EFFICIENCY MODELLING IN COLLABORATIVE FILTERING-BASED RECOMMENDATION SYSTEMS

IULIA PAUN (NÉE POPESCU)

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

SCHOOL OF COMPUTING SCIENCE

COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

MAY 2022

Abstract

In the past decade, Machine Learning (ML) models have become a critical part of large scale analytics frameworks to solve different problems, such as identify trends and patterns in the data, manipulate images, classify text, and produce recommendations. For the latter (i.e., produce recommendations), ML frameworks have been extended to incorporate both specific recommendation algorithms (e.g., SlopeOne [1]), but also more generalised models (e.g., K-Nearest Neighbours (KNN) [2]) that can be applied not only to recommendation tasks, such as rating prediction or item ranking, but also other classes of ML problems. This thesis examines an important and popular area of the Recommendation Systems (RS) design space, focusing on algorithms that are both specifically designed for producing recommendations, as well as other types of algorithms that are also found in the wider ML field. However, the latter will be only showcased in RS-based use-cases to allow comparison with specific RS models.

Throughout the past years, there have been increased interest in RS from both academia and industry, which led to the development of numerous recommendation algorithms [3]. While there are different families of recommendation models (e.g., Matrix Factorisation (MF)-based, K-Nearest Neighbours (KNN)-based), they can be grouped in three classes as follows: Collaborative Filtering (CF), Content-based Filtering (CBF), and Hybrid Approaches (HA). This thesis investigates the most popular class of RS, namely Collaborative Filtering-based (CF) recommendation algorithms, which recommend items to a user based on similar users' preferences. One of the current challenges in building CF engines is the selection of the algorithms to be used for producing recommendations. 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. This challenge is exacerbated by the constant growth of the input data, which in turn impacts the efficiency of these models, as more computational resources are required to train the algorithms on large collections to attain a predefined/desired quality of recommendations. In CF, these challenges have also impacted the way providers deliver content to the users, as they need to strike a balance between revenue maximisation (i.e., how many resources are spent for training the CF models) and the users' satisfaction (i.e., produce relevant recommendations for the users). In addition, CF models need to be periodically retrained to capture the latest user preferences and interactions with the items, and hence, content providers have to decide whether and when to retrain their CF algorithms, such that the high training times and resource utilisation costs are kept within the operational and monetary budget. Therefore, the problem of estimating resource consumption for CF becomes of critical importance.

In this thesis, we address the pressing challenge of predicting the efficiency (i.e., computational resources spent during training) of traditional and neural CF for a number of popular representatives, including algorithms based on Matrix Factorisation (MF), K-Nearest Neighbours (KNN), Co-clustering, Slope One schemes, as well as well-known types of Deep Learning (DL) architectures, such as Variational Autoencoder (VAE), Multi-layer Perceptron (MLP), and Convolutional Neural Network (CNN). 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, memory overhead, and GPU utilisation of the CF's training phase. Our contributions further include an adaptive sampling strategy, to address the trade-off between the computational cost of sampling the dataset and training the CF models on the said samples and the accuracy of the estimated resource consumption of the CF trained on a full collection. Furthermore, we provide a framework which quantifies both the training efficiency (i.e., resource consumption) of CF, as well as the quality of the recommendations produced by the said CF once it has been trained. Finally, systematic experimental evaluations demonstrate that our methodology outperforms state-of-the-art regression schemes (i.e., BB/GBM) by a considerable margin (e.g., for predicting the processing time of CF, the accuracy of WB/LR is 160% higher than the one of BB/GBM), with an overhead that is a small fraction (e.g., 3-4 times smaller) of the overall requirements of CF training.

Acknowledgements

The completion of the doctoral studies would have been an impossible accomplishment without the support of some remarkable people, who not only helped me to materialise my research ideas into a tangible thesis, but also shaped me into a better researcher and person.

My first thank you goes to my supervisor, Dr Nikos Ntarmos, who granted me the opportunity to work with him, and who trusted me with the challenging yet highly rewarding task of completing a Ph.D. Nikos, thank you for your patience, guidance, and mentorship throughout the past four years. I would also like to thank Dr Yashar Moshfeghi for his feedback and support, as well as the interesting discussions that emerged during our research meetings. Yashar, thank you for making me see the bright side of things, even when hardships were encountered.

I would like to acknowledge my progress review committee, Dr Richard McCreadie and Dr Simon Rogers, for their constructive feedback, which helped me clarify my research questions and contributions. In addition, I would like to say thank you to Dr Christos Anagnostopoulos and Dr Craig Macdonald for providing me with their expertise when it was required.

Sincere thanks are dedicated to fellow Ph.D. colleagues, who over the past years also became my friends: Alex, Fotis, Gözel, Maha, Mihail, Paddy, thank you for your encouragement and for cheering me up when things did not go as planned. A special thank you is reserved for my office mate and best friend, Natascha, who always supported me, and who remains one of my main pillars of strength to this day.

Last but not least, I would like to thank my family, close and extended, and friends, for their kind and uplifting words of support, which encouraged me to complete my Ph.D. journey, as well as to look forward to the next chapter of my life.

Table of Contents

1 Introduction **1**

1.1 Context and Motivation 2

1.2 Research Questions and Contributions 5

1.3 Thesis Statement 8

1.4 List of Publications 8

1.5 Thesis Outline 10

2 Background **12**

2.1 Information Filtering Systems 13

2.1.1 Overview 13

2.1.2 Recommendation Systems 13

2.1.2.1 Types of Input and Recommendation Tasks 14

2.1.2.2 Collaborative Filtering 16

2.1.2.3 Content-based Filtering 16

2.1.2.4 Hybrid Approaches 17

2.1.2.5 Neural Architectures for Recommendation Systems . . . 18

2.1.3 Evaluation of Recommendation Systems 20

2.1.3.1 Evaluation Methodologies 21

2.1.3.2 Effectiveness-oriented Evaluation Metrics 22

2.1.3.3 Efficiency-oriented Evaluation Metrics 24

2.2 Formal Definition of Selected CF RS Algorithms 25

2.2.1 Traditional Models 26

2.2.1.1 Basic Algorithms 27

2.2.1.2 K-Nearest Neighbours-based Algorithms 27
2.2.1.3 Matrix Factorisation-based Algorithms 29
2.2.1.4 Slope-based Algorithms 29
2.2.1.5 Co-clustering-based Algorithms 30
2.2.2 Neural Models 30
2.2.2.1 Bilateral Variational Autoencoder for Collaborative Filtering (BiVAE) Algorithm 31
2.2.2.2 Variational Autoencoder for Collaborative Filtering (VAECF) Algorithm 32
2.2.2.3 Neural Network-based Collaborative Filtering (NCF) Algorithm 33
2.2.2.4 Convolutional Matrix Factorisation (ConvMF) Algorithm 34
2.3 Related Work 35
2.3.1 Efficiency-oriented Performance Studies for CF 36
2.3.2 Effectiveness Prediction in CF Models 37
2.3.3 Using Asymptotic Complexity in Efficiency Evaluation 38
2.3.4 Resource Consumption Estimation Problem in Traditional and Neural Models 39
2.3.5 Sampling Approaches for CF Evaluation 40
2.4 Chapter Summary 41

3 Methodology **42**
3.1 Overview of Cost Models for CF 42
3.2 Resource Consumption Prediction Models 44
3.2.1 White-Box Approach 45
3.2.2 Contenders and Black-Box Approach 49
3.3 Recommendations’ Quality Prediction Models 51
3.4 CF Frameworks 53
3.4.1 Surprise 53
3.4.2 Cornac 54
3.5 Experimental Apparatus 54

3.5.1 Experimental Environment 54

3.5.1.1 Traditional CF 54

3.5.1.2 Neural CF 55

3.5.2 Datasets 55

3.5.3 Implementation 55

3.6 Chapter Summary 57

4 Sampling Strategies and Workloads for Evaluating CF Performance 58

4.1 Problem Overview 58

4.2 Standard Practice Sampling Techniques 59

4.2.1 Effectiveness-oriented Sampling Techniques 59

4.2.2 Limitations of Standard Practice Sampling Approaches 60

4.3 Efficiency-oriented Sampling Approach 61

4.4 Adaptive Sampling Methodology 62

4.4.1 Extended Sampling Approach 62

4.4.2 Adaptive Sampling Algorithm 64

4.5 Performance Evaluation of Sampling Approaches 66

4.5.1 Evaluation Metrics and Baselines 67

4.5.2 Effectiveness-oriented Evaluation of Sampling Approaches 67

4.5.3 Efficiency-oriented Evaluation of Sampling Approaches 68

4.5.4 Sample Quality Analysis 70

4.5.5 Assessment of Sampling vs. Training Cost for CF 71

4.5.6 Evaluation of the Adaptive Sampling Methodology 74

4.6 Chapter Summary 75

5 Efficiency Cost Models for Traditional CF 76

5.1 Problem Overview 76

5.2 Complexity Analysis 77

5.2.1 Processing Time Complexity Equations 78

5.2.2 Space (Memory) Complexity Equations 80

5.3 Experimental Settings for Traditional CF 81

5.3.1 Evaluation Metrics and Baselines 82

5.3.2 Sampling Strategy 82

5.4 Experimental Evaluation 83

5.4.1 Processing Time Prediction Models for CF 83

5.4.2 Comparative Assessment of the Cost of Prediction Models 85

5.4.3 Memory Overhead Prediction Models for CF 88

5.4.4 Recommendations' Quality Prediction Models for CF 90

5.5 Chapter Summary 92

6 Modelling Resource Consumption in Deep Learning CF 93

6.1 Problem Overview 94

6.2 Complexity Analysis 94

6.2.1 Processing Time Complexity Equations 95

6.2.2 Space Complexity Equations 95

6.3 Experimental Settings for Neural CF 96

6.3.1 Evaluation Metrics and Baselines 96

6.3.2 Parameter Configuration 97

6.3.3 Sampling Strategy 98

6.4 Experimental Evaluation 98

6.4.1 Processing Time Cost Models 98

6.4.2 GPU Memory Cost Models 101

6.4.3 Effectiveness Cost Models 103

6.5 Chapter Summary 106

7 Conclusions 107

7.1 Thesis Summary 107

7.2 Research Questions 108

7.2.1 Sampling Strategies 108

7.2.2 Traditional CF 109

7.2.3 Neural CF 109

7.3 Contributions 110

7.4 Limitations and Future Research Directions 112
7.5 Conclusion 113

Bibliography **114**

List of Tables

4.1 Average CoV across 5 samples for various values of S over the MovieLens 100K dataset. 71

4.2 Average CoV across 5 samples for various values of S over the MovieLens 1M dataset. 71

4.3 Average NRMSE of the predicted processing time for various values of S over MovieLens 1M for a CoV of 0.1 and a time budget T of 50 seconds. . 74

4.4 Average NRMSE of the predicted processing time for various values of CoV over MovieLens 1M for an upper sample size S of 40% and a time budget T of 50 seconds. 74

4.5 Average NRMSE of the predicted processing time for various values of T over MovieLens 1M for an upper sample size S of 40% and a CoV of 0.1. . 74

5.1 Time complexity equations for traditional CF derived from literature and implementation. 80

5.2 Space complexity equations for traditional CF derived from literature and implementation. 81

5.3 Mean and standard deviation of processing time (i.e., training and prediction time) for the prediction models across the three datasets (MovieLens 100K, MovieLens 1M, and GoodBooks 10K). 86

5.4 Estimated training cost vs. prediction accuracy w.r.t. different sample sizes on GoodBooks 10K. 87

6.1 Time complexity equations for neural CF. 95

6.2 Space complexity equations for neural CF. 96

List of Figures

1.1 Overview of the process of choosing a recommendation algorithm given a recommendation task and a set of computational resources. 4

2.1 An example of an User-item Rating Matrix (URM). The red cross denotes the user-item interaction or relationship. 14

2.2 An illustration of a Convolutional Neural Network (CNN) architecture adopted from [4]. 18

2.3 An example of a Multi-layer Perceptron (MLP) architecture adopted from [5]. 19

2.4 An illustration of a Variational Autoencoder (VAE) architecture adopted from [6]. 20

2.5 An illustration of the Bilateral Variational Autoencoder for Collaborative Filtering (BiVAE) model adopted from [7]. 31

2.6 An illustration of the Variational Autoencoder for Collaborative Filtering (VAECF) model (left) compared to BiVAE (right) adopted from [7, 8]. . . 32

2.7 User and item representation in the Variational Autoencoder for Collaborative Filtering (VAECF) model adopted from [7]. 33

2.8 The architecture of the Neural Network-based Collaborative Filtering (NCF) model adopted from [9]. 34

2.9 The architecture of the Convolutional Matrix Factorisation (ConvMF) algorithm adopted from [10]. 35

3.1 Overview of the proposed pipeline for sampling the input and training the CF models, while gathering performance metrics. 43

3.2 Overview of the Black-Box methodology for predicting the effectiveness (i.e., recommendations' quality) of CF. 51

4.1 Actual CF effectiveness (i.e., quality of the produced recommendations) for the full MovieLens 100K dataset using the standard practice (SP) strategy (red curve) compared to our proposed sampling approach described in Section 4.3 (blue curve). 67

4.2 Actual CF effectiveness (i.e., quality of the produced recommendations) for the full MovieLens 1M dataset using the standard practice (SP) strategy (red curve) compared to our proposed sampling approach described in Section 4.3 (blue curve). 68

4.3 Predicted processing times for the full MovieLens 100K collection using models trained on samples drawn with the standard practice sampling approach (non-dashed curves) and the proposed sampling strategy (dashed curves). 69

4.4 Predicted processing times for the full MovieLens 1M collection using models trained on samples drawn with the standard practice sampling approach (non-dashed curves) and the proposed sampling strategy (dashed curves) 69

4.5 Sampling (blue bars) and training (red bars) time cost for MovieLens 100K w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line). 72

4.6 Sampling (blue bars) and training (red bars) time cost for MovieLens 1M w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line). 73

4.7 GoodBooks 10K 73

4.8 Sampling (blue bars) and training (red bars) time cost for GoodBooks 10K w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line). 73

5.1 Predicted processing times for MovieLens 100K for various values of S using our approaches (WB/LR and WB/Bayes) and the two baselines (WB/Lit/LR, and state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. . . 84

5.2 Predicted processing times for MovieLens 1M for various values of S using our approaches (WB/LR and WB/Bayes) and the two baselines (WB/Lit/LR, and state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. 84

5.3 Predicted processing times for GoodBooks 10K for various values of S using our approaches (WB/LR and WB/Bayes) and the two baselines (WB/Lit/LR, and state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. 85

5.4 Predicted memory overhead for MovieLens 100K for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead for the entire dataset. 89

5.5 Predicted memory overhead for MovieLens 1M for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead for the entire dataset. 89

5.6 Predicted memory overhead for GoodBooks 10K for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead for the entire dataset. 90

5.7 Predicted effectiveness (i.e., recommendation quality) for MovieLens 100K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection. 91

5.8 Predicted effectiveness (i.e., recommendation quality) for MovieLens 1M for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection. 91

5.9 Predicted effectiveness (i.e., recommendation quality) for GoodBooks 10K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection. 92

6.1 Predicted processing time for MovieLens 100K using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. 99

6.2 Predicted processing time for MovieLens 1M using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. 99

6.3 Predicted processing time for MovieLens 100K using 2 GPUs for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. 100

6.4 Predicted processing time for MovieLens 1M using 2 GPUs for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset. 100

6.5 Predicted GPU memory for MovieLens 100K using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset. 101

6.6 Predicted GPU memory for MovieLens 1M using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset. 102

6.7 Predicted GPU memory for MovieLens 100K using 2 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset. 102

6.8 Predicted GPU memory for MovieLens 1M using 2 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset. 102

6.9 Actual RMSE of the recommendations for MovieLens 100K for various values of the upper sample size S. 103

6.10 Actual RMSE of the recommendations for MovieLens 1M for various values of the upper sample size S. 103

6.11 Actual NDCG of the recommendations for MovieLens 100K for various values of the upper sample size S. 104

6.12 Actual NDCG of the recommendations for MovieLens 1M for various values of the upper sample size S. 104

6.13 Predicted RMSE of the recommendations for MovieLens 100K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection. 104

6.14 Predicted RMSE of the recommendations for MovieLens 1M for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection. 105

6.15 Predicted NDCG of the recommendations for MovieLens 100K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (NDCG) over the entire collection. 105

6.16 Predicted NDCG of the recommendations for MovieLens 1M for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (NDCG) over the entire collection. 105

List of Algorithms

1 White-Box Resource Consumption Estimation Algorithm 47
2 Black-Box Resource Consumption Estimation Algorithm 50
3 Black-Box Recommendation Quality Estimation Algorithm 52

4 Proposed Sampling Algorithm 63
5 Adaptive Sampling Algorithm 65

Abbreviations

A

ABR: Ada Boost Regressor
ALS: Alternating Least Squares
AWS: Amazon Web Services

B

BiVAE: Bilateral Variational Autoencoder for Collaborative Filtering

C

CBF: Content-based Filtering
CF: Collaborative Filtering
CNN: Convolutional Neural Network
ConvMF: Convolutional Matrix Factorisation
CoV: Coefficient of Variation
CPU: Central Processing Unit

D

DCG: Discounted Cumulative Gain
DL: Deep Learning

G

GBM: Gradient Boosting Machine
GPU: Graphics Processing Unit

H

HA: Hybrid Approaches

I

IDCG: Ideal Discounted Cumulative Gain

IFS: Information Filtering Systems
IR: Information Retrieval

K

KNN: K-Nearest Neighbours

M

MAE: Mean Absolute Error
MF: Matrix Factorisation
ML: Machine Learning
MLE: Maximum Likelihood Estimation
MLP: Multi-layer Perceptron

N

NCF: Neural Network-based Collaborative Filtering
NDCG: Normalised Discounted Cumulative Gain
NMF: Non-negative Matrix Factorisation
NRMSE: Normalised Root Mean Squared Error
NVML: NVIDIA Management Library

P

P: Precision
PMF: Probabilistic Matrix Factorisation

R

R: Recall
RAM: Random Access Memory
RMSE: Root Mean Squared Error
RQ: Research Question
RS: Recommendation Systems

S

SGD: Stochastic Gradient Descent
SVD: Singular Value Decomposition
SVR: Support Vector Regressor

U

URM: User-item Rating Matrix

V

VAE: Variational Autoencoder
VAECF: Variational Autoencoder for Collaborative Filtering

W

WB: White-Box

Glossary

A

accuracy: a measure to quantify how close is a result to the correct/true value;
adaptive sampling: a type of sampling, where the selection criteria of the observations/values are updated throughout the sampling process based on preliminary results;
asymptotic complexity: the amount of resources (e.g., time, memory) required by an algorithm/model/program to train/run on a given set of inputs;

C

complexity equation: a mathematical equation which accounts the size of the input and describes the total time and/or space required by an algorithm to produce an output and solve a problem;
computational resources: the hardware resources used by a model/algorithm to solve a problem;
cost model: a method of determining the total number of resources and/or effort required by an algorithm;

E

effectiveness: the degree to which an algorithm produces the desired/expected result; in CF, effectiveness quantifies the degree to which a model recommends relevant content for a user;
efficiency: amount of computational resources used by an algorithm/model to solve a task and produce an output;

M

memory usage: the amount of RAM utilised by an algorithm;

O

overhead: additional resources (e.g., time, memory) required by an algorithm to complete a specific task;

P

performance: the amount of useful work accomplished by a system to solve a problem;
processing time: the time taken by an algorithm to train on a given input;

Q

quality: the degree to which information reflects an event or object; in CF, quality of recommendations refers to how well do the recommended items reflect the users' preferences;

S

sampling: a process through which a predefined number of values/observations are selected from a larger population;

U

user satisfaction: the perception of a user on a system that they use; in CF, user satisfaction is represented by the user's opinion on the recommended items;
user-item rating matrix: a rectangular (2D) array which contains users as columns and items as rows; the values represent the ratings given by a user on an item;

Chapter 1

Introduction

Over the years, Machine Learning algorithms, including Recommendation Systems (RS) models, have played a key role in the development and success of large scale analytics frameworks. However, the fast growing datasets often require huge computational power, which usually translates into a high operational budget [11]. This challenge is also reflected in the efficiency of the models, as more resources are required to train and retrain the algorithms such that they attain good accuracy [12]. While these challenges are present in many fields (e.g., estimating query processing times, selecting optimal planning algorithms, etc.), RS have been identified as an area of particular interest, for both academia and industry, where models consume large amounts of computational resources during training [13, 12]. Therefore, a vital challenge faced by recommendation providers is to decide how and when the RS algorithms need to be trained, such that they reflect the rapidly changing user preferences, and in turn, they produce relevant recommendations [13].

Recommendation Systems (RS) have been developed to address the information overload problem, helping users to select appropriate items by narrowing down the number of choices that are presented. This process is achieved through three approaches, namely Collaborative Filtering (CF), Content-based Filtering (CBF), and hybrid methods. While in all approaches, the goal is to produce tailored recommendations, which are in line with the users' tastes and requirements [14], CF-based models are the most widely utilised RS [3]. The motivation behind the popularity of CF models comes from the fact that these algorithms produce recommendations based on users with similar preferences. This approach (i.e., CF) outputs recommendations of a higher quality than those obtained from CBF or hybrid methods [14, 3], as the users' tastes are taken into consideration. Furthermore, presenting the right products to the users is dependent on (a) the content available from the providers, and (b) the recommendation algorithm used. While the former is out of the scope of this thesis, we concentrate our attention and efforts on the latter, with a particular focus on CF, as one of the current challenges in deploying recommendation engines is which model should be chosen.

This decision is based on the model's effectiveness (i.e., the quality of recommendations), as well as its operational constraints (e.g., processing time, memory overhead, etc.). To select an appropriate recommendation algorithm, a set of evaluation criteria is required.

The evaluation of recommendation systems spans across multiple dimensions, such as accuracy, relevance, novelty, diversity, as well as user and content providers' satisfaction [14]. Over the years, the literature has established how one can evaluate the recommendations along these dimensions and what tools/metrics can be used [15, 16, 17, 18]. However, the efficiency and resource consumption of recommendation models should not be overlooked, especially for the algorithms with highly expensive training requirements. Therefore, focusing only on the nominal accuracy of a recommendation model is a fallacy, as there is an inherent trade-off between its efficiency (e.g., computational resources) and the quality of the recommendations [19, 20]. Moreover, given the large design space with numerous recommendation algorithms and implementations, it is often difficult to decide which one should be selected given a recommendation task and/or a set of operational requirements (e.g., how many CPUs/GPUs are available for training). Consequently, this leads to the problem formulation and motivation of the doctoral work, as presented in the next section.

1.1 Context and 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 [13]. 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 [12, 13]).

Furthermore, high training times/resource utilisation also translate to expensive energy and monetary costs for the content providers [12], and even raise concerns for damage to the environment as a result of increased amounts of CO_2 emissions [21]. These problems are further exacerbated by the ongoing data generation growth, identified by a recent study released by Amazon [12] as one of the key factors which impact the scalability of CF algorithms. Also, as described in [22, 23], the problem of limited computational resources is worsening for Deep Learning (DL) CF algorithms, as most neural jobs fail because they run out of GPU memory or the processing time is far too high. In addition, the implementation of the CF models plays a critical role, as the same model can be implemented more or less efficiently. For example, in [24] the runtimes of the two approaches presented vary by a factor of 600, making the more expensive implementation to become unsuitable for many scenarios and recommendation tasks.

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 [25, 26, 27]. Throughout the thesis, we will refer to this approach as the *black-box* performance evaluation.

It is often the case that the more advanced a model, the higher its training time and the higher the 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 [25, 26, 27] (see also Chapter 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 (Chapter 5 and Chapter 6) 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 (WB)*[1] approach, as opposed to the black-box technique described above. Furthermore, as part of this thesis, 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 (Chapter 4).

Given the context above, let us examine Figure 1.1, which illustrates how a CF model is selected. In step 1, users interact with various items (e.g., books, films, songs) generating input data for the recommendation models, captured as URMs. Then, in step 2 of Figure 1.1, several algorithms are trained and tested against the inputs, and during training, their resource consumption (e.g., processing time, RAM, GPU), as well as the quality of the recommendations (e.g., RMSE, NDCG) are examined. Furthermore, if one would like to know which algorithm to use for producing recommendations, they would need to investigate (a) the effectiveness (e.g., RMSE, NDCG) of the CF on the given user-item rating matrix, and (b) the cost w.r.t. resource consumption (e.g., processing time, RAM, GPU) of the CF model during training. While the former has been investigated in the RS literature [26, 25], the latter (i.e., estimating resource consumption of CF) has received little to no attention.

[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.

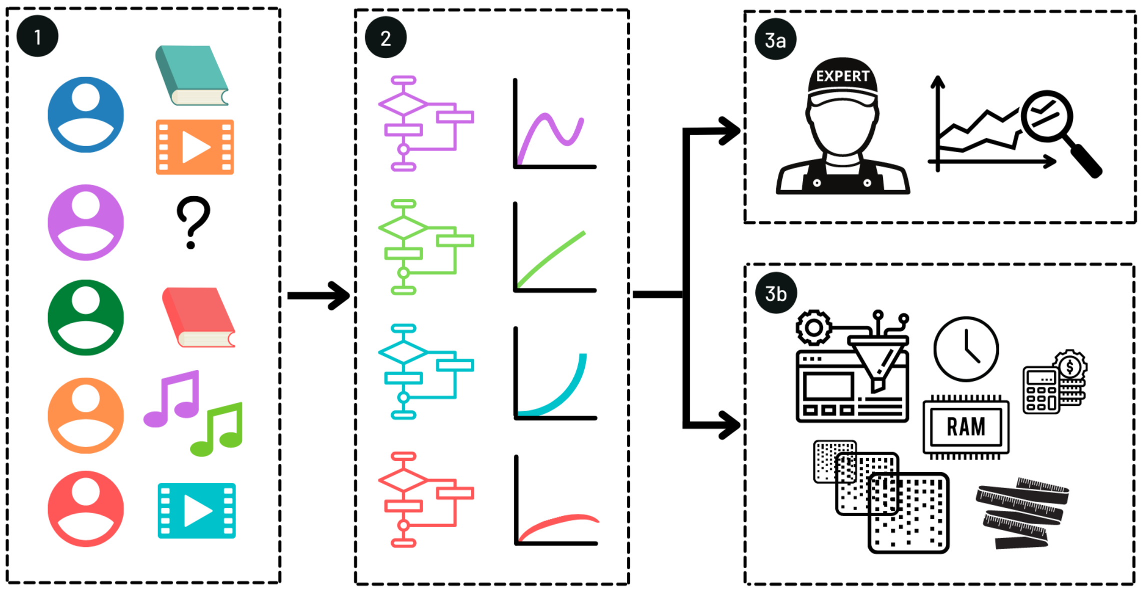

Figure 1.1: Overview of the process of choosing a recommendation algorithm given a recommendation task and a set of computational resources.

Consequently, there are two approaches for choosing the appropriate recommendation model and predicting its resource consumption. The first one, depicted in step 3a of Figure 1.1, involves having an expert in the field who knows how to tweak the knobs of the algorithms or which one would fit within the operational constraints based on previous experience. However, this approach might not be feasible for all recommendation providers, as smaller establishments might not be able to employ and benefit from the knowledge of such experts. To this end, we present the second approach, illustrated in step 3b of Figure 1.1, which does not require specialist knowledge and involves a framework that predicts the efficiency (i.e., processing time, RAM, GPU) and effectiveness (i.e., quality of recommendations) of a CF model without training it on the whole collection. For the scope of this thesis, we focus on the latter (i.e., step 3b), contributing the white-box and black-box methodologies, as well as an adaptive sampling component, which can be accurately applied, achieving a low error rate (NRMSE), for estimating the computational cost of training both traditional and neural CF models, with minimal effort and resource usage.

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.) [28]. Over the years, different research communities have been working towards building and optimising more efficient models to save energy [29, 30], and minimise resource consumption [31, 32]. 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 cloud-based platforms (e.g., Google Collab [33], Amazon Web Services (AWS) [34], etc.). This, in turn, could help cloud-based providers 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 [35] and evaluation studies [36, 27][2].

1.2 Research Questions and Contributions

Following the problem definition and research context above, we formulate the central research questions (**RQx**) investigated in this thesis. For each RQ, we also provide the contributions (**Cx.y**) that emerged from the work conducted. The research questions and contributions are structured around each experimental chapter of this thesis.

Chapter 4 provides an overview of the current standard practice sampling techniques [25, 26] for sampling a dataset, represented as an URM, with the purpose of estimating the *effectiveness* of a CF model on the full dataset. This chapter also investigates whether the current standard practice sampling approaches can be used for drawing samples from an URM to predict the *efficiency* of a CF algorithm on the complete dataset. Based on these challenges, we formulate the following research questions and contributions:

RQ1: How should we sample the base data (i.e., URM), such that the quality of the predicted efficiency/effectiveness of a CF model is not harmed?

- **C1.1:** An investigation into how current standard practice sampling strategies used for effectiveness prediction perform in the task of estimating resource consumption for CF models.
- **C1.2:** An adaptive sampling strategy that dynamically draws samples from the URM to jointly satisfy a user-defined Coefficient of Variation (CoV) and/or a predefined resource constraint (e.g., maximum time quota) that can be used for both efficiency and effectiveness prediction of CF algorithms.
- **C1.3:** An experimental evaluation which compares the proposed adaptive sampling strategy with the current standard practice approaches.

[2]These challenges were also discussed with industry professionals (e.g., Amazon, Netflix), as part of the Doctoral Symposium in ACM RecSys 2020 [19], who highlighted the importance of model selection, training, and deployment based on the available resources and the targeted effectiveness/accuracy outcome.

RQ2: Given an upper sample size S% from a dataset, how do we determine when to stop sampling based on the number of samples for which we obtain consistent resource usage measurements (e.g., the processing times are in a tight interval measured using the variance of the values)?

C2.1: A sampling algorithm that assesses the quality of the samples based on a given confidence interval and a mean, and then decides if to further sample the URM based on the operational time and memory constraints.

C2.2: An empirical study which analyses the variance and quality of the samples produced with the proposed algorithm and adaptive sampling strategy.

Chapter 5 covers the *efficiency* cost models for predicting the resource consumption in traditional CF algorithms. This chapter also illustrates and compares the two main approaches for estimating efficiency, namely, the *White-Box* and *Black-Box* methodologies. To this end, we present the following research questions and contributions:

RQ3: 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?

C3.1: An initial analysis into how the processing time and memory overhead scale with respect to samples, drawn from URM, of different sizes.

C3.2: We conceptualised the baseline approach for estimating efficiency of CF models using solely characteristics of the input and state-of-the-art regressors (i.e., Black-Box approach).

C3.3: We proposed a novel methodology for predicting the processing time and memory overhead using complexity analysis and curve fitting primitives (i.e., White-Box approach).

RQ4: Can the efficiency and effectiveness 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?

C4.1: An empirical study which investigates the best state-of-the-art regressors for estimating the efficiency of CF algorithms.

C4.2: An experimental evaluation which assesses the accuracy of the Black-Box approach for predicting resource consumption for CF models.

C4.3: A comparative assessment across Black-Box based regressors for predicting the effectiveness (i.e., quality of recommendations) for traditional CF algorithms.

RQ5: How can we use complexity analysis to estimate the resource consumption of a CF algorithm?

- **C5.1:** A theoretical analysis of the time and space complexity for traditional CF models using their implementations showcased in the popular Surprise framework [37].
- **C5.2:** An efficiency cost model based on time and space complexity equation (i.e., White-Box approach) for estimating processing time and memory overhead for traditional CF algorithms.
- **C5.3:** An experimental study which compares the White-Box methodology against the Black-Box approach, and efficiency cost models based on complexity equations from the relevant literature.

Chapter 6 discusses the resource cost models for neural CF algorithms. As part of this chapter, we investigate how *resource consumption* (e.g., processing time, GPU memory) varies during the training phase of the DL CF models, as well as how it can be predicted using characteristics of the input and different methodologies. This work led to the following research questions and contributions:

RQ6: Given the processing times and GPU memory overhead of a neural CF algorithm on a smaller sample of the URM, how can we estimate the overall processing time and GPU memory of the trained algorithm on the complete dataset?

- **C6.1:** A theoretical analysis of the time and space complexity for DL CF models using their implementations showcased in the Cornac framework [38].
- **C6.2:** An empirical study which determines how the processing time and GPU memory vary with respect to the size of the input when one or multiple GPU cards are used for training the CF model.
- **C6.3:** We extended the proposed White-Box and Black-Box methodologies to also estimate training cost for DL-based recommendations.
- **C6.4:** A comparative assessment between the White-Box and Black-Box methodologies, which evaluates the accuracy of each method for predicting the resource consumption of neural CF algorithms.

RQ7: Given the effectiveness, measured as RMSE/NDCG, of a DL CF algorithm on a smaller sample of the input, how can we predict the overall quality of recommendations (e.g., RMSE, NDCG) of the model on the full collection (URM)?

- **C7.1:** An initial investigation into how the effectiveness of a neural CF model varies across inputs/samples of different sizes when the model is trained on one and multiple GPUs.

C7.2: An experimental evaluation of various off-the-shelf standard practice predictive models (i.e., Black-Box approach) used to estimate the quality of recommendations produced by a DL CF algorithm on a complete dataset given its effectiveness on a set of sub-samples.

1.3 Thesis Statement

This thesis argues that the resource consumption of the CF models during training phase can be estimated using characteristics of the input (i.e., URM), sampling-based probabilistic analysis, and efficiency cost curves. The scope of this work is threefold: firstly, we map the design space by examining state-of-the-art CF algorithms, including traditional and neural approaches, with respect to different characteristics, such as algorithmic complexity, internal representation of input data, sets of computations, etc.; secondly, we show how efficiency cost models can be used to predict the CF algorithms' performance, based on the white-box and black-box approaches; finally, we provide an evaluation methodology that assesses the CF models on accuracy (i.e., quality of the recommendations) versus training cost (e.g., processing time, memory overhead) curves, by exploiting the efficiency-effectiveness trade-offs and limitations present in the implementations, and experimenting with different sampling techniques.

1.4 List of Publications

The material presented in this thesis has been peer-reviewed in several venues (i.e., conference, workshop, journal). Some of the studies are already published and others are under review, as noted below. The work conducted in the Ph.D. spans across the following categories:

- conceptual and theoretical developments towards modelling resource consumption for CF algorithms during their training phase;
- efficiency cost models to predict computational requirements for traditional and neural CF models;
- several experimental evaluations.

Ideas developed in this thesis contributed to a paper (**P1**) published in the RecSys '20 Doctoral Symposium. In this work, we formalised the problem of efficiency estimation for CF algorithms, conceptualised the resource cost models, and gathered feedback from experts in the field from both academia and industry.

P1: 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

A central piece of the doctoral work has been dedicated towards building and evaluating the efficiency cost models for CF algorithms. To this end, we initially proposed and developed efficiency (i.e., processing time) cost models for traditional CF algorithms (see *Chapter 5*), using the White-Box and Black-Box approaches as presented in **P2**, and augmented to also estimate memory consumption in **P3**. Then, an adaptive sampling strategy (see *Chapter 4*) was conceptualised in **P2** and further described and evaluated in **P3**.

P2: 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.

P3: Iulia Paun, Yashar Moshfeghi, and Nikos Ntarmos. 2021. White-Box: On the Prediction of Collaborative Filtering Recommendation Systems’ Performance. ACM Transactions on Internet Technology (TOIT). *[Under Review]*

Following the investigation of modelling resource consumption in traditional CF models, we centred our attention towards estimating the efficiency (e.g., processing time, GPU memory) in neural CF algorithms (see *Chapter 6*). This study is in preparation to be submitted towards peer-review and publication as part of **P4**.

P4: Iulia Paun, Yashar Moshfeghi, and Nikos Ntarmos. 2021/2022. Modelling Resource Consumption in Neural CF Models.

During the doctoral studies, apart from modelling efficiency in CF models, other research directions, such as graph clustering for graph-based recommendations, and context-aware recommendations, were explored and led to the publications listed below.

P5: Ivan Kyosev, Iulia Paun, Yashar Moshfeghi, and Nikos Ntarmos. 2020. Measuring Distances Among Graphs En Route To Graph Clustering. In Proceedings of the 5th IEEE Workshop on Advances in High-Dimensional Big Data (AdHD). IEEE, Atlanta, GA, USA, 3632-3641.

P6: Conor Morgan, Iulia Paun, and Nikos Ntarmos. 2020. Exploring Contextual Paradigms in Context-Aware Recommendations. In Proceedings of the 4th IEEE Workshop on Human-in-the-Loop Methods and Future of Work in Big Data (HMData). IEEE, Atlanta, GA, USA, 3079-3084.

1.5 Thesis Outline

This section presents the structure of the remaining chapters of the thesis and summarises the content illustrated in each chapter.

Chapter 2 provides a brief overview of the Information Filtering Systems (IFS) field, with a particular focus on Recommendation Systems (RS). We discuss the main classes of recommendation models and present the standard practices of evaluation for RS. Then, we detail the most popular and recent CF algorithms, including both traditional and neural approaches. This chapter also highlights the related work for performance modelling in CF algorithms, previously proposed in the literature, addressing topics of interest, such as sampling-based performance estimation, efficiency/effectiveness cost models, approximating probabilistic analysis, etc. Finally, the last section of this chapter presents adjacent previous studies, conducted in the doctoral work, which led to the problem formulation of efficiency prediction for CF models, and the core research presented in the following chapters.

Chapter 3 provides an overview of the theoretical methods for estimating the CFs' performance (i.e., resource consumption and recommendations' quality). Initially, we present the proposed framework for sampling the input, training the CF on the samples, and predicting the overall efficiency and effectiveness of the CF models. Then, each component of the framework (i.e., sampling strategy, White- and Black-Box models) is described in more depth. Additionally, we discuss the Surprise [37] and Cornac [38] frameworks utilised to investigate traditional and neural CF, respectively. Lastly, Section 3.5 illustrates the hardware and environment used for the experiments, the datasets on which the CF algorithms have been trained, as well as the implementation details for the proposed methodologies.

Chapter 4 presents our proposed adaptive sampling approach, which given an URM, draws samples that can be used for both efficiency and effectiveness prediction. Then, we compare our sampling algorithm with the standard practice sampling baseline, focusing on the accuracy of the predicted processing times and the effectiveness (i.e., quality of recommendations) of CF models. The empirical study demonstrates that standard practice sampling techniques cannot capture the complexity characteristics of the base data, and hence, a different approach is needed for efficiency prediction. This chapter also covers **RQ1** and **RQ2** showcasing how the quality of the samples can be gauged, and how to stop sampling the dataset to jointly satisfy a user-defined Coefficient of Variation (CoV) and/or a predefined operational constraint (e.g., maximum sample size, maximum time budget).

Chapter 5 implements the efficiency cost models for traditional CF algorithms. This work firstly addresses **RQ3** with an initial investigation into how efficiency can be quantified in CF models, what are the current state-of-the-art methods for performance prediction, and how can we accurately predict processing time and memory overhead using complexity analysis. Then, we showcase the baseline solution for efficiency prediction using characteristics of the input and the best off-the-shelf regressors (i.e., Black-Box approach) covering **RQ4**. Furthermore, this chapter presents the proposed resource consumption cost models for CF algorithms using the White-Box methodology. As part of answering **RQ5**, we provide an implementation-based complexity analysis for traditional CF models, followed by our proposed efficiency cost model for estimating processing time and memory overhead during the training phase of CF models. Finally, we report a comparative assessment of the White-Box and Black-Box methodologies, highlighting the benefits and limitations of each approach.

Chapter 6 illustrates how efficiency can be modelled in neural CF algorithms. As part of this work, we investigate how processing time and GPU memory utilisation vary while training the neural CF models on inputs/samples of different sizes using one and multiple GPU cards. To this end, we analyse the training cost of the DL CF models, and answer **RQ6**. As part of this research question, we build a framework that measures and records the processing time and GPU utilisation of the various neural CF algorithms. Then, we use these measurements with the proposed White-Box and Black-Box approaches to compute the training cost of DL CF algorithms, and discuss the trade-offs of each method. In this chapter, we also extend the previously mentioned framework to also capture and log the effectiveness of the neural models, which will be further used in conjunction with the adaptive sampling strategy to predict the accuracy of the models on the complete datasets as part of **RQ7**.

Chapter 7 summarises the work that was undertaken in the Ph.D., and revisits the contributions and research outcomes. Then, we review potential directions and ideas for future work, and discuss any limitations that have not been outlined before.

Chapter 2

Background

This chapter provides an overview of Information Filtering Systems (IFS) with an emphasis on Recommendation Systems (RS). We explain the different types of inputs and recommendation tasks found in the literature, as well as the three main methodologies for producing recommendations: Collaborative Filtering (CF), Content-based Filtering (CBF), and Hybrid Approaches (HA). Then, we describe the main evaluation techniques for RS, noting the lack of efficiency-oriented approaches, and why this evaluation dimension should not be overlooked. Furthermore, we outline the common architectures of deep neural networks for RS. Section 2.2 presents the traditional and neural recommendation algorithms investigated in this thesis, covering an important area of the RS design space. In particular, we explored and chose representatives from each main family of recommendation models with the goal of predicting their efficiency as presented in Chapter 5 and Chapter 6. Finally, Section 2.3 discusses the related work for the performance estimation problem. To this end, we first review efficiency-oriented studies for CF; in particular, we highlight the scarcity of these works, as well as the need of developers to report on the complexity of their proposed solutions. Furthermore, we describe what are the common practice methods for predicting the accuracy/effectiveness of the CF models using characteristics of the URM. Then, we illustrate the use of asymptotic complexity (i.e., estimating the time or space required by a CF as function of the size of the input) in empirical studies, as well as how the resource consumption prediction problem is tackled by different research communities, emphasising its importance and the challenges associated with it. The last part of Section 2.3 presents the current approaches for evaluating the effectiveness of CF recommendation algorithms using standard practice sampling techniques. Finally, a summary of the chapter is outlined.

2.1 Information Filtering Systems

2.1.1 Overview

The recent growth in data generated by users' interactions with digital media has led to the need of summarising and filtering the information that is being collected and processed. The field of Information Filtering Systems (IFS) is focused on how large volumes of data are managed and presented to the users [39]. The scope of IFS is to select only the data that is relevant for the users and help them with the information overload problem. IFS have seen many developments in the past decades, such as filters for search results on the Web, email filters based on personal preferences and predefined rules, filters designed for children to give them access to only suitable pages and applications [39].

In this thesis, we explore the IFS that are related to and based on the IR systems. Some works [40, 41] present IFS as a sub-type of IR, with the common aim of filtering the information to aid users in their choices. However, there are some differences between IFS and IR outlined as follows. Firstly, in IR, the users' needs for information are represented as queries, while in IFS, these needs are captured in user profiles. Secondly, in IR users do not have to be registered on a platform to input a query (i.e., usually search engines do not require an account) [41]. In IFS platforms, users have accounts and/or profiles with particular attributes that represent their preferences. Finally, IR chooses items (e.g., pages, documents) that match a given query, while in IFS, the information that comes from various sources is processed and only items from certain categories are presented to the users based on their tastes and previous interactions [41]. A special case of IFS are RS, which focus on modelling and predicting users' preferences with respect to a set of items [42], based on various attributes and methodologies as described in the next sections.

2.1.2 Recommendation Systems

Recommendation Systems (RS) are a collection of tools and techniques that have been designed to alleviate the information overload problem [14]. This problem is often encountered when there is an overwhelming amount of choices (i.e., referred to as items in RS literature) and a person (i.e., user) does not know which one to choose, explore or interact with [3]. To facilitate the selection process, RS produce personalised suggestions based on users' tastes and preferences. This process is not only increasing user satisfaction [15], but also contributing to revenue maximisation across the content providers [13]. Recommendation engines are relying on the fundamental principle that relationships exist across user-item interactions [3]. For example, based on this principle, users who enjoy science fiction novels will more likely be interested in the same literary genre, rather than a different one (e.g., historical). The more a user interacts with a certain category of items, the easier it is to construct recommendation models [14]. These interactions are often captured and referred to as ratings. Thus, given a set of users, a set of items, and a collection of ratings, one can build an User-item Rating Matrix (URM) [3] as described in the next paragraph.

In RS, a URM provides a numerical representation of the user-item interactions. In practice, it is built using the users' IDs/names as the columns of the matrix, and the items' IDs/names as the rows of the matrix or vice-versa. For each user-item pair, if the user interacted with the item, a rating, which can be numerical (e.g., 1 – 5) or binary (e.g., like/dislike), is placed in the corresponding cell of the matrix as seen in Figure 2.1. However, if a user did not interact with an item, the cell will be left blank with no rating value.

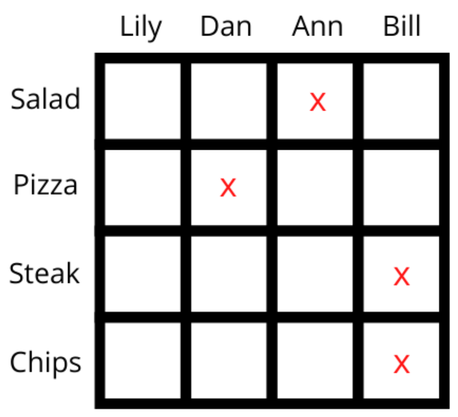

Figure 2.1: An example of an User-item Rating Matrix (URM). The red cross denotes the user-item interaction or relationship.

Once the URM has been built, recommendation models learn the relationships and dependencies across users and items and once they are trained, they can predict if a certain user will enjoy a new item or not. Over the past decades, RS emerged as a popular topic in both academia and industry, which led to the development of many recommendation algorithms [3]. While there are different families of recommendation models, three main classes of RS stand out: Collaborative Filtering (CF), Content-based Filtering (CBF), and Hybrid Approaches (HA). These will be presented in further detail in the next sections.

2.1.2.1 Types of Input and Recommendation Tasks

Before exploring the different classes of RS, it is important to note the types of input used by recommendation models (i.e., explicit and implicit), as well as the two main recommendation tasks (i.e., rating estimation and item ranking).

One of the key characteristics that needs to be taken into account when developing RS is the type of input that can be processed by the recommendation models. The most common input for a recommendation algorithm is in the form of ratings. These ratings are very useful in the recommendation process as they indicate the users' tastes and preferences with respect to different items [14]. In literature, we can find two main rating categories: (a) interval-based (e.g., -2, -1, 0, 1, 2 or 1 –5) and (b) binary ratings (e.g., 0/1, like/dislike) [3]. While both categories can be used to show how much a user enjoyed a certain item, in interval-based ratings, multiple values can indicate a positive interaction. For example, in Netflix's rating system [43], "3" is mapped to "liked it", "4" means "really liked it", and "5" denotes that it was one of the best items. There is also another subclass of ratings, named unary ratings [3], where a user can show their satisfaction using a liking function/button. However, in unary ratings, there is no mechanism to indicate that a user did not enjoy an item.

The two main types of ratings set the direction of the collections used in training and evaluating recommendation models. To this end, we present (a) explicit feedback and (b) implicit feedback datasets. In explicit feedback collections [44], users state their preferences directly or explicitly by allocating a rating value to an item, which is then captured in the URM. In these settings, a higher rating value indicates that the user was more satisfied with the product/item. In explicit feedback datasets, the empty entries in the URM denote that either the users did not specify their tastes or they have not interacted with those items. On the other hand, in implicit feedback datasets, the users' tastes are inferred from their interaction with the items. For example, if a user listens to a song multiple times or recurrently buys a certain product will be translated into a "like" under the implicit feedback assumption. However, one of the challenges that is often encountered is modelling the users' "dislike" towards items [45]. This emerges from the fact that if a user did not interact with an item does not necessarily mean that they would not enjoy that product. Furthermore, while it has been argued that recommendation models should be less intrusive, and hence, more focused on utilising implicit feedback [46], explicit feedback is preferred as it is deemed more reliable and accurate [47]. All in all, the type of input and feedback available in the collection is one of the salient factors which determines what recommendation models will be selected and deployed.

Besides explicit and implicit feedback, another characteristic that is taken into account when designing, building, and evaluating RS is the recommendation task or recommendation problem. To this end, there are two main approaches, (a) rating estimation and (b) item ranking, described below.

The rating estimation approach assumes that the URM is incomplete and we need to predict the rating value for a user-item pair based on the other ratings that are available in the matrix. In these settings, a recommendation algorithm is selected and trained on the "actual/observed" ratings to predict the missing ones. This approach is also known as the "matrix completion problem" [3]. The other main approach to produce recommendations is item ranking. In some instances, we do not need to know the exact rating a user would assign to an item. Instead, the goal is to recommend the top-K items, where K represents the total number of suggestions presented to the user. Choosing K depends on different factors and constraints (e.g., the user interface which provides the top-K list of suggestions) [48, 14]; however, common values of K are 5, 10, 20, etc. [45]. The item ranking approach is often referred to as the "top-K recommendation problem" [3] and aims to provide the users with a list of sorted items where the most relevant item is at the top and the least relevant item is at the bottom [14]. The items are sorted with respect to a predefined criterion (e.g., similarity to another item that was consumed by the user), which depends on the RS that is being used.

2.1.2.2 Collaborative Filtering

There are three techniques used to produce recommendations, and each of these techniques will be presented in the next sections. The first technique, namely Collaborative Filtering (CF), utilises models that are trained based on the user-item interactions, often captured as ratings in the URM [14]. The main principle behind CF algorithms is that similar users will have similar tastes [3]. To this end, if we were to solve a rating estimation problem, the missing ratings would be predicted using CF models, which would exploit correlations amongst similar items or similar users. The user/item similarity can be determined using the following approaches: (a) memory-based CF methods, and (b) model-based CF methods.

Memory-based CF methods were one of the earliest techniques to produce recommendations, as they are easy to implement and highly explainable [3]. They are centred around neighbourhood-based algorithms, which aim to cluster users/items based on similarity metrics [14]. One of the limitations of memory-based CF methods is their accuracy on sparse URMs [3], and also their expensive processing time [20] exemplified in Chapter 5.

On the other hand, model-based CF methods utilise machine learning techniques to infer the similarity across users/items. To this end, the models are parametrised and the parameters are learnt as an optimisation problem [3]. The most common representatives of model-based CF methods are latent factor approaches, such as Matrix Factorisation (MF) [14, 49]. These methods are often preferred over model-based CF techniques as they work well with sparse URMs [3], also shown in Chapter 5.

2.1.2.3 Content-based Filtering

Content-based Filtering (CBF) models are built using the descriptive attributes of items, rather than the actual ratings, to solve a classification or regression problem [3], which aims to find similar items. In CBF algorithms, items are marked as being alike if their descriptive attributes (e.g., genre) are similar and/or match [14]. A CBF approach is preferred over CF for recommending new items, which have not been rated yet. In this case, a CBF model will be used to find items that have been rated and present similar content features to the ones from the new items. Then, these new items will be shown to a user as recommendations [14].

However, CBF approaches are a less popular choice for building recommendation engines [50], due to the following limitations. Firstly, CBF models often suggest obvious items as these contain similar descriptive attributes [3]. Furthermore, the diversity of the recommendations is being impacted, as only certain categories of items with similar keywords will be presented to the user [14]. Secondly, CBF algorithms are not suitable for producing recommendations for new users [3]. This happens because new users have not interacted with enough items, and hence they have no or limited rating history [3]. Therefore, CBF methods are rarely used on their own. Instead, they are combined with CF methods or they are being incorporated in more specialised recommendation engines, such as knowledge-based systems [51, 3], where users can select which descriptive attributes are relevant for them.

2.1.2.4 Hybrid Approaches

As the RS methods have evolved, more methods were developed to support the interaction across users and items. This, in turn, generated more types of input data for recommendation models [14], which also require different recommendation techniques [3]. Consequently, various methodologies or parts of methodologies are combined into hybrid RS to be able to (a) process a wider variety of inputs, and (b) produce accurate recommendations, while leveraging the techniques from all relevant classes of RS [3].

Hybrid Approaches (HA) are often related to ensemble data analysis [52]. In this field, multiple machine learning techniques are combined to achieve a hybrid model with better accuracy than the individual ones [52]. Furthermore, in ensemble RS, multiple algorithms, often belonging to the same family, are combined to improve the accuracy of the recommendations given multiple sources of input [14, 3]. This technique was inspired from ensemble classifiers [52, 3]. While different recommendation techniques can be employed to increase the users' satisfaction [15, 16], or the recommendations' quality [17, 18], or the revenue of the content providers [13], all of them have to be evaluated with respect to their performance [14, 3], quantifying both effectiveness and efficiency, as described in the next section.

2.1.2.5 Neural Architectures for Recommendation Systems

Deep Learning (DL) is part of a wide family of ML models based on artificial neural networks. These networks aim to replicate the behaviour of the human brain aiming to learn representations of the data using high-level features from large inputs. DL models have also been successfully used in a broad number of cases and applications [53], including CF [14, 3]. In this section, we present the most common neural architectures that are found in Deep Learning (DL) models, such as Convolutional Neural Networks, Multi-Layer Perceptron, and Variational Autoencoders.

2.1.2.5.1 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a type of neural architecture that has been mostly used for processing images, as well as object detection and classification [54]. CNN is a regularised version of the Multi-layer Perceptron (MLP) architecture, and are also inspired by biological processes [55]. A CNN is comprised on an input layer, hidden layers that perform convolutions, and an output layer as shown in Figure 2.2.

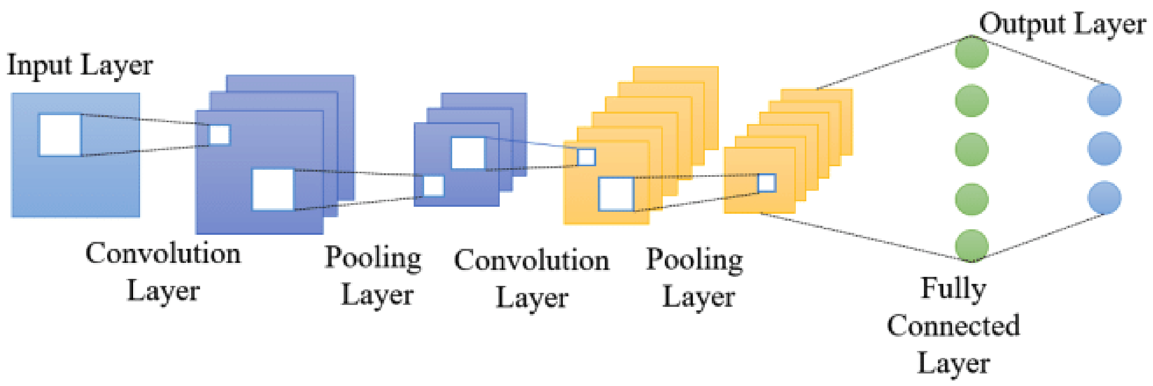

Figure 2.2: An illustration of a Convolutional Neural Network (CNN) architecture adopted from [4].

In the convolutional layers, the input is convolved and passed to the next layer for pooling/sub-sampling. The pooling layers are responsible to reduce the dimensions of the data by mapping the output of a cluster of neurons into a single neuron to be passed to the next layer. Pooling layers can work locally or globally depending on the size of the cluster that needs to be processed [4]. Lastly, the fully connected layers connect each neuron in one layer to all neurons in another layer, and then the output/prediction layer is mapped in an application-specific way [4], depending on the task that the model is trying to learn (e.g., image/text classification, rating prediction).

2.1.2.5.2 Multi-Layer Perceptron (MLP)

A very popular neural architecture is the Multi-layer Perceptron (MLP). This is inspired from the simple idea of a single-layer network, where the input is mapped to the output using a linear activation function ($a()$). This is often called a perceptron and is defined in Equation 2.1 [56].

$$\hat{y} = a(Wx + b) \tag{2.1}$$

In MLP the perceptrons are spread across different layers and they are connected to each other as exemplified in Figure 2.3. In these settings, the output of the perceptrons in the first (input) layer is mapped to the input of the perceptrons in the next layer, and so on. This is captured in Equation 2.2, where L represents the number of layers of the MLP network.

$$\hat{y} = a_L \left(W_L \left(\dots a_1 \left(W_1 x + b_1 \right) \right) + b_L \right) \tag{2.2}$$

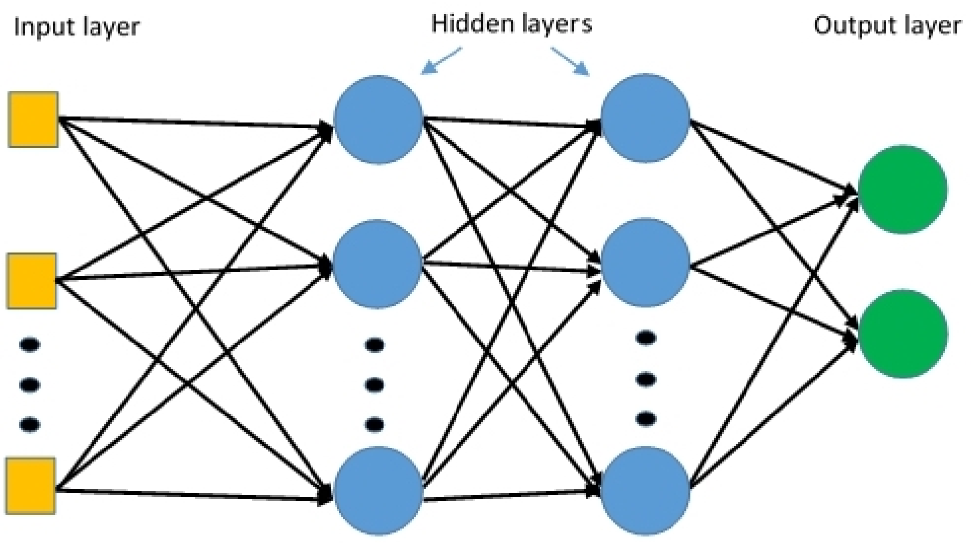

Figure 2.3: An example of a Multi-layer Perceptron (MLP) architecture adopted from [5].

It should be noted that different types of activation functions can be used in various layers and the choice of the activation function depends on the task that is being solved (e.g., regression, classification, computing probabilities) [56]. One of the strengths of the MLP architecture is that it can model and capture the dependencies across each dimension of the input vectors. In a recommendation task, this can be translated into learning the user-item interaction function using item and user embeddings [3].

2.1.2.5.3 Variational Autoencoder (VAE)

A Variational Autoencoder (VAE) architecture is similar to an autoencoder [57], where the input data is compressed into a multivariate latent distribution (i.e., encoding) and then it is reconstructed as effectively as possible (i.e., decoding).

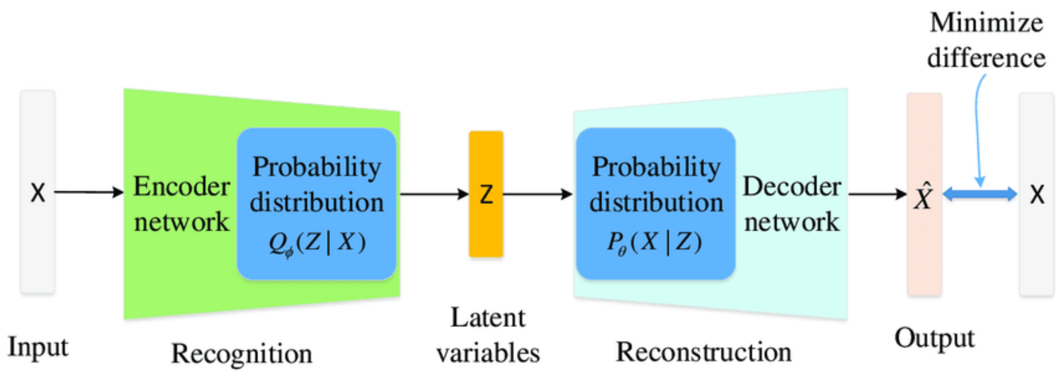

Figure 2.4: An illustration of a Variational Autoencoder (VAE) architecture adopted from [6].

In VAE, the input is sampled from a parametrised distribution, while the encoder and decoder are trained in tandem until the output reconstruction error is minimised [6], as shown in Figure 2.4. The VAE architecture captures an observation in the latent space in a probabilistic way. To this end, for describing a latent feature, instead of using an encoder that outputs a single value, a probability distribution is employed, which offers a smooth continuous representation of the latent space [6].

2.1.3 Evaluation of Recommendation Systems

As we have seen above, there are many ways in which we can solve a recommendation problem using different types of algorithms [14, 3]. However, it is often the case that with limited resources (e.g., time, computational power), we have to select the best or most appropriate RS model for our task [45]. This is often achieved by comparing the proposed algorithms using experimental evaluations. Also, when a new model is being developed, it has to be tested against state-of-the-art solutions and baselines [3]. The evaluation of RS algorithms has been a popular and active topic in the IR community with numerous studies addressing this theme [14, 58, 36]. In this section, we present the main evaluation methodologies found in the RS literature, as well as the most frequently used metrics. We also highlight the lack of efficiency-oriented metrics [19, 20], and we discuss which attributes could be used towards quantifying this aspect.

2.1.3.1 Evaluation Methodologies

RS can be evaluated using three methodologies: (a) online evaluations, (b) offline evaluations, and (c) user studies and expert interviews. Each methodology has its advantages and limitations as described below.

Online evaluations, as the name suggests, are conducted in real-time using different recommendation models and several users. Online evaluations are more frequently encountered in commercial recommendation engines [3], as the users can interact with the RS directly without interrupting their usual activities. In online evaluations, the user selection process plays a key role [59] to avoid introducing bias in the assessment. This can be alleviated using A/B testing [60] as follows. The user population is randomly split into two groups A and B. Then, group A users interact with a recommendation model M_1, while group B uses another model M_2, while keeping the other experimental conditions (e.g., number of recommendations, the user interface) as similar as possible. While the two user groups receive recommendations using M_1 and M_2, user engagement metrics (e.g., click-through rate) are collected [59]. Finally, the performance of M_1 and M_2 is gauged using the metrics and any additional user feedback [3]. Some of the limitations of online evaluations are that they are expensive to conduct (e.g., reward the users, spend time to recruit them) [3], and also in most cases we can only get a comparison across two or more models, rather than an absolute evaluation [60, 61].

In offline evaluations, previously collected data (e.g., ratings, clicks) is utilised to assess the performance of RS. In these settings, we can replicate the interactions between the recommendation engine and the users in a cheaper way than with online evaluations [3]. A key point in offline evaluations is the assumption that the behaviour and tastes of the users when the data was collected is similar to the ones the users would elicit when the RS would be deployed and used in real-time [14, 3]. Offline evaluation is the most used methodology in the RS literature due to its many advantages, such as explainable recommendations, low cost, robustness [3, 45, 62, 63]. Another benefit of conducting offline evaluations is that the data can be used as a benchmark towards assessing multiple models. and also, for reproducibility purposes [14]. Additionally, offline evaluation can be used to tune the hyper-parameters of RS models [3], which would be impractical in user studies or online evaluations. One of the drawbacks of offline evaluation is that the collected data is not necessarily periodically updated. Therefore, it might not reflect the users' tastes and needs in the future. This can be mitigated by frequently updating the offline evaluation datasets and collections to reflect the latest user interactions and preferences [14].

Lastly, user studies and expert interviews can be used as a methodology to assess the RS. In this case, users interact with the recommendation models while their actions are logged. Usually, user studies and expert interviews are conducted with a small number of participants who are selected based on their knowledge and expertise in the field [64]. Moreover, during this evaluation methodology users can be asked to fill in questionnaires or give feedback before, during, and after interacting with the recommendation engine. The advantage of this method consists in the richness of the information that is being captured under controlled conditions [3]. In user studies and expert interviews, the RS algorithms are not the only ones which are being assessed; additionally, the user interface where the recommendations are being presented is also evaluated [64]. Apart from the expensive cost of recruiting users, one of the main limitations of this evaluation technique is that the users become aware of their actions which could introduce bias in their decisions and skew the experimental procedure [64, 14]. Consequently, user studies and expert interviews are less frequently used to capture the RS performance [14].

One of the current limitations in RS evaluation is that the studies only report the quality of the recommendations through effectiveness metrics and methodologies [62, 63] with limited or no information about their efficiency [19, 20]. However, recent work [65] has started addressing this topic by presenting some insights into the observed efficiency (processing time) of the models. Thus, and in the context of environmental awareness [21], 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. In this thesis, we address this gap (i.e., lack of efficiency studies) by introducing and formalising efficiency-oriented evaluation methodologies and metrics as described in the next sections and chapters.

2.1.3.2 Effectiveness-oriented Evaluation Metrics

Given an evaluation methodology, RS are assessed based on the recommendation task/problem that is being solved. To this end, to measure the effectiveness of a RS, the literature [14, 3] provides two types of metrics: (a) rating prediction, and (b) ranking-based metrics. For the former, the most frequently used metrics are Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) [49, 66, 37]. For the latter, the metrics that are widely adopted are Precision (P), Recall (R), and Normalised Discounted Cumulative Gain (NDCG) [14, 3]. Each of these metrics will be further described below. We note that there are other effectiveness-oriented metrics, such as novelty, diversity, serendipity [67, 68], however these are not applicable to the models investigated in this thesis, as they require additional features (e.g., textual data, clicks, views) which are not present in the publicly available URMs/datasets utilised in the experimental evaluation.

Rating prediction metrics are utilised to determine the accuracy of the estimated values for the missing ratings in the URM [14]. Two well-known rating prediction metrics are MAE (Equation 2.3) and RMSE (Equation 2.4) defined below.

$$MAE = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{r}_{ui}| \tag{2.3}$$

$$RMSE = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2} \tag{2.4}$$

To compute these metrics one needs the actual rating r_{ui} provided by user u to item i, as well as the predicted rating $\hat{r}_{ui}$. The distinction between the two metrics is that MAE measures the average magnitude of the error in a prediction, while RMSE is the square root of the average of the squared difference between the predicted rating and the actual rating. Following its definition, RMSE applies a higher penalty for large errors, as the errors are squared before they are averaged [14]. In contrast, MAE does not assign a high weight to large errors. In both cases, a lower RMSE/MAE value means better accuracy for the recommendations. Given these differences between the two rating prediction metrics, RMSE is a more popular choice in the literature [14, 3].

The other type of metrics commonly used to assess RS are ranking-based metrics. These are used to measure the effectiveness of the ranking order of the top-K items [14]. The most common metrics for measuring the quality of the ranking are adopted from IR: namely, Precision (P), Recall (R), and Normalised Discounted Cumulative Gain (NDCG). Additionally, we note that there are other ranking-based metrics that are also used for RS evaluation, such as Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Hit Ratio (HR) [14]. However, these were out of the scope of the work conducted in this thesis.

Precision (Equation 2.5) is a ranking-based metric focused on determining the fraction of relevant items recommended out of all items recommended, while recall (Equation 2.6) measures the fraction of relevant items recommended out of all relevant items [3]. Precision and recall are binary metrics, requiring knowledge about whether an item is relevant or not. To this end, we define $rel_u(i)$ as the relevance function for user u and item i. This function outputs 1 if the item i is relevant to user u, and 0 otherwise. For both precision and recall, a higher value is translated to a more accurate model [14, 3]. Furthermore, to formally define precision and recall below, we use R_u as the set of the top-K recommendations for user u.

$$Precision@K = \frac{\sum_{i \in R_u} rel_u(i)}{|R_u|} \tag{2.5}$$

$$Recall@K = \frac{\sum_{i \in R_u} rel_u(i)}{\sum_{i \in I} rel_u(i)} \tag{2.6}$$

The last effectiveness-oriented metric that we discuss is the Normalised Discounted

Cumulative Gain (NDCG), as proposed in [69]. In RS, this metric has been successfully used to evaluate recommendations that yield multiple relevance grades within the ground-truth data [14]. Therefore, NDCG is more suitable for fine-grained URMs, which contain ratings on a scale (i.e., 1 to 5) and not binary ratings (i.e., like/dislike). This makes NDCG an appropriate metric to utilise in the experimental evaluation of this thesis, since our datasets contain fine-grained ratings. To this end, we firstly define the Discounted Cumulative Gain (DCG) metric (Equation 2.7).

$$DCG = \sum_{i=1}^{K} \frac{2^{rel_i}}{\log_2 i + 1} \tag{2.7}$$

In DCG, rel_i denotes the ground-truth relevance score of the item at position i, and rel_i is 1 if the item is relevant, and 0 otherwise. Then, we also define the Ideal Discounted Cumulative Gain (IDCG), which is a score that would have been assigned to a perfect ranking with respect to the ground-truth data. Finally, the NDCG (Equation 2.8) metric is computed below.

$$NDCG = \frac{DCG}{IDCG} \tag{2.8}$$

2.1.3.3 Efficiency-oriented Evaluation Metrics

As we discussed above, many studies focused on developing better methodologies and metrics for assessing the effectiveness of RS. However, the efficiency of the models, particularly during their highly expensive training phase, received little to no attention [19, 20]. We note that while the RS literature does not specifically provide efficiency-oriented metrics and methodologies, the performance of the algorithms can be quantified in terms of resource consumption [21]. To this end, we propose to use systemic metrics [70], such as processing time, memory overhead, etc., to reflect the efficiency of RS.

Processing time is used to quantify the amount of time that has elapsed since executing a process or a call [70]. In a system, the time can be classified in three ways: (a) real, (b) user, and (c) system time [70]. The former, real time, also equivalent to wall clock time, represents all elapsed time including time slices used by other processes, as well as the time spent by process while it is being blocked (e.g., waiting for I/O to complete). User time measures how much time is spent within a process in user-mode without taking into account the time spent when the process is blocked or by other processes. Finally, system time quantifies how much time is spent in the kernel within the process. Similarly to user time, this only relates to the time used by the actual process. In the experimental chapters of the thesis, we will refer to processing time as the overall time in both user and system modes. This allows us to measure the CPU (and GPU for DL models) time when the RS are trained. Another approach to quantify the amount of CPU needed to train the models, would be to measure the utilisation in % of the load. However, we believe that reporting the CPU time required by a CF is more intuitive for the users of our framework and methodology (e.g., KNN takes 3 hours to train vs. KNN utilises 70% of the CPU).

Memory overhead is another attribute that can be measured to determine the efficiency of a model [70]. In the experimental evaluations, we log the RAM utilisation and GPU memory while the CF algorithms are being trained. This allows us to measure the amount of data that is being stored and needed in CPU and/or GPU computations [70] before having a trained algorithm which can produce recommendations. In both cases, a lower value in processing time and memory overhead indicates a more efficient model [71, 20].

Recent studies have highlighted the issues that emerge from working with models with high resource usage [12, 13, 21]), such as expensive energy and monetary costs, and damage to the environment as a result of increased amounts of CO_2 emissions. We note that for RS, it is the training phase that has a high resource cost, often with several orders of magnitude higher than the time it takes to produce a recommendation given a trained algorithm [12]. Consequently, we can further quantify the efficiency of a model with respect to its sustainability/“greenness” (e.g., carbon footprint) [12] or its monetary cost (e.g., how expensive it is to train a model in a given currency) [21]. However, these “advanced” efficiency metrics cannot be computed without the overall resource/hardware utilisation. Therefore, in our framework and methodology, we assess the efficiency of RS with respect to processing time (for CPU and/or GPU) and memory overhead (i.e., RAM and GPU memory).

2.2 Formal Definition of Selected CF RS Algorithms

The previous section provided an overview of the Recommendation Systems (RS) detailing how various models work depending on which recommendation task is addressed, the type of available inputs, as well as which of the three techniques (i.e., CF, CBF, and HA) is leveraged to produce recommendations. In addition, we also described how RS can be evaluated, noting the standard practice methodologies and metrics. Since CF-based algorithms are the most frequently used type of RS to produce recommendations [3], we channel our attention and efforts on this important area of the design space. To this end, this section discusses the most notable CF algorithms, selected based on popularity and novelty ((i.e., latest algorithm that was proposed in the literature) criteria, covering both traditional and neural approaches.

2.2.1 Traditional Models

In this thesis, we refer to traditional CF as the models that are not based on Deep Learning (DL) approaches. In this category, we present the following families of algorithms that were investigated: (a) basic, (b) K-nearest neighbours based, (c) variants of matrix factorisation, (d) slope-based, and (e) co-clustering approaches. To simplify the description of each model, we will use the nomenclature outlined below throughout the remainder of this section.

- R: the set of all ratings.
- U: the set of all users; u and v refer to distinct users.
- I: the set of all items; i and j refer to distinct items.
- m: the cardinality of U.
- n: the cardinality of I.
- U_i: the set of all users who rated item i.
- U_{ij}: the set of all users who rated both items i and j.
- I_u: the set of all items rated by user u.
- I_{uv}: the set of all items rated by both users u and v.
- b_{ui}: the baseline rating of user u for item i.
- R_{train}, R_{test}, $\hat{R}$: the training set, the test set, and the set of predicted ratings.
- $R_i(u)$: the set of relevant items rated by user u that also have at least one common user with item i.
- b_u, b_i: the deviations of user u, item i respectively, from the overall mean rating.
- $N_i^k(u)$: the k nearest neighbors of user u that rated item i.
- r_{ui}: the actual rating given by user u to item i.
- $\hat{r}_{ui}$: the predicted rating given by user u to item i.
- p_u, q_i: feature vectors for users, and items, respectively.
- p_{uf}, q_{if}: factors of user u and item i.
- λ_u, λ_i: regularisaton parameters for user u and item i.

- μ: the mean of all ratings.
- μ_u: the mean of all ratings given by user u.
- μ_i: the mean of all ratings given to item i.
- δ: the density of a dataset, computed using the rating matrix.
- C_u, C_i, C_{ui}: clusters for u and i, respectively, as well as the co-cluster of ui.
- $\overline{C_u}$, $\overline{C_i}$, $\overline{C_{ui}}$: the average rating in cluster C_u, C_i, and co-cluster C_{ui}.

2.2.1.1 Basic Algorithms

As part of the basic approaches, we investigated the rating prediction problem using *baseline* estimates and a Maximum Likelihood Estimation (MLE) based *random* approach.

Baseline algorithm, derived from [72], is used to address the large user and item effects present in the URM, such as some users rating items higher or some items receiving better scores than others. To alleviate these issues, the baseline estimate (i.e., predicted rating) for a given user and item is computed as follows:

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i \tag{2.9}$$

where each baseline was, in turn, computed using the Alternating Least Squares (ALS) method [73].

Random algorithm was used to compute random ratings based on a normal distribution, $N(\hat{\mu}, \hat{\sigma}^2)$, of the training set, with $\hat{\mu}$ and $\hat{\sigma}$ estimated using MLE [74], as shown in Equations 2.10 and 2.11.

$$\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \in R_{train}} r_{ui} \tag{2.10}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}} \tag{2.11}$$

2.2.1.2 K-Nearest Neighbours-based Algorithms

The next category we investigated comprised algorithms derived from KNN approaches. More specifically, we used *KNN*, *Centred KNN*, and *KNN Baseline*. Since the scope of this thesis includes CF models, we determined the nearest neighbours based on the similarity across users. In these settings, user-user methods [75] predict the values of missing ratings based on the existing ratings from users with similar tastes. Neighbourhood-based algorithms are also recognised in the literature for their high explainability, and for being easy to implement [72].

In the following paragraphs, we will explain the different similarity metrics that can be employed to determine alike users/items. For KNN and Centred KNN similarity was computed using Mean Squared Difference (MSD) [76], as described in equations 2.12 and 2.13.

$$MSD(u,v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2 \tag{2.12}$$

$$MSD\_Similarity(u,v) = \frac{1}{MSD(u,v)+1} \tag{2.13}$$

For KNN Baseline, the preferred approach for similarity computation was using Pearson correlation coefficients, centred using baselines, as shown in Equation 2.14. This approach yielded the most accurate rating predictions [72].

$$PB\_Similarity(u,v) = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - b_{ui}) \cdot (r_{vi} - b_{vi})}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - b_{ui})^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} (r_{vi} - b_{vi})^2}} \tag{2.14}$$

KNN algorithm was used to predict unknown ratings, $\hat{r}_{ui}$, based on the MSD similarity between users, as follows:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} MSD\_Similarity(u,v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} MSD\_Similarity(u,v)} \tag{2.15}$$

where $N_i^k(u)$ include neighbours, where the value of the similarity metric is positive.

Centred KNN algorithm, is an extension of KNN, where the mean ratings of each user are incorporated in rating prediction, as shown below.

$$\hat{r}_{ui} = \mu_u + \frac{\sum\limits_{v \in N_i^k(u)} MSD\_Similarity(u,v) \cdot (r_{vi} - \mu_v)}{\sum\limits_{v \in N_i^k(u)} MSD\_Similarity(u,v)} \tag{2.16}$$

KNN Baseline algorithm, based on [72], was the last traditional model that we studied in the neighbourhood-based category. It differs from a standard KNN approach by using the users' baseline ratings when the missing ratings are predicted, as described in Equation 2.17. Thus, when neighbourhood information is not present, this method still produces adequate results [72].

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{v \in N_i^k(u)} PB\_Similarity(u,v) \cdot (r_{vi} - b_{vi})}{\sum\limits_{v \in N_i^k(u)} PB\_Similarity(u,v)} \tag{2.17}$$

2.2.1.3 Matrix Factorisation-based Algorithms

Matrix Factorisation (MF) algorithms are another well-known approach for the rating prediction problem. These are based on latent factor models, which derive and use the latent features from existing/observed ratings to estimate the values of the missing ratings [72, 14]. In our work, we explored two MF-based recommendation systems: Non-negative Matrix Factorization (NMF) and Singular Value Decomposition (SVD). Both algorithms produce recommendations, by factorising the user and item feature vectors as shown below (Equation 2.18).

$$\hat{r}_{ui} = q_i^T p_u \tag{2.18}$$

Non-negative Matrix Factorization algorithm was implemented from [77], and follows an optimisation procedure for user and items factors (i.e., p_{uf}, q_{if}, respectively) based on Stochastic Gradient Descent (SGD). For each step in SGD, p_{uf} and q_{if} are updated as follows:

$$p_{uf} \leftarrow p_{uf} \cdot \frac{\sum_{i \in I_u} q_{if} \cdot r_{ui}}{\sum_{i \in I_u} q_{if} \cdot \hat{r_{ui}} + \lambda_u |I_u| p_{uf}} \tag{2.19}$$

$$q_{if} \leftarrow q_{if} \cdot \frac{\sum_{u \in U_i} p_{uf} \cdot r_{ui}}{\sum_{u \in U_i} p_{uf} \cdot \hat{r_{ui}} + \lambda_i |U_i| q_{if}} \tag{2.20}$$

Singular Value Decomposition algorithm, derived from [14], and also advertised in the Netflix Prize competition [43], was used to compute $\hat{r}_{ui}$ as illustrated in Equation 2.21.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{2.21}$$

2.2.1.4 Slope-based Algorithms

Another traditional CF approach that was investigated was based on Slope One schemes [1], which predict the ratings for each user by taking into account sets of relevant items, $R_i(u)$.

In **Slope One** algorithm, each unknown rating is predicted as follows:

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} dev(i,j) \tag{2.22}$$

$$dev(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj} \tag{2.23}$$

where $dev(i,j)$ is the average difference between the ratings of items i and j.

2.2.1.5 Co-clustering-based Algorithms

The last category of traditional CF included in our analysis was based on the co-clustering approach, which computes the missing ratings by utilising clusters and co-clusters of users and items.

A **Co-clustering** algorithm, as presented in [78], was used to predict $\hat{r}_{ui}$ using the average ratings in each cluster and co-cluster, as shown in Equation 2.24.

$$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i}) \tag{2.24}$$

Traditional CF approaches have been extensively utilised to successfully produce recommendations in the past few decades [14, 3]. While these methods are still being used as benchmarks in literature [37, 25], the ongoing data generation and growth have put a strain on the feasibility of these models when applied to large collections [12]. To address this, and based on the recent advances in DL studies [79], neural recommendation models were developed as presented in the next section.

2.2.2 Neural Models

In the past years, the recommendation systems community has shifted the focus from traditional methods (e.g., clustering, nearest neighbours, latent factors) to the neural ones. The success of Deep Learning (DL) and its wide applications [53], ranging from image recognition to natural language processing, has also been adopted for CF models. Consequently, we investigate representatives from multiple types of neural architectures (illustrated in Section 2.1.2.5), such as Bilateral Variational Autoencoder for Collaborative Filtering (BiVAE), Variational Autoencoder for Collaborative Filtering (VAECF), Neural Network-based Collaborative Filtering (NCF), and Convolutional Matrix Factorisation (ConvMF), as presented in the next sections. These models were selected based on popularity (i.e., how frequently they are used) and novelty (i.e., latest model from a specific DL CF family) criteria. Specifically, BiVAE [8] is the latest state-of-the-art CF model which uses a

Variational Autoencoder architecture, while the VAECF algorithm [80] has been frequently used in recommendation tasks where a VAE-based architecture was required. Furthermore, the NCF model [9] is the first and also the most popular algorithm which implements the MF technique in a neural context. Lastly, the ConvMF model [10] represents one of the most popular solutions which combines the CF technique with a CNN-based architecture.

2.2.2.1 Bilateral Variational Autoencoder for Collaborative Filtering (BiVAE) Algorithm

As we have seen from CF literature, the URM data consists of users, items, and interactions/ratings across pairs of users and items. This representation falls under dyadic data, where the values are associated with pairs of elements belonging to two distinct sets of objects (i.e., items set and users set) [81]. One of the latest neural models which aims to produce better and richer recommendations that also capture dyadic data is the Bilateral Variational Autoencoder for Collaborative Filtering (BiVAE) [8]. This approach is based on a generative model, which captures user-item interactions, and a pair of inference models (user- and item-based) parameterised using multilayer neural networks. These are then all combined together in a model that autoencodes dyadic data. One of the main differences between BiVAE and a Variational Autoencoder (VAE) architecture is that the former is "bilateral"; hence, it treats users and items symmetrically, making it suitable for the user-item dyadic interactions. Furthermore, BiVAE can also model the uncertainty on both sides of dyadic data, which, in turn, makes it more accurate on sparse URMs, compared to traditional one-sided variational autoencoders.

In BiVAE [8], depicted in Figure 2.5, the goal is to learn the input represented as an URM of size $U \times I$, denoted $\mathbf{R} = (r_{ui})$, where r_{ui} is the rating given by user u to item i. Then, notation $\mathbf{r}_{u*}$ is used to refer to the row in $\mathbf{R}$ corresponding to user u. Similarly, $\mathbf{r}_{*i}$ refers to the ith column of $\mathbf{R}$. The latent user, item respectively, variables are represented using θ_u, β_i respectively, $\in \mathbb{R}^K$.

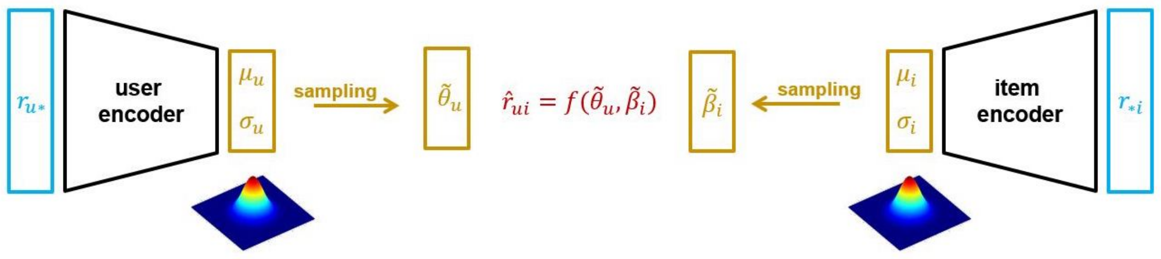

Figure 2.5: An illustration of the Bilateral Variational Autoencoder for Collaborative Filtering (BiVAE) model adopted from [7].

$$r_{ui} \sim \mathrm{EXPFAM}(r_{ui}; \eta(\theta_{\mathbf{u}}; \beta_{\mathbf{i}}; \omega)) \tag{2.25}$$

BiVAE's latent variables are drawn from prior distributions, while the observations are drawn from a univariate exponential family as seen in Equation 2.25 [8]. BiVAE also supports many well-known univariate distributions [82], such as Poisson, Bernoulli, and Gaussian. [8]. Therefore, this neural model can work with a wide range of rating types (e.g., binary, continuous) making it suitable for different recommendation tasks.

2.2.2.2 Variational Autoencoder for Collaborative Filtering (VAECF) Algorithm

One of the precursors of BiVAE model is the Variational Autoencoder for Collaborative Filtering (VAECF) for CF as presented in [80]. This neural algorithm is based on a generative model with multinomial likelihood and uses Bayesian inference for parameter estimation.

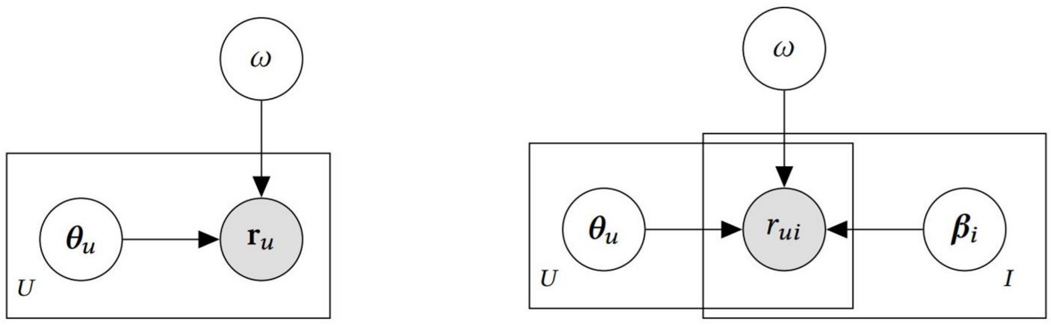

Figure 2.6: An illustration of the Variational Autoencoder for Collaborative Filtering (VAECF) model (left) compared to BiVAE (right) adopted from [7, 8].

Traditionally, VAEs have been primarily used for image processing and generation [83, 84]. However, this architecture has been successfully applied in recommendation systems as demonstrated in [80]. One of the reasons for its accuracy on CF data is related to its probabilistic nature. To this end, VAECF's goal is to learn distributions over representations in the latent space, which, in turn, allows it to model the uncertainty as well [80].

The main differences between the VAECF and BiVAE models are illustrated in Figure 2.6. Even if VAECF attains a high accuracy when compared to other neural models [80], it was originally designed for vector based-data. Therefore, it would often fail to capture the properties of dyadic data (e.g., only users are explicitly represented, while items are treated as features in a vector space of users and vice-versa as shown in Figure 2.7). However, this limitation has been addressed in the BiVAE model described above.

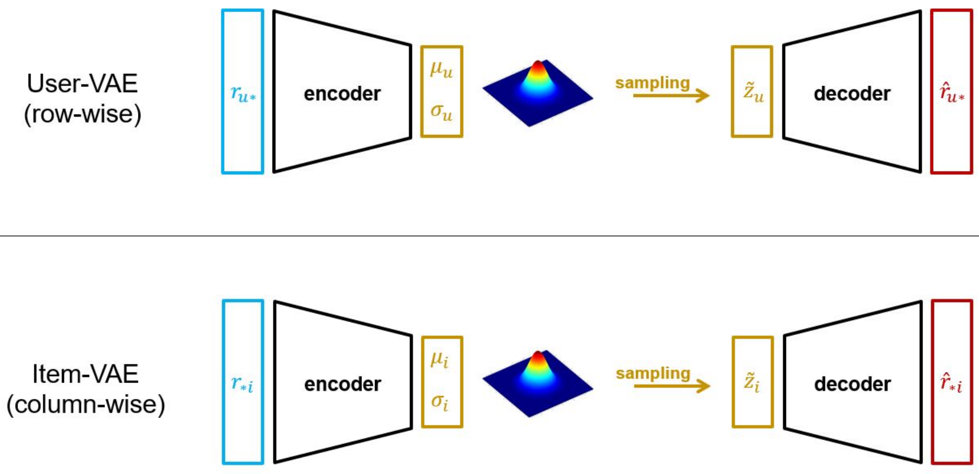

Figure 2.7: User and item representation in the Variational Autoencoder for Collaborative Filtering (VAECF) model adopted from [7].

2.2.2.3 Neural Network-based Collaborative Filtering (NCF) Algorithm

Given the popularity of the MF technique to produce recommendations, it has been further extended to incorporate neural networks. One of the most well-known algorithms which combines this approach with a Multi-layer Perceptron (MLP) architecture is the Neural Network-based Collaborative Filtering (NCF) model [9]. Although, NCF [9] has been originally designed for implicit feedback inputs, it has been extended to also work with explicit ratings. In short, this model proposes a neural architecture which replaces the inner product in traditional MF [14], and learns an arbitrary function of the data using the latent features of users, and items, respectively [9].

NCF's architecture, presented in Figure 2.8, proposes a multi-layer representation to model the interaction between users and items, denoted as y_{ui}, where the output of one layer becomes the input of the next one, and so on [9]. The bottom input layer of the model comprises two feature vectors that correspond to user u and item i, respectively. Then, above the input layer, we have the embedding layer, which is a fully connected layer that transforms the sparse representation to a dense vector [9]. We thus obtain the corresponding users/items embeddings that are equivalent to the latent vectors for users/items. The users and items embeddings are then inputted into a multi-layer neural model, which maps the latent vectors to prediction scores to produce recommendations [9]. While the model is trained, the goal is to reduce the pointwise loss between $\hat{y}_{ui}$ and the actual target value y_{ui}. Finally, the last output layer consists of the predicted score, $\hat{y}_{ui}$, which is then presented as a potential recommendation to the user [9].

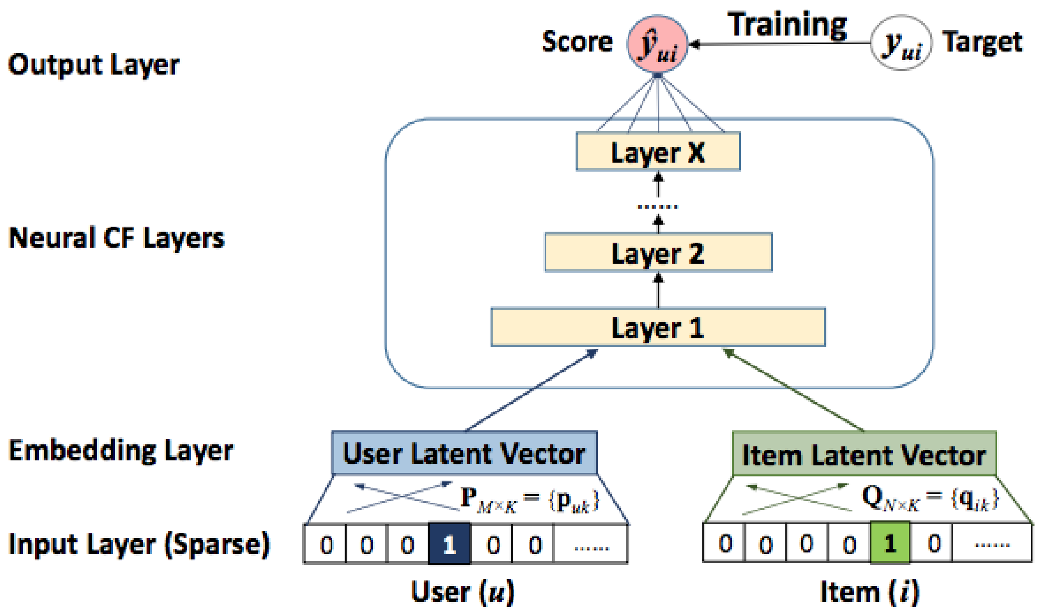

Figure 2.8: The architecture of the Neural Network-based Collaborative Filtering (NCF) model adopted from [9].

2.2.2.4 Convolutional Matrix Factorisation (ConvMF) Algorithm

Another class of Deep Learning (DL) CF models that we proposed to investigate is based on a Convolutional Neural Network (CNN) architecture. In this category, the selected representative is the Convolutional Matrix Factorisation (ConvMF) algorithm, as presented in [10], as it is one of the most frequently used models which combines the CF technique with a CNN-based architecture as noted in [85]. This model addresses the low quality of recommendations produced using sparse data, by incorporating contextual information into the rating prediction process. ConvMF [10] brings together a Convolutional Neural Network (CNN) architecture and a Probabilistic Matrix Factorisation (PMF) model.

CNN-based architectures are frequently used in different domains (e.g., computer vision [86], natural language processing [87]) as they can effectively capture features of images and/or documents through local receptive fields, shared weights, and sub-sampling (pooling) approaches [86]. However, they cannot be directly applied for producing recommendations. Consequently, the ConvMF model also utilises a PMF algorithm [88], which captures the interaction across users and items through user/item latent factors computed using inner products.

ConvMF's architecture [10], presented in Figure 2.9, comprises four layers, covering embedding, convolution, pooling, and output. The embedding layer encapsulates a document (e.g., item description) into a dense numeric matrix that is used for the next convolution layer. Then, the convolution layer is responsible for capturing contextual features. The

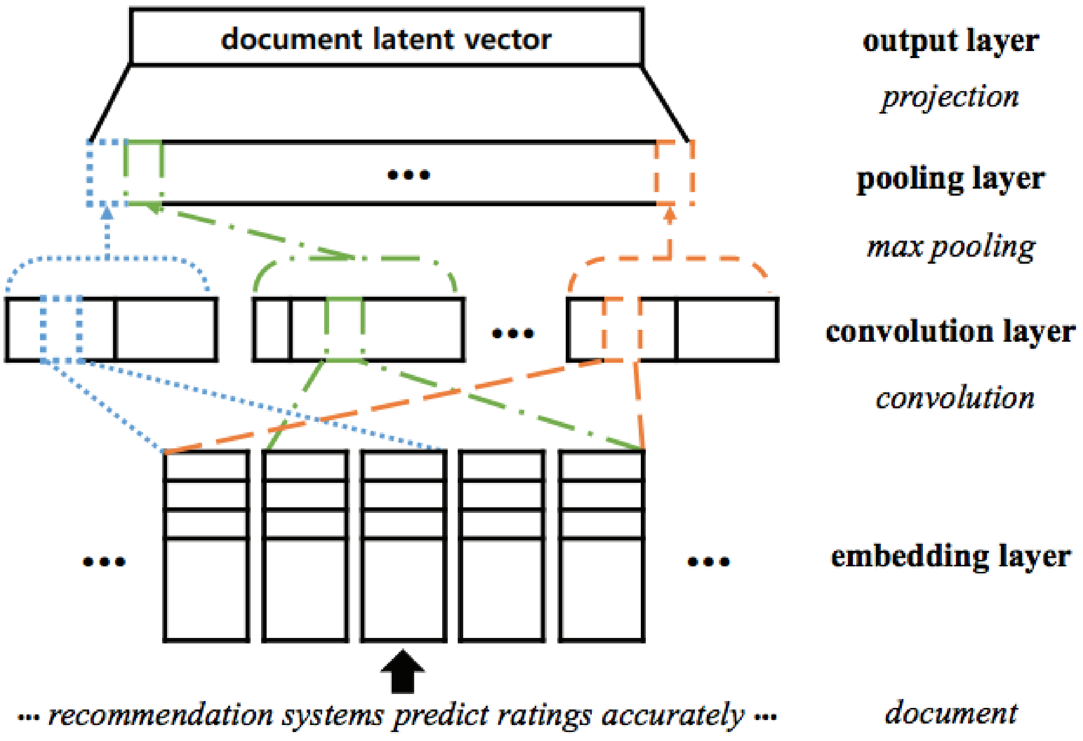

Figure 2.9: The architecture of the Convolutional Matrix Factorisation (ConvMF) algorithm adopted from [10].

pooling layer not only extracts features from the convolution layer, but also captures documents of different lengths through a pooling (sub-sampling) operation, which are further built into fixed-length feature vectors. Finally, the output layer incorporates the features processed in the previous layers into user/item latent models, which are then used to produce recommendations.

2.3 Related Work

This section covers the related work and previous studies on modelling and predicting the efficiency of CF models. To this end, we discuss the latest works that cover the efficiency challenges regarding CF models, followed by previous studies which investigate how the effectiveness of CF can be estimated. Then, we present related works on using asymptotic complexity for determining the performance of a system/algorithm, as well as experimental studies that report on the resource consumption prediction problem for various models. Finally, we present the current standard practice sampling techniques employed for gauging the effectiveness of CF algorithms.

2.3.1 Efficiency-oriented Performance Studies for CF

One of the drawbacks of CF evaluation is that the studies only report the quality of the recommendations through effectiveness metrics [62, 63], which capture how accurate are the recommendations produced for a given user. However, recent works [65, 89, 90] also present some insights into the observed efficiency (processing time) of the models. In [65], recent neural algorithms are compared to state-of-the-art traditional CF models based on accuracy, captured using precision and recall, and efficiency, reported as runtime. An interesting conclusion is revealed by the experimental evaluation presented in [65], as the traditional CF algorithms outperform the neural approaches w.r.t. both effectiveness and efficiency. Another study [89] proposes an automated tool for selecting and configuring traditional CF algorithms. This work builds on the algorithms benchmarked in [37] and reports the RMSE and MAE of the recommendations, as well as the overall training time [89]. While, some researchers directly report the runtime of their proposed models, others, such as Sun et al. [90] discuss the computational complexity of their methods, captured using the big O notation [91]. In later chapters of this thesis, we argue this is of great importance (i.e., reporting big O complexity), as the efficiency cost of the CF models can be accurately predicted using the time and space complexity of the CF algorithms.

Early works [92, 93, 94, 95] also take into account and outline the computational resources used by their models, using memory and CPU time. In [92] the authors are proposing a methodology to detect when regularised MF models need to be retrained to reflect the latest user preferences and/or the newly introduced items. To this end, they address the issue of retraining large CF algorithms without impacting their effectiveness, and report the runtime of training the models, as well as the runtime of propagating the updates. Another example, illustrated in [93], stresses the importance of outlining not only the accuracy of the recommendations, but also their computational cost, captured as runtime. Out of the surveyed methods in [93], only 11% outline the efficiency of the CF, and even fewer report the computational complexity of the algorithms. Beel et al. [93] also discuses the feasibility of using models with a high computational cost in practice, and argues that content providers would not choose models that cannot be scaled for their infrastructure. Furthermore, in [24] the runtimes of the two approaches presented vary by a factor of 600, making the less efficient model to become unsuitable for many scenarios and recommendation tasks.

As discussed above, there are increasing concerns w.r.t. efficiency of CF algorithms, as some studies are just presenting the accuracy of the proposed methods, omitting the computational costs. Thus, and in the context of environmental awareness [21], 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. Consequently, as part of solving the efficiency prediction problem, we formulate **RQ3** – *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?*. This research question is addressed in Chapter 5.

2.3.2 Effectiveness Prediction in CF Models

For the past few decades, investigating the accuracy of CF models has been deemed a very popular topic [96, 97, 98, 14]. When analysing the effectiveness of the recommendations, particular interest is noted in how the popularity bias (i.e., popularity of the items) affects the accuracy of the model. For example, Steck [98] investigated whether some of the ratings are missing at random, and how this can further impact the performance of the CF algorithms. Another study by Bellogín et al. [96] showed the presence of popularity bias when evaluating the effectiveness of the recommendations using IR methodologies. Their work [96] also proposed new experimental approaches, which aim to alleviate sparsity and popularity biases in assessing the accuracy of the recommendations. Finally, Jannach et al. [97] presented how the popularity of the items affects the output of state-of-the-art traditional CF models. This study [97] covers some of the algorithms investigated in Chapter 5 of this thesis, such as neighbourhood based methods [72, 75], SlopeOne [1], MF solutions [14], and propose ways of countering popularity biases, which impact the accuracy of the recommendations, with solutions based on algorithmic design and hyperparameter tuning.

Recent works [36, 27] also investigated how different characteristics of the URM impact the effectiveness of the CF models. In particular, popularity bias is one of the key components that can affect the accuracy of the algorithms, alongside the distribution of ratings, and the sparsity of the input. Furthermore, in [26, 25], the authors explore the effect of the structural properties of the URM regarding the accuracy and robustness (i.e., how a model tackles and removes fake item and user profiles) of the CF models used in the studies. Their results confirm a relationship between dataset characteristics and the CF models' behaviour and effectiveness. In [19], 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. Therefore, since the effectiveness of the recommendations can be quantified using characteristics of the input as demonstrated in the above literature, we propose to examine whether the efficiency of the CF algorithms can also be modelled using similar approaches. To this end, we formulate **RQ4** – *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?*. This research question is investigated and detailed in Chapter 5.

2.3.3 Using Asymptotic Complexity in Efficiency Evaluation

The performance of an algorithm can be quantified by computing its asymptotic complexity given a particular input size [91]. Asymptotic complexity represents an unbiased tool for determining the efficiency of a snippet of code, as it is hardware-agnostic. The time complexity of an algorithm outlines how its processing time increases or decreased with respect to the input size. Similarly, space complexity provides information about how much memory (i.e., RAM) an algorithm would need based on the size of the input. These complexity metrics do not capture constant factors as these are hardware-dependent [91]. Moreover, it has been proven that for sufficiently large inputs, asymptotic effects will alleviate any constant factors gains/differences [91].

One of the asymptotic complexity's purposes is to offer an estimation of the resources needed by an algorithm [91]. This idea has been explored in the past decades, with an initial investigation [99] proposing to compute the constant factors in the complexity functions to determine the runtime. However, having an appropriate time and space complexity function is critical for the accurate prediction of resources needed, as showcased in Chapter 5. Another approach [100] suggested that instead of using the theoretical complexity analysis, one can determine the performance and behaviour of a program by measuring empirical computational complexity. This method [100] requires inputs of different sizes spanning across several orders of magnitude, which are used to gauge the asymptotic behaviour of an algorithm by running it against these inputs and measuring its performance. Another study [101] addresses the runtime prediction problem for sorting algorithms by quantifying their performance with respect to the input parameters. This work [101] investigates how the processing time of a snippet of code in conjunction with its computational complexity change on different types of processor architectures. This can be particularly useful for selecting which implementation of a certain algorithm is best suited for the hardware that is available [101].

As outlined above, the use of asymptotic complexity for quantifying the efficiency of a model has been investigated for different purposes. In RS literature, reporting the complexity of the proposed algorithms is not yet a common practice [93], with very few works discussing this important aspect of the model. Furthermore, popular CF algorithms might have multiple implementations, and only the theoretical asymptotic complexity of the models is discussed in the literature. Therefore, we address this gap, by investigating and devising expected time and space complexity equations for well-known CF algorithms. This also led to the formulation of **RQ5** – *how can we use complexity analysis to estimate the resource consumption of a CF algorithm?*. This research question is further discussed in Chapter 5.

2.3.4 Resource Consumption Estimation Problem in Traditional and Neural 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 [102]. Other works focused on predicting the runtime of various planning algorithms, for selecting which algorithm to run and for how long [103, 104, 105]. 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 [106, 107].

Another area explored consists of runtime prediction applications, such as determining instance hardness [108] and parameter optimisation/tuning [106]. 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 [109] 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) [109]. To this end, existing solutions [109] 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 [110, 111] focus on computing cost models for estimating the resource consumption of processing different types of queries. We believe the same scenarios apply to CF models, 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 the experimental chapters of this thesis.

Furthermore, the problem of having access to limited computational resources is worsening for DL models. Everyday, there are numerous neural jobs that are executed on clusters and fail because they require powerful computational resources that are not always available [22]. One of the main resources that quickly becomes exhausted in a DL job is GPU memory [22]. This often happens due to the complexity of the structure of the neural network, which often requires many epochs and large batch sizes, so that it can accurately learn representations of the input data [22]. Moreover, a study of thousands of Stack Overflow posts revealed that GPU memory consumption and processing time (i.e., runtime) are two of the biggest problems associated with training DL models [23]. Therefore, the problem of estimating the resource consumption of neural algorithms becomes of critical importance as the computational resources are finite [112]. To this end, we aim to answer **RQ6** – *given the processing times and GPU memory overhead of a neural CF algorithm on a smaller sample* *of the URM, how can we estimate the overall processing time and GPU memory of the trained algorithm on the complete dataset?*. Additionally, knowing when a neural algorithm has achieved the desired accuracy, or which DL model would minimise the loss in a quick, yet accurate way, would alleviate problems associated with overtraining the model, and would also limit the computational resource consumption [112]. This led to the formulation of **RQ7** – *given the effectiveness of a DL CF algorithm on a smaller sample of the input, how can we predict the overall accuracy of the model on the full collection (URM)?*. Both RQ6 and RQ7 are further investigated in Chapter 6.

2.3.5 Sampling Approaches for CF Evaluation

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 [62]. While there are still ongoing debates about which evaluation methodology is more suitable (i.e., online versus offline evaluation) [63, 62], 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 [63, 62]. This also motivated us to adopt the offline evaluation methodology throughout the experimental chapters of the thesis. One of the key aspects of offline evaluation in CF is 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 [96]. 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 [113, 114, 115]. Other works focused on computing the optimal bound for the number of samples needed for satisfying a predefined error metric [116, 117, 118]. Furthermore, efficient sampling techniques have been developed to address the limited computational resources availability for analysing large datasets [119]. 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 impact the accuracy of the predictions [120].

In CF, random sampling techniques have been utilised to create inputs (e.g., URMs) with different characteristics. For example, Adomavicius et al. [26] proposed a random sampling procedure, which is based on the density of the desired sample. This method [26] works by selecting a random subset of items and users alongside the corresponding ratings from the URM, which is then filtered using a density threshold. This constraint is imposed to ensure that the selected samples have enough data to reflect the characteristics of the original dataset [26]. A similar sampling approach is also used by Deldjoo et al. [25] to mitigate shilling attacks against CF. In their work [25], random samples are drawn from the URM in a similar manner to [26] to simulate various attacks as well as investigate how the characteristics of the input impact the reliability of the CF models.

As we have seen from [26, 25], random sampling techniques have proven to be a reliable approach to gauge the performance (i.e., accuracy, safety) of CF. Consequently, this approach was probed to investigate whether it could be used to also quantify the efficiency of CF algorithms. This led to the formulation of **RQ1** – *how should we sample the base data (i.e., URM), such that the quality of the predicted efficiency/effectiveness of a CF model is not harmed?*, as well as **RQ2** – *given an upper sample size S% from a dataset, how do we determine when to stop sampling based on the number of samples for which we obtain consistent resource usage measurements (e.g., the processing time values are in a tight interval)?*. Both research questions, as well as the challenges associated with sampling for efficiency and effectiveness prediction, are discussed and addressed in Chapter 4.

2.4 Chapter Summary

In this chapter, we provided an overview of the Information Filtering Systems (IFS), particularly focusing on the Recommendation Systems (RS) field. To this end, we covered the different types of inputs and recommendation tasks that arise in RS, as well as the three main methodologies (i.e., Collaborative Filtering (CF), Content-based Filtering (CBF), Hybrid Approaches (HA)) for producing recommendations. The chapter continued with a review of the various evaluation methods and metrics for CF models for both efficiency and effectiveness purposes. Then, we discuss the most well-known Deep Learning (DL) architectures that are also found in neural CF algorithms. Section 2.2 presented the chosen CF representatives for which we will investigate the efficiency estimation problem in Chapter 5 and Chapter 6. Finally, Section 2.3 discussed the previous studies and related works that covered the performance evaluation of various CF algorithms in several scenarios. Furthermore, throughout this section, we linked the proposed research questions (RQs) with the relevant literature. In Section 2.3, we noted the lack of efficiency-oriented studies for CF, and argued for using asymptotic complexity to determine the training cost of the CF algorithms, as well as presented related work that supports the feasibility of this approach. Moreover, Section 2.3 illustrated how the resource consumption prediction problem is addressed by different research communities, and highlighted the current solutions and their limitations. Finally, we reviewed how current standard practice sampling techniques found in the literature can be used for gauging the effectiveness of the recommendations.

Chapter 3

Methodology

This chapter presents an overview of the proposed methodology for estimating resource consumption (i.e., processing time, RAM, GPU) of Collaborative Filtering (CF) algorithms during their training phase. In Section 3.1, we outline the current standard practice approaches for predicting CF performance, and highlight the differences between this approach and the proposed cost models. Then, in Section 3.2, we conceptualise the methodology for estimating resource consumption, and present the White- and Black-Box approaches, noting their advantages and limitations. Section 3.3 discusses the methodology for predicting the recommendations' quality in CF, and describes how we utilised the off-the-shelf Black-Box regressors to solve this problem. In Section 3.4, we illustrate the two frameworks, Surprise [37] and Cornac [38], which we used to analyse the selected CF representatives focusing on both traditional and neural approaches. Finally, Section 3.5 describes the hardware and experimental environment used for our studies, the datasets on which we trained the CF and probed our methodologies, as well as the implementation details for the cost models developed in this thesis.

3.1 Overview of Cost Models for CF

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 iteratively by: (i) randomly sampling over all ratings in the dataset, (ii) training the CF over the sample, (iii) evaluating its effectiveness over the sample [25]. We follow a similar strategy with a few notable changes summarised in Figure 3.1. 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.

Based on Figure 3.1, given get 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,

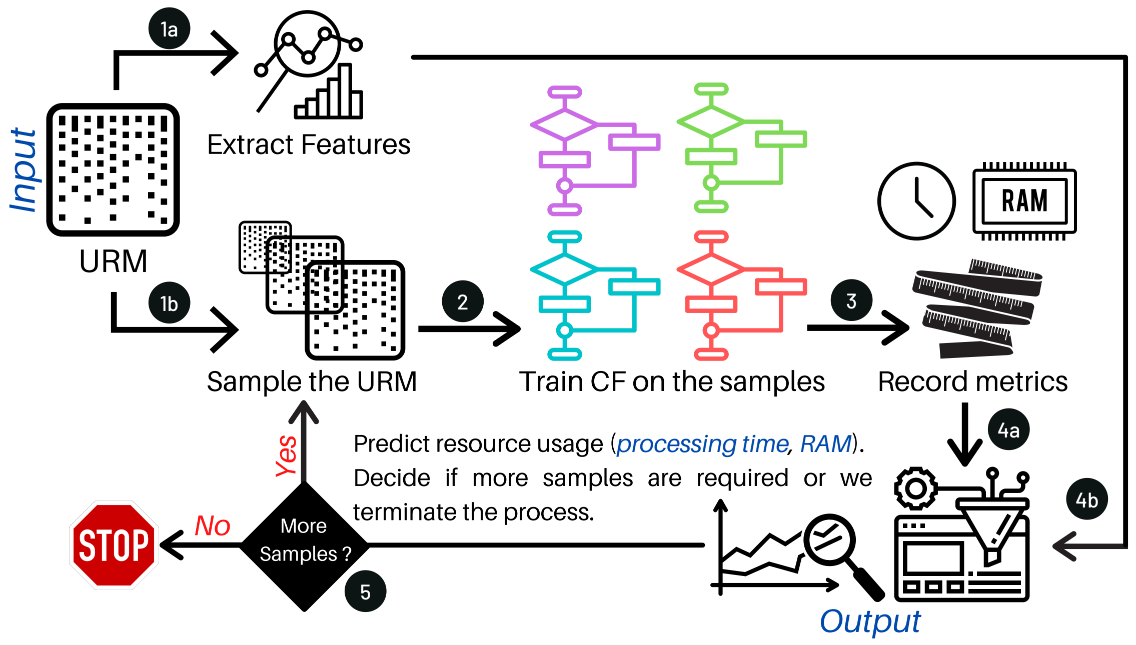

Figure 3.1: Overview of the proposed pipeline for sampling the input and training the CF models, while gathering performance metrics.

etc.; and (step 1b) sample the URM following the strategy described in Section 4.3 and Section 4.4. In step 2, we train the various classes of CF models 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 (e.g., RMSE for the predicted rating values). In steps 4a and 4b, we train our proposed prediction models, detailed in Chapter 5 and Chapter 6, 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).

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 3.1 step 1b) described in Section 4.3.

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. 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 [121], to recursively subsample to produce even smaller samples. The time complexity of the sampling scheme is highly dependent on the size of the input, as sampling from a sample is much faster/lighter than sampling from the complete dataset. Moreover, our proposed 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 3.1 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 (Figure 3.1 step 5).

Furthermore, the number of samples we draw should be based on *(a) a user-defined upper limit* and *(b) a user-defined metric of accuracy*, which in our case, it is quantified using an upper bound to the *Coefficient of Variation (CoV)* (depicted in Equation 3.1, where σ is the standard deviation and μ is the mean) of the resource usage of CFs trained over subsamples, as a proxy of how tightly said values are distributed [122].

$$CoV = \frac{\sigma}{\mu} \tag{3.1}$$

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 Section 4.5 for the related results).

3.2 Resource Consumption 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 [91]. However, in practice, the likelihood of encountering inputs that elicit the worst-case processing time of an algorithm is relatively low [91]. 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 proposed methodology (i.e., White-Box approach) is based on *approximating probabilistic analysis*, through an adaptive sampling strategy (Chapter 4), 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.

However, a different approach towards predicting the efficiency of a CF model is to treat it like a black-box. To this end, one would sample the input (e.g., URM), and measure the processing time and memory requirements of the CF on the samples. Then, these measurements would be used with off-the-shelf state-of-the-art regressors to learn the resource consumption of CF algorithms, and then predict its efficiency on the full dataset.

Throughout this chapter, we will denote this methodology as the Black-Box approach. In the following sections, we will describe in further detail the proposed White- and Black-Box approaches, including their advantages and limitations.

3.2.1 White-Box Approach

Using the time and space algorithmic complexities outlined in Section 5.2, the processing times and memory usage measured across different inputs, and the characteristics of the data (i.e., number of users, items, ratings, the density of URM etc.), we propose the *White-Box approach*, which is based on estimating the hidden factors (or unknown parameters) in the time and space complexity equations derived from the algorithms' implementations. 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 [123]. 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.

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:

$$performance = f(X) = \alpha X + \beta \tag{3.2}$$

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 in Section 5.4.

The features of the White-Box approach are the number of users, number of items, and number of ratings present in the sample drawn from the URM, as well as the resource we would like to estimate (e.g., time, memory). The goal of the model is to learn the previously mentioned slope and intercept hidden factors in the complexity equations based on these input features. For example, to estimate the processing time for a CF, initially, the White-Box approach will compute the slope and intercept for the 100% sample size using the number of users, items, ratings, and processing times recorded for the 10% and 20% sample sizes. We note that we need the measurements and features from at least two different sample sizes to be able to solve the equations system to compute the unknown slope and intercept. Then, if the user-defined operational constraints (e.g., time quote, upper sample size) permit, we will recompute the slope and intercept using the features from the 10%, 20%, 30% sample sizes and so on until we have reached the upper sample size or we hit the time budget. Once the slope and intercept have been learnt using the White-Box approach, we plug them into the complexity equation of a CF, alongside the other features (number of users, items, ratings) to estimate the processing time and/or the memory as presented in Algorithm 1. For example, let us assume that a CF has a time complexity of O(m+n) and the slope we computed has a value of 0.5, while the intercept has a value of 8. Then, the total processing/training time of the algorithm on a dataset with 50 users (m) and 120 items (n) is estimated to take $(50 + 120) \times 0.5 + 8 = 93$ seconds.

Additionally, to quantify the uncertainty of our predicted time and memory usage using a White-Box linear regression model, we compute the *prediction error interval* [124, 125]. 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[1] [126, 125].

Apart from White-Box linear regression on the complexity equations, we also investigated if off-the-shelf polynomials can be utilised to predict the processing time and space required by the CF. As we know that the complexity of an algorithm could be described by a polynomial, another solution for our problem would have been to learn the hidden factors/coefficients in the equations by fitting polynomials of various degrees on the input data (i.e., number of users, items, ratings, and time/space for each sample size). However, this approach did not yield the expected results, as (a) several polynomials with different degrees had to be fitted to the data to try and find the one that best describes the expected resource consumption of the CF at each sample size, and (b) for each polynomial a different coefficient had to be learnt for each term, which, in turn, translates into higher processing costs for our predictors. On the other hand, using linear regression, we only had to learn two coefficients, slope and intercept, and this approach was also the cheapest and most accurate for estimating processing time and memory of the CF, as depicted in the experimental evaluation (Section 5.4).

Another approach that we explored was to estimate the efficiency of the CF algorithms using *Bayesian inference* [127]. In this setup, our aim is still to compute the hidden coefficients of the complexity equations from Section 5.2, 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 [127], characterised by mean and variance, as seen below:

[1]An example of 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`.

Algorithm 1: White-Box Resource Consumption Estimation Algorithm

```
Input  : Dataset D, Algorithm A, max sample size S, max time budget T
Output: The time and memory consumption of the CF.
resource_consumption = []; stats = [];
T' = T; S' = S; D' = D;
/* initialise White-Box regressor                                        */
WB = White-Box.init();
/* compute number of users, items, ratings of the 100% sample           */
m, n, $\rho$ = compute_features(D);
/* compute the complexity equation for a given CF                       */
Function compute_equation(m, n, $\rho$):
    /* insert the complexity equation of the CF                          */
    /* for example the time equation for NMF is m+n+$\rho$                */
    return (m+n+$\rho$);
end
do
    /* set sample size and time budget for current iteration            */
    T_cur = T'; S_cur = S';
    /* sample D' with the constraints                                   */
    d = sample(D', S_cur);
    /* compute input features of the current sample size                */
    m', n', $\rho$' = compute_features(d);
    t = 0;
    do
        t_start = time();
        /* measure processing time, memory by training A on d           */
        stats.push(get_memory_and_processing_time(A, d));
        /* train the White-Box model on the input features and the
           resources measured and compute the slope and intercept       */
        slope, intercept = WB.fit(complexity_equation(m, n, $\rho$), stats);
        /* predict resource consumption for the full dataset using the
           slope and intercept                                          */
        estimated_resource = slope$\times$(compute_equation(m', n', $\rho$')+intercept;
        resource_consumption.push(estimated_resource);
        t_end = time();
        t += (t_end - t_start);
    while (t <= T_cur);
    T' -= t; D' = d; S' = S_cur;
while (T' > 0);
return (resource_consumption)
```

$$performance \sim \mathcal{n}(\alpha X + \beta,\, \sigma^2) \tag{3.3}$$

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 [127], 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 [128].

In Bayesian inference, the main goal is to use sampling methods to draw samples from a distribution to approximate the posterior [127]. According to the standard practice [127, 128], we can use Monte Carlo methods [129] 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 [127, 128] is Markov Chain Monte Carlo (MCMC) sampling [130][2]. 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 [127]. The rule of thumb is that convergence has been achieved when $\hat{R}$ is very close to 1.0 [128].

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 [128]. MCSE can be used to quantify the *uncertainty of the predictions* for processing time and memory usage in MCMC models by computing the standard deviation and variance around the posterior mean of the samples[3] [128, 131].

Moreover, the uncertainty and the prediction error interval of our White-Box models allow us to measure and control the quality of the estimated resource consumption for CF algorithms, without having to collect or rely on the ground truth data, as they provide probabilistic upper and lower bounds on the estimate of the efficiency of the CF. Consequently, to know whether more or larger samples are required to improve the accuracy of the estimated processing time and memory overhead, one would analyse the uncertainty/prediction error interval and observe the range of the predicted resources (i.e., processing time, memory) based on the current sample size, and decide if further training is needed for the White-Box models. Generally, as the accuracy of the predictions improve, the uncertainty/prediction error interval would become tighter [128]. This can also be noted in Section 5.4, where we present an experimental evaluation that includes a discussion regarding the accuracy of the predicted processing time and memory overhead based on the uncertainty and prediction error interval.

Besides the uncertainty and prediction error interval, we also evaluated our predictions for processing time and memory overhead using normalised RMSE (NRMSE) computed as:

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

[3]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`.

$$NRMSE = \frac{RMSE}{\overline{y}} \tag{3.4}$$

where $\overline{y}$ is the mean of the actual time/memory values in the corresponding input.

3.2.2 Contenders and Black-Box Approach

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 (denoted WB/Lit/LR in Section 5.4); 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).

In the *Black-Box approach*, 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, $density$) of the Black-Box approach, with rating distribution/frequency-related features. Specifically, in line with the practice in the state-of-the-art methods for predicting the performance of CF [25, 26], we modelled the concentration of users' (items', respectively) ratings by using the **Gini** coefficient [132] as described in Equation 3.5, where $\mathfrak{w}$ is the number of users (items, respectively), $\rho_{\mathfrak{k}}$ is the number of ratings given by a user (or received by an item, respectively), and ρ_{total} is the total number of ratings.

$$Gini_w = 1 - 2 \times \sum_{k=1}^{\mathfrak{w}} \left(\frac{\mathfrak{w} + 1 - k}{\mathfrak{w} + 1} \right) \times \left(\frac{\rho_{\mathfrak{k}}}{\rho_{total}} \right) \tag{3.5}$$

We thus compute the Gini coefficient for users, $Gini_{users}$, and items, $Gini_{items}$, and include them as extra input features for the Black-Box approach.

Compared to the White-Box approach, in the proposed Black-Box models, the goal is to learn the CF resource consumption (i.e., time, memory) directly, without computing hidden coefficients of the complexity equations. As seen in Algorithm 2, the Black-Box approach estimates the efficiency of CF based on the input features, captured as the structural characteristics (i.e., number of users, items, ratings, density, Gini coefficients), of the samples drawn from the URM. For example, given the memory consumption of a CF on the 10% and 20% samples, we will fit the Black-Box regressor on these measurements alongside the number of users, items, ratings, and Gini coefficients of the 10% and 20% samples, and once the Black-Box regressor has been trained, we will predict the memory consumption for the 100% sample size (i.e., full dataset) using its structural features. This process is then repeated, adding more sample sizes (e.g., 30%, 40%, ..., upper sample size S) until the maximum time quota is reached.

Algorithm 2: Black-Box Resource Consumption Estimation Algorithm

```
Input  : Dataset D, Algorithm A, max sample size S, max time budget T
Output: The time and memory consumption of the CF.
resource_consumption = []; stats = [];
T’ = T; S’ = S; D’ = D;
/* initialise Black-Box regressor                                        */
BB = Black-Box.init();
/* compute features of the 100% sample                                   */
m, n, $\rho$, dens, g_i, g_u = compute_features(D);
do
    /* set sample size and time budget for current iteration             */
    T_cur = T’; S_cur = S’;
    /* sample D’ with the constraints                                    */
    d = sample(D’, S_cur);
    /* compute input features of the current sample size                 */
    m’, n’, $\rho$’, dens’, g_i’, g_u’ = compute_features(d);
    t = 0;
    do
        t_start = time();
        /* measure processing time, memory by training A on d            */
        stats.push(get_memory_and_processing_time(A, d));
        /* train the Black-Box model on the input features and the
           resources measured                                            */
        BB.fit(m, n, $\rho$, dens, g_i, g_u, stats);
        /* predict resource consumption for the full dataset             */
        resource_consumption.push(BB.predict(m’, n’, $\rho$’, dens’, g_i’, g_u’));
        t_end = time();
        t += (t_end - t_start);
    while (t <= T_cur);
    T’ -= t; D’ = d; S’ = S_cur;
while (T’ > 0);
return (resource_consumption)
```

The Black-Box approach was implemented and tested using several off-the-shelf state-of-the-art regression algorithms available through the H2O AutoML analytics platform[4] [133]. This platform allowed us to build Black-Box regressors based on many well-known statistical methods and ML algorithms, as well as characteristics of the input (i.e., URM). A few examples of the tested models include Random Forest, Deep Neural Networks, Support Vector Machine (SVM), and Adaptive Boosting. These Black-Box regressors were ranked based on their performance, using the $sort\_metric$ parameter in AutoML, which was set to

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

$RMSE$. Other parameters that were amended in AutoML were the maximum number of seconds, $max\_runtime\_secs$, and the maximum number of models, max_models, tried for the regression task. For the former, we experimented with values between 30 seconds and 1 hour, converging on a value of 5 minutes (300 seconds) as larger values would go over the maximum time quota T set for estimating resource consumption of the CF, while for the latter parameter we used a value of 10 models.

However, as demonstrated in Section 5.4, solutions based on the Black-Box method are less accurate than those based on computational complexity equations (i.e., White-Box approach), and hence, we only report on the results of the best Black-Box performer with regards to prediction accuracy (i.e., lowest RMSE), namely Gradient Boosting Machine (GBM) [134]. GBM was ranked as the best state-of-the-art regression model, given the previously described parameters utilised for AutoML [133], since it acquired the lowest RMSE, following the K-fold cross-validation procedure described in [133].

3.3 Recommendations' Quality Prediction Models

The previous section described how the efficiency of the CF can be predicted using (a) methods that capture the underlying computational complexity of the CF algorithms, and (b) methods that learn the resource consumption of the said CF models based on different structural and rating frequency-based features of the input (i.e., URM).

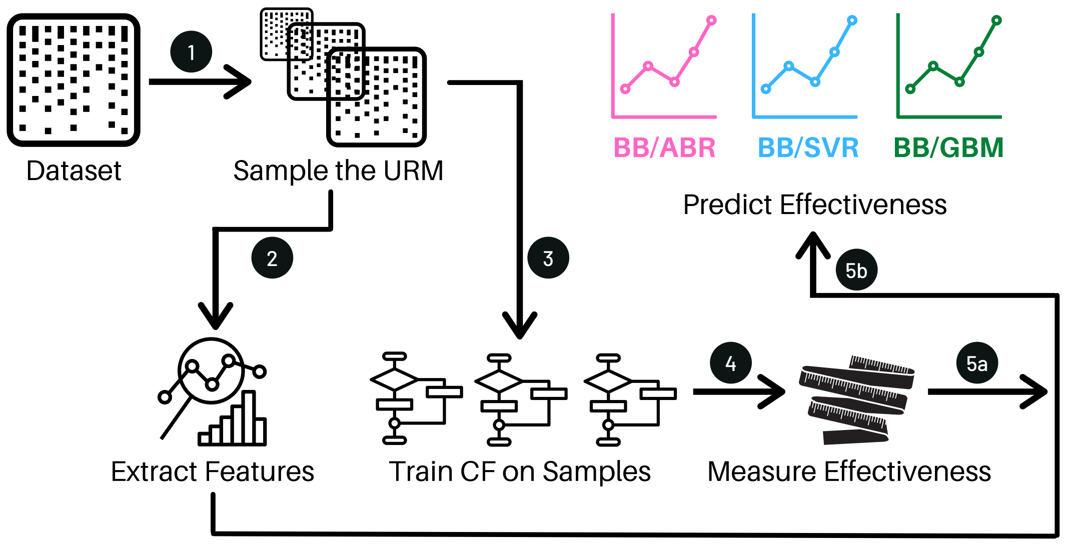

Figure 3.2: Overview of the Black-Box methodology for predicting the effectiveness (i.e., recommendations' quality) of CF.

Algorithm 3: Black-Box Recommendation Quality Estimation Algorithm

Input : Dataset D, Algorithm A, max sample size S, max time budget T

Output: The quality of the recommendations produced by the CF.

```
quality = []; stats = [];
T' = T; S' = S; D' = D;
/* initialise Black-Box regressor, which could be GBM, SVR, ABR */
BB = Black-Box.init();
/* compute features of the 100% sample */
m, n, $\rho$, dens, g_i, g_u = compute_features(D);
do
    /* set sample size and time budget for current iteration */
    T_cur = T'; S_cur = S';
    /* sample D' with the constraints */
    d = sample(D', S_cur);
    /* compute input features of the current sample size */
    m', n', $\rho$', dens', g_i', g_u' = compute_features(d);
    t = 0;
    do
        t_start = time();
        /* measure the quality of recommendations by training A on d */
        stats.push(get_recommendation_quality(A, d));
        /* train the Black-Box model on the input features and the
           effectiveness of the CF */
        BB.fit(m, n, $\rho$, dens, g_i, g_u, stats);
        /* predict quality of recommendations for the full dataset */
        quality.push(BB.predict(m', n', $\rho$', dens', g_i', g_u'));
        t_end = time();
        t += (t_end - t_start);
    while (t <= T_cur);
    T' -= t; D' = d; S' = S_cur;
while (T' > 0);
return (quality)
```

As the computational complexity only allows us to determine an upper bound to the way an algorithm's processing time grows or declines with respect to the size of its input [91], it cannot be utilised to gauge the effectiveness/quality of recommendations for the CF models. However, in line with the practice in the state-of-the-art methods for predicting the performance of CF [25, 26], we can employ off-the-shelf regressors using the Black-Box approach to estimate the effectiveness of the neural CF using characteristics of the input and the effectiveness measured across different samples, as seen in Figure 3.2.

First, we sample the dataset (step 1) and extract the structural URM features (step 2), such as the number of users m, number of items n, number of ratings ρ, and density, alongside frequency-related URM features, such as the *Gini* coefficient for users/items [132]. Then, we train the CF on the samples (step 3) and record quality of recommendations metrics (step

4) (i.e., RMSE for rating prediction, NDCG for ranking). The next step is to fit the Black-Box regressors on these measurements (step 5a) and then predict the overall effectiveness of the CF using the input features (step 5b) for the 100% sample. This process is repeated until the upper sample size has been reached or we until we hit the maximum time budget, as illustrated in Algorithm 3. The Black-Box approach for estimating effectiveness was implemented using several off-the-shelf regressors, such as Ada Boost Regressor (ABR), Support Vector Regressor (SVR), and Gradient Boosting Machine (GBM), build using the H2O analytics platform [133] as described in 3.5.3. These three models were ranked as the top performers, acquiring the lowest RMSE, following the K-fold cross-validation procedure described in [133].

3.4 CF Frameworks

In this section, we present a popular CF framework, Surprise [37], that implements the various traditional CF models studied in Chapter 5 and outlined in Section 2.2.1. Then, we describe Cornac [38], a DL-based recommendation framework, which we utilised for analysing and training the various neural CF representatives illustrated in Chapter 6.

3.4.1 Surprise

Surprise [37] 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, as there are many well-known CF algorithms implemented in this engine, as well as various tools to assess the models' performance. Surprise also allows users to experiment with built-in datasets (e.g., Movielens [135]), but also to incorporate their bespoke collections. For the comparative performance assessment of the efficiency of the CF representatives, we used the latter, as the algorithms were trained on inputs sampled in a custom manner, described in Section 5.3.2.

Surprise engine comprises many ready-to-use traditional CF models for solving the rating prediction problem [3]. As part of the scope of Chapter 5 and the efficiency cost methodology for traditional CF, we analyse and experiment with various algorithms provided by Surprise. As this framework covers a wide range of CF classes, such as Alternating Least Squares (ALS), Matrix Factorisation (MF), K-Nearest Neighbours (KNN), etc., we selected all the available representatives in each category and measured their efficiency (i.e., processing time, RAM overhead), and effectiveness (i.e., RMSE).

3.4.2 Cornac

Cornac [38] is an open-source Python-based framework designed for multimodal CF. This recommendation engine comprises multiple state-of-the-art models, including traditional and neural ones. One of the advantages of Cornac is that is allows users to quickly build and train CF algorithms on both well-known datasets, as well as custom external collections. Moreover, when using this framework, each CF can be assessed with numerous effectiveness-oriented metrics, such as Precision (P), Recall (R), Root Mean Squared Error (RMSE), and many others. In addition, Cornac supports explicit and implicit feedback datasets, as well as multimodal auxiliary data, such as clicks, items' descriptions, etc., which can enrich the user-item interaction information.

We chose this CF engine for undertaking the proposed DL CF work, as it includes the latest and the original implementation of many well-known neural algorithms. Furthermore, Cornac is built on and compatible with existing ML tools and libraries, such as PyTorch [136] and TensorFlow [137]. Out of the numerous neural CF models, we selected the latest model for the VAE architecture, namely BiVAE, as well as the original VAE-based CF, VAECF. Then, we also included the well-known NCF algorithm based on a MLP architecture, and a CNN-based CF, namely ConvMF. These models were selected based on popularity and novelty criteria as detailed in Section 2.2.2.

3.5 Experimental Apparatus

This section describes the experimental apparatus used in the experimental evaluations presented in this thesis. Specifically, we discuss the hardware and environment used to obtain the empirical results, for both traditional and neural CF, as well as the datasets on which we tested our sampling algorithm and the proposed White- and Black-Box methodologies. In addition, we also present the implementation details for the proposed efficiency and effectiveness cost models for CF.

3.5.1 Experimental Environment

3.5.1.1 Traditional CF

The experiments related to sampling the URM (Chapter 4) and training the traditional CF (Chapter 5) 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.

3.5.1.2 Neural CF

All neural CF experiments were carried out on an OpenShift cluster [138], orchestrated with Kubernetes [139], and which contains nodes with Intel Xeon Gold 6244 processors and 500 GB of RAM, and run on Ubuntu Linux 18.04. For the GPU cards, we utilised Titan RTX with 24 GB of memory and CUDA 11.4. The Python scrips built for logging the processing time, GPU utilisation, and other metrics were encapsulated in Docker containers [140] with custom Docker images, where additional libraries were installed. Finally, the scrips used to train and record efficiency/effectiveness of the various neural CF were scheduled to run on the cluster as different jobs, having specific YAML [141] configuration files that reflected their hardware requirements (i.e., number of CPU cores, RAM quota, number of GPU cards).

3.5.2 Datasets

For the studies conducted in Chapters 4, 5, and 6, we used the MovieLens (ML) 100K and 1M collections [135], as well as the GoodBooks (GB) 10K dataset [142]. 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, 9724 items and 10^5 ratings, while ML 1M is a larger dataset with 6040 users, 3706 items, and 10^6 ratings. In addition, GoodBooks (GB) 10K has 53424 users, 10000 items, and 6×10^6 ratings.

For all collections, the corresponding rating densities were computed, and these are 0.017 (ML 100K), 0.045 (ML 1M), and 0.012 (GB 10K) respectively. The different characteristics and sizes of these datasets allowed us to experiment with our proposed sampling tool and methodology, showing that it can be successfully used on collections with different properties.

3.5.3 Implementation

The proposed White- and Black-Box methodologies were implemented in Python 3.7 [143] using regular scripts (i.e., *.py) for sampling the URM, measuring the resource consumption and quality of recommendations. The recorded measurements were stored into Comma Separated Values (CSV) files, and processed with Jupyter Notebooks [144] and Pandas dataframes [145]. The specific implementation details for measuring efficiency in CF are further described in Section 5.3 for traditional models, and in Section 6.3 for neural algorithms, respectively.

The White-Box models were implemented using the Linear Regression model from SciKit-Learn library [146]. When fitted on the input data, the regressor computes and returns the slope and intercept from the corresponding complexity equations. Then, we use these coefficients with the input features (number of users, items, ratings) to predict the processing time and memory required on the full dataset. The other type of White-Box predictor, Bayesian Regression, was implemented using the PyMC3 library [147]. This framework allowed us to build a White-Box model that learns the hidden terms in the complexity equations as probability distributions rather than point estimates used in linear models. While these models can accurately predict the resource consumption of CF, as showcased in Sections 5.4 and 6.4, they are very expensive w.r.t their training requirements. This behaviour was also highlighted in Table 5.3, where the Bayesian models are several orders of magnitude slower than the rest of the contenders. This is due to the parameter tuning process in Bayesian regressors, which is often complex, requiring expert knowledge on selecting the priors and the number of samples drawn from the posterior to attain model convergence, based on the $\hat{R}$ value, as described in Section 3.2.1. When building the White-Box Bayesian models, we used normal distributions as priors to represent the slope and intercept parameters, and a value of 500 samples for the posterior computation. Other sampling configurations were explored, where values smaller than 500 lead to large values of $\hat{R}$, which indicate that convergence was not attained. On the other hand, larger values, above 500, for sampling the posterior, were more expensive w.r.t processing time and led to similar slope and intercept values to the ones computed using 500 samples.

Finally, the Black-Box approach was implemented using the H2O AutoML framework [133]. Initially, for choosing the best off-the-shelf regressors, we selected RMSE as the metric to rank the different algorithms, since these would be used in a prediction task and the goal was to minimise the error/loss between the estimated value and the ground truth data. The RMSE of the Black-Box predictors was computed over multiple iterations using a K-fold cross-validation (K=10) procedure described in [133]. The AutoML library was initialised with the default parameter values, apart from the maximum number of seconds, $max\_runtime\_secs$, and the maximum number of models, max_models, which were amended, to 5 minutes (300 seconds) and 10 models, respectively, to reflect the experimental conditions described in Section 3.2.2. Using these settings, the best performer in the resource estimation for CF task was BB/GBM, while for effectiveness (quality of recommendations), the top three models were BB/ABR, BB/SVR, and BB/GBM. To train these models on the input features of the URM, we used the $train()$ method, and for estimating the efficiency and effectiveness of the CF algorithms, we used the $predict()$ method, outlined in [133].

3.6 Chapter Summary

This chapter introduced and formalised the White- and Black-Box methodologies for estimating the efficiency and effectiveness of CF. We also discussed the current standard practice approaches for predicting the performance of CF, and noted how the sampling strategy needs to be updated to reflect both the structural and complexity characteristics of the base data. For each proposed cost model (i.e., White-Box efficiency, Black-Box efficiency, Black-Box effectiveness), we provided an overview of the theoretical concepts used by each approach, as well as illustrated how these can be implemented using pseudocode. Finally, the apparatus used in the experimental chapters was presented, covering the CF frameworks, datasets, hardware, and implementation details.

Chapter 4

Sampling Strategies and Workloads for Evaluating CF Performance

This chapter presents the sampling strategies which can be employed to determine the performance of CF models, quantifying both efficiency and effectiveness. Thus, we firstly introduce the problem of how we can draw samples from the URM, while maintaining the structural properties of the input, as discussed in Section 4.1. Then, Section 4.2 covers the standard practice sampling techniques used in the CF literature, noting their limitations for quantifying the efficiency of a CF algorithm. Section 4.3 illustrates the proposed sampling approach, which allows us to determine both the efficiency and effectiveness of CF models, while preserving the structural properties of the input. We further augment the proposed sampling approach with an adaptive component, which determines how many sub-samples are required within each sample size based on their quality and time budget, as described in Section 4.4. Section 4.5 discusses the comparative performance assessment of the standard practice sampling strategy and the proposed approach, providing empirical answers to **RQ1** and **RQ2**. Finally, a summary of the chapter is outlined in Section 4.6.

4.1 Problem Overview

Drawing random samples from a population to make predictions about the entire population is a very popular technique in many areas, including CF [14, 3]. To this end, random triplets of the form $(user, item, rating)$ would be chosen from the input (i.e., URM), with the goal of emulating the characteristics of the entire URM. Sometimes the quality and number of the samples can be filtered based on a predefined characteristic of the input (e.g., density), as showcased in [25, 26]. To the best of our knowledge, in CF literature, all the standard practice sampling approaches are used to only quantify the effectiveness of CF (i.e., the quality of the recommendations), without addressing the efficiency of the models. To this end, we investigated if the current sampling approaches [25, 26] can be used to determine the computational cost of CF algorithms during their training phase. However, the performance evaluation (Section 4.5) revealed that the standard practice sampling technique does not work well for drawing samples to predict the efficiency (i.e., processing time, memory) of the CF models. This lead to the formulation of the following problem: *given an URM and a computational resource budget, we need a sampling methodology that preserves the complexity and structural characteristics of the input, such that we can further utilise the samples to estimate the efficiency and effectiveness of CF algorithms.* While solving this research problem, we also answered **RQ1** and **RQ2**.

4.2 Standard Practice Sampling Techniques

This section presents the standard practice sampling techniques that are frequently used in the CF literature to sample the URM to determine the accuracy of the recommendations. In addition, we cover the input characteristics that are frequently explored in CF sampling studies [26, 25], and discuss how these can be used to quantify the effectiveness of a CF model. Then, we describe how the current standard practice sampling techniques could be employed to sample the URM to predict the efficiency of a CF algorithm, highlighting the challenges and limitations.

4.2.1 Effectiveness-oriented Sampling Techniques

As argued in [26], there are still many CF algorithms for which there is limited knowledge about what characteristics of the input work well, and are compatible with the logic of the CF models. To this end, previous studies [26, 25] have tried to achieve a better understanding on how these CF algorithms perform under various recommendation tasks given inputs with different structural properties. In these settings, the impact of the density of the URM on the CF algorithm's performance is of high interest. Other aspects, such as the layout and distribution of the ratings are also taken into account for estimating the accuracy of the recommendations [26, 25].

One of the key properties that affects the performance of a CF model is the structure of the rating space. Given a dataset, the rating space is comprised of all items that can be rated by all users [26, 14]. For the rating space, there are two main properties: size and shape, where the former can be computed as $|U| \times |I|$. Some families of CF algorithms are directly impacted by the size of the rating space. For example, in neighbourhood-based models, a larger rating space size is tied to better recommendations as there are more users with similar preferences. On the other hand, if the size of the rating space is small, there are high chances that the CF algorithms will perform poorly since there are fewer "neighbours" [26, 14, 3]. The shape of the rating space is defined as the ratio between the number of users and the number of items in the URM (i.e., $|U| / |I|$). This property of the input is another factor that can influence the performance of CF models. Going back to the neighbourhood-based CF example, if the input has a high users-to-items ratio, it is very easy to find other users with similar tastes. Furthermore, an URM with many users and few items would advantage a user-oriented CF technique, while inputs with many items and few users would be favourable for item-based CF algorithms [26].

In addition to the size and shape of the rating space, the density of the ratings is another critical characteristic that impacts the performance of CF models [26, 25, 14, 3]. Traditionally, density is defined as the number of available/known ratings divided by the size of the rating space. A very common problem in the CF literature is the sparsity of the data [46, 148, 149]. This happens because users cannot consume/interact with a lot of items or there is limited information about the new users/items in the recommendation engine. The latter is also known as the cold-start problem [14]. If the density of the ratings is low, there is a reduced probability that two or more users will have similar items in common, which, in turn, can cause performance issues to the CF models [26, 25].

Given the most important input characteristics, as described previously, one can use them to determine the behaviour and performance of a CF algorithm. 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 [26, 25]. The sample of the dataset is acquired using a random sampling procedure as described in [26, 25]. In these settings, a smaller matrix (or submatrix) is drawn from the original URM containing random triplets of the form $(user, item, rating)$. These are further filtered based on the density of the submatrix, with a minimum predefined density threshold. This threshold is chosen such that the random samples contain enough ratings to allow for accurate recommendations [26, 25]. While this approach works well for estimating the effectiveness of the CF models, it does not fully capture the structural and complexity characteristics of the input, as detailed in Section 4.5.

4.2.2 Limitations of Standard Practice Sampling Approaches

In the previous section, we presented the standard practice sampling approaches that are applied in the literature to gauge the accuracy of the recommendations [26, 25]. Furthermore, we also outlined the key characteristics of the input data (i.e., URM) that may impact the performance and behaviour of the CF models. However, when we tried to apply the standard practice sampling techniques for the resource estimation problem in CF, we noted that these sampling approaches fail to capture the complexity characteristics of the base data in the samples drawn, as illustrated in Section 4.5.

In addition, it is well-known 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 which also reflects the efficiency or computational requirements of the CF model [19, 20]. To this end, another limitation that we encountered in standard practice sampling techniques is that the samples are filtered based on a density threshold, which on its own does not provide enough control over the structural and complexity properties of the samples, as demonstrated in the experimental evaluation (Section 4.5). To address these drawbacks found in the standard practice sampling approaches, we propose a simple sampling strategy (Section 4.3), which provides samples that capture the underlying characteristics of the URM and can be used for predicting both the efficiency and effectiveness of the CF algorithms at the same time.

4.3 Efficiency-oriented Sampling Approach

This section details the proposed sampling strategy, which can be reliably used to draw samples from the URM that capture the underlying complexity and structure of the data. These samples can be further used to jointly estimate the efficiency and effectiveness of the CF algorithms as described in Chapter 5 for traditional CF, and Chapter 6 for neural approaches.

For sampling the URM using Algorithm 4, the user-defined upper limit is captured through a maximum time budget T, a maximum sample size S, as well as a Coefficient of Variation (CoV). As each sample is drawn, the remaining time budget and sample size are (re)calculated for each iteration taking into account the previous time budget T' and sample size S'. The proposed approach 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 μ of the resource usage (processing time or memory overhead) values recorded while training on the current sample, and the standard deviation, σ, selected by the user. In this computation, we discard the measurement for the first iteration to avoid the 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.

Furthermore, our sampling algorithm can also use a user-defined Coefficient of Variation (CoV) as part of its input. The iterative execution on a single sample then may terminate early if the ratio c/m satisfies the said threshold. Thus we ensure that the processing times or memory overhead measured on the samples has 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 Section 4.5.

The proposed sampling approach, presented in Algorithm 4, allows us to trade off the number of samples needed for processing time and memory overhead prediction without sacrificing the accuracy of the resource cost models. Moreover, using a predefined CoV led 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 size is to analyse the *prediction error interval* in linear regression models and/or the *uncertainty* in Bayesian models described in Chapter 5. However, as this method depends on the resource prediction model used (i.e., works only with linear regression 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 a CoV which reflects the quality of the samples. Moreover, this sampling 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.

4.4 Adaptive Sampling Methodology

In the previous section, we described the rationale for using a random sampling technique to draw samples from a dataset, such that the structural and complexity characteristics of the input are preserved, and the samples can be further used to estimate the efficiency and effectiveness of a CF model on the complete dataset. We also introduced Algorithm 4, which proposes a user- and item-based sampling approach, which draws random samples from the URM, given operational constraints (e.g., time budget, maximum sample size), as well as a Coefficient of Variation (CoV) that controls the quality of the samples. In this section, we further augment the proposed sampling technique, by incorporating an adaptive component, which allows one to dynamically sample the URM, by estimating the training cost the CF models on the drawn samples, as well as how many sub-samples can be selected within the operational constraints.

4.4.1 Extended Sampling Approach

One of the drawbacks of the proposed sampling technique, described in Section 4.3, is that there is limited knowledge about how many samples we can draw within each sample size, based on their computational cost. For example, given a sub-sample within an upper

Algorithm 4: Proposed Sampling Algorithm

Input : Dataset D, Algorithm A, max sample size S, number of subsamples per sample size N, max time budget T, coefficient of variation cov.

Output: The required samples and their stats.

```
1  train_samples = []; stats = [];
2  T' = T; S' = S; D' = D;
3  do
       /* Set sample size and time budget for current iteration   */
4      T_cur = T'; S_cur = S';
       /* sample D' with the constraints                          */
5      d = sample(D', S_cur, N);
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         $\mu$ = stats.mean();
           /* compute confidence interval                         */
12         $\sigma$ = stats.standard_deviation();
13         t_end = time();
14         t += (t_end - t_start);
15     while ($\sigma/\mu >= cov$ and $t <= T\_cur$);
16     T' -= t; D' = d; S' = S_cur;
17 while ($T' > 0$);
18 return (train_samples, stats)
```

sample size of 70% and its processing time cost of 50 seconds, which includes the time to draw the sample, as well as the time needed to train the CF on the said sample, we would like to estimate the cost of acquiring a sub-sample in the next sample size (e.g., 60%). Thus, we would like to dynamically compute and execute a schedule of sample drawing/training/metric gathering subject to two conditions: (a) never exceed the overall allocated time budget, and (b) stop drawing samples at a specified sample size if their quality, measured with a predefined CoV, has met the desired threshold. To this end, we augment the proposed sampling methodology to include an adaptive component, which given an upper sample size and its computational cost, as well as the operational constraints described in Section 4.3, gauges how many sub-samples can be drawn at each lower sample size, and produces a sampling schedule, which can be further used to determine how many sub-samples will be selected based on the maximum time budget and the desired CoV.

The extended adaptive sampling algorithm that is described in the next section, takes as input the dataset D, the maximum time budget T, an upper sample size S (%), and a Coefficient of Variation cov. For a given CF algorithm, A, the adaptive sampler will firstly draw a sub-sample at the S (%) sample size, train A on it and measure the resource utilisation spent during this process. Then, this computational cost is used to compute a sampling schedule over the rest of the lower sample sizes to approximate how many sub-samples can be drawn assuming a linear scaling relationship across sample sizes. This design assumption allowed us to compute a crude estimate of the processing time of the next lower sample size and compare its cost with the overall time budget to decide if it can be fitted within the schedule or whether we would have to acquire an even smaller sample or to stop sampling if there is not enough time within the budget. Furthermore, as the sampling schedule is updated, when a sub-sample is drawn and the CF model is trained on it, the overall time budget is also updated by subtracting the processing time of the said sample from the budget. Additionally, the quality (i.e., variance) of the sub-samples is computed and compared to the CoV set by the user. The adaptive sampling algorithm stops if either the CoV has been met for the sub-samples or if there is not enough time left in the operational time budget T to draw more sub-samples.

4.4.2 Adaptive Sampling Algorithm

This section presents the proposed adaptive sampling approach as described in Algorithm 5. The algorithm begins with a boot-up phase, where we draw a sub-sample at the upper sample size S, and measure the resource consumption (i.e., processing time) while training a CF algorithm on it. Then, we defined two maps, $predicted\_times$ and $actual\_times$, where the key is the sample size and the values are a list of the processing times. In the next step, the adaptive sampling algorithm will update the $predicted\_times$ map by predicting the processing times of smaller sample sizes using linear scaling. For example, if we know that the processing time of a sample size 50% is $30s$, by applying a linear scaling transformation we can estimate the processing time of the next sample size at 40% as $\frac{40\times 30}{50}$ (or $24s$). This process is repeated across all smaller sample sizes to estimate their cost and to determine how many sub-samples at each sample size can be added in the schedule without exceeding the operational time budget. In the proposed design, we chose to utilise a linear scaling of the processing times as it offers an initial upper bound estimate of the processing time needed to draw a sample and train the CF on it. Moreover, the upper bound is guaranteed since the training times increase monotonically w.r.t. the size of the samples (i.e., a 60% sample would always require less time than a 70% sample).

The boot-up phase is followed by the scheduling phase, where we create a sampling schedule using $time\_budget$ as the total budget, and the processing times in $predicted\_times$ for the various sample sizes. The schedule is filled using a greedy approach [150] as follows: for every sample size, starting with the upper sample size S, if the $time\_budget$ permits we draw a sub-sample and update the remaining time based on the predicted processing time,

Algorithm 5: Adaptive Sampling Algorithm

Input : Dataset D, max sample size S, max time budget T, coefficient of variation cov

Output: A sampling schedule containing sub-samples, their processing time, and the corresponding quality metrics.

```
$s_0$ = sample(D, S);
$t_0$ = get_training_time($s_0$);
predicted_times = {}; actual_times = {}; stats = []; s = S;
while s >= 10 do
    predicted_time = $t_0$ / S * s;
    predicted_times.add(s, predicted_time);
    s -= 10;
end
time_budget = T - $t_0$;
threshold = predicted_times.get(10);
schedule = {}; s = S; schedule[s] = 1; t' = time_budget;
while time_budget >= threshold and c/m >= cov do
    while s >= 10 do
        if (t' - predicted_times.get(s)) >= 0 then
            t' -= predicted_times.get(s);
            schedule[s] += 1;
        end
        s -= 10;
    end
    time_budget = T - $t_0$;
    foreach (s, num) in schedule do
        i = 0;
        while i < num do
            t = get_training_time(s);
            stats.push(t);
            m = stats.mean(t);
            c = stats.confidence_interval(t);
            time_budget -= t;
            actual_times.upsert(s, t);
            i += 1;
        end
        predicted_times.put(s, average(actual_times.get(s)));
        s' = s - 10;
        while s' >= 10 do
            if not actual_times.contains_keys(s') then
                predicted_times.put(s', predicted_times.get(s) / s * s');
            end
        end
    end
    threshold = min(threshold, predicted_times.max(10));
end
return (actual_times, stats)
```

while adding the sub-sample to the $schedule$ map. These steps are repeated until we reach the smallest sample size (or default 10%). The default value of sampling in decrements of 10% was selected based on empirical evidence. We experimented with larger increments (e.g., 20%, 40%), but rate at which the processing times change w.r.t. the size of the input does not allow one to accurately compute the slope and intercept needed for the complexity equations. We also tried a finer granularity for our method, where the sampling was done in decrements of 5% and 2.5%. In this approach, the resource cost for sampling was higher as more samples were drawn based on the finer granularity, however, the changes observed in the processing times based on the sample size did not lead to accurate computations of the coefficients of complexity equations (e.g., our predictors would require an upper sample size of 40% to accurately predict processing time for the CF, but with a finer granularity we would still need the samples from 35% all the way to 5%).

Then, the adaptive algorithm advances into the schedule play-out phase. Here, we use the schedule previously computed to draw samples from the dataset, train the CF on them, measure quality metrics, and compute the final sampling schedule with the actual processing times. For each (key, value) pair in the $schedule$, and for every sub-samples within a certain sample size, we record the processing time during training and compute the corresponding sample quality metrics (i.e., c/m ratio) as described in Section 4.3. Then, the $actual\_times$ map is updated based on the current sample size and the processing times previously computed. The last part of Algorithm 5 updates the predicted processing times for smaller sample sizes, if we are still using the original pessimistic predictions computed using linear scaling. Finally, the adaptive sampler outputs the $actual\_times$ map containing sample sizes as keys and a list of the processing times for each sub-samples as values, as well as the corresponding quality metrics stored in the $stats$ array.

4.5 Performance Evaluation of Sampling Approaches

This section describes the performance evaluation of the proposed sampling approach compared to the standard practice sampling techniques. To this end, we firstly described the metrics and baselines utilised in the experiments and investigate both sampling strategies with respect to the effectiveness (i.e., quality) of the recommendations. Then, we expand this assessment to also include the efficiency of the CF algorithms based on the different sampling techniques. In addition, we examine the quality of the drawn samples, as well as the cost of sampling compared to the cost of training the CF algorithms on the chosen collections. Finally, we show that the proposed adaptive sampling methodology produces samples that can be accurately used to predict the resource consumption of the CF given a time budget, an upper sample size, and an user-defined CoV.

4.5.1 Evaluation Metrics and Baselines

To measure the processing times of the various CF algorithms on inputs sampled as described above, 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*)[1]. Moreover, the accuracy of the recommendations produced using the standard practice and proposed sampling approaches was computed using RMSE. To assess the success and suitability of the proposed sampling technique for the problem of estimating the efficiency and effectiveness of CF models, we will use the standard practice sampling algorithm described in [26, 25] as a baseline and contender.

4.5.2 Effectiveness-oriented Evaluation of Sampling Approaches

The first experiment examined 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 4.1 and 4.2 present the raw measured effectiveness of a CF algorithm on an upper sample S% that has been chosen using (i) standard practice sampling baseline (Section 4.2) [25, 26] and (ii) the proposed sampling mechanism (Section 4.3).

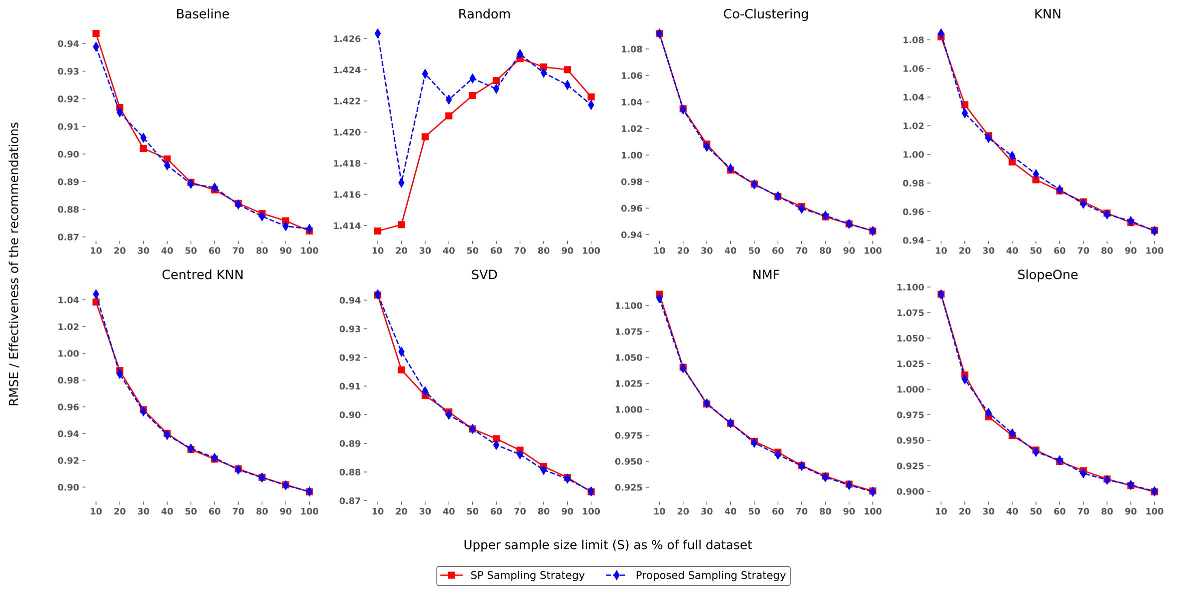

Figure 4.1: Actual CF effectiveness (i.e., quality of the produced recommendations) for the full MovieLens 100K dataset using the standard practice (SP) strategy (red curve) compared to our proposed sampling approach described in Section 4.3 (blue curve).

Interestingly, the two sampling strategies have a similar performance regarding the accuracy of the produced recommendations. For most CF models, the recorded RMSE values obtained

[1]Please see `https://docs.python.org/3/library/resource.html` for further information.

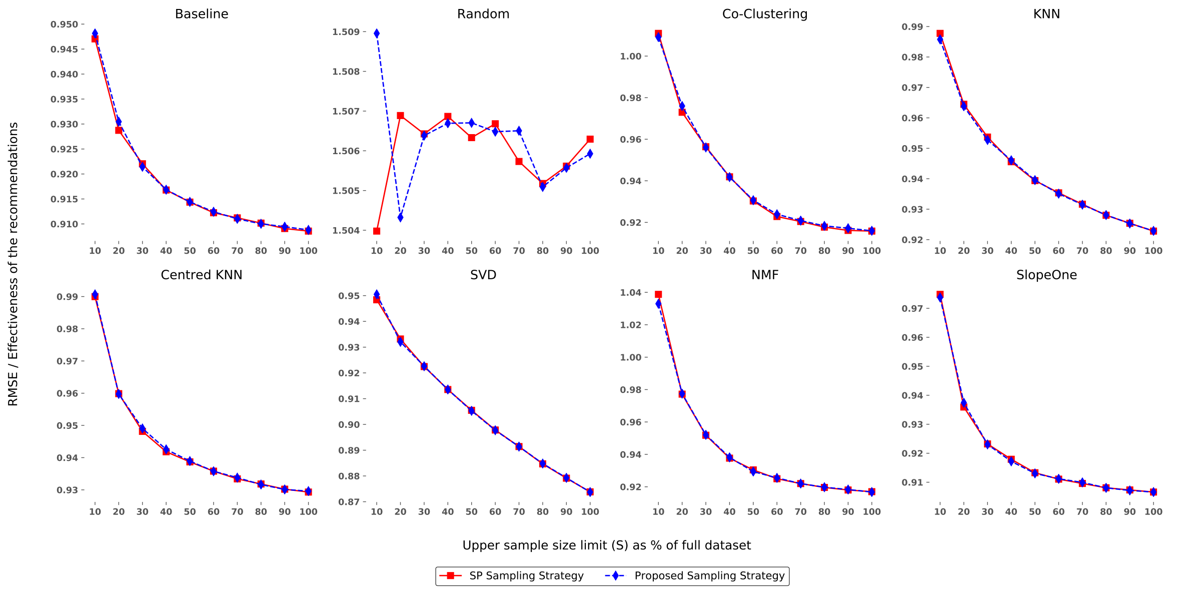

Figure 4.2: Actual CF effectiveness (i.e., quality of the produced recommendations) for the full MovieLens 1M dataset using the standard practice (SP) strategy (red curve) compared to our proposed sampling approach described in Section 4.3 (blue curve).

for the samples drawn with the standard practice sampling strategy, as well as the proposed sampling approach, follow a similar trend and curvature. For both sampling strategies, as the size of the sample increases, the CF model is trained on more data, and hence its accuracy improves, while the error (i.e., RMSE) decreases. 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.

4.5.3 Efficiency-oriented Evaluation of Sampling Approaches

The second experiment was centred on examining the performance of the two sampling strategies for drawing samples that can be used to predict the efficiency (e.g., processing time) of a CF model. This is also tied with **RQ1**, which analyses *how should we sample the base data (i.e., URM), such that the quality of the predicted efficiency/effectiveness of a CF model is not harmed*. When we assessed the two sampling approaches regarding the efficiency of the CF algorithms, more notable changes were observed.

Figures 4.3 and 4.4 show the performance of the prediction models, described in Chapter 5, 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 more accurate in predicting the efficiency

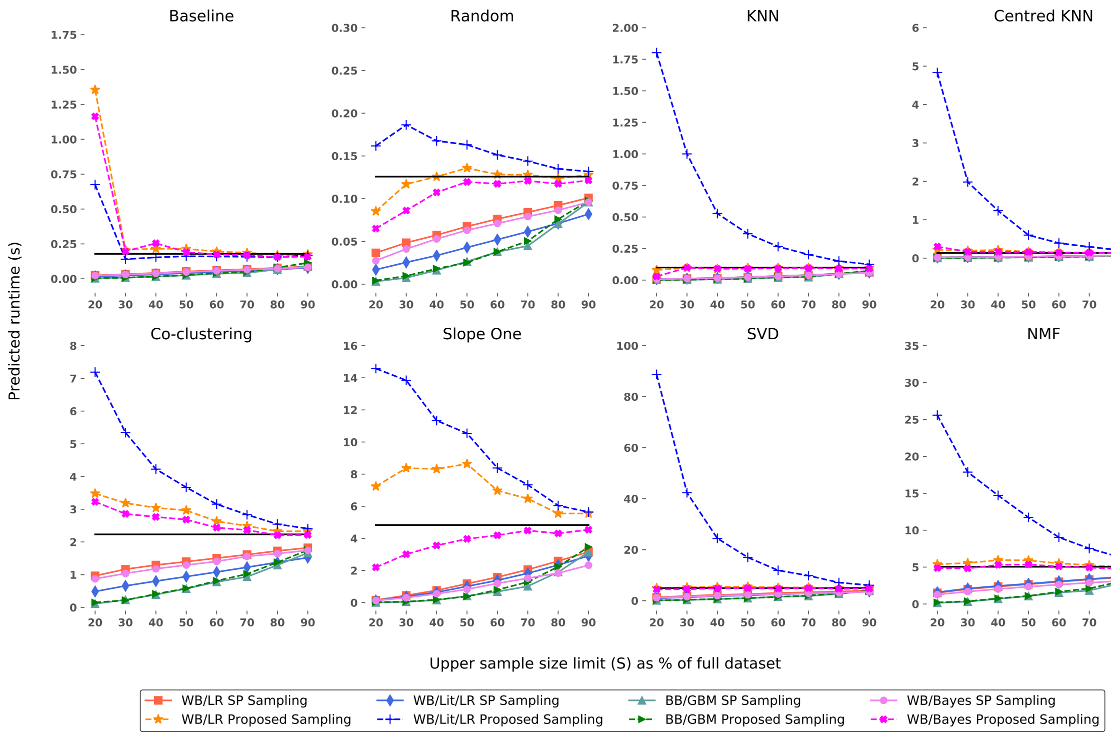

Figure 4.3: Predicted processing times for the full MovieLens 100K collection using models trained on samples drawn with the standard practice sampling approach (non-dashed curves) and the proposed sampling strategy (dashed curves).

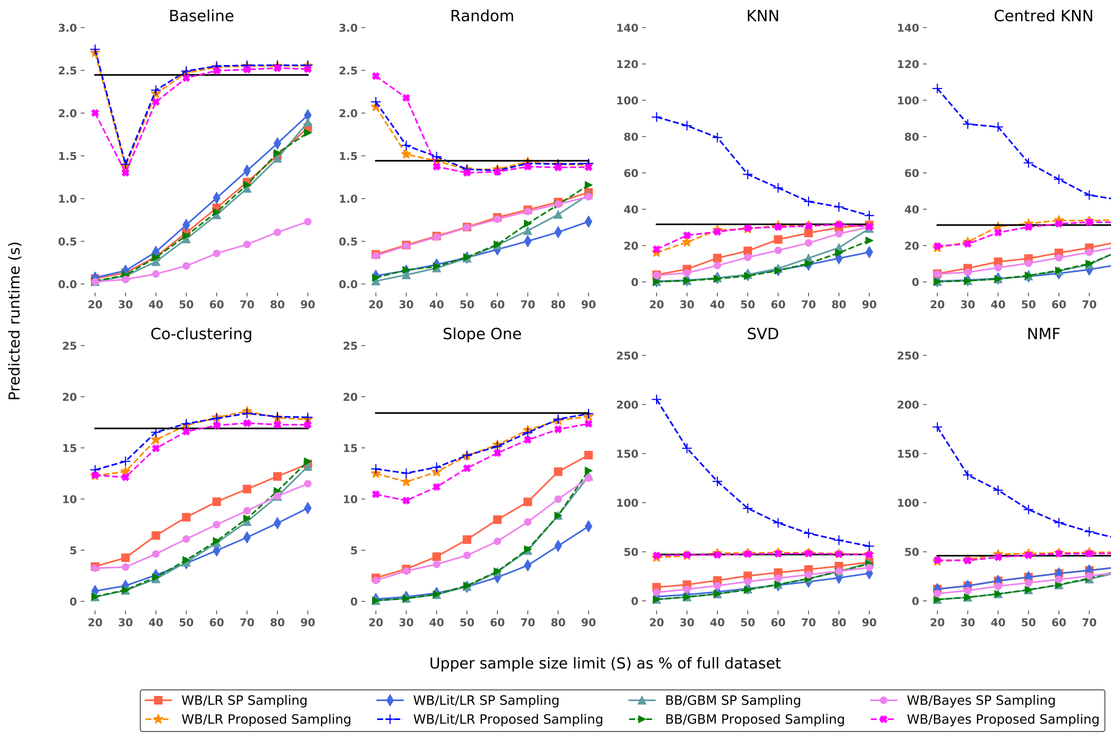

Figure 4.4: Predicted processing times for the full MovieLens 1M collection using models trained on samples drawn with the standard practice sampling approach (non-dashed curves) and the proposed sampling strategy (dashed curves)

of the CF algorithms, and hence much closer to the actual training time (black horizontal line). We note that the comparison between the sampling strategies is done on w.r.t similar prediction models (i.e., WB/LR trained on samples drawn with SP vs. WB/LR trained on the samples produced using our proposed approach). Furthermore, Figures 4.3 and 4.4 indicate that even with the proposed sampling strategy, the best state-of-the-art regressor (BB/GBM) cannot capture the rate at which the processing time grows w.r.t. sample size across the collections. This, in turn, shows that even if the samples reflect the underlying structural and complexity characteristics of the URM, as demonstrated on WB/LR trained using the proposed sampling method, the type of the resource cost predictor (i.e., BB/GBM, LR/WB, WB/Bayes, WB/Lit/LR) also impacts the accuracy at which one can estimate the processing time and memory of CF algorithms. Also, efficiency cost models based on complexity equations derived from the literature (i.e., WB/Lit/LR) would often fail to estimate the processing time of the CF, as they utilise worst-case time complexity, which does not reflect the complexity found in the implementation of the CF models.

This performance assessment of the two sampling strategies revealed that our proposed sampling algorithm (Section 4.3) works well for both efficiency and effectiveness prediction purposes. Consequently, we argue that is better to sample the user set, then the item set, and select the corresponding ratings, rather than sampling directly the ratings set as described by the standard practice sampling techniques. In the next experimental sections, we also demonstrate how the quality of the samples can be controlled, such that the drawn samples can still reflect the structural and complexity characteristics of the base data. Finally, given the limitations of the standard practice sampling approaches, the proposed sampling algorithm (Section 4.3) has been also used for the experimental evaluations in Chapter 5 and Chapter 6.

4.5.4 Sample Quality Analysis

To address **RQ2**, before determining *when to stop sampling based on the number of samples for which we obtain consistent resource usage measurements* (e.g., the processing time values are in a tight interval), we investigated the quality of the samples. To measure the latter, we propose using the CoV as described in Section 4.3, and computed as part of Algorithm 4.

In Table 4.1, we report the average CoV (where 0 corresponds to 0% and 1 corresponds to 100%) across samples in different subsets of data for a predefined CoV, which selects samples that acquired 10-15% variance in the processing time values. While most algorithms seem to achieve a low CoV 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).

Table 4.1: Average CoV 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

However, as the size of the input data increases, the CoV across samples decreases, resulting in steadier processing time measurements. This behaviour can be observed on the larger datasets (e.g., MovieLens 1M), where the results shown in Table 4.2, demonstrate that larger URMs and the corresponding samples exhibit smaller CoV for the processing times. Consequently, a low CoV indicates that the drawn samples are of high quality, and hence, fewer sub-samples are required in each sample size for the resource consumption prediction models.

Table 4.2: Average CoV across 5 samples for various values of S over the MovieLens 1M dataset.

Split %	Baseline	Centred KNN	Random	Co-clustering	KNN	KNN Baseline	NMF	Slope One	SVD
10	0.025491	0.022084	0.023630	0.008927	0.029161	0.024757	0.017445	0.018102	0.019674
20	0.014945	0.025723	0.015636	0.045791	0.049176	0.051136	0.014432	0.019948	0.015563
30	0.020766	0.021560	0.012175	0.021083	0.022854	0.041594	0.013201	0.017881	0.011518
40	0.050760	0.021382	0.009874	0.012800	0.029847	0.037930	0.012041	0.037178	0.010624
50	0.054502	0.018089	0.007475	0.008698	0.029333	0.031964	0.007347	0.014521	0.007209
60	0.042309	0.014702	0.006587	0.006039	0.039048	0.017752	0.005994	0.012075	0.006096
70	0.048106	0.015246	0.006263	0.011951	0.036194	0.015900	0.004816	0.010409	0.005268
80	0.054952	0.035928	0.004961	0.008230	0.031214	0.024864	0.004577	0.029441	0.004250
90	0.053097	0.015678	0.015987	0.004480	0.026020	0.029724	0.007800	0.006731	0.008820
100	0.052503	0.012373	0.012846	0.027127	0.032868	0.031828	0.008704	0.031676	0.006164

A further investigation into **RQ2**, highlighting the stopping criteria for the proposed sampling methodology is described in Section 4.5.6. However, prior to this, we present an assessment between the cost of sampling the input and the cost of training a CF model on the same collection, illustrated in Section 4.5.5.

4.5.5 Assessment of Sampling vs. Training Cost for CF

When assessing the overall performance of our framework for predicting the resource consumption of the CF models, we also need to consider the cost of sampling the dataset, as well as training the CF algorithms on the samples. 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 sampling strategy that is cheaper than training the CF models on the full collections, and which also reflects the complexity characteristics of the input data.

To this end, and in line with **RQ1** and **RQ2**, Figures 4.5, 4.6, and 4.7 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 in cases where a set of samples is drawn for a bigger sample size (i.e., 70-80% and above) it takes more time to train the CF on the said samples. This is due to the fact that the time reported for these sample sizes also includes the time necessary to draw the smaller samples (i.e., the processing time of an 80% sample would also include the times for the samples between 70% and 10%). Hence, this is why in Figures 4.5, 4.6, and 4.7 the processing time reported in the composite blue-red bars is sometimes above the actual training time on the full collection depicted with a black horizontal line.

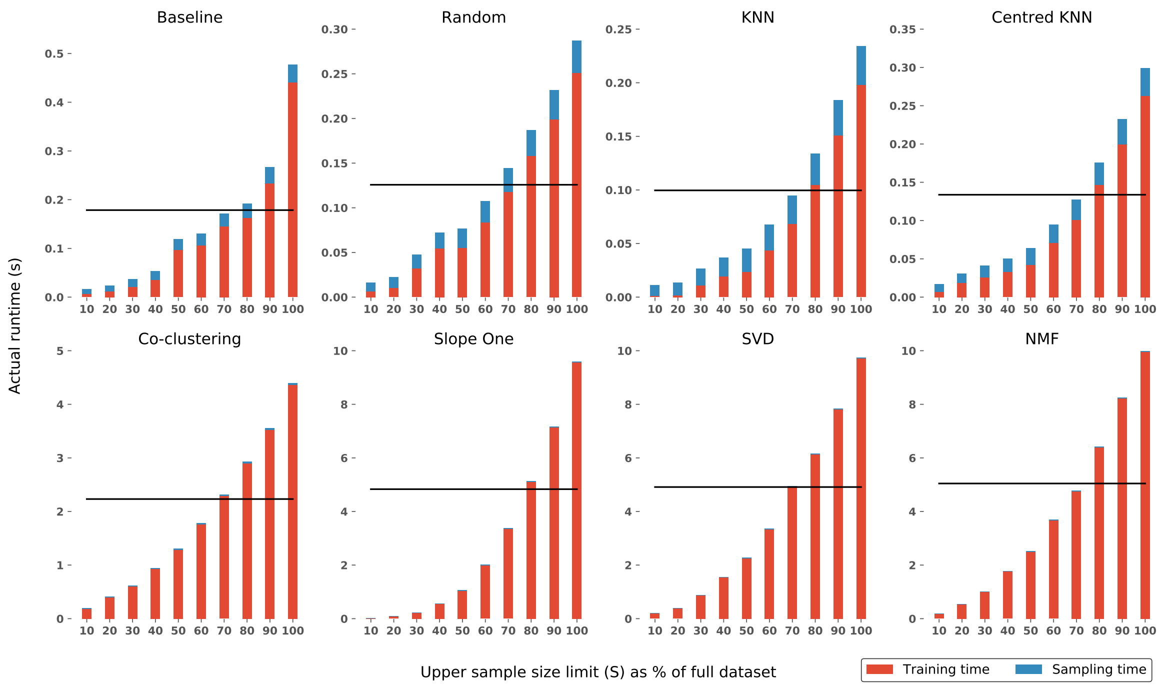

Figure 4.5: Sampling (blue bars) and training (red bars) time cost for MovieLens 100K w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line).

We note that for CF algorithms with more expensive training costs (e.g., KNN-based CF), the processing time exhibits a steeper increase across samples. Also, the bigger the dataset/URM is, the cost of training the CF algorithms is far greater than the one for sampling the input,

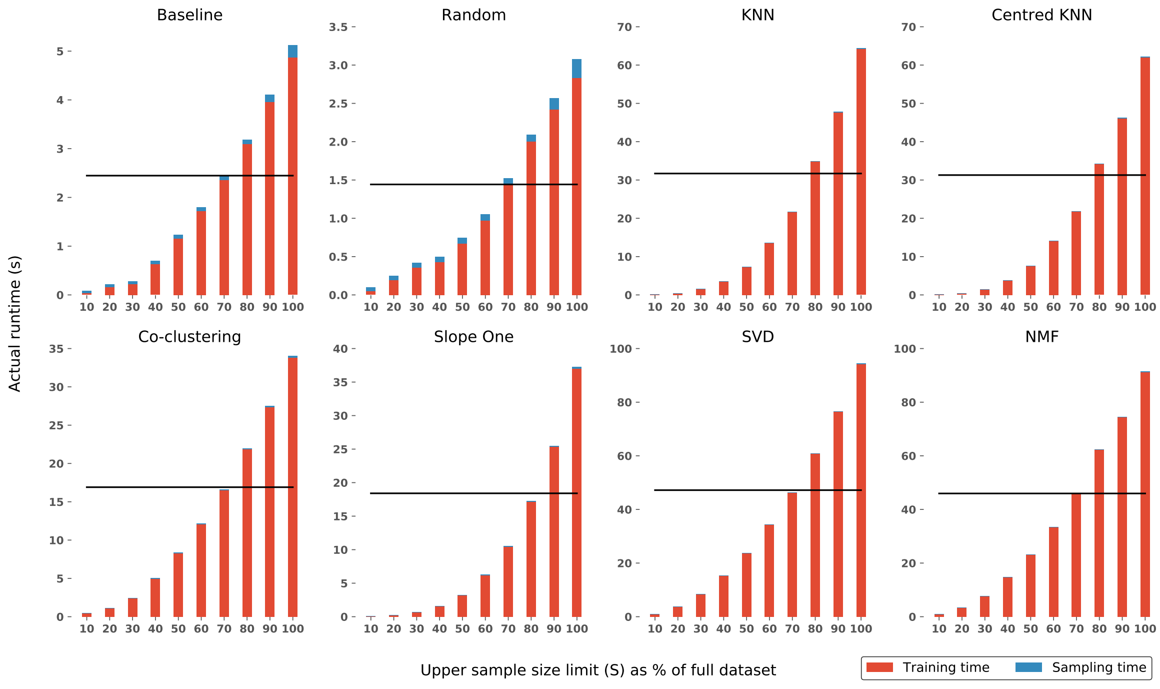

Figure 4.6: Sampling (blue bars) and training (red bars) time cost for MovieLens 1M w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line).

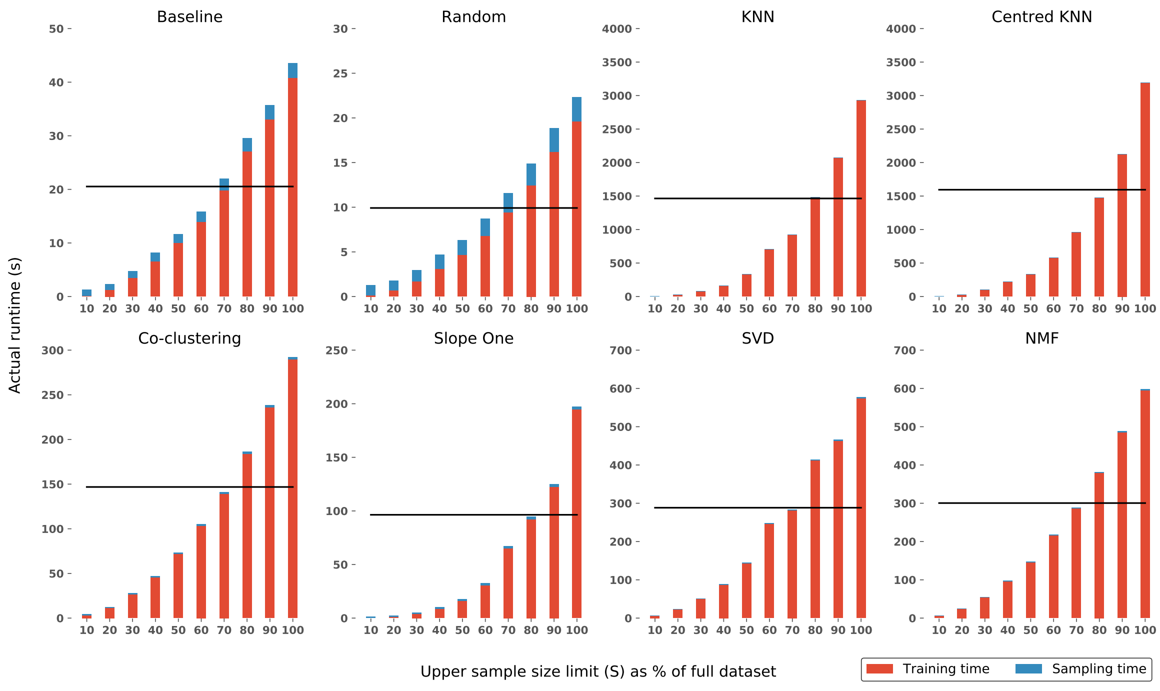

Figure 4.7: GoodBooks 10K

Figure 4.8: Sampling (blue bars) and training (red bars) time cost for GoodBooks 10K w.r.t. upper sample size S% compared to the processing time for the entire dataset (black line).

as depicted in Figure 4.5 for most CF algorithms. 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.

4.5.6 Evaluation of the Adaptive Sampling Methodology

In this section, we illustrate an experimental evaluation of the adaptive sampling methodology, presented in Section 4.4, through an ablation study [81].

Table 4.3: Average NRMSE of the predicted processing time for various values of S over MovieLens 1M for a CoV of 0.1 and a time budget T of 50 seconds.

S%	**Baseline**	**Random**	**KNN**	**C. KNN**	**KNN B.**	**NMF**	**SVD**	**Slope One**	**Co-clustering**
20	0.054	0.373	0.5	0.411	0.372	0.17	0.091	0.464	0.355
40	0.219	0.209	0.248	0.319	0.26	0.103	0.038	0.439	0.25
60	0.138	0.148	0.191	0.218	0.134	0.068	0.029	0.376	0.161
80	0.102	0.096	0.137	0.153	0.098	0.051	0.024	0.297	0.115

Table 4.4: Average NRMSE of the predicted processing time for various values of CoV over MovieLens 1M for an upper sample size S of 40% and a time budget T of 50 seconds.

CoV	**Baseline**	**Random**	**KNN**	**C. KNN**	**KNN B.**	**NMF**	**SVD**	**Slope One**	**Co-clustering**
0.01	0.218	0.192	0.273	0.318	0.249	0.09	0.035	0.446	0.258
0.05	0.216	0.187	0.273	0.347	0.265	0.095	0.041	0.444	0.242
0.1	0.219	0.171	0.301	0.334	0.255	0.091	0.047	0.449	0.24
0.2	0.217	0.289	0.296	0.329	0.292	0.089	0.046	0.432	0.245

Table 4.5: Average NRMSE of the predicted processing time for various values of T over MovieLens 1M for an upper sample size S of 40% and a CoV of 0.1.

T	**Baseline**	**Random**	**KNN**	**C. KNN**	**KNN B.**	**NMF**	**SVD**	**Slope One**	**Co-clustering**
15	0.22	0.334	0.296	0.324	0.253	0.152	0.089	0.454	0.264
25	0.218	0.192	0.284	0.311	0.275	0.102	0.022	0.443	0.269
50	0.215	0.192	0.289	0.314	0.248	0.104	0.065	0.434	0.249
100	0.219	0.127	0.27	0.334	0.243	0.093	0.036	0.451	0.237

Thus, we fix each of the input parameters of Algorithm 5, apart from one, for which we set different values, and then we use the outputted sampling schedule and processing time measurements to predict the resource consumption of various CF models on the medium sized dataset (MovieLens 1M) using our best predictor (WB/LR). Tables 4.3, 4.4, and 4.5 present the ablation study, where we report the normalised RMSE of the predicted processing time, for various values of the upper sample size S, Coefficient of Variation CoV, and time budget T. Our findings indicate that a larger sample size lowers the prediction error, while a

5-10% variance across measurements (i.e., CoV=0.05 or 0.1) offers a good trade-off between the quality of the predictions and the number of samples drawn. Also, a greater time budget would allow for more samples to be drawn, and hence would improve the estimation of the resource consumption; however, setting a very large time budget does not always lead to better predictions. Lastly, our proposed adaptive sampler would terminate sampling the URM earlier if the CoV threshold is achieved, regardless of the used time budget.

4.6 Chapter Summary

This chapter outlined and addressed the problem of how to sample the URM, such that the drawn samples reflect the complexity and structural properties of the input and can be further used to predict the resource consumption and effectiveness of various CF. Firstly, we examined the standard practice sampling strategy that is currently used in the literature [25, 26] to predict the performance of CF models, noting how it cannot be successfully applied to estimate the efficiency of CF, as it does not capture the complexity of the base data. Secondly, Section 4.3 addressed the limitations of the sampling methodology described in Section 4.2 by proposing a sampling approach tailored for efficiency and effectiveness prediction, while maintaining both the complexity and structural characteristics of the URM. Furthermore, Section 4.4 described an adaptive sampling approach that can be employed to dynamically sample the URM given the operational constraints (e.g., time budget, maximum sample size), and the desired quality of the samples based on a user-defined CoV [122]. Lastly, Section 4.5 discusses the performance differences between the standard practice sampling strategy and the proposed one. Here, we demonstrated how both approaches can be successfully used to draw samples for effectiveness prediction; however, only the proposed sampling strategy can be accurately used to predict the resource consumption of the CF outperforming the standard practice baseline.

Chapter 5

Efficiency Cost Models for Traditional CF

This chapter describes the efficiency cost methodologies for predicting resource consumption during training for a number of well-known traditional CF models. First, we provide an overview of the challenges associated with efficiency estimation in CF, and how this topic has been broadly overlooked by the literature. Then, we formalise the research problem centred on how we can gauge how many resources will be needed to train a traditional CF, based on the size of the input. Secondly, Section 5.2 provides an in-depth analysis of the time and space computational complexity of the chosen CF algorithms. Here, we present the complexity equations provided by the relevant literature, and then, by examining the implementation of the CF models, we propose alternative time and space equations that reflect the resource cost of the CF in a more accurate manner. Thirdly, we illustrate the evaluation metrics and baselines used, as well as how we employed and configured the proposed sampling strategy to draw samples from the URM. Then, we present the experimental evaluation of our efficiency cost methodologies, investigating the performance of the White- and Black- box approaches towards estimating the hardware resources utilised during training by traditional CF algorithms. Finally, we provide a summary of the chapter in Section 5.5.

5.1 Problem Overview

As highlighted in Section 2.3, the efficiency of the CF is often omitted in evaluation studies. Beel et al. [93] highlighted that one of the key problems in the CF field is that many papers do not report the runtimes of the proposed algorithms and often the computational complexity is not discussed. These aspects play a critical role for the content providers, who deal with large volumes of data and need to produce recommendations for their users in near-real-time.

Moreover, the diversity of the CF approaches and their corresponding implementations make it difficult to compare across them given a predefined recommendation task (e.g., rating prediction problem). Baselines are often used to alleviate this problem, but how these are selected is not always trivial due to a large design space with many variables (e.g., the internal representation of input data, sets of computations, algorithms and data structures, datasets).

Consequently, we formalise the problem of efficiency prediction for traditional CF algorithms, and we propose several solutions based on *cost models* that report on the algorithmic complexity and estimate the CF's resource consumption during training (e.g., runtime, memory footprint) with respect to the size and the characteristics (e.g., number of users/items/ratings, sparsity) of the input data. Furthermore, in this chapter we answer **RQ3**, **RQ4**, and **RQ5** outlined in Section 1.2.

5.2 Complexity Analysis

Traditionally, the performance of an algorithm is captured through asymptotic worst-case complexity equations using big-O notation [91]. 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 [66, 151, 152, 153, 154, 155, 156].

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 [157]. This is also why, as highlighted in the experimental evaluation (Section 5.4), predictors based on literature complexity equations cannot accurately estimate the time and memory required by the CF during training, as the literature reports the worst case/upper bound of the resource consumption. We thus formulate and propose time and space complexity equations based on the actual implementation of said CF models. In the following sections, we list the algorithmic complexities based on *(a) literature* [66, 151, 152, 153, 154, 155, 156] and *(b) implementation*. For the latter, we used Surprise's [37] documentation and implementation, and derived the expected time and space complexity of CF models captured through big-O notation and the following characteristics of the input (URM): 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, $\boldsymbol{\delta}$, computed in Equation 5.1.

$$\delta = \frac{\rho}{mn} \tag{5.1}$$

For the purpose of our approach and proposed methodology, the number f of latent factors as well as the number e of epochs, where applicable, are considered constants set to the predefined/recommended values by [37]. Furthermore, we note that the complexity equations derived and presented in the next sections are based on the implementations found in Surprise [37], and alternative implementations of the CF may lead to different equations. However, as Surprise is one of the most popular recommendation engines, we selected it to demonstrate the feasibility of our methodology for estimating resource consumption of CF algorithms. If one would utilise alternative implementations of CF, our methodology would not change, however, the expected/average complexity equations may need to be updated to accurately estimate time and memory of the CF.

5.2.1 Processing Time Complexity Equations

The **baseline** CF is based on the Alternating Least Squares (ALS) algorithm, the naive solver[1] version, which has a time complexity of $O(mnf)$, where f is the number of latent factors. If we further fix f to a default/recommended number, the complexity can be further abstracted to $\boldsymbol{O(mn)}$ [66]. 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 $\boldsymbol{O(m^2 + n^2)}$.

The **random** algorithm, based on Maximum Likelihood Estimation (MLE), predicts the missing ratings over a normal distribution, computed in maximum $\boldsymbol{O(mn)}$ steps [151]. 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 $\boldsymbol{O(\rho)}$.

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 $\boldsymbol{O(m^2n^2)}$ [152, 158]. 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 the URMs from all datasets, we

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

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 [75], 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., $\boldsymbol{O(\frac{\rho^2}{x} + x^2 + \rho)}$.

The **NMF** model is based on the SGD algorithm, which achieves a time computational complexity of $O(em\rho)$, where e is the number of epochs (iterations)[153]. If we fix the number of epochs, the complexity can be reduced to $\boldsymbol{O(m\rho)}$. In the Surprise framework [37], 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, e, and factors, f).

SVD, a popular CF-based approach, has been intensively used to produce recommendations on explicit datasets. Over time, multiple variations of SVD have developed [154], leading to a significant number of implementations. However, most of them converge to a time complexity of $\boldsymbol{O(mn^2)}$, even though other studies, such as [2], claim that the overall complexity of SVD is close to $O(n^2m + m^2n)$. The SVD's implementation found in Surprise [37] 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 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 [155]. 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 $\boldsymbol{O(\frac{\rho^2}{m} + n^2 + \rho)}$.

The **Co-clustering** CF with a fixed number of user-item clusters converges towards a computation complexity of $\boldsymbol{O(mn)}$ [156]. 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 $O(m+n+\rho)$ time.

Finally, we summarise the time complexity of the CF presented in literature and derived from the implementation in Table 5.1. It should be noted that for KNNs, x can be either m for user-based KNN, or n, for item-based KNN, respectively.

Table 5.1: Time complexity equations for traditional CF derived from literature and implementation.

	Literature Complexity	**Implementation Complexity**
Baseline	$O(mn)$ [66]	$O(m^2+n^2)$
Random	$O(mn)$ [151]	$O(\rho)$
KNN	$O(m^2n^2)$ [152, 158]	$O(\frac{\rho^2}{x}+x^2)$
Centred KNN	$O(m^2n^2)$ [152, 158]	$O(\frac{\rho^2}{x}+x^2+\rho)$
KNN Baseline	$O(m^2n^2)$ [152, 158]	$O(\frac{\rho^2}{x}+x^2+\rho)$
Co-clustering	$O(mn)$ [156]	$O(m+n+\rho)$
Slope One	$O(mn^2)$ [155]	$O(\frac{\rho^2}{m}+n^2+\rho)$
SVD	$O(mn^2)$ [154]	$O(\rho)$
NMF	$O(m\rho)$ [153]	$O(\rho+m+n)$

5.2.2 Space (Memory) Complexity Equations

For memory usage, apart from the size of the URM, which in all cases takes $O(\rho)$ of memory, the **baseline** algorithm stores the users', items' resp., baseline in an array of size $O(m)$, $O(n)$ resp.; since, both baselines are used by this CF, the overall *space complexity* is $O(m+n)$.

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 $O(\rho)$.

To compute the distances and similarities between pairs of users, items resp., the **KNN-based** CF allocates additional memory space for matrices of size $O(m^2)$, $O(n^2)$ resp. Therefore, the *memory usage* for KNN, Centred KNN, and KNN baseline models during training is $O(m^2)$ or $O(n^2)$ depending on whether distances/similarities are computed across users or across items.

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

For memory requirements, during training, **SVD** follows similar storage requirements as NMF, having a *space complexity* of $O(mf + nf)$.

Slope One CF requires memory space for two additional matrices to compute the frequency and deviation between pairs of items across the dataset during its training process. Consequently, the expected *memory usage* is $O(n^2)$.

During training, the **Co-clustering** CF computes the users and items mean across the entire collection and stores them in arrays of size $O(m)$, $O(n)$ resp. Then, users and items clusters and co-clusters are built, which require another $O(m)$, $O(n)$, and $O(mn)$ resp. of memory. This brings the overall *space complexity* to $O(m + n + mn)$.

Similarly to the time complexity equations, Table 5.2 summarises the space complexity equations derived from the implementation of traditional CF. For KNNs, x should be replaced with either the number of users, m, or items, n, depending on which option is used as a similarity metric. Furthermore, for the selected CF models, where a dash line is depicted in the table, the literature does not report the corresponding complexity of the algorithms.

Table 5.2: Space complexity equations for traditional CF derived from literature and implementation.

	Literature Complexity	**Implementation Complexity**
Baseline	–	$O(m+n)$
Random	–	$O(\rho)$
KNN	–	$O(x^2)$
Centred KNN	–	$O(x^2)$
KNN Baseline	–	$O(x^2)$
Co-clustering	–	$O(m+n+mn)$
Slope One	–	$O(n^2)$
SVD	–	$O(mf+nf)$
NMF	–	$O(mf+nf)$

5.3 Experimental Settings for Traditional CF

This section firstly presents the evaluation metrics used to assess the quality of the White- and Black-box regressors used to estimate the efficiency and resource consumption of traditional CF during their training phase. Then, we explain how we acquired the processing time and memory measurements of the CF models, as well as how the proposed sampling strategy, described in Chapter 4, was used to draw samples from the URM, which were further used to train the CF models and predict their efficiency and effectiveness.

5.3.1 Evaluation Metrics and Baselines

As part of the experimental evaluation, we assessed the efficiency of CF by examining how the processing time and memory overhead vary during training for inputs of various sizes. To this end, we first logged the resource consumption exhibited by each CF representative. The processing time was captured using the *getrusage* method from Python's *resource* module as explained in Section 4.5.1. To retrieve the memory usage of the CF, we recorded the information from the proc filesystem via the "/proc/[pid]/status" file[2], 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*[3] module from Python, and ensured that the results reported by the profiler match the ones recorded using the proc filesystem. Furthermore, we also measured the effectiveness of the CF by recording the RMSE of the predicted rating values.

To evaluate the proposed efficiency cost methodology (i.e., White-Box approach) for traditional CF, we used the off-the-shelf state-of-the-art regressors (i.e., Black-Box method) as the main baseline. Moreover, we also compare the proposed time complexity equations, derived from the implementations found in [37], with the baseline equations provided by the relevant literature [66, 151, 152, 153, 154, 155, 156].

5.3.2 Sampling Strategy

For sampling the URM and acquiring the resource consumption of CF, 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 4 (to be discussed in Chapter 4). For each draw, we also used a fixed random seed $\text{\textit{ȝ}}$ from a predefined set of seeds to ensure reproducibility. A corollary of our sampling strategy is that each smaller sample is a sub-sample of a large one (e.g., a 10% sample is a sub-sample of a 20% sample); this approach allowed us to measure how the training time and memory usage change when the input size gradually changes.

For the scope of our experimental evaluation, in Algorithm 4, we set the confidence to 99%, and use a 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 a Coefficient

[2]More information about the proc filesystem can be found at `https://man7.org/linux/man-pages/man5/proc.5.html`.

[3]Memory-profiler is available at `https://pypi.org/project/memory-profiler/`.

of Variation (CoV) of 0.1. Last, we set the overall time quota T to 2000, 500, 100 seconds per sample for GB 10K, ML 1M, and ML 100K, respectively. The time quota values were selected based on the available resources (both temporal and hardware), while the CoV was chosen based on the preliminary results outlined in Section 4.5 . In addition, the results presented in Section 5.4 indicate that this setup offered a good trade-off between the number of samples needed and their quality.

5.4 Experimental Evaluation

In the following experimental evaluation, we firstly present a comparative assessment between the proposed White-Box approach and the Black-Box contender for the problem of processing time prediction for CF. Then, we investigate the cost of training the White- and Black-Box methods, highlighting that the most accurate approach (WB/LR) is also the fastest, and hence, the cheapest with respect to hardware usage. Section 5.4.3 covers the memory overhead prediction models for CF, while the last part, Section 5.4.4, presents how our framework can also estimate the effectiveness of the recommendations.

5.4.1 Processing Time Prediction Models for CF

As part of **RQ3**, we built the two main efficiency cost estimation methodologies, namely the Black- and White-Box approaches, further investigated in **RQ4**, and **RQ5**, respectively. To highlight the advantages and drawbacks of each approach, we first examine them in the CF's processing time prediction task.

Figures 5.1, 5.2, and 5.3 depict the estimated training time for the complete dataset, 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 (BB/GBM)). 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 were 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 Section 3.2.

Our results indicate that WB/LR, using simple linear regression, outperforms the much more complex best performing state-of-the-art regressor (BB/GBM). In most cases, a 30%-40% upper sample size limit seems enough to allow WB/LR to achieve highly accurate prediction.

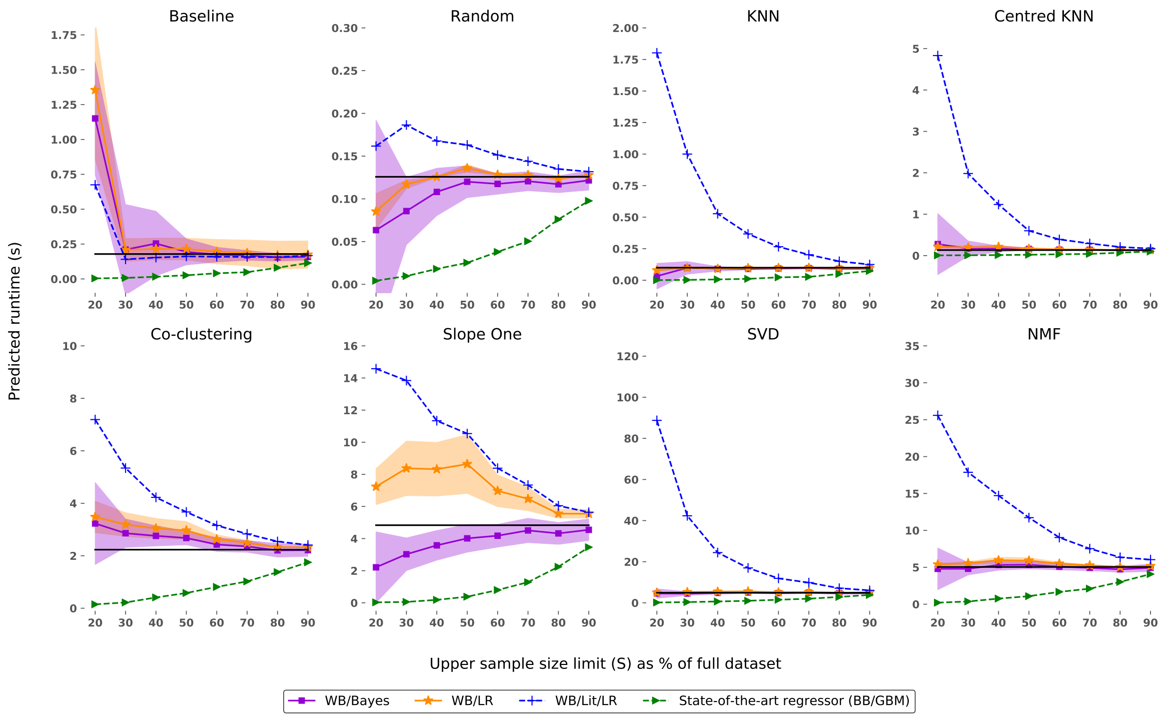

Figure 5.1: Predicted processing times for MovieLens 100K for various values of S using our approaches (WB/LR and WB/Bayes) and the two baselines (WB/Lit/LR, and state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

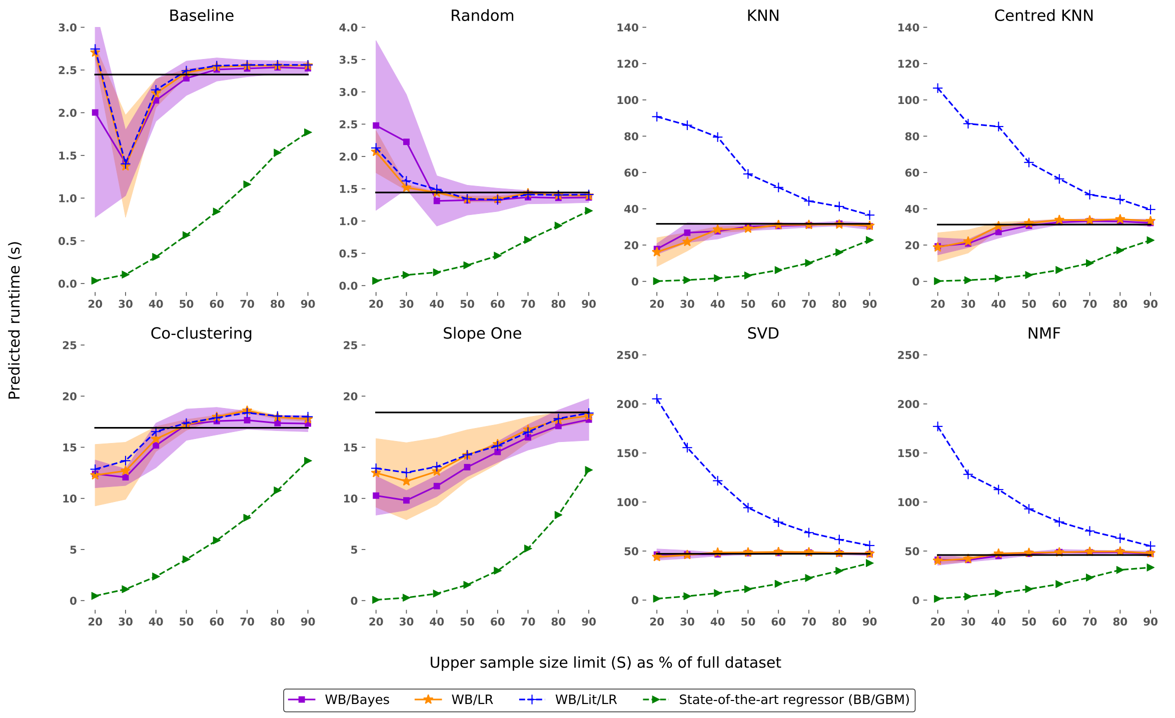

Figure 5.2: Predicted processing times for MovieLens 1M for various values of S using our approaches (WB/LR and WB/Bayes) and the two baselines (WB/Lit/LR, and state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

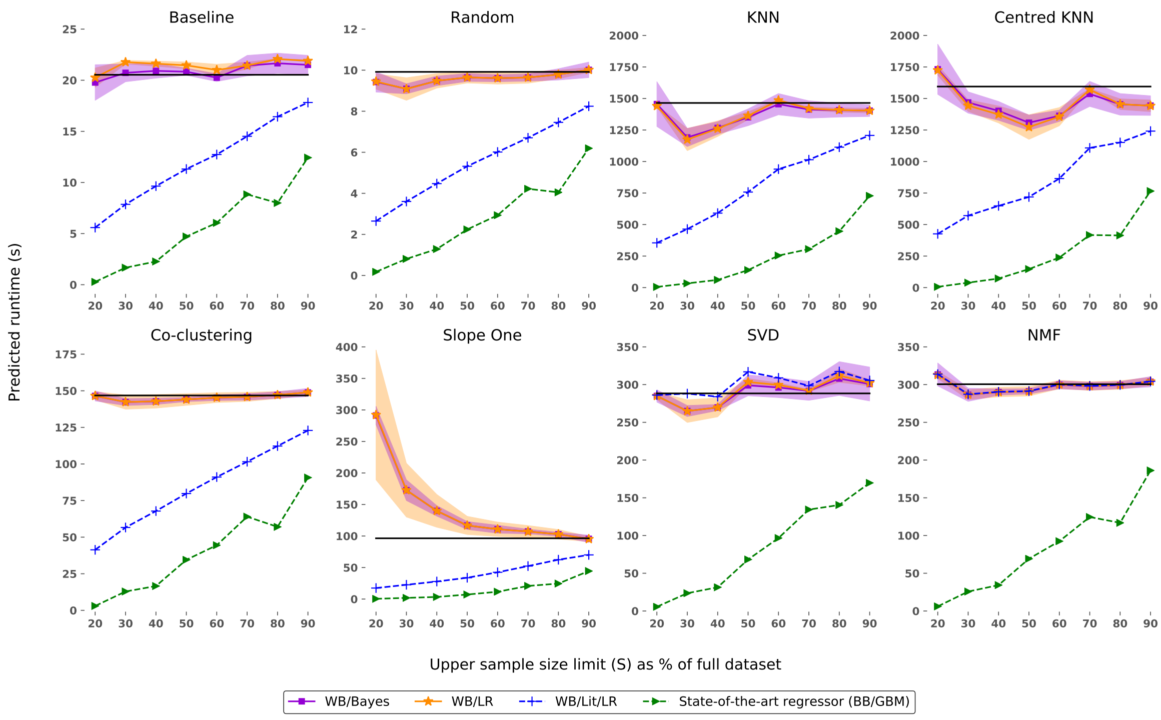

Figure 5.3: Predicted processing times for GoodBooks 10K for various values of S using our approaches (WB/LR and WB/Bayes) and the two baselines (WB/Lit/LR, and state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

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 in the next section. In addition, our contender, BB/GBM, is underestimating the resource consumption for both time and memory for all traditional CF algorithms. We believe this is due to the features utilised by this predictor, as the structural properties of the URM used on their own do not suffice to capture the complexity of the base data. Furthermore, the linear regression models trained on the complexity equations reported in the literature (i.e., WB/Lit/LR) are very far from from the ground truth data (i.e., black horizontal line) in most cases. This is due to the fact that in literature, only the worst case/upper bound of complexity is reported for a given CF, while in our use-case, we require the expected/average time and space complexity equations to be able to accurately estimate resource consumption.

5.4.2 Comparative Assessment of the Cost of Prediction Models

In Section 4.5.5, 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 5.3 presents the average time taken by each prediction model across the three datasets. As we can see, the fastest 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).

Table 5.3: Mean and standard deviation of processing time (i.e., training and prediction time) for the prediction models across the three datasets (MovieLens 100K, MovieLens 1M, and GoodBooks 10K).

	Mean Runtime (s)				**Standard Deviation**			
	WB/LR	*WB/Bayes*	*WB/Lit/LR*	*BB/GBM*	*WB/LR*	*WB/Bayesian*	*WB/Lit/LR*	*BB/GBM*
Baseline	0.000846	12.97	0.000821	0.56	0.000076	0.63	0.000008	0.19
Random	0.000811	12.49	0.000831	0.49	0.000014	0.63	0.000017	0.07
KNN	0.000828	12.78	0.000821	0.47	0.000009	0.37	0.000007	0.01
Centred KNN	0.000835	12.34	0.000834	0.47	0.000019	0.55	0.000019	0.02
KNN Baseline	0.000806	13.30	0.000805	0.48	0.000012	0.64	0.000020	0.01
Co-clustering	0.000793	12.84	0.000801	0.46	0.000014	0.78	0.000012	0.02
Slope One	0.000829	13.28	0.001018	0.50	0.000013	0.62	0.000140	0.07
SVD	0.000811	13.43	0.000833	0.52	0.000015	0.61	0.000029	0.09
NMF	0.000840	13.33	0.000817	0.50	0.000040	0.50	0.000009	0.02
Geometric Mean	***0.000821***	**12.97**	**0.000840**	**0.49**	***0.000018***	**0.58**	**0.000019**	**0.04**

As presented in Table 5.4, 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 ($). As the models have been trained on an Intel-Xeon E7-4870[4], which has an average power consumption of 130 kWh, we computed the overall power reported in Table 5.4 as detailed below:

$$power = \frac{time \times 130}{60 \times 60} \tag{5.2}$$

where the time represents the total training time of the CF on a specific sample size (e.g., 20%, 40%, 60%). Then, to translate the utilised power into a monetary value, we multiplied the power with the average cost of the electricity in the US in 2021[5] (i.e., 0.12$ per kWh).

$$cost = power \times 0.12 \tag{5.3}$$

Furthermore, the power and monetary costs are compared to the normalised RMSE values

[4]Specifications of Intel-Xeon E7-4870 are available at `https://ark.intel.com/content/www/us/en/ark/products/53579/intel-xeon-processor-e74870-30m-cache-2-40-ghz-6-40-gts-intel-qpi.html`.

[5]Electricity prices are avialble at `https://www.statista.com/statistics/183700/us-average-retail-electricity-price-since-1990/`.

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 (BB/GBM) baselines. This study was conducted on the most computationally intensive dataset (i.e., GoodBooks 10K), as we believe it is a good example to illustrate that a bigger sample, which is also more expensive, does not necessarily improve the accuracy of the predictions.

Table 5.4: 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*	*BB/GBM*
	20	3.5	0.13	0.02	0.16	13.59	12.35	19.98
Baseline	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
	20	1.86	0.07	0.01	0.05	0.02	5.55	9.57
Random	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
	20	64.45	2.33	**0.28**	0.39	**0.05**	840.67	1453.77
KNN	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
	20	76.39	2.76	**0.33**	**10.5**	11.75	855.98	1582.24
Centred KNN	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
	20	83.38	3.01	0.36	228.02	247.41	569.47	1330.45
KNN Baseline	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
	20	29.7	1.07	**0.13**	0.19	**0.01**	80.49	140.93
Co-clustering	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
	20	3.62	0.13	0.02	466.88	393.51	57.44	95.66
Slope One	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
	20	56.98	2.06	**0.25**	0.32	**0.04**	0.25	277.31
SVD	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
	20	62.34	2.25	0.27	10.78	0.55	10.49	288.54
NMF	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

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. In addition, KNN exhibits a similar behaviour where the best prediction accuracy (i.e., NRMSE) is achieved at a 20% sample size for an estimated cost of 0.28$. Nevertheless, for other CF models, such as SlopeOne, a larger and more expensive (0.34$) sample size of up to 60% is required to predict the processing time.

Based on the results presented in Tables 5.3 and 5.4, 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. Finally, as showcased in Table 5.4, the most accurate resource consumption predictors are the proposed WB/LR and WB/Bayes models. However, if we analyse the processing times of WB/LR and WB/Bayes in Table 5.3, we notice that the former model is several orders of magnitude faster than the latter. Consequently, besides knowing the training cost of the CF, it is important to also select a cheap and accurate predictor for estimating the resource consumption, such as the proposed WB/LR model.

5.4.3 Memory Overhead Prediction Models for CF

The proposed efficiency cost methodology can also predict the memory usage required for training the CF algorithms. To achieve this, we use the same approach as the one for predicting the processing time. We first sample the dataset (if the samples are not already available), then train the CF on the samples and measure the additional memory used during training. Lastly, we use the proposed prediction models captured in the White-Box method to learn the hidden coefficients from the space complexity equations, outlined in Section 5.2.2, as well as our Black-Box baseline (BB/GBM) to estimate the memory usage by using characteristics of the URM.

Figures 5.4, 5.5, and 5.6 illustrate the accuracy of the predictions for memory usage across the three datasets. These figures are similar in design to the processing time figures (i.e., 5.1, 5.2, and 5.3); 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 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 (BB/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. If we compare the error interval and uncertainty depicted in the time (Section 5.4.1) and memory estimation figures, we will notice that the error/uncertainty areas are smaller for the RAM predictions. This is due to the fact that there is less variance in recorded memory values compared to the variance measured in the processing times, and this leads to more accurate predictions, as well as smaller error/uncertainty margins. Finally, WB/Lit/LR is not shown on the memory consumption figures as, alas, the literature around the CF models considered here provides no space complexity analysis.

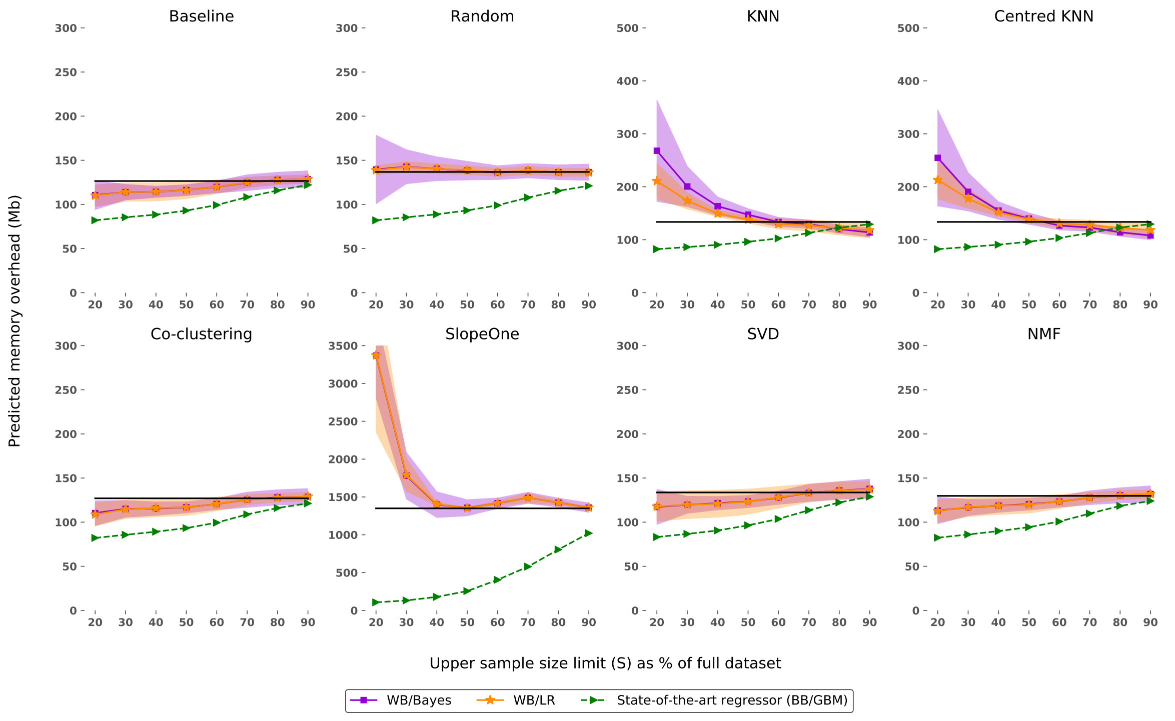

Figure 5.4: Predicted memory overhead for MovieLens 100K for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead for the entire dataset.

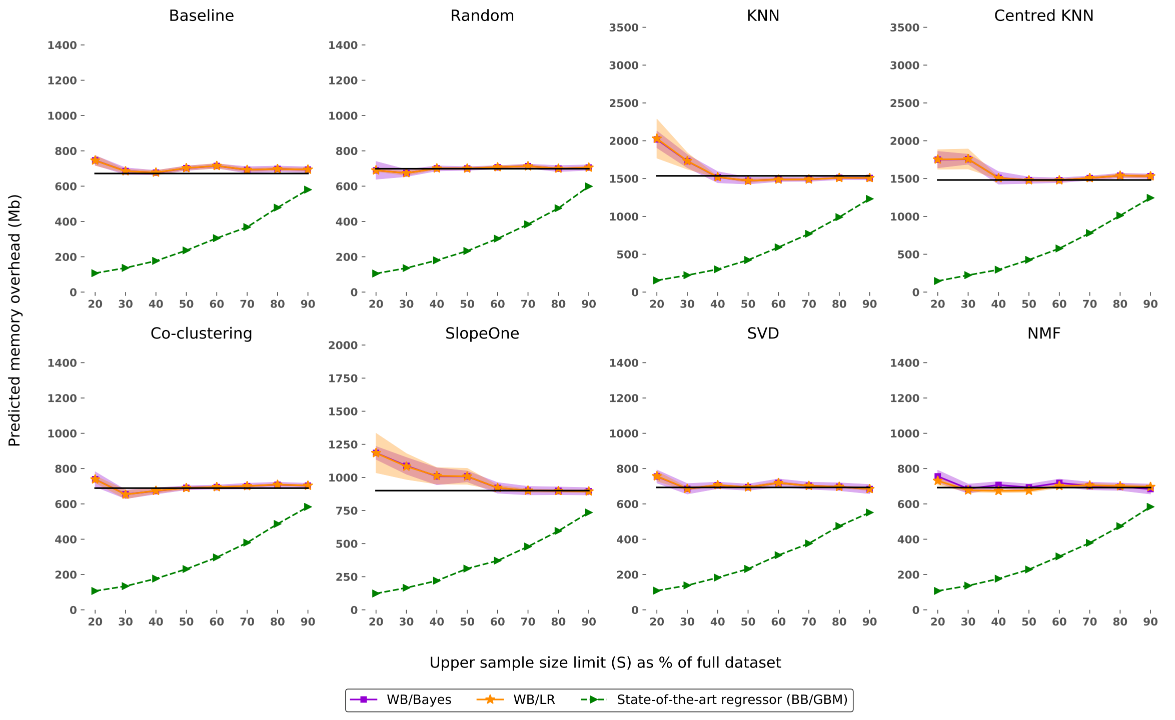

Figure 5.5: Predicted memory overhead for MovieLens 1M for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead for the entire dataset.

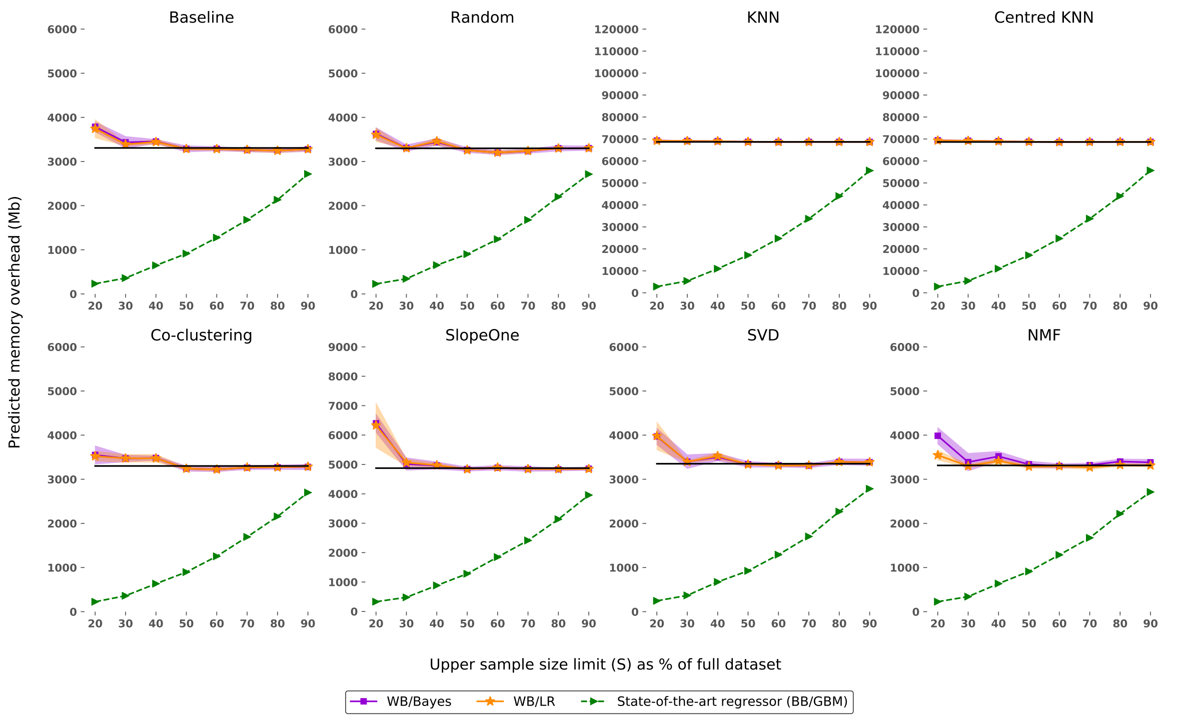

Figure 5.6: Predicted memory overhead for GoodBooks 10K for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead for the entire dataset.

5.4.4 Recommendations' Quality Prediction Models for CF

To fully assess the performance of a CF algorithm on a given input, we need to examine both efficiency and effectiveness. Although, the latter (i.e., effectiveness) has been previously studied in the literature [25, 26, 58], and it is not within the main scope of this thesis. However, for the purpose of completion and comprehensiveness, we demonstrate that our method also works for predicting the quality of the recommendations.

The results above illustrated how the proposed sampling strategy and White-Box models could address the efficiency prediction problem for traditional CF. For estimating effectiveness, we employ state-of-the-art Black-Box regression models, such as Ada Boost Regressor (ABR), Support Vector Regressor (SVR), and Gradient Boosting Machine (GBM) to learn the quality of the recommendations given a CF algorithm, a collection of samples (drawn using the strategy in Algorithm 4), and characteristics of the input. These Black-Box regressors were selected for this task, as they achieved the lowest error (i.e., RMSE), while predicting the effectiveness of the recommendations based on various values of S.

Figures 5.7, 5.8, and 5.9 show the predicted recommendation effectiveness for the popular CF algorithms implemented in Surprise [37]. We note that a sample size of 30-40% (same size as for efficiency models) attains good accuracy for predicting the effectiveness of a

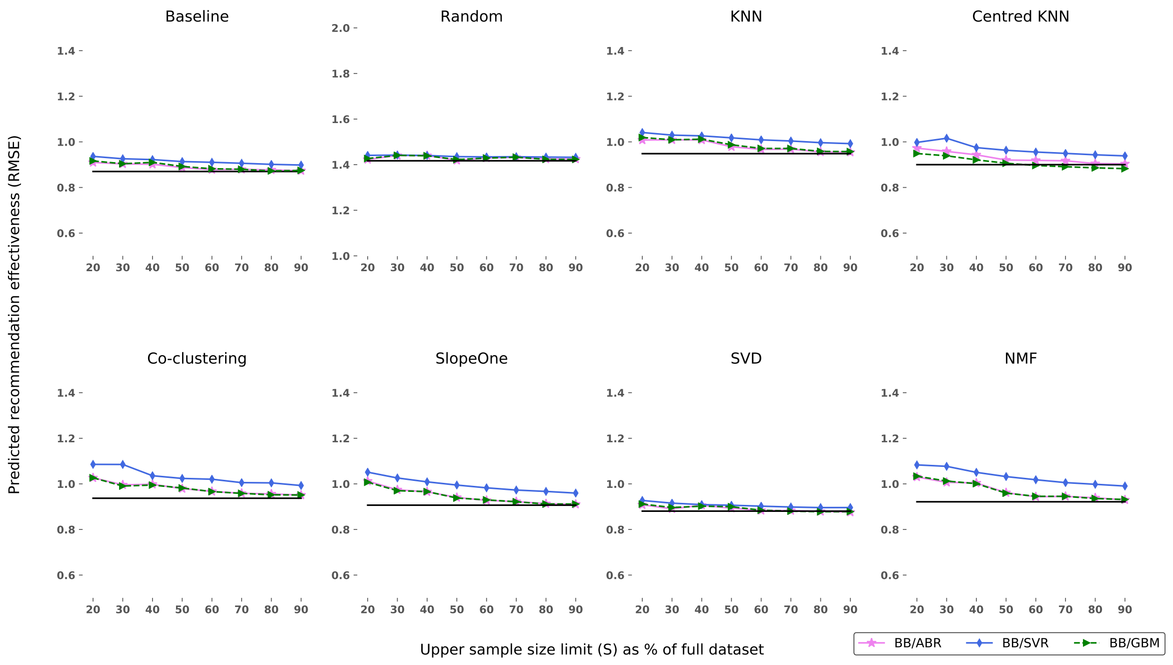

Figure 5.7: Predicted effectiveness (i.e., recommendation quality) for MovieLens 100K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection.

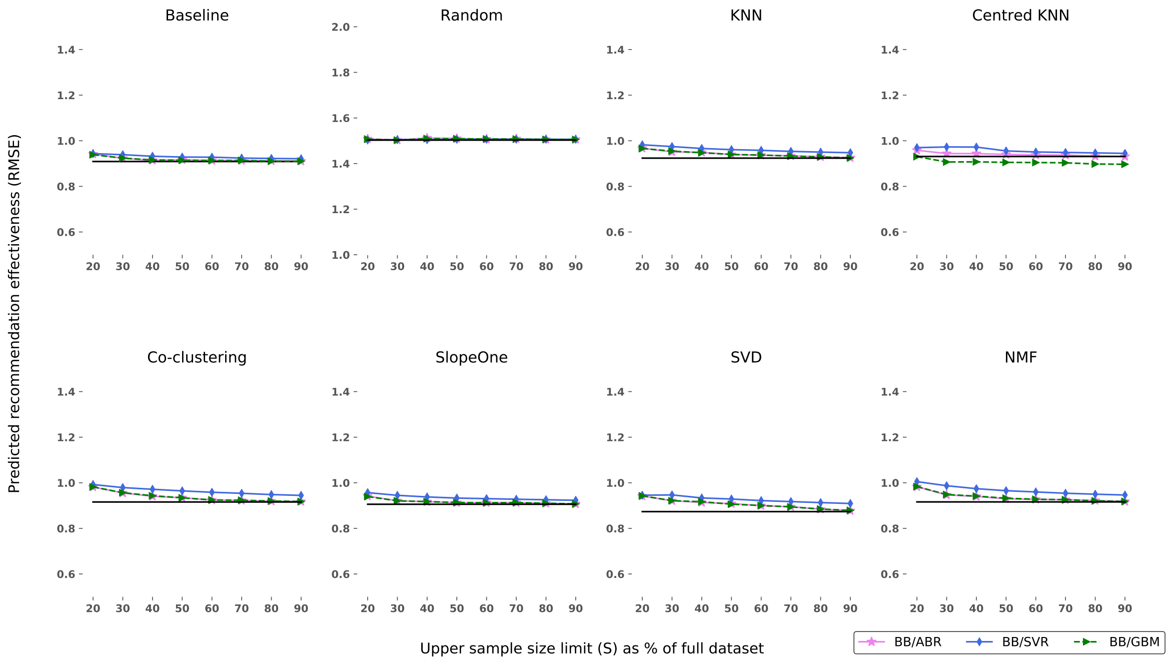

Figure 5.8: Predicted effectiveness (i.e., recommendation quality) for MovieLens 1M for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection.

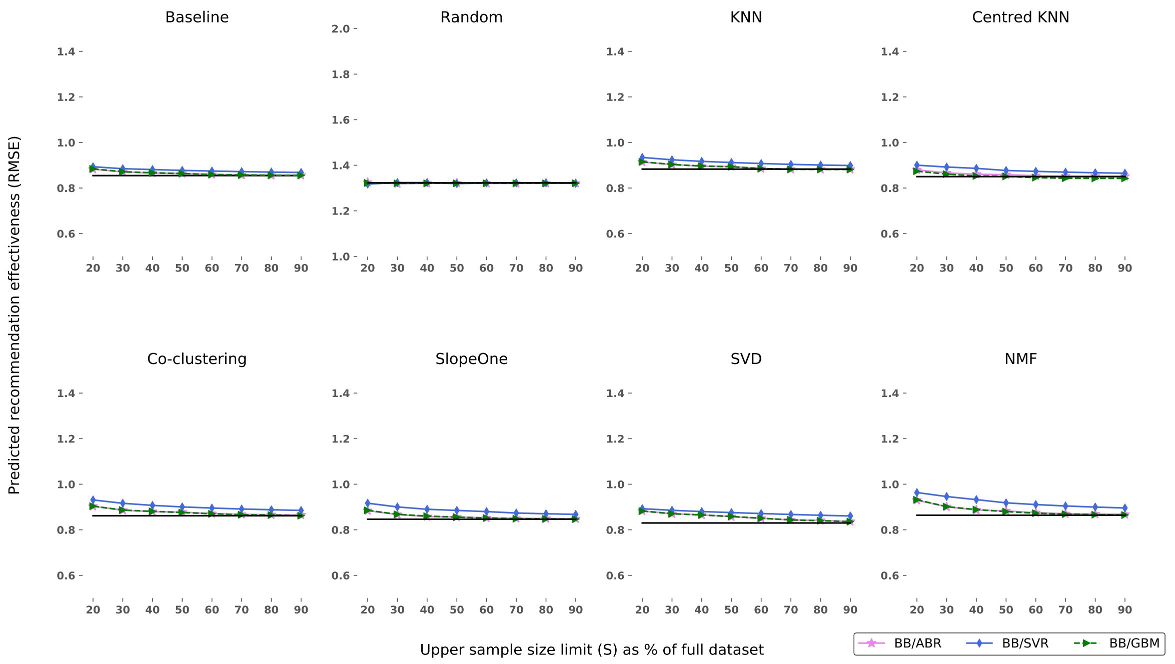

Figure 5.9: Predicted effectiveness (i.e., recommendation quality) for GoodBooks 10K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection.

CF model. Consequently, our cost models allow one to draw a single set of samples for predicting both the effectiveness and efficiency of a CF in one go.

5.5 Chapter Summary

In this chapter we have introduced the problem of resource consumption estimation for traditional CF models. Then, we analysed the time and space complexity of the CF, captured using big-O, described in the literature. However, it is often the case that design decisions may make the complexity characteristics of particular CF implementations to diverge from the theoretical bounds. To address this, we derived and proposed time and space complexity equations based on the actual implementation of the CF representatives found in [37]. In Section 5.3 we illustrated the evaluation metrics and baselines used, and how the proposed sampling strategy was employed during the experimental evaluation. Finally, Section 5.4 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. Despite its simplicity, our proposed methodology and resource cost models for CF algorithms managed to considerably outperform in accuracy even the best Black-Box regressor (BB/GBM). Furthermore, our approach was also faster than all contenders, as highlighted in Tables 5.3 and 5.4, and utilised only a fraction of the resources used by them.

Chapter 6

Modelling Resource Consumption in Deep Learning CF

This chapter presents the resource consumption cost models for predicting the processing time and GPU overhead for CF representatives from different neural architectures. We first discuss the problem of estimating efficiency in DL CF, with a particular focus on GPU utilisation, and the hardships associated with it. Then, we explore the neural CF algorithms' implementations investigated in this thesis, and formulate time and space complexity equations that capture the resource requirements of the DL CF models. Section 6.3 illustrates the evaluation metrics and describe how we collected GPU memory measurements using an NVIDIA Management Library (NVML)-based profiler. Furthermore, we describe the parameter configuration used for each DL CF based on the suggestions provided in the corresponding literature, and we explain how to sample the URM using Algorithm 4 outlined in Chapter 4. Section 6.4 showcases the experimental evaluation of the efficiency and effectiveness cost models using two main use-cases: (a) when the neural CF have been trained using only one GPU card, and (b) when the same algorithms have been trained using multiple GPUs. As the neural CF models are used in both rating prediction and ranking tasks we use different metrics for effectiveness cost models to assess the quality of recommendations in each scenario. Specifically, the rating prediction quality is measured using Root Mean Squared Error (RMSE), while the quality of the ranking is gauged using Normalised Discounted Cumulative Gain (NDCG). Also, in Section 6.4, we demonstrated that the resource consumption of the DL CF can be predicted using the same methodology irrespective of the number of GPUs used. Finally, a summary of the chapter is outlined in Section 6.5.

6.1 Problem Overview

The previous chapter described the problem of efficiency prediction during the highly expensive training phase for different representatives from the traditional CF design space. These issues extend to other families of models too, such as the Deep Learning (DL) CF, where the time- and resource-hungry algorithms are trained and deployed using specialised and expensive hardware [32]. Neural CF are not as available and easy to use as traditional CF as they often require long training times and powerful GPU cards [159]. Moreover, the cost of resources, in particular, GPUs is not negligible, as cloud providers, such as Amazon Web Services (AWS), charge fixed rates per hour depending on the type and size of the instance used [160, 161]. If the users do not know the processing time of the DL workloads or how much GPU memory is needed they might (a) underestimate the hardware needs choosing a smaller instance that will not be able to process the load, or (b) overestimate the resource utilisation, and therefore, they might end up overpaying for the requested hardware [160, 161]. Thus, the accurate prediction of the resource consumption during the training phase of neural CF models gains exceptional practical interest to establishments of all sizes from both academia and industry.

One of the current drawbacks in determining GPU utilisation in orchestration frameworks, such as Kubernetes [139], is that there is limited information about the workload of a job prior to deploying it on the nodes, which could lead to performance issues and bottlenecks for both users and hardware providers (e.g., cloud-based services or local datacentres found in research labs) [162]. These issues are also encountered when training neural CF [159]. Another limitation which impacts individuals who require GPUs is based on the fact that it is non-trivial to determine and/or calculate GPU utilisation [163]. This is often achieved using specialised tools, such as GPU profilers, and custom infrastructure for executing the jobs on isolated machines/nodes [162]. Thus, predicting resource consumption for DL workloads, including neural CF, becomes of critical importance. Thus, we propose addressing this problem for a number of popular DL CF architectures answering **RQ6** and **RQ7** presented in Section 1.2.

6.2 Complexity Analysis

This section presents the asymptotic time and space complexity equations of the DL CF representatives, which capture the upper bound on how resource consumption (i.e., processing time, GPU memory) grows or declines based on the size of the input [91]. As evidenced in Section 6.4, knowing the computational complexity of the neural CF is a key component, which enables the accurate prediction of their efficiency.

6.2.1 Processing Time Complexity Equations

The authors of **BiVAE** algorithm report in [8] that the time complexity of one optimization epoch over the user and item objectives is $O(mnf)$. However, if we further fix the number of latent features, f, BiVAE can be trained in $\boldsymbol{O(mn)}$ time.

For the **VAECF** model, as presented in [80], an asymptotic complexity analysis is not provided by the literature. Nevertheless, by examining its implementation, showcased in Cornac [38], we concluded that the user and item objectives are computed in $\boldsymbol{O(mn)}$ steps, in a similar manner as BiVAE.

The implementation of **NCF** revealed that during training, the algorithm decomposes the URM given a fixed number of latent factors f, computing the user objectives in $O(mf)$ and the item objectives in $O(nf)$ steps, respectively. This brings the overall time complexity of NCF to $\boldsymbol{O(mf + nf)}$ considering the latent factors f, or $\boldsymbol{O(m + n)}$ for a fixed number of factors.

The authors of **ConvMF** [10] report that for each epoch, the user and item latent models are updated in $O(mf^3 + nf^3 + \rho f^2)$, where f represents the number of latent factors. If we further fix f, ConvMF trains in $\boldsymbol{O(m + n + \rho)}$ steps.

Table 6.1 presents the literature and implementation time complexity equations for the neural CF representatives. For BiVAE and ConvMF the equations reported in the literature are also reflected in the implementation, while for VAECF and NCF the literature does not report any time complexity analysis.

Table 6.1: Time complexity equations for neural CF.

	Literature Complexity	**Implementation Complexity**
BiVAE	$O(mnf)$ [8]	$O(mnf)$
VAECF	–	$O(mnf)$
NCF	–	$O(mf + nf)$
ConvMF	$O(mf^3 + nf^3 + \rho f^2)$ [10]	$O(mf^3 + nf^3 + \rho f^2)$

6.2.2 Space Complexity Equations

When **BiVAE** computes the user and item objectives as part of its training process, user and item encoders are defined and stored in matrices of size $O(mf)$ and $O(nf)$, respectively, where f represents the fixed number of latent factors. To this end, BiVAE requires an additional $\boldsymbol{O(mf + nf)}$ of storage during training or $\boldsymbol{O(m + n)}$ if the fixed number of latent factors is disregarded.

Since **VAECF** is based on a VAE neural architecture, its space requirements are similar to the ones of BiVAE. Therefore, the user and item encoders are stored in matrices of size $O(mf)$ and $O(nf)$, with an overall space complexity of $\boldsymbol{O(mf + nf)}$.

During training, **NCF** estimates the scores a user would give to a set of items. This operation requires an additional array of size n to store the predicted scores. Hence, the space complexity of NCF during training is $\boldsymbol{O(n)}$.

ConvMF stores its user and item latent models in tensors of size $O(m)$ and $O(n)$, respectively, requiring a total of $\boldsymbol{O(m + n)}$ of storage.

Similarly to the traditional CF, the literature does not report the additional space required by the neural CF during training. However, the space complexity equations derived from the implementation of the algorithms are summarised in Table 6.2.

Table 6.2: Space complexity equations for neural CF.

	Literature Complexity	**Implementation Complexity**
BiVAE	–	$O(mnf)$
VAECF	–	$O(mnf)$
NCF	–	$O(mf + nf)$
ConvMF	–	$O(mf^3 + nf^3 + \rho f^2)$

6.3 Experimental Settings for Neural CF

This section provides an overview of the evaluation metrics used and explains how the GPU memory measurements were acquired during DL CF training phase. Then, we describe how we utilised the sampling strategy outlined in Chapter 4, to sample the input and retrieve samples with different characteristics to estimate the efficiency and effectiveness of the neural algorithms.

6.3.1 Evaluation Metrics and Baselines

Similarly to the traditional CF, for the neural models, we recorded their processing time and GPU memory while training on samples of various sizes. The processing time was captured using the *getrusage* method from Python's *resource* module as explained in Section 4.5.1 and Section 5.3.1. As the DL CF models were trained on GPU cards, we logged the corresponding memory overhead required during training. This was achieved by testing several GPU profilers, out of which we used the *pyNVML* module[1]. This library provides

[1]pyNVML's documentation is available at `https://pypi.org/project/nvidia-ml-py3/`.

Python bindings to the NVIDIA Management Library (NVML)[2], which logs information about GPU management, as well as its monitoring functions. NVML-based profilers are a popular choice when it comes to logging GPU usage as shown in the literature [164, 165, 166]. For our experimental evaluation, using *pyNVML* we first initialise the current GPU card or set of GPU cards that is/are being used for training the neural CF. Then, through the *nvmlDeviceGetMemoryInfo()* method we can record information about the GPU memory that has been used, and also about the total memory of the GPU card. Using this approach we take an initial snapshot of the GPU memory before the CF model is being trained and another snapshot once the training is done. Then, the overall GPU memory overhead is computed as the absolute difference between the two snapshots.

For measuring the effectiveness of the DL CF models, we use the built-in metrics implemented in Cornac [38]. Thus, for each neural model we log its Root Mean Squared Error (RMSE) and Normalised Discounted Cumulative Gain (NDCG) with respect to different sample sizes S. Once a DL CF algorithm has been trained on a set of sub-samples we record its accuracy with a test collection split using the 80/20% rule [81].

To predict resource consumption of neural CF, we use the Black-Box based regressor (GBM) as a baseline and contender to the proposed linear and Bayesian regressors based on the White-Box approach. Similarly to traditional CF, to estimate the effectiveness of the recommendations using different sample sizes, we employ various off-the-shelf state-of-the-art regressors, such as ABR, SVR, and GBM.

6.3.2 Parameter Configuration

Our resource consumption cost models for DL CF are agnostic to the parameter configuration required by the neural models. While the parameter values can be tuned using grid search methods or by experts in the field [167], we utilised the recommended values and activation functions provided by the literature and the authors of the models [8, 80, 9, 10]. Thus, for the VAE-based models, we utilised a batch size of 128 and 500 epochs, while the learning rate was set to 0.001 as suggested by [8, 80]. For the NCF model, we utilised a set of [64, 32, 16, 8] layers, 10 epochs and a batch size of 256, while keeping a learning late of 0.001 [9]. Finally, for ConvMF, we used 5 epochs for optimizing the CNN for each overall training epoch, while the corpus and vocabulary were precomputed based on the documents (i.e., plot summaries) provided by the authors as described in [10].

[2]NVML's documentation is available at `https://developer.nvidia.com/nvidia-management-library-nvml`.

6.3.3 Sampling Strategy

The samples used for training the DL CF were acquired in a similar manner to those for traditional CF, as presented in Section 5.3.2. To this end, 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%, $\ldots$, 100%) using Algorithm 4. Then, for each draw, we used a fixed random seed $\mathcal{s}$ from a predefined set of seeds for reproducibility purposes.

When sampling the URM for DL CF, in Algorithm 4, we set the confidence to 99%, and use a predefined Coefficient of Variation (CoV) of 0.1. For the time budget, we set the overall time quota T to 200 seconds for smaller dataset (i.e., ML 100K), apart from NCF which has a more expensive training time and required a T of 1000 seconds. For the larger dataset (i.e., ML 1M) a time budget of 400 seconds sufficed for most neural CF, while NCF required a T of 10000 seconds. Similarly to the samples drawn for traditional CF, this setup offered a good trade-off between the number of samples needed and their quality. These settings were selected based on the available resources for T (e.g., amount of time dedicated to the experimental evaluation), and empirical evidence for CoV as shown in Section 4.5.4.

6.4 Experimental Evaluation

This section discusses the empirical studies which assess the proposed efficiency and effectiveness cost models for neural CF. Thus, we first present a comparative assessment across the processing time prediction methodologies for DL CF that have been trained on one vs. multiple GPUs. Then, Section 6.4.2 shows how the GPU memory overhead for CF can be estimated using the White- and Black-Box methods. Finally, we illustrate the effectiveness cost models for gauging the quality of the recommendations based on various input characteristics and state-of-the-art regressors.

6.4.1 Processing Time Cost Models

In our experimental evaluation, we address the processing time prediction problem for DL CF that have been trained on one and two GPUs, as described in **RQ6**. Consequently, we assess the proposed White-Box approach (WB/LR and WB/Bayes) against the Black-Box method, namely an off-the-shelf baseline regressor (BB/GBM). Figures 6.1 and 6.2 depict the estimated training time for the complete datasets for various values of S (upper sample size limit). 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 were applied. The horizontal black line represents the actual training time over the entire dataset. Similarly to the traditional CF efficiency cost models, presented and evaluated in Chapter 5, 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. In addition, as there is less variance in the processing times of neural CF, the prediction error interval and uncertainty areas are smaller/tighter than the ones depicted in the evaluation of traditional CF models (Chapter 5).

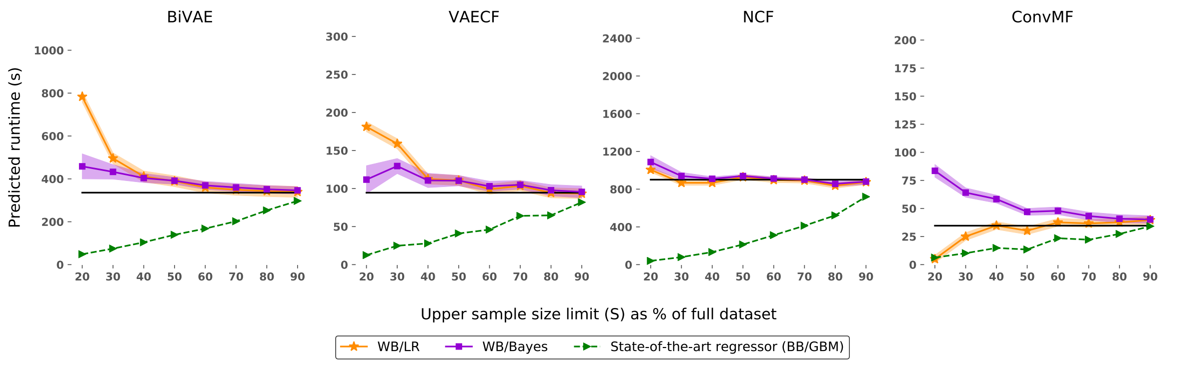

Figure 6.1: Predicted processing time for MovieLens 100K using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

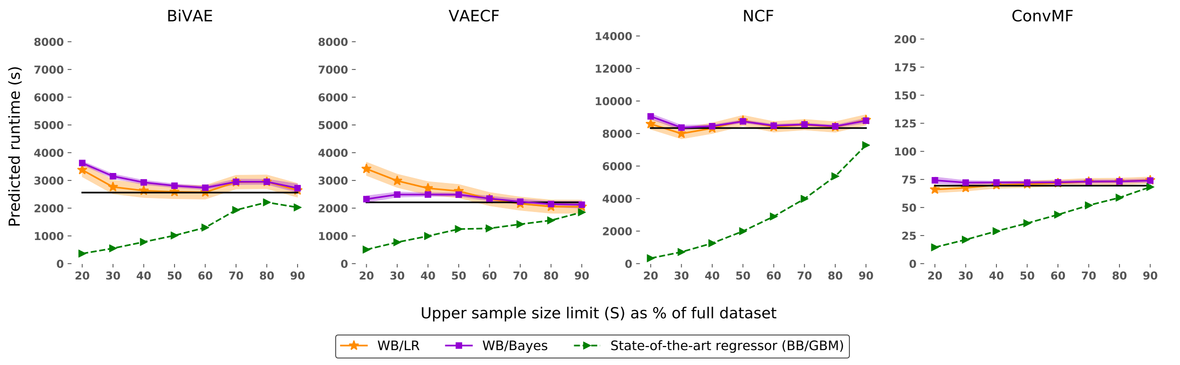

Figure 6.2: Predicted processing time for MovieLens 1M using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

Furthermore, the second part of the evaluation of the processing time cost models investigates how the proposed approaches can be employed to estimate the training time of the neural CF when these have been trained on two GPUs. Figures 6.3 and 6.4 showcase the White-Box prediction models against the main contender (i.e., Black-Box regressor BB/GBM). Irrespective of whether the DL CF have been trained on one or multiple GPU cards, the results indicate that our approach (i.e., linear and Bayesian regressors trained over the time

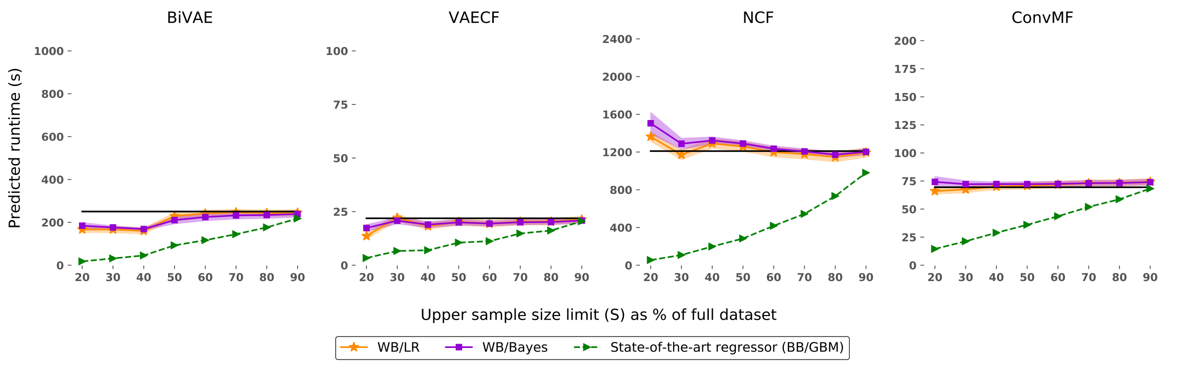

Figure 6.3: Predicted processing time for MovieLens 100K using 2 GPUs for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

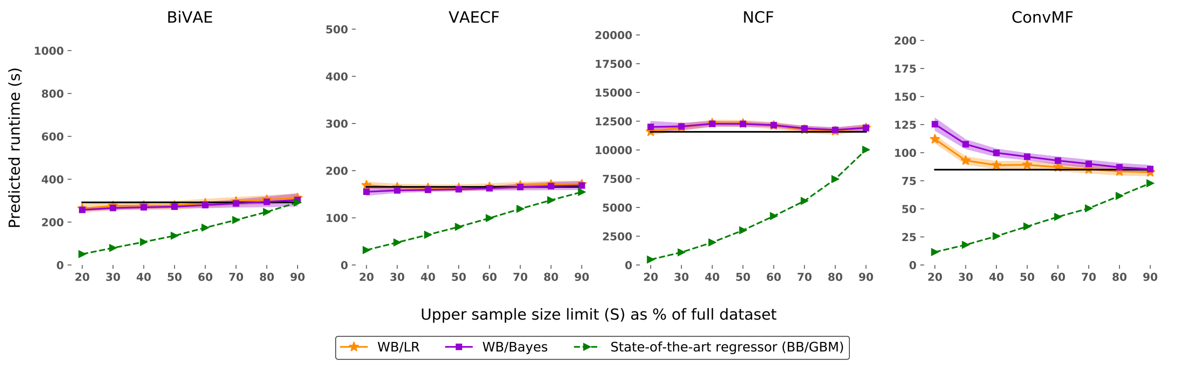

Figure 6.4: Predicted processing time for MovieLens 1M using 2 GPUs for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual training time over the entire dataset.

complexity equations) outperforms the much more complex best performing state-of-the-art regressor (BB/GBM). In most cases, a 30%-40% upper sample size limit seems enough to allow WB/LR to achieve highly accurate prediction.

Interestingly, training the neural CF on multiple GPUs reduces their processing time for some architectures, such as the VAE-based CF (i.e., BiVAE and VAECF), especially for larger datasets. For these neural CF, our approach initially underestimates the processing time for smaller sample sizes, as seen in Figures 6.3 and 6.4. We believe this behaviour could be due to the overheads in splitting the workloads across two GPUs. However, further investigation on how the processing time scales across multiple (i.e., 4, 8) GPUs for neural CF would be required to gain more insights into why the predictors are initially underestimating the resource consumption. In addition, other DL CF representatives (e.g., NCF and ConvMF) exhibit similar processing times when trained on one vs. two GPUs. As discussed in the literature [168], if the training task of the neural models is simple enough, using multiple

GPUs will not bring much value in the reduction of the processing time. Nevertheless, our proposed efficiency cost modelling framework for neural CF can successfully estimate the processing time of the DL CF algorithms regardless of the number of utilised GPUs.

6.4.2 GPU Memory Cost Models

The previous section demonstrated how the proposed approach (the White-Box method) can be applied to estimate the processing time of DL CF during training. However, as part of the proposed framework that addresses the problem of resource consumption estimation in neural CF, we also build and investigate GPU memory cost models to address **RQ6**. To this end, we provide a comparative assessment of the White- and Black-Box methods for gauging the GPU memory overhead during training (on one vs. two GPUs) for different DL CF. Figures 6.5 and 6.6 depict the estimated GPU overhead for the complete datasets for various values of S (upper sample size limit) for neural CF trained on one GPU, using the proposed approach (WB/LR and WB/Bayes) and the Black-Box baseline (BB/GBM). As the GPU utilisation steadily increases across samples, both the proposed approach (White-Box) and the contender (Black-Box) achieve high accuracy in the DL CF memory prediction task. However, for BiVAE and VAECF, WB/Bayes is slightly closer to the ground truth data (i.e., black horizontal line) than the contender BB/GBM. This result is also exhibited when the DL CF are trained on two GPUs as illustrated in Figures 6.7 and 6.8. When estimating the GPU memory for NCF, in most cases, all predictors can accurately predict the memory consumption. An interesting case arises when NCF is trained using 2 GPUs on the smaller dataset, as seen in Figure 6.7. In this case, the contender BB/GBM provides a more accurate estimation than WB/LR and WB/Bayes. We believe this could be due to the overheads present when training on 2 GPUs, which are not captured by the complexity equations.

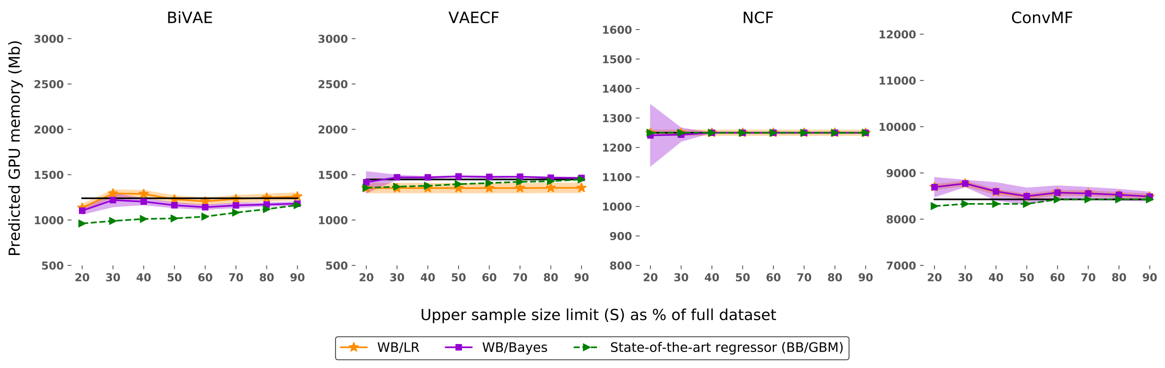

Figure 6.5: Predicted GPU memory for MovieLens 100K using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset.

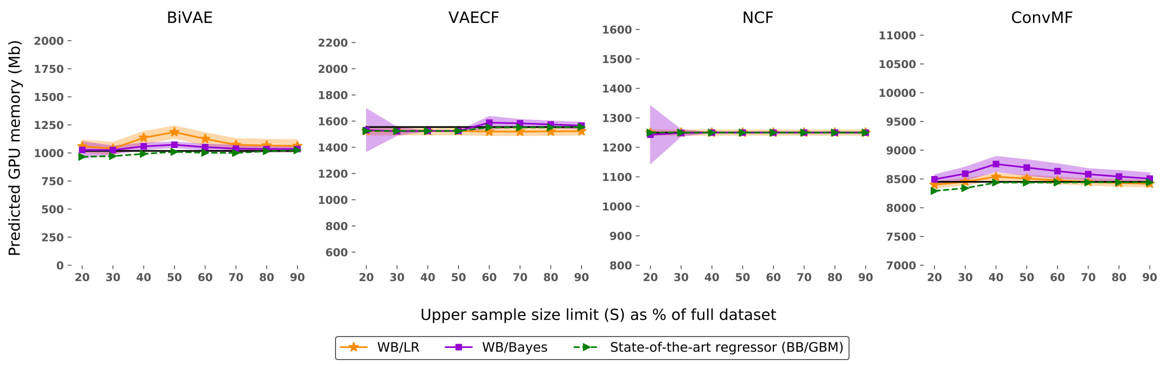

Figure 6.6: Predicted GPU memory for MovieLens 1M using 1 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset.

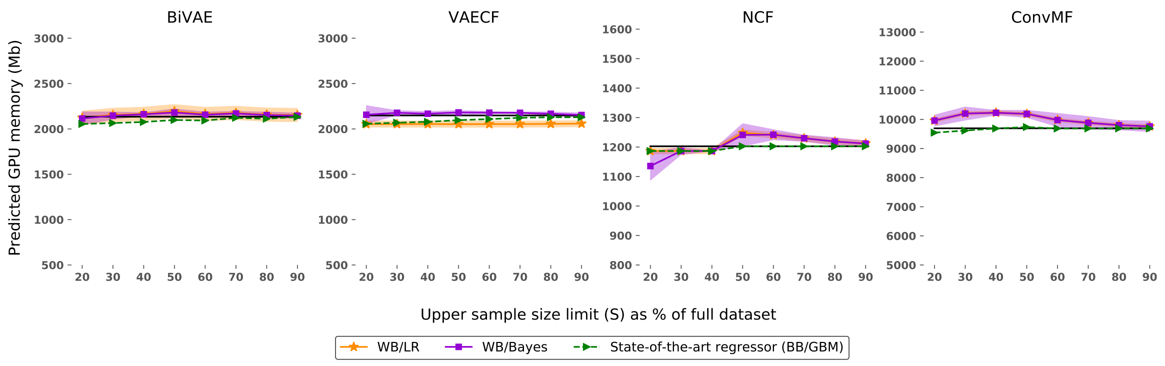

Figure 6.7: Predicted GPU memory for MovieLens 100K using 2 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset.

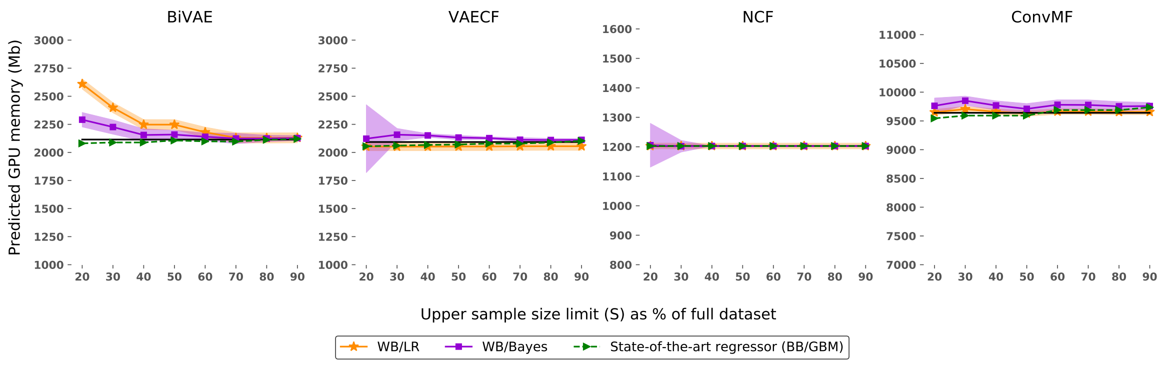

Figure 6.8: Predicted GPU memory for MovieLens 1M using 2 GPU for various values of S using our approaches (WB/LR and WB/Bayes) and the baseline (state-of-the-art regressor (BB/GBM)). The black horizontal line represents the actual memory overhead over the entire dataset.

In addition, Figures 6.7 and 6.8 depict the estimated GPU memory utilisation when the DL CF have been trained on two GPUs. Similarly to the processing time cost models, evaluated in Section 6.4.1, the White-Box methodology can successfully quantify GPU overhead even when the neural CF have been trained on multiple GPUs. Regardless of the number of GPUs used during training, the Black-Box methodology remains a strong contender, as it can predict the memory overhead based on characteristics of the input and the GPU memory measured across different samples.

6.4.3 Effectiveness Cost Models

As part of **RQ7**, we initially investigated how the effectiveness of neural CF varies across different sample sizes when the models are trained on one vs. two GPUs. Consequently, we trained the DL CF in both scenarios, and recorded their RMSE and NDCG for each upper sample size S. Figures 6.9 and 6.10 present the actual RMSE of the predicted ratings of each DL CF representative, while Figures 6.11 and 6.12 depict the actual NDCG acquired in the items ranking task by each neural CF. As expected, in most cases, the effectiveness of the DL CF is not impacted if the models are trained on one or multiple GPUs. However, as outlined in Section 6.4.1, for VAE-based CF the processing time changes depending on the number of GPUs utilised during training.

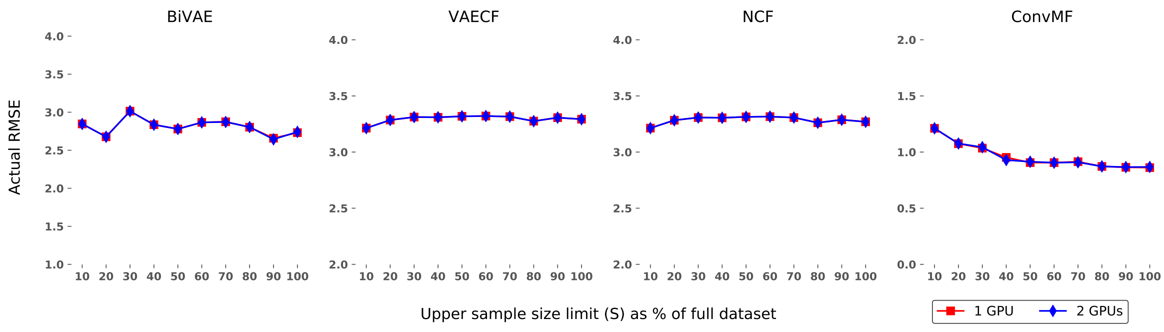

Figure 6.9: Actual RMSE of the recommendations for MovieLens 100K for various values of the upper sample size S.

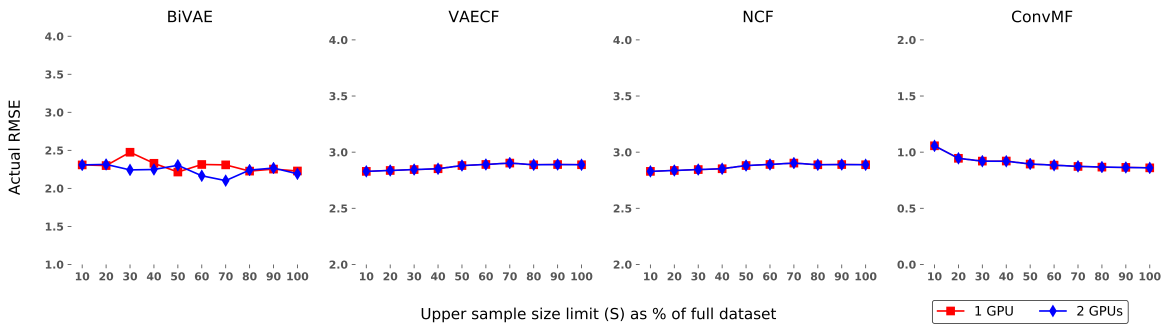

Figure 6.10: Actual RMSE of the recommendations for MovieLens 1M for various values of the upper sample size S.

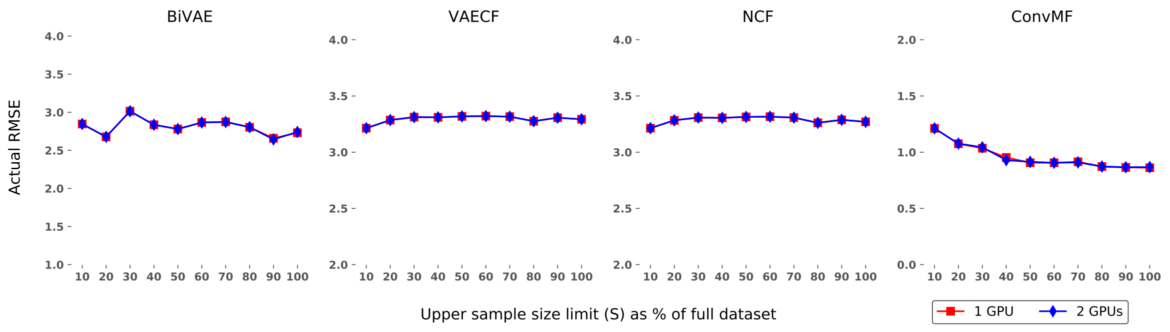

Figure 6.11: Actual NDCG of the recommendations for MovieLens 100K for various values of the upper sample size S.

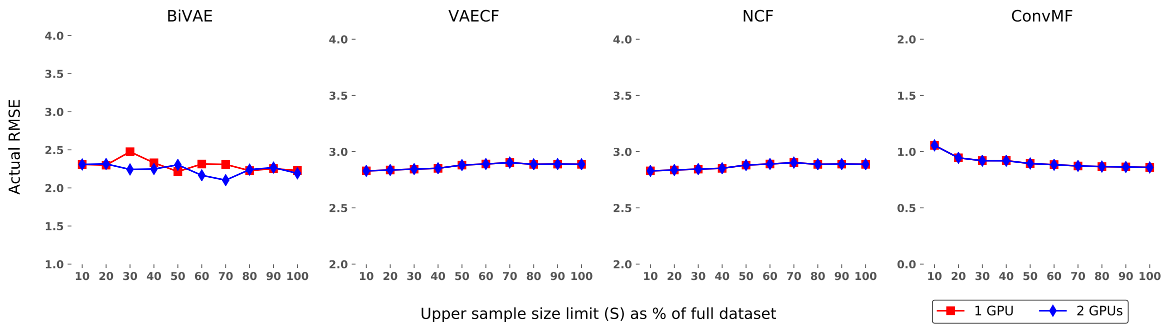

Figure 6.12: Actual NDCG of the recommendations for MovieLens 1M for various values of the upper sample size S.

Similarly to the traditional CF, we utilised effectiveness cost models that predict the accuracy of the recommendations (e.g., RMSE, NDCG) based on characteristics of their input. To this end, we utilised off-the-shelf state-of-the-art Black-Box regression models, such as Ada Boost Regressor (ABR), Support Vector Regressor (SVR), and Gradient Boosting Machine (GBM) to estimate the effectiveness metrics of the entire collection for a given neural CF, based on the accuracy measured on a set of samples drawn using the strategy described in Algorithm 4.

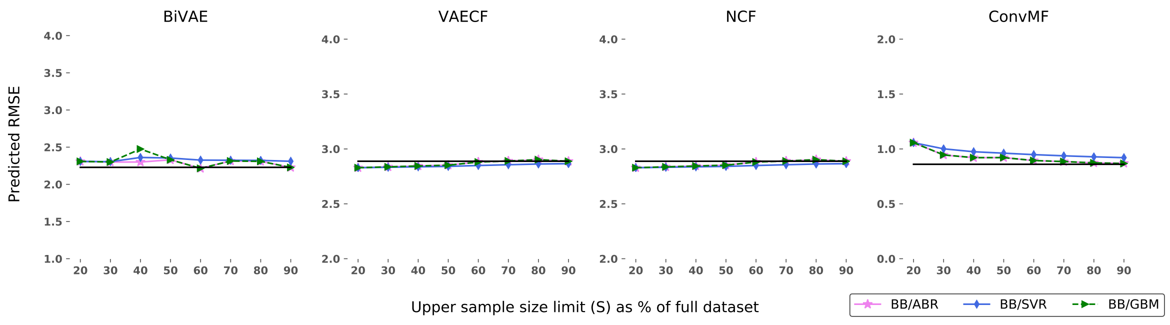

Figure 6.13: Predicted RMSE of the recommendations for MovieLens 100K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection.

Figures 6.13 and 6.14 depict the predicted RMSE of the recommendations for the selected

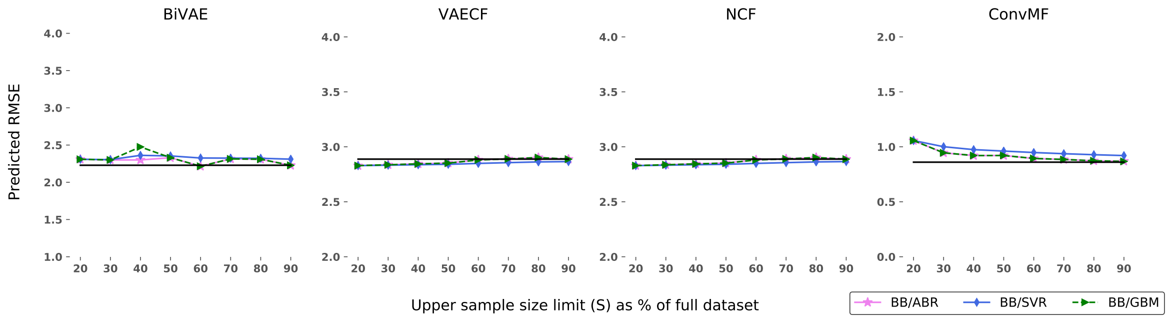

Figure 6.14: Predicted RMSE of the recommendations for MovieLens 1M for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (RMSE) over the entire collection.

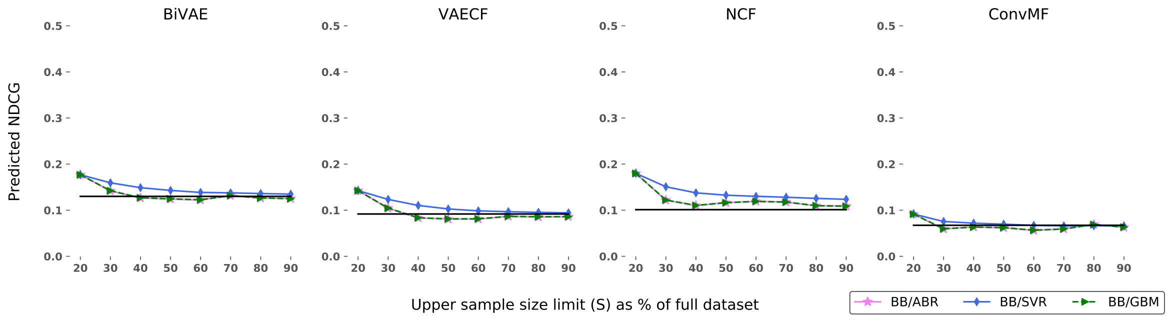

Figure 6.15: Predicted NDCG of the recommendations for MovieLens 100K for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (NDCG) over the entire collection.

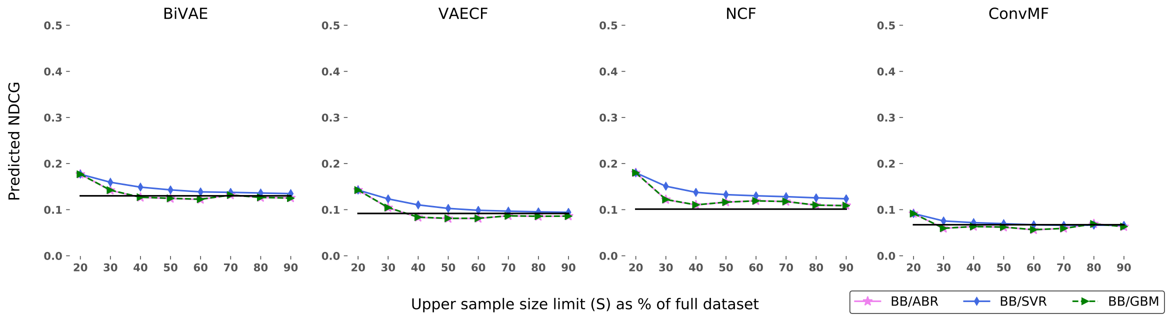

Figure 6.16: Predicted NDCG of the recommendations for MovieLens 1M for various values of S using state of the art regressors (Black-Box approach). The black horizontal line represents the actual recommendation quality (NDCG) over the entire collection.

DL CF. For most representatives, the empirical evaluation showed that an upper sample S of 30-40$\%$ suffices for gauging the effectiveness of the CF on the entire dataset. Comparably, if the DL CF were assessed in a ranking task [3], and hence a ranking metric (e.g., NDCG) would have to be estimated, our proposed solution can successfully address this, as illustrated in Figures 6.15 and 6.16.

6.5 Chapter Summary

This chapter described the problem of efficiency estimation for various architectures of DL CF, highlighting how the proposed White- and Black-Box approaches can be extended to predict resource consumption during training for the neural CF. After introducing the research problem on how we can gauge the efficiency and effectiveness of the DL CF when these are trained on one vs. multiple GPUs, we outlined and analysed the time and space complexity of the DL CF using big-O notation [91] as part of Section 6.2. If the asymptotic complexity of the neural CF was not presented in the relevant literature, we provided complexity equations derived from the algorithms' implementations found in Cornac [38]. Section 6.3 described the evaluation metrics and the profilers used to collect processing time and GPU utilisation for the various neural CF. In addition, we also discussed the parameter configuration for each DL representative, and how the proposed sampling strategy was employed during the experimental evaluation. Lastly, Section 6.4 presented a comparative assessment of the performance of White- and Black-Box approaches in the problem of resource estimation for DL CF, which were trained on both one and two GPUs. The results demonstrated that for predicting the processing time of the neural CF, the linear and Bayesian White-Box regressors computed over the time complexity equations outperformed the best off-the-shelf Black-Box predictor (GBM) trained solely on characteristics of the URM. However, for the GPU overhead estimation task, both the White- and Black-Box methods successfully managed to gauge the additional memory required during training of the DL CF. Furthermore, similarly to traditional CF, the effectiveness of neural CF can be predicted using off-the-shelf Black-Box regressors, irrespective of the number of GPUs used for training the DL CF.

Chapter 7

Conclusions

The final chapter summarises the thesis statement introduced in Chapter 1, revisits the research questions investigated throughout the experimental chapters (Chapter 4, Chapter 5, and Chapter 6), noting the findings and contributions. In addition, we discuss the limitations of the work undertaken in the Ph.D., and explore a number of potential future research directions.

7.1 Thesis Summary

This thesis investigated how the efficiency of CF algorithms, with a particular focus on resource consumption (e.g., processing time, RAM, GPU memory) can be quantified and predicted using two main methodologies (i.e., the White- and Black-Box approaches) for a wide area of the CF design space, containing both traditional and neural CF representatives. The proposed work was first introduced in the thesis statement outlined below.

This thesis argues that the resource consumption of the CF models during training phase can be estimated using characteristics of the input (i.e., URM), sampling-based probabilistic analysis, and efficiency cost curves. The scope of this work is threefold: firstly, we map the design space by examining state-of-the-art CF algorithms, including traditional and neural approaches, with respect to different characteristics, such as algorithmic complexity, internal representation of input data, sets of computations, etc.; secondly, we show how efficiency cost models can be used to predict the CF algorithms' performance, based on the white-box and black-box approaches; finally, we provide an evaluation methodology that assesses the CF models on accuracy (i.e., quality of the recommendations) versus training cost (e.g., processing time, memory overhead) curves, by exploiting the efficiency-effectiveness trade-offs and limitations present in the implementations, and experimenting with different sampling techniques.

In the chapters that followed, research that supports this statement through methodologies, sampling algorithms, and experimental evaluations was presented, covering the main research questions explored in this thesis, as well as their answers and corresponding theoretical and empirical findings. Specifically, **Chapter 4** provided an overview of the current standard practice sampling techniques [25, 26] for sampling a dataset, represented as an URM, with the purpose of estimating the *effectiveness* of a CF model on the full dataset. This chapter also examined if the current standard practice sampling approaches can be used for drawing samples from an URM to predict the *efficiency* of a CF algorithm on the complete dataset. Our findings indicated a number of drawbacks of the latter, and hence, we proposed a new sampling approach, and showcased its applicability for both efficiency and effectiveness prediction purposes. In addition, we augmented the proposed sampling scheme with an adaptive component, which allows users to sample the URM dynamically given the operational constraints (e.g., time budget), and the desired quality of the samples, captured through a Coefficient of Variation (CoV) threshold. Then, **Chapter 5** covered the *efficiency* and *effectiveness* cost models for predicting the resource consumption in traditional CF algorithms. This chapter also illustrated and compared the two main approaches for estimating efficiency, namely, the *White-Box* and *Black-Box* methodologies. Finally, **Chapter 6** discussed the resource cost models for neural CF algorithms. As part of this chapter, we investigated how *resource consumption* (e.g., processing time, GPU memory) and *recommendations' quality* varied during the training phase of the DL CF models on one and multiple GPUs, as well as how they can be predicted using characteristics of the input and different methodologies.

7.2 Research Questions

While investigating and modelling the efficiency of traditional and neural CF through theoretical and practical methods, a number of novel and interesting areas of discussion emerged. This section revisits the challenges and findings of our research, guided by the main research questions (RQs) of each chapter outlined below.

7.2.1 Sampling Strategies

The first RQs examined in this thesis were investigated as part of **Chapter 4**, with a particular focus on a comparative performance assessment between standard practice sampling schemes, presented in [26, 25], and our proposed adaptive sampling approach, outlined in Algorithms 4 and 5, which given an URM, it can draw samples that can be used for both efficiency and effectiveness prediction. In this chapter, we evaluated our sampling algorithms and the standard practice sampling baseline, by measuring the accuracy of the predicted processing times, showcased in Figures 4.3 and 4.4, and the effectiveness (i.e., quality of recommendations) of CF models, illustrated in Figures 4.1 and 4.2, using samples acquired with both strategies. Then, to answer **RQ1**, we undertook an empirical study which demonstrated that standard practice sampling techniques cannot capture the complexity characteristics of the base data, as demonstrated by the non-dashed models in Figures 4.1 and 4.2, and hence, a different approach is needed for efficiency prediction, which led to the development of a revised adaptive sampling scheme, as presented in Section 4.4. This chapter also addressed **RQ2** through an ablation study, summarised in Tables 4.3, 4.4, and 4.5, that showcased how the quality of the samples can be gauged, and how to stop sampling the dataset to jointly satisfy a user-defined Coefficient of Variation (CoV) and/or a predefined operational constraint (e.g., maximum sample size, maximum time budget).

7.2.2 Traditional CF

The next three RQs were analysed and answered as part of **Chapter 5**, which presented the efficiency cost models for traditional CF algorithms. In this chapter, we firstly addressed **RQ3** with an initial investigation into how efficiency can be quantified in CF models, what are the current state-of-the-art methods for performance prediction, and how can we accurately predict processing time and memory overhead using complexity analysis. Then, we used the baseline solution for efficiency prediction using characteristics of the input and the best off-the-shelf regressors (i.e., Black-Box approach) covering **RQ4**. Furthermore, this chapter described the proposed resource consumption cost models for CF algorithms using the White-Box methodology. As part of answering **RQ5**, we provided an implementation-based complexity analysis for traditional CF models, described in Sections 5.2.1 and 5.2.2, followed by our proposed White-Box efficiency cost model for estimating processing time and memory overhead during the training phase of CF models. Finally, we conducted a comparative assessment of the White-box and Black-box methodologies, highlighting the benefits and limitations of each approach, as evidenced in Section 5.4. Furthermore, we showed that using a smaller sample of a dataset, the traditional CF models' performance and resource cost, could be estimated with our proposed methodology (WB/LR) without training on the entire collection, as illustrated in Table 5.4.

7.2.3 Neural CF

The last two RQs were addressed in **Chapter 6**, which illustrated how efficiency can be modelled in neural CF algorithms. As part of this work, we investigated how processing time and GPU memory utilisation vary while training the neural CF models on inputs/samples of different sizes using one and multiple GPU cards with the empirical results presented in

Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, and 6.8 . To this end, we analysed the training cost of the DL CF models, and answered **RQ4**. As part of this research question, we built a framework that measures and records the processing time and GPU utilisation of the various neural CF algorithms. Then, we used these measurements with the proposed White-Box and Black-Box approaches to compute the training cost of DL CF algorithms, and discussed the trade-offs of each method in Section 6.4. In this chapter, we also extended the previously mentioned framework to also capture and estimate the effectiveness of the neural models, depicted in Figures 6.13, 6.14, 6.15, and 6.16, which was further used in conjunction with the proposed sampling strategy to predict the accuracy of the DL CF models on the complete datasets, as part of answering **RQ5**.

7.3 Contributions

This thesis made a number of key contributions, outlined in Section 1.2, towards the accurate prediction of efficiency in CF. As part of this work, we provided conceptual and theoretical developments towards modelling resource consumption in CF algorithms during their highly expensive training phase, we built efficiency cost models to predict computational requirements for traditional and neural CF models, and we assessed them through several experimental evaluations. In the following paragraphs, we note the specific contributions that emerged from the experimental chapters.

In **Chapter 4**, the key contribution is a sampling strategy, described in Algorithms 4 and 5, that captures both the complexity and structural properties of the URM, and produces samples that can be further utilised for predicting both the efficiency and effectiveness of CF in one go. Thus, as part of **C1.1**, we presented an investigation, shown in Figures 4.3 and 4.4, into how current standard practice sampling strategies used for effectiveness prediction, described in Section 4.2, perform in the task of estimating resource consumption for CF models. Then, **C1.2** featured an adaptive sampling strategy, outlined in Algorithm 5, that dynamically draws samples from the URM to jointly satisfy a user-defined Coefficient of Variation (CoV) and/or a predefined resource constraint (e.g., maximum time quota) that can be used for both efficiency and effectiveness prediction of CF algorithms. As part of **C1.3**, we presented an experimental evaluation, which compares the proposed adaptive sampling strategy with the current standard practice approaches, as seen in Section 4.5.2 and Section 4.5.3. In addition, a sampling algorithm that assesses the quality of the samples based on a given confidence interval and a smart mean, and then decides if to further sample the URM based on the operational time and memory constraints has been developed as part of **C2.1** and presented as part of Section 4.3. Finally, **C2.2** comprised an empirical study, featured in Section 4.5.4 and Section 4.5.6, through Tables 4.1, 4.2, 4.3, 4.4, and 4.5, which analysed the variance and quality of the samples produced with the proposed algorithm and adaptive sampling strategy.

The main contribution of **Chapter 5** consists of the efficiency cost models for traditional CF. Prior to the development of these models, we first analysed how the processing time and memory overhead scale with respect to samples, drawn from URM, of different sizes as part of **C3.1** in Figures 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6. Then, we conceptualised the baseline approach for estimating the efficiency of CF models using solely characteristics of the input and state-of-the-art regressors (i.e., Black-Box approach) as part of **C3.2**. In **C3.3**, we proposed a novel methodology for predicting the processing time and memory overhead using complexity analysis and curve fitting primitives (i.e., White-Box approach), evaluated in Section 5.4. In addition, in **C4.1**, we conducted an empirical study which investigates the best state-of-the-art regressors for estimating the efficiency of CF algorithms, and in **C4.2**, we provided an experimental evaluation which assesses the accuracy of the Black-Box approach for predicting resource consumption for CF models, as shown in Section 5.4.1 and Section 5.4.3. Furthermore, as part of **C4.3**, we provided a comparative assessment across Black-Box based regressors for predicting the effectiveness (i.e., quality of recommendations) of traditional CF, as illustrated in Figures 5.7, 5.8, and 5.9. **C5.1** discussed the theoretical analysis of the time and space complexity, summarised in Tables 5.1 and 5.2 for traditional CF models using their implementations, showcased in the popular Surprise framework [37], throughout Section 5.2. Then, as part of **C5.2**, we developed an efficiency cost model based on time and space complexity equations (i.e., White-Box approach) for estimating processing time and memory overhead for traditional CF algorithms. Finally, **C5.3** covered an experimental study which compares the White-Box methodology against the Black-Box approach, and efficiency cost models based on complexity equations from the relevant literature, presented in 5.1, 5.2, and 5.3, as illustrated in Section 5.4.

Chapter 6 contributed a framework for estimating the processing time and GPU memory during training for DL CF. Specifically, in **C6.1**, we provided a theoretical analysis of the time and space complexity for neural CF models, using their implementations showcased in the Cornac framework [38], as presented in Section 6.2 and summarised in Tables 6.1 and 6.2. In addition, **C6.2** depicted an empirical study, which investigates how the processing time and GPU memory vary with respect to the size of the input when one or multiple GPU cards are used for training the DL CF models, as seen in Section 6.4.1 and Section 6.4.2, as well as Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, and 6.8. As part of **C6.3**, we extended the proposed White-Box and Black-Box methodologies to estimate the training cost of DL CF. Then, throughout Section 6.4, **C6.4** focused on providing a comparative assessment between the White-Box and Black-Box methodologies, and evaluated the accuracy of each method for predicting the resource consumption of neural CF algorithms. Furthermore, **C7.1** illustrated an initial investigation into how the effectiveness of a neural CF model varies across inputs/samples of different sizes when the model is trained on one and multiple GPUs with the empirical findings depicted in Figures 6.9, 6.10, 6.11, and 6.12. Lastly, as part of **C7.2**, we provided an experimental evaluation of various off-the-shelf standard practice predictive models (i.e., Black-Box approach) used to estimate the quality of recommendations produced by DL CF algorithms on a complete dataset given their effectiveness on a set of sub-samples, as seen in Figures 6.13, 6.14, 6.15, and 6.16. Both C7.1 and C7.2 are presented in Section 6.4.

7.4 Limitations and Future Research Directions

This section discusses the limitations of this thesis, considering areas of the design space that could be further explored, and formulates a number of potential avenues for future work.

This thesis introduced the problem of efficiency estimation and provided resource consumption cost models for several traditional and neural families of CF algorithms that have been trained on explicit data (i.e., ratings). To cover an even wider area of the CF design space, we propose to extend our framework, methodology, and sampling scheme to estimate the efficiency and effectiveness of implicit feedback CF, investigating well-known representatives [3], such as Bayesian Personalized Ranking (BPR) [169], Logistic Matrix Factorization (LMF) [170], Implicit Matrix Factorisation (IMF) [49], and others. In addition, so far, we have only examined CF that were trained on numerical URMs. Thus, another extension of this work would be to include other types of input for CF, such as textual data. As part of future work of this thesis, we propose predicting the resource consumption of CF, such as processing time, memory usage, network transfer load, etc., in distributed training CF frameworks. To further augment the research conducted in the Ph.D., we could build different types of regression models depending on the general form of the time and space complexity equations of CF. Lastly, the efficiency prediction framework presented in this thesis could be deployed as a packaged Dockerised tool for estimating resource consumption of CF and potentially other types of jobs in clusters and cloud platforms.

The proposed White- and Black-Box methodologies were conceptualised, developed, and probed with different categories of CF models. However, we believe they could be applied and used to quantify the resource requirements for other classes of algorithms and use cases, such as predicting the execution time of various query processing algorithms [171]. Knowing the execution time is particularly useful as this information can facilitate latency-aware query scheduling [172], or bypass “rogue” queries that have a very expensive execution time [173], as well as it can allow Databases as a Service (DBaaS) providers to scale their systems and acquire the necessary hardware [174]. Existing works tried to address this problem by treating the database as a black box and tried to learn the queries processing/execution time as a function of the input [175, 176]. Nevertheless, we believe a similar approach to the White-Box methodology, employed for predicting the processing time of CF, could be built for estimating query execution time based on the computational complexity of query processing algorithms. Furthermore, our methodology could be also utilised to estimate the execution time of a plan or parts of a plan in a single-node database, as part of the query optimiser's cost estimations. Recent works [177, 178] tackled this problem by deploying DL-based solutions to learn the correlations between different columns/tables. However, there is an inherent trade-off between the accuracy of the predictor and its training cost, and thus, using a more complex model could often be too expensive (i.e., take too long to train) for the optimiser's pipeline [179, 180]. Consequently, simpler and faster regressors, such as the linear ones, could be an alternative solution to the query optimiser's cost prediction problem.

As part of this thesis, we introduced a sampling strategy that allows one to sample the URM and create smaller partitions of the input, while maintaining its structural and complexity properties. In future, this sampling methodology could be extended to also upsample the URM, and allow the users of our framework to determine how the resource consumption of the CF would vary if the size of the input grows. The upsampling component would be based on a workload generator for CF, which could produce larger (or smaller for downsampling) URMs, which emulate the complexity and structural characteristics of a given real dataset. A good starting point for exploring this research direction can be found in the CF simulation literature, which supports standard practice evaluation methodologies and encourages reproducibility [181]. Moreover, capturing the relative performance of CF models on both real and synthesised collections offers a better understanding of (a) the behaviour of the algorithms, and (b) how the characteristics of the input affect the recommendation process [182]. Also, being able to manage the properties of simulated collections leads to numerous experimental setups, which can not only increase the explainability of the recommendations, but also highlight which CF models are best suited for different scenarios (i.e., recommendation tasks) and/or types of URMs [25, 26].

7.5 Conclusion

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 has overlooked this pressing problem. This thesis addresses the problem of predicting processing time, memory overhead, and GPU utilisation for a number of popular traditional and neural CF algorithms using a simple yet highly effective methodology. Our solution is built using a fit-for-purpose sampling strategy, as well as a fast and accurate linear regression scheme over time and space complexity equations drawn from the 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 approach (White-Box) without training on the entire collection, as shown in Table 5.4. The proposed sampling strategy also allows the prediction of both efficiency and effectiveness while paying the cost of sampling the based data only once, as otherwise, alternative solutions require multiple/different sampling rounds, which translate into higher costs. Despite its simplicity, our framework and resource cost models for CF representatives manage to considerably outperform in accuracy even the best performing off-the-shelf state-of-the-art Black-Box regressor. In addition, our findings, summarised in Table 5.3, indicated that the White-Box approach is also faster and cheaper than all contenders, utilising only a fraction of the resources used by them. To the best of our knowledge, the research presented in this thesis is one of the first works that addresses the challenge of predicting the efficiency of both traditional and DL CF in a well-rounded manner, which investigated and proposed various methods to estimate CF's resource consumption, and paved the way towards a systematic exploration of the efficiency-effectiveness trade-offs inherent in modern recommendation systems.

Bibliography

[1] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005, pp. 471–475.

[2] S. K. Lam, A. LaPitz, G. Karypis, J. Riedl *et al.*, "Towards a scalable knn cf algorithm: Exploring effective applications of clustering," in *International Workshop on Knowledge Discovery on the Web*, 2006, pp. 147–166.

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

[4] H. Gu, Y. Wang, S. Hong, and G. Gui, "Blind channel identification aided generalized automatic modulation recognition based on deep learning," *IEEE Access*, vol. 7, pp. 110 722–110 729, 2019.

[5] G. Zaccone, *Getting started with TensorFlow.* Packt Publishing Birmingham, 2016.

[6] Y. Yang, K. Zheng, C. Wu, and Y. Yang, "Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network," *Sensors*, vol. 19, no. 11, p. 2528, 2019.

[7] "Bilateral Variational Autoencoder (BiVAE)," https://github.com/microsoft/recommenders/blob/main/examples/02_model_collaborative_filtering/cornac_bivae_deep_dive.ipynb, Last Accessed: 2021-10-27. [Online].

[8] Q.-T. Truong, A. Salah, and H. W. Lauw, "Bilateral variational autoencoder for collaborative filtering," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 292–300.

[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[10] D. Kim, C. Park, J. Oh, S. Lee, and H. Yu, "Convolutional matrix factorization for document context-aware recommendation," in *Proceedings of the 10th ACM conference on recommender systems*, 2016, pp. 233–240.

[11] G.-J. Qi and J. Luo, “Small data challenges in big data era: A survey of recent progress on unsupervised and semi-supervised methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[12] B. Smith and G. Linden, “Two decades of recommender systems at amazon. com,” *Ieee internet computing*, vol. 21, no. 3, pp. 12–18, 2017.

[13] M. Beladev, L. Rokach, and B. Shapira, “Recommender systems for product bundling,” *Knowledge-Based Systems*, vol. 111, pp. 193–206, 2016.

[14] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, “Recommender systems handbook,” Boston, Massachusetts, 2010.

[15] T.-P. Liang, H.-J. Lai, and Y.-C. Ku, “Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings,” *Journal of Management Information Systems*, vol. 23, no. 3, pp. 45–70, 2006.

[16] L.-L. Wu, Y.-J. Joung, and J. Lee, “Recommendation systems and consumer satisfaction online: moderating effects of consumer product awareness,” Wailea, Maui, HI, United States, pp. 2753–2762, 2013.

[17] M. Yeomans, A. Shah, S. Mullainathan, and J. Kleinberg, “Making sense of recommendations,” *Journal of Behavioral Decision Making*, vol. 32, no. 4, pp. 403–414, 2019.

[18] L. Rook, A. Sabic, and M. Zanker, “Engagement in proactive recommendations,” *Journal of Intelligent Information Systems*, vol. 54, no. 1, pp. 79–100, 2020.

[19] I. Paun, “Efficiency-effectiveness trade-offs in recommendation systems,” in *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys ’20)*. Rio de Janeiro, Brazil: ACM, 2020, pp. 770–775.

[20] I. Paun, Y. Moshfeghi, and N. Ntarmos, “Are we there yet? estimating training time for recommendation systems,” in *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys)*. Edinburgh, United Kingdom: ACM, 2021, pp. 1–9.

[21] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” *CoRR*, vol. abs/1906.02243, pp. 1–6, 2019. [Online]. Available: http://arxiv.org/abs/1906.02243

[22] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang, “An empirical study on program failures of deep learning jobs,” in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1159–1170.

[23] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, “A comprehensive study on deep learning bug characteristics,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 510–520.

[24] W. Huang, S. Kataria, C. Caragea, P. Mitra, C. L. Giles, and L. Rokach, “Recommending citations: translating papers into references,” in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 1910–1914.

[25] Y. Deldjoo, T. Di Noia, E. Di Sciascio, and F. A. Merra, “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*. Xi’an, China: ACM, 2020, pp. 951–960.

[26] G. Adomavicius and J. Zhang, “Impact of data characteristics on recommender systems performance,” *ACM Transactions on Management Information Systems (TMIS)*, vol. 3, no. 1, pp. 1–17, 2012.

[27] R. Cañamares and P. Castells, “On target item sampling in offline recommender system evaluation,” Rio de Janeiro, Brazil, pp. 259–268, 2020.

[28] A. L’heureux, K. Grolinger, H. F. Elyamany, and M. A. Capretz, “Machine learning with big data: Challenges and approaches,” *Ieee Access*, vol. 5, pp. 7776–7797, 2017.

[29] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang, and K. Li, “Energy-efficient resource utilization for heterogeneous embedded computing systems,” *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1518–1531, 2017.

[30] D.-M. Bui, Y. Yoon, E.-N. Huh, S. Jun, and S. Lee, “Energy efficiency for cloud computing system based on predictive optimization,” *Journal of Parallel and Distributed Computing*, vol. 102, pp. 103–114, 2017.

[31] D. Dice and A. Kogan, “Optimizing inference performance of transformers on cpus,” in *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys)*. Edinburgh, United Kingdom: ACM, 2021, pp. 1–8.

[32] G. Yeung, D. Borowiec, A. Friday, R. Harper, and P. Garraghan, “Towards GPU utilization prediction for cloud deep learning,” in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020, pp. 1–9.

[33] E. Bisong, “Google colaboratory,” in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, 2019, pp. 59–64.

[34] S. Mathew and J. Varia, “Overview of amazon web services,” *Amazon Whitepapers*, vol. 105, pp. 1–22, 2014.

[35] V. S. Vairale and S. Shukla, “Recommendation of food items for thyroid patients using content-based knn method,” in *Data Science and Security*, 2021, pp. 71–77.

[36] R. Cañamares and P. Castells, “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*. Michigan, United States: ACM, 2018, pp. 415–424.

[37] N. Hug, “Surprise: A python library for recommender systems,” *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020. [Online]. Available: https://doi.org/10.21105/joss.02174

[38] A. Salah, Q.-T. Truong, and H. W. Lauw, “Cornac: A comparative framework for multimodal recommender systems,” *Journal of Machine Learning Research*, vol. 21, no. 95, pp. 1–5, 2020.

[39] S. Loeb and E. Panagos, “Information filtering and personalization: Context, serendipity and group profile effects,” in *2011 IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, 2011, pp. 393–398.

[40] N. J. Belkin and W. B. Croft, “Information filtering and information retrieval: Two sides of the same coin?” *Communications of the ACM*, vol. 35, no. 12, pp. 29–38, 1992.

[41] U. Hanani, B. Shapira, and P. Shoval, “Information filtering: Overview of issues, research and systems,” *User modeling and user-adapted interaction*, vol. 11, no. 3, pp. 203–259, 2001.

[42] F. O. Isinkaye, Y. Folajimi, and B. A. Ojokoh, “Recommendation systems: Principles, methods and evaluation,” *Egyptian informatics journal*, vol. 16, no. 3, pp. 261–273, 2015.

[43] J. Bennett, S. Lanning *et al.*, “The netflix prize,” in *Proceedings of KDD cup and workshop*, vol. 2007. San Jose, California: Citeseer, 2007, p. 35.

[44] D. Margaris, C. Vassilakis, and D. Spiliotopoulos, “What makes a review a reliable rating in recommender systems?” *Information Processing & Management*, vol. 57, no. 6, p. 102304, 2020.

[45] F. Ricci, L. Rokach, and B. Shapira, “Recommender systems: introduction and challenges,” in *Recommender systems handbook*. Springer, 2015, pp. 1–34.

[46] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[47] G. Jawaheer, P. Weller, and P. Kostkova, “Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback,” *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 4, no. 2, pp. 1–26, 2014.

[48] G. Schröder, M. Thiele, and W. Lehner, “Setting goals and choosing metrics for recommender system evaluations,” in *UCERSTI2 workshop at the 5th ACM conference on recommender systems, Chicago, USA*, vol. 23, 2011, p. 53.

[49] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[50] P. B. Thorat, R. Goudar, and S. Barve, “Survey on collaborative filtering, content-based filtering and hybrid recommendation system,” *International Journal of Computer Applications*, vol. 110, no. 4, pp. 31–36, 2015.

[51] R. Burke, “Knowledge-based recommender systems,” *Encyclopedia of library and information systems*, vol. 69, no. Supplement 32, pp. 175–186, 2000.

[52] R. A. Berk, “An introduction to ensemble methods for data analysis,” *Sociological methods & research*, vol. 34, no. 3, pp. 263–295, 2006.

[53] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, “Deep learning applications and challenges in big data analytics,” *Journal of big data*, vol. 2, no. 1, pp. 1–21, 2015.

[54] M. V. Valueva, N. Nagornov, P. A. Lyakhov, G. V. Valuev, and N. I. Chervyakov, “Application of the residue number system to reduce hardware costs of the convolutional neural network implementation,” *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, 2020.

[55] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda, “Subject independent facial expression recognition with robust face detection using a convolutional neural network,” *Neural Networks*, vol. 16, no. 5-6, pp. 555–559, 2003.

[56] C. C. Aggarwal *et al.*, “Neural networks and deep learning,” *Springer*, vol. 10, pp. 978–3, 2018.

[57] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[58] A. Bellogín and P. Castells, “A performance prediction approach to enhance collaborative filtering performance,” in *European Conference on Information Retrieval*. Berlin, Heidelberg: Springer, 2010, pp. 382–393.

[59] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, “Controlled experiments on the web: survey and practical guide,” *Data mining and knowledge discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[60] R. Kohavi and R. Longbotham, “Online controlled experiments and a/b testing.” *Encyclopedia of machine learning and data mining*, vol. 7, no. 8, pp. 922–929, 2017.

[61] E. Kharitonov, “Using interaction data for improving the offline and online evaluation of search engines,” Ph.D. dissertation, University of Glasgow, 2016.

[62] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[63] A. Gunawardana and G. Shani, “Evaluating recommender systems,” in *Recommender systems handbook*, 2015, pp. 265–308.

[64] R. R. Sinha, K. Swearingen *et al.*, “Comparing recommendations made by online systems and friends,” *DELOS*, vol. 106, 2001.

[65] M. Ludewig, N. Mauro, S. Latifi, and D. Jannach, “Performance comparison of neural and non-neural approaches to session-based recommendation,” in *Proceedings of the 13th ACM conference on recommender systems*, 2019, pp. 462–466.

[66] Y. Hu, Y. Koren, and C. Volinsky, “Collaborative filtering for implicit feedback datasets,” in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 263–272.

[67] P. Castells, N. J. Hurley, and S. Vargas, “Novelty and diversity in recommender systems,” in *Recommender systems handbook*, 2015, pp. 881–918.

[68] M. Ge, C. Delgado-Battenfeld, and D. Jannach, “Beyond accuracy: evaluating recommender systems by coverage and serendipity,” in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 257–260.

[69] K. Järvelin and J. Kekäläinen, “Cumulated gain-based evaluation of ir techniques,” *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[70] R. E. Bryant, O. David Richard, and O. David Richard, “Computer systems: a programmer’s perspective,” 2003.

[71] N. Rozanski and E. Woods, “Software systems architecture: working with stakeholders using viewpoints and perspectives,” 2012.

[72] Y. Koren, “Factor in the neighbors: Scalable and accurate collaborative filtering,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 1, pp. 1–24, 2010.

[73] G. Takács and D. Tikk, “Alternating least squares for personalized ranking,” in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 83–90.

[74] P. Moulin and V. V. Veeravalli, “Maximum likelihood estimation,” in *Statistical Inference for Engineers and Data Scientists*, Cambridge, 2018, p. 319–357.

[75] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, “An algorithmic framework for performing collaborative filtering,” in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, pp. 230–237.

[76] F. Cacheda, V. Carneiro, D. Fernández, and V. Formoso, “Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems,” *ACM Trans. Web*, vol. 5, no. 1, 2011.

[77] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, “An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1273–1284, 2014.

[78] T. George and S. Merugu, “A scalable collaborative filtering framework based on co-clustering,” in *Fifth IEEE International Conference on Data Mining (ICDM’05)*. Houston, Texas: IEEE, 2005, pp. 4–pp.

[79] L. Wu, X. He, X. Wang, K. Zhang, and M. Wang, “A survey on neural recommendation: From collaborative filtering to content and context enriched recommendation,” *CoRR*, vol. abs/2104.13030, 2021, Last Accessed: 2021-08-10. [Online]. Available: https://arxiv.org/abs/2104.13030

[80] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, “Variational autoencoders for collaborative filtering,” in *Proceedings of the 2018 world wide web conference*, 2018, pp. 689–698.

[81] C. M. Bishop, “Pattern recognition and machine learning (information science and statistics),” Berlin, Heidelberg, 2006.

[82] J. Rice, *Mathematical Statistics and Data Analysis*, ser. Duxbury advanced series. Duxbury Press, 1995.

[83] L. Zhou, C. Cai, Y. Gao, S. Su, and J. Wu, "Variational autoencoder for low bit-rate image compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2617–2620.

[84] Z.-S. Liu, W.-C. Siu, and L.-W. Wang, "Variational autoencoder for reference based image super-resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 516–525.

[85] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[86] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[87] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014, Last Accessed: 2021-10-27. [Online].

[88] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2008, pp. 1257–1264.

[89] R. Anand and J. Beel, "Auto-surprise: An automated recommender-system (autorecsys) library with tree of parzens estimator (tpe) optimization," in *Fourteenth ACM Conference on Recommender Systems*, 2020, pp. 585–587.

[90] Y. Sun, J. Pan, A. Zhang, and A. Flores, "Fm2: Field-matrixed factorization machines for recommender systems," in *Proceedings of the Web Conference 2021*, 2021, pp. 2828–2837.

[91] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms," Cambridge, Massachusetts, 2009.

[92] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 251–258.

[93] J. Beel, S. Langer, M. Genzmehr, B. Gipp, C. Breitinger, and A. Nürnberger, "Research paper recommender system evaluation: a quantitative literature survey," in *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, 2013, pp. 15–22.

[94] D. Kowald, E. Lacic, and C. Trattner, “Tagrec: Towards a standardized tag recommender benchmarking framework,” in *Proceedings of the 25th ACM conference on Hypertext and social media*, 2014, pp. 305–307.

[95] D. Kowald and E. Lex, “Evaluating tag recommender algorithms in real-world folksonomies: A comparative study,” in *Proceedings of the 9th ACM Conference on Recommender Systems*, 2015, pp. 265–268.

[96] A. Bellogín, P. Castells, and I. Cantador, “Statistical biases in information retrieval metrics for recommender systems,” *Information Retrieval Journal*, vol. 20, no. 6, pp. 606–634, 2017.

[97] D. Jannach, L. Lerche, I. Kamehkhosh, and M. Jugovac, “What recommenders recommend: an analysis of recommendation biases and possible countermeasures,” *User Modeling and User-Adapted Interaction*, vol. 25, no. 5, pp. 427–491, 2015.

[98] H. Steck, “Item popularity and recommendation accuracy,” in *Proceedings of the fifth ACM conference on Recommender systems*, 2011, pp. 125–132.

[99] U. Finkler and K. Mehlhorn, “Runtime prediction of real programs on real machines,” *Max-Planck-Institut für Informatik*, 1996.

[100] S. F. Goldsmith, A. S. Aiken, and D. S. Wilkerson, “Measuring empirical computational complexity,” in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 395–404.

[101] A. A. Sidnev, “Runtime prediction on new architectures,” in *Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia*, 2014, pp. 1–7.

[102] E. A. Brewer, “High-level optimization via automated statistical modeling,” *ACM SIGPLAN Notices*, vol. 30, no. 8, pp. 80–91, 1995.

[103] E. Fink, “How to solve it automatically: Selection among problem solving methods,” in *AIPS*, 1998, pp. 128–136.

[104] A. E. Howe, E. Dahlman, C. Hansen, M. Scheetz, and A. Von Mayrhauser, “Exploiting competitive planner performance,” in *European Conference on Planning*, 1999, pp. 62–72.

[105] M. Roberts, A. Howe, and L. Flom, “Learned models of performance for many planners,” in *ICAPS 2007 Workshop AI Planning and Learning*, 2007, pp. 36–40.

[106] E. Ridge and D. Kudenko, “Tuning the performance of the MMAS heuristic,” in *International Workshop on Engineering Stochastic Local Search Algorithms*, 2007, pp. 46–60.

[107] T. Bartz-Beielstein and S. Markon, “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, 2004, pp. 1111–1118.

[108] K. Leyton-Brown, E. Nudelman, and Y. Shoham, “Learning the empirical hardness of optimization problems: The case of combinatorial auctions,” in *International Conference on Principles and Practice of Constraint Programming*, 2002, pp. 556–572.

[109] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, “Automatic database management system tuning through large-scale machine learning,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1009–1024.

[110] J. Li, A. C. König, V. Narasayya, and S. Chaudhuri, “Robust estimation of resource consumption for sql queries using statistical techniques,” *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1–12, 2012.

[111] H. Lan, Z. Bao, and Y. Peng, “A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration,” *Data Science and Engineering*, vol. 6, no. 1, pp. 1–16, 2021.

[112] C. Chen, P. Zhang, H. Zhang, J. Dai, Y. Yi, H. Zhang, and Y. Zhang, “Deep learning on computational-resource-limited platforms: a survey,” *Mobile Information Systems*, vol. 2020, 2020.

[113] P. J. Haas and A. N. Swami, “Sequential sampling procedures for query size estimation,” in *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, 1992, pp. 341–350.

[114] W.-C. Hou, G. Ozsoyoglu, and E. Dogdu, “Error-constrained count query evaluation in relational databases,” *ACM SIGMOD Record*, vol. 20, no. 2, pp. 278–287, 1991.

[115] R. J. Lipton and J. F. Naughton, “Query size estimation by adaptive sampling,” *Journal of Computer and System Sciences*, vol. 51, no. 1, pp. 18–25, 1995.

[116] P. B. Gibbons, Y. Matias, and V. Poosala, “Fast incremental maintenance of approximate histograms,” *ACM Transactions on Database Systems (TODS)*, vol. 27, no. 3, pp. 261–298, 2002.

[117] S. Chaudhuri, R. Motwani, and V. Narasayya, “Random sampling for histogram construction: How much is enough?” *ACM SIGMOD Record*, vol. 27, no. 2, pp. 436–447, 1998.

[118] H. W. Austin, “Sample size: How much is enough?” *Quality and Quantity*, vol. 17, no. 3, pp. 239–245, 1983.

[119] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, “Random sampling techniques for space efficient online computation of order statistics of large datasets,” *ACM SIGMOD Record*, vol. 28, no. 2, pp. 251–262, 1999.

[120] K. P. Murphy, “Machine learning: a probabilistic perspective,” London, UK, 2012.

[121] Z. Tan, “Monte carlo integration with acceptance-rejection,” *Journal of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 735–752, 2006.

[122] B. S. Everitt and A. Skrondal, *The Cambridge dictionary of statistics*. Cambridge University Press, 2010.

[123] A. Bjorck, *Numerical methods for least squares problems*. Philadelphia, USA: Siam, 1996.

[124] W. Q. Meeker, G. J. Hahn, and L. A. Escobar, *Statistical intervals: a guide for practitioners and researchers*. Hoboken, NJ, United States: John Wiley & Sons, 2017, vol. 541.

[125] N. R. Draper and H. Smith, *Applied regression analysis*. Hoboken, NJ, United States: John Wiley & Sons, 1998, vol. 326.

[126] D. L. Shrestha and D. P. Solomatine, “Machine learning approaches for estimation of prediction interval for the model output,” *Neural Networks*, vol. 19, no. 2, pp. 225–235, 2006.

[127] B. P. Carlin and T. A. Louis, *Bayesian methods for data analysis*. Florida, United States: CRC Press, 2008.

[128] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian data analysis*. Florida, United States: CRC press, 2013.

[129] A. Shapiro, “Monte carlo sampling methods,” *Handbooks in operations research and management science*, vol. 10, pp. 353–425, 2003.

[130] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, *Handbook of markov chain monte carlo*. Florida, United States: CRC press, 2011.

[131] R. C. Smith, *Uncertainty quantification: theory, implementation, and applications*. Philadelphia, United States: Siam, 2013, vol. 12.

[132] C. Gini, "Measurement of inequality of incomes," *The economic journal*, vol. 31, no. 121, pp. 124–126, 1921.

[133] E. LeDell and S. Poirier, "H2O AutoML: Scalable automatic machine learning," in *7th ICML Workshop on Automated Machine Learning (AutoML)*. Vienna, Austria: ICML, July 2020, pp. 1–16. [Online]. Available: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[134] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. New York: Springer series in statistics, 2001, vol. 1 (10).

[135] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (TiiS)*, vol. 5, no. 4, pp. 1–19, 2015.

[136] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[137] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[138] S. Pousty and K. Miller, *Getting Started with OpenShift: A Guide for Impatient Beginners*. " O'Reilly Media, Inc", 2014.

[139] B. Burns, J. Beda, and K. Hightower, *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, 2019.

[140] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[141] O. Ben-Kiki, C. Evans, and B. Ingerson, "Yaml ain't markup language (yaml™) version 1.1," *Working Draft 2008-05*, vol. 11, 2009.

[142] Z. Zajac, "Goodbooks-10k: a new dataset for book recommendations," http://fastml.com/goodbooks-10k, 2017, Last Accessed: 2021-02-12. [Online].

[143] "Python 3.7," https://www.python.org/downloads/release/python-370/, Last Accessed: 2022-04-27. [Online].

[144] "Jupyter notebook," https://jupyter.org/, Last Accessed: 2022-04-27. [Online].

[145] "Pandas - python data analysis library," https://pandas.pydata.org/, Last Accessed: 2022-04-27. [Online].

[146] "Scikit-learn - machine learning in python," https://scikit-learn.org/stable/index.html, Last Accessed: 2022-04-29. [Online].

[147] "Pymc3 - probabilistic programming in python," https://docs.pymc.io/en/v3/, Last Accessed: 2022-04-29. [Online].

[148] B. Li, Q. Yang, and X. Xue, "Can movies and books collaborate? cross-domain collaborative filtering for sparsity reduction," in *Twenty-First international joint conference on artificial intelligence*, 2009.

[149] N. Mishra, S. Chaturvedi, V. Mishra, R. Srivastava, and P. Bargah, "Solving sparsity problem in rating-based movie recommendation system," in *Computational Intelligence in Data Mining*. Springer, 2017, pp. 111–117.

[150] H. Bhasin, *Algorithms: design and analysis*. Oxford University Press, 2015.

[151] C. Tosh and S. Dasgupta, "The relative complexity of maximum likelihood estimation, map estimation, and sampling," in *Conference on Learning Theory*, 2019, pp. 2993–3035.

[152] Z. Wang, Y. Liu, and P. Ma, "A cuda-enabled parallel implementation of collaborative filtering," *Procedia Computer Science*, vol. 30, pp. 66–74, 2014.

[153] Scikit-learn.org, "Stochastic gradient descent 0.23.0 documentation," https://scikit-learn.org/stable/modules/sgd.html#complexity, 2020, Last Accessed: 2021-09-15. [Online].

[154] A. K. Cline and I. S. Dhillon, "Computation of the singular value decomposition," in *Handbook of Linear Algebra*, L. Hogben, Ed., 2007, ch. 45, pp. 45–1–45–13.

[155] Z. Sun, N. Luo, and W. Kuang, "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, 2011, pp. 1826–1830.

[156] L. Bulteau, V. Froese, S. Hartung, and R. Niedermeier, "Co-clustering under the maximum norm," *Algorithms*, vol. 9, no. 1, p. 17, 2016.

[157] S. Arora and B. Barak, "Computational complexity: a modern approach," Cambridge, UK, 2009.

[158] H.-R. Zhang, F. Min, Z.-H. Zhang, and S. Wang, “Efficient collaborative filtering recommendations with multi-channel feature vectors,” *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 5, pp. 1165–1172, 2019.

[159] Z. Batmaz, A. Yurekli, A. Bilge, and C. Kaleli, “A review on deep learning for recommender systems: challenges and remedies,” *Artificial Intelligence Review*, vol. 52, no. 1, pp. 1–37, 2019.

[160] E. M. Malta, S. Avila, and E. Borin, “Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 21–29.

[161] V. Leis and M. Kuschewski, “Towards cost-optimal query processing in the cloud,” *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1606–1612, 2021.

[162] Y. Ukidave, X. Li, and D. Kaeli, “Mystic: Predictive scheduling for gpu based cloud servers using machine learning,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 353–362.

[163] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser, “Many-core vs. many-thread machines: Stay away from the valley,” *IEEE Computer Architecture Letters*, vol. 8, no. 1, pp. 25–28, 2009.

[164] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, “Power aware computing on gpus,” in *2012 Symposium on Application Accelerators in High Performance Computing*. IEEE, 2012, pp. 64–73.

[165] V. M. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula, J. Ralph, J. Nelson, P. Mucci, T. Mohan, and S. Moore, “Papi 5: Measuring power, energy, and the cloud,” in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 124–125.

[166] Y. Arafa, A. ElWazir, A. ElKanishy, Y. Aly, A. Elsayed, A.-H. Badawy, G. Chennupati, S. Eidenbenz, and N. Santhi, “Verified instruction-level energy consumption measurement for nvidia gpus,” in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 60–70.

[167] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, 2019.

[168] R. Farber, *CUDA application design and development*. Elsevier, 2011.

[169] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," *arXiv preprint arXiv:1205.2618*, 2012, Last Accessed: 2021-05-20. [Online].

[170] C. C. Johnson, "Logistic matrix factorization for implicit feedback data," *Advances in Neural Information Processing Systems*, vol. 27, no. 78, pp. 1–9, 2014.

[171] W. Kim, D. S. Reiner, and D. Batory, *Query processing in database systems*. Springer Science & Business Media, 2012.

[172] Y. Chi, H. J. Moon, and H. Hacigümüş, "iCBS: incremental cost-based scheduling under piecewise linear SLAs," *Proceedings of the VLDB Endowment*, vol. 4, no. 9, pp. 563–574, 2011.

[173] C. Mishra and N. Koudas, "The design of a query monitoring system," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 1, pp. 1–51, 2009.

[174] T. J. Wasserman, P. Martin, D. B. Skillicorn, and H. Rizvi, "Developing a characterization of business intelligence workloads for sizing new database systems," in *Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP*, 2004, pp. 7–13.

[175] M. Akdere, U. Cetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based query performance modeling and prediction," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 390–401.

[176] S. Tozer, T. Brecht, and A. Aboulnaga, "Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010, pp. 397–408.

[177] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," *arXiv preprint arXiv:1809.00677*, 2018, Last Accessed: 2021-07-08. [Online].

[178] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi, "Learning state representations for query optimization with deep reinforcement learning," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018, pp. 1–4.

[179] X. Zhou, C. Chai, G. Li, and J. Sun, "Database meets artificial intelligence: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 3, pp. 1096–1116, 2020.

[180] G. Li, X. Zhou, and L. Cao, "Ai meets database: Ai4db and db4ai," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2859–2866.

[181] A. Bellogín and A. Said, "Improving accountability in recommender systems research through reproducibility," *arXiv preprint arXiv:2102.00482*, 2021, Last Accessed: 2021-11-02. [Online].

[182] M. Slokom, M. Larson, and A. Hanjalic, "Data masking for recommender systems: Prediction performance and rating hiding," in *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019.