too easy to rank, thus causing difficulties in the learning process. We are going to explore this with an example.

1.9.1.1 General Intuition

Let's consider a scenario of a music recommendation system like Spotify, with $|U|$ being the set of users of the platform and $|I|$ the set of songs. Sampling triplets for learning the BPR criterion in this setting means extracting:

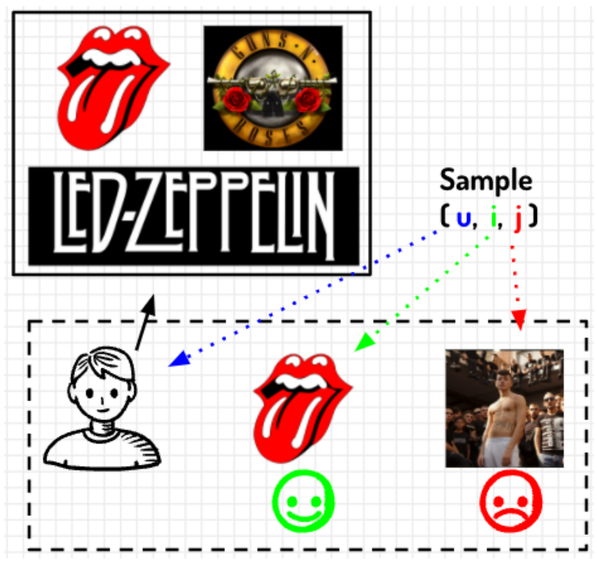


Figure 1.9.1: This image shows the uniform sampling process that randomly extracts a positive interaction and a negative one, resulting in a too easy ranking thus in a poor learning process.

- u, a user;
- i, a song that user u liked in the past;
- j, a song that u did not interact with either because the user doesn't like it or because the user haven't had the chance to listen to it yet.

Since we can expect that the song popularity distribution follows the long tail model explained in Section 1.6.1, most of the triples extracted by a uniform sampler will present to the algorithm the preference of user u for the probably popular song i with respect to the probably niche song j; moreover, this kind of sampling strategy does not take into account the taste of user u. Let's imagine that user u is into 70s/80s Rock music; a random sampler that extracts "I can't get no satisfaction" by The Rollings Stones as positive example and "Rozzi" by Paky[12] as negative example is not teaching much to the algorithm, as the task of predicting that u prefers the Stone's song is way too easy and even a top-popular baseline can succeed in predicting this user preference (Figure 1.9.1). In order to make the algorithm *really* learn the user's preference, the sampler needs to extract trickier and more informative negatives: "I can't get no satisfaction" by The Rollings Stones or "Born in the U.S.A." by Bruce Springsteen?, "Welcome to the Jungle" by Guns N' Roses or "Free Bird" by Lynyrd Skynyrd? Asking these questions to the algorithm

[12]Paky is an Italian trap/rap artist born in 1999.

makes it understand u's taste on a deeper level.

1.9.1.2 Gradient Magnitude

From a mathematical standpoint, the previous example can be explained by the fact that after a few training epochs the model is not able to learn anything more due to the too **small gradient magnitude**. In machine learning and data mining, especially in the context of artificial neural networks [25, 5], *vanishing gradient* is a widely discussed topic that affects those techniques that use gradient descent based methods and *backpropagation*[13] and verifies when the gradient is so small that it practically prevents the models weights from changing their values. In the context of BPR, the logical choice for optimizing the criterion is stochastic gradient descent since the criterion is differentiable and batch updates would be too computationally expensive; as we noted in the previous example though, uniformly sampled negative items are very likely to be ranked correctly below a (random) positive item and thus the gradient of the pair is near 0 [50]. From Equation 1.20, we can compute the gradient with respect to the model parameters as follows:

$$\begin{aligned}\frac{\partial \text{BPR}-\text{OPT}}{\partial \Theta} &= \sum_{(u,i,j)\in D_S} \frac{\partial}{\partial \Theta} \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta \frac{\partial}{\partial \Theta} \|\Theta\|^2 \\ &\propto \sum_{(u,i,j)\in D_S} \frac{-e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} - \lambda_\Theta \Theta\end{aligned} \tag{1.23}$$

From this we can obtain the stochastic gradient descend single update for the model parameters, which is the one in Algorithm 1:

$$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta \right) \tag{1.24}$$

From this formula, we can observe that, at each step, the gradient $\frac{\partial}{\partial \Theta} \hat{x}_{uij}$ has a multiplicative factor that can be rewritten using Equation 1.19:

$$\begin{aligned}\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} = 1 - \frac{1}{1+e^{-\hat{x}_{uij}}} = 1 - \sigma\left(\hat{x}_{uij}\right) &= 1 - \sigma\left(\hat{x}_{ui} - \hat{x}_{uj}\right) \\ &= 1 - p\left(i >_u j\right) = \Delta_{uij}\end{aligned} \tag{1.25}$$

[13]Popular technique used for training feed-forward neural networks.

This quantity depends on how the model is able to discern between the positive item i and the negative one j for the user u and in particular represents the **probability of incorrectly ranking** the pair of items, so it'll be close to zero if the model correctly assigns a larger score to item i and close to 1 if, on the contrary, j is incorrectly assigned a larger score than i. In other words, Δ_{uij} represents the gradient magnitude and can be viewed as a measure of the influence that a specific sample (u,i,j) has in updating the model parameters Θ. In particular, if Δ_{uij} is close to zero we have the vanishing gradient condition that we presented, meaning that the model parameter θ is not affected by the updating of the sample (u,i,j). This suggests that a great number of pairwise samples extracted with a uniform sampler are meaningless for updating the model, and only a small number of them are valuable.

What should be clear from this analysis is that sampling *meaningful*, *difficult* and *informative* **negative items** is vital for the learning process of the model.

1.9.2 Negative Sampling Classes

After the original BPR paper publication, a few sampling techniques have been proposed with the purpose of identifying meaningful negative candidates that can increase the gradient magnitude in order to improve the training phase. Among these methods we can identify some common properties.

1.9.2.1 Static vs Dynamic

The first distinction is determined by how the sampling probability distribution changes over the training epochs.

- **Static** sampling techniques adopt a strategy that extracts triples (u,i,j) from a distribution which is not affected by the training epoch at which the triple is sampled.
- In **dynamic** samplers, on the other hand, the sampling probability distribution is not fixed as it changes during the training. Usually the distribution is affected by the updates of the model parameters, which means that the probability of a negative item to be extracted in a specific instant of training is proportional to its model score at that time instant (in this case, the sampler is also called **adaptive**).

The uniform sampler that we described is static, since it randomly chooses negative instances from a fixed uniform distribution.

Let's compare the two approaches in two key aspects:

1. **Complexity Analysis**: static samplers are very convenient because sampling from a fixed parametric distribution is simple. We can sample a rank r from the fixed distribution (e.g. a Geometric distribution) in $\mathcal{O}(1)$ and then retrieve the item in that position again in constant time. The complexity obviously changes in the case of a dynamic sampler, as the relative ranks of the items change at each epoch due to the model updates, thus the scores need to be computed at each epoch.

2. **Sample Quality**: sampling from a fixed distribution does not take into account what the model is able to learn during the training (i.e. the *current belief*) and, moreover, a dynamic sampling strategy that uses the model weights makes the algorithm gradually learn the users preference by first sampling simple items and then, as the training continues, selecting more difficult samples at each epoch. The reason behind this is that the weights are usually initialized at random with some seed, hence the first round of sampling will be quite close to a standard uniform sampling; at each epoch the model weights are updated and the distribution of the scores becomes more defined so the sampling probability of the items will change accordingly, resulting in process that is somehow similar to fine-tuning the algorithm on the most difficult items.

1.9.2.2 Global vs Context-dependent

In **global** sampling techniques, the probability for an item to be sampled is independent from the current user and positive item in the triple. For example, the uniform sampler is global, in fact when choosing the negative item j for the positive interaction (u,i) it extracts a negative interaction completely at random regardless of u and i.

On the contrary, **context-dependent** strategies employ the context in the sampling process, that means that the probability for an item to be extracted **changes for each context**, hence we have a different distribution for each different user (or even for each different pair (u,i)).

1.9.2.3 Single-feedback vs Multi-feedback

As we discussed in Section 1.2, different types of information can be collected in a recommender system, from the simple non-scored interactions to the user or item features to the external context.

The sampling techniques that, like the uniform sampler, only take into account the historic interactions between users and items are called from now on **single feedback** methods; these methods are the most general ones, as they can always be applied to every RS dataset, since they only need the user rating matrix.

With the term **multi-feedback** sampling techniques instead we refer to all those techniques that cannot be applied to every dataset and that usually employ information that is highly domain specific. These methods can include content information like the item features rather than different types of feedback (e.g. clicks vs purchases) rather than external information like images or videos. As these techniques are domain specific and exploit additional information, they usually greatly improve the model's performance compared to the standard single-feedback sampling methods that only use the rating matrix, at the cost of being not always applicable.

1.9.2.4 Heuristics vs Model Based

So far we only talked about some generic probability distribution for the negative items, distinguishing the techniques on the basis of the properties of the distribution, but we haven't yet explored how these distributions are generated.

Heuristic samplers are a class of samplers that define the probability sampling distribution using heuristics: randomly (i.e. the uniform sampler), utilizing the popularity, the scores etc. In contrast to the heuristic samplers, popular sampling methods delegate to an *external model* the job of learning the optimal sampling distribution for training the original model; we will refer to these techniques as **model based** samplers.

The growth of model based sampling techniques was heavily boosted by the rise of Generative Adversarial Networks (GANs) [8, 59, 14], which is a class of Machine Learning techniques proposed by Bengio and Goodfellow in which two models (specifically, two neural networks) contest each other in a *minmax* game. A generative model G tries to capture the data distribution and trick a discriminative model that learns to determine whether a sample comes from the model distribution or the data distribution (Figure 1.9.2). This concept constitutes one of the most successful popular trends in the machine learning

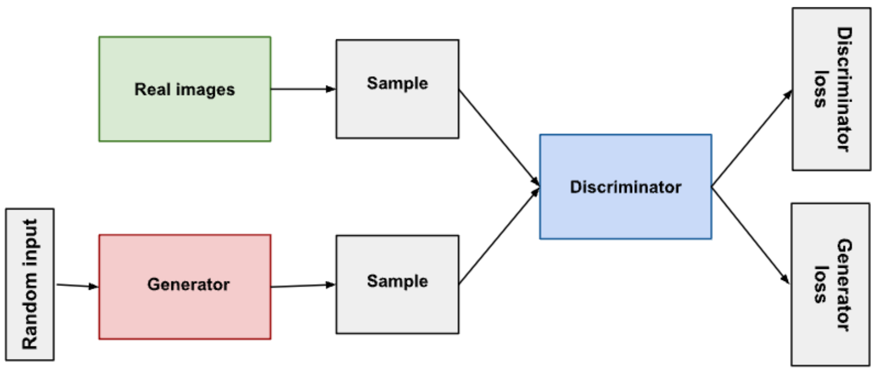


Figure 1.9.2: The architecture of GANs applied to Computer Vision. The generator output is connected directly to the discriminator input. Through the mechanism of backpropagation the discriminator's classification provides a signal that the generator uses to update its weights.

Figure 1.9.3: This images shows an application of GANs to computer vision. The faces are examples of photo-realistic GAN-generated faces, taken from a recent paper from 2017. [32]

field in recent years[14] and it is the same one adopted by some of the model based sampling techniques that have been proposed in the context of BPR, like the very popular framework IRGAN [59, 7, 11]. In these approaches, the generative model is exactly the sampler, that incrementally learns to fit users' preference distribution and thus adaptively generate positive-like informative instances [7].

Although these techniques are very attractive, they pose a lot of challenges and won't be explored in depth in this thesis. It should be easy to understand that, for the most part, these samplers are incredibly difficult to implement

[14]GANs have been employed also in various domains outside the academic area, they especially gained interest and success in arts and music.

and tune, they need a huge quantity of training samples and they are extremely slow and computationally expensive since training a GAN means training two different models and tune two different sets of hyperparameters [56]. Moreover, especially in the case of adversarial samplers, model based techniques tend to yield a biased estimate of the gradients since its uneven sampling distribution will skew the instance contribution, resulting in sub-optimal results [7] so that the model will be biased to over-fit the difficult positive-like negative instances that are likely to be adopted by the user in the future. Lastly, it should be noted that even though deep learning approaches are very appealing and are claimed to be the state of the art in nearly every ML domain, the enthusiasm that surrounds these techniques is not always supported nor confirmed by experimental results; this is particularly true for the top-n recommendation tasks, as precisely discussed in a recent paper proposed by Dacrema et al. [9], in which properly tuned collaborative baselines like the ones discussed in Section 1.3.1 still perform better than deep learning based methods.

1.10 BPR Sampling Algorithms

In this section we introduce the more significant and discussed negative sampling techniques for the Bayesian Personalized Ranking. Here we only deal with those techniques that are *strictly* proposed to work with the classical formulation of the *BPR* in Equation 1.20 and that only consider the user rating matrix (i.e.*single-feedback* and *heuristics*) due to their generality.

1.10.1 Uniform Sampling

This is the original sampling algorithm proposed in the BPR paper, which we have already discussed extensively in section 1.9. According to the previous definitions, this sampling technique is *static* and *global*, as the sampling distribution does not change over time and is not affected by the specific positive interaction.

Rendle et al. also proposed to use a uniform distribution with a bootstrap sampling approach with replacement so that stopping can be performed at any step [51], since the number of the interactions is usually so big that a full cycle over all the interactions at each epoch is not feasible, so a bootstrap approach is the one proposed. We want to point out now that this bootstrap approach is affecting the *positive* sampling strategy, which we will not cover in depth since we are interested in understanding the possibilities and limitations of the current negative sampling strategies.

1.10.2 Popularity Oversampling

Considering that item distribution of the items in a RS typically follows the *long tail model* (Figure 1.6.1), using a uniform strategy is not very efficient since most of the pairs will be uninformative, as explained Section 1.9.1. The **popularity oversampling** strategy [50] tries to address the issue by using a non uniform distribution that favors the sampling of the popular negative items. In order to oversample popular items, one could use the *empirical distribution* of the item popularity or a *parametric distribution*.

Whether one chooses the empirical distribution or its parametric counterpart, the idea behind it is exactly the same, albeit the parametric one is usually preferred due to its readability. This sampling technique is **static** since the sampling distribution is defined once for all before the sampling process begins and never changes and does not take into account how the estimated rank of an item changes during learning. It is also **global** as the sampling does not depend on the context but only on the overall popularity of the items.

1.10.2.1 Empirical Distribution

In the empirical case, the sampling distribution is defined by how many times the item appeared as a positive interaction in the dataset:

$$p(j \mid u) \propto \left|\{(u', j') \in S : j = j'\}\right| \tag{1.26}$$

Designing a sampler for drawing a negative candidate for the triple $(u,i,?)$ is easy, as the sampler can simply extract an observation (u',j) uniformly and discard u, assuming that $(u,j) \notin S$ (i.e. u did not interacted with j). The procedure is showed in Algorithm 2; the input (u,i) is the positive interaction uniformly selected from the observations S.

Algorithm 2 Empirical Popularity Oversampling strategy.

```
procedure EmpPopOversampling($(u,i)$)
    draw $(u',j)$ uniformly from $S$, with $j \in I \setminus I_u^+$:
    return negative item $j$ for sample $(u,i,j)$
end procedure
```

1.10.2.2 Parametric Distribution

If one decides to use a parametric distribution, the usual choices are analytical laws that are very close to the long tail model, as the Geometric or Zipf[15] distributions [50]. Let's take as example the Geometric distribution with the following formulation:

$$p(j \mid u) = \gamma(1-\gamma)^{r(j)}, \quad \gamma \in (0,1) \tag{1.27}$$

this can also be rewritten as:

$$p(j \mid u) \propto \exp(-r(j)/\lambda), \quad \lambda \in \mathbb{R}^+ \tag{1.28}$$

[15]Zipf's law, proposed by the American linguist George Kingsley Zipf, refers to the fact that for data types the rank-frequency distribution is an inverse relation.

where $r(j)$ is the rank of item j according to the global popularity and λ is a parameter that identifies the expected rank in the distribution. In this case, the sampler first extracts a rank r from the Geometric distribution and then returns the item j on position r according to the item popularity (Algorithm 3).

Algorithm 3 Parametric Popularity Oversampling strategy.

```
procedure $ParamPopOversampling(u, i)$
    sample $r$ from the Geometric distribution;
    get $j \in I \backslash I_u^+$ in position $r$ in the popularity ranking;
    return negative item $j$ for sample $(u, i, j)$
end procedure
```

1.10.3 Personalized Popularity Oversampling

This technique [50] stems from the same idea as the previous sampling method. Such method is still **static** but it is different in the fact that this time the sampling distribution is **context-specific** because in this case, instead of defining a global popularity distribution, the sampling distribution is personalized for each user:

$$p(j \mid u) \propto \left|\{(u', j') \in S : u = u', j = j'\}\right| \tag{1.29}$$

Such distribution, even if logically better than the previous one as it takes the user into account, is not really applicable in real situations as it is defined on very few samples due to the sparsity of the user rating matrix.

1.10.4 Imbalance Rejection Sampling

This sampler is a slight variation of the popularity oversampling that acts as a ”**rejection sampler**”.

Rejection sampling (also commonly called ”acceptance-rejection method”) is a general method applied in statistics and numerical analysis whose aim is sampling points independently from a probability density function $f(x)$ defining an *envelope* function $g(x)$ that acts like a proxy distribution from which we can sample[13].

Algorithm 4 The Rejection sampler that uses the Imbalance Value.

```
procedure $ImbRejectionSampling(u, i, s, \pi)$
    initialize $selected\_j = -1$, $max\_pop = -1$
    for $iter \leftarrow 1$ to $s$ do
        draw $j$ uniformly from $I \setminus I_u^+$
        if $\pi(j) > max\_pop$ then
            $max\_pop = \pi(j)$
            $selected\_j = j$
        end if
        $reject\_ratio = 1 - \min\left\{\frac{\pi(j)}{\pi(i)}, 1\right\}$
        if $random.uniform() > reject\_ratio$ then
            $selected\_j = j$
            break
        end if
    end for
    return negative item $j$ for sample $(u, i, j)$
end procedure
```

In [65], the authors analyze the problem from the graph theory perspective shown in section 1.6.4.1. They define the *"vertex-level imbalance"* as the issue that happens when the number of times that a vertex (i.e an item i) appears in observed edges (i.e. an arc from user u to item i) is extremely smaller or larger than that in the unobserved ones (i.e. the class imbalance problem already mentioned). In practice, the *imbalance value* is the ratio of an item positive occurrence over the negative one. This quantity is obviously intrinsically related to the the vertex degree, which is the item's popularity. The key insight of this sampler is that it aims at favoring the sampling of negative items with a larger degree than the positive item i, so that the probability distribution changes according to the positive item in the sample. The reason behind this is that ranking a very popular item (the negative one) versus a less popular one (the positive one) should be a difficult task, thus the gradient should be close to 1. The pseudocode of the procedure is in Algorithm 4.

As you can see, the algorithm takes two inputs: an integer s and a function π. The meaning of s is the number of negatives to extract (the "max shots"") to use, it's an hyperparameter to tune; a higher s means an increased probability of finding a *difficult* negative item at the cost of increasing the training time. The function π is defined as the global popularity of the items. From the pseudocode, we can point out that the sampling returns in two cases:

1. if it finds a negative item j that has a greater popularity with respect to the positive item i sampled;
2. if it reaches the max shots, then it returns the item with max popularity.

Notice that the first condition makes this sampler an actual rejection sampler. With respect to the notation in the previous sections, we can define this sampling technique as **static** and **context-dependent**.

1.10.5 VINS

Algorithm 5 VINS.

```
procedure $VINS(u, i, max\_iterations, s, \pi, \epsilon)$
    initialize $selected\_j = -1$, $max\_score = -1inf$
    for $iter \leftarrow 1$ to $max\_iterations$ do
        $j = ImbRejectionSampling(u, i, s, \pi)$
        $x_{uji} = x_{uj} - x_{ui} + \epsilon$
        if $x_{uj} > max\_score$ then
            $max\_score = x_{uj}$
            $selected\_j = j$
        end if
        if $x_{uji} > 0$ then
            break
        end if
    end for
    $r_i = \left\lfloor \frac{Z}{\min(max\_iterations, iter)} \right\rfloor$
    return negative item $j$ for sample $(u, i, j)$, $w_{ui}(r_i)$
end procedure
```

VINS stands for Vital Negative Sampler, it's a sampler proposed on the same paper [65] that includes the strategy that we just presented. Actually, the rejection sampler in the previous section is exactly the first phase of the VINS sampler, while the second phase extends the rejection sampler by considering the dynamic relative rank position of positive and negative items for finding more informative negative samples. We can see this sampler as an hybrid sampler that uses *both* a **static** and a **dynamic** adaptive one.

Looking at Algorithm 5, we want to point out two main things. First of all, similarly to the imbalance sampler in Section 1.10.4, VINS terminates whether one of the two following condition verifies:

1. the *max_iterations* parameter has been reached. In this case, the sampled negative item is the candidate with largest score;
2. the negative item has a greater score than the positive one ($x_{uji} > 0$). The idea is that if we are able to find such a negative item, then the sample is for certain a difficult one to rank. In [65], this negative instance is called "violated negative sample".

The second key aspect that we want to highlight is that, as we can see in the algorithm, this sampler differentiates from the others that we presented as it not only returns the triple (u,i,j) but also a number w_{ui}. This value is intended as a measure of the difficulty of finding the negative candidate j as a **sample weight**. The exact formula to compute w_{ui} is the following:

$$w_{ui}(r_i) = \frac{1 + 0.5 \cdot (\lceil \log_2(r_i + 1) \rceil - 1)}{1 + 0.5 \cdot (\lceil \log_2(Z + 1) \rceil - 1)} \tag{1.30}$$

where $r_i = \left\lfloor \frac{Z}{\min(max\_iterations, iter)} \right\rfloor$ and $Z = \sum_{i \in I} \pi(i)$. With reference to the BPR criterion in Equation 1.20, the sample weight acts in this way in the criterion:

$$\sum_{(u,i,j) \in D_S} w_{ui} \ln \sigma \left(\hat{x}_{uij} \right) - \lambda_\Theta \|\Theta\|^2 \tag{1.31}$$

modifying the weight of each sample update in the loss. In particular, the negative candidate is said to be difficult if it needs a lot of iterations in order to find the one that breaks the second condition ($x_{uji} > 0$) or if it reaches the *max_iterations* parameter; in this case, w_{ui} is smaller, thus reducing the weight of the sample due to the arduousness of finding a difficult one.

VINS is an hybrid sampler, it is both **static** (in its first phase) and **dynamic** (second phase), it is **context-dependent** and **adaptive** and applies a *dynamic* sample weighting since the weight is not statically defined but needs to be computed at each iteration. From a complexity point of view, it is clear that VINS is very much slower and computationally expensive, and this also highly depends on the s and *max_iterations* parameters. Obviously larger values should give better results, but they also have a more significant impact on the performance.

1.10.6 Ranking-aware Rejection Sampling

Wang et al. proposed this sampling technique [66] in order to speed up the training of *BPR* and *LambdaRank*[16] without having huge impact on the performance. This paper was published one year before the Rendle paper focused on sampling and it is based on the same ideas of finding a dynamic adaptive sampler to improve the static ones. While Rendle proposed an integrated framework to reduce the complexity, Wang et al. dealt with the performance issue by suggesting an approach that samples only a small group of candidates in an uniform manner and then extracts the negative candidate according to a rejection strategy that uses a *sampling weighting function* $g(x_i)$, which can be a linear or polynomial function, that depends on the relative ranking x_i of item i according to the current model scores. This sampling is **dynamic**, **adaptive** and **context-dependent** and, even if dynamic sampling is slower due to the computation of the item scores, offers great advantages in terms of performance since it only considers an arbitrary small portion of the negative items.

1.10.6.1 Linear Weigthing Function

This sampling algorithm takes as input the current positive interaction (u, i), a parameter β that identifies the rejection probability and a scoring function s, which in our case is the model itself.

Algorithm 6 Sampling strategy that uses a linear weighting.

```
procedure $RankingAwareRejectionLinear(u, i, \beta, s)$
    draw $j$ and $l$ uniformly from $I \backslash I_u^+$
    compute the scores $s(j)$ and $s(l)$
    if $s(j) > s(l)$ then
        return $j$ with probability $\frac{1}{1+\beta}$, $l$ otherwise
    end if
    return $l$ with probability $\frac{1}{1+\beta}$, $i$ otherwise
end procedure
```

Therefore, the probability for an item j to be extracted is the following:

[16] Class of algorithms proposed by Microsoft researchers in [6].

$$
\begin{aligned}
p_j &= \frac{1}{1+\beta}\Pr(s(j) > s(l)) + \frac{\beta}{1+\beta}\Pr(s(j) \leq s(l)) \\
&\propto \left(1 - x_j\right) + \beta x_j = -(1-\beta)x_j + 1 \equiv g\left(x_j\right)
\end{aligned}
\tag{1.32}
$$

In this case the time complexity is significantly lower with respect to computing all the item scores, as it is $O(1)$ compared to $O(I \backslash |I_u^+|)$.

1.10.6.2 General Polynomial Weighting Function

This version of the sampling algorithm is exactly the same as the previous linear one but extended to a larger number of uniformly sampled negative candidates. The probability distribution in this case follows a multinomial distribution:

$$
p_j \propto 1 C_n^0 \left(1 - x_j\right)^n + \sum_{k=1}^{n} \beta_k C_n^k x_j^k \left(1 - x_j\right)^{n-k} \tag{1.33}
$$

In particular, setting all the parameters β_k to zero, one obtains a sampler that extracts the negative item with the largest score among the $n+1$ selected; this can be done in $O(n+1)$ time complexity.

1.11 BPR Variants

The BPR criterion in Equation 1.20 has been revised in various manners for different reasons. The original BPR starts from a generic implicit feedback scenario, in which we can only monitor if a user interacted with an item or not. As already discussed in section 1.2, in real-world scenarios we are typically able to track multi-feedback information like clicks, view times, purchases etc. and external contextual information like item properties, social connections etc. Some of the following algorithms take into account this auxiliary domain specific information in order to better formulate the criterion for the task at hand, while other try to improve certain aspects of the algorithm.

In this section we want to briefly outline some of the more interesting derivations, without going into too much detail, just to give some intuitions that can help understanding the aim of this work.

1.11.1 AOBPR

This [50] was the article written by Rendle and Freudenthaler in order to improve the sampling technique of the BPR. The key aspect of this research is what we discussed in Section 1.9.1.

Rendle and Freudenthaler propose to use the model scoring function to define the sampling distribution. This approach would make the sampling **dynamic**, **adaptive** and **context-dependent**. The idea is similar to the ranking aware rejection sampling and to VINS, in the sense that ideally we want to choose negative items with large scores because they increase the gradient magnitude Δ_{uij}. Instead of formulating the problem using the scores, it is better to use the concept of ranks as they represent an absolute value. The sampling probability for a negative item j to be extracted is:

$$p(j \mid u) \propto \exp(-\hat{r}(j \mid u)/\lambda), \quad \lambda \in \mathbb{R}^{+} \tag{1.34}$$

with $\hat{r}(j \mid u)$ being the expected rank and λ the parameter of the exponential distribution. The great difference between this algorithm and the ones proposed in Sections 1.10.5 and 1.10.6 is that in this case the distribution probability is defined over *all* the negative instances. Consequently, in each iteration, in order to extract the negative items all the scores need to be computed from scratch and this highly increases the training time, so much that it becomes unfeasible. In fact, if we call T_{pred} the time for computing a score, a dynamic sampler that extracts a rank r and than needs to compute all the scores increase the complexity by a factor $\mathcal{O}\Big(|I| \cdot T_{\text{pred}} + |I|\log(|I|)\Big)$.

To overcome this issue, Rendle et al. designed an efficient framework, commonly called AOBPR (Adaptive Oversampling BPR), that is able to approximate the sampler in Equation 1.34 in amortized time using some mathematical transformations in the context of matrix factorization models.

1.11.2 GBPR

Group Preference Based Bayesian Personalized Ranking (GBPR) is a BPR variant proposed by Pan and Chen in [45]. This work is based on the fact that the two assumptions of the BPR discussed in Section 1.8, namely (i) the *individual pairwise preference* over two items and (2) *independence between two users* may not always hold. As a response to the possible violations of these two fundamental assumptions, the authors propose a new assumption and introduce the notion of *group preference* as the "overall preference score of a group of users on an item" [45], in contrast with the *individual preference* that is the preference score of a user on an item. The assumption is that the group preference $\mathcal{G} \subseteq \mathcal{U}_i^{tr}$ on an item i is likely to be stronger than the individual preference of a user over that item. This idea is also supported by the fact the users tend to be influenced by other users. The GBPR criterion is the following:

$$\text{GBPR}(u) = \prod_{i \in \mathcal{I}_u^{tr}} \prod_{j \in \mathcal{I}^{tr} \setminus \mathcal{I}_u^{tr}} \Pr\Big(\hat{r}_{\mathcal{G}ui} > \hat{r}_{uj}\Big)\Big[1 - \Pr\Big(\hat{r}_{uj} > \hat{r}_{\mathcal{G}ui}\Big)\Big] \tag{1.35}$$

1.11.3 PRIS

This [37] recent article by Chen et al. introduced a new ranking loss based on "importance sampling" so that more informative negative samples can be utilised better. Among the various things discussed in the paper, the key aspect we should focus on is that it points out one important problem in BPR: it ignores the **negative confidence**, so that all negative items are treated equally. In their PRIS (Personalized Ranking with Importance Sampling) framework, the probability of an item being negative is modeled using the model scores. Recalling that $\sigma(\hat{x}_{uij})$ represents the probability for user u to prefer item i over j, a higher $\sigma(\hat{x}_{uij})$ means a lower probability for j to be negative. Thus, the probability of an item j being negative can be expressed as:

$$P(j \mid u, i) = \frac{\exp\left(-\hat{x}_{uij}\right)}{\sum_{j' \in I \setminus I_u} \exp\left(-\hat{x}_{uij'}\right)} \tag{1.36}$$

This value is than used as sample weight in the loss as follows:

$$\sum_{(u,i,j) \in D_S} P(j \mid u, i) \times \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta \|\Theta\|^2 \tag{1.37}$$

From the equation above, we can see that PRIS pays more attention those unobserved items that present higher negative probability. We can immediately see that the probability in Equation 1.36 is extremely time consuming to compute due to the very large number of unobserved items (we already said that the user rating matrix is often very sparse), that's why the authors changed the BPR loss to make it more efficient.

1.11.4 Graded Implicit Feedback

Lerche and Jannach adapted the BPR to scenarios in which we can exploit additional information about the interactions between users and items, such as the date or the number of times that a certain interaction happened [36]. They point out that the original dataset definition in Equation 1.14 is only able to encode single, positive-only statements like non-repeated item purchases or "like" statements on social platforms. For this reason, they propose a different dataset generation strategy:

$$D_S^{++} := \{(u,i,j) \mid \text{ pweight } (u,i) > \text{ pweight } (u,j), i \in I, j \in I\} \tag{1.38}$$

that takes into account the *weight* of a sample. The weight of a sample can be the number of times that an interaction occured or a number associated to how recent an interaction is. Each weighting strategy identifies a different model. This idea is also related to [12] by Gantner et al. for the KDD Cup[17] 2011, which proposes a weighted BPR (WBPR) that takes into account the item popularity in order to balance the contribute of each single triple.

[17]The KDD (Data Mining and Knowledge Discovery) Cup is a competition organized every year by ACM Special Interest Group on Knowledge Discovery and Data Mining since 1997. More information can be found at https://kdd.org/kdd-cup.

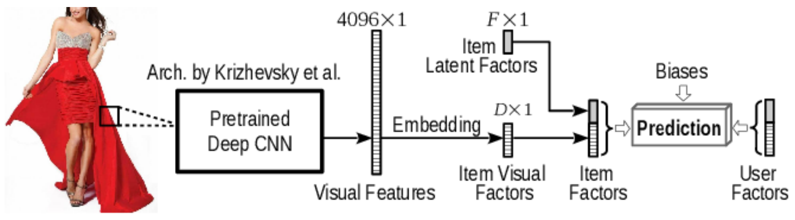


Figure 1.11.1: The image represents the architecture of the VBPR framework.

1.11.5 MF-BPR

Pagano et al. took inspiration from the previous approach by extending the BPR sampling method, that equates the feedback sources, with different "levels". Their approach exploits **multiple feedback** sources simultaneously during the training process [40]. Additional feedback types are: clicks, views, likes etc. Each one of this types of feedback strengthens the user profile in a different manner, and the sampling method can exploit these differences.

In Section 1.8 we described the procedure for creating the training data D_S for the BPR, which basically distinguishes a positive, a negative and a missing layer. In the MF-BPR (MF stands for Multi Feedback) approach the various types of feedback are mapped onto various *levels* that reflect the expected contribution of each type. Let $\mathbb{L} = \left(L_1, \ldots, L_p\right)$ represent a given ordered set of levels such that a feedback in L_i is a stronger indicator of interest compared to a feedback in L_{i+1}, that is $L_i > L_{i+1}$. Then the training data for this methods can be characterized as follows:

$$D_{MF} = \{(u,i,j) \mid i \in I_{L,u} \wedge j \in I_{N,u} \wedge L \in \mathbb{L}^{+} \wedge L > N\} \qquad (1.39)$$

From this dataset, the sampling process is designed according to the different levels of feedback, so the sampling probability distribution will reflect the different types of interactions.

1.11.6 VBPR

This [21] work by He et al. was widely cited and discussed in recent years. Their paper points out that, typically, one feature which is ignored by existing personalized recommendation techniques is the **visual appearance** of the items being considered. VBPR (Visual BPR) tries to incorporate the visual features of the items to model the user's preference.

The scenario is still an implicit one, but in addition to the user-item interactions, a *single image* for each item is also available. As you can observe in Figure 1.11.1, there are two types of parameters that need to be updated at each iteration:

- the BPR model parameters, which are updated according to the original paper;
- the parameters of the item visual latent features extracted from a pre-trained CNN.

This method highly improves the model performance especially in *cold start* scenarios and for settings such as e-commerce platforms, in which the visual appearance of an item is one of the key factors that contributes to the purchase of said item.

1.11.7 Social Information

Various algorithms have been proposed to take advantage of social interactions among users. These works, as [68, 35, 67], incorporate the social feedback to define different sampling strategies often improving the performance of the BPR in particular in *cold case* scenarios, as the previous algorithm. The idea of these algorithms is to leverage the social relationships in a manner which is similar to GBPR approach proposed by Chen and Pan in Section 1.11.2.

1.12 Introducing Our Work

In this chapter, we described what Recommender Systems are about, which are the reasons that drive the interest in this field and which are the major challenges to be faced. We presented the different types of datasets and data structures and introduced the main state of the art models. We focused on MF techniques and especially on the BPR, explaining its novel approach and advantages, and then moved our attention to the existing sampling algorithms adopted in the BPR.

In this scenario, this thesis focuses on implicit feedback situations in which no other information is available and tries to investigate the negative sampling process of the BPR criterion applied to the Matrix Factorization algorithm (Section 1.8.1). In particular, lots of studies [44, 28, 12, 60] point out the idea of weighting each sample according to some metric and, amongst them, a couple of recent works [65, 37] investigate this aspect in the context of the BPR. Our thesis analyzes these techniques and proposes a new sampling strategy based on the concept of negative confidence, which is the measure of how much we can be sure of the negativeness of an item for a certain user. Defining the confidence value is not a trivial task and for this purpose we use the collaborative methods in Section 1.3.1, which to our best knowledge have not been used by other approaches directly in the BPR sampling strategies. This confidence sampling strategy can bring great benefits in term of performance without burdening on the model complexity.

Chapter 2

Models

In the previous chapter we focused our attention on the BPR model and, in particular, on the several sampling techniques that have been proposed to speed up the model convergence and improve the recommender ability to produce a personalized ranking. While the first works [66, 50, 51] on this topic focus on the sampler, the more recent ones [65, 37] try to weight each sample in order to increase or decrease the importance that each triple has in the model parameters updates.
In this chapter, we present our strategy for improving the current techniques.

2.1 Sampling Issues

Firstly let's have a brief recap of the main challenges and important aspects we should take into account when dealing with sampling.

- **Draw informative pairs**: this is the key aspect that started the discussion about sampling in the first place. As discussed in the previous chapter, selecting difficult negative instances is vital for the learning process of the model.
- **Complexity**: for a sampling technique to be applicable and usable, it's important for the complexity to be relatively low in terms of time and

hardware consumption.

- **Tuning**: correspondingly, a sampling technique that aims at being general and widely used should not ideally have too many hyperparameters, which are often difficult to tune and also increase the training time.

Ultimately, we want to design a negative sampling strategy that is **general** and widely applicable, **not very computationally expensive** and that is able to select **difficult** negative instances. Since we don't want our strategy to only be applicable in certain domains, we only focus on strictly implicit feedback scenarios in which only one type of feedback is observable (e.g. a purchase action, but not a view action) and we don't have any auxiliary information about users, items or any temporal or social knowledge.

In the following paragraphs, we discuss the two main ideas that motivated our work.

2.1.1 The risks of oversampling difficult instances

We remarked in multiple occasions that the **difficulty** of a sample is directly related to the **gradient magnitude** and thus to the ability of the recommender to output good suggestions (see Section 1.9.1.2).
We agree with this reasoning as it brings undoubted benefits, but we want to argue that this approach can lead to tricky situations in which too difficult negative candidates can prevent the recommender to take into consideration certain items that a user may like in the future.

Going back to the music recommendation example in Section 1.9.1.1 once again, let's take u to be a user who's into 70s/80s Rock'n'Roll, and in particular Rolling Stones, Guns N Roses, but that does not like AC/DC or The Eagles much.
We are going to try to give a real life context to the state of the art sampling techniques:

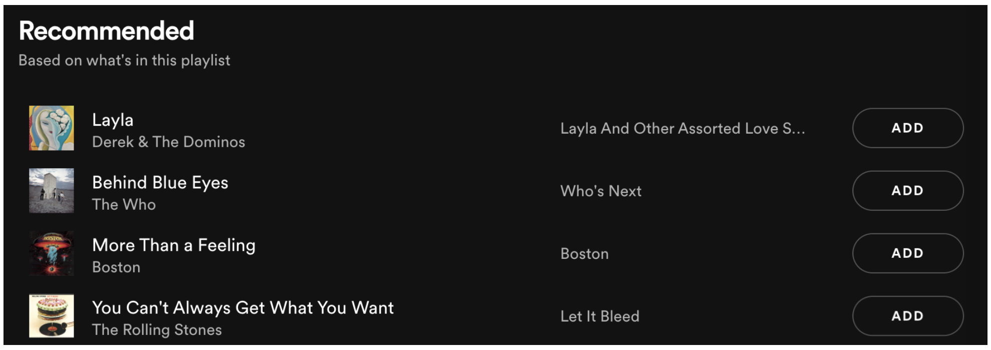
- **uniform**: the uniform sampler randomly chooses a song from the incredibly vast list of songs available on our Spotify-like music streaming service. We already pointed out why this is not optimal in 1.9.1.
- **popularity based**: popularity oversamplers favor the selection of negative items with a high popularity. This means that negative samples that will be extracted multiple times by this technique would probably

be: “Shape of You” by Ed Sheeran, “Blinding Lights” by The Weeknd, “Bad Guy” by Billie Eilish or “One Dance” by Drake[1]. Oversampling this kind of songs can speed up the training in the initial phases, as the model should be able to learn that u is not really into 2010s Pop music, but in the long run this kind of strategy doesn’t increase the difficulty of the samples and, on the contrary, it even prevents the sampler to choose niche items that could be meaningful for the user such as less popular Rock songs of the 70s/80s that are not among the current top streamed songs. This matter is also proven by experimental results [50, 65, 22], that show that the popularity oversamplers performance are worse even with respect to the random sampler, probably because, as we just mentioned, popular items may be overtrained locally at the expense of less popular items which would then be undertrained [22]. Another thing to consider is that, since the item popularity distribution is typically long tailed, the most popular items are likely to become future positive interactions.

- **dynamic adaptive samplers**: in the case of a dynamic sampler, that also takes into account the current model belief [50, 65, 66], it’s trickier to formulate a real life example because the sampling distribution is not dependent on some tangible property of users or items but it’s defined by the latent user and item features identified by the matrix factorization model. Still, if the underlying model is good enough at some point, it should be able to output decent quality recommendations, thus we can imagine that some high scored negative items for our users could be: “Back in Black” by AC/DC and “Take It Easy” by The Eagles, “Hearth of Gold” by Neil Young or “All Right Now” by Free[2]. For the AC/DC and Eagles songs, we can be pretty sure that these are good negatives since u does not particularly love those bands, but the same reasoning cannot be applied for the other two songs because the fact u has not listened to them could be due to some random reason. For example, u could not be aware of their existence or perhaps u owns a vinyl that contains those songs and therefore prefers to play them on the turntable. This leads to a tricky scenario: u would probably like Young’s and Free’s songs, but drawing them as negative examples makes the model lower their scores, so that these items would probably

[1] All these songs appear in the list of the top 10 most streamed songs on spotify. You can check the full list at https://en.wikipedia.org/wiki/List_of_most-streamed_songs_on_Spotify.

[2] These songs are all included in the Spotify Classic Rock 70s/80s.



Recommended
Based on what's in this playlist

Layla Derek & The Dominos	Layla And Other Assorted Love S...	ADD
Behind Blue Eyes The Who	Who's Next	ADD
More Than a Feeling Boston	Boston	ADD
You Can't Always Get What You Want The Rolling Stones	Let It Bleed	ADD

Figure 2.1.1: The Spotify "Recommended" section at the end of a user-generated playlist, that contains the top recommendations to add to the playlist.

not appear into u in the "Recommended" section (Figure 2.1.1) as one of the top items even if they would probably become positive interactions.

This issue is also related to what we highlighted at the end of Section 1.2.1, which is that implicit feedback is noisy by nature and that there are lots of reasons why an interaction might not have happened yet. We can conclude that, even if we definitely want to find solid negative items, this can lead to confusing situations in which very difficult and positive-like negative items are almost *too much* positive-like, so that it's too hard to discern whether these items are actual negatives or they are future positives (i.e. false negatives) that the user has ignored for some undefined reason. This is why we would like to try to measure this uncertainty and include it in the scores updating phase.

2.1.2 Adding Similarities

Our work takes place in implicit feedback scenarios, in which only a single type of feedback is observable and there is no additional information of any type. For this reason, most of the techniques explained in Section 1.11 cannot be applied in such a situation. Moreover, we would like to propose a sampling technique that is not computationally expensive and does not rely on external complex models, so we are rather referring to the heuristic sampling algorithms described in Section 1.10. Among these sampling techniques, there are two types of heuristic that are commonly used for modeling the negative sampling probability of the items:

- the item *popularity*, whether global or restricted to each user, that is typically used by static samplers to draw the negative items;

- the model *scores*, which are used by the dynamic adaptive samplers in order to compute the sampling distribution based on the current model belief.

These two metrics are appealing because they are always available and do not rely on external computationally expensive models like the generative approaches (Section 1.9.2.4), but there is some other heuristics that is always available from the user rating matrix and very easy to compute: **collaborative similarities**. Several works [26, 31] have been proposed to take into account the similarity based techniques inside Matrix Factorization models, but, to the best of our knowledge, there is no such thing in the specific context of the BPR sampling. The idea is that we can try to use the item-item similarity information, which derives from the simple calculations of heuristics such as the cosine or Jaccard similarities (see Section 1.4), to improve the sampling process in uncertainty situations like the one we described in the previous section.

2.2 Our Solution

Practically speaking, we would like to design a strategy which is able to draw difficult and informative triples without increasing the model complexity and which is also able to handle that a difficult negative item could become a positive one in the future. In particular, we want to handle the following cases in two separate manners:

1. if we are sure that the negative sampled item is an actual negative, but the model is ranking it as one of the top items, then we certainly want to push this item to the lower ranks;
2. if we are not entirely sure about the negative nature of the sample because we have some indication that it could become a positive interaction in the future, then we still want to lower its score but in a softer and smoother way.

In other words, we want to take into account the **confidence** of a negative sample, i.e. we would like to have a measure related to the probability of a sample of being a **true negative**. In order to design our strategy, we need to clarify a few aspects:

1. How can we measure the confidence of an item being negative?
2. How do we use this confidence value inside the BPR sampling?

3. Which sampler is likely to benefit the most from the confidence value?

2.2.1 How can we measure the confidence of an item of being negative?

The notion of confidence has been proposed in several works [60, 28, 26, 16, 44], especially as a measure of **positive confidence**, for example in contexts in which additional information can be observed as the number of times that an interaction occurred rather than the date of the last interaction. To our knowledge, the closest ideas to our thesis are in the two recent works [65, 37]. Let's compare the two:

- In [65], the confidence level is measured as the hardness of finding a good negative candidate, in particular as the number of iterations needed to find a "violated negative" instance (as described in Section 1.10.5). This notion of confidence is not a direct measure of confidence as it depends on the sampler; nevertheless, it should statistically give a good estimation. This confidence value is dynamic and needs to be computed for every single sample.
- In [37], confidence is formulated as the probability of an item of being negative and it directly uses the model scores. This confidence level, like the previous one, needs to be computed dynamically for each sample. Such operation makes it computationally expensive as it needs to be normalized over all the scores, which means that the the scores need to be computed at each iteration. That is why Chen et al. proposed an approximation of this algorithm, as discussed in Section 1.11.3.

In our case, instead of using the scores, we propose to employ the collaborative knowledge derived from the **item similarity** to give an estimation of the negativeness of a user-negative item pair. A first idea could be to measure the confidence of an item j of being negative with respect to the positive interaction (u, i) utilizing the item-item similarity $\text{sim}(i, j)$ between the positive and the negative items. In fact, if $\text{sim}(i, j)$ is close to 1, then the two items are similar, so if u likes i then u would probably also like j in the future (this is the main assumption on which collaborative filtering approaches are based); we can therefore use the $\text{sim}(i, j)$ as a measure of how likely j will become a positive item.
In order to improve this idea, we can extend this to the whole set of items liked by u. Similarly to the ItemKNN described in Section 1.6, we can compute the similarity between items using some of the metrics described in Sec-

tion 1.4 and then weight the similarity scores with the user profile in order to obtain a *personalized score* for each user-item pair. In particular, the personalized item similarity score that we obtain for user u and negative item j is:

$$\overline{r_{uj}} = \sum_{i \in I} r_{ui} \cdot \mathrm{sim}(i,j) \tag{2.1}$$

where $\mathrm{sim}(i,j)$ is the item-item similarity between items i and j.
From this computation, we get a set of scores for each user, but instead of using the notion of personalized similarity score as a direct measure of confidence, it is better to formalize this concept using **ranks**. That is because the vastness of scores is relative with respect to other items and it is also dataset specific, while ranks are absolute values. For this purpose, from the values obtained with Equation 2.1, we can obtain the item ranks for each user by sorting the user's scores in ascending order (from the least similar to the most similar item).

We can now define the **item similarity based confidence** of the not observed pair (u,j) as follows:

$$c(u,j) = \frac{1}{\hat{\rho}_{uj}}^{\alpha} \tag{2.2}$$

where $\hat{\rho}_{uj}$ is the rank of item j for the user u with respect to the item similarity weighted with u's profile and α is a hyperparamter to be tuned. It is important to note that the ranks are defined by sorting the similarity scores in ascending order, which means that a small rank (close to 1) indicates an item with low similarity, while a high one implies a large item similarity score. In other words, we introduce the use of the item similarity weighted with users' profiles to compute a measure of how likely an unobserved pair is to become a positive one. The role of α is to either smooth or increase the impact of the sample weights and it is dataset specific[3]; in Figure 2.2.1 you can see how different values of alpha affect the confidence function.

More in detail, we can observe how $\hat{\rho}_{uj} >> 1$ (i.e. a large score $\overline{r_{uj}}$ with respect to the other items) indicates that, from a similarity perspective, item j

[3]As a rule of thumb, when the number of items increases α would generally need to be decreased to yield optimal results.

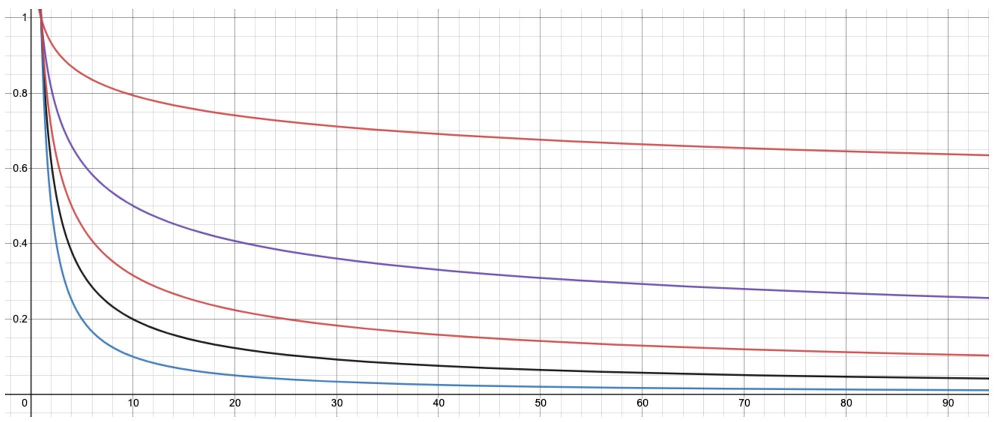


Figure 2.2.1: Plots for the function $\frac{1}{x}^{\alpha}$ for different values of α. The bigger is α, the steeper is the function.

is ranked among the top items, hence it is likely to become a positive interaction for u. This directly affects the confidence level as the confidence of an item with very large score will be very low. On the other hand, if $\overline{r_{uj}}$ is small, $\hat{\rho}_{uj}$ would be close to 1, thus its confidence will also be close to 1 (exactly one if $\hat{\rho}_{uj}$ is exactly 1).

2.2.2 How do we use this confidence value inside the BPR sampling?

Now that we have defined our confidence measure, we want to be able to use it in our BPR framework. As we have already mentioned, we are willing to use the original definition of the BPR in Equation 1.20 and we want to prevent changes in the loss. One way to proceed would be to use this sample confidence as **weight** of the sample as in [65, 37, 12] in the following way:

$$\sum_{(u,i,j)\in D_S} c(u,j) \cdot \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta ||\Theta||^2 \tag{2.3}$$

where $c(u,j)$ is the confidence value defined in Equation 2.3. In this manner, the confidence value acts as a multiplying factor that has a direct impact on the gradient:

$$\begin{aligned} &\sum_{(u,i,j)\in D_S} c(u,j) \cdot \frac{\partial}{\partial\Theta} \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta \frac{\partial}{\partial\Theta} ||\Theta||^2 \\ \propto &\sum_{(u,i,j)\in D_S} \frac{-e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot c(u,j) \cdot \frac{\partial}{\partial\Theta} \hat{x}_{uij} - \lambda_\Theta \Theta \end{aligned} \tag{2.4}$$

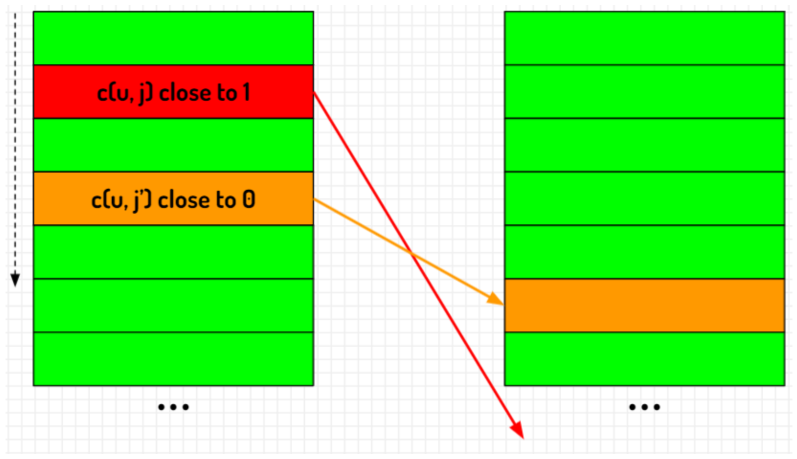


Figure 2.2.2: The two stacks represent the items sorted in descending order by the current model score, before (left) and after (right) the updates.

This is how we can handle the two cases described at the beginning of this section, as you can see in Figure 2.2.2. The figure shows the relative ranks of the items for user u in descending order, according to the scores assigned by the model. Let's suppose that the sampler extracts the negative instance j (in red), which has a high score according to the current model belief and a high confidence of being a true negative from a collaborative perspective; in this case, the model updates its score and pushes it to the bottom. In the other case, the negative sampled item j' (in orange) has a high model score but a low confidence of being negative, thus the model updates its score by a small factor taking this uncertainty into account and pushing it just a few ranks down.

Our sample weighting strategy has the following characteristics:

- It is **static**, since each sample's weight can be easily computed a priori and then remains fixed. It makes a remarkable difference with respect to [65, 37], and it is motivated by our efforts to avoid additional complexity. In fact, once the weights are computed, the confidence for the pair (u, j) can be obtained in constant time ($\mathcal{O}(1)$).
- It is **context-dependent**, the confidence formulated in Equation 2.3 is not defined for the single item but for the pair (u, j), so $c(u, j)$ and $c(u', j)$ generally are different values, and this is due to the fact that the item similarity score is weighted with the user's profile (Equation 2.1).
- It employs the **item similarity** power of items directly inside the BPR framework. This differentiates from [65, 37] since they utilise the ranks

of the items according to the model scores. What made this choice preferable, in addition to the reduced complexity and the great value of the item similarity information, is that we wanted to try something different with respect to the scores. We are willing to use dynamic samplers, as they are proved to be state of the art [50, 65, 59, 37, 66], however, it is not ideal to utilize the same exact scores for both drawing and weighting the sample since the sampler would draw high ranked items and, at the same time, reduce their importance. Using a different metric for weighting the sample means that we don't always penalize a sample that has a high score, since this depends exclusively on the item similarity between the items in the user profile and the negative item in the sample. Adding a metric such as item similarity, even if static, could bring benefits to the model as it is able to describe the data in a different manner compared to what the model already knows.

- It is totally **general**. The concept of confidence formulated in Equation 2.3 does not depend on the scores (like [37]) neither on the specific underlying sampler (like [65]), but it is only based on similarity information that we can extract from the original user rating matrix. This means that this exact same confidence weighting strategy can be applied to a wide variety of algorithms (similar to [44, 12]) and sampling methods. From a broader perspective, we formulated the concept of item similarity based confidence, but the same thing can be done with the user similarity as well as with some more elaborate collaborative methods such as the graph based similarities mentioned in Section 1.6.4.1 or even with a more complicated model.

2.2.3 Which sampler is likely to most benefit from the confidence value?

All the sampling techniques we discussed should benefit from this strategy, but this is particularly true for those methods that are able to select the more difficult negative items, because they are also prone to extract future positives.
Among the existing techniques, we would like to make a couple considerations:

- the static samplers that we presented will likely draw a small number of future positives at each epoch for what we said in Section 2.1.1. Moreover, as they are static, the percentage of future positive items sampled at each epoch will not significantly change. A different case could be represented by the popular oversampler, which could be able to select a good number of future positive instances in those datasets that have a very long tailed item popularity distribution.

- VINS (Section 1.10.5) is a hybrid sampler that first samples a set of items using the global popularity and then takes into account the item scores. A possible issue with this approach is that the first sampling phase that uses the popularity can limit the model performance due to what we discussed in Section 2.1.1.
- the ranking-aware rejection sampling in [66] (Section 1.10.6) is a good trade-off between recommendation quality and performance. Experimental results [66, 26] show that limiting the number of items in the group leads to a better performance in both complexity and accuracy. In this regard, we suspect one of the possible reasons behind this to be that using larger groups of items means selecting the negative instances that are ranked highest, i.e. the most positive-like items, which could be particularly impactful in reducing the ranking performance as the model accuracy increases over the training epochs.

For these reasons, we suggest to use the confidence sample weighting in combination with the dynamic adaptive sampler and especially with the ranking-aware rejection sampling [66] due to the clear benefits we have amply discussed so far. These samplers have the great advantage of adapting to the model and improve the learning process as it goes on and they are proved to be able to select the more difficult samples with respect to the other techniques.
In the following chapters we are applying our strategy to four real-life datasets in order to compare the relative improvements that the similarity-based confidence weighting strategy is able to bring.

Chapter 3

Evaluation

A proper design of the evaluation of a RS is crucial in order to get an understanding of the effectiveness of the various algorithms, and it is often multifaceted, meaning that a single criterion cannot capture many of the goals of the designer, and a difficult and tricky task for several reasons [23, 10]. The same algorithm can perform differently on different datasets, better or worse depending on the dataset peculiarities and on the algorithm strenghts and weaknesses. The same algorithm can also perform differently on the very same dataset due to the way the data is prepared or splitted, due to the parameter tuning phase of the algorithm or even due to the different goals that one could have.
In this part we introduce the metrics, dataset and evaluation criteria for measuring the performance of our proposed similarity-based sample weighting.

3.1 Evaluation Metrics

A RS can be evaluated using either **online** or **offline** methods. Online evaluations originated from online advertisement and e-commerce and they generally measure the acceptance rates of recommendations in real-world cases, using classic measures such as the click-through rate (CTR) which is the ratio of user clicks on items that were recommended [4]. Since online evaluations require active user participation, it is often not feasible to use them in benchmarking and research, hence why we resort to the offline methodology to evaluate our experiments. Offline evaluations typically measure the quality of a recommender system based on a ground-truth according to a metric. There are generally three broad classes of metrics:

- **Predictive accuracy** metrics, which evaluate how close the ratings estimated by a recommender are to the true user ratings.
- **Classification accuracy** metrics, that measure the quality of the recommendation by assessing the number of times that the RS is correct about predicting that there will be an interaction between a user and an item.
- **Rank accuracy** metrics, which give an evaluation of the RS's ability of producing an ordered list of items to the user that actually matches the order of preferences that a user has.

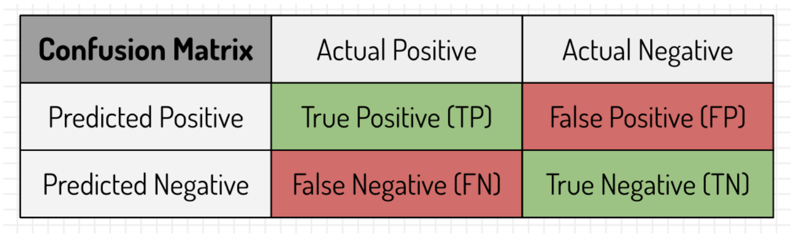
We only deal with implicit feedback datasets, therefore predictive accuracy metrics aren't useful for our analysis since the ratings are expressed with a binary value. In the following paragraphs we present the metrics used to evaluate our proposed strategy.

3.1.1 Classification Metrics

Classification metrics are appropriate for tasks such as those involving binary feedback, as these metrics do not attempt to directly measure the ability of an algorithm to accurately predict ratings. Some deviations from actual ratings are tolerated, as long as they do not lead to classification errors [20]. These metrics measure the recommender's ability to distinguish *relevant* from *not relevant* items. The definition of *relevance* has been extensively discussed in the data mining and information retrieval field, so we first want to give a brief overview of some basic definitions that are necessary to describe the metrics that we are going to use.

Suppose we have a dataset D with two classes of items[1], containing both relevant (referred to as P) and not relevant items (referred to as N). The predictive accuracy of a classification algorithm on D can be summarized using a 2×2

[1] In a RS scenario, if the dataset is implicit then the positive items are the relevant ones. If the dataset is explicit, the ratings need to be transformed to binary scale through an implicitization.



Confusion Matrix	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Figure 3.1.1: The confusion matrix.

error matrix called **confusion matrix** (Figure 3.1.1), which has the instances of the predicted class as rows and the true ones as columns.

The four entries in the confusion matrix have the following meanings:

- **True Positive** (TP): relevant items correctly predicted as relevant.
- **False Positive** (FP): not relevant items incorrectly predicted as relevant.
- **True Negative** (TN): not relevant items correctly predicted as not relevant.
- **False Negative** (FN): relevant items incorrectly predicted as not relevant.

By combining these values we can obtain the metrics that we are going to use to evaluate the classification capability of the implemented algorithms: **precision** and **recall**.

3.1.1.1 Precision

The precision metric, also called positive predictive value, is the ratio of relevant items among the retrieved items, thus it represents the probability of a selected item being relevant:

$$Precision = \frac{\text{Number of relevant items recommended}}{\text{Number of recommended items}} \tag{3.1}$$

Precision can be computed from the confusion matrix as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

In RS, the goal is to produce a personalized list of relevant items for a specific user and the problem is defined as a top-N recommendation task, as described in the first chapter. In order to address this matter, the classic definition of precision is adapted in the RS world using the notion of "**cutoff**" k, which is an integer number expressing that we do not compute the precision over the whole set of items but only on the first k recommended ones. The precision with cutoff k, indicated as $Precision@k$, is calculated as:

$$Precision@k = \frac{\text{Number of relevant items@ } k}{k} \tag{3.3}$$

3.1.1.2 Recall

Recall is defined as the ratio of relevant instances selected among the total number of relevant items available, thus it's the probability of selecting a relevant item:

$$Recall = \frac{\text{Number of relevant items recommended}}{\text{Number of relevant items in the ground truth}} \tag{3.4}$$

It is also called **sensitivity** and it can be calculated with the confusion matrix:

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

The same considerations we made about cutoff can also apply for recall, with *Recall@k* defined as the proportion of relevant items found in the top-k recommendations:

$$Recall@k = \frac{\text{Number of relevant items@ } k}{\text{Number of relevant items in the ground truth}} \tag{3.6}$$

3.1.2 Ranking Metrics

Rank accuracy metrics evaluate the ability of a RS to produce an ordered list of items that matches how the user would have ordered those same items [23]. As these metrics check the order of the items in the list, they are best suited for explicit feedback dataset, since the ordering can be measured in a more accurate way using the explicit ratings given by the users. In implicit feedback domains, these metrics could be very sensitive because the only available information is if an instance is a relevant or a not relevant one, without further measure of said relevance.

These metrics differ from the prediction accuracy metrics because the absolute preference of the user for an item does not matter in this case, the only thing that matters is the relative order of the recommended items.

3.1.2.1 NDCG

NDCG, that stands for Normalized Discounted Cumulatative Gain, is an evaluation metric derived from the information retrieval area, often used to measure effectiveness of web search engine algorithms or related applications.
In such scenarios, when examining the ranked results of a query, there are two crucial observations to be made [29]:

- highly relevant instances are more valuable than marginally relevant instances;
- the greater the ranked position of a relevant instance, the less valuable it is for the user, because it is less likely that the user will ever examine the item.

First of all, we define the relevance of an item rel_i:

$$rel_i = \begin{cases} 1 & \text{item in position } i \text{ is relevant} \\ 0 & \text{item in position } i \text{ is not relevant} \end{cases} \tag{3.7}$$

The Cumulative Gain (CG) is formulated as the sum of all the relevance scores in a the retrieved set:

$$CG = \sum_{i=1}^{n} rel_i \tag{3.8}$$

The CG can be evaluated only at rank p and not on all the instances, so that the summation can be restricted to p documents.

From the CG, we can define the Discounted Cumulative Gain (DCG) that formalizes the previous assumptions by crediting a retrieval system for retrieving relevant instances by their possibly weighted degree of relevance, which is discounted by a decreasing function of the rank of the item. The discounting function is needed because it progressively reduces the score of the document as its rank increases but not too heavily, to allow for user persistence in examining further documents [29]. The most common discounting function is the logarithm. The DCG at rank p is calculated as:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_b(i+1)} \tag{3.9}$$

where b is the base of the logarithm. In particular, a small logarithm base models an ”impatient” user, while a larger one models a more patient and persistent user.

The list of the search results depends inevitably on the query, therefore comparing a search engine’s performance from one query to the another cannot be consistently achieved using DCG alone. The Normalized Discounted Cumulatative Gain is defined for this purpose, combining the DCG with the Ideal DCG (IDCG) at p:

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_b(i+1)} \tag{3.10}$$

which is the ideal reordering of the documents obtained by sorting them according to their relevance, where $|REL_p|$ is the list of relevant items, ordered by relevance, in the ground truth up to position p. We can finally define the NDCG as the ratio:

$$NDCG_p = \frac{DCG_p}{IDCG_p} \tag{3.11}$$

In RS, this metric has the same meaning if we consider that the retrieved document set corresponds with the ordered list recommended items for the user and limiting at top p means applying a cutoff.

3.1.2.2 MAP

Mean Average Precision is another widely used metric for measuring the search engines’ ability to correctly rank the retrieved documents and it is one of the most important metrics in the RS literature. As the name suggests, it is computed as the mean of the Average Precision (AP); the AP is the average of the precision value obtained for the set of top n items existing after each relevant item is retrieved. The formula for the AP is the following:

$$AP@n = \frac{\sum_{k=1}^{n} \text{Precision } @k \cdot rel_k}{\text{Number of relevant documents}} \tag{3.12}$$

where rel_k is defined as in Equation 3.7 and the $Precision@k$ is the precision computed on the first k items, as in Equation 3.3. From this definition, we can compute the MAP as:

$$MAP@n = \frac{1}{n} \sum_{k=1}^{n} AP@k \tag{3.13}$$

From this formula we can see that the MAP, even if it is a variant of the Precision, is a ranking metric because it penalizes the wrongly ordered items.

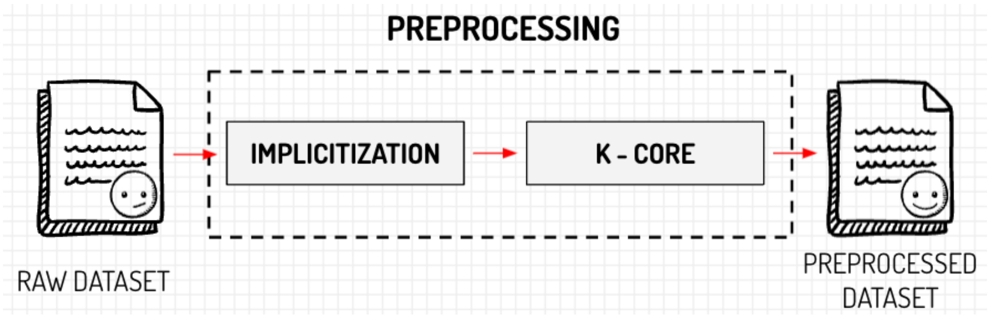


Figure 3.2.1: The preprocessing phase.

3.2 Preprocessing

Before diving into details of the four real datasets we used for our experimental evaluation, we want to briefly present the preprocessing steps that have been applied to these datasets, i.e. all the procedures that prepare and transform the original raw data into the one that we are directly using to evaluate our algorithms. Learning which preprocessing steps have been executed is often crucial for understanding the performance of the algorithms, as different preprocessings can hardly affect the algorithms overall and their relative performance. Figure 3.2.1 shows the full preprocessing pipeline.

3.2.1 Implicitization

Among the datasets we used, three are explicit and one is implicit. As previously discussed, our analysis takes place in implicit feedback scenarios, so the first thing we have to do is transforming the explicit ratings into a binary scale using the so called implicitization process that we mentioned in Section 1.2.1. Given a threshold t, this procedure maps the dataset explicit ratings r_{ij} to:

- 1 if $r_{ij} > t$, which means that the rating expressed is above the threshold, therefore positive;
- 0 if $r_{ij} \leq t$, so if the rating is missing or is present but below the threshold.

Since each dataset has its own rating system, the threshold value t is dataset specific.

The example in figure 3.2.2 shows the process of transforming an explicit dataset with a 5 star rating system using the following logic:

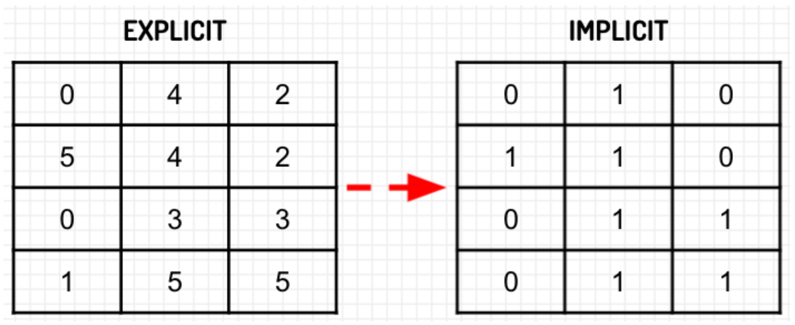


Figure 3.2.2: The process of transforming an explicit dataset with a 5-star rating to an implicit one, using a threshold of 2.

- unobserved interactions remain unobserved;
- observed interactions in the original data with a rating value less than or equal to 3 are considered as negative interactions, so these interaction have 0 rating in the implicit dataset;
- observed interactions with a rating greater than 3 are considered positive interactions, thus have 1 rating in the implicit dataset.

In other words, after setting 3 as the threshold, r_{ij} is 1 in the implicit dataset only when $r_{ij} > 3$ in the original explicit dataset, otherwise it is 0.

3.2.2 K-core

After the implicitization process, we have obtained an implicit version of the original dataset. The next step would be removing those items and users that have very few interactions, since these items/users are proved to create complications in the learning problem. This issue also takes the name of *cold case* scenario and it is a broadly discussed topic in the literature [38, 15]. From now on, we are going to refer to this step as **k-core** preprocessing.
The procedure can affect both items and users and, as we mentioned above, it consists of removing those items/users that have less than s interactions, i.e. that have less than s non-zero entries in the respective row (for the users) or column (for the items).
This step is crucial and can highly affect the algorithm performance. The reason for applying this technique is the benefit of only having users or items with a minimum number of interactions, in order to remove noisy data. It is worth noticing that a high k-core could favor certain algorithms over others, especially the ones that perform worse in cold case scenarios.
In all our datasets we apply a k-core of five to keep only those users and items

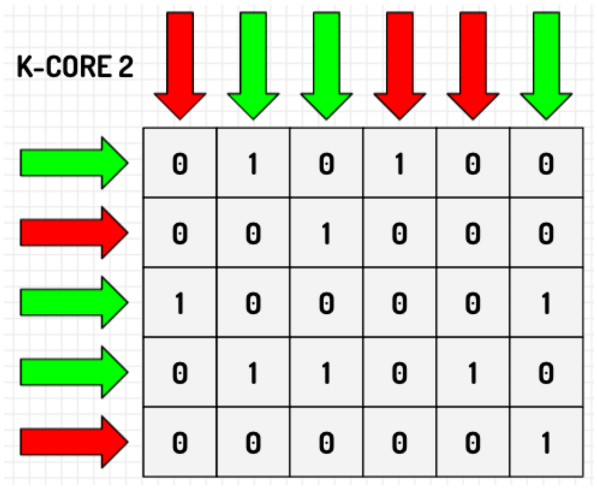


Figure 3.2.3: The k-core at 2 preprocessing. The green arrows represent the rows/columns (users/items) that remain at the end of this phase, i.e. those rows/columns with at least two interactions.

that have at least five interactions.

3.2.3 Dataset Partitioning

Splitting a dataset is essential in machine learning and data mining to obtain an unbiased evaluation of the prediction performance.

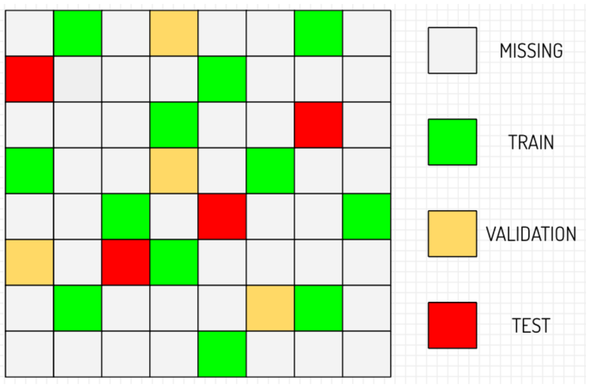


Figure 3.2.4: The split procedure adopted to obtain the training and testing sets.

In RS, the dataset splits are represented by different user rating matrices. We split every preprocessed user rating matrix in three separate subsets (Figure 3.2.4)

- **Train** R_{train}: this accounts for 60% of the dataset interactions;
- **Validation** R_{val}: it contains 20% of the interactions;

- **Test** R_{test}: it is composed by 20% of the interactions.

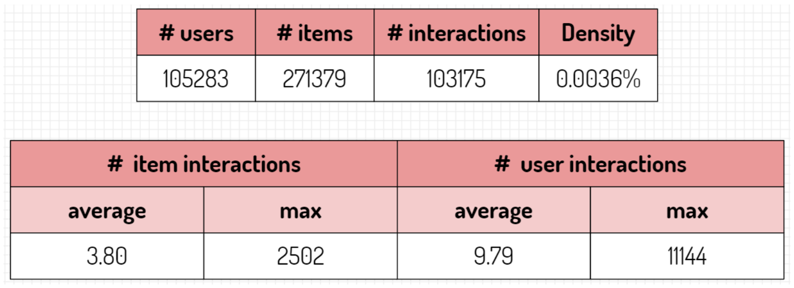
R_{train} is used as training set for tuning the hyperparameters and evaluating the offline performance. For the evaluation on the final test set, the training data is obtained by merging R_{train} and R_{val} and the performances are evaluated on R_{test}. This way, the training data is composed by 80% of the interactions but the training and testing data are still completely disjointed.

3.3 Datasets

As already mentioned, the evaluation has been performed using four real datasets that are widely adopted in the RS field: BookCrossing, Movielens1M, LastFMHetrec2011 and Yahoo Movies.

3.3.1 BookCrossing

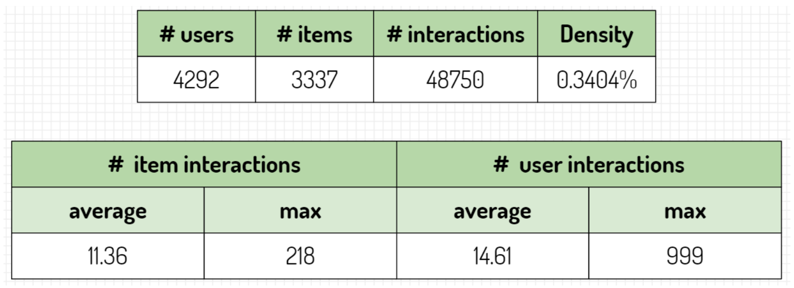
The BookCrossing dataset[2] contains the data collected during a 4-week crawl from the BookCrossing community (www.bookcrossing.com).
This dataset contains 105.283 users (anonymized but with demographic information) providing 1.031.175 ratings on 271.379 books. The ratings are explicit, in a 1-10 scale. Figure 3.3.1 sums up the raw dataset information.



# users	# items	# interactions	Density
105283	271379	103175	0.0036%

# item interactions		# user interactions	
average	max	average	max
3.80	2502	9.79	11144

Figure 3.3.1: The original BookCrossing dataset information.



# users	# items	# interactions	Density
4292	3337	48750	0.3404%

# item interactions		# user interactions	
average	max	average	max
11.36	218	14.61	999

Figure 3.3.2: The BookCrossing dataset information after the implicitization and k-core steps.

Since the data is explicit, we have to perform both implicitization and k-core. For the implicitization, we use a threshold of $t = 7$ to consider the interactions that have a higher rating as positive interactions. The k-core is set to 5.
Table 3.1 shows how the interactions are splitted in the train, validation and test matrices.

[2] At http://www2.informatik.uni-freiburg.de/ cziegler/BX you can find the whole dataset.

Train	Validation	Test
29284	9716	9693

Table 3.1: Interactions split among train, validation and test in the preprocessed BookCrossing dataset.

The preprocessing steps hardly reduced the dimensions of this dataset, in fact this is the smallest one we consider. Although the preprocessing increased the dataset density, this dataset is still the sparsest dataset that we use for our analysis, with only 0.3404% of positive feedback.
Figure 3.3.3 shows the popularity distribution of the remained items, from which it is easy to see the long tail phenomenon described in Section 1.6.1.

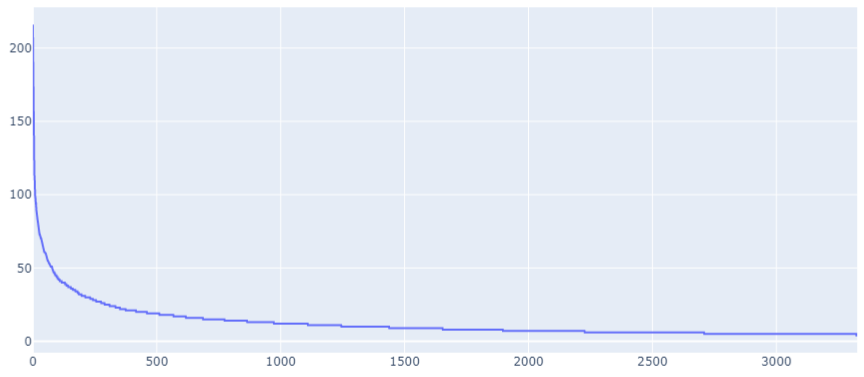
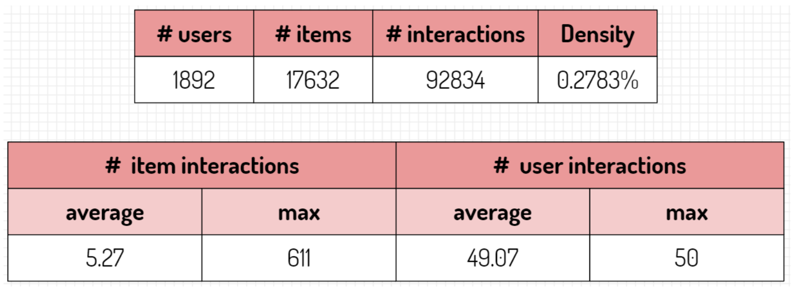


Figure 3.3.3: The popularity distribution in the preprocessed BookCrossing dataset.

3.3.2 LastFm

The LastFMHetRec2011 dataset [24] was collected for the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems, held in Chicago in 2011. The HetRec workshop represented a meeting point for researchers and practitioners interested in addressing the challenges posed by information heterogeneity in recommender systems and studying information fusion in this context [24]. In fact, this data contains various information about social networking, tagging and music listening of a set of users from Last.fm online music service with the the purpose of providing better personalized services in many information-seeking and e-commerce applications.

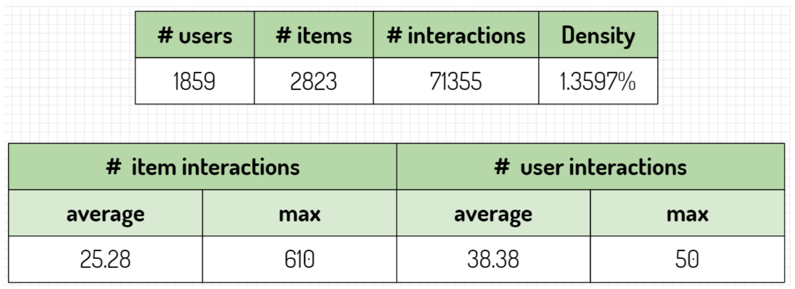
For this thesis, we are only interested in the interactions between users and items and, in this regard, we want to point out that the LastFM dataset is the only intrinsically implicit feedback dataset that we use for our evaluation. The original dataset has a sparsity $> 99\%$, with 92834 total interactions, 1892 users and 17632 items. As you can observe (Figure 3.3.4), the number of items is much higher than the number of users (almost 10 times higher), but most of these items have very few interactions. In fact, after applying the k-core at five preprocessing[3], only 16% of the items has been kept in the dataset (Figure 3.3.5). The number of average interactions per user and items is higher with respect to BookCrossing as the density is ca. four times higher.



# users	# items	# interactions	Density
1892	17632	92834	0.2783%

# item interactions		# user interactions	
average	max	average	max
5.27	611	49.07	50

Figure 3.3.4: The original LastFm dataset information.



# users	# items	# interactions	Density
1859	2823	71355	1.3597%

# item interactions		# user interactions	
average	max	average	max
25.28	610	38.38	50

Figure 3.3.5: The LastFm dataset information after the implicitization and k-core steps.

Figure 3.3.6 shows the popularity of the items; once more, we can see the classic long tail, but a bit less accentuated with respect to BookCrossing.

[3]This is the only preprocessing procedure that we applied, as the dataset natively comes as implicit.

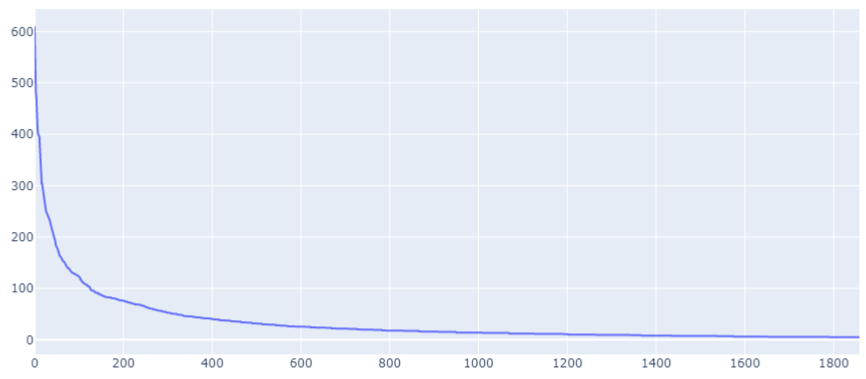


Figure 3.3.6: The popularity distribution in the preprocessed LastFm dataset.

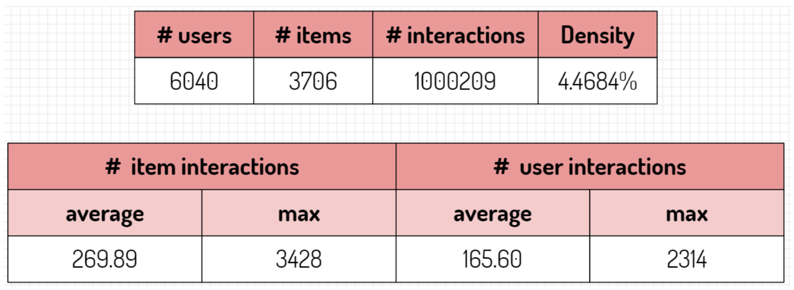
Table 3.2 sums up how the interactions are splitted in the train, validation and test rating matrices.

Train	Validation	Test
42798	14348	14209

Table 3.2: Interactions split among train, validation and test in the preprocessed LastFm dataset.

3.3.3 Movielens1M

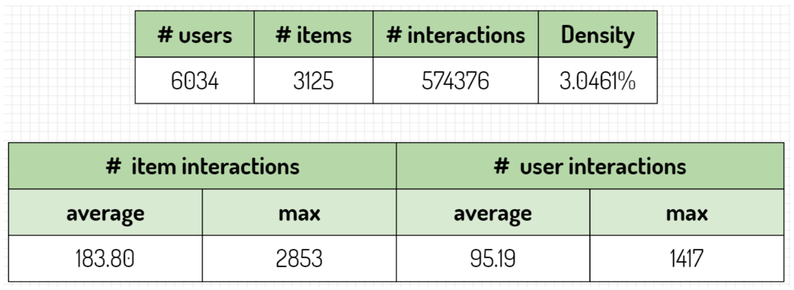
The Movielens dataset comes in three standard versions that have different names according to the number of interactions the dataset contains: Movielens100K, Movielens1M (the one we chose) and Movielens20M. These datasets are among the most used and discussed datasets, not only in the RS literature but also in other research areas and in the industry field. They are heavily downloaded (140,000+ downloads in 2014) and referenced (7,500+ references to “movielens” in Google Scholar)[19]. These datasets are a product of member activity in the MovieLens movie recommendation system, an active research platform that has hosted many experiments since its launch in 1997 [19].



# users	# items	# interactions	Density
6040	3706	1000209	4.4684%

# item interactions		# user interactions	
average	max	average	max
269.89	3428	165.60	2314

Figure 3.3.7: The original Movilens1M dataset information.



# users	# items	# interactions	Density
6034	3125	574376	3.0461%

# item interactions		# user interactions	
average	max	average	max
183.80	2853	95.19	1417

Figure 3.3.8: The Movielens1M dataset information after the implicitization and k-core steps.

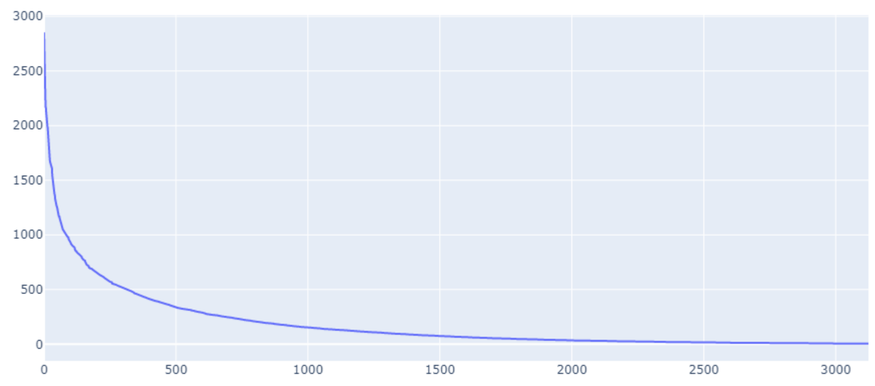


Figure 3.3.9: The popularity distribution in the preprocessed Movielens1M dataset.

As we mentioned before, we adopted the Movielens1M version, which has about 1 million interactions regarding 6040 users and 3706 items. Movielens datasets contain anonymized information about people's expressed preferences for movies and the ratings are in the 1 to 5 stars range, hence the need for an implicitization step. In details, we set a threshold value of 3 to keep the 4 or 5 stars interactions as positive ratings, and then we applied the usual

k-core at five. The results of the preprocessing steps are summed up in Figure 3.3.7 (the raw data) and Figure 3.3.8 (the preprocessed data). As you can observe from the figures, almost all users and items have been kept from the raw to the preprocessed dataset.
One noticeable difference is the density of this dataset, which is almost 10 times higher in its preprocessed version with respect to BookCrossing. Additionally, the average number of interactions for users and items is much higher with respect to the other ones (w.r.t. BookCrossing ca. 16x for average item interactions and ca. 7x for the users).

In Table 3.3, we reported the interactions' split between train, test and validation after the preprocessing.

Train	Validation	Test
344380	115185	114811

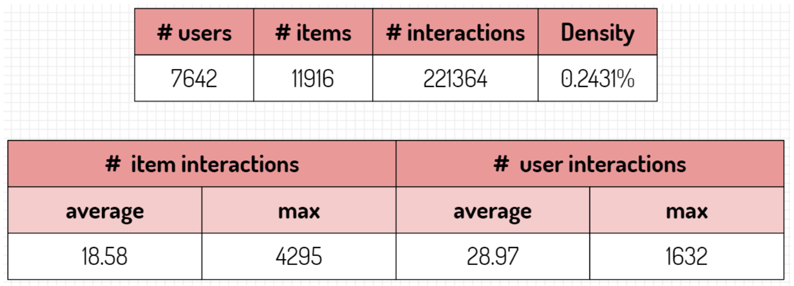
Table 3.3: Interactions split among train, validation and test in the preprocessed Movielens1M dataset.

3.3.4 Yahoo Movies

Yahoo Movies is a dataset [63] containing a small sample of the Yahoo! Movies community's preferences for various movies. The original dataset has 221364 ratings from 7642 users that interacted with 11916 movies. The user ratings are explicit, from 1 to 5, same as the Movielens dataset. We applied a threshold of 3 for the implicitazion and the usual k-core at five.
The preprocessing phase removed less than the 30% of the total interactions and the final density of the preprocessed dataset in 0.6699%, which makes this dataset the second most sparse after BookCrossing. Table 3.4 shows how the interactions are splitted among the matrices:

Train	Validation	Test
96875	32305	32376

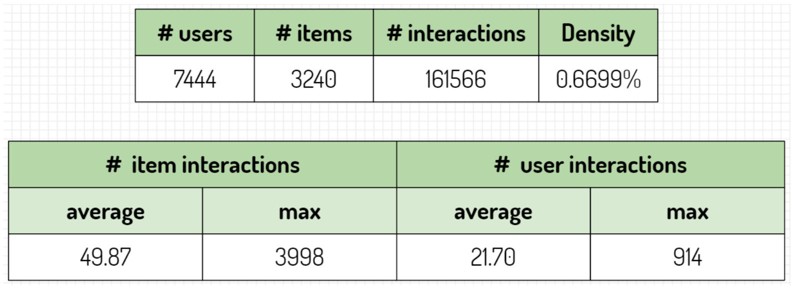
Table 3.4: Interactions split among train, validation and test in the preprocessed Yahoo Movies dataset.



# users	# items	# interactions	Density
7642	11916	221364	0.2431%

# item interactions		# user interactions	
average	max	average	max
18.58	4295	28.97	1632

Figure 3.3.10: The original Yahoo Movies dataset information.



# users	# items	# interactions	Density
7444	3240	161566	0.6699%

# item interactions		# user interactions	
average	max	average	max
49.87	3998	21.70	914

Figure 3.3.11: The Yahoo Movies dataset information after the implicitization and k-core steps.

From Figure 3.3.12 we can notice how the long tail phenomenon is particularly visible in the Yahoo Movies dataset, with 5% of the items accounting for over 50% of the total interactions.

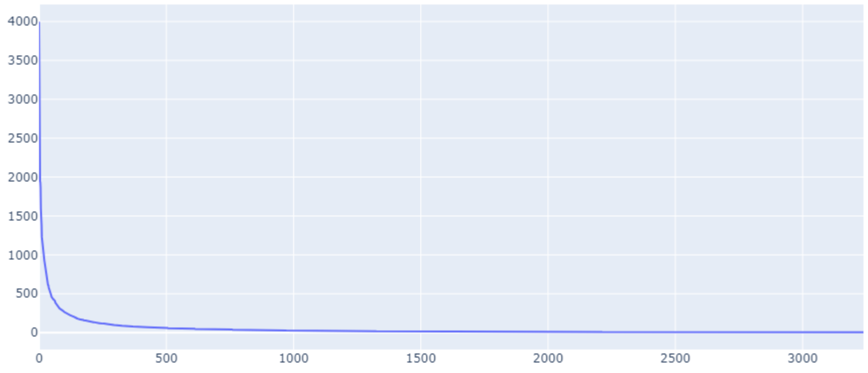


Figure 3.3.12: The popularity distribution of the items in the preprocessed Yahoo Movies dataset.

3.4 Implementation

In this section we briefly outline the most salient implementation details about the technologies we used and the way algorithms have been coded and tuned.

3.4.1 Technologies

The whole research has been backed by extensive experiments written in Python 3.9[47]. Most of the work was supported by the following libraries:

- NumPy [42]: a C based library that offers comprehensive mathematical functions.
- SciPy [54]: another very popular C/C++ based library that contains several optimization algorithms and utilities for scientific computation tasks. In particular, we relied on this library for the sparse matrix representation, using the CSC, CSR and COO matrix formats.
- SimilariPy [55]: package written in Cython for the fast computation of matrix multiplications and similarity metrics, we used it for computing the cosine similarity.

In addition to these tools, we also wrote the most computationally expensive part in Cython, a superset of the Python language that supports calling C functions and declaring C types; this made the process consistently faster and improved the performance. All the algorithms ran on a 32GB RAM Windows PC without the use of a GPU.

3.4.2 Hyperarameter Tuning

There are a lot of parameters in both the BPR and the analyzed sampling methods. In order to find the optimal ones, we adopted a random search approach over the hyperparameter space using 50 different configurations for every sampling algorithm. We used the same seed for every run so that each algorithm within a dataset was tested against the same set of configurations. The BPRMF parameters tuned by the random optimizer are the following:

- *Number of Latent Factors*: the number of factors for the MF model, ranged between 150 and 300;
- *Positive Regularization*: the regularization value for the positive class, in range 0.0001 to 0.001;

- *Negative Regularization*: the regularization value for the negative class, in range 0.0001 to 0.001.

For the sample weighting confidence, we ran the optimizer for different values of the α hyperparamter in Equation 2.3 in order to find the best value, generally small values in the 0.3-0.6 range seem to yield good results.

Regarding the samplings, for VINS [65] we followed the authors' suggestion and set s to 5 and the $max\_iterations$ to 64. For the ranking aware rejection sampler we used small groups (5, 10, 15 and 20) as suggested in [66, 26] and decided to set all the β_k parameters to zero, thus extracting the item with largest score as described in Section 1.10.6.2 for two reasons:

- complexity, since this way the scaling is linear (see Section 1.10.6.2);
- instead of analyzing the single sampler, we are rather interested in measuring the relative improvement of the item similarity based confidence sampling that we proposed.

Finally, for what concerns the item similarity used for computing the confidence, we used the cosine similarity (defined in Appendix 1.4) with a shrink factor of 5 to reduce the noise. Note that we did not use any feature weighting algorithm like TF-IDF and we did not tune the topK parameter as our main focus is just to have a straight item-item score for each item, since tuning an ItemKNN recommender is not our purpose. However, we do not want to exclude that a different tuning of the similarity could also improve our results.

3.4.3 Number of Epochs and Early Stopping

As already mentioned, the BPR loss can be optimized with gradient descent optimization methods. We opted for the Adam gradient descent based algorithm [33] as it is lightweight and it also achieved the best performance. These optimizers have two key hyperparameters:

- **Learning Rate**, which controls the update size at each iteration while moving toward a minimum of the loss function;
- **Batch Size**: the number of training examples utilized in one iteration for the gradient update.

These parameters are strongly related and affect the number of epochs needed for the algorithm to converge.

Since one of the aspects that we want to evaluate is the convergence speed, measured as the **number of epochs** needed to reach the convergence, we decided to keep the learning rate and batch size parameters fixed, otherwise we could not measure the convergence speed of two different algorithms that have different learning rate and batch size using the number of epochs.

To find the number of epochs for reaching the convergence, we decided to use the Early Stopping method, which is a very popular technique adopted in ML. In practice, once every n iterations, the performance of the algorithm is evaluated in the validation set, according to a certain predefined metric. If the metric stops growing for m evaluations, the procedure is stopped and we conclude that the best number of epochs is the last one in which the metric has grown. The m value is sometimes called *patience*, as it is the number of times that the algorithm can underperform before it is stopped. In our case, we evaluated the recall 3.1.1.2 metric at 10 every 5 epochs and we used a patience of 2, so that the training stopped if the recall did not improve after two consecutive evaluations. Moreover, since some configurations got stuck in very small updates, we also stopped the training if after two consecutive evaluations (10 epochs) the relative improvement was lower then the 0.5%.

Chapter 4

Results

In this chapter, we are finally ready to assess the idea of weighting samples using the item similarity confidence defined in the second chapter. In particular, the analysis is carried out on the following samplings:

- **UNI**: the random sampler described in Section 1.10.1.
- **POP**: the popularity oversampling strategy described in Section 1.10.2, utilizing the empirical distribution of the items.
- **IMB**: the imbalance rejection sampling in Section 1.10.4.
- **DYN**: the ranking aware rejection sampler in Section 1.10.6. For this kind of sampler, we tried various group sizes:
  - **DYN-5**: groups of 5.
  - **DYN-10**: groups of 10.
  - **DYN-15**: groups of 15.
  - **DYN-20**: groups of 20.

Moreover, we also evaluate the weighted versions of these algorithms, especially focusing on the dynamic ones as they seem to be able to yield the best results. To highlight the fact that the item similarity weighting has been used for one of the previous algorithms, we use the letter **W** so that with W-DYN-5 we refer to the weighted version of DYN-5 enhanced with our proposed strategy.

4.1 Prediction Accuracy

In the following paragraphs we show the accuracy results obtained on the dataset for the classification and ranking accuracy metrics as well as a couple observations that emerged from this analysis.
The results are showed in tabular form as well as with different plots. For the tables, the upper part of each table contains the standard versions of the samplings, as described in Section 1.10, while the bottom part shows the results of the same samplings enhanced with our proposed item similarity based confidence described in Chapter 2. In each table, both for the weighted and non-weighted sampling methods, the best technique is in bold while the second best is underlined.

4.1.1 Classification Metrics

The following four pages contains the tables and plots that sum up the classification accuracy of the algorithms on the four datasets.

Sampling	Prec@5	Prec@10	Prec@25	Rec@5	Rec@10	Rec@25
UNI	0.0378	0.0260	0.0159	0.0637	0.0845	0.1235
POP	0.0242	0.0174	0.0108	0.0434	0.0582	0.0854
IMB	0.0370	0.0256	0.0152	0.0621	0.0829	0.1191
VINS	0.0364	0.0257	0.0155	0.0613	0.0840	0.1231
DYN-5	**0.0388**	**0.0273**	**0.0169**	**0.0664**	**0.0924**	**0.1350**
DYN-10	0.0384	0.0272	0.0155	0.0649	0.0889	0.1224
DYN-15	0.0383	0.0265	0.0154	0.0646	0.0866	0.1234
DYN-20	0.0374	0.0254	0.0148	0.0638	0.0839	0.1165
W-POP	0.0307	0.0231	0.0147	0.0538	0.0772	0.1153
W-IMB	0.0433	0.0307	0.0187	0.0724	0.1007	0.1454
W-DYN-5	0.0463	**0.0335**	**0.0201**	0.0751	0.1052	**0.1544**
W-DYN-10	**0.0477**	0.0334	0.0196	**0.0823**	**0.1089**	0.1522
W-DYN-15	0.0464	0.0331	0.0195	0.0771	0.1075	0.1511
W-DYN-20	0.0457	0.0312	0.0182	0.0772	0.1019	0.1409

Table 4.1: Precision and recall scores with cutoff 5,10 and 25 on the BookCrossing dataset.

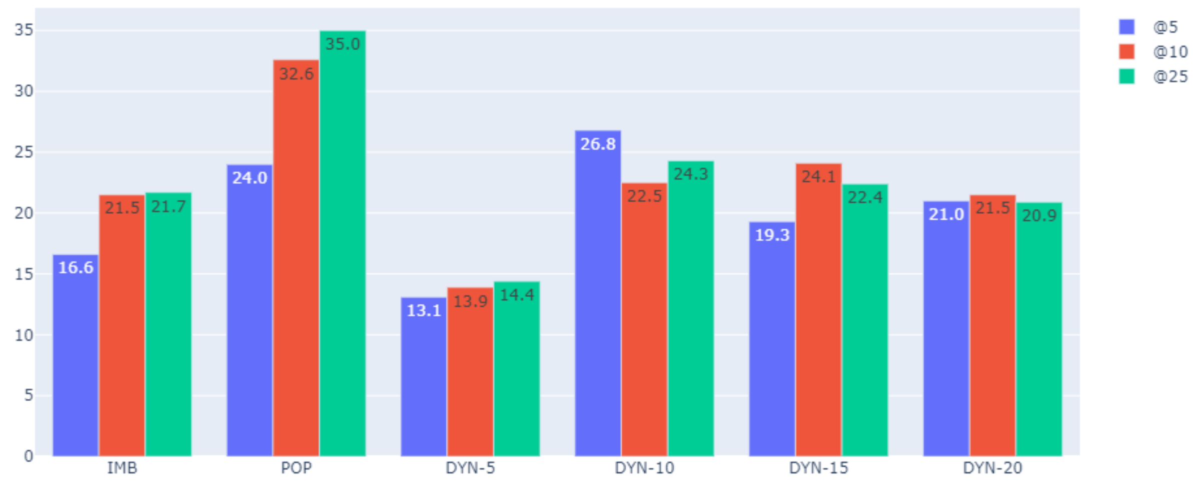


Figure 4.1.1: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the recall scores on BookCrossing.

Sampling	**Prec@5**	**Prec@10**	**Prec@25**	**Rec@5**	**Rec@10**	**Rec@25**
UNI	0.2327	0.1778	0.1130	0.1505	0.2298	0.3616
POP	0.1179	0.0974	0.0695	0.0801	0.1299	0.2266
IMB	0.2330	0.1750	0.1131	0.1500	0.2250	0.3630
VINS	0.2087	0.1686	0.1103	0.1341	0.2176	0.3553
DYN-5	**0.2390**	**0.1796**	**0.1132**	0.1532	**0.2314**	0.3635
DYN-10	0.2256	0.1716	0.1101	0.1453	0.2199	0.3553
DYN-15	0.2372	0.1755	0.1099	**0.1534**	0.2260	0.3516
DYN-20	0.2301	0.1757	0.1076	0.1475	0.2231	0.3426
W-POP	0.1461	0.1172	0.083	0.0993	0.1565	0.2724
W-IMB	0.2518	0.1861	0.1168	0.1629	0.2399	0.3768
W-DYN-5	0.2587	0.1914	0.1206	0.1706	0.2474	0.3910
W-DYN-10	0.2669	**0.1987**	**0.1235**	0.1749	0.2569	**0.4006**
W-DYN-15	**0.2707**	0.1985	0.1212	**0.1759**	**0.2574**	0.3915
W-DYN-20	0.2653	0.1963	0.1201	0.1727	0.2539	0.3876

Table 4.2: Precision and recall scores for cutoffs at 5, 10 and 25 on the LastFm dataset.

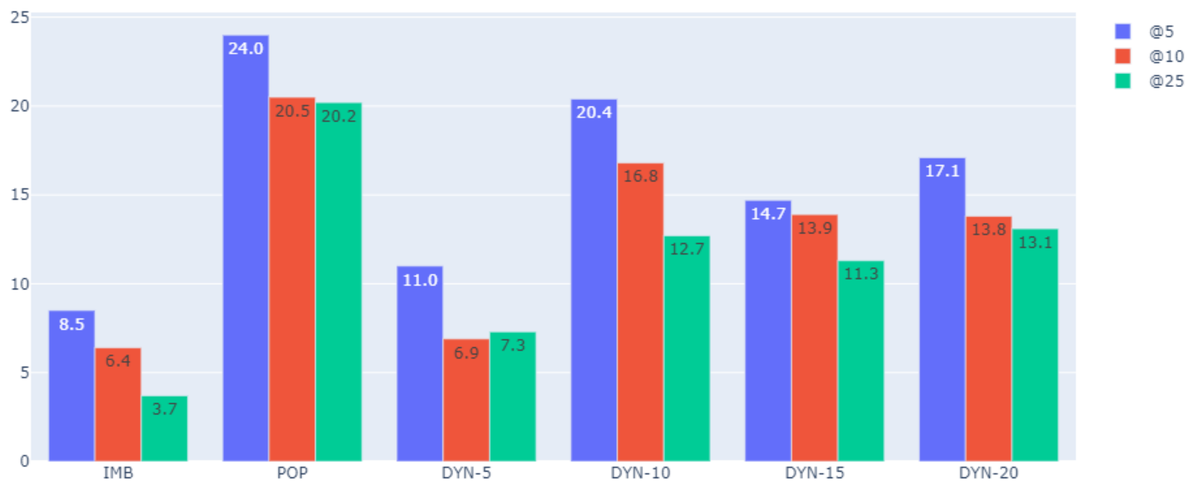


Figure 4.1.2: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the recall scores on LastFm.

Sampling	Prec@5	Prec@10	Prec@25	Rec@5	Rec@10	Rec@25
UNI	0.2327	0.1952	0.1485	0.0888	0.1442	0.2604
POP	0.1203	0.1052	0.0830	0.0505	0.0868	0.1601
IMB	0.2717	0.2263	0.1661	0.1056	0.1689	0.2907
VINS	0.1907	0.1619	0.1255	0.0749	0.1211	0.2186
DYN-5	**0.2907**	**0.2434**	**0.1781**	**0.1098**	**0.1771**	**0.3011**
DYN-10	0.2894	0.2398	0.1753	0.1091	0.1724	0.2958
DYN-15	0.2733	0.2269	0.1643	0.1047	0.1672	0.2845
DYN-20	0.2534	0.2112	0.1526	0.0993	0.1591	0.2706
W-POP	0.1242	0.1118	0.0922	0.0532	0.0934	0.1809
W-IMB	0.2875	0.2411	0.1765	0.1078	0.1730	0.3017
W-DYN-5	**0.3046**	**0.2539**	**0.1843**	0.1151	0.1822	0.3105
W-DYN-10	0.3033	0.2516	0.1814	0.1170	0.1853	**0.3142**
W-DYN-15	0.3045	0.2503	0.1787	**0.1186**	**0.1881**	0.3140
W-DYN-20	0.2973	0.2409	0.1709	0.1176	0.1831	0.3047

Table 4.3: Precision and recall scores for cutoffs at 5, 10 and 25 on the Movielens1M dataset.

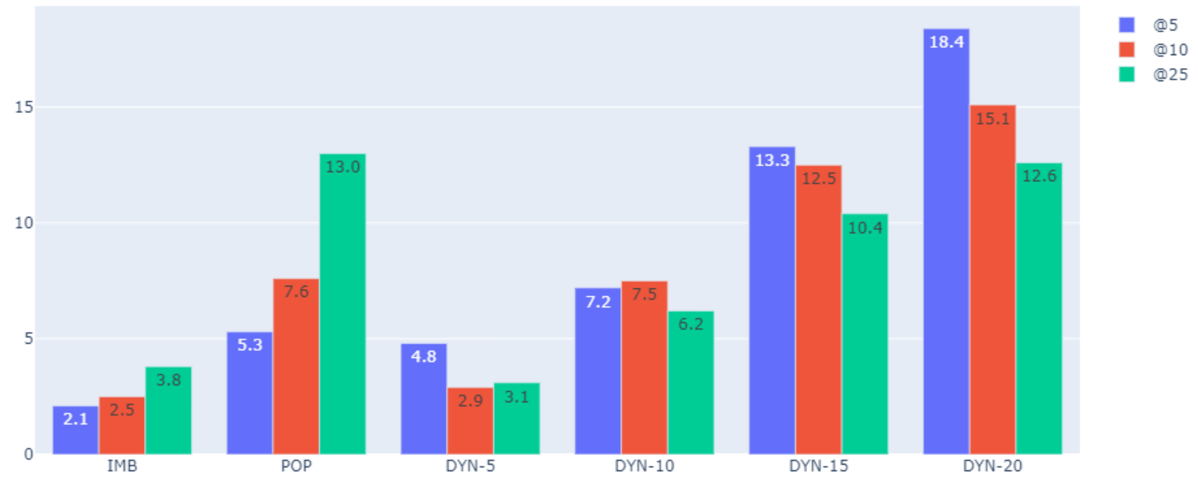


Figure 4.1.3: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the recall scores on Movielens1M.

Sampling	Prec@5	Prec@10	Prec@25	Rec@5	Rec@10	Rec@25
UNI	0.1641	0.1219	0.0751	0.2381	0.3437	0.4994
POP	0.0660	0.0544	0.0409	0.0942	0.1559	0.2919
IMB	0.1672	0.1243	0.0760	0.2466	0.3503	0.5102
VINS	0.1676	0.1252	0.0763	0.2518	0.3608	0.5183
DYN-5	0.1711	0.1251	0.0769	0.2526	0.3539	0.5123
DYN-10	**0.1743**	**0.1284**	**0.0771**	**0.2575**	**0.3652**	**0.5209**
DYN-15	0.1702	0.1250	0.0753	0.2493	0.3539	0.5056
DYN-20	0.1703	0.1241	0.0745	0.2520	0.3552	0.5066
W-POP	0.1089	0.0856	0.0573	0.1560	0.2423	0.3912
W-IMB	0.1725	0.1286	0.0789	0.2522	0.3594	0.5246
W-DYN-5	0.1759	0.1311	0.0801	0.2566	0.3657	0.5343
W-DYN-10	0.1810	0.1332	0.0803	0.2667	0.3736	**0.5388**
W-DYN-15	0.1780	0.1315	0.0797	0.2603	0.3661	0.5287
W-DYN-20	**0.1843**	**0.1345**	**0.0805**	**0.2670**	**0.3746**	0.5342

Table 4.4: Precision and recall scores for cutoffs at 5, 10 and 25 on the Yahoo dataset.

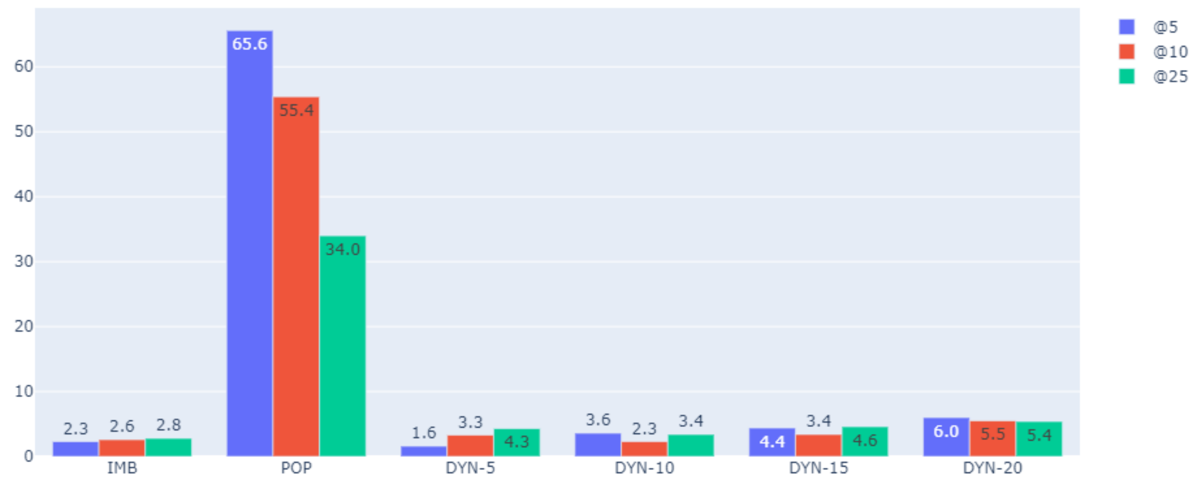


Figure 4.1.4: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the recall scores on Yahoo Movies.

Tables 4.1, 4.2, 4.3 and 4.4 contain the full results of the BPR sampling algorithms for the precision and recall metrics (see Section 3.1.1) with different cutoffs (5, 10 and 25) on the four datasets taken into consideration.
Looking at the tables, we can observe that in each dataset, even if with different impacts, our proposed weighting strategy is able to **always improve** the classification accuracy of the base BPR sampling algorithms. Among the existing samplers, the one that performs better is generally DYN, while POP is by far the worst technique, as expected. In DYN, we can see that small groups (DYN-5) seem to be able to obtain the best results in the non-weighted versions and bigger groups (DYN-10/DYN-15) excel in the weighted version instead, taking advantage of our confidence weighting. One surprising observation that we can make is that the uniform sampler, except for Movielens1M, is able to achieve quite good accuracy even compared to the other more computationally expensive sampling methods.

BookCrossing The BookCrossing dataset (Table 4.1 and Figure 4.1.1) is the one that most benefits from the sample weighting. The improvements on the recall (Figure 4.1.1) are all above 13%, and interestingly they are even larger on the precision; for example, at 5 cutoffs the precision improvement of W-DYN-5 is 19% with respect to the 13% on the recall. Looking at DYN, we can see that DYN-5 is the best overall and DYN-10, DYN-15 and DYN-20 achieve very similar results. As for the non DYN samplers, UNI performs really well, even topping DYN-20. IMB and VINS results seem very closely related, with VINS doing slightly better with larger cutoffs. In general, except the POP sampler, all the baselines are concentrated in a very small range of scores; for example, for the precision with cutoff 5, the maximum base score (0.0388 by DNY-5) is just roughly 6% higher than the lowest score (0.0364 by VINS).

LastFm In LastFm (Table 4.2 and Figure 4.1.2) we can spot similar trends and observe that the improvement of our strategy is really effective. We can note that the IMB sampler accuracy is good compared to the other baselines (it is the second strongest baseline in precision and recall at 25), but the confidence weighting effect is limited with respect to the other sampling techniques. A great difference between LastFm and BookCrossing is that VINS is not able to achieve decent classification scores, in fact the only baseline that performs worse is POP. For what concerns UNI, we can see that it is able to achieve very similar precision and recall scores to the DYN as in BookCrossing and it is even the second strongest base sampling in both the metrics with 10 cutoffs. As we can see from Figure 4.1.2, our proposed weighting in this case brings the best benefits for the metrics evaluated with cutoff at 5.

Movielens1M Table 4.3 shows the classification accuracy on Movielens1M. At first glance, we can immediately identify a considerable difference between this and the other two previous datasets: while in BookCrossing and LastFm the group size does not seem to change much, in Movielens1M larger groups perform significantly worse. More in details, DYN-5 performs better than DYN-20 on the precision by ca. 15% and on the recall by ca. 11%. We are able to notice a similar trend in the VINS performance, which practically acts as a variation of DYN sampler with group size of 64, whereas the group is composed by negative instances pre-sampled by IMB; we can see that IMB performs even 42% better on *Prec@5* with respect to VINS. Moreover, while in the other dataset the IMB results are pretty close to the UNI's one, in Movielens1M IMB does even 16% better on *Prec@5* and 19% on *Rec@5*. As for the relative improvement (Figure 4.1.3) brought by confidence weighting, the effect is not as evident as in the previous two dataset but it is still visible. In particular, the item similarity confidence weighting is very beneficial for larger group sizes in DYN, so that W-DYN-10 and W-DYN-15 are the best overall performing algorithms in the recall.

Yahoo Movies Finally, Table 4.4 contains the precision and recall scores for Yahoo Movies. Interestingly, this is the dataset in which larger groups of DYN perform better than DYN-5, which seemed to be the best non-weighted sampler in the other datasets. In fact, DYN-10 is the best on all the metrics while the W-DYN-20 is the best overall. Similarly to the previous reasoning, the fact that larger group sizes in DYN perform better also reflects in the very good classification accuracy achieved by VINS which is the second best baselines for *Rec@10* and *Rec@25*. About the confidence weighting, Figure 4.1.4 shows that the improvement of our strategy is the smallest among all the datasets, ranging between 1.6% to 6% for all the samplings except for the POP, which is greatly improved. In particular, on *Rec@5* the POP score is raised by over 65% and this could be related to the fact that, as we pointed out in Section 3.3.4, the long tail phenomenon is very accentuated in Yahoo Movies.

4.1.2 Ranking metrics

As for classification results, in the next four pages we collected all the results of the MAP and NDCG on the four datasets. The plots and the tables are to be read exactly as before.

Sampling	**MAP@5**	**MAP@10**	**MAP@25**	**NDCG@5**	**NDCG@10**	**NDCG@25**
UNI	0.0458	0.0465	0.0491	0.0594	0.0687	0.0820
POP	0.0309	0.0320	0.0340	0.0405	0.0473	0.0566
IMB	0.0438	0.0451	0.0474	0.0578	0.0671	0.0794
VINS	0.0416	0.0433	0.0458	0.0556	0.0658	0.0786
DYN-5	**0.0482**	**0.0498**	**0.0528**	**0.0625**	**0.0737**	**0.0881**
DYN-10	<u>0.0476</u>	<u>0.0487</u>	0.0507	<u>0.0616</u>	<u>0.0723</u>	<u>0.0835</u>
DYN-15	0.0462	0.0474	<u>0.0498</u>	0.0599	0.0698	0.0820
DYN-20	0.0458	0.0467	0.0489	0.0598	0.0688	0.0800
W-POP	0.0328	0.0350	0.0377	0.0457	0.0560	0.0689
W-IMB	0.0491	0.0507	0.0536	0.0645	0.0770	0.0923
W-DYN5	0.0552	0.0567	<u>0.0600</u>	0.0707	0.0841	<u>0.1006</u>
W-DYN-10	**0.0577**	**0.0588**	**0.0617**	**0.0745**	**0.0867**	**0.1016**
W-DYN-15	<u>0.0559</u>	<u>0.0571</u>	0.0599	<u>0.0717</u>	<u>0.0851</u>	0.1001
W-DYN-20	0.0551	0.0560	0.0587	0.0714	0.0825	0.0960

Table 4.5: MAP and NDCG scores for cutoffs at 5, 10 and 25 for BookCrossing.

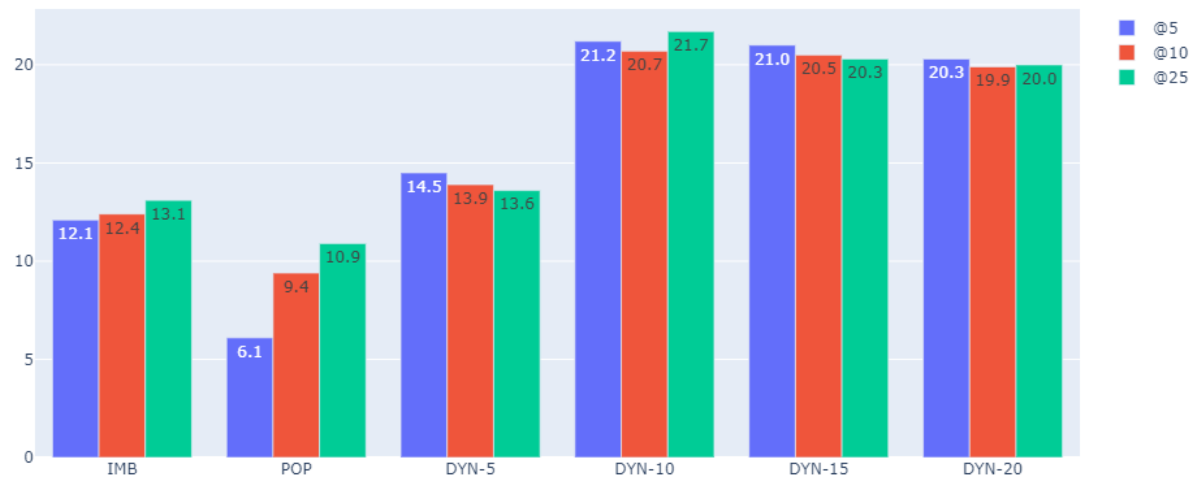


Figure 4.1.5: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the MAP scores on BookCrossing.

Sampling	**MAP@5**	**MAP@10**	**MAP@25**	**NDCG@5**	**NDCG@10**	**NDCG@25**
UNI	0.1751	0.1397	0.1587	0.1927	0.2426	0.3061
POP	0.0777	0.0646	0.0764	0.0996	0.1311	0.1774
IMB	0.1763	0.1390	0.1590	0.1940	0.2416	0.3073
VINS	0.1454	0.1207	0.1418	0.1678	0.2203	0.2863
DYN-5	0.1792	**0.1422**	**0.1613**	0.1972	**0.2462**	**0.3097**
DYN-10	0.1712	0.1361	0.1547	0.1879	0.2353	0.2995
DYN-15	**0.1810**	0.1421	0.1597	**0.1977**	0.2435	0.3038
DYN-20	0.1764	0.1402	0.1571	0.1926	0.2410	0.2984
W-POP	0.0947	0.0789	0.0938	0.1209	0.1567	0.2117
W-IMB	0.1941	0.1518	0.1712	0.2101	0.2588	0.3237
W-DYN5	0.2017	0.1597	0.1801	0.2201	0.2692	0.3368
W-DYN-10	0.207	0.1653	0.1865	0.2254	0.2776	**0.3455**
W-DYN-15	**0.2131**	**0.1684**	**0.1878**	**0.2284**	**0.2798**	0.3436
W-DYN-20	0.2079	0.1646	0.1840	0.2234	0.2749	0.3384

Table 4.6: MAP and NDCG scores for cutoffs at 5, 10 and 25 for LastFm.

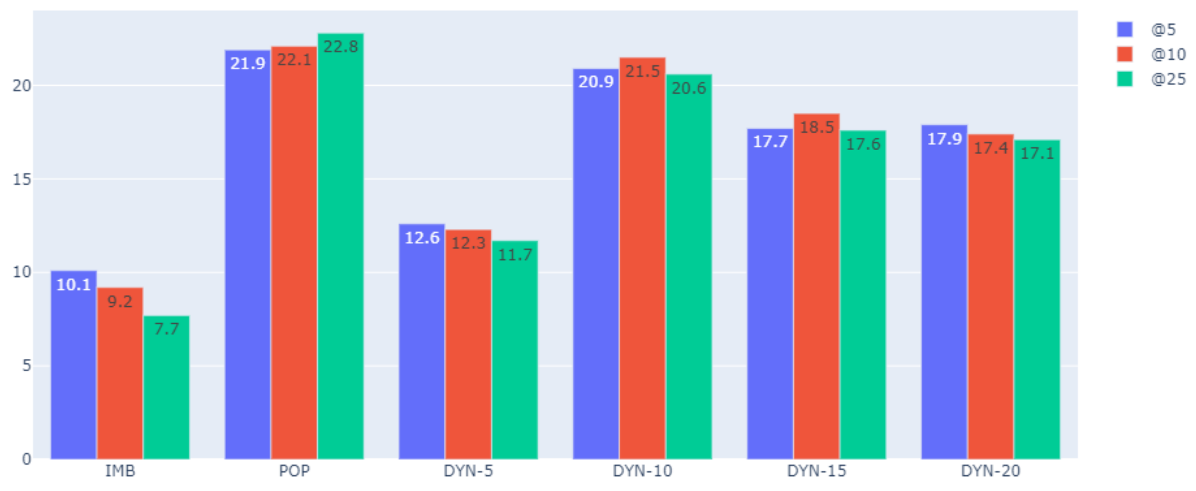


Figure 4.1.6: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the MAP scores on LastFm.

Sampling	**MAP@5**	**MAP@10**	**MAP@25**	**NDCG@5**	**NDCG@10**	**NDCG@25**
UNI	0.1656	0.1282	0.1111	0.1224	0.1623	0.2268
POP	0.0730	0.0570	0.0513	0.065	0.0899	0.1295
IMB	0.2019	0.1584	0.1354	0.1442	0.1901	0.2586
VINS	0.1312	0.1012	0.089	0.1033	0.1368	0.1919
DYN-5	**0.2232**	**0.1768**	**0.1507**	**0.1541**	**0.2030**	**0.2739**
DYN-10	0.2214	0.1734	0.1464	0.1520	0.1986	0.2690
DYN-15	0.2061	0.1604	0.1357	0.1454	0.1910	0.2574
DYN-20	0.1863	0.1447	0.1234	0.1369	0.1802	0.2427
W-POP	0.0768	0.0611	0.0574	0.0688	0.0964	0.1430
W-IMB	0.2198	0.1745	0.1475	0.1512	0.199	0.2710
W-DYN5	0.2351	**0.1862**	**0.1571**	0.1599	0.2092	0.2821
W-DYN-10	0.2349	0.1852	0.1570	0.1623	0.2124	0.2849
W-DYN-15	**0.2364**	0.1855	0.1564	**0.1646**	**0.2143**	**0.2853**
W-DYN-20	0.2282	0.1763	0.1487	0.1619	0.2091	0.2773

Table 4.7: MAP and NDCG scores for cutoffs at 5, 10 and 25 for Movielens1M.

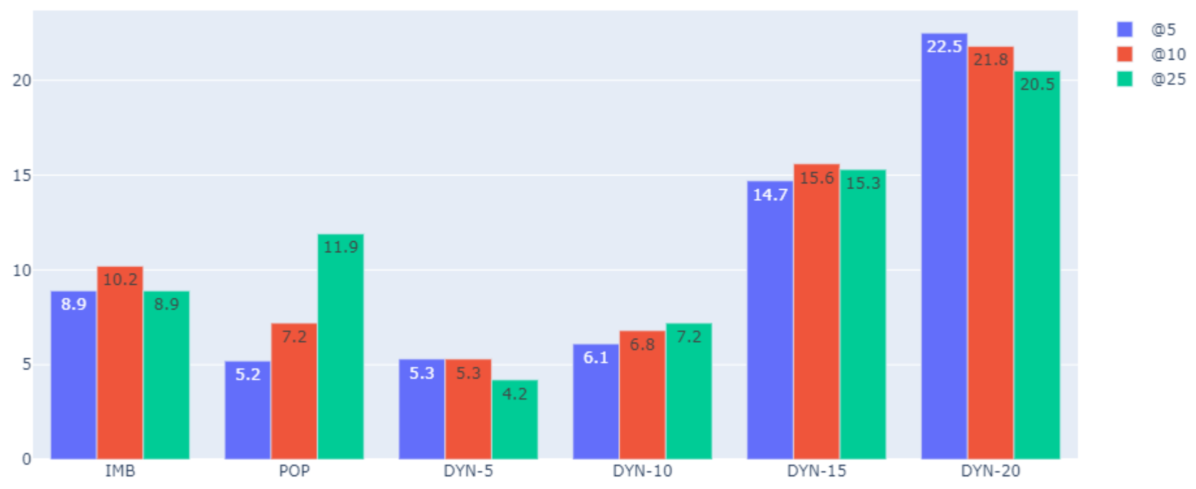


Figure 4.1.7: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the MAP scores on Movielens1M.

Sampling	MAP@5	MAP@10	MAP@25	NDCG@5	NDCG@10	NDCG@25
UNI	0.1770	0.1868	0.2039	0.2233	0.2721	0.3286
POP	0.0612	0.0676	0.0812	0.0841	0.1119	0.1584
IMB	0.1833	0.1940	0.2120	0.2310	0.2810	0.3386
VINS	0.1845	0.1976	0.2165	0.2358	0.2868	0.3438
DYN-5	0.1895	0.1992	0.2173	0.2384	0.2861	0.3435
DYN-10	**0.1946**	**0.2058**	**0.2238**	**0.2450**	**0.2952**	**0.3513**
DYN-15	0.1892	0.1994	0.2165	0.2375	0.2863	0.3413
DYN-20	0.1896	0.1999	0.2176	0.2393	0.2873	0.3421
W-POP	0.1085	0.1171	0.1321	0.1436	0.1830	0.2355
W-IMB	0.1874	0.1978	0.2164	0.2360	0.2866	0.3463
W-DYN5	0.1959	0.2070	0.2262	0.2446	0.2963	0.3571
W-DYN-10	0.2017	0.2119	0.2310	0.2522	0.3029	0.3623
W-DYN-15	0.1960	0.2065	0.2248	0.2454	0.2954	0.3543
W-DYN-20	**0.2063**	**0.2158**	**0.2335**	**0.2553**	**0.3060**	**0.3639**

Table 4.8: MAP and NDCG scores for cutoffs at 5, 10 and 25 for Yahoo Movies.

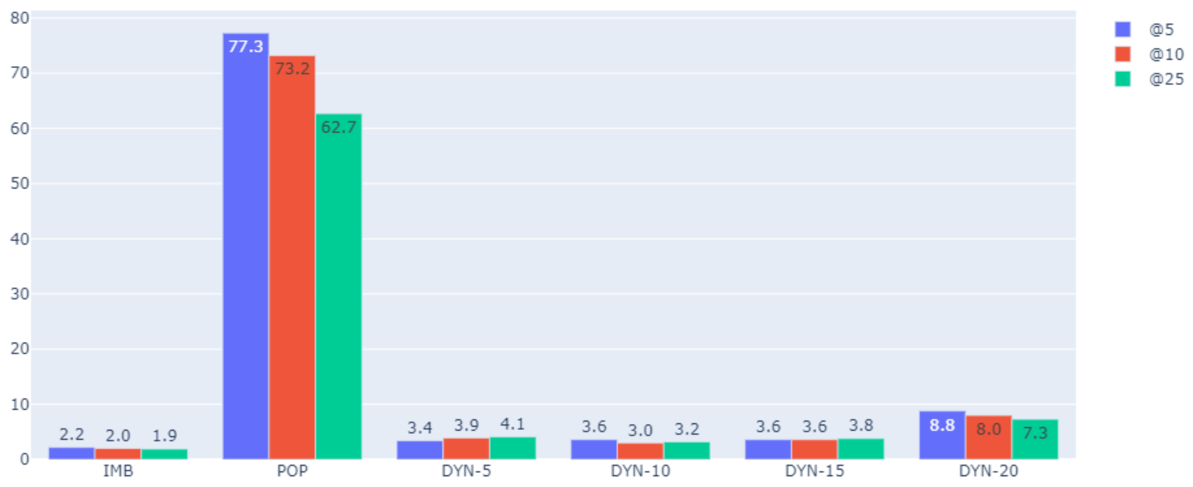


Figure 4.1.8: Bar plot that shows the percentage of gain brought by the similarity sample weighting for the MAP scores on Yahoo Movies.

Tables 4.5, 4.6, 4.7 and 4.8 show the ranking accuracy of the sampling methods on each one of the four datasets, while Figures 4.1.5, 4.1.6, 4.1.7 and 4.1.8 display the percentage of improvement due to our proposed technique. The metrics used for the evaluation are MAP and NDCG (described in Section 3.1.2). The comments on the ranking accuracy are a bit shorted because most of the things we pointed out for the precision and recall results also apply for MAP and NDCG.

BookCrossing In BookCrossing (Table 4.5) DYN-5 is the best non-weighted algorithm in both MAP and NDCG for all the considered cutoffs, whereas the W-DYN with groups of 10 items is the best weighted and overall algorithm in every ranking metric, being able to outperform the non-weighted versions of more than 15%. Comparing these scores with the ones for accuracy (Table 4.3), we can observe that W-DYN-5 is slightly more accurate on the classification metrics for larger cutoffs. In Figure 4.1.5 we can observe that the larger improvements of item similarity weighting are in DYN with groups of 10, 15 and 20. Comparing Figure 4.1.5 with Figure 4.1.1, the benefits appear to be less evident on the MAP and, in particular, W-POP performs way better on the classification metrics.

LastFm On LastFm (Table 4.6), the best baselines are DYN-15 for *MAP@5* and *NDCG@5* and DNY-5 for bigger cutoffs. As for classification accuracy, VINS decreases the IMB accuracy quite significantly, similarly to what we saw on the precision and recall scores. As can be seen from Figure 4.1.6, all the sampling algorithms are greatly improved by the item similarity confidence weighting, especially, as in the other cases, the DYN samplers with 10, 15 and 20 negative candidates and POP.

Movielens1M The ranking results on Movielens1M in Table 4.7 are very similar to the classification ones, yet the improvement of our proposed strategy is even larger (see Figure 4.1.6). DYN-20 does way worse with respect to DYN-5 both on MAP and NDCG, with a difference of more than 20%. DYN-20 is also the strategy that most benefits from the weighting, with an improvement of ca. 20% on the MAP compared to ca. 5% on DYN-5.

Yahoo Movies The same observations we made in the previous section also apply for the ranking results in Yahoo Movies (Table 4.8). In particular, big groups are able to achieve the best ranking accuracy with DYN-10 and W-DYN-20 as the best non-weighted and weighted sampling algorithms on MAP and NDCG for every cutoffs. From Figure 4.1.8 the great improvement brought by the confidence weighting to the POP sampler is again indisputable.

4.1.3 POP vs. IMB

An aspect that can be detected from both the classification and ranking results is that IMB **greatly outperforms** POP, even if they both use popularity of the items to define the negative sampling distribution. Moreover, the POP sampler, which was obtained from the empirical distribution of the items, is the worst performing sampling algorithm in each dataset as also noted in several of the formerly cited works [50, 65, 22] while IMB tends to have similar or better performance with respect to the original uniform negative sampling.

There are two key differences between IMB and POP:

1. the rejection strategy that instantly draws a negative if it has a larger popularity than the selected positive;
2. the fact that IMB draws the negative item among a **small** group of randomly selected candidates (5 in our case).

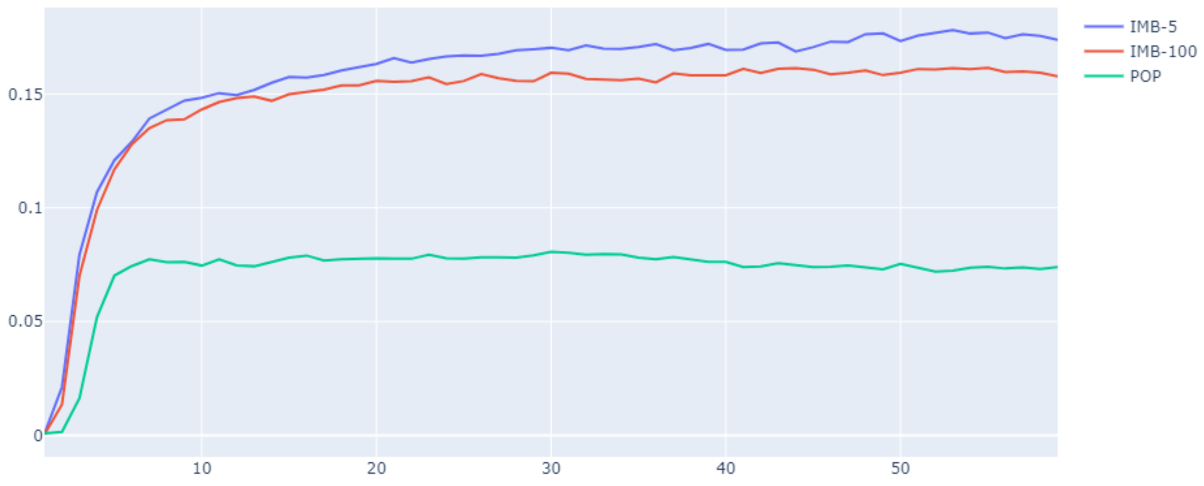


Figure 4.1.9: MAP scores for each epoch of the IMB sampler with groups of 5 (IMB-5) and 100 (IMB-100) on LastFm.

These two cases have the effect of pushing IMB towards the UNI with the

advantage of being able to still select popular items. In fact, we can observe how the rejection strategy accepts the first item with popularity greater than the positive, thus it does not always accept the most popular negative but just the first one that appears when looping in the group that meets the rejection condition and, since the items in the group are extracted uniformly, the ordering is random. This way, since the candidates are uniformly drawn and the item popularity distribution follows the long tail model, most of the times the negative sampled item will not be in the "small head" (see Figure 1.6.1 for the long tail model).
Increasing the size of the randomly selected candidates obviously increases the probability that the selected negative item is among the most popular items in the dataset, thus decreasing the IMB performances towards the POP sampler as can be seen in Figure 4.1.9, which shows the performance of IMB with groups of 5 and groups of 100. Despite being able to notice that accuracy is lower for IMB-100, the rejection strategy is still able to strongly limit this effect.

4.1.4 Cutoff vs. DYN Group Size

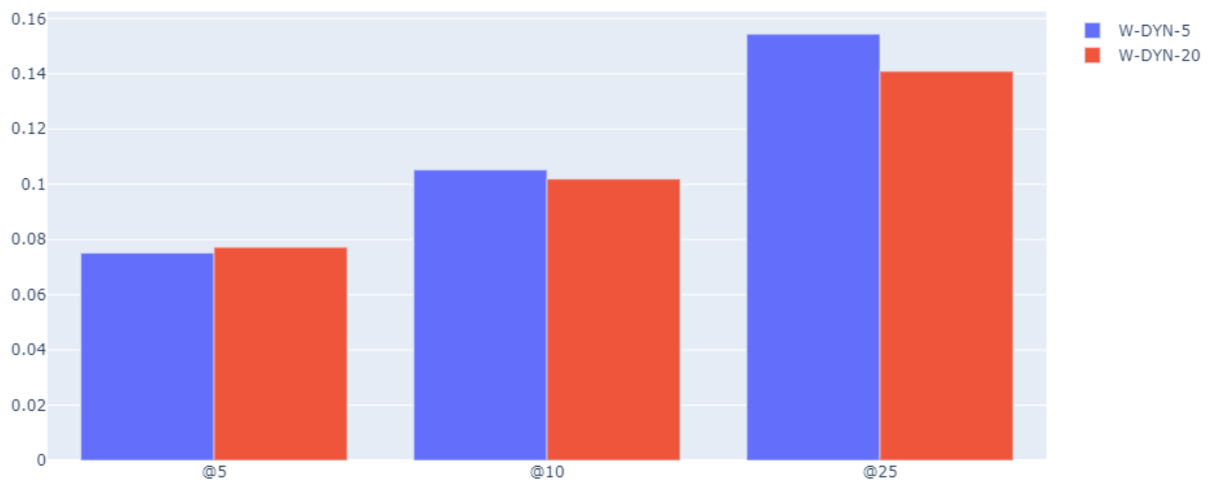


Figure 4.1.10: Difference between recall scores of the weighted DYN-5 and DYN-20 at different cutoffs on BookCrossing.

A curious point that we want to emphasize is that the sampling performances of DYN seem to slightly change according to the different **group size** for each cutoff, both for the weighted and their non-weighted counterparts. More in detail, bigger groups seem to perform somewhat better with respect to the others on the metrics with smaller cutoffs, while smaller groups seem stronger with bigger cutoffs. This appears clear from Figure 4.1.10, which shows the recall scores for BookCrossing for W-DYN-5 and W-DYN-20 (see Table 4.1); while with cutoff 5 the best one is the W-DYN-20, as the cutoff increases the W-DYN-20 performs way worse with respect to the W-DYN-5. This difference is gradual from 5 to 20, with groups of 10 and 15 generally performing well on both 5, 10 and 25 cutoffs. Even when DYN-5 or DYN-20 are the best overall, the difference in performance with 5, 10 and 25 cutoff seems to sometimes follow this trend, both for the classification and ranking metrics, even if there are some exceptions.

Interestingly, even if we compared VINS and DYN highlighting the similarity between the two approaches, this fact does not apply to VINS, probably due to the first phase of sampling through IMB.

4.2 Future Positive Rate

Aside from the classification and ranking metrics, we are also interested in understanding why some samplings perform better than others and why our confidence weighting is able to bring greater benefits to certain samplings.

We define the **Future Positive Rate** (FPR) in the context of the BPR sampling as the percentage of negative items sampled during the training that appear as positive items in the test set. This percentage gives an idea of the difficulty of the negative instances drawn by the sampler: the higher the FPR, the harder it is for the algorithm to distinguish between negative and positive items. Since future positive items should generally be similar to the items in the user profile, we expect this quantity to be correlated to the improvements that our weighting strategy can bring. In fact, the higher the Future Positive Rate for a certain sampler is, the more evident the benefits of the item similarity based confidence weighting should be since it reduces the penalization of the items that are likely to become future positives according to the item similarity.
Let's compare the FPR for the various analyzed samplings, with a couple examples on LastFm and Yahoo Movies for simplicity, but the very same considerations can also be applied in the other two datasets without loss of generality.

4.2.1 Example 1: POP improvement

Figure 4.2.1 shows the FPR for each epoch for the UNI, POP and IMB sampler on LastFm; as expected, since the samplers are static, the FPR is almost a line and does not change during the epochs. The value of the FPR has been obtained, for each epoch, as the percentage of negative items that are then positive in the test set, averaged for all the users. As we can see, IMB selects more than *twice* the number of future positives with respect to UNI, while the POP almost *six* times with respect to UNI; this may be surprising at first glance since both the samplers use the item popularity, but the cause is related to what we explained in Section 4.1.3 about both the limited number of randomly chosen candidates taken into consideration and the rejection strategy of the IMB technique. Comparing this figure with 4.1.6, we can observe that the improvement brought by the confidence weighting is more than double for the POP with respect to the IMB sampling. That is because, since POP samples more future positives as negative items, reducing the importance of the items with a high confidence of being positive has a greater impact.

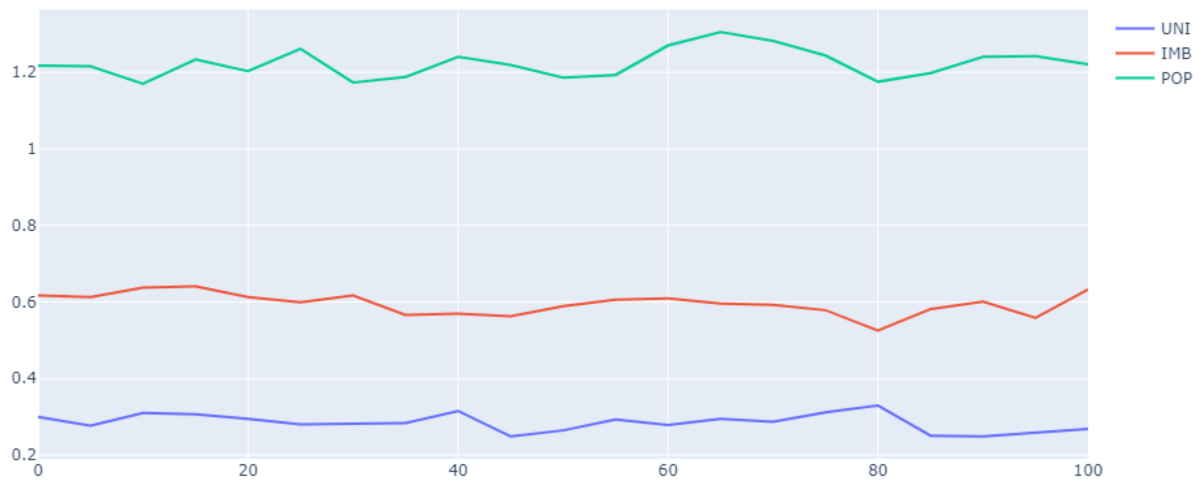


Figure 4.2.1: FPR per epoch on the LastFm dataset.

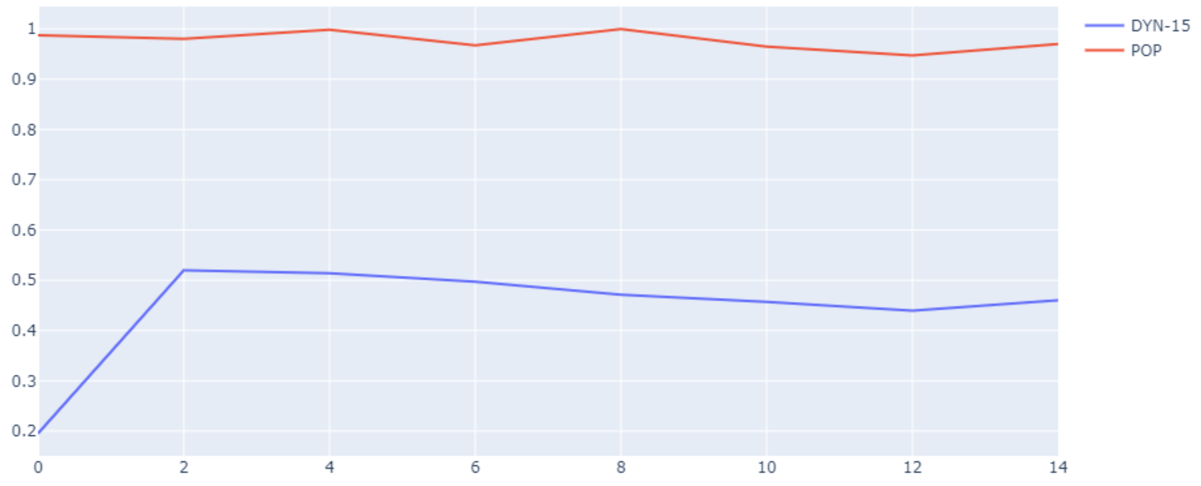


Figure 4.2.2: FPR for the POP and DYN-20 samplers on Yahoo Movies.

A similar consideration about POP can be made observing the incredible impact that confidence weighting has on the Yahoo Movies dataset (see Figure 4.1.8), boosting the POP strategy by more than 75% on *MAP@5*. As we already noticed, this boost can be explained by the fact that the long tail phenomenon is really accentuated in this dataset. Interestingly, this aspect can also be viewed clearly through the FPR results in Figure 4.2.2, as the POP sampler selects a lot of positive instances, even more (ca. double) than the DYN-20, which is the best performing one. The motivation is again the fact that the proposed confidence weighting is able to lower the scores of those selected future positive items without penalizing them too much.

4.2.2 Example 2: DYN group size

Figures 4.2.3 and 4.2.4 highlight the differences in DYN according to the group size on LastFm and Yahoo Movies. As we can notice, the number of future positive items selected grows with the dimension of the group; this is easily explainable by the fact that increasing the number of candidates among which extracting the one with largest score inevitably grows the probability of selecting one of the items ranked at the top (i.e. one of the items that will likely be recommended, so a possible future positive).

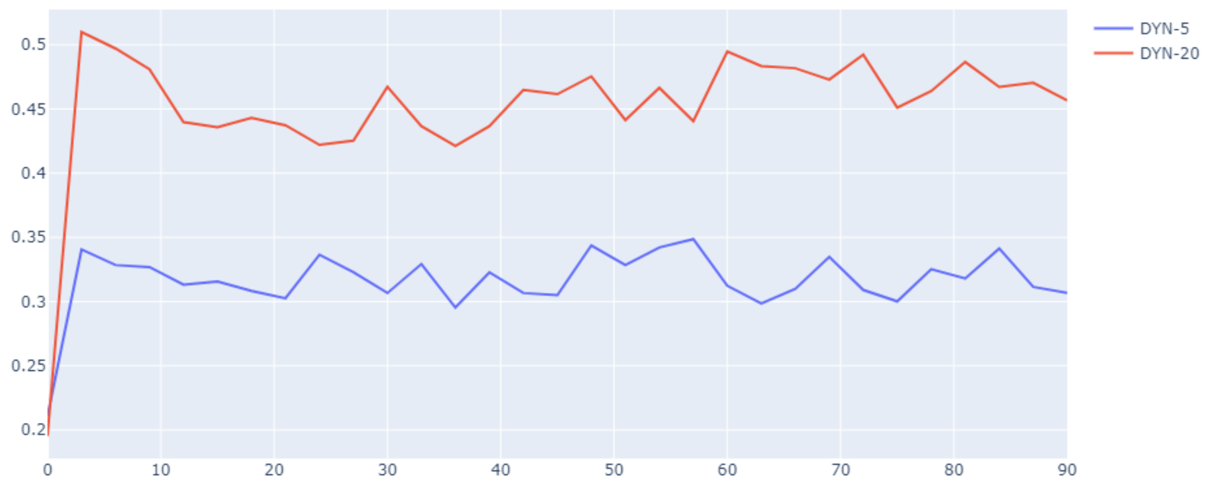


Figure 4.2.3: Percentage of future positive items, for DYN-5 and DYN-20, on Yahoo Movies.

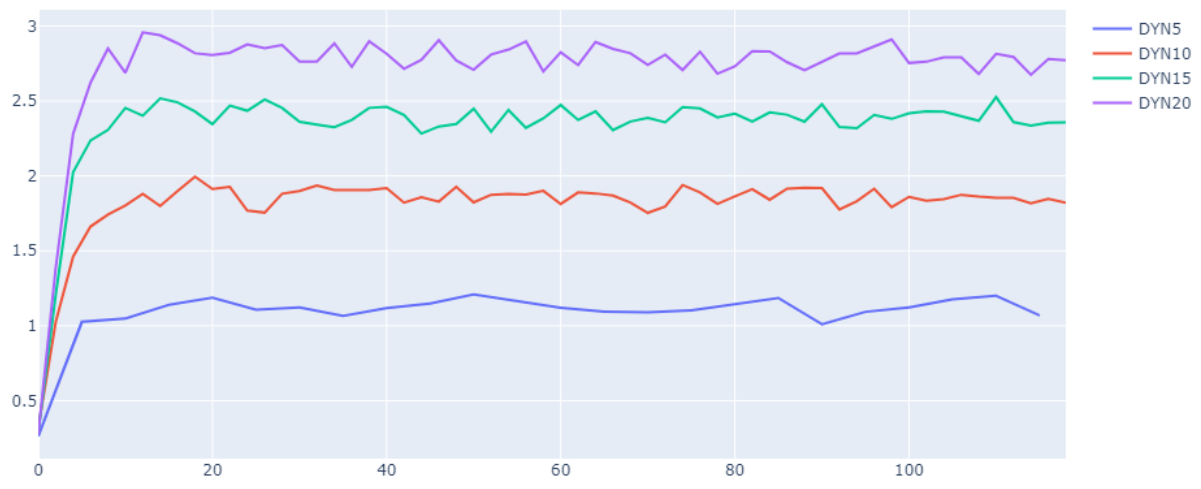


Figure 4.2.4: Percentage of negative items that are future positive items, for different group sizes, on LastFm.

DYN-5 is the one that typically achieves the best scores in the non-weighted version, as can be seen from the classification and ranking results on every dataset except from Yahoo Movies. It is interesting to notice that, usually, larger groups are the ones obtaining greater advantages from the item similarity weights. That is exactly what happens in Yahoo Movies, with DYN-20 having almost double of FPR with respect to DYN-5: it reflects directly on the weighted results with DYN-20 being boosted by our proposed confidence by ca. two times with respect to DYN-5 (Figures 4.1.4 and 4.1.8). A similar trend can be spotted in the Movielens1M results (Table 4.3), in which we can see how DYN-5 strongly outperforms the other DYN variations, but again our strategy turns out to be most helpful for larger sizes of the groups. The reason behind this is the same one we illustrated previously while motivating how POP received a greater improvement compared to IMB, which is that larger groups are able to select more future positives.

In addition to this, from Figure 4.2.5 and 4.2.6 we can observe more in detail that the confidence weighting job is *not* reducing the number of positive instances selected by the sampler, but instead *smoothing* the penalization of the possible future positives.

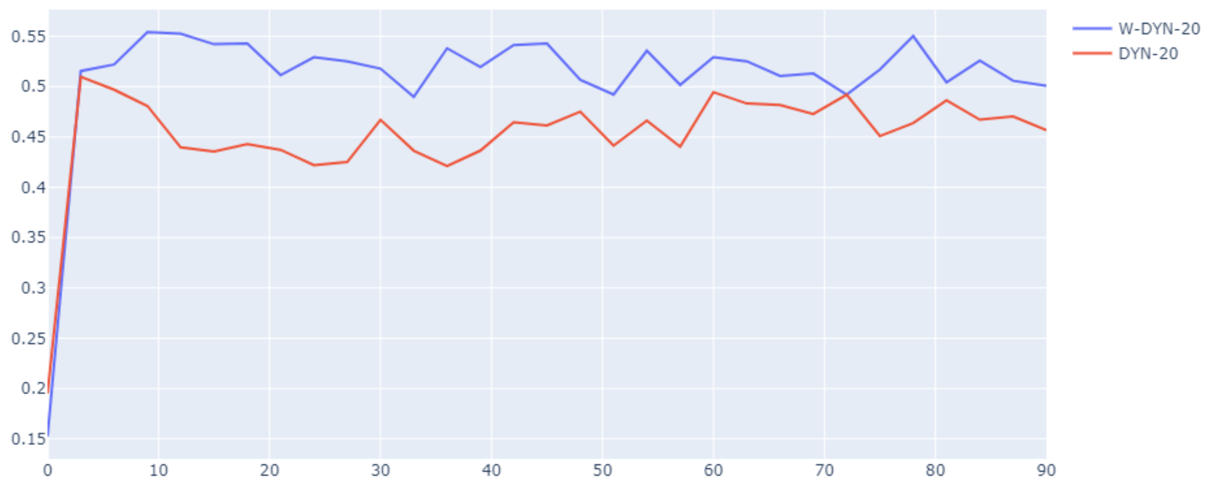


Figure 4.2.5: FPR for DYN-20 and its weighted counterpart on Yahoo Movies.

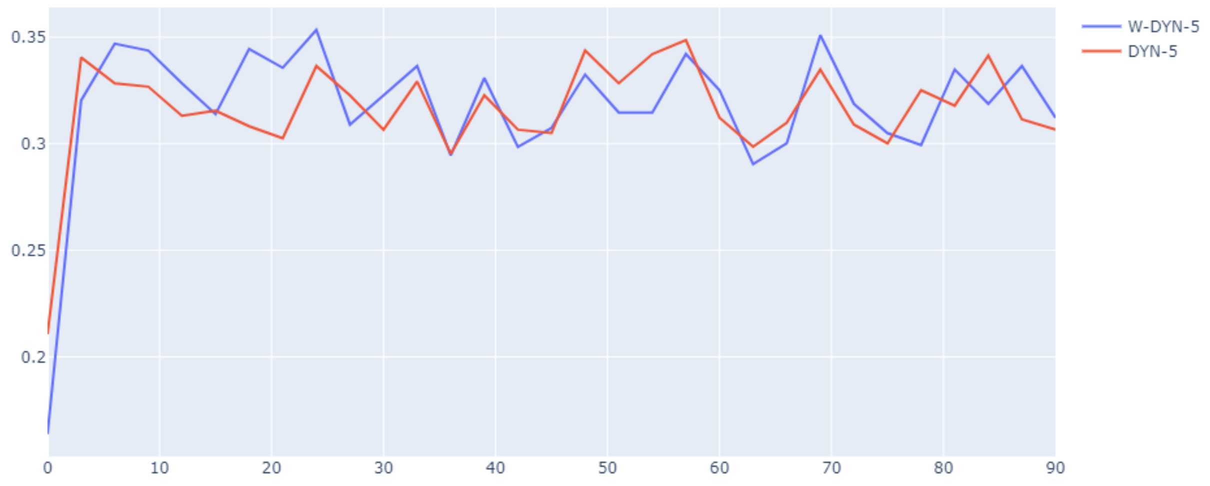


Figure 4.2.6: FPR for DYN-5 and W-DYN-5 on Yahoo Movies.

Indeed, the FPR of the W-DYN-20 is even higher than the one of its non-weighted counterpart and still remains the best overall sampling strategy on the Yahoo dataset. In order for us to give an explanation, we should consider that the model is more accurate and thus produces a better ranking according to the scores, therefore selecting a future positive with larger groups is more probable. The same matter is less evident when comparing DYN-5 and W-DYN-5 (Figure 4.2.6), which can be justified by the limited dimension of the randomly selected candidates that inherently narrows down the probability of finding a very difficult negative (i.e. a possible future positive) just randomly drawing 5 items.

4.2.3 FPR vs Accuracy

We observed that DYN samplers with bigger groups tend to perform worse (without weighting) because they are more prone to select future positive instances and the same reasoning applies to POP and IMB. Despite this being true, one could be tempted to think that high FPR is a bad indicator of the accuracy, which is definitely incorrect. In fact, the plain FPR alone is **not** enough for evaluating the good or bad value of a sampling algorithm.

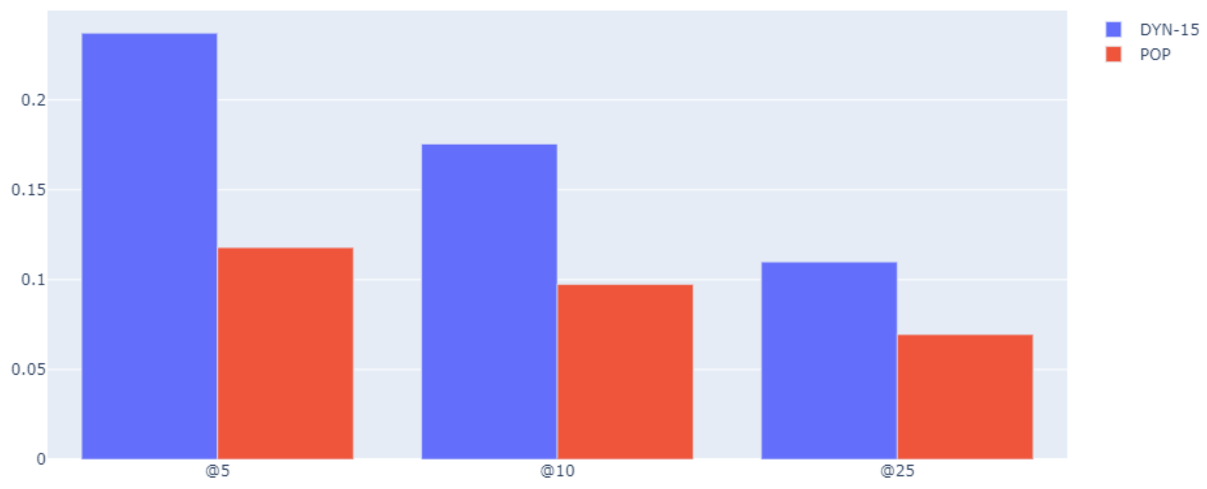


Figure 4.2.7: Comparison between the precision scores on LastFm between POP and DYN-15.

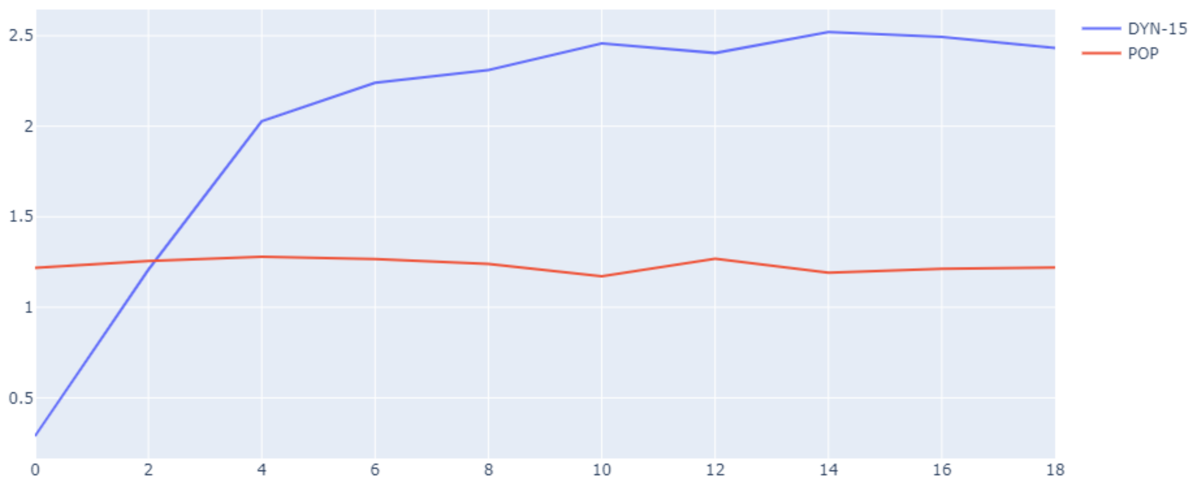


Figure 4.2.8: Comparison between FPR of POP and DYN-15 on LastFm.

We can deduce that from Figures 4.2.7 and 4.2.8, which show the FPR for DYN-15 and POP on LastFm and a comparison on the precision scores. We can observe how, despite DYN-15 having a much higher FPR (ca. 2.4% vs 1.3%), the ranking and classification metrics show how much more accurate DYN-15 is. Consequently, another crucial aspect that we should consider when looking at the FPR is that even if two sampling algorithms have similar FPR, this does not mean that they selected the exact same future positives. As discussed in Section 2.1.1, the POP selects popular items that do not really define the taste of a user, because it prevents them from selecting the niche items that are generally more capable to describe the user's profile, while this should not happen for the DYN sampler.

On this matter, we want to conclude by saying that the FPR is related to a general issue in RS which is that the negative interactions are exactly the ones that need to be recommended in the future. From this perspective, the FPR is an indication of how well a model that extracts the negative samples as recommendations would perform.

4.3 Convergence Speed

Tables 4.9 and 4.10 show the number of epochs necessary for reaching convergence on the four datasets. From these results we can deduce that there is not a single defined trend in the convergence but we can still spot some similarities and differences.

Sampling	Number of Epochs
UNI	50
POP	10
IMB	40
VINS	5
DYN-5	15
DYN-10	10
DYN-15	10
DYN-20	10
W-POP	50
W-IMB	60
W-DYN-5	55
W-DYN-10	55
W-DYN-15	55
W-DYN-20	65

Sampling	Number of Epochs
UNI	100
POP	50
IMB	100
VINS	100
DYN-5	100
DYN-10	85
DYN-15	90
DYN-20	70
W-POP	80
W-IMB	100
W-DYN-5	55
W-DYN-10	40
W-DYN-15	50
W-DYN-20	65

Table 4.9: Number of epochs for reaching convergence on Movielens1M (left) and BookCrossing (right).

Sampling	Number of Epochs
UNI	80
POP	25
IMB	75
VINS	50
DYN-5	55
DYN-10	30
DYN-15	20
DYN-20	15
W-POP	90
W-IMB	60
W-DYN-5	70
W-DYN-10	105
W-DYN-15	90
W-DYN-20	75

Sampling	Number of Epochs
UNI	90
POP	5
IMB	110
VINS	150
DYN-5	75
DYN-10	85
DYN-15	85
DYN-20	130
W-POP	110
W-IMB	100
W-DYN-5	75
W-DYN-10	130
W-DYN-15	90
W-DYN-20	50

Table 4.10: Number of epochs for reaching convergence on LastFm (left) and Yahoo Movies (right).

First of all, we can see that UNI and IMB are the **slowest**. This is not surprising at all for UNI, as we highlighted multiple times the issues with this strategy in selecting informative pairs. About IMB, this is clearly explainable with the previous study on the analysis between IMB, POP and UNI that shows how IMB is much closer to UNI rather than to POP.

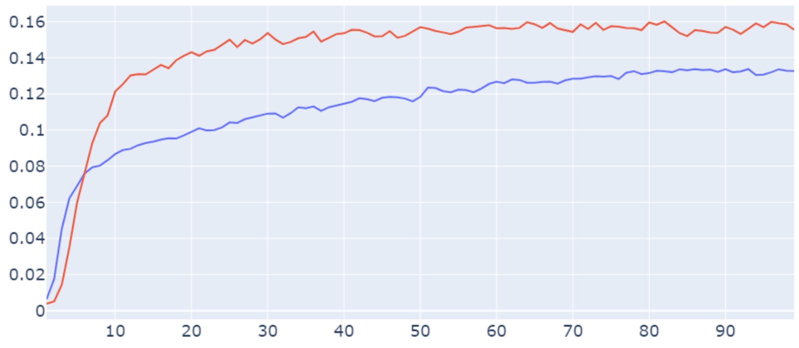
On the contrary, POP is always the **fastest** algorithm. This is again an expected result, as we already pointed out that oversampling popular items improves the learning process in the very first stages but than causes the algorithm to get stuck, making POP the fastest but also worst baseline from an accuracy standpoint. One of the reasons that make POP converge so fast could be related to the FPR, which also explains how DYN always requires (with the exception of Yahoo Movies) a lower number of epochs to converge in case of larger groups. On Yahoo Movies we can observe that this trend is practically inverted and, moreover, we already noticed that this dataset is the one in which large groups perform best, which is visible both in the results of VINS and DYN.

About the impact of our confidence weighting, we can spot two separate trends:

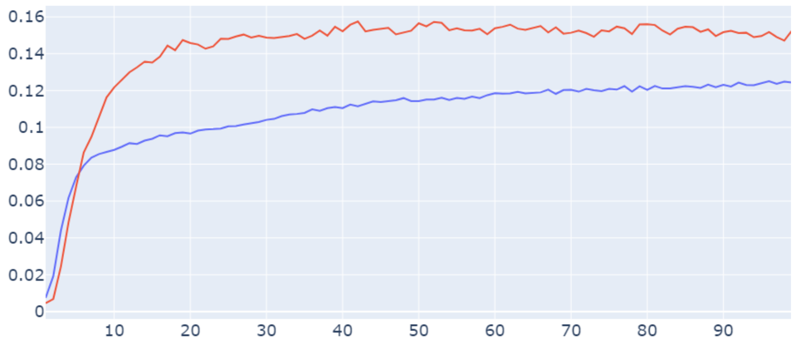
- on Movielens1M and LastFm, the weights severely **increase** the number of epochs;
- on BookCrossing it really **speeds up** the learning.

On Yahoo Movies, we cannot outline a clear trend, with W-DYN-5/15 having roughly the same convergence speed in terms of epochs as DYN-5/15 while DYN-10 is much faster than W-DYN-10 and DYN-20 in much slower than W-DYN-20. In the next pages, Figures 4.3.1 and 4.3.2 show the *Rec@25* results of DYN and W-DYN for every group size on each epoch, on BookCrossing and LastFm. From Figure 4.3.1, we can observe that the convergence is really fast with the weighting on BookCrossing while, on the contrary, we can spot the opposite trend on LastFm (Figure 4.3.2). From Figure 4.3.2 it is evident that, as the FPR seems to grow with the number of epochs until convergence (see Figure 4.2.4), once convergence is reached, accuracy drops by a good portion, especially with larger group sizes. On the other hand, confidence weighting allows the model to continue learning even with a high FPR, that is why (in addition to the fact that small weights reduce the convergence speed) the convergence is a slightly slower and we are not able to see an instant drop in accuracy once convergence is reached.

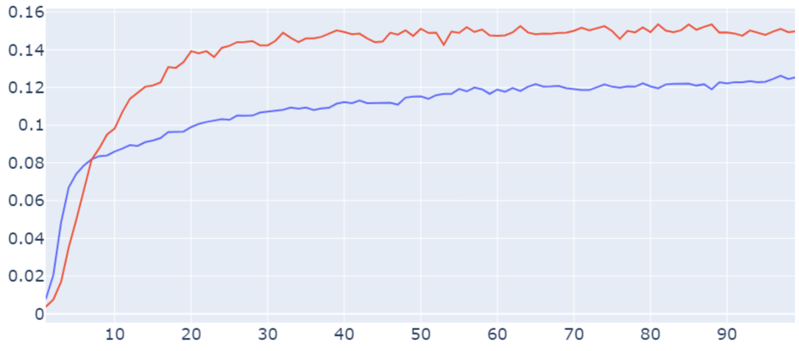
Finally, about the impact of our confidence weighting on the **computational complexity**, we already stated that it only introduces a constant time for retrieving the weight associated to each sample, which is a fixed value. This offers great advantages in terms of ease of computation with respect to [65, 37], especially because this sampling strategy, if combined with a static sampler (e.g. W-IMB), can even improve a more computational expensive dynamic adaptive strategy (non-weighted) such as DYN, as we saw in the LastFm (Tables 4.2 and 4.5) and BookCrossing (Tables 4.1 and 4.5).



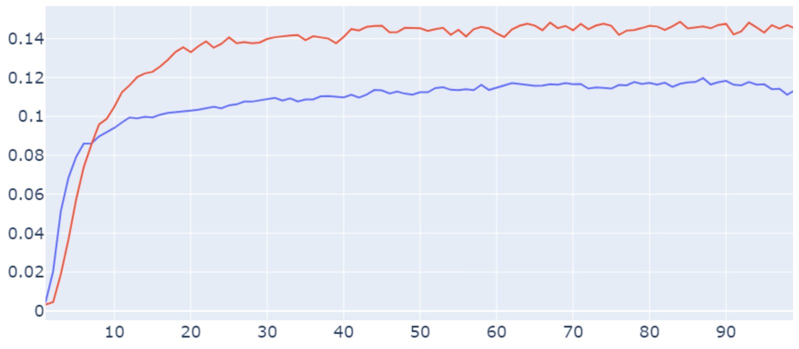
(a) DYN5



(b) DYN10

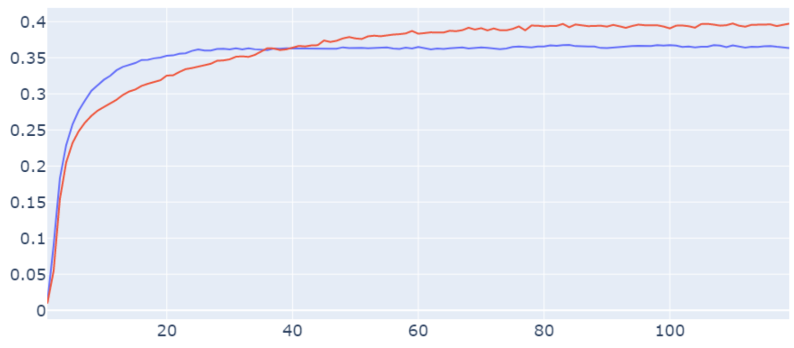


(c) DYN15

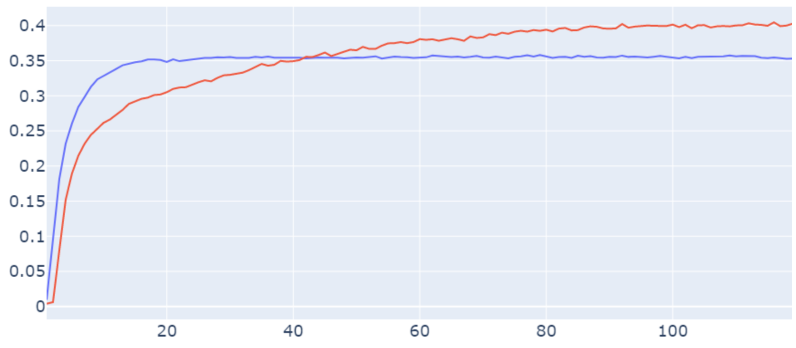


(d) DYN20

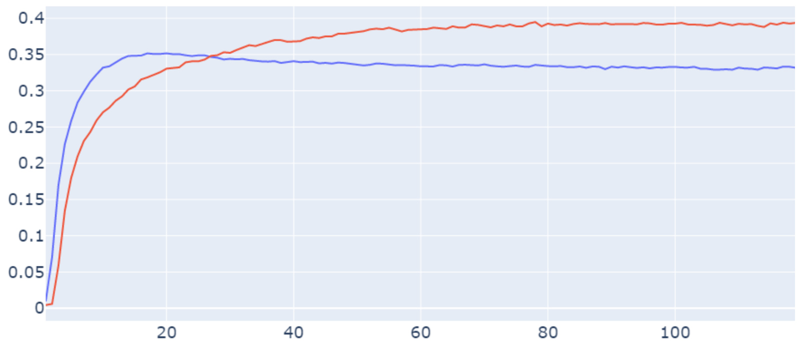
Figure 4.3.1: *Rec@25* for each epoch on the BookCrossing dataset for the DYN samplers; the red ones are the weighted versions, the blue ones are the non weighted standard DYN.



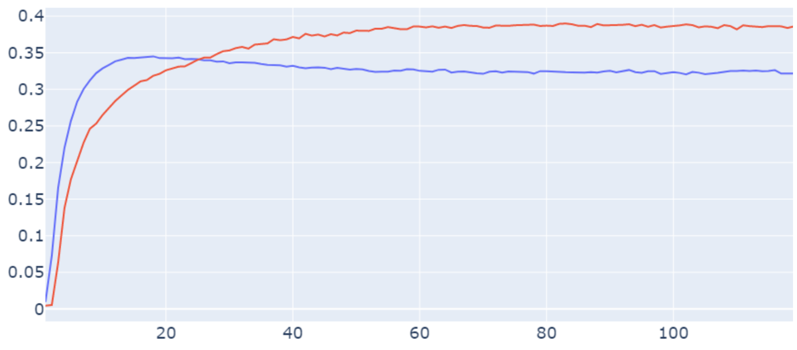
(a) DYN5



(b) DYN10



(c) DYN15



(d) DYN20

Figure 4.3.2: Comparison between DYN and W-DYN in *Rec@25* on LastFm.

Chapter 5

Conclusions

In this thesis, we firstly presented a very detailed review of the state of the art, starting from the high level understanding of what are RS about and which are the main challenges and existing solutions, to then move to Matrix Factorization models and, finally, to the Bayesian Personalized Ranking criterion. We deeply analyzed the original BPR formulation as well as the most recent BPR variant that aim at improving the learning process and employing additional knowledge. We then focused on a very important aspect of the BPR: the negative sampling process. After describing in depth what it means to draw a sample in the context of the BPR learning algorithm, we highlighted the strengths and weaknesses of each technique.
From the review of the state of the art we noticed that the more recent studies move the attention from the selection of the negative instances to sample weighting and, in particular, to the difficulty of a sample. Moreover, the state of the art strategies often tend to be very computationally expensive and need some approximations in order to be usable; with this in mind, we introduced the concept of negative confidence, deriving it directly from the collaborative item-item similarity. This idea stemmed from the assumption that, even though sampling difficult negatives has been proven to have several benefits, samples which are too difficult could decrease the model accuracy because they are likely to become positives in the future; one way to measure this un-

certainty is using the collaborative similarity between the items. We showed how this item similarity based confidence can be utilized directly in the context of the BPR sampling as a sample weight to tune the updates of the different samples without degrading the computational complexity of the overall process.
In the final chapter, we exhibited the experimental results of the BPR sampling on four popular research datasets, by which we were able to show how our proposed weighting strategy managed to improve all the existing techniques in both classification and ranking metrics. Moreover, we explained some of the results using the concept of future positive rate, as the number of selected negative samples that appear as positive feedback in the test set.
The main contribution of this thesis, apart from a very detailed comparative analysis of all the existing sampling techniques, is to introduce the problem of distinguishing between actual negatives and future positives using the collaborative item-item similarity. This technique can be applied to every implicit feedback dataset for all the BPR formulations and can even be used in other losses that, similarly to the BPR, adopt the negative sampling during the learning process. Finally, we want to point out that our proposed confidence makes use of the standard item cosine similarity but the same idea can also be implemented using other collaborative models. Generally speaking, while most of the works focus on using an additional model (e.g. a GAN) for generating the training sample, an interesting application could be to use additional models to weight each sample according to the model scores in a similar way to what we did with the collaborative similarity.

Bibliography

[1] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. “Managing popularity bias in recommender systems with personalized re-ranking”. In: *The thirty-second international flairs conference*. 2019.

[2] Gediminas Adomavicius and Alexander Tuzhilin. “Context-aware recommender systems”. In: *Recommender systems handbook*. Springer, 2011, pp. 217–253.

[3] Charu C Aggarwal et al. *Recommender systems*. Vol. 1. Springer, 2016.

[4] Joeran Beel and Stefan Langer. “A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems”. In: *International conference on theory and practice of digital libraries*. Springer. 2015, pp. 153–168.

[5] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[6] Christopher Burges, Robert Ragno, and Quoc Le. “Learning to rank with nonsmooth cost functions”. In: *Advances in neural information processing systems* 19 (2006), pp. 193–200.

[7] Jiawei Chen et al. “CoSam: An Efficient Collaborative Adaptive Sampler for Recommendation”. In: *ACM Transactions on Information Systems (TOIS)* 39.3 (2021), pp. 1–24.

[8] Antonia Creswell et al. “Generative Adversarial Networks: An Overview”. In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65. DOI: 10.1109/MSP.2017.2765202.

[9] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. “Are we really making much progress? A worrying analysis of recent neural recommendation approaches”. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. 2019, pp. 101–109.

[10] Félix Hernández Del Olmo and Elena Gaudioso. “Evaluation of recommender systems: A new approach”. In: *Expert Systems with Applications* 35.3 (2008), pp. 790–804.

[11] Jingtao Ding et al. “Reinforced Negative Sampling for Recommendation with Exposure Data.” In: *IJCAI*. 2019, pp. 2230–2236.

[12] Zeno Gantner et al. “Personalized ranking for non-uniformly sampled items”. In: *Proceedings of KDD Cup 2011*. PMLR. 2012, pp. 231–247.

[13] Walter R Gilks and Pascal Wild. “Adaptive rejection sampling for Gibbs sampling”. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 41.2 (1992), pp. 337–348.

[14] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[15] Jyotirmoy Gope and Sanjay Kumar Jain. “A survey on solving cold start problem in recommender systems”. In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE. 2017, pp. 133–138.

[16] Quanquan Gu, Jie Zhou, and Chris Ding. “Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs”. In: *Proceedings of the 2010 SIAM international conference on data mining*. SIAM. 2010, pp. 199–210.

[17] Ido Guy. “Social recommender systems”. In: *Recommender systems handbook*. Springer, 2015, pp. 511–543.

[18] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI].

[19] F Maxwell Harper and Joseph A Konstan. “The movielens datasets: History and context”. In: *Acm transactions on interactive intelligent systems (tiis)* 5.4 (2015), pp. 1–19.

[20] Stephen P Harter. “Variations in relevance assessments and the measurement of retrieval effectiveness”. In: *Journal of the American Society for Information Science* 47.1 (1996), pp. 37–49.

[21] Ruining He and Julian McAuley. “VBPR: visual bayesian personalized ranking from implicit feedback”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[22] Xiangnan He et al. *Fast Matrix Factorization for Online Recommendation with Implicit Feedback*. 2017. arXiv: 1708.05024 [cs.IR].

[23] Jonathan L Herlocker et al. “Evaluating collaborative filtering recommender systems”. In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 5–53.

[24] *HetRec '11: Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*. Chicago, Illinois: Association for Computing Machinery, 2011. ISBN: 9781450310277.

[25] Sepp Hochreiter. "The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6.2 (Apr. 1998), 107–116. ISSN: 0218-4885. DOI: 10.1142/S0218488598000094. URL: https://doi.org/10.1142/S0218488598000094.

[26] Cheng-Kang Hsieh et al. "Collaborative metric learning". In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 193–201.

[27] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 263–272. DOI: 10.1109/ICDM.2008.22.

[28] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets". In: *2008 Eighth IEEE International Conference on Data Mining*. Ieee. 2008, pp. 263–272.

[29] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[30] Glen Jeh and Jennifer Widom. "Simrank: a measure of structural-context similarity". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, pp. 538–543.

[31] Santosh Kabbur, Xia Ning, and George Karypis. "Fism: factored item similarity models for top-n recommender systems". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 659–667.

[32] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].

[33] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[34] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization techniques for Recommender Systems". In: (2009). DOI: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5197422.

[35] Artus Krohn-Grimberghe et al. "Multi-relational matrix factorization using bayesian personalized ranking for social network data". In: *Pro-*

ceedings of the fifth ACM international conference on Web search and data mining. 2012, pp. 173–182.

[36] Lukas Lerche and Dietmar Jannach. “Using graded implicit feedback for bayesian personalized ranking”. In: *Proceedings of the 8th ACM Conference on Recommender systems*. 2014, pp. 353–356.

[37] Defu Lian, Qi Liu, and Enhong Chen. “Personalized ranking with importance sampling”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 1093–1103.

[38] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. “Facing the cold start problem in recommender systems”. In: *Expert Systems with Applications* 41.4 (2014), pp. 2065–2073.

[39] Greg Linden, Brent Smith, and Jeremy York. “Amazon. com recommendations: Item-to-item collaborative filtering”. In: *IEEE Internet computing* 7.1 (2003), pp. 76–80.

[40] Babak Loni et al. “Bayesian personalized ranking with multi-channel user feedback”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. 2016, pp. 361–364.

[41] Xia Ning and George Karypis. “Slim: Sparse linear methods for top-n recommender systems”. In: *2011 IEEE 11th International Conference on Data Mining*. IEEE. 2011, pp. 497–506.

[42] *NumPy*. URL: `https://numpy.org/`.

[43] Lawrence Page et al. *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[44] Rong Pan et al. “One-Class Collaborative Filtering”. In: (). DOI: `http://www.rongpan.net/publications/pan-oneclasscf.pdf`.

[45] Weike Pan and Li Chen. “Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering”. In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.

[46] Yoon-Joo Park and Alexander Tuzhilin. “The long tail of recommender systems and how to leverage it”. In: *Proceedings of the 2008 ACM conference on Recommender systems*. 2008, pp. 11–18.

[47] *Python*. URL: `https://www.python.org/`.

[48] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. “Sequence-aware recommender systems”. In: *ACM Computing Surveys (CSUR)* 51.4 (2018), pp. 1–36.

[49] Massimo Quadrana et al. “Personalizing session-based recommendations with hierarchical recurrent neural networks”. In: *Proceedings of*

the Eleventh ACM Conference on Recommender Systems. 2017, pp. 130–137.

[50] Steffen Rendle and Christoph Freudenthaler. "Improving Pairwise Learning for Item Recommendation from Implicit Feedback". In: (2012). DOI: `http://webia.lip6.fr/~gallinar/gallinari/uploads/Teaching/WSDM2014-rendle.pdf`.

[51] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *CoRR* abs/1205.2618 (2012). arXiv: `1205.2618`. URL: `http://arxiv.org/abs/1205.2618`.

[52] Badrul Sarwar et al. "Application of Dimensionality Reduction in Recommender System - A Case Study". In: (2000). DOI: `https://apps.dtic.mil/sti/pdfs/ADA439541.pdf`.

[53] Badrul Sarwar et al. *Application of dimensionality reduction in recommender system-a case study*. Tech. rep. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[54] *SciPy*. URL: `https://numpy.org/`.

[55] *SimilariPy*. URL: `https://pypi.org/project/similaripy/`.

[56] Riku Togashi et al. "Scalable Personalised Item Ranking through Parametric Density Estimation". In: *arXiv preprint arXiv:2105.04769* (2021).

[57] Katrien Verbert et al. "Context-aware recommender systems for learning: a survey and future challenges". In: *IEEE transactions on learning technologies* 5.4 (2012), pp. 318–335.

[58] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. "Singular value decomposition and principal component analysis". In: *A practical approach to microarray data analysis*. Springer, 2003, pp. 91–109.

[59] Jun Wang et al. "Irgan: A minimax game for unifying generative and discriminative information retrieval models". In: *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 2017, pp. 515–524.

[60] Jason Weston, Samy Bengio, and Nicolas Usunier. "Large scale image annotation: learning to rank with joint word-image embeddings". In: *Machine learning* 81.1 (2010), pp. 21–35.

[61] Shiwen Wu et al. *Graph Neural Networks in Recommender Systems: A Survey*. 2021. arXiv: `2011.02260 [cs.IR]`.

[62] Shu Wu et al. "Session-based recommendation with graph neural networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 346–353.

[63] *Yahoo Dataset*. URL: `http://webscope.sandbox.yahoo.com/catalog.php?datatype=r`.

[64] Xiwang Yang et al. "A survey of collaborative filtering based social recommender systems". In: *Computer communications* 41 (2014), pp. 1–10.

[65] Lu Yu et al. "Addressing Class-Imbalance Problem in Personalized Ranking". In: *arXiv preprint arXiv:2005.09272* (2020).

[66] Weinan Zhang et al. "Optimizing top-n collaborative filtering via dynamic negative item sampling". In: *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. 2013, pp. 785–788.

[67] Feng Zhao et al. "SDBPR: Social distance-aware Bayesian personalized ranking for recommendation". In: *Future Generation Computer Systems* 95 (2019), pp. 372–381.

[68] Tong Zhao, Julian McAuley, and Irwin King. "Leveraging social connections to improve personalized ranking for collaborative filtering". In: *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*. 2014, pp. 261–270.