Are we there yet? Estimating Training Time for Recommendation Systems

Iulia Paun
School of Computing Science
University of Glasgow
Glasgow, Scotland, UK
iulia.paun@glasgow.ac.uk

Yashar Moshfeghi
Department of Computer and Information Sciences
University of Strathclyde
Glasgow, Scotland, UK
yashar.moshfeghi@strath.ac.uk

Nikos Ntarmos
School of Computing Science
University of Glasgow
Glasgow, Scotland, UK
nikos.ntarmos@glasgow.ac.uk

Abstract

Recommendation systems (RS) are a key component of modern commercial platforms, with Collaborative Filtering (CF) based RSs being the centrepiece. Relevant research has long focused on measuring and improving the effectiveness of such CF systems, but alas their efficiency – especially with regards to their time- and resource-consuming training phase – has received little to no attention. This work is a first step in the direction of addressing this gap. To do so, we first perform a methodical study of the computational complexity of the training phase for a number of highly popular CF-based RSs, including approaches based on matrix factorisation, k-nearest neighbours, co-clustering, and slope one schemes. Based on this, we then build a simple yet effective predictor that, given a small sample of a dataset, is able to predict training times over the complete dataset. Our systematic experimental evaluation shows that our approach outperforms state-of-the-art regression schemes by a considerable margin.

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

Keywords: Recommendation systems, sampling-based processing time prediction.

1 Introduction

A key aspect of modern recommendation systems is delivering relevant content for users, while maximising the revenue for providers. Capturing users' preferences and producing personalised recommendations has been deemed as one of the most effective techniques to keep the targeted audience engaged, which in turn, means bigger gains for the recommendation platforms and content providers. An important class of RS models are Collaborative Filtering (CF) based systems, which recommend items to a user based on similar users' preferences. Consequently, this aids users in the item selection process, thus alleviating the information overload problem. CF models[1] often rely on explicit feedback datasets, where the users' tastes are captured through exact 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). One of the core decisions to be made when one builds a recommendation system, is the selection of algorithm to be used.

Focusing only on the nominal accuracy of mainstream algorithms is often a fallacy, as there is an inherent trade-off between the efficiency (processing time) of the system and the effectiveness (accuracy/quality) of the recommendations. Furthermore, we would like to highlight that in CF approaches, it is the training efficiency (i.e., processing time for the training phase) of the RS that could make-or-break its usability [29, 32], also demonstrated in our experimental evaluation (§4.5), as the models are trained over large collections. Once the CF models are trained, the cost (in terms of processing time) of producing recommendations for users is minimal (i.e., the recommendations are produced almost instantly, compared to the training time). While there has been a significant body of research focusing on the latter (effectiveness), the former (efficiency) seems to have been largely overlooked.

Of exceptional practical interest is the highly resource- and time-consuming training stage of CF-based recommendation systems. CF RSs need to be updated (retrained) periodically to reflect the latest information, in the face of continuously growing and highly dynamic user-item interaction data. Choosing which algorithm to train and deploy is critical to strike a balance across quality of recommendations and freshness of the RS model, as well as time and resource consumption during training. For the latter, it is important to have an insight into how many resources a model will consume before deploying it on the full dataset. Note that the training time is several orders of magnitude higher than the time it takes to produce a recommendation given a trained CF RS, and higher training times/resources translate to higher energy/monetary costs, and even to damage to the environment as a result of increased amounts of CO_2 emissions from training complex models on large collections [39]; hence being able to predict the required training time can lead to informed decisions with limited cost. There are also practical extensions as training is a periodic process and such a predictor can help in computing an appropriate re-training

[1] For space reasons, we overload the terminology and use RS to refer to CF-based RSs.

This is a peer-reviewed, accepted author manuscript of the following article: Paun, I., Moshfeghi, Y., & Ntarmos, N. (2021). Are we there yet? Estimating training time for recommendation systems. In Proceedings of the 1st Workshop on Machine Learning and Systems, EuroMLSys 2021: Proceedings of the 1st Workshop on Machine Learning and Systems (pp. 39-47). (Proceedings of the 1st Workshop on Machine Learning and Systems, EuroMLSys 2021). ACM. https://doi.org/10.1145/3437984.3458832

interval or choosing a CF RS that can fit the operational time constraints.[2]

For effectiveness (accuracy) purposes, it is standard practice to train the selected RS on a sample of the dataset and to use its offline measured accuracy as a proxy for the accuracy over the complete dataset [11]. However, no similar approach exists for predicting their training time over the complete dataset. Based on preliminary analysis, we argue that this is a combination of two factors: (a) the samples produced by the 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, even state-of-the-art regressors fail to produce accurate results.

To address the latter, we fist perform a methodical study of the computational complexity of a number of highly popular CF algorithms for explicit datasets, covering a wide area of the design space (including approaches based on singular value decomposition, k-nearest neighbours, slope-one schemes, matrix factorisation, etc. – see §3.1) and build regression models (§3.2) that try to quantify the hidden constant factors. Then, to address the former, we revisit the sampling strategy to ensure the sample is drawn along the key dimensions of computational complexity (§3.3). For lack of a better term, we could call this methodology the white-box performance evaluation, as opposed to the black-box approach of blindly picking samples and building regression models over them. In both cases, our proposed solution is much simpler than the current (black-box) state-of-the-art, yet manages to vastly outperform it in accurately predicting the training time of the CF RSs over the complete dataset, as evidenced by our extensive experimental evaluation.

2 Related Work

The prediction of an algorithms' execution time has been 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 [6]. Other works focused on predicting the processing time of various planning algorithms, with the aim of selecting which algorithm to run and for how long [13, 21, 35]. Predicting the processing times of parameterised algorithms has drawn high interest from the research community, with existing solutions incorporating the parameters as additional inputs for the prediction models [3, 34]. Another area that has been explored consists of applications of processing time prediction, such as determining instance hardness [28] and parameter optimisation/tuning [34]. To our knowledge (and surprise), this is the first work to apply processing time prediction models on the highly time and resource consuming training stage of (CF-based) recommendation systems.

Evaluation is a major area of interest in CF RSs, as numerous studies and projects tried to determine the best metrics and practices in this field. Herlocker et al. [20] offer a survey and critical evaluation of metrics and methods used to assess the effectiveness (accuracy, coverage, novelty, serendipity, etc.) of the recommendations. Over the years, most works [17, 20] only report the quality of the recommendations through effectiveness metrics, but recent studies [29] also present some insights into the observed efficiency (processing time) of the models. Thus, and in the context of environmental awareness [39], 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 why it is very important to be able to predict the efficiency cost of RSs, without performing the actual training of the models.

Regarding online versus offline evaluation, it has been generally agreed that offline assessment should be used as a tool for establishing the overall performance of a recommendation system [17, 20]. Traditionally, this offline evaluation of the CF RS focuses on splitting the dataset/input into training and testing collections, which are then used to assess the output of the recommendation system. It is known that blindly applying this method allows for the sparsity and popularity biases inherent in the dataset to affect the evaluation protocol [4]. Hence, the standard practice in this field is to draw random samples from the dataset, then use them to train and evaluate the effectiveness of CF RSs as a proxy for their performance on the complete dataset [11]. Consequently, the study of how dataset characteristics affect the quality of the recommendations, as well as their impact on the model's effectiveness, has grown in interest. In [1, 11], the authors explore the effect of the structural properties of the user-item rating matrices with respect to the accuracy and robustness of the collaborative filtering algorithms used in the studies. Their results confirm that there is a relationship between dataset characteristics and the recommendation systems' behaviour, and also highlight the standard practice of using samples to evaluate the effectiveness of RSs, while alleviating the high processing costs of testing on the complete dataset. In [32], we argued that properties of the input data further affect the inherent trade-off between the efficiency and effectiveness of a system, and that the choice of RS should be based on the latter as well. This work is a step in the direction of addressing this gap, for a set of highly popular and impactful CF-based RSs.

3 Predicting RS Training Time

When faced with the task of predicting the effectiveness of a CF RS on a dataset, based on its behaviour on a sample

[2]The above were also confirmed in discussions with industry professionals (e.g., Amazon, Netflix) at the ACM RecSys 2020 Doctoral Symposium [32].

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 RS over the sample, (iii) evaluating its effectiveness over the sample [11]. We follow a similar strategy with a few notable changes. First, what we measure in each such iteration is, of course, the training time. Second, even the best state-of-the-art regression model requires such time measurements over almost the complete dataset to produce semi-accurate predictions (see §4); §3.2 discusses our white-box approach to this task. Last, early experiments (omitted for space reasons) showed that sampling along the ratings dimension fails to capture the characteristics of the dataset that affects the scaling of its training time; §3.3 discusses a simple sampling scheme that alleviates this issue.

Figure 1 presents an overview of the proposed pipeline, and the steps that our users need to follow to estimate the processing time for training a recommendation system on a chosen dataset. In the following paragraphs, we discuss each of the steps in more depth, and how using the expected complexity of a CF algorithm and a sample of the data, can lead to accurate predictions of the runtime.

3.1 Complexity Analysis

Asymptotic worst-case (a.k.a. big-O) complexity analysis [10] determines an upper bound to the way an algorithm's processing time grows or declines as a function of characteristics of its input. The CF RSs studied in this work are based on well-known algorithms, for which big-O analysis has been provided by the relevant literature [7, 9, 22, 36, 40, 42, 43]. 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 [2]. We thus further formulate and propose complexity equations based on the actual implementation of said CF RSs, as found in the highly popular Surprise [23] library (full derivation omitted for space reasons). The analysis is done in terms of characteristics of the input (user-item rating matrix); namely, 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, $\boldsymbol{\delta}(= \frac{\rho}{mn})$. 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 [23].

To cover as wide an area of the design space as possible, we opted to analyse algorithms across several categories of explicit CF RSs; specifically, *(a)* basic algorithms: *Baseline* algorithm derived from [24]; and Maximum Likelihood Estimation based *Random* approach [31]; *(b)* algorithms based on K-nearest neighbours: *Basic KNN* (*KNN*) [19], *KNN* taking into account the mean rating of each user (*Centred KNN*) [12], and *KNN* taking into account a baseline rating (*KNN Baseline*) (formula (3), §2.2 in [24]) - the former two use Mean Squared Difference (MSD) [8] as the distance metric, while the latter uses Pearson correlation coefficients [19] centred using baseline scores; *(c)* variants of matrix factorisation: Non-negative Matrix Factorisation (NMF) [30] and Singular Value Decomposition (SVD) derived from [33]; *(d)* slope-based algorithms: Slope One scheme [27]; and *(e)* co-clustering approaches: algorithm presented in [15].

The **baseline** recommendation system is based on the ALS algorithm, the naive solver[3] version, which has a complexity of $O(mnf)$. If we further fix f to its default/recommended value, the complexity can be further abstracted to $\boldsymbol{O(mn)}$ [22]. However, by examining the implementation of the baseline recommendation algorithm, in each epoch, 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 complexity is $\boldsymbol{O(m^2 + n^2)}$.

The **random** algorithm, based on Maximum Likelihood Estimation (MLE), predicts the missing ratings over a normal distribution, computed in maximum $\boldsymbol{O(mn)}$ steps [42]. The implementation reveals that the random recommendation system 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 complexity is in $\boldsymbol{O(\rho)}$.

For the neighbourhood based 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 $\boldsymbol{O(m^2n^2)}$ [43, 45]. However, at the implementation level, for **KNN** we derived a 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 recommendation algorithms, 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 using Gini coefficients [16], we noticed that users' and items' ratings follow a uniform distribution (i.e., $\frac{\rho}{m}$ for users or $\frac{\rho}{n}$ for items); consequently, 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 or items in m^2 or n^2 time, 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 complexity to $\boldsymbol{O(\frac{\rho^2}{x} + x^2 + \rho)}$. **KNN Baseline** is also based on KNN, and

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

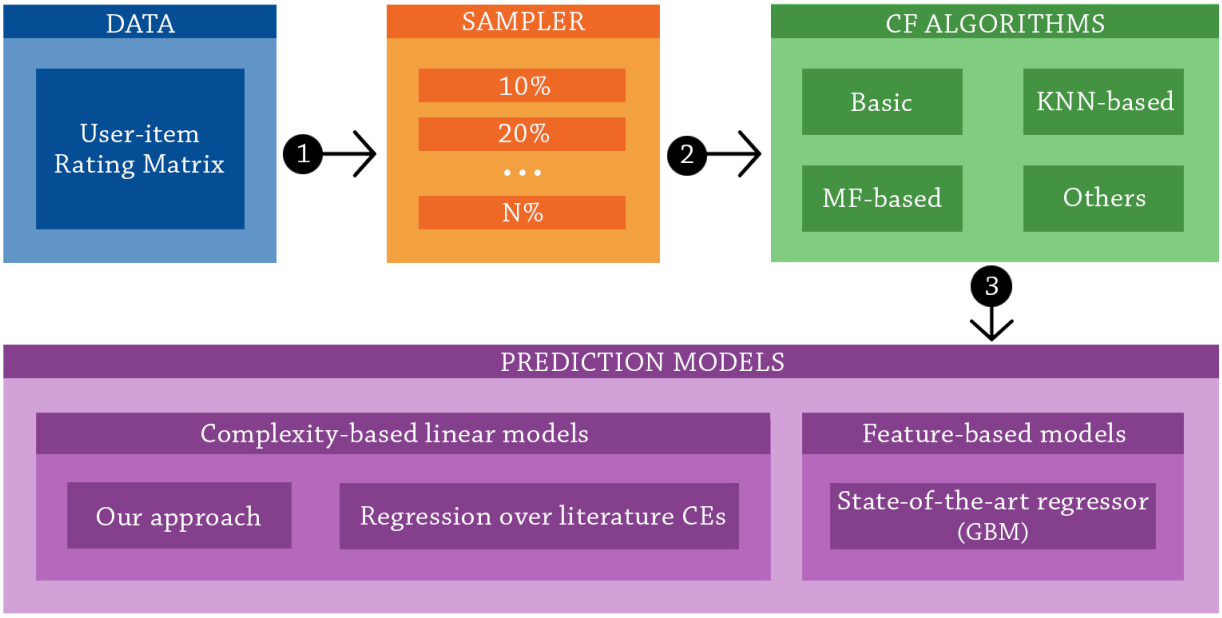


Figure 1. Overview of the experimental pipeline. We firstly get the data, represented as user-item rating matrices (URMs) (step 1), then sample it using the strategy described in §3.3. In step 2, we train the various classes of CF recommendation systems (§3.1) on the samples drawn and measure the processing time for each algorithm. Finally, as part of step 3, we employ prediction models, detailed in §3.2, which given an upper sample S% of the data and characteristics of the URM, estimate the expected processing time of the recommendation system on the full dataset.

computes distances across users (items, respectively) using Pearson correlation coefficients [19], and takes into account baseline ratings. Its overall complexity, as derived from its implementation, is the same as the one for Centred KNN - i.e., $\boldsymbol{O(\frac{\rho^2}{x} + x^2 + \rho)}$.

The **NMF** recommendation system is based on the SGD algorithm, which achieves a computational complexity of $O(em\rho)$ [36]. If we fix the number of epochs, the complexity can be reduced to $\boldsymbol{O(m\rho)}$. In the Surprise framework [23], 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 the number of epochs and factors).

SVD, a popular Matrix Factorisation-based approach, has been extensively used to produce recommendations on explicit datasets. Over time, multiple variations of SVD were developed [9], 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 [25], claim that the overall complexity of SVD is close to $O(n^2m + m^2n)$. The SVD implementation in Surprise [23] uses the user-item ratings matrix in $\boldsymbol{O(e\rho f)}$ time to compute the corresponding user and item factors. For fixed values of e and f, SVD's complexity can be simplified to $\boldsymbol{O(\rho)}$.

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 [40]. 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 complexity of $\boldsymbol{O(\frac{\rho^2}{m} + n^2 + \rho)}$.

Lastly, the **Co-clustering** recommendation system using a fixed number of user-item clusters converges towards a computation complexity of $\boldsymbol{O(mn)}$ [7]. 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.

3.2 Training Time Prediction Models

Using the above algorithmic complexities, the training times measured across different inputs, and the characteristics of the data, we propose solving the processing time prediction problem by building regression models (figure 1 step 3) to predict the hidden factors in the complexity equations mentioned in §3.1. To this end, we propose to abstract

out all previous complexity equations to the general form $time = f(X) = \alpha X + \beta$, where X is a combination of the independent variables m, n and ρ, as dictated by the respective complexity equation, while α and β are the slope and intercept respectively. For example, the equation for *baseline* becomes $\alpha(m^2+n^2)+\beta$, that of *NMF* becomes $\alpha(\rho+m+n)+\beta$, etc. The main reason behind this design is that it reduces the execution time of our prediction model by allowing us to use simple linear regression, while allowing us to capture the salient features of earlier complexity equations, as will be evidenced by our experimental results.

More specifically, the training time measured during each iteration of the process outlined at the beginning of this section, produces a value of $f(X)$ for which we further know m, n and ρ (computed from the sample itself). Given that we then have an overdetermined system, with more sets of equations than unknowns, we construct our models based on the *least squares* approach [5]. This technique is based on minimising the sum of the squares of the residuals (i.e., the difference between the observed/measured values and the predicted/fitted values), by computing appropriate values for the free parameters α and β. At the end of this process we have a closed form equation that is then used to predict the training time over arbitrarily the complete dataset (or virtually any sample size).

3.3 Sampling Strategy

The standard practice with regards to sampling for the purpose of evaluating the effectiveness of an RS, roughly consists of tossing a biased coin for each rating (i.e., triplet of the form $(user, item, rating)$) in the dataset, to decide whether to include it in the sample. Our early experiments (omitted for space reasons) concluded that samples drawn using this method did not allow for accurate prediction of the training time over the complete dataset. More specifically, fitting either our models or state-of-the-art regression models over the training times over these samples, required sample sizes close to the complete dataset. Furthermore, drawing such samples from the dataset is not for free, especially when the number of users/items/ratings grows large.

To this end, we propose the following sampling strategy (figure 1 step 1). Initially, the user of our system provides us with an upper sample size – say S (%) – as well as with a time budget B for our method. 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. We then use a strategy similar to Monte Carlo rejection sampling [41], 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 RS and record its training time (figure 1 step 2); we then decide whether to proceed with more samples given the so-far cumulative execution time of the above process and the time budget B.

4 Experimental Evaluation

4.1 Contenders

We compare our training time prediction models 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 regression model is built over the sample training times and the characteristics of the input. To make the latter model richer, we further included an extra indicator, as advised by [11]; namely, the *Gini* coefficient [16] of users' (items', resp.) ratings, defined as:

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

where w is the number of users (items, resp.), ρ_k is the number of ratings given by a user (or received by an item, resp.), and ρ_{total} is the total number of ratings.

This approach was tested using several off-the-shelf state-of-the-art regression algorithms available through the H2O analytics platform[4]. A few examples of the tested regressors include Random Forest, Deep Neural Networks, 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, namely Gradient Boosting Machine (GBM) [14]. 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 as described in [26].

4.2 Datasets and Recommendation Task

For this study, we used the well-known MovieLens (ML) 100K and 1M collections [18], as well as the GoodBooks (GB) 10K dataset [44] (results for ML 1M omitted due to space reasons). 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). ML 100K contains 610 users, 9,724 items, and 100,000 ratings, while ML 1M incorporates 6,040 users, 3,706 items, and 1,000,000 ratings. Finally, GB 10K has 53,424 users, 10,000 items, and 6,000,000 ratings.

The recommendation task investigated in this paper refers to the standard prediction of the relevance of a given item to a user [37]; in short, the recommendation system estimates the rating the user would give to an unseen item, and if the rating is above a certain threshold value then the item is presented to the user as a recommendation [38].

[4] A complete list of the regression algorithms and the documentation of H2O are available at http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html

4.3 Evaluation Environment

All experiments were carried out on Linux servers, each having 2 Intel Xeon E5-2660 CPUs (8 physical cores each with 2-way SMT) 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 SMT) 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.

4.4 Training Time Measurements and Evaluation

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%). As an exception, the measurements on the KNN-based algorithms over the GoodBooks dataset proved too computationally intensive, and therefore we only used values of S up to 70% of the dataset. We omit an evaluation of the effect of B (time budget) and instead use 6 samples for each (sub)sample size. In each such set of 6 samples, we discarded the measurement for the first sample (to avoid effects of cold caches and overheads of the language runtime), and used the measurements of the remaining 5 samples as the input to the regression algorithms. For space reasons, since KNN Baseline and Centred KNN follow the same complexity and show similar training time behaviour, we only provide results for one of these representatives.

4.5 Results

Figures 2a and 2b provide an overview of the training time prediction models and baselines across the smallest and biggest datasets with respect to the algorithms discussed in §3.1. The black horizontal line represents the ground truth – i.e., the actual average training time of each RS over the complete dataset. The remaining three curves depict the predicted training time over the complete dataset, using sample sizes as defined by the x-axis labels; i.e, the points in these figures should be read as "predicted training time over the complete dataset when upper sample size set to x-axis value".

We can see that in most cases, our proposed model (yellow star curve) converges to an accurate prediction much faster than its two contenders and for as little a value of S as 30-40%. As a baseline, we developed a similar regression model built over the stock (literature) complexity equations (blue cross curve), and as observed, this often fails to produce accurate predictions. Last, our proposed solution clearly outperforms the state-of-the-art GBM regressor (green triangle curve), highlighting the strengths of our methodology and framework towards accurate training time prediction.

5 Conclusions

The accurate prediction of the training time of CF RSs is of exceptional practical interest to establishments of all sizes; yet, it has gone largely unnoticed in the relevant literature. This paper addresses this pressing problem using simple but highly efficient techniques, combining a fit-for-purpose sampling scheme and a fast but accurate linear regression scheme over complexity equations drawn from the algorithms' implementations. Despite its simplicity, our model manages to considerably outperform in accuracy even the best performing off-the-shelf state-of-the-art regressor. We view this work as the first step towards a systematic exploration of the efficiency-effectiveness trade-offs inherent in modern recommendation systems. In the near future we plan to extend our approach to predict usage of other resources (memory footprint, GPU utilisation, etc.) and to expand its applicability to RSs based on deep learning methods.

Acknowledgments

The work is co-funded by the Erasmus+ Programme of the European Union under the PRIMES project (no. 2016-1-UK01-KA201-024631) and by the UK Government through an EPSRC grant (no. EP/N509668/1).

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

[3] 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.

[4] 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.

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

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

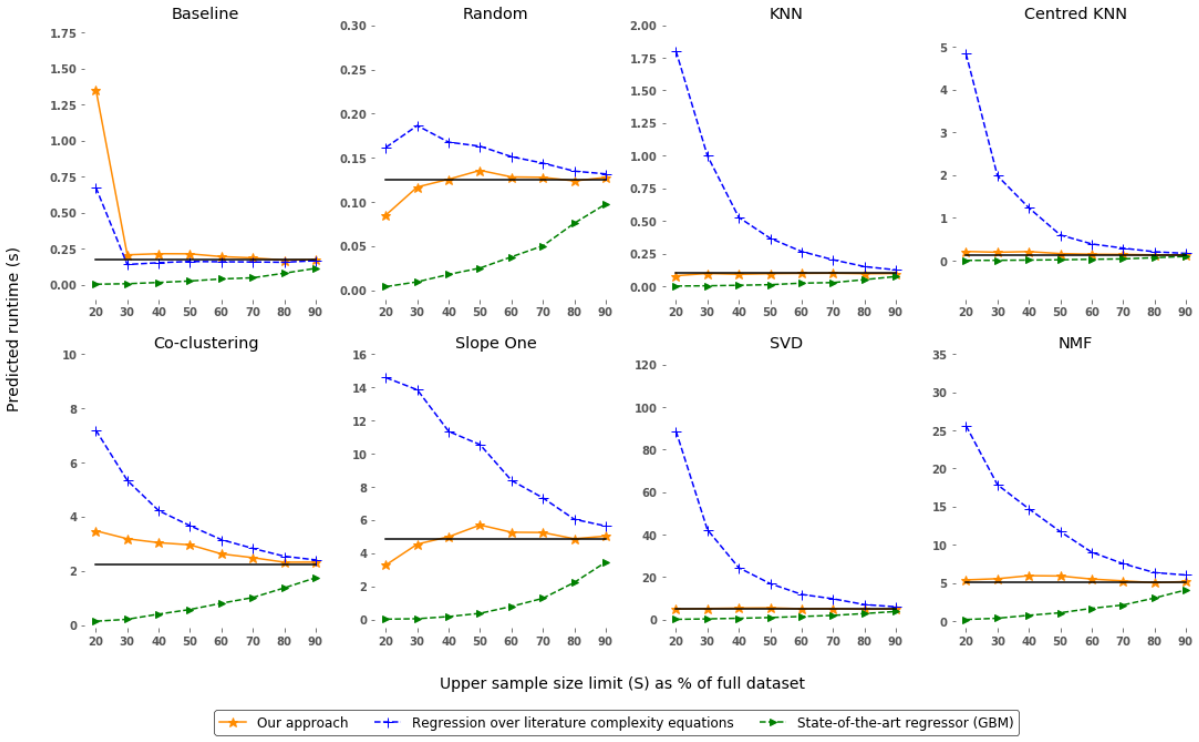
[7] Laurent Bulteau, Vincent Froese, Sepp Hartung, and Rolf Niedermeier. 2016. Co-clustering under the maximum norm. *Algorithms* 9, 1 (2016), 17.

[8] 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 2 (Feb. 2011), 33 pages. https://doi.org/10.1145/1921591.1921593

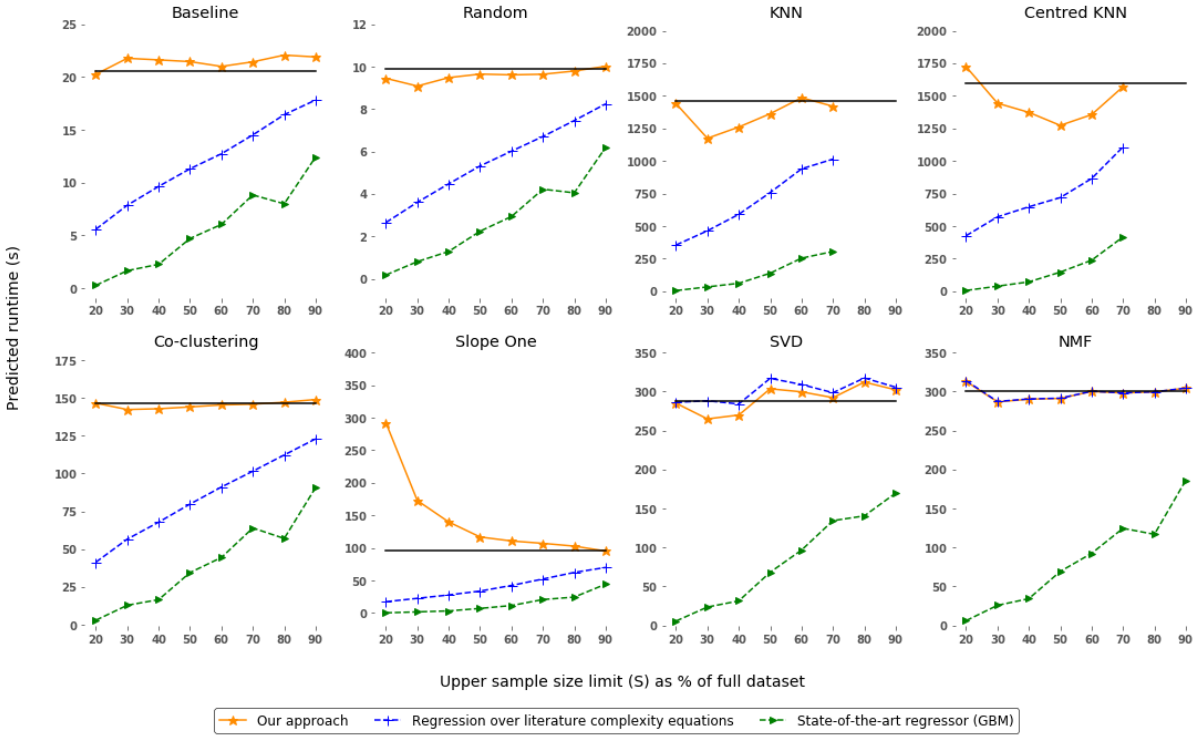
[9] Alan Kaylor Cline and Inderjit S Dhillon. 2007. Computation of the singular value decomposition. In *Handbook of Linear Algebra*, Leslie Hogben (Ed.). Chapman & Hall/CRC, Boca Raton, FL, USA, Chapter 45, 45–1–45–13.

[10] 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.

[11] 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*



(a) MovieLens 100K



(b) GoodBooks 10K

Figure 2. Predicted training times for (a) MovieLens 100K and (b) GoodBooks 10K collections, for various values of S (upper sample size limit), using our approach and the two baselines (linear regression over literature complexity equations; and state-of-the-art regressor (GBM)). The black horizontal line represents the actual training time over the entire dataset.

International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, Xi'an, China, 951–960.

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

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

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

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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

[24] 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.

[25] 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.

[26] 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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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

[32] 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.

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

[34] 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.

[35] 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.

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

[37] 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.

[38] Harald Steck. 2013. Evaluation of recommendations: rating-prediction and ranking. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, Hong Kong, 213–220.

[39] 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

[40] 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.

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

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

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

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

[45] 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.