White-Box: On the Prediction of Collaborative Filtering Recommendation Systems' Performance

IULIA PAUN, School of Computing Science
University of Glasgow, Scotland, UK
YASHAR MOSHFEGHI, NeuraSearch Laboratory
University of Strathclyde, Scotland, UK
NIKOS NTARMOS, School of Computing Science
University of Glasgow, Scotland, UK

Collaborative Filtering recommendation algorithms (CF) are a popular solution to the information overload problem, aiding users in the item selection process. Relevant research has long focused on refining and improving these models to produce better (more effective) recommendations, and has converged on a methodology to predict their effectiveness on target datasets by evaluating them on random samples of the latter. However, predicting the efficiency of the solutions – especially with regards to their time- and resource-hungry training phase, whose requirements dwarf those of the prediction/recommendation phase – has received little to no attention in the literature. This paper addresses this gap for a number of representative and highly popular CF models, including algorithms based on matrix factorisation, k-nearest neighbours, co-clustering, and slope one schemes. To this end, we first study the computational complexity of the training phase of said CF models and derive time and space complexity equations. Then, using characteristics of the input and the aforementioned equations, we contribute a methodology for predicting the processing time and memory usage of their training phase. Our contributions further include an adaptive sampling strategy, to address the trade-off between resource usage costs and prediction accuracy, and a framework which quantifies both the efficiency and effectiveness of CF. Finally, a systematic experimental evaluation demonstrates that our method outperforms state-of-the-art regression schemes by a considerable margin, with an overhead that is a small fraction of the overall requirements of CF training.

CCS Concepts: • **Information systems → Recommender systems**; *Evaluation of retrieval results.*

Additional Key Words and Phrases: Recommendation systems, efficiency evaluation, effectiveness evaluation, sampling-based time and memory prediction.

1 INTRODUCTION

Recommendation systems have been extensively used to aid users in the item selection process by producing tailored content in line with the users' tastes and needs [66]. This, in turn, has also impacted the way providers deliver content to the users, as they need to strike a balance between revenue maximisation and the users' satisfaction [7]. An important class of recommendation systems are Collaborative Filtering-based recommendation systems (CF), which recommend items to a user based on similar users' preferences. CF models can rely on explicit feedback datasets, where the users' tastes are captured through ratings given by users to items (e.g., the ratings could be in the range of 1 to 5, where a higher rating value indicates that the user enjoyed the item), or on implicit data (e.g., users interacting with items – watching a movie, reading an article, etc. – without providing explicit feedback).

One of the challenges in building CF engines is the selection of the algorithms to be used. While there is a growing body of literature that focuses on developing novel and better algorithms for attaining users' satisfaction [54, 84] and producing accurate recommendations [69, 86], less attention and efforts have been allocated for studying the prediction of the efficiency and resource consumption of CF models, particularly during their highly expensive training phase. Focusing only on the nominal accuracy of the CF algorithms is a fallacy, as there is an inherent trade-off between the efficiency of the system (training phase resource consumption) and the effectiveness (accuracy) of the recommendations [63, 64].

1.1 Motivation

It is often the case that a one-CF-model-fits-all solution becomes unfeasible due to the dynamic relationship between users and items, and the rate at which new algorithms are proposed in the literature. At the same time, the models need to be periodically retrained to capture the latest user preferences and needs [7]. Companies then have to decide whether and when to retrain their CF models, since the training time and memory usage are often quite high (surely several orders of magnitude higher than the time it takes to produce a recommendation given a trained CF [7, 74]), while high training times/resource utilisation also translate to expensive energy and monetary costs for the content providers [74], and even raise concerns for damage to the environment as a result of increased amounts of CO_2 emissions [76]. These problems are further exacerbated by the ongoing data generation growth, identified by a recent study released by Amazon [74] as one of the key factors which impact the scalability of CF algorithms.

From the point of view of effectiveness, the problem of choosing a model is routinely addressed by drawing random samples from the target dataset, training and evaluating the candidate models on these samples, then using the resulting effectiveness figures as proxies for the effectiveness of the model on the complete dataset [1, 18, 24]. Throughout the paper, we will refer to this approach as the black-box performance evaluation. It is often the case that a model with higher time/space complexity requires higher training time and also a higher rate at which the latter increases as a function of the size of the input. This makes the above sampling-based approach quite appealing, as the loss in accuracy is more than offset by the gain in processing time. However, extending it to the prediction of efficiency (training time, memory usage, etc.) over the complete dataset is far from straightforward. On one hand, the sampling strategies typically used in the relevant literature fail to capture the characteristics of the input that define the model's efficiency [1, 18, 24] (see also §3.4). On the other hand, the very processing time/resource consumption scaling characteristics that make this approach appealing, also make efficiency prediction challenging; one can no longer just use the resulting efficiency figures as proxies for the complete dataset but rather needs to build a regression model to predict these quantities. To this end, we propose developing regression models (§3.3) that try to quantify the constant factors hidden by the time and space complexity analysis of CF algorithms. We call this methodology the *White-Box*[1] approach, as opposed to the black-box technique described above. Finally, we revisit the sampling strategies used for effectiveness prediction purposes, and propose a scheme whose output can be used to offer accurate predictions for *both* effectiveness and efficiency purposes.

Predicting and modelling resource utilisation has always been a popular topic due to the ever-growing data and the challenges associated with it (e.g., availability, security, consistency, etc.) [58]. Over the years, different research communities have been working towards building and optimising more efficient models to save energy [13, 43], and minimise resource consumption [26, 87]. On this note, we channel our attention towards evaluating and predicting the CF models' resource consumption, a topic that the literature has largely overlooked. Our work could be used by users (e.g., companies that use and deploy CF engines) to allow for better planning of resources, such as CPU and memory, to be provisioned for their applications in local clusters/data centres. This, in turn, could be potentially extended to cloud-based providers, in the same vein as [50], where a simple model can determine the optimal instance configuration based on the the workload,

[1]The term "white box" is also used to refer to simple predictive models, such as linear regression and decision trees, that are easy to explain and interpret. The fact that our proposed solution utterly also uses linear regression is a lucky coincidence.

hardware, and cost, to map the requested resources to physical resources better, maximising the number of users who can use their services concurrently. Moreover, modelling the highly resource- and time-consuming training stage of CF models is beneficial for both the research community and industry, since these CF algorithms are frequently utilised in recommendation engines [81] and evaluation studies [17, 18][2]. Consequently, we formalise the research problem addressed in this work, as described in the following paragraphs.

1.2 Problem Formulation

As discussed in previous section, the aim of this work is to estimate resource consumption, alongside recommendations' quality, for CF models using a white-box approach. In particular, determining when a CF algorithm could have an expensive training time or memory usage is an essential challenge, leading us to our problem formulation: *given a CF algorithm A in the families of algorithms considered in this work, and an input dataset B, predict the time and memory required to train algorithm A on B*. This problem fits within system resource tracking and management topic, which has been broadly studied in the context of computational performance [65, 77].

Our goal is to produce a framework and methodology, which allows the accurate prediction of the processing time and memory usage during the training phase of CF models, based on measurements acquired on samples of the input dataset. To this end, we focus on well-known CF algorithms including singular value decomposition, k-nearest neighbours, slope-one schemes, matrix factorisation, etc. [45]. In our study, we included all CF models implemented in the Surprise framework [45] covering a wide area of the design space. The selection criterion for these CF was based on their popularity and frequency in recent and related benchmarking works, such as [17–19, 24]. Furthermore, we propose using the time and space complexity analysis of these CF algorithms combined with curve fitting primitives as a reliable solution towards the accurate prediction of their training times and memory usage. Then, we sample the input (i.e., the user-item rating matrix – URM) and extract characteristics of each sample (e.g., number of users/items/ratings, density of the rating matrix, etc.). As part of the sampling process, we investigate what sampling strategies could be used to make the resource utilisation prediction problem viable (i.e., how many samples are enough for accurate predictions of processing time or memory usage versus the processing cost of training a given algorithm on those samples). To this end, we formulate the following central research question:

RQ: Given the processing times and memory usage of a CF algorithm on a subset of the data, how can we quantify its expected time/memory consumption for the full dataset?

As part of the central RQ, we have examined the following secondary RQs: **(a)** How can we use complexity analysis to estimate the resource consumption of a CF algorithm? **(b)** Can the efficiency of a CF model on the full dataset be predicted using solely characteristics of the input (i.e., URM) and the efficiency of the CF model on a set of samples? **(c)** Given an upper sample size S%, how do we determine when to stop sampling based on the number of samples for which we obtain consistent resource usage measurements (i.e., the time/memory values are in a tight interval)? **(d)** How should we sample the base data, such that the quality of the predicted efficiency/effectiveness of a CF model is not harmed?

[2]These challenges were also discussed with industry professionals (e.g., Amazon, Netflix), as part of the Doctoral Symposium in ACM RecSys 2020 [63], who highlighted the importance of model selection, training, and deployment based on the available resources and the targeted effectiveness/accuracy outcome.

1.3 Contributions

As discussed in the next section, a large body of literature investigated the effectiveness of the CF, including how the quality (i.e., precision, recall, accuracy) of the recommendations changes with respect to different dataset characteristics [15, 24, 44]. Therefore, this does not fall within the main scope of our work, but we refer to the relevant literature [14, 22, 42, 70, 78, 80, 83] and benchmarks [45] for comparing the selected CF algorithms regarding their accuracy in recommendation tasks. However, to demonstrate the flexibility of our sampling strategy (§3.4) for both efficiency (e.g., training processing time and memory) and effectiveness estimation, we also incorporate in our framework a component that predicts the quality of the recommendations based on the characteristics of the input (§4.5).

To the best of our knowledge, this is the first study that explores the CF algorithms' efficiency performance, addressing the processing time and memory estimation problem through an adaptive sampling-based strategy and different curve-fitting approaches. Our efforts led to the following contributions:

- **C1:** An adaptive sampling strategy that dynamically draws samples to jointly satisfy a user-defined accuracy/error threshold and/or a predefined resource constraint (e.g., maximum time quota);
- **C2:** A methodology that assesses the efficiency of a CF algorithm through training processing time and memory cost models based on its computational complexity;
- **C3:** A tool and framework that predicts the training processing time and memory usage, and effectiveness of the CF algorithms through sampling-based probabilistic analysis and characteristics of the input;
- **C4:** An extensive experimental evaluation, comparing our framework above against state-of-the-art regression models.

Finally, although in this work we have focused on applying the above on CF models, we believe that the proposed methodology and prediction tools are flexible enough to also be applicable to other algorithms and use cases; exploring such directions is outside the scope of this work.

2 BACKGROUND AND RELATED WORK

CF evaluation is a major area of interest, as numerous studies and projects tried to determine the best metrics and practices in this field. So far, both efficiency and effectiveness have been established as the critical areas towards assessing the CF performance [39]. While there are still ongoing debates about online versus offline evaluation [35, 39], it is notably harder (if not impossible) to reproduce online studies. Consequently, offline assessment has been used as a primary tool for establishing the overall performance of a CF and gaining insights into its behaviour under certain constraints and limitations [35, 39]. Traditionally, the evaluation of the CF focuses on splitting the dataset/input into training and testing collections, which are then used to assess the output of the CF model. The limitation of this approach is that often sparsity and popularity biases affect the evaluation protocol [9]. This issue can be alleviated using "random" sampling.

Random sampling techniques have been intensively used in the past decades in various contexts and applications. For example, in databases, the size of the results for a given query was predicted using random sampling [36, 40, 55]. Other works focused on computing the optimal bound for the number of samples needed for satisfying a predefined error metric [4, 21, 33]. Furthermore, efficient sampling techniques have been developed to address the limited computational resources availability for analysing large datasets [59]. All these efforts have contributed towards better ways of drawing samples, which is critical for predicting a chosen quantity since the number of samples and their distribution impacts the accuracy of the predictions [62].

One of the drawbacks of CF evaluation is that the studies only report the quality of the recommendations through effectiveness metrics [35, 39]. However, recent work [56] also presents some insights into the observed efficiency (processing time) of the models. Thus, and in the context of environmental awareness [76], we speculate that as more complex models will be developed, the community will move their attention and efforts to (a) report the (resource) cost of new models, and (b) incorporate ways of minimising the hardware usage. This is yet another reason why it is essential to be able to predict the efficiency cost of CF by performing the training of the models on only small samples of the target datasets.

Recent interest has been shown towards investigating how dataset characteristics (e.g., number of users, items, ratings density/sparsity) affect the quality of the recommendations and their impact on the CF effectiveness. In [1, 24], the authors explore the effect of the structural properties of the user-item rating matrices regarding the accuracy and robustness of the CF algorithms used in the studies. Their results confirm a relationship between dataset characteristics and the CF models' behaviour and highlight the standard practice of using samples to evaluate the effectiveness of CF while alleviating the high processing costs of testing on the complete dataset. In [63], we argued that properties of the input data further affect the inherent trade-off between the efficiency and effectiveness of a CF and that the choice of the algorithm should be based on the latter as well.

Lately, CF have also been evaluated with respect to their accuracy using sampling-based probabilistic analysis methods [17, 18]. To this end, the standard practice consists of training the selected CF on a sample of the dataset and using its offline measured accuracy as a proxy for the effectiveness over the complete dataset [24]. This approach could also be extended to quantify the efficiency of these CF algorithms as discussed in [63, 64]. However, several challenges are associated with sampling for efficiency and effectiveness prediction, as demonstrated in the experimental evaluation (§4.5). Thus, we argue that this is a combination of two factors: (a) the samples produced by the standard practice sampling strategies usually employed for effectiveness purposes do not lend themselves well to efficiency predictions; and (b) due to the inherent (often) non-linear scaling of computational requirements over the dataset size. This work aims to address these gaps, and the resource consumption prediction problem, for a set of highly popular and impactful CF models.

In the past decades, algorithms' processing time prediction problem has been extensively studied across different communities with numerous results. For example, in parallel computing, linear regression models have been used to predict the processing time of different library implementations for multiprocessors [11]. Other works focused on predicting the runtime of various planning algorithms, for selecting which algorithm to run and for how long [28, 41, 68]. Predicting the processing time of parameterised algorithms has drawn high interest from the research community, with existing solutions incorporating the parameters as additional inputs for the prediction models [6, 67]. Another area explored consists of runtime prediction applications, such as determining instance hardness [52] and parameter optimisation/tuning [67]. Additionally, in Database Management Systems (DBMS), in the past 15 years, the number of potential database designs (e.g. indexes, table partitions) and configurations (i.e., knobs to turn and fine-tune) has grown by 3× for Postgres and by nearly 6× for MySQL [82] making the Database Administrators' (DBAs) job very challenging. The main issues with configuration knobs are that there is a large number of parameters that have to be optimised and they control many aspects of the database system (e.g., disk I/O, memory, etc) [82]. To this end, existing solutions [82] focus on auto-tuning these knobs and suggest configuration plans that are better than those derived by human experts. To further optimise DBMS, other works [48, 53] focus on computing cost models for estimating the resource consumption of processing different types of queries. We believe the same scenarios apply to CF, as there are numerous algorithms which can be used, and each one has different configurations and parameters to optimise/tune, resource usage costs, and accuracy rates as showcased in our experimental evaluation. Therefore, the problem of quantifying the performance of CF models w.r.t. both resource consumption and effectiveness becomes more interesting.

3 THE WHITE-BOX APPROACH

When faced with the task of predicting the effectiveness of a CF on a dataset, based on its behaviour on a sample of that dataset, the standard practice consists of building a regression model over data points gathered through iteratively: (i) randomly sampling over all ratings in the dataset, (ii) training the CF over the sample, (iii) evaluating its effectiveness over the sample [24]. We follow a similar strategy with a few notable changes summarised in figure 1. The proposed pipeline covers the steps that our users need to follow to estimate the processing time and memory usage for training a CF algorithm on a chosen dataset. In the following subsections, we discuss each of the steps in more depth, and show how using the expected complexity of a CF algorithm, samples of the data, and a simple regression model, can lead to fast, accurate and interpretable predictions of the efficiency and effectiveness of the CF model.

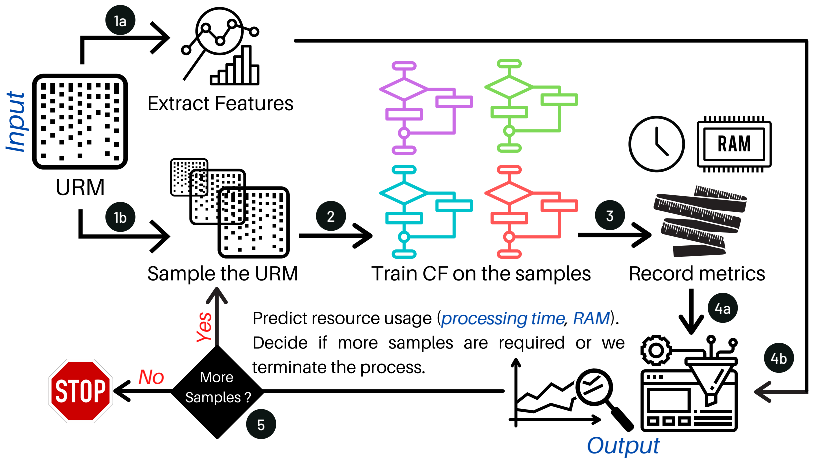


Fig. 1. Overview of the proposed pipeline. Given the the input data (user-item rating matrix – URM), we: (step 1a) extract features such as the number of users, items, ratings, the density of the matrix, etc.; and (step 1b) sample the URM following the strategy described in §3.4. In step 2 we train the various classes of CF models (§3.2) on the samples drawn, while (step 3) gathering efficiency metrics, such as processing time and memory overhead, and effectiveness metrics for the quality of the recommendations (i.e., RMSE for the predicted rating values). In steps 4a and 4b, we train our proposed prediction models, detailed in §3.3, given the recorded metrics, then learn and predict the efficiency (processing time, memory) and effectiveness (RMSE) of the CF on the full dataset. This process (steps 2–4) is repeated until the user-defined termination condition (prediction accuracy, time budget, etc.) is met (step 5).

At the core of our method is the idea that if we understand the computational complexity of each CF algorithm (captured using big-O time and space complexity analysis), we can predict its efficiency (i.e., processing time and memory usage). Throughout the paper, we refer to *time complexity* as the processing time taken by an algorithm to train on a particular input. Similarly, we name *space complexity* the memory usage incurred when a CF model is trained on an input. For all the CF analysed, we define the characteristics (1 step 1a) of their input (i.e., the user-item rating matrix – URM), with respect to the number of users, $\boldsymbol{m}$, the number of items, $\boldsymbol{n}$, the total number of ratings, $\boldsymbol{\rho}$, and the density of the rating matrix, $\boldsymbol{\delta}(=\frac{\rho}{m \times n})$.

3.1 Algorithms

A CF algorithm is considered explicit if its input is based on fixed ratings (usually from 1 to 5) emerging from scores awarded by users to items. In this work, we analyse the following CF categories:

(i) **Basic** algorithms; chosen representatives include a *Baseline* algorithm derived from [46], as well as a Maximum Likelihood Estimation based *Random* approach [61].
(ii) Algorithms based on **K-nearest neighbours**; chosen representatives include *Basic KNN* (*KNN*) [38], *KNN* taking into account the mean rating of each user (*Centred KNN*) [25], and *KNN* taking into account a baseline rating (*KNN Baseline*) (formula (3), section 2.2 in [46]). The former two use Mean Squared Difference (MSD) [16] as the distance metric, while the latter uses Pearson correlation coefficients [38] centred using baseline scores.
(iii) Variants of **matrix factorisation** (MF); chosen algorithms include Non-negative Matrix Factorisation (NMF) [57] and Singular Value Decomposition (SVD) derived from [66].
(iv) **Slope-based** algorithms; chosen representatives include the Slope One scheme [51].
(v) **Co-clustering approaches**; chosen representatives include the algorithm presented in [32].

For further details on these algorithms and their implementation, we refer interested readers to the documentation of the popular Surprise framework [45], which was also used for our experiments. Surprise is a Python-based CF engine that allows users to build and test CF algorithms, which work on explicit feedback datasets. This framework allows researchers to quickly set up their experimental evaluations, provides complete control over the experiments, and contains various tool and metrics to assess the CFs' performance. Surprise also allows users to experiment with built-in datasets (e.g., Movielens [37]), but also to incorporate their bespoke collections. Surprise engine comprises many ready-to-use traditional CF models, described in the previous paragraph, for solving the rating prediction problem [2], and is frequently used as a benchmark in the research community [85]. Our methodology naturally extends to implicit CF algorithms, which rely on inferring users' preferences based on their interactions with the items, such as which pages they visited and for how long, where they clicked, etc.; we omit discussing them in this work due to space constraints.

3.2 Complexity Analysis

Traditionally, the performance of an algorithm is captured through asymptotic worst-case complexity equations using big-O notation [23]. This method allows us to determine an upper bound to the way an algorithm's processing time grows or declines as a function of characteristics of its input. The CF models studied in this work are based on well-known algorithms, for which big-O analysis has been provided by the relevant literature [14, 22, 42, 70, 78, 80, 83]. However, it is often the case that design decisions may make the complexity characteristics of particular implementations to diverge from the theoretical bounds – a fact often hidden behind constant factors or terms ignored during big-O analysis [3]. We thus further formulate and propose time and space complexity equations based on the actual implementation of said CF models. In the following paragraphs, we list the algorithmic complexities based on *(a) literature* [14, 22, 42, 70, 78, 80, 83] and *(b) implementation*. For the latter, we used Surprise's [45] documentation and implementation, and derived the expected time and space complexity of CF models captured through big-O notation and the characteristics of the input outlined in §3. For the purpose of our approach, the number $\boldsymbol{f}$ of latent factors as well as the number $\boldsymbol{e}$ of epochs, where applicable, are considered constants set to the predefined/recommended values by [45].

The **baseline** CF is based on the ALS (Alternating Least Squares) algorithm, the naive solver[3] version, which has a complexity of $O(mnf)$. If we further fix f to a default/recommended number, the complexity can be further abstracted to $\boldsymbol{O(mn)}$ [42]. However, by examining the implementation of the baseline CF algorithm, for a given number of epochs e, firstly the users' baseline is computed in m^2 steps, followed by the items' baseline which takes n^2 operations. If we fix e to a predefined/recommended value, baseline's overall *time complexity* is $\boldsymbol{O(m^2+n^2)}$. For memory usage, apart from the size of the URM, which in all cases takes $\boldsymbol{O(\rho)}$ of memory, the baseline algorithm stores the users', items' resp., baseline in an array of size $\boldsymbol{O(m)}$, $\boldsymbol{O(n)}$ resp.; since, both baselines are used by this CF, the overall *space complexity* is $\boldsymbol{O(m+n)}$.

The **random** algorithm, based on Maximum Likelihood Estimation (MLE), predicts the missing ratings over a normal distribution, computed in maximum $\boldsymbol{O(mn)}$ steps [80]. The implementation reveals that the random CF computes a global mean and standard deviation during its training phase. These are typically done in two stages (first, compute the mean, then the standard deviation), each of which scans over all rating values. As such, the algorithm's *time complexity* is in $\boldsymbol{O(\rho)}$. The random CF does not require additional memory during its training phase as there are no auxiliary data structures that need to be allocated for the computing of the mean and standard deviation. Therefore, its *space complexity* matches the size of the URM, namely $\boldsymbol{O(\rho)}$.

For the neighbourhood based CF algorithms (i.e., **KNN**, **centred KNN**, and **KNN baseline**), the training phase computes the distance of every user to every other user (or every item to every other item, depending on whether the approach is user- or item-centric), taking into account only the items (users, respectively) that are common across users (items, respectively). This leads to a complexity of $\boldsymbol{O(m^2n^2)}$ [83, 89]. However, at the implementation level, for **KNN** we derived a *time complexity* of $\boldsymbol{O(\frac{\rho^2}{x}+x^2)}$, where $\boldsymbol{x}$ can be either $\boldsymbol{m}$ for user-based KNN, or $\boldsymbol{n}$, for item-based KNN, respectively. At the core of the KNN-based CF, the similarity function computes the distance across the relevant users or items with respect to (a) the ratings they gave (for users) and (b) the rated items. By investigating the rating frequency distribution of all datasets outlined in section §4.1, we concluded that the number of per-user and/or per-item ratings follows a uniform distribution (i.e., $\frac{\rho}{m}$ ratings per user, or $\frac{\rho}{n}$ ratings per items). We thus make the following simplifying assumption: in the similarity function, the distances are computed in $x \times \frac{\rho^2}{x^2}$, which can be simplified to $\frac{\rho^2}{x}$. Then, the distance is computed for pairs of common users/items in x^2 time (m^2 for users or n^2 for items, respectively). **Centred KNN** has a similar complexity to KNN, as they use the same similarity metric (MSD), but takes an extra $\boldsymbol{(\rho)}$ step to compute the mean ratings of each user (item, respectively), which brings the overall *time complexity* to $\boldsymbol{O(\frac{\rho^2}{x}+x^2+\rho)}$. **KNN Baseline** is also based on KNN, and computes distances across users (items, respectively) using Pearson correlation coefficients [38], and takes into account baseline ratings. It's overall *time complexity*, as derived from its implementation, is the same as the one for Centred KNN - i.e., $\boldsymbol{O(\frac{\rho^2}{x}+x^2+\rho)}$. To compute the distances and similarities between pairs of users, items resp., the **KNN-based** CF utilise additional matrices of size $\boldsymbol{O(m^2)}$, $\boldsymbol{O(n^2)}$ resp. Therefore, the *memory usage* for KNNs during training is $\boldsymbol{O(m^2)}$ or $\boldsymbol{O(n^2)}$ depending on whether distances/similarities are computed across users or across items.

The **NMF** model is based on the SGD algorithm, which achieves a computational complexity of $O(em\rho)$ [70]. If we fix the number of epochs, the complexity can be reduced to $\boldsymbol{O(m\rho)}$. In the Surprise framework [45], for a fixed number of epochs and factors, NMF decomposes a given user-item ratings matrix, with respect to the number of users (m), items (n), and ratings (ρ). Therefore, the missing ratings are computed in $\boldsymbol{O(\rho+m+n)}$ (or $\boldsymbol{O(ef(\rho+m+n))}$, including

[3]Naive ALS is described in http://web.cs.ucla.edu/~chohsieh/teaching/CS260_Winter2019/lecture13.pdf

the number of epochs, e, and factors, f). The implementation also reveals that during training additional memory is allocated for two matrices, one for the user latent factors, which takes $\boldsymbol{O(mf)}$ of memory, and the other one for item latent factors, which is stored in $\boldsymbol{O(nf)}$. This brings NMF's overall *memory usage* to $\boldsymbol{O(mf + nf)}$.

SVD, a popular CF-based approach, has been intensively used to produce recommendations on explicit datasets. Over time, multiple variations of SVD have developed [22], leading to a significant number of implementations. However, most of them converge to a complexity of $\boldsymbol{O(mn^2)}$, even though other studies, such as [47], claim that the overall complexity of SVD is close to $O(n^2m + m^2n)$. The SVD's implementation found in Surprise [45] uses the user-item ratings matrix in $\boldsymbol{O(e\rho f)}$ time to factorise the corresponding user and item factors. SVD's *time complexity* can be simplified to $\boldsymbol{O(\rho)}$ for a fixed number of epochs (e) and factors (f). For memory requirements, during training, SVD follows similar storage requirements as NMF, having a *space complexity* of $\boldsymbol{O(mf + nf)}$.

For slope-based solutions, the **Slope One** algorithm has a generic time complexity of $\boldsymbol{O(mn^2)}$, as it computes the average difference between pairs of relevant items as described in [78]. At implementation level, Slope One firstly computes the frequency of the pairs of items *(i, j)*, followed by the deviation between item *i*'s ratings and item *j*'s ratings. This is achieved in $O(\frac{\rho^2}{m} + n^2)$. Then, the relevant ratings are predicted using the users' mean ratings combined with the aforementioned frequency and deviation arrays, which means another $O(\rho)$, leading to an overall *time complexity* of $\boldsymbol{O(\frac{\rho^2}{m} + n^2 + \rho)}$. For memory requirements, during training, Slope One allocates two additional matrices to compute the frequency and deviation between pairs of items across the dataset. Consequently, the expected *memory usage* is $\boldsymbol{O(n^2)}$.

Lastly, the **Co-clustering** CF with a fixed number of user-item clusters converges towards a computation complexity of $\boldsymbol{O(mn)}$ [14]. By examining its implementation, Co-clustering splits users and items into clusters in $O(m) + O(n)$ and co-clusters in $O(\rho)$ steps, using an assignment technique similar to K-means. This makes Co-clustering train in $\boldsymbol{O(m + n + \rho)}$ time. During training, the Co-clustering CF computes the users and items mean across the entire collection and stores them in arrays of size $\boldsymbol{O(m)}$, $\boldsymbol{O(n)}$ resp. Then, users and items clusters and co-clusters are built, which require another $\boldsymbol{O(m)}$, $\boldsymbol{O(n)}$, and $\boldsymbol{O(mn)}$ resp. of memory. This brings the overall *space complexity* to $\boldsymbol{O(m + n + mn)}$.

3.3 Prediction Models

Knowing the worst-case (big-O) complexity can help determine the upper bound on the number of resources used by an algorithm while being executed against all possible inputs [23]. However, in practice, the likelihood of encountering inputs that elicit the worst-case processing time of an algorithm is relatively low [23]. Therefore, the computational complexity theory has devised the average-case complexity to measure the efficiency of an algorithm through its expected processing time/memory averaged over all inputs. Computing average-case complexity is often a hard problem since the distribution of all possible inputs is required to derive theoretical bounds analytically. Instead, our methodology is based on *approximating probabilistic analysis*, through an adaptive sampling strategy (1 step 1b), which predicts the expected processing time/memory of a given CF algorithm over an input/dataset. We employ this strategy for determining both the processing time and memory usage requirements.

Using the above algorithmic complexities, the processing times and memory usage measured across different inputs (1 step 3), and the characteristics of the data, we propose building the following types of models for time/memory prediction. Our approach is based on estimating the hidden factors (or unknown parameters) in the time and space complexity equations derived from the algorithms' implementations, mentioned in §3.2. Given that the processing time/memory estimation is based on an overdetermined system, with more sets of equations than unknowns, we constructed our models based on the *least squares* approach [10]. This technique is based on minimising the sum of the squares of the residuals (i.e., the difference between the observed/measured processing times/memory usage and the predicted/fitted values) computed in the equations.

3.3.1 Linear Models. Since for each sample of the input we know the fixed number of users, items, and ratings, we encapsulate the performance (i.e., processing time and memory) of the CF algorithms through complexity equations summarised as follows:

$$performance = f(X) = \alpha X + \beta \tag{1}$$

where X is a combination of the independent variables m, n, and ρ, while α and β are the slope and intercept that were computed using *linear least-squares regression.* For example, the equation for computing processing time for *baseline* becomes $\alpha(m^2+n^2)+\beta$, that of *NMF* becomes $\alpha(\rho+m+n)+\beta$, etc. Similarly, for predicting memory usage of SVD, we compute α and β using $\alpha(mf + nf) + \beta$. This approach allows us to quickly compute the hidden factors of the complexity equations, while capturing the characteristics of the URM, as evidenced by our experiments. Additionally, we also compute the *prediction error interval* [27, 60], which quantifies the uncertainty of our predicted time and memory usage. This allows us to provide upper and lower bounds on the estimates at each sample size. Furthermore, we compute these intervals using a combination of the variance of the outcome variable (i.e., time, memory) and the estimated variance of the model[4] [27, 73].

3.3.2 Bayesian Models. Another approach that we explored was to estimate the *performance* of the CF algorithms using a Bayesian inference [20]. In this setup, our aim is still to compute the hidden coefficients of the complexity equations from §3.2, but using probability distributions rather than point estimates. Therefore, our predicted variable (i.e., processing time, memory) will be drawn from a probability distribution. To this end, we infer the performance of the CF using a normal (Gaussian) distribution [20], characterised by mean and variance, as seen below:

$$performance \sim \mathcal{N}(\alpha X + \beta, \sigma^2) \tag{2}$$

where α, β, σ^2 are also coming from distributions. Since σ^2 will always be a positive number, we chose a prior distribution, which yields only positive values, such as the Exponential distribution [20], where $\sigma^2 \sim Exp(1)$. For α and β coefficients, we used normal (Gaussian) distributions and restricted the parameter space using priors learnt with the previous linear regression models [31]. In Bayesian inference, the main goal is to use sampling methods to draw samples from the posterior to approximate the posterior [20]. According to the standard practice [20, 31], we can use Monte Carlo methods [71] to draw random samples from a distribution to approximate the said distribution. While there are several ways to perform Monte Carlo sampling, the most common and currently used [20, 31] is Markov Chain Monte Carlo (MCMC) sampling [12][5]. One of the challenges of fitting the Bayesian models is to ensure that all parameters show convergence. This can be checked by computing the *potential scale reduction* ($\hat{R}$), which should always have a value below 1.1 [20]. The rule of thumb is that convergence has been achieved when $\hat{R}$ is very close to 1.0 [31].

As with the linear regression models, for Bayesian regressors, we can compute the Monte Carlo Standard Error (MCSE), which is an estimate of the inaccuracy of Monte Carlo samples in MCMC algorithms [31]. MCSE can be used to quantify the *uncertainty of the predictions* for processing time

[4]An example on how to compute the prediction error intervals is provided in https://learnche.org/pid/least-squares-modelling/least-squares-model-analysis#prediction-error-estimates-for-the-y-variable.

[5]In Python, MCMC is implemented using the PyMC3 library available at https://docs.pymc.io/. This is also what we used in our experiments.

and memory usage in MCMC models by computing the standard deviation and variance around the posterior mean of the samples[6] [31, 75].

3.3.3 Baselines and Contenders. We compare our proposed processing time and memory prediction models (white-box approach) against two types of baselines: (i) a *hard* baseline using linear regression to learn the hidden factors in the complexity equations described in the literature; and (ii) a *soft* baseline, which assumes that the complexity of the algorithms is unknown, and therefore the processing time and memory predictions will be computed using just the characteristics of the input (black-box method). The latter was tested using several off-the-shelf state-of-the-art regression algorithms available through the H2O analytics platform[7]. A few examples of the tested regressors include Random Forest, Deep Neural Net, Support Vector Machine (SVM), and Adaptive Boosting. In the experimental evaluation, we only report on the results of the best performer with regards to prediction accuracy (i.e., lowest RMSE), namely Gradient Boosting Machine (GBM) [30]. GBM was ranked as the best state-of-the-art regression model since it acquired the lowest RMSE, following the K-fold cross-validation procedure described in [49].

In our experimental evaluation (§4.5), we also predict the *effectiveness* (i.e., quality of the recommendations) of the CF given a sample of the input data and characteristics of the URM. To this end, we have employed state-of-the-art regressors, which include Gradient Boosting Machine (GBM) [30], AdaBoost Regressor (ABR) [29], and Support Vector Regressor (SVR) [5]. These models were selected by running the H2O AutoML tool [49] in a process similar to the one described in the previous paragraph. Since the effectiveness of the CF algorithms is a well-studied area [1, 8, 24], it is not within the main scope of this paper. However, as demonstrated in our results (§4.5), by using our adaptive sampling technique (§3.4), we do not need to employ different sampling strategies to draw samples for predicting efficiency and effectiveness. Consequently, using the sampler in one go, we gain information about the recommendations' quality and the performance and resources used while training the CF algorithms.

3.4 Adaptive Sampling

The standard practice of drawing samples for assessing the effectiveness of the recommendations involves choosing random triplets of the form $(user, item, rating)$ from the input, and then filtering them based on a predefined characteristic of the URM (e.g., the density of the sample needs to be above/below a certain threshold) [1, 24]. However, this approach does not work well for drawing samples to predict the efficiency (i.e., processing time, memory) of the CF algorithms; see §4.3 for the related results. Instead, we propose a simple yet reliable sampling strategy, which provides good samples that can be used for predicting both the efficiency and effectiveness of the CF at the same time. It should be noted that drawing samples from a dataset is not for free, and therefore, we believe that it is essential to use a strategy that can be used not only to determine the accuracy of the recommendations, but also reflects the complexity characteristics of the input data.

When sampling the URM, we asked ourselves how many samples and how large should they be to get a representative measure of the CF algorithms' performance. To this end, we propose the following sampling strategy (figure 1 step 1b) described in algorithm 1. Initially, the user of our system provides us with an upper sample size – say S (%) – as well as with a time budget T for our method; however, if these are not provided or unknown the default values, as outlined in §4.3, will be used for sampling the URM. We then draw an initial sample by uniformly at random selecting a $S\%$ subset of the users and $S\%$ subset of the items, and including in the sample all associated ratings.

[6]A practical example for quantifying uncertainty can be found in https://towardsdatascience.com/pymc3-and-bayesian-inference-for-parameter-uncertainty-quantification-towards-non-linear-models-a03c3303e6fa.

[7]Fro the list of regressors and documentation of H2O, see: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html

We then use a strategy similar to Monte Carlo rejection sampling [79], to recursively subsample to produce even smaller samples. This strategy has two key characteristics: (a) sub-sampling allows us to produce a number of samples at different sampling rates at a fraction of the cost of sampling the complete dataset; and (b) by sampling user/item IDs, the sample better reflects the complexity characteristics of the base data. For each sample drawn, we train the CF models and record its training time (figure 1 steps 2 and 3); we then decide whether to proceed with more samples given the so-far cumulative execution time of the above process and the time budget T (1 step 5). Furthermore, the number of samples we draw using algorithm 1 should be dynamic and based on *(a) a user-defined upper limit* and *(b) a user-defined accuracy/error threshold*. Ideally, we would like to have a number of samples that provides good accuracy for the processing time and memory prediction models. However, we should not use too many samples, such that their total processing time would be larger than the training time of the entire dataset (see §4.5 for the related results).

Algorithm 1: Adaptive Sampling Algorithm

```
    Input  :Dataset D, Algorithm A, max sample size S, max time budget T, accuracy_threshold.
    Output :The required samples and their stats.
1   train_samples = []; stats = [];
2   T' = T; S' = S; D' = D;
3   do
        /* Compute sample size and time budget for current iteration    */
4       (T_cur, S_cur) = adapt(T', S');
        /* sample D' with the constraints                                */
5       d = sample(D', S_cur);
6       train_samples.push(d');
7       t = 0;
8       do
9           t_start = time();
            /* measure processing time, memory by training A on d       */
10          stats.push(get_memory_and_processing_time(A, d));
            /* disregard high-end outliers                               */
11          m = stats.smart_mean();
            /* compute confidence interval                               */
12          c = stats.smart_confidence_interval();
13          t_end = time();
14          t += (t_end - t_start);
15      while (c/m >= accuracy_threshold and t <= T_cur);
16      T' -= t; D' = d; S' = S_cur;
17  while (T' > 0);
18  return (train_samples, stats)
```

For sampling the URM using the proposed algorithm, the user-defined upper limit is captured through a maximum time budget T and a maximum sample size S. As each sample is drawn, the remaining time budget and sample size are (re)calculated for each iteration through the *adapt()* function, which takes into account the previous time budget T' and sample size S'. The *adapt()* function gradually reduces the next sample size that will be drawn (e.g., in decrements of 10%) and the amount of time allocated for this operation, based on the time t spent so far on drawing the current sample and the available time budget. The measurements are repeated multiple times for each (sub)sample to compute the average m of the resource usage (processing time or memory overhead) values recorded while training on the current sample, and the confidence interval c selected by the user (or default 99%). In this computation, we discard the measurement for the first iteration to avoid effects of cold caches and overheads of the language runtime. Thus, the algorithm is filtering high-end outliers that can skew the accuracy of the processing time and memory prediction models. The proposed sampling algorithm stops drawing sub-samples for a given sample size if either there is not enough time left within the time budget T or the *accuracy_threshold* has been met. Furthermore, our sampling algorithm can also use a user-defined *accuracy_threshold* as part of its input. This *accuracy_threshold* controls the variance of the measurements of the processing times recorded by training the CF on the sub-samples within a sample size. The iterative execution on a single sample then may terminate early if the ratio c/m satisfies said threshold. Thus we ensure that the processing times or memory overhead measured on the samples have low variance (i.e., the recorded time/memory values are in a tight interval). This is an important aspect to be considered while sampling, as the quality of the samples can impact the accuracy of the resource cost models as demonstrated in §4.5.

The proposed adaptive sampling algorithm allows us to minimise the number of samples needed for processing time and memory overhead prediction without sacrificing the accuracy of the resource cost models. Moreover, using an accuracy threshold and a fixed confidence interval lead to a reliable *stopping strategy* for drawing random samples and collecting time and memory measurements. Another idea that we investigated for determining when we have acquired a large enough sample is to analyse the *prediction error interval* in LR models and/or the *uncertainty* in Bayesian models described in §3.3 and showcased in §4.5. However, as this method depends on the resource prediction model used (i.e., works only with LR and Bayesian models), we prefer and propose a model-agnostic sampling strategy based on an upper limit for time and sample size, as well as an accuracy/error threshold which reflects the quality of the samples.

Moreover, this tool can be easily customised by the users, allowing them to determine the optimal bounds for the number of needed samples based on how much variance is allowed to be present in the generated samples. In our experiments, we used a 0.1 accuracy threshold, corresponding to 10% - 15% variance and 99% confidence interval (as described in §4.3). However, depending on the constraints imposed on the random samples, the users could alter the accuracy threshold values to be in line with their needs and requirements. We note that a smaller accuracy threshold will map to less variance in the processing time values, and hence the predictions are more accurate.

4 EXPERIMENTAL EVALUATION

This section describes our experimental evaluation methodology and discusses our results. In brief, our extensive experiments show that by using a relatively small subset of a dataset, $\mathcal{D}_s$, and the time it takes to train a CF algorithm $\mathcal{A}$ on $\mathcal{D}_s$, we can accurately predict $\mathcal{A}$'s expected *processing time* and *memory usage* on the full dataset $\mathcal{D}$. Additionally, we show that there is an inherent efficiency-effectiveness trade-off between the quality of the predictions and their procurement cost. Further, we provide insights into the cost of sampling compared to the cost of training the CF models, as well as the cost of running our prediction models on the base data. Lastly, we discuss the advantages of the proposed sampling strategy compared to current standard practices.

4.1 Datasets and Recommendation Task

For this study, we used the MovieLens (ML) 100K, 1M, and 20M collections [37], as well as the GoodBooks (GB) 10K dataset [88]. Each of these datasets consists of explicit ratings, from 1 to 5, given by users to items (i.e., films for ML and books for GB). While ML 100K (610 users and 9724 items) and 1M (6040 users and 3706 items) are smaller collections with densities of 0.017 and 0.045, respectively, GB 10K (53424 users and 10000 items) is a larger collection with a density of 0.012. In addition, we also probed our efficiency cost models with ML 20M, a very large dataset containing 138493 users, 26744 items, and a density of 0.0043. The different datasets characteristics and sizes allowed us to experiment with our proposed performance prediction tool and methodology, showing that it can be successfully used on collections with different properties.

The recommendation task investigated in this paper refers to predicting the usefulness or relevance of a given item to a user [72]. In short, after an item is selected, the CF model estimates the rating the user would give to this item, and if the rating is above a particular threshold value, then the item is presented to the user as a recommendation.

4.2 Evaluation Protocol

All experiments were carried out on Linux servers, each having 2 Intel Xeon E5-2660 CPUs (8 physical cores each with 2-way hyper-threading (HT)) and 64GB of RAM, running Ubuntu Linux 14.04.6. As the GoodBooks dataset is significantly larger and denser, we ran the corresponding experiments on a higher-spec Linux server with 4 Intel Xeon E7-4870 v2 CPUs (15 physical cores each with 2-way HT) and 512 GB of RAM, running Ubuntu Linux 16.04.7. During the experimental evaluation, all resource-intensive processes were suspended to avoid interference with our measurements.

We measured the processing times of the various CF algorithms on inputs sampled as described above. To this end, we utilised the *getrusage* method from Python's *resource* module, with the overall *training time* for each sample computed as the sum of the time spent executing in user mode (*ru_utime*) and system mode (*ru_stime*) [8]. To retrieve the memory usage of the CF, we recorded the information from the proc filesystem via the "/proc/ [pid]/status" file[9], which contains the utilised memory by the current process (identified by *pid*) reported directly by the Linux kernel. From this file, we based our memory usage computations on the "VmSize" field, which returns the overall memory used by a specific process. To cross-check our approach regarding memory measurements, we also recorded the memory usage while training the CF model using a memory profiler. For this task, we used the *memory-profiler*[10] module from Python, and ensured that the results reported by the profiler match the ones recorded using the proc filesystem.

We gathered measurements for samples as provided by our sampling strategy, trained our models on the produced statistics, and used them to predict the respective resource usage over the complete dataset. We evaluated our predictions using normalised RMSE (NRMSE) computed as:

$$NRMSE = \frac{RMSE}{\overline{y}} \tag{3}$$

where $\overline{y}$ is the mean of the actual time/memory values in the corresponding sample. As KNN Baseline and Centred KNN followed the same time and space complexity and showed similar processing time and memory usage behaviour in all experiments, we only show results for the latter.

4.3 Sampling Strategy

We gathered measurements for values of S (upper limit to the sample size) ranging from 10% all the way to 100% in increments of 10% (i.e., 10%, 20%, . . ., 100%) using algorithm 1. For each draw, we also used a fixed random seed $\mathfrak{z}$ from a predefined set of seeds to ensure reproducibility. For the scope of our experimental evaluation, in algorithm 1, we set the confidence to 99%, and use a

[8]Please see https://docs.python.org/3/library/resource.html for further information.
[9]More information about the proc filesystem can be found at https://man7.org/linux/man-pages/man5/proc.5.html.
[10]Memory-profiler is available at https://pypi.org/project/memory-profiler/.

predefined variance of the processing times and memory overhead in each sample size (e.g., 10%, . . ., 100% of the data) in the range of 10% to 15% coupled with the default accuracy threshold of 0.1. Last, we set the overall time quota T to the default values of 50000, 2000, 500, 100 seconds per sample for ML20M, GB 10K, ML 1M, and ML 100K, respectively. The results (§4.5) indicate that this setup offered a good trade-off between the number of samples needed and their quality. Other values were explored but led to comparable results and are omitted for space reasons. Furthermore, to highlight the performance of our resource consumption prediction models we sampled the input data all the way to 100%, which accounts for the full dataset; however, our empirical findings (§4.5) indicate that a default sample size of 30-40% produces accurate predictions of the processing time and memory overhead of CF required during training.

Finally, we also analyse whether the standard practice sampling strategy [1, 24] used for determining the effectiveness of a CF, performs for predicting its efficiency (i.e., training time and memory), and we show that our proposed sampling approach leads to good accuracy for both efficiency and effectiveness prediction purposes.

4.4 Contenders

For **RQ (a)** and **(b)**, we investigated whether the dependent variable (i.e., the *expected training time* and *memory usage* of a CF algorithm on a dataset $\mathscr{D}$) can be predicted using samples of $\mathscr{D}$ and the characteristics of the input representing the independent variables as described in §3. As part of **RQ (a)**, we conducted an extensive experimental evaluation of our White-Box approach, as discussed in §3.2 (denoted in the remainder of this section as WB/LR and WB/Bayes, when using a linear regression or Bayesian prediction approach respectively). We compare our solution against two types of baseline approaches: (i) a *hard* baseline (denoted WB/Lit/LR), in the form of a white-box linear regressor built using algorithmic complexities provided by the relevant literature, and (ii) a *soft* baseline, in the form of the best performing state-of-the-art black-box regressor (GBM).

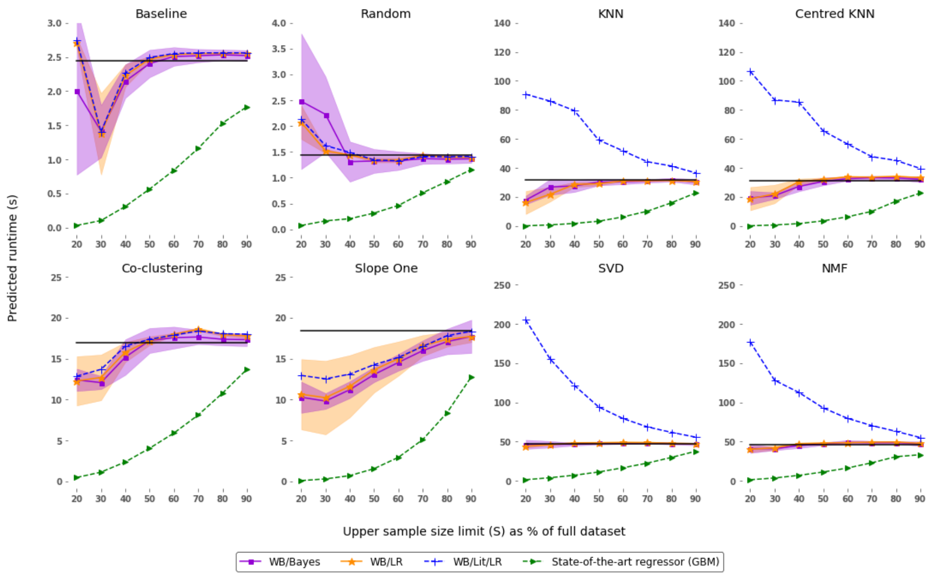
In the latter case, the regression model is trained on characteristics resulting from training the CF models on samples of the input dataset, without having any knowledge of the inner space/time complexity of the latter. To this end, we augmented the structural input features (m, n, ρ), with rating distribution/frequency-related features. Specifically, in line with the practice in the state-of-the-art [1, 24], we modelled the concentration of users' (items', respectively) ratings by using the **Gini** coefficient [34] as described in equation 4, where $\mathcal{w}$ is the number of users (items, respectively), $\rho_{\mathcal{k}}$ is the number of ratings given by a user (or received by an item, respectively), and $\rho_{\mathcal{total}}$ is the total number of ratings.

$$Gini_w = 1 - 2 \times \sum_{k=1}^{\mathcal{w}} \left(\frac{\mathcal{w} + 1 - k}{\mathcal{w} + 1} \right) \times \left(\frac{\rho_{\mathcal{k}}}{\rho_{\mathcal{total}}} \right) \tag{4}$$

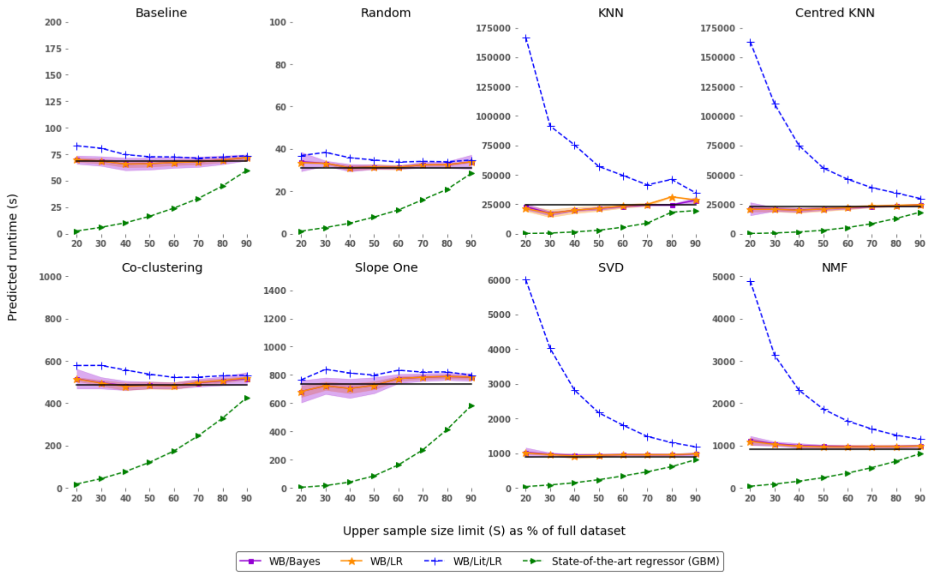
We thus compute the Gini coefficient for users, $Gini_{users}$, and items, $Gini_{items}$, and include them as extra features for the black-box approach. We experimented with multiple state-of-the-art regressors and report results for the best performer, Gradient Boosting Machine (GBM).

4.5 Results

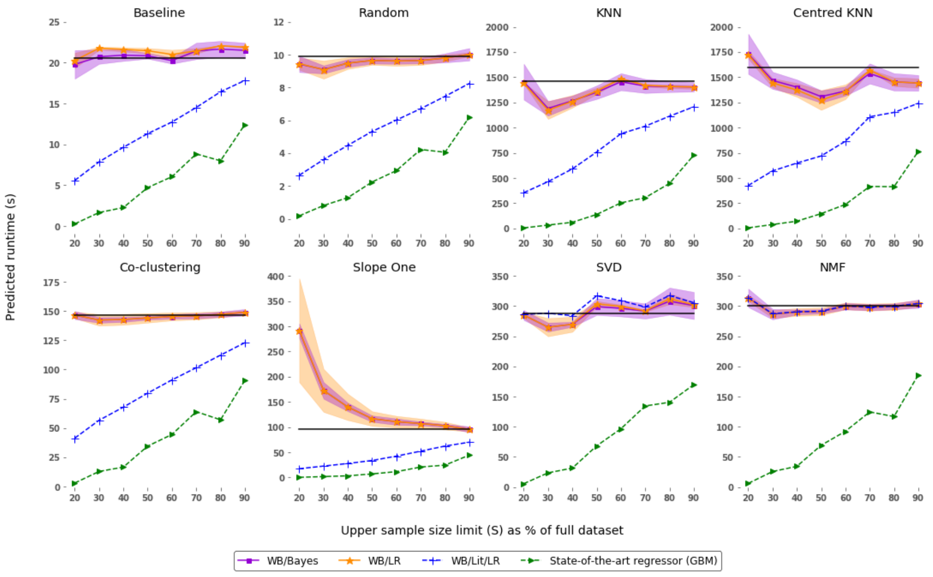
Figures 2a, 2b, and 2c depict the predicted training time for the complete dataset, as predicted when using different upper sample sizes. Specifically, the curves on these figures show the predicted full-dataset training time (y-axis) versus the upper sample size limit (x-axis) on which the contenders where applied. The horizontal black line represents the actual training time over the entire dataset. In other words, the closer a curve is to the black line the more accurate the prediction, and the earlier a curve approaches the black line the smaller a sample is required to achieve this result. The orange and purple areas show the *prediction error interval* and *uncertainty* for the predictions of WB/LR and WB/Bayes respectively, as presented in §3.3. Our results indicate that WB/LR, using simple linear regression, outperforms the much more complex best performing state-of-the-art regressor (GBM). In most cases a 30%-40% upper sample size limit seems enough to allow WB/LR to achieve highly accurate prediction. Interestingly, WB/Bayes also seems to achieve good accuracy in the processing time prediction task with similarly small sample sizes; however, its training cost is considerably greater than the one for WB/LR and WB/Lit/LR as discussed shortly (table 1).



(a) MovieLens 1M



(b) MovieLens 20M



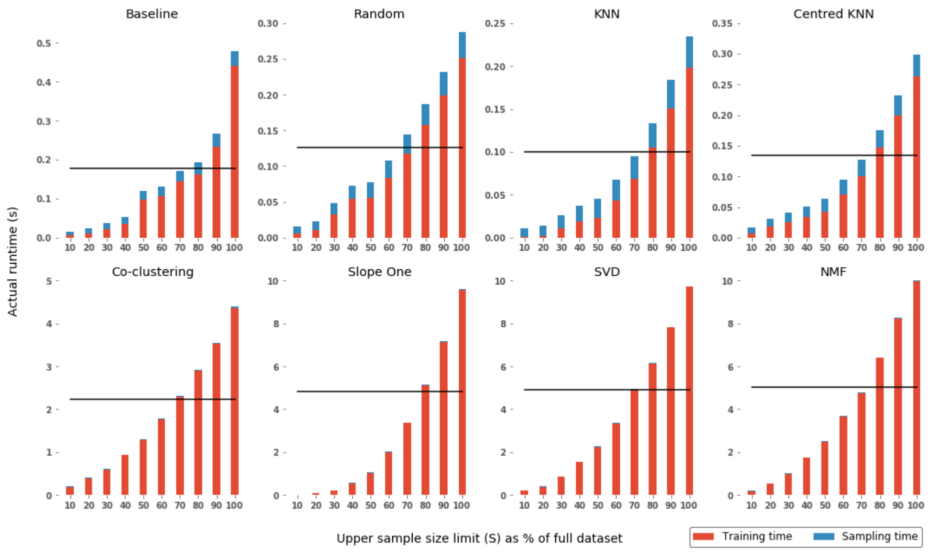
(c) GoodBooks 10K

Fig. 2. Predicted processing times for (a) MovieLens 1M, (b) MovieLens 20M, and (c) GoodBooks 10K collections, for various values of S (upper sample size limit), using our approaches (WB/LR and WB/Bayes) and the two baselines (linear regression over literature complexity equations (WB/Lit/LR), and state-of-the-art regressor (GBM)). The black horizontal line represents the actual training time over the entire dataset.

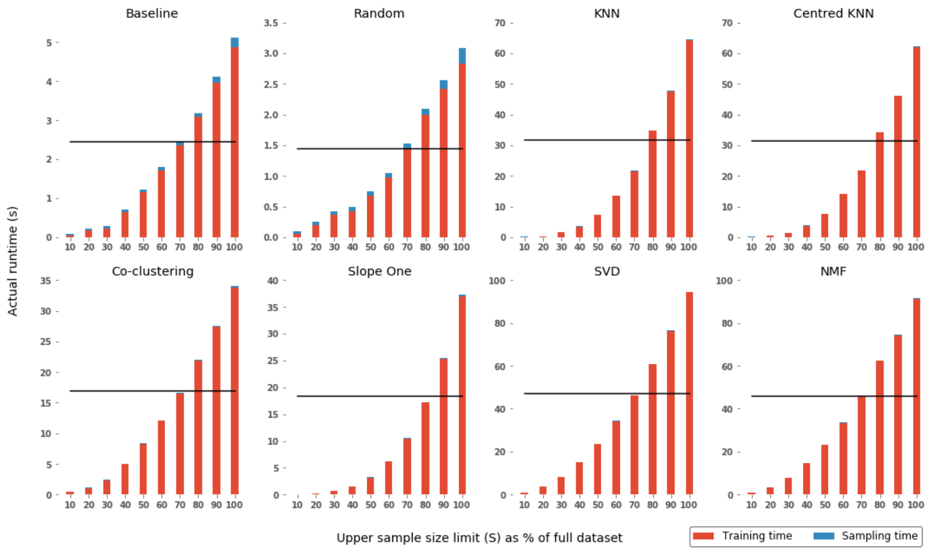
When assessing the overall performance of our framework, we also need to consider the cost of sampling the dataset, training the CF algorithms on the samples, and running the prediction models on the base data (i.e., (processing time and memory usage). To this end, figures 3a, 3b, and 3c highlight the cost of acquiring samples from the dataset (blue bars) stacked on top of the cost for training the CF on these samples (red bars). The training time for the full dataset is depicted with a black horizontal line. In other words, when a composite blue-red bar reaches or exceeds the black horizontal line, then this denotes that it is more efficient to train the CF algorithms directly on the full dataset rather than draw samples and train on them. We note that for CF algorithms with more expensive training cost (e.g., KNN-based CF), the processing time exhibits a steeper increase across samples. Hence, our framework is more valuable for predicting the efficiency of such CF models, as we can draw a small and cheap sample of the data, while accurately estimating the processing time and memory of the CF on the full dataset.

So far, we examined how the processing time varies when sampling the datasets and training the CF algorithms on the given samples. However, training and running the prediction models on the base data also involves a cost. Table 1 presents the average time taken by each prediction model across ML100K, ML1M, and GB10K. As we can see, the cheapest models are WB/LR and WB/Lit/LR, followed by GBM, while WB/Bayes is by far the slowest (by several orders of magnitude). Therefore, for predicting the efficiency of a CF model, it is important not only to select an adequate sampling strategy but also a cheap and accurate predictor, such as our proposed approach (WB/LR).

Figures 4a, 4b, and 4c illustrate the accuracy of the predictions for memory usage across the three datasets. These figures are similar in design to 2; on the y-axis we have the memory usage for training the CF models over the complete dataset, as predicted using as the upper sample size



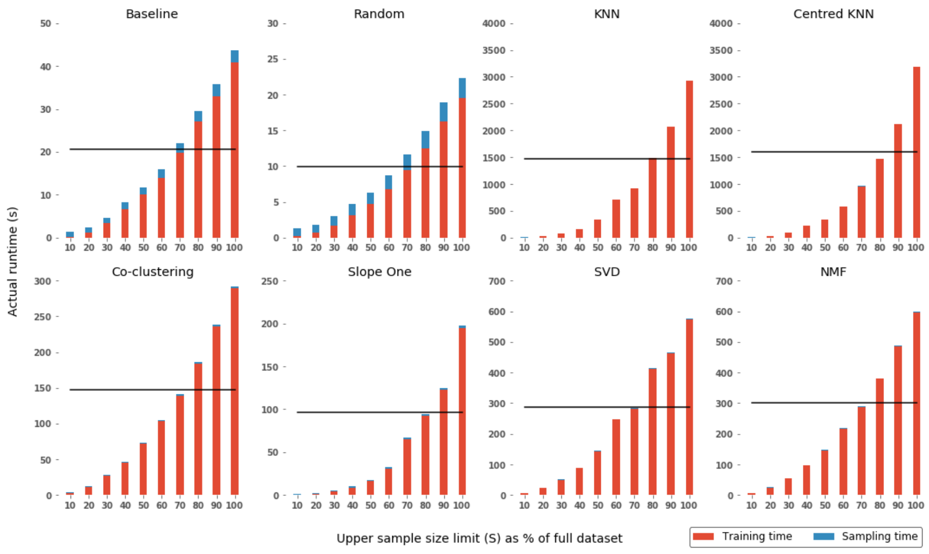
(a) MovieLens 100K



(b) MovieLens 1M

limit the value on the x-axis, while the horizontal black line depicts the actual memory usage for training the CF models over the complete input dataset. Similarly to the time prediction models, the proposed approach (WB/LR, orange curve) outperforms the best state-of-the-art regressor (GBM, green curve). WB/Bayes (purple curve) again achieves good accuracy in the prediction task, but for a much higher training cost as discussed earlier. Again, a 30%-40% sample suffices for WB/LR and WB/Bayes to produce highly accurate predictions. WB/Lit/LR is not shown on this figure as, alas, the literature around the CF models considered here provides no space complexity analysis.

Another benefit of using the processing time and memory prediction models is knowing when to stop sampling the input data and training the CF models. We have discussed in §4.3 a simple



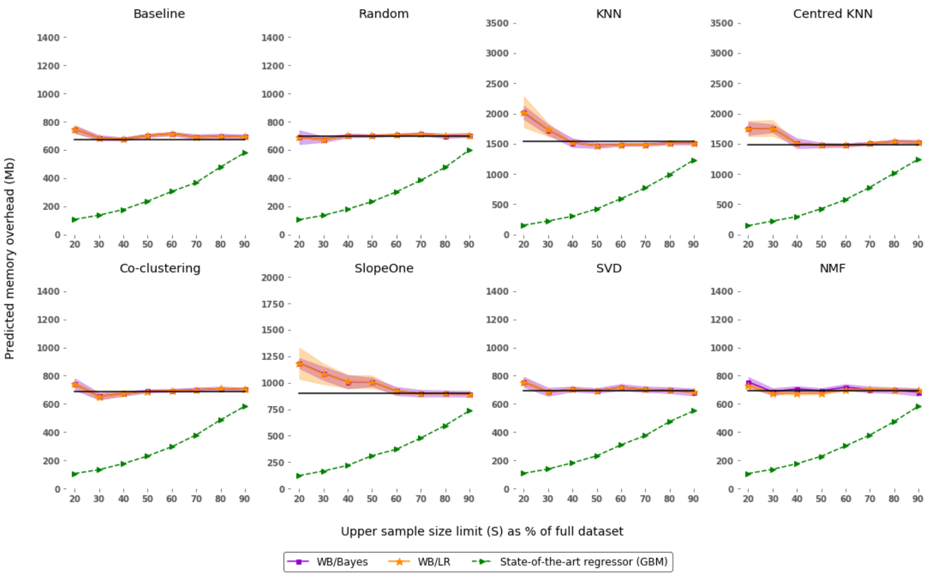
(c) GoodBooks 10K

Fig. 3. Sampling and training time cost across datasets ((a) MovieLens 100K, (b) MovieLens 1M, and (c) GoodBooks 10K) w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line).

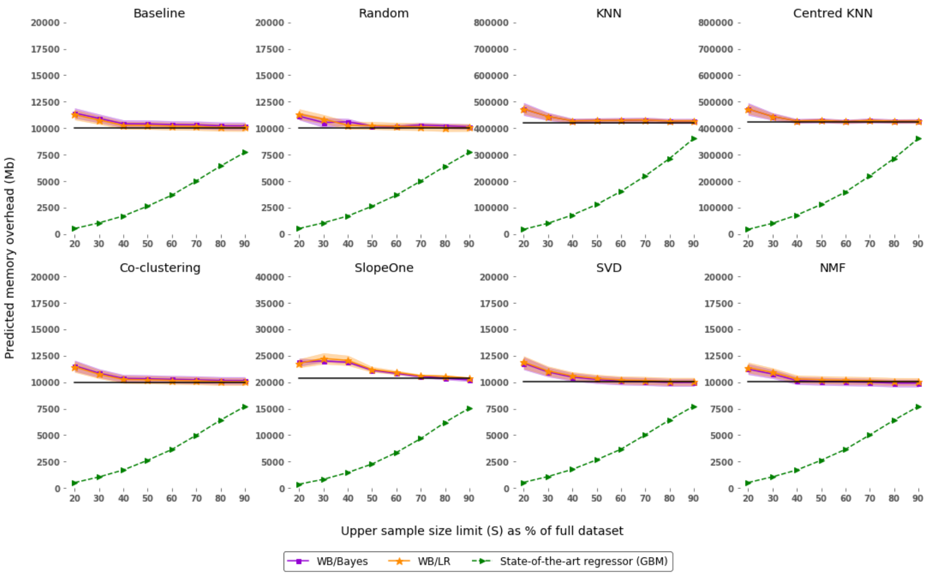
Table 1. Mean and standard deviation of processing time (i.e., training and prediction time) for the prediction models across MovieLens 100K, MovieLens 1M, and GoodBooks 10K.

	Mean Runtime (s)				**Standard Deviation**			
	WB/LR	*WB/Bayes*	*WB/Lit/LR*	*GBM*	*WB/LR*	*WB/Bayesian*	*WB/Lit/LR*	*GBM*
Baseline	0.000846	12.976536	0.000821	0.560964	0.000076	0.630854	0.000008	0.193972
Random	0.000811	12.490760	0.000831	0.498211	0.000014	0.636542	0.000017	0.074976
KNN	0.000828	12.782280	0.000821	0.478027	0.000009	0.372070	0.000007	0.019466
Centred KNN	0.000835	12.342325	0.000834	0.471252	0.000019	0.553207	0.000019	0.028891
KNN Baseline	0.000806	13.305759	0.000805	0.486257	0.000012	0.648463	0.000020	0.017886
Co-clustering	0.000793	12.845628	0.000801	0.467534	0.000014	0.782792	0.000012	0.022713
Slope One	0.000829	13.280002	0.001018	0.501125	0.000013	0.629301	0.000140	0.072633
SVD	0.000811	13.438938	0.000833	0.524837	0.000015	0.619231	0.000029	0.091137
NMF	0.000840	13.330891	0.000817	0.508564	0.000040	0.503423	0.000009	0.027130

stopping strategy based on the variance of the resource (i.e., time, memory) measurements across samples. However, we can assess the quality of the predictions on a given upper sample size S% by quantifying the *prediction error interval* in linear models and *uncertainty* in the Bayesian models as presented in §3.3. As the predictors get more accurate, the prediction error interval and the uncertainty decrease/shrink; hence, we could stop sampling the dataset and training the models. In our experiments, we noticed that an upper sample size of 30-40% produces good estimations for the processing time and memory usage. From figures 3a, 3b, and 3c we can see that sampling and training our best predictor (LR) on a sample size of 30-40% of the entire collection is much cheaper than training the CF models on the complete datasets.

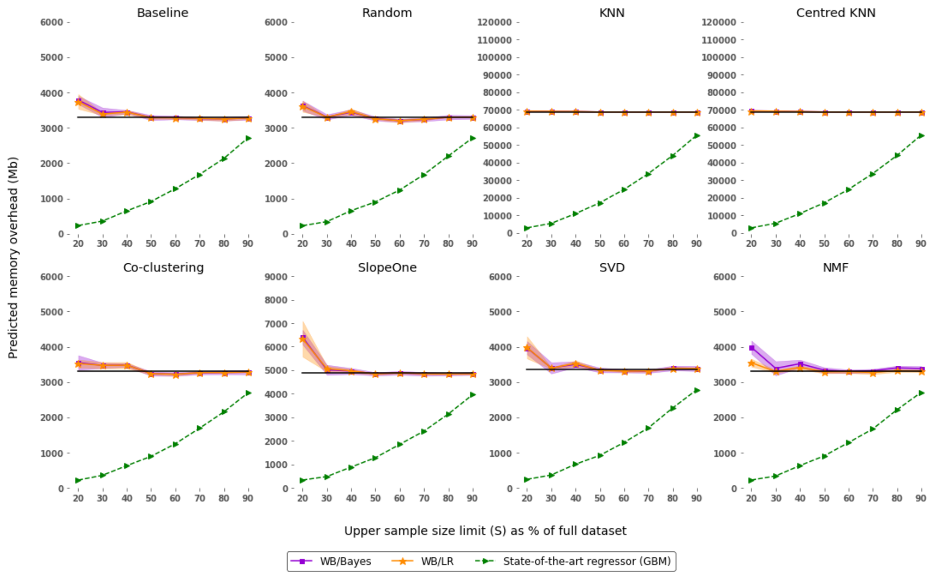


(a) MovieLens 1M



(b) MovieLens 20M

To fully assess the performance of a CF algorithm on a given input, we need to examine both efficiency and effectiveness. The results above demonstrated how the proposed sampling strategy and models could address the former (i.e., efficiency). For the latter (i.e., effectiveness), we employ state-of-the-art regression models to learn the quality of the recommendations given a CF algorithm, a collection of samples (drawn using the strategy in §3.4), and characteristics of the input. Figures 5a, 5b, and 5c show the predicted recommendation effectiveness for the popular CF algorithms implemented in Surprise [45]. We note that a sample size of 30-40% (same size as for efficiency



(c) GoodBooks 10K

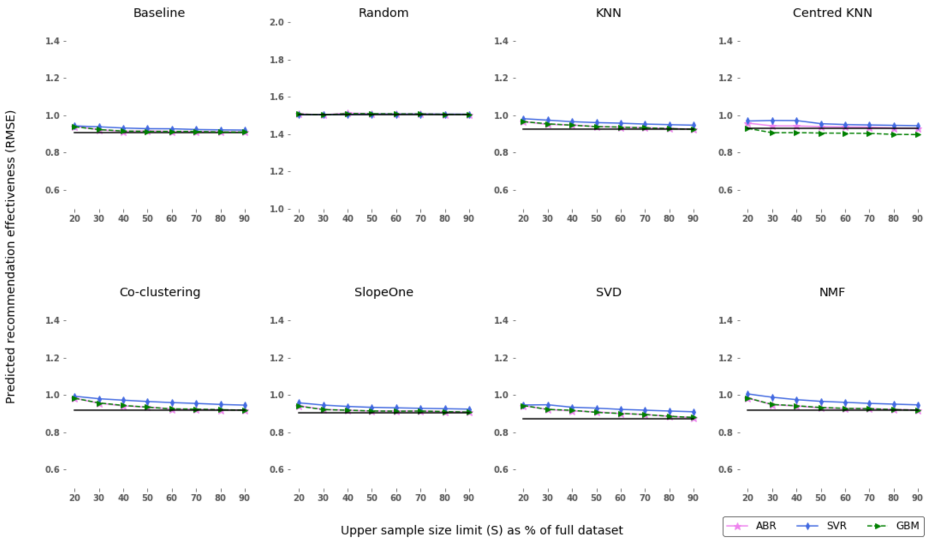
Fig. 4. Predicted memory usage for the CF algorithms on the full (a) MovieLens 1M, (b) MovieLens 20M, and (c) GoodBooks 10K datasets using our approaches (WB/LR and WB/Bayes) and the GBM baseline. The black horizontal line represents the actual memory usage over the entire collection.

models) attains good accuracy for predicting the effectiveness of a CF model. Consequently, our tool and methodology allow one to draw a single set of samples for predicting both the effectiveness and efficiency of a CF in one.

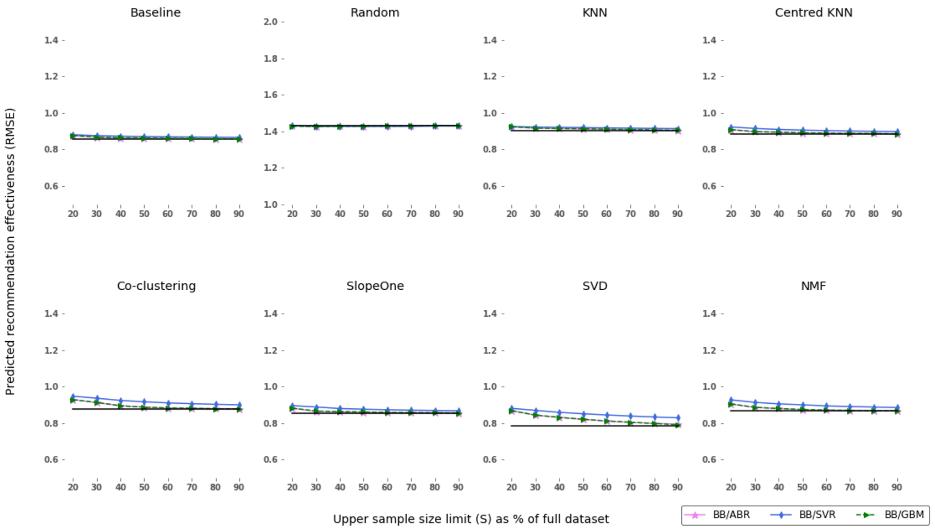
As presented in table 2, we also analysed the efficiency-effectiveness trade-offs emerging from the training cost on various sample sizes compared to the normalised RMSE for the estimated processing times. For the selected CF algorithms, we measured the training cost in terms of time (seconds) and power consumption (kWh), which combined can indicate the monetary cost quantified in US dollars ($). Then, this cost is compared to the normalised RMSE values obtained from predicting the processing time of the entire dataset using our proposed methods (WB/LR and WB/Bayes), as well as the hard (WB/Lit/LR) and soft (GBM) baselines. This study was conducted on GoodBooks 10K, as we believe it is a good case of when a bigger sample that is more expensive does not necessarily improve the accuracy of the predictions.

For example, let us examine the baseline CF model. As the sample size increases and therefore the training cost, the prediction error decreases, leading to more accurate expected processing times. However, not all CF algorithms display this behaviour. For SVD, a bigger and more expensive sample does not necessarily minimise the NRMSE value; this means that while the cost of training increases, the processing time predictions' accuracy remains relatively constant, without further improvements. Therefore, we believe that knowing the training cost of a CF algorithm versus its nominal effectiveness (i.e., the quality of the recommendations) is a critical aspect that should be taken into consideration for building and deploying efficient CF models without sacrificing the users' satisfaction while maximising the content providers' revenue.

Another interesting aspect to examine when choosing a CF algorithm is its effectiveness, as well as training and deployment costs. For the former, benchmarks such as the one from [45]



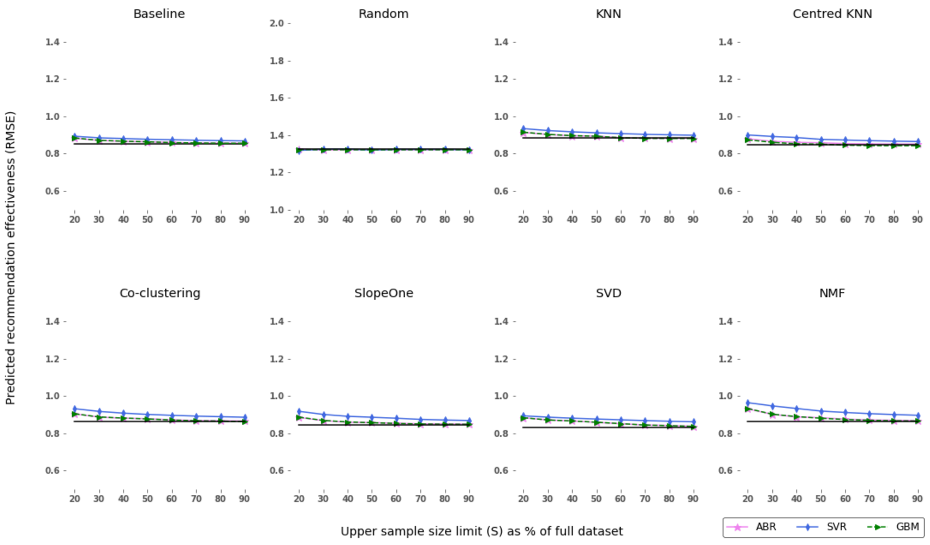
(a) MovieLens 1M



(b) MovieLens 20M

and also our results (i.e., figures 5a, 5b, and 5c) show that different CF models can have similar recommendation quality (e.g., RMSE and MAE for KNN and SVD). However, the training cost is considerably higher for one CF model over the other one. Therefore, when CF algorithm selection represents a critical decision based on the infrastructure available, we propose using our prediction tool, methodology, and sampling strategy to determine if a CF model would be feasible and fit the operational time and memory constraints before allocating and spending many resources for it.

As part of **RQ (c)**, in table 3, we report the average variance ratio (where 0 corresponds to 0% and 1 corresponds to 100%) across samples in different subsets of data for a fixed confidence value of 99% and the constraints from §4.3 (i.e., 10-15% variance in the processing time values). While



(c) GoodBooks 10K

Fig. 5. Predicted effectiveness (i.e., recommendation quality) for the full (a) MovieLens 1M, (b) MovieLens 20M, and (c) GoodBooks 10K datasets using state of the art regressors, such as Ada Boost Regressor (ABR), Support Vector Regressor (SVR), and Gradient Boosting Machine (GBM). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection.

Table 2. Estimated training cost vs. prediction accuracy w.r.t. different sample sizes on GoodBooks 10K.

		Training Resources			**Predictions' Accuracy (NRMSE)**			
	Upper Sample Size S%	*Time (s)*	*Power (kWh)*	*Cost ($)*	*WB/LR*	*WB/Bayes*	*WB/Lit/LR*	*GBM*
Baseline	20	3.5	0.13	0.02	0.16	13.59	12.35	19.98
	40	16.67	0.6	0.07	0.01	4.25	7.03	16.25
	60	37.13	1.34	0.16	0.06	0.01	3.98	10.23
Random	20	1.86	0.07	0.01	0.05	0.02	5.55	9.57
	40	7.91	0.29	0.03	0.05	0.01	3.18	7.52
	60	17.67	0.64	0.08	0.03	0.009	1.69	4.92
KNN	20	64.45	2.33	0.28	0.39	0.05	840.67	1453.77
	40	455.25	16.44	1.97	28.39	26.79	521.28	1344.44
	60	1729.19	62.44	7.49	0.3	0.06	187.97	1000.23
Centred KNN	20	76.39	2.76	0.33	10.5	11.75	855.98	1582.24
	40	496.68	17.94	2.15	30.73	23.26	561.38	1453.82
	60	1595.96	57.63	6.92	35.8	33.72	333.77	1153.67
KNN Baseline	20	83.38	3.01	0.36	228.02	247.41	569.47	1330.45
	40	482.86	17.44	2.09	0.59	0.01	390.03	1185.3
	60	1629.22	58.83	7.06	4.49	4.65	141.52	943.11
Co-clustering	20	29.7	1.07	0.13	0.19	0.01	80.49	140.93
	40	119.89	4.33	0.52	0.55	0.11	46.56	115.53
	60	269.85	9.74	1.17	0.29	0.01	24.39	71.38
Slope One	20	3.62	0.13	0.02	466.88	393.51	57.44	95.66
	40	21.84	0.79	0.09	29.81	19.59	42.01	89.85
	60	78.93	2.85	0.34	5.73	2.14	24.37	74.62
SVD	20	56.98	2.06	0.25	0.32	0.04	0.25	277.31
	40	226.85	8.19	0.98	2.09	1.20	0.39	228.91
	60	545.71	19.71	2.37	0.1	0.21	0.71	127.48
NMF	20	62.34	2.25	0.27	10.78	0.55	10.49	288.54
	40	244.28	8.82	1.06	18.87	0.36	18.71	235.87
	60	561.34	20.27	2.43	14.97	0.002	14.9	144.11

most algorithms seem to achieve a low variance in the processing times from the 20% subset of data on MovieLens 100K, we can observe that for the 10% partition, which also corresponds to the smallest input, some CF, such as KNN, KNN Baseline, and Baseline are less stable. This was expected since these algorithms were the fastest on very small inputs, and the recorded training times were as small as 0.78 milliseconds (KNN). However, as the size of the input data increases, the variance across samples decreases, resulting in steadier measurements. This behaviour was also observed on the larger datasets (ML 1M and GB 10K), which yielded similar results (i.e. 5% variance, or 0.05 in table 3, starting with the 10% data split); these are not included for space considerations.

Table 3. Average variance across 5 samples for various values of S over the MovieLens 100K dataset.

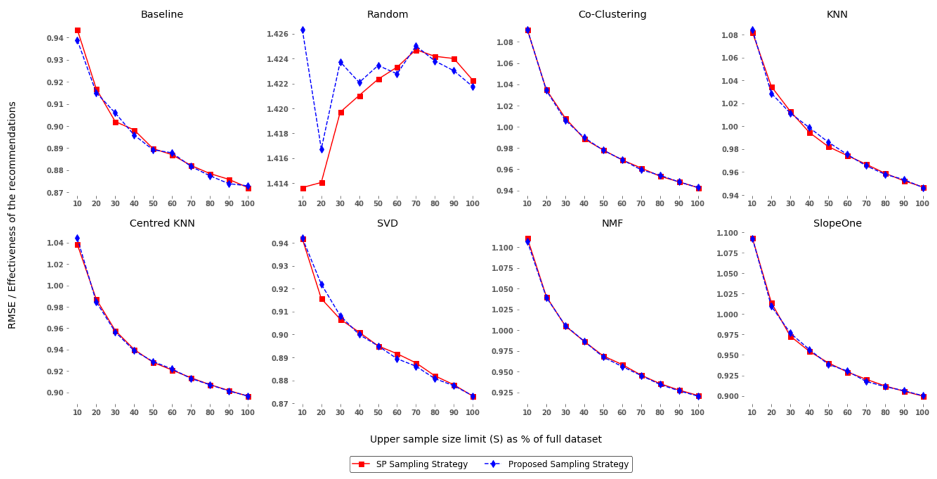
S %	Baseline	Centred KNN	Random	Co-clustering	KNN	KNN Baseline	NMF	Slope One	SVD
10	0.250706	0.027964	0.067672	0.044952	0.966998	0.430209	0.048225	0.097964	0.055360
20	0.049200	0.018693	0.040723	0.033381	0.173672	0.043722	0.035976	0.090190	0.038803
30	0.032348	0.018011	0.029437	0.026288	0.044634	0.049973	0.028211	0.084882	0.029211
40	0.026480	0.017744	0.025777	0.022463	0.027991	0.026873	0.023757	0.062660	0.030115
50	0.019236	0.014401	0.018921	0.017123	0.023000	0.018624	0.017933	0.060335	0.018390
60	0.015513	0.014273	0.016325	0.025878	0.018855	0.018155	0.014537	0.054240	0.015502
70	0.012500	0.012400	0.011676	0.016157	0.016994	0.015884	0.013080	0.032698	0.015423
80	0.010277	0.011007	0.010337	0.009992	0.011521	0.010502	0.009411	0.026711	0.010177
90	0.006314	0.010413	0.006731	0.006102	0.009067	0.015581	0.006380	0.015045	0.005691
100	0.025653	0.008101	0.002862	0.005187	0.050106	0.017862	0.005243	0.006064	0.006068

For **RQ (d)**, we firstly investigated if the standard practice sampling strategy can be employed to draw samples that capture both the efficiency and effectiveness of CF on a subset of the data. Figures 6a and 6b present the raw measured effectiveness of a CF algorithm on an upper sample S% that has been chosen using (i) standard practice sampling [1, 24] and (ii) the proposed sampling mechanism (§3.4). Interestingly, the two sampling strategies have a similar performance regarding the accuracy of the produced recommendations. We note that for the random algorithm, it is expected to have different RMSEs every time a sample is drawn, using either strategy, since the recommendations are produced at random. Thus, for effectiveness, we conclude that both strategies can be successfully used with similar performance.

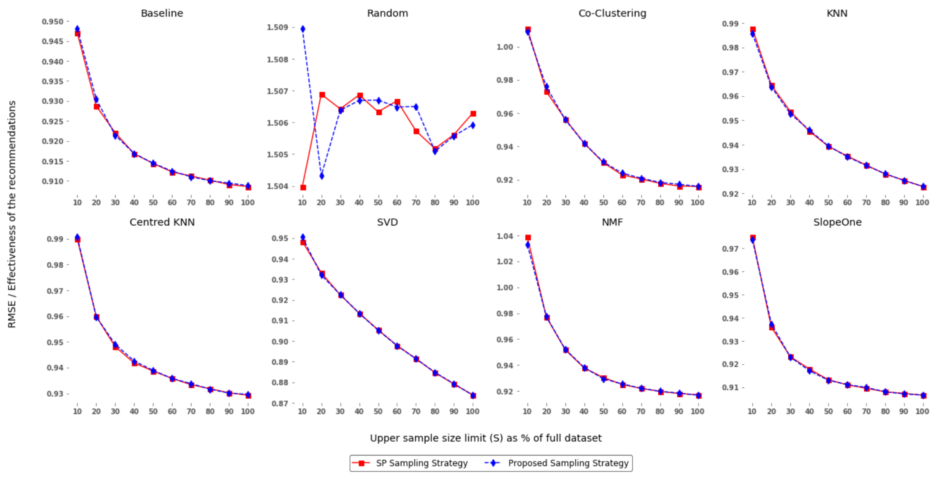
However, if we examine the two sampling strategies to predict the efficiency (e.g., processing time) of a CF model, more notable changes are observed. Figures 7a and 7b show the performance of the prediction models, which have been trained on samples drawn using the two strategies. Our results indicate that the standard practice sampling strategy fails to capture the complexity characteristics of the base data, as the prediction models (non-dashed curves) are far away from the ground truth (black horizontal line). On the other hand, the models that have been trained using the proposed sampling approach (dashed curves) are a lot more accurate in predicting the efficiency of the CF algorithms. Consequently, since our adaptive sampling algorithm (§3.4) works well for both efficiency and effectiveness prediction, it has been used throughout the experimental evaluation.

5 CONCLUSIONS

The accurate prediction of the resource consumption during the training phase of CF models is of exceptional practical interest to establishments of all sizes from both academia and industry. However, so far, the relevant literature does not capture this pressing problem. This paper addresses the challenge of predicting processing time and memory overhead of CF algorithms using a simple yet highly effective approach. This incorporates a fit-for-purpose sampling scheme and a fast but accurate linear regression scheme over time and space complexity equations drawn from the



(a) MovieLens 100K

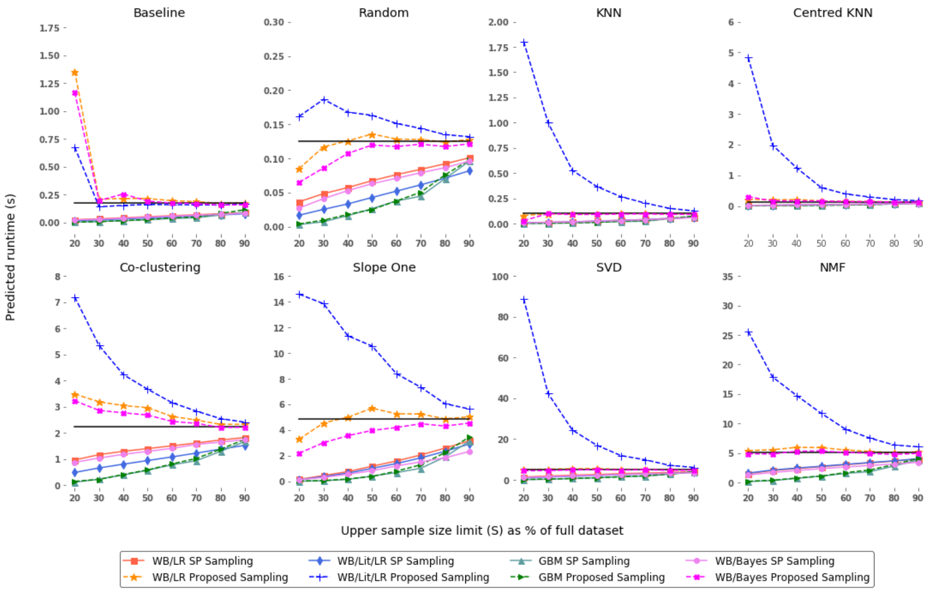


(b) MovieLens 1M

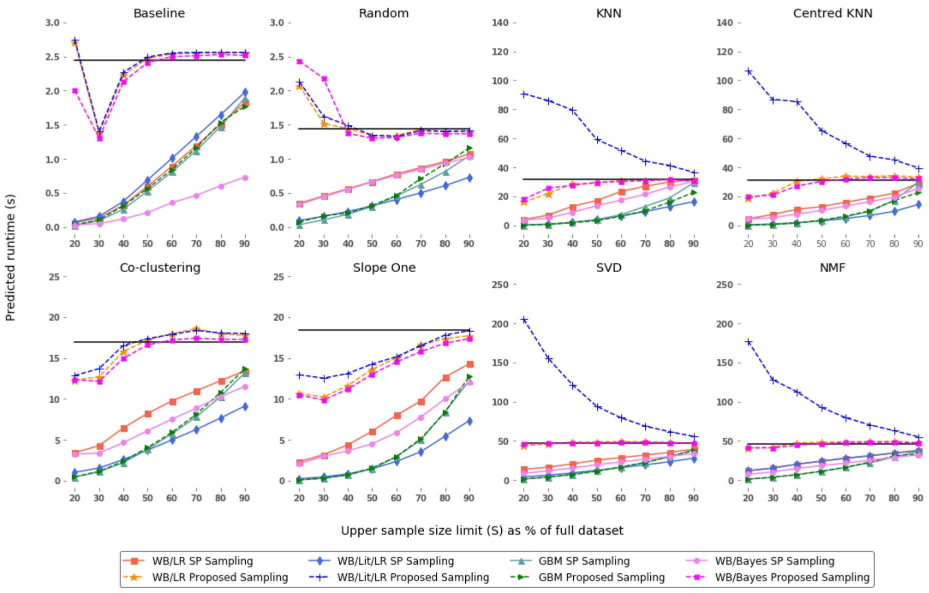
Fig. 6. Actual CF effectiveness (i.e., quality of the produced recommendations) for the full (a) MovieLens 100K and (b) MovieLens 1M datasets using the standard practice (SP) strategy (red curve) compared to our proposed adaptive sampling approach (blue curve).

algorithms' implementations. Furthermore, we showed that using a smaller sample of a dataset, the CF models' performance and resource cost could be estimated with our methodology without training on the entire collection. Our sampling strategy also allows the prediction of both efficiency and effectiveness while paying the cost of sampling only once.

Despite its simplicity, our proposed methodology and resource cost models for CF algorithms manage to considerably outperform in accuracy even the best performing off-the-shelf state-of-the-art regressor. Moreover, our approach is also faster than all contenders, including the best state-of-the-art regressor, and utilises only a fraction of the resources used by them. We view this work as one of the first core steps towards a systematic exploration of the efficiency-effectiveness trade-offs inherent in modern recommendation systems. While our methodology was developed and probed



(a) MovieLens 100K



(b) MovieLens 1M

Fig. 7. Predicted processing times for the full (a) MovieLens 100K and (b) MovieLens 1M datasets using our approaches (WB/LR and WB/Bayes) and the 2 baselines (WB/Lit/LR and GBM). Each prediction model has been trained and tested against the two sampling strategies w.r.t. ratings/interactions (i.e., standard practice (SP) sampling), as well as users and items (i.e., proposed sampling).

with CF models, we believe it could be applied and used for other classes of algorithms and use cases. In the near future we plan to expand our methodology to other areas of the recommendation systems' design space, such as deep learning models, and estimate the training cost for other resources (e.g., GPU usage).

REFERENCES

[1] Gediminas Adomavicius and Jingjing Zhang. 2012. Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems (TMIS)* 3, 1 (2012), 1–17.

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

[3] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach.* Cambridge University Press, Cambridge, UK.

[4] Homer W Austin. 1983. Sample size: How much is enough? *Quality and Quantity* 17, 3 (1983), 239–245.

[5] Mariette Awad and Rahul Khanna. 2015. Support vector regression. In *Efficient learning machines*. Springer, Berlin, Heidelberg, 67–80.

[6] Thomas Bartz-Beielstein and Sandor Markon. 2004. Tuning search algorithms for real-world applications: A regression tree based approach. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, Vol. 1. IEEE, Portland, USA, 1111–1118.

[7] Moran Beladev, Lior Rokach, and Bracha Shapira. 2016. Recommender systems for product bundling. *Knowledge-Based Systems* 111 (2016), 193–206.

[8] Alejandro Bellogín and Pablo Castells. 2010. A performance prediction approach to enhance collaborative filtering performance. In *European Conference on Information Retrieval*. Springer, Berlin, Heidelberg, 382–393.

[9] Alejandro Bellogín, Pablo Castells, and Iván Cantador. 2017. Statistical biases in Information Retrieval metrics for recommender systems. *Information Retrieval Journal* 20, 6 (2017), 606–634.

[10] Ake Bjorck. 1996. *Numerical methods for least squares problems*. Vol. 51. Siam, Philadelphia, USA.

[11] Eric A Brewer. 1995. High-level optimization via automated statistical modeling. *ACM SIGPLAN Notices* 30, 8 (1995), 80–91.

[12] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. 2011. *Handbook of markov chain monte carlo.* CRC press, Florida, United States.

[13] Dinh-Mao Bui, YongIk Yoon, Eui-Nam Huh, SungIk Jun, and Sungyoung Lee. 2017. Energy efficiency for cloud computing system based on predictive optimization. *J. Parallel and Distrib. Comput.* 102 (2017), 103–114.

[14] Laurent Bulteau, Vincent Froese, Sepp Hartung, and Rolf Niedermeier. 2016. Co-clustering under the maximum norm. *Algorithms* 9, 1 (2016), 17.

[15] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. 2011. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web (TWEB)* 5, 1 (2011), 1–33.

[16] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. 2011. Comparison of Collaborative Filtering Algorithms: Limitations of Current Techniques and Proposals for Scalable, High-Performance Recommender Systems. *ACM Trans. Web* 5, 1, Article Article 2 (Feb. 2011), 33 pages. https://doi.org/10.1145/1921591.1921593

[17] Rocío Cañamares and Pablo Castells. 2018. Should I follow the crowd? A probabilistic analysis of the effectiveness of popularity in recommender systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, Michigan, United States, 415–424.

[18] Rocío Cañamares and Pablo Castells. 2020. On Target Item Sampling in Offline Recommender System Evaluation. In *Fourteenth ACM Conference on Recommender Systems*. ACM, Rio de Janeiro, Brazil, 259–268.

[19] Rocío Cañamares, Pablo Castells, and Alistair Moffat. 2020. Offline evaluation options for recommender systems. *Information Retrieval Journal* 23, 4 (2020), 387–410.

[20] Bradley P Carlin and Thomas A Louis. 2008. *Bayesian methods for data analysis.* CRC Press, Florida, United States.

[21] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1998. Random sampling for histogram construction: How much is enough? *ACM SIGMOD Record* 27, 2 (1998), 436–447.

[22] Alan Kaylor Cline and Inderjit S Dhillon. 2007. Computation of the singular value decomposition. In *Handbook of Linear Algebra*, Leslie Hogben (Ed.). Chapman & Hall/CRC, Boca Raton, FL, USA, Chapter 45, 45–1–45–13.

[23] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press, Cambridge, Massachusetts.

[24] Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, and Felice Antonio Merra. 2020. How Dataset Characteristics Affect the Robustness of Collaborative Recommendation Models. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Xi'an, China, 951–960.

[25] Christian Desrosiers and George Karypis. 2011. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*. Springer, Boston, MA, USA, 107–144.

[26] Dave Dice and Alex Kogan. 2021. Optimizing Inference Performance of Transformers on CPUs. In *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys)*. ACM, Edinburgh, United Kingdom, 1–8.

[27] Norman R Draper and Harry Smith. 1998. *Applied regression analysis*. Vol. 326. John Wiley & Sons, Hoboken, NJ, United States.

[28] Eugene Fink. 1998. How to Solve It Automatically: Selection Among Problem Solving Methods.. In *AIPS*. AAAI, Wisconsin, USA, 128–136.

[29] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[30] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1 (10). Springer series in statistics, New York.

[31] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. 2013. *Bayesian data analysis*. CRC press, Florida, United States.

[32] Thomas George and Srujana Merugu. 2005. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, Houston, Texas, 4–pp.

[33] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. 2002. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems (TODS)* 27, 3 (2002), 261–298.

[34] Corrado Gini. 1921. Measurement of inequality of incomes. *The economic journal* 31, 121 (1921), 124–126.

[35] Asela Gunawardana and Guy Shani. 2015. Evaluating recommender systems. In *Recommender systems handbook*. Springer, Boston, MA, 265–308.

[36] Peter J Haas and Arun N Swami. 1992. Sequential sampling procedures for query size estimation. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*. ACM New York, NY, USA, 341–350.

[37] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[38] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, Berkeley, CA, USA, 230–237.

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

[40] Wen-Chi Hou, Gultekin Ozsoyoglu, and Erdogan Dogdu. 1991. Error-constrained COUNT query evaluation in relational databases. *ACM SIGMOD Record* 20, 2 (1991), 278–287.

[41] Adele E Howe, Eric Dahlman, Christopher Hansen, Michael Scheetz, and Anneliese Von Mayrhauser. 1999. Exploiting competitive planner performance. In *European Conference on Planning*. Springer, Berlin, Heidelberg, 62–72.

[42] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, Pisa, Italy, 263–272.

[43] Jing Huang, Renfa Li, Jiyao An, Derrick Ntalasha, Fan Yang, and Keqin Li. 2017. Energy-efficient resource utilization for heterogeneous embedded computing systems. *IEEE Trans. Comput.* 66, 9 (2017), 1518–1531.

[44] Zan Huang, Daniel Zeng, and Hsinchun Chen. 2007. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems* 22, 5 (2007), 68–78.

[45] Nicolas Hug. 2020. Surprise: A Python library for recommender systems. *Journal of Open Source Software* 5, 52 (2020), 2174. https://doi.org/10.21105/joss.02174

[46] Yehuda Koren. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4, 1 (2010), 1–24.

[47] Shyong K Lam, Adam LaPitz, George Karypis, John Riedl, et al. 2006. Towards a scalable kNN CF algorithm: Exploring effective applications of clustering. In *International Workshop on Knowledge Discovery on the Web*. Springer, Berlin, Heidelberg, 147–166.

[48] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration. *Data Science and Engineering* 6, 1 (2021), 1–16.

[49] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable Automatic Machine Learning. In *7th ICML Workshop on Automated Machine Learning (AutoML)*. ICML, Vienna, Austria, 1–16. https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[50] Viktor Leis and Maximilian Kuschewski. 2021. Towards cost-optimal query processing in the cloud. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1606–1612.

[51] Daniel Lemire and Anna Maclachlan. 2005. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, Newport Beach, California, 471–475.

[52] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *International Conference on Principles and Practice of Constraint Programming*. Springer, Ithaca, NY, USA, 556–572.

[53] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1–12.

[54] Ting-Peng Liang, Hung-Jen Lai, and Yi-Cheng Ku. 2006. Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings. *Journal of Management Information Systems* 23, 3 (2006), 45–70.

[55] Richard J Lipton and Jeffrey F Naughton. 1995. Query size estimation by adaptive sampling. *J. Comput. System Sci.* 51, 1 (1995), 18–25.

[56] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. 2019. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM conference on recommender systems*. ACM, Copenhagen, Denmark, 462–466.

[57] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284.

[58] Alexandra L'heureux, Katarina Grolinger, Hany F Elyamany, and Miriam AM Capretz. 2017. Machine learning with big data: Challenges and approaches. *Ieee Access* 5 (2017), 7776–7797.

[59] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1999. Random sampling techniques for space efficient online computation of order statistics of large datasets. *ACM SIGMOD Record* 28, 2 (1999), 251–262.

[60] William Q Meeker, Gerald J Hahn, and Luis A Escobar. 2017. *Statistical intervals: a guide for practitioners and researchers.* Vol. 541. John Wiley & Sons, Hoboken, NJ, United States.

[61] Pierre Moulin and Venugopal V. Veeravalli. 2018. Maximum Likelihood Estimation. In *Statistical Inference for Engineers and Data Scientists*. Cambridge University Press, Cambridge, 319–357. https://doi.org/10.1017/9781107185920.017

[62] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective.* MIT press, London, UK.

[63] Iulia Paun. 2020. Efficiency-Effectiveness Trade-offs in Recommendation Systems. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys '20)*. ACM, Rio de Janeiro, Brazil, 770–775.

[64] Iulia Paun, Yashar Moshfeghi, and Nikos Ntarmos. 2021. Are we there yet? Estimating Training Time for Recommendation Systems. In *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys)*. ACM, Edinburgh, United Kingdom, 1–9.

[65] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the third ACM symposium on cloud computing*. ACM, San Jose, CA, United States, 1–13.

[66] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2010. *Recommender Systems Handbook* (1st ed.). Springer, Boston, Massachusetts.

[67] Enda Ridge and Daniel Kudenko. 2007. Tuning the performance of the MMAS heuristic. In *International Workshop on Engineering Stochastic Local Search Algorithms*. Springer, Berlin, Heidelberg, 46–60.

[68] Mark Roberts, Adele Howe, and Landon Flom. 2007. Learned models of performance for many planners. In *ICAPS 2007 Workshop AI Planning and Learning*. ICAPS, Rhode Island, USA, 36–40.

[69] Laurens Rook, Adem Sabic, and Markus Zanker. 2020. Engagement in proactive recommendations. *Journal of Intelligent Information Systems* 54, 1 (2020), 79–100.

[70] Scikit-learn.org. 2020. Stochastic gradient descent 0.23.0 documentation. https://scikit-learn.org/stable/modules/sgd.html#complexity.

[71] Alexander Shapiro. 2003. Monte Carlo sampling methods. *Handbooks in operations research and management science* 10 (2003), 353–425.

[72] Upendra Shardanand and Pattie Maes. 1995. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM New York, Denver, Colorado, 210–217.

[73] Durga L Shrestha and Dimitri P Solomatine. 2006. Machine learning approaches for estimation of prediction interval for the model output. *Neural Networks* 19, 2 (2006), 225–235.

[74] Brent Smith and Greg Linden. 2017. Two decades of recommender systems at Amazon. com. *Ieee internet computing* 21, 3 (2017), 12–18.

[75] Ralph C Smith. 2013. *Uncertainty quantification: theory, implementation, and applications.* Vol. 12. Siam, Philadelphia, United States.

[76] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. *CoRR* abs/1906.02243 (2019), 1–6. arXiv:1906.02243 http://arxiv.org/abs/1906.02243

[77] Xiaoyang Sun, Chunming Hu, Renyu Yang, Peter Garraghan, Tianyu Wo, Jie Xu, Jianyong Zhu, and Chao Li. 2018. Rose: Cluster resource scheduling via speculative over-subscription. In *2018 IEEE 38th International Conference on*

Distributed Computing Systems (ICDCS). IEEE, Vienna, Austria, 949–960.

[78] Zilei Sun, Nianlong Luo, and Wei Kuang. 2011. One real-time personalized recommendation systems based on slope one algorithm. In *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Vol. 3. IEEE, Shanghai, China, 1826–1830.

[79] Zhiqiang Tan. 2006. Monte Carlo integration with acceptance-rejection. *Journal of Computational and Graphical Statistics* 15, 3 (2006), 735–752.

[80] Christopher Tosh and Sanjoy Dasgupta. 2019. The relative complexity of maximum likelihood estimation, map estimation, and sampling. In *Conference on Learning Theory*. PMLR, Phoenix, United States, 2993–3035.

[81] Vaishali S Vairale and Samiksha Shukla. 2021. Recommendation of Food Items for Thyroid Patients Using Content-Based KNN Method. In *Data Science and Security*. Springer, Berlin, Heidelberg, 71–77.

[82] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, Chicago, US, 1009–1024.

[83] Zhongya Wang, Ying Liu, and Pengshan Ma. 2014. A CUDA-enabled parallel implementation of collaborative filtering. *Procedia Computer Science* 30 (2014), 66–74.

[84] Ling-Ling Wu, Yuh-Jzer Joung, and Jonglin Lee. 2013. Recommendation systems and consumer satisfaction online: moderating effects of consumer product awareness. In *2013 46th Hawaii International Conference on System Sciences*. IEEE, Wailea, Maui, HI, United States, 2753–2762.

[85] Longqi Yang, Eugene Bagdasaryan, Joshua Gruenstein, Cheng-Kang Hsieh, and Deborah Estrin. 2018. Openrec: A modular framework for extensible and adaptable recommendation algorithms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, Los Angeles, CA, United States, 664–672.

[86] Michael Yeomans, Anuj Shah, Sendhil Mullainathan, and Jon Kleinberg. 2019. Making sense of recommendations. *Journal of Behavioral Decision Making* 32, 4 (2019), 403–414.

[87] Gingfung Yeung, Damian Borowiec, Adrian Friday, Richard Harper, and Peter Garraghan. 2020. Towards {GPU} Utilization Prediction for Cloud Deep Learning. In *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX, Online, 1–9.

[88] Zygmunt Zajac. 2017. Goodbooks-10k: a new dataset for book recommendations. http://fastml.com/goodbooks-10k.

[89] Heng-Ru Zhang, Fan Min, Zhi-Heng Zhang, and Song Wang. 2019. Efficient collaborative filtering recommendations with multi-channel feature vectors. *International Journal of Machine Learning and Cybernetics* 10, 5 (2019), 1165–1172.