FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

μ-cf2vec: Representation Learning for Personalized Algorithm Selection in Recommender Systems

Tomas Trigueiros de Sousa Pereira

FOR JURY EVALUATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Carlos Soares, PhD

Co-Supervisor: Tiago Cunha, PhD

February 8, 2021

Abstract

Collaborative Filtering (CF) has become the standard approach to solve recommendation systems problems. There are multiple CF algorithms, each one of them with its own biases. It is the Machine Learning practitioner that has to choose the best algorithm for each task beforehand. However, from the *No Free Lunches* theorem, we can extrapolate that it is very unlikely that one single algorithm will be the best for all tasks. In Recommender Systems, different algorithms have different performance for different users within the same dataset. Meta Learning has been used to choose the best algorithm for a given problem. MtL can lean from prior experience in such a way that can be used in other tasks. This dissertation presents a new meta-learning based framework named μ-cf2vec to select the best algorithm for each user. Meta Learning is usually applied to select algorithms for a whole dataset. Adapting it to select the to the algorithm for a single user in a RS involves several challenges. The most important is the design of the metafeatures which, in typical meta learning, characterize datasets while here, they must characterize a single user. We propose using Representation Learning techniques to extract the metafeatures. Representation Learning tries to extract representations that can be reused in other learning tasks. We are going to compare two different RL techniques to evaluate which one can be more useful to solve this task. In the meta level, the meta learning model will use the metafeatures to extract knowledge that will be used to predict the best algorithm for each user. We evaluated an implementation of this framework using MovieLens 20M dataset. Our implementation achieved consistent gains in the meta level, however, in the base level we only achieved marginal gains.

Resumo

Collaborative Filtering tornou-se a técnica standard usada para resolver problemas de sistemas de recomendação. Existem várias implementações desta técnica, cada uma com o seu viés. Tem de ser o *Data Scientist* a escolher manualmente qual o melhor algoritmo a utilizar. No entanto, é improvável que apenas um algoritmo tenha o melhor desempenho para todos os problemas. Esta dissertação apresenta uma *framework* baseada em técnicas de *Meta Learning* chamada μ-cf2vec. A *framework* é dividida em dois níveis: nível base e nível meta. No nível base, μ-cf2vec avalia o desempenho de um grupo de *base learners* que irá ser usado como *metatarget*. Este *metatarget* vai ser utilizado em conjunto com *metafeatures* para criar um *metadataset*. As *metafeatures* vão ser geradas usando técnicas de *Representation Learning*. No nível meta o modelo irá utilizar este *metadataset* para escolher o melhor algoritmo para cada utilizador. Iremos avaliar uma implementação desta framework utilizando o *dataset MovieLens 20M*. Nesta implementação obtivemos ganhos constantes no nível meta e ganhos marginais no nível base.

Publications

This work has produced a paper which was submitted to: *NeuRec 2020: ICDM Workshop on Recommender Systems 2020*.

Acknowledgements

I would like to thank my supervisor Professor Carlos Soares for accepting my proposal and for the guidance during the work on this dissertation.

I would also like to thank Tiago Cunha for helping me create this proposal and all the guidance through this work.

Finally, I want to thank my father for pushing me.

Tomas Sousa Pereira

This work is partly funded by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project CONTEXTWA (FCT PTDC/EEI-SCR/6945/2014 - POCI-01-0145-FEDER-016883).

"Work hard.
Play later"
Gil Dias

Contents

List of Figures

List of Tables

Abbreviations

ALS	Alternating Least Squares
BPR	Bayesian Personalized Ranking
CDAE	Collaborative Denoising Auto-Encoders
CF	Collaborative Filtering
DCG	Discounted Cumulative Gain
IDCG	Ideal Discounted Cumulative Gain
LMF	Logistic Matrix Factorization
MLP	Multilayer Perceptron
MtL	Meta Learning
NDCG	Normalized Discounted Cumulative Gain
PCA	Principal Component Analysis
RBM	Restricted Boltzmann Machines
RL	Representation Learning
RS	Recommender Systems
RQ	Research Question
SVM	Support Vector Machines
TF-IDF	Term Frequency times Inverse Document Frequency
VAE	Variational Autoencoders

Chapter 1

Introduction

Over the last few years, recommender systems (RS) have been used in many different applications in our life, such as recommending products that a customer will most likely buy, suggesting movies [2], videos [3] to watch and many others [4]. Various approaches for recommender systems have been developed, which are used in many disciplines: machine learning, text mining, artificial intelligence, network analysis, databases, human computer interaction, and many more [5]. However, despite many successes, the predictive performance of RS is still not satisfactory in most applications [6]; Challenges to improve their performance include: better methods for representing user behavior and the items to be recommended; more advanced recommendation modeling methods; incorporation of various contextual information into the recommendation process; more explainable recommendations; and others [4].

Although investigation about recommender systems dates back to earlier work – namely, in cognitive science [7], approximation theory, information retrieval [8] and others –, recommender systems emerged as an independent research area when researchers started focusing on recommendation problems by explicitly relying on the ratings structures [4]. There are two common formulations of this problem. The recommendation problem is reduced to the task of estimating a rating for items that have not been seen by the user. Once the unknown ratings are estimated the system will recommend the items with a higher rating for that user, the prediction problem. Another possible approach is to reduce the recommendation task to the problem of recognizing if an item is interesting for an user. In this way the system will recommend the top N items of an user, the item ranking problem [9].

The new ratings formulation of the problem introduced new techniques [10]. From these different techniques, Collaborative Filtering (CF) has become the standard approach to solve recommendations problems [11]. CF algorithms are relatively simple to implement while having a satisfactory performance in many applications, when compared to other algorithms. Different algorithms have different characteristics, and biases. Their performance varies so choosing the right algorithm is a step towards improving the performance.

We have many algorithms that not only perform differently across different datasets, but also even for different users of the same dataset. Research in fairness-aware recommender systems has

shown that the outputs of recommendations algorithms are, in some cases biased, against specific groups of users [12] [13]. As a result, discrimination among users will degrade the performance of our system. Discrimination in recommendations is originated from different sources: underlying biases in the input data, or the result of recommendation algorithms [13]. Different algorithms have different impacts on the recommendations for different groups of users.

1.1 Motivation

We expect the performance of the algorithms to vary not only for datasets but even within the same dataset, for instance, in RS, it is different for different users. Recommender systems aim to make good recommendations for all users, ideally the biases to groups of users should be removed. However, the bias is directly linked to the algorithms [13]. Choosing the right algorithm is a step towards improving the performance of the system. This can be called the algorithm selection problem.

The challenge of Meta Learning is to learn from prior experience in a systematic, data driven way [14] and can be used to tackle algorithm selection problem. We need to collect metadata, by extracting features that describe prior learning tasks. A Meta Learning model will learn from the metadata to transfer knowledge for future tasks [14]. There are different ways to extract features from prior tasks: Statistical and Information-Theory Characterization, Model-Based Characterization, Landmarking [15]. There are many challenges when extracting metafeatures: there are different metrics to characterize a task and we have to manually select which we should use. Using Meta Learning to solve the problem of different performance for different users within the same dataset can be challenging. This challenges arise from the difficulty of developing metafeatures for a single user.

Representation Learning shares the same goal of the metafeatures, i.e, learn good representation about the data. In Representation Learning we try to extract compact representations from heterogeneous types of data in a numerical representation that can be reused in other learning tasks. The nature of Representation Learning induces the hypothesis of using these techniques to retrieve metafeatures about the data. These techniques can be used to learn representations about the data while keeping as much information as possible [16]. This might indicate that they can be used to retrieve metafeatures at the user level.

1.1.1 Proposal

We propose a completely new solution to the algorithm selection in recommender systems on the user level. We aim to solve the problem of different algorithms that have a different bias in users. To do so, we propose to use Meta Learning. When using Meta Learning we have to extract features about the task we want to solve. We suggest using Representation Learning techniques to retrieve these features. To the best of our knowledge, this solution is the first of its kind: using Representation Learning for Personalized Algorithm Selection in Recommender Systems.

1.1.2 Research Questions

One approach, that was introduced in this work is to use Representation Learning to retrieve meta features on the user level. These metafeatures are going to be used by the meta model to recommend an algorithm for each user. The first research question, **Can we use Representation Learning to obtain useful metafeatures on the User Level?**, will help us understand if a meta learning model trained on metafeatures retrieved by Representation Learning techniques can have a better performance than a single CF algorithm.

There are many different representation learning techniques each one of them with its biases. The meta model will be using the representations obtained by these techniques to extract knowledge, that will then be used to select the best algorithm for each user. **Which Representation Learning Technique is the best?** is the second research question we aim to solve.

This framework was proposed to reduce the bias of a single Collaborative Filtering algorithm. All users should have good performance, the ultimate goal of this framework is to improve the performance on the user level (base level). The third research question **What is the impact on the base level performance achieved by μ-cf2vec?** aims to understand the impact of this framework on the user level.

While developing this framework we encountered a new challenge: there are some users for which none of the base learners can make good recommendations. To understand the impact of these users we seek to answer the final research question **What is the impact of the zeroes class on μ-cf2vec?**

1.2 Contributions

This work has two main contributions - μ-cf2vec: Representation Learning for Personalized Algorithm Selection in Recommender Systems - which is a framework that uses Representation Learning techniques to retrieve user embeddings, and will be used by a Meta Learning model to solve the algorithm selection problem on the user level. The contribution of this framework is to automatically retrieve metafeatures for the users using RL. Using these techniques we remove the need to generate handcrafted metafeatures which can be slow and inefficient.

The second contribution is an empirical study of the proposed framework. We implemented a version of the μ-cf2vec framework. In this implementation we used two Representation Learning techniques: Variational Auto-encoders and Denoising Auto-encoders for Collaborative Filtering. We used a group of 5 different base learners and a set of 5 meta learners.

1.3 Organization of the dissertation

The rest of the dissertation is organized into five additional chapters:

- Chapter 2 discusses the related work to this dissertation. It gives an overview of the following themes: Collaborative Filtering techniques and evaluation metrics, Representation Learning techniques, and Meta Learning strategies.
- Chapter 3 gives an overview of the framework.
- Chapter 4 puts forward an empirical study where we first present the experimental setup, in both base level and meta level, and then we describe the experiments made. To close there are four research questions, which are the core of this work, being discussed in this chapter.
- The final chapter 5 deals with concluding remarks, limitation of this work and proposals for future work.

Chapter 2

Literature Review

2.1 Collaborative Filtering

Collaborative Filtering (CF) has become the standard approach to address recommender systems problems. Collaborative Filtering algorithms try to make predictions about interests of a user by collecting the personal interests from multiple users. In a CF problem, we have a list of m users and a list of n items, each user has a list of items which he has rated or or the personal preferences of the user can be implicitly inferred from his behavior. Figure 2.1 presents a diagram of the Collaborative Filtering processs. The datasets can be explicit if the user gives ratings to the items. In this case the model will predict the ratings from a user to items and then recommend the items with the best rating [9]. There is another type of CF datasets: implicit datasets where the interaction is 1 if the user clicked/purchased the item and 0 otherwise. When faced with this type of problem the model will only recommend items [9].

Collaborative Filtering refers to making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating) [17]. CF systems suffer from some challenges: Cold Start problem, scalability and they have to deal with high sparse data. The Sparsity is connected to the high number of users and items. It is unlikely that each user has clicked on every item available. This challenge is also connected to the Cold Start problem; this happens when a new user or a new item has just entered the system. Since these items/users are new and we don't have any interactions, the system cannot generate useful recommendations [18]. The recommender systems also need to be scalable. In real world applications the number of users and items will be very high and will keep increasing [19].

The CF techniques can be classified in different categories such as: Memory-Based Collaborative Filtering, Model-Based Collaborative Filtering and Hybrid Collaborative Filtering [11]. For the purpose of this work we are only going to focus on the first two categories.

Memory-Based Collaborative Filtering was the first generation of CF systems and uses similarity functions to compute the similarity between users or items and making recommendations based on those similarity values [21]. These systems are easy to implement and can perform well. However, Memory-Based Collaborative Filtering algorithms face scalability problems due to their

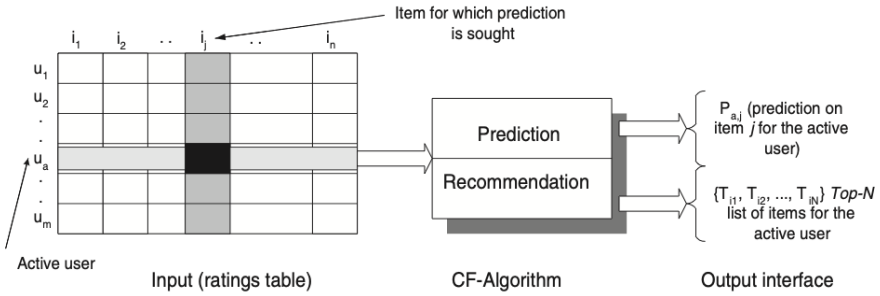


Figure 2.1: Collaborative Filtering Process [20]

design [22]. Since the algorithms have to compute the similarity between users or items, if we have high sparse data, the algorithms will underperform. Due to the same reasons, this technique also does not perform well when faced with the new items or users, i.e. the Cold Start problem. Item-item Collaborative Filtering is an example of a Memory-Based Collaborative Filtering technique. Figure 2.2 helps to understand the architecture of the algorithm. In this example: U3 clicks on Item C, therefore is likely that U3 will also click on an item similar to the Item C. As is the case here, if item C is similar to item A (a similarity which is measured using similarity function), we can then infer that U3 may find Item A appealing. Therefore, we are going to recommend Item A to the U3.

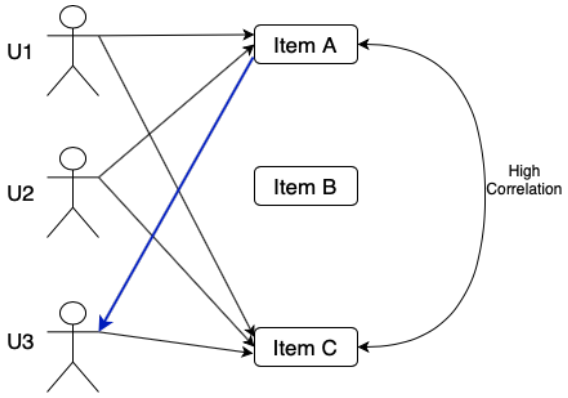


Figure 2.2: Item-based knn

We have multiple similarity functions [20][23]. Cosine similarity function is a standard function. Here two items are two vectors in the m dimensional user space. The similarity is then measured by computing the cosine of the angle between these two vectors. In the ratings matrix in the figure 2.1 the cosine similarity between items i and j is given by (2.1) [20].

$$\text{sim}(i,j) = \cos(\theta) = \frac{i \cdot j}{\|i\|j\|} \tag{2.1}$$

Term Frequency times Inverse Document Frequency (TF-IDF) can also be used as a similarity function for CF techniques[24]. In item-based Collaborative Filtering the items are the documents,

and the users the terms. TF-IDF is a numerical statistic that can measure how relevant a given item is. This relevance can be used to correlate the items and can be calculated from equation (2.2), where $tf_{i,j}$ is the number of occurrences i in j; df_i is the number of documents containing i; and N is the total number of documents [25].

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \tag{2.2}$$

BM25 can also be used as a similarity function in Collaborative Filtering. Similar to TF-IDF, BM25 ranks a set of documents (items in CF) similar to a query (users). BM25 weight can be instantiated by (2.3). Where w_i is the usual RSJ weight, $f(tf_i) = \frac{(k_1+1)tf_i}{K+tf_i}$, $K = k_1\left((1-b)+b*\frac{dl}{avdl}\right)$, dl is the document length; $avdl$ is the average document length; k_1 and b are global parameters which are in general unknown, but may be tuned on the basis of evaluation data [25]. For this work these three are the more relevant, however there are more similarity functions [26].

$$\text{BM25 weight } w_i = f(tf_i) * w_i^{(1)} \tag{2.3}$$

Model-based Collaborative Filtering algorithms can improve the performance compared to Memory based CF, and can address the sparsity problem. Since the Netflix Prize [27], Matrix Factorization algorithms have become very popular to solve CF tasks. These algorithms work by creating a representation for each user u with a k dimensional vector x_u and for each item i with k dimension vector y_i. We can call these vectors factors. The goal is to predict the users u rating for item i, $r_{ui} \approx x_u^\top y_i$. We can also write this in matrix notation, knowing that $x_1, \ldots, x_n \in \mathbb{R}^k$ are the factors for the users, and $y_1, \ldots, y_m \in \mathbb{R}^\kappa$ are the factors for the items. The $k \times n$ user matrix X and the $k \times m$ item matrix Y are then defined by:

$$X = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_n \\ | & & | \end{bmatrix}, Y = \begin{bmatrix} | & & | \\ y_1 & \cdots & y_m \\ | & & | \end{bmatrix}$$

The target is to complete the ratings matrix $R \approx X^T Y$. To achieve it we can see this as a problem in which the goal is to minimize the objective functions and find the optimal X and Y. There are multiple Matrix Factorization techniques presented in [28]. For this work we are going to focus on three algorithms: Alternating Least Squares, Bayesian Personalized Ranking, and Logistic Matrix Factorization.

2.1.1 ALS

Alternating Least Squares [29] [30] [31] is a matrix factorization technique that aims to minimize the least squares error of the observed ratings (and regularize). Regularization tries to reduce the variance of the model without a substantial increase in its bias. The parameter λ can be tunned in regularization techniques by controlling the impact on bias and variance. We can formulate this problem to minimize the objective function (2.4).

$$\min_{X,Y} \sum_{r_{ui} \text{ observed}} \left(r_{ui} - x_u^\top y_i\right)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2\right) \tag{2.4}$$

A common way to optimize the objective function is by using gradient descent [32]. However, for this case gradient descent turns out to be slow and computationally expensive. Another technique we can use is to fix a set of variables X as constants, and optimize the function for Y. Afterwards we fix Y and then optimize for X. We keep repeating this process until convergence. The algorithm (1) can help us visualize the procedure. This alternation brings the name Alternating Least Squares.

Algorithm 1: ALS Matrix Completion

Initialize X, Y;
repeat
for $u = 1 \dots n$ **do**

$$x_u = \left(\sum_{r_{ui} \in r_{u*}} y_i y_i^\top + \lambda I_k\right)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} y_i \tag{2.5}$$

end
for $i = 1 \dots m$ **do**

$$y_i = \left(\sum_{r_{ui} \in r_{*i}} x_u x_u^\top + \lambda I_k\right)^{-1} \sum_{r_{ui} \in r_{*i}} r_{ui} x_u \tag{2.6}$$

end
until convergence

2.1.2 BPR

Bayesian Personalized Ranking (BPR) [33] is focused on solving the personalized ranking task. The goal of this algorithm is to provide user u a personalized ranking, denoted by $>_u$. As mentioned previously, the usual approach for item recommendation is to predict a personalized score $\hat{r}_{ui}$ for an item that reflects the user's preference for that item. However, BPR uses a different approach. If a user has interacted with item i then we assume that the user prefers this item over all other non-observed items. The goal of BPR is to recommend the items that the users prefer such as belong to $>_u$. This method uses a Bayesian approach. Assuming Θ is the parameter of the model that determines the personalized ranking, BPR's goal is to maximize the posterior probability [34]:

$$p(\Theta \mid i >_u j) \propto p(i >_u j \mid \Theta)\, p(\Theta) \tag{2.7}$$

Where $p(i >_u j \mid \Theta)$ is the likelihood function that captures the individual probability that a user really prefers item i over j and is computed in the following way:

$$p(i >_u j \mid \Theta) := \sigma(\hat{x}_{uij}(\Theta)) \tag{2.8}$$

Where $\sigma(x) = \frac{1}{1+e^{-x}}$, and $r_{uij}(\Theta)$ captures the relationship between user u, with the items i and j. The framework delegates the task of modelling the relationship between u, i and, j to an underlying model class, which is in charge of estimating $r_{uij}(\Theta)$. To complete the Bayesian modelling approach of the personalized ranking task, it is necessary to introduce the general prior density $p(\Theta)$ which is a normal distibution with zero mean and variance-covariance matrix Σ_Θ. The prior density is given by: $p(\Theta) \sim N(0, \Sigma_\Theta)$. To reduce the number of unkown hyperparameters $\Sigma_\Theta = \lambda_\Theta I$ is set. The maximum posterior estimator can be formuated to derive the generic optimization criterion for personalized ranking BPR-OPT

$$\begin{aligned} \text{BPR} - \text{OPT} &:= \ln p(\Theta \mid >_u) \\ &= \ln p(>_u \mid \Theta)\, p(\Theta) \\ &= \ln \prod_{(u,i,j) \in D_S} \sigma(\hat{x}_{uij})\, p(\Theta) \\ &= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\ &= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \|\Theta\|^2 \end{aligned} \tag{2.9}$$

Where λ_Θ are model specific regularization parameters. Since the criterion is differentiable, it is possible to use gradient descent based algorithms for maximization. Limited by the characteristics of the Collaborative Filtering data, the maximization is achieved by using a stochastic gradient descent technique that chooses the triplets u, i, and j randomly [33].

2.1.3 LMF

Logistic Matrix Factorization (LMF) [35] has a similar approach to ALS by factorizing the ratings matrix R by 2 lower dimensional matrices $X_{n\times f}$ and $Y_{n\times f}$, where f is the number of latent factors. Similar to the ALS algorithm, X matrix represents the taste of the users and Y^T represents the items. LMF uses a probabilistic approach, different from ALS. Assuming that $l_{u,i}$ denotes the event that a user u has chosen to interact with the item i, the probability of this event occurring can be distributed according to a logistic function. This function is parametrized by the sum of the inner product of user and item latent factors and user and item biases.

$$p(l_{ui} \mid x_u, y_i, \beta_u, \beta_i) = \frac{\exp\left(x_i y_i^T + \beta_u + \beta_i\right)}{1 + \exp\left(x_u y_i^T + \beta_u + \beta_i\right)} \tag{2.10}$$

Where the β_u and β_i terms represent user and item biases. Some users will tend to interact with a diverse assortment of items while others will only interact with a small subset. Non zero entries of the observation matrix represent positive observations and the zero entries represent negative observations. The "confidence" in the entries of R is given by $c = \alpha r_{ui}$ where α is a tuning parameter. To achieve the best results α is chosen to balance the positive and negative observations. By making the assumption that all entries of R are independent, the likelihood of the observations R is given by the parameters X, Y and β and it is defined by:

$$\mathscr{L}(R \mid X, Y, \beta) = \prod_{u,i} p\left(l_{ui} \mid x_u, y_i, \beta_u, \beta_i\right)^{\alpha r_{ui}} \left(1 - p\left(l_{ui} \mid x_u, y_i, \beta_u, \beta_i\right)\right) \tag{2.11}$$

The goal of the model is to learn X, Y and β that maximize the log posterior.

$$\arg\max X, Y, \beta \log p(X, Y, \beta \mid R) \tag{2.12}$$

A local maximum of the objective defined in equation (2.12) can be found by performing an alternating gradient ascent procedure. In each iteration, the user vectors X and biases β are fixed to take a step towards the gradient of the item vectors Y and biases β. Next, the item vectors Y and biases β are fixed to take a step towards the gradient of the user vectors X and biases β. In this situation, the number of iterations required for convergence can be reduced dramatically by choosing the gradient step sizes adaptively [35].

2.2 Evaluation

The evaluation of Collaborative Filtering algorithms is usually performed by procedures that split the dataset into training and test subsets using sampling strategies. There are different techniques such as k-fold cross-validation and split the dataset in train validation and test [36]. Figure 2.3 presents a typical Collaborative Filtering problem data partitioning. In CF tasks the users are split in three groups: train, validation, and test. The training users' interactions are used to train the model. The validation users allow for the tunning of the parameters, and the test users are used to make an unbiased evaluation of the model.

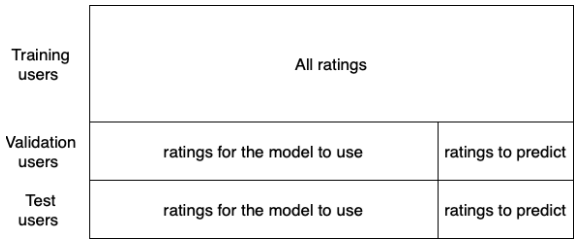


Figure 2.3: CF data splitting

As stated previously, Collaborative Filtering tasks can have two different categories of datasets, explicit and implicit. Consequently, there are two different ways to estimate the performance of the model: rating prediction, and item ranking [37]. In rating prediction, our model will try to estimate the rating r_{ui} from a user u to an item i. To measure the performance of our model we can use metrics like Root Mean Squared Error (RMSE) or Normalized Mean Absolute Error (NMAE). This metrics will try to measure what was the error of our model prediction. In a ranking task, the model will try to rank the items that the user u is more likely to have an interaction with. To evaluate the performance of our model we need to use metrics that can measure the ranking of an item. Normalized Discounted Cumulative Gain (NDCG) can be used for this task. To make use of

this metric we need to compute the Discounted Cumulative Gain which is given by the equation (2.13) where rel_i is the relevance of the result at position i [38].

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)} \tag{2.13}$$

The premise of this metric is: items that are relevant for the user but appear in the last positions of the recommendation list should be penalized. Therefore the relevance of an item will be affected if recommended in the latest positions. To normalize this metric in each position we have to compute the Ideal Discounted Cumulative Gain (IDCG) that is given by the equation (2.14), where REL_p represents the list of relevant documents (ordered by their relevance).

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{2.14}$$

With this information, we can finally compute the NDCG

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p} \tag{2.15}$$

Normalized Discounted Cumulative Gain, as stated previously, is a metric that measurs the ranking of an item. This ranking is only made for the top K results. $nDCG$ can be written $nDCG@K$, where K is the number of the top results.

2.3 Representation Learning

Representation Learning (RL) is about acquiring compact representations from heterogeneous types and different structures of data in a numerical representation that can be reused in other learning tasks. Representation Learning has become a new research area and there are multiple different techniques. For the purpose of this work we are only going to cover the deep learning methods [16]. They learn representations of the data by forming a composition of multiple non-linear transformations with the goal of yielding more abstract, and ultimately more useful. Representations can be very interesting because they can express many different details about the world around us, i.e. they are not task-specific but can be used in many different ways [16]: Speech Recognition and Signal Processing, Object Recognition, Natural Language Processing, Multi-Task and Transfer Learning, Domain adaptation and Recommendation Systems.

In Recommender Systems, state of the art results are achieved by using representation learning techniques [39] [40] [41] [42] [43] [44] [45] [46].

The performance of machine learning methods is critically dependent on the representations it learns to output. Good representations are expressive, meaning that a reasonably sized learned representation can learn a huge number of possible input configurations [16]. We can measure the expressiveness of a model by counting how many parameters it requires compared to the number of input regions it can distinguish [16]. Deep algorithms promote re-use of features and can lead to

more abstract features at higher layers of representation. The representations should disentangle the factors of variation. We want our factors to be as disentangled as possible and to discard as little information about the data as possible. In classification tasks we have a clear goal (i.e minimizing the misclassifications). When learning representations the goal is far-removed from the ultimate objective, which is typically learning a classifier or some other predictor[16]. We want our representation to be disentangled, however, it is not an easy task to implement that into the training criteria.

Representation Learning [16] can be used in supervised and unsupervised tasks. There are multiple techniques that can learn good representations. Principal Component Analysis is one of the most used and oldest algorithm using representation learning techniques. The idea is to reduce the dimensionality of the data while preserving as much statistical information as possible [47]. Restricted Boltzmann Machines (RBM) are two-layer neural networks composed of two types of units, visible and hidden. After successful learning, an RBM provides a closed-form representation of the distribution underlying the observations [48]. Auto-encoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion [49]. Figure 2.4 presents a simplified architecture for easier readability.

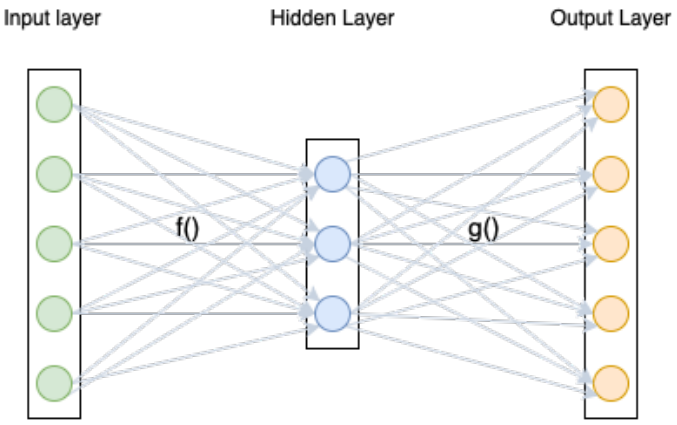


Figure 2.4: Auto-encoder architecture

Auto-encoders are obtained by training a neural network to reproduce the input vector in the output vector using a hidden layer with less neurons than the input layer. For such, the network learns two functions: an encoding function f and a decoding function g. Since this hidden layer is able to preserve useful properties of the data, it can represent the input in a latent space [50]. As a consequence Auto-encoders can reduce the data dimension (compress/encode), and then are able to reconstruct the data as close as possible to the original input data. Theoretically, auto-encoders can be used in any machine learning task, from natural language processing to Collaborative Filtering. For the purpose of this work we are going to look in-depth at two auto-encoder architectures that make auto-encoders the state of the art technique in Collaborative Filtering tasks [44] [45].

An auto-encoder has two components: encoder and decoder. The encoder tries to encode or compress the input data into a hidden layer that is a low dimension latent-space representation, the bottleneck. The Decoder is the part of the network which tries to reconstruct the encoded data

to the original dimension. The decoder data tries to be as close as possible to the original data. Reconstruction Loss is the method that measures how close the decoder output is to the original input. Auto-encoders have multiple different configurations. For our purpose we are going to explore denoising auto-encoders and variational auto-encoders.

2.3.1 Denoising Auto-encoders

Denoising auto-encoders try to solve the problem of corrupted data. This auto encoder randomly corrupts the inputs by introducing some kind of noise, then the auto-encoder must reconstruct or denoise the data. This noise can be generated in multiple ways: assign random subset of inputs with 0, or add Gaussian noise [51][52]. Figure 2.5 presents the denoising auto-encoder architecture, these auto-encoders are obtained by adding noise to the input layer, after which the auto-encoder, using the encoder f() and the decoder g(), will try to reconstruct the data as close as possible to the data without the noise.

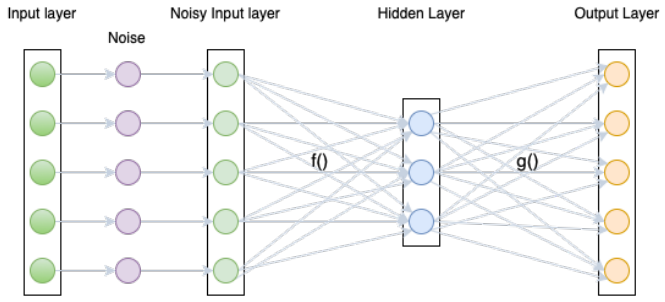


Figure 2.5: Denoising Auto-encoder architecture.

This algorithm is also used in Collaborative filtering tasks [44] and can achieve state of the art results.

2.3.2 Variational Auto-encoders

A Variational Auto-encoder (VAE) provides a probabilistic way of describing an observation in latent space, such that the encoder describes a probability distribution for each latent attribute. Figure 2.6 presents the architecture of a VAE. The encoder learns to project the high-dimensional input to a low-dimensional latent space. The decoder learns to decode a vector in the low-dimensional latent space to reproduce the high-dimensional input. A VAE is different than the previous discussed auto-encoders. It is designed in such way that the generated latent factors follow a unit Gaussian distribution [53]. Instead of each latent space having a single value, here we will have the probability distribution for each latent attribute. To achieve a continuous latent space vector the encoder, f(), encodes the data to two vectors μ and σ. Then these two vectors are sampled and the decoder, g(), reconstructs the data. This unique property of the Variational Auto-encoders makes their latent space designed to be continuous, allowing random sampling. The latent space will be probabilistic over the latent attributes, creating more information than in

the previous auto-encoder approaches [54]. This architecture is used in many different areas, in Collaborative Filtering can achieve state of the art results [45].

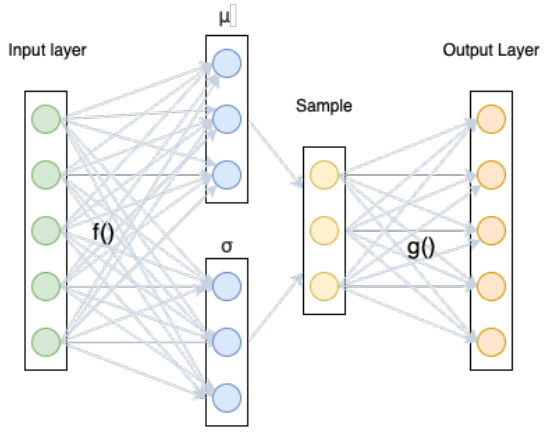


Figure 2.6: Variational Auto-encoder architecture.

Another area that will be important for this work is distributed representations. Distributed representations are an important component in order to achieve good representations. The idea is to represent each entity by a pattern distributed over many elements, and each element participates in the representation of many different entities [55]. Distributed representations have been used in many different problems. Word2vec [56] has become the state of the art Natural Language Processing technique, and it works by assuming that if two words have similar contexts then they have similar representations. These representations can describe the context of the words and can be used to predict the surrounding words given the current word. Representations can be used to translate a wide variety of different entities. We can use representations to identify similar entities based on the context that the representations map. These entities can be words [56], image search queries [57], collaborative filtering algorithms [58].

2.4 Meta Learning

Meta Learning (MtL) is also known as learning how to learn. Meta learning aims to learn from prior experience in a data driven way to extract knowledge to then use in other tasks [14]. During base learning, a base learner algorithm solves a task defined by a dataset and objective. During meta-learning, a meta learner algorithm updates the base learner(s), such that the base level improves the meta level objective [59]. Meta learning can be used in any type of learning based on previous tasks. The more similar those previous tasks are the more metadata we can leverage [14].

We can learn purely from model evaluations. These techniques can be used to recommend the best configurations of the model for similar tasks [14]. Our metadataset will be a set of possible configurations: all known tasks, a set of learning algorithms, the configurations of the algorithms, hyperparameter settings, etc [14]. The final goal will be to train a meta learner model to predict the best configurations for a new task[14].

We can also transfer trained models parameters between tasks inherently similar. Transfer learning focuses on transferring knowledge across domains [60], is an example of this technique. Using Transfer Learning, we are going to build a metadataset with the machine learning models themselves. Our meta learner learns how to train a new base learner for a new task given similar tasks [14].

Another way to solve meta learning problems is to characterize tasks more explicitly. We can try to extract task similarity and build metadatasets with the task characterization. The meta-models will learn relationships between data characteristics. Using these relationships the meta-models can rank promising configurations, predict the algorithm performance, select the algorithm and others [14]. For the purpose of this work we will focus on Algorithm Selection that was firstly conceptualized by Rice's work in [61]. Figure 2.7 presents a diagram of the conceptualized framework which defines several search spaces: problem, feature, algorithm and performance by P, F, A and Y. A problem is described as: for a given problem $p \in P$, with features $f(p) \in F$, find the mapping $S(f(p))$ into A, such that the selected algorithm $a \in A$ maximizes $y(a(p)) \in Y$ [61]. Therefore, algorithm selection can be formulated as a learning task whose goal is to learn a meta model able to recommend algorithms for a new task.

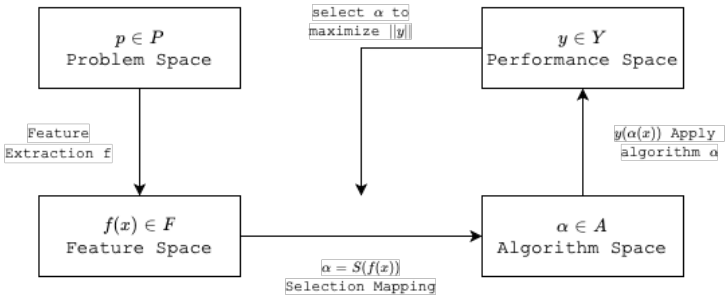


Figure 2.7: Schematic diagram of Rice's Algorithm selection conceptual framework [1]

Meta learning can be used to address Algorithm Selection problem. We have different meta-learning levels. Base level is where the base learners accumulate experience on the learning task. Meta-level is where meta learners learn about the performance of multiple base-learners in multiple learning tasks. The metafeatures are used by the meta-learners to extract knowledge about the task and use this knowledge to select the best algorithm. Therefore it is important that these metafeatures characterize the data as best as they can. The challenge we face in this approach is how to get the meta-features. There are different techniques to extract relevant information about the task under analysis [15]:

- Statistical and Information-Theory Characterization. Characterize the dataset by extracting statistical and information-theory parameters estimated from the training set. Measures include the number of classes, number of features, degree of correlation between features, etc[62].

- Model-Based Characterization. This technique exploits properties of the induced hypothesis as a form of representing the dataset itself. It has several advantages: the dataset is

summarized into a data structure where the resulting representation can serve as a basis to explain the reasons behind the performance of the learning algorithm.

- Landmarking. This concept exploits the information obtained from the performance of a set of simple learners that exhibit significant differences in their learning mechanism[14].

Retrieving expressive metafeatures is a challenging task. As stated previously there are different techniques to retrieve the metafeatures. For the purpose of this work we are going to focus on using Representation Learning to extract the metafeatures. Representation Learning was described in 2.3 and has the goal of learning representations of the data while preserving as much data as possible. Using Representation Learning to extract metafeatures about the data is not a new idea [58] [63]. Metafeatures and Representation Learning techniques share the same goal of extracting the maximum information about the task/data. From the results presented in [58] [63] we can infer that it is possible to use RL techniques to retrieve metafeatures.

2.5 Related Work

This work uses meta learning techniques to solve the algorithm selection problem on the user level. The metalearning techniques use metafeatures that were retrieved using Representation Learning techniques. To the best of our knowledge, there is no existent research on the problem we aim to solve.

This work proposes to use Representation Learning to extract metafeatures. This idea has been used in different implementations [58] [63]. However, both of these implementations use representation learning on the dataset level.

Our work also proposes to solve the metalearning task of algorithm selection problem on the user level.

There is research in algorithm selection on the user level [64]. This approach uses hand crafted metafeatures to characterize each user. It also uses a set of base learners, each one of them with their own bias. The meta-model will then use the performance label and the handcrafted metafeatures to predict the performance of each algorithm for a new user.

Research has also been done on meta learning at user level using CTR model [65]. Given a collection of models, the goal is to select the best model for each user. This work uses: model and task agnostic techniques to extract meta features. The meta model will output the best base model for each user by learning the relationships from the metafeatures.

Chapter 3

Approach

This chapter gives an overview of the problem we tried to solve and the we explain the framework proposed to address this task.

3.1 Problem Definition

Collaborative Filtering is the standard approach to tackle recommendation systems problems. We expect the performance of CF algorithms to vary not only for datasets, but even within the same dataset, for instance, it is different for different users. For each user there is an algorithm that obtains the best performance. The goal is to predict the best algorithm for each user; by doing this we could reduce the biases of the algorithms against groups of users. As stated in chapter 2 this can be formulated as an algorithm selection problem in Meta Learning.

From a set of algorithms we expect the performance of the algorithms to vary not only for datasets but even within the same dataset. Different users have different algorithms performances. Figure 3.1 presents a traditional Collaborative Filtering dataset with the exception of the last column. The last column represents a list of algorithms with their performances for each user.

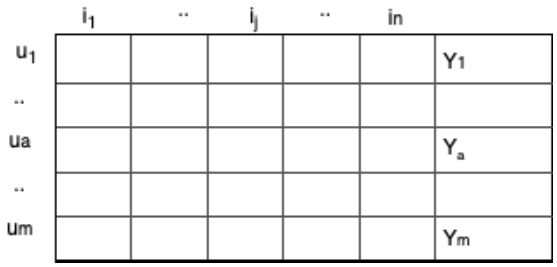


Figure 3.1: Each CF user has a group of algorithms with different performances (Y)

We can use Meta Learning to address the algorithm selection problem. From the performances of each algorithm we can use MtL to address different issues: label ranking, selecting algorithm, predicting algorithms performances, selecting hyperparameters and others.

For a given dataset $d_i \in D$, where D is a set of Collaborative Filtering datasets, we want to predict the best algorithm for each user. To do so, we have a set of CF algorithms B that have different performance for each user. The performance os these algorithms can be used as a label for the MtL task. Let us consider a finite set of labels $L = \{l_1, l_2, \ldots, l_t\}$, where t is the total number of labels available. These labels are general and are dependent on the task we aim to solve. They are related to the algorithms that we want to choose. These can be used as a classification problem, to predict the base learners errors, ranking the labels and others.

For MtL be able to solve the algorithm selection problem on the user-level, we need to characterize the user. This characterization intends to create representation about the users. The representations (metafeatures) will allow the meta models to extract knowledge and recommend the best algorithm for each user. The representations of the users can be manually created using landmarkers, using information-theoretical measures and others. Metafeatures are obtained by using a function $f : u_a \rightarrow \mathbb{R}^k$ that, when applied to each user u_a in a dataset d_i, returns a set of k values that characterize the user.

3.2 Framework Overview

To try to tackle the problems previously addressed we propose a framework named μ-cf2vec: Representation Learning for Personalized Algorithm Selection in Recommender Systems. This framework μ-cf2vec proposes using Meta Learning to solve the Personalized Algorithm Selection in Recommender Systems. This framework proposes the usage of Representation Learning Techniques to automatically retrieve expressive meta features that the meta learning models will use to solve the algorithm selection problem. Figure 3.2 presents the procedure this framework follows.

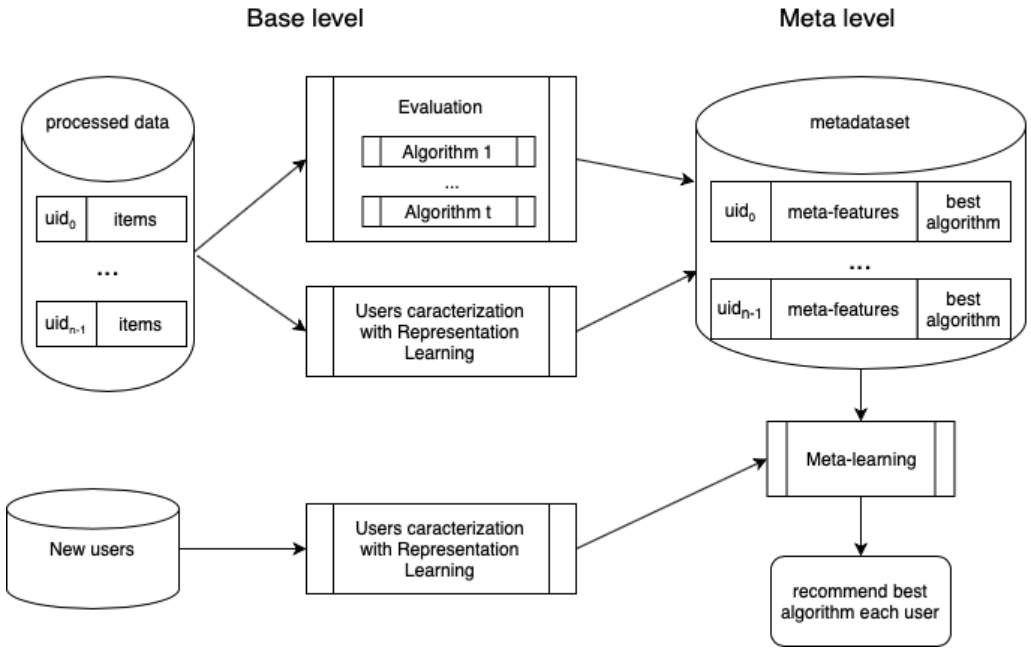


Figure 3.2: Overview of the μ-cf2vec procedure

The procedure presented in figure 3.2 have some resembles to the traditional Meta Learning procedure. However, if we analyse carefully we can observe major diferences. While the traditional MtL tasks try to solve the algorithm selection problem on the dataset level, here we propose to solve on the user level. In traditional MtL procedure we would characterize the dataset, and we would recommend the best algorithm for each dataset. In this framework we aim to select the best algorithm for each user. To do so, we have to evaluate the algorithms for each user and characterize the user. The meta learner will then use these information to recommend the best algorithm for each user.

3.2.1 Metadata Collection

Given a CF dataset d_i – that has a set of users U, that have interactions with a set of items I – we can create a metadataset. Figure 3.3 presents the process of collecting the metadata for each user that is going to be used to create a metadataset. Each row of the metadataset relates to an user and it is composed by the metafeatures, and the label. The metafeatures are a set of k values that characterize the user. The metatarget is a label l_t that can be chosen according to the task.

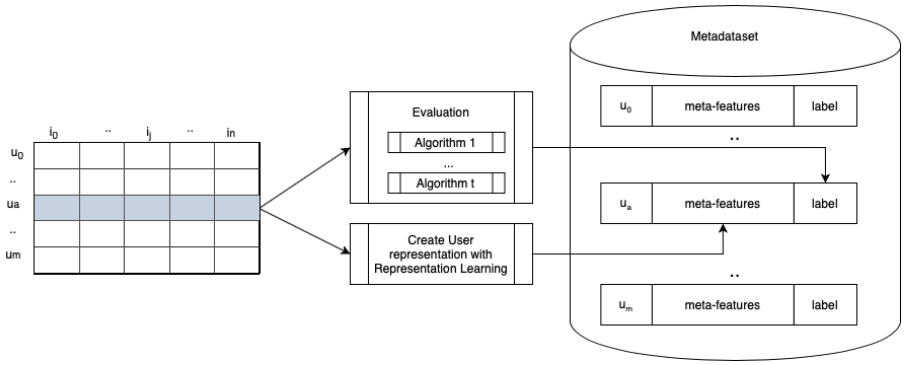


Figure 3.3: Procedure overview to collect metadata for each user

Since this framework proposes to solve a meta learning problem, we need to train and evaluate our models on all users. Figure 3.4 presents the strategy used to split the data. The splitting strategy is different when comparing with the CF tasks. In this MtL we need to characterize each user, therefore we need interactions from each user. Instead of dividing all the users in three groups: train, validation and test; we divide each user's ratings in train, validation, and test. This splitting is done by the framework.

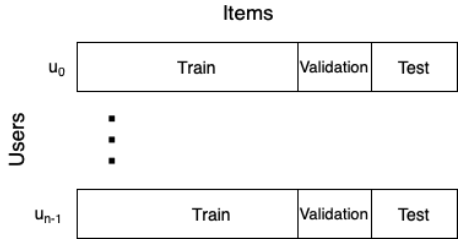


Figure 3.4: Splitting Data Strategy

3.2.1.1 Base Level Evaluation

From a set of base learners B, different users have different performances for different base learners. The base learners are trained using the training interactions from all users. The validation interactions are used to tune the parameters and the test interactions are used to make an unbiased evaluation of the model. There is an evaluation of every base learner for each user.

The evaluation is linked to the metatarget of each user. This evaluation is chosen accordingly to the task we want to solve. The metatarget represents the meta-variable that the meta learner wishes to understand or predict i.e. the algorithm with the best performance for a given task. We can select the metatarget to rank the base learners, select the best base learner, predict the performance of each base learner and others. This makes the framework flexible. We can select the meta target to optimize the task we want to solve.

3.2.1.2 Retrieving metafeatures

One of the contributions of this framework is using Representation Learning to retrieve metafeatures at the user level. The RL techniques allow us to automatically generate metafeatures for each user. Using this approach we remove the need of manually creating metafeatures. Therefore, μ-cf2vec proposes to use Representation Learning techniques to retrieve the metafeatures for each user.

Representation Learning (RL) is about acquiring compact representations from heterogeneous types and different structures of data in a numerical representation that can be reused in other learning tasks. Using these techniques to automatically retrieve metafeatures for each user, is firstly introduced in this framework. μ-cf2vec creates representations for all users using RL techniques. These representations are created using the train interactions from all users. The metafeatures created by applying these RL techniques are going to be used to select the best algorithm for each user. The metafeatures are a latent representation of each user and (ideally) should allow the meta models to relate users. The meta models will leverage from these automatically created metafeatures to select the best algorithm for each user.

3.2.2 Meta Learning

There are a set of M meta learners that are going to be used to solve the MtL task. Meta learning aims to learn from prior experience in a data driven way to extract knowledge, so that it can be used in another task.

μ-cf2vec aims to solve the algorithm selection problem on the user level. As stated previously, the metadataset is built using metafeatures and a meta target for each user. The meta learners will use the metadataset to extract knowledge about the user. This knowledege aims to create relations between users with the same label. These relations are then used to recommend the best algorithm for each user.

There are different evaluations techniques that can be used in meta learning techniques. The framework defines clearly the metadataset, therefore we can use common techniques such as cross

validation. The meta learners recommend the best algorithm for each user by learning from the training data. The training data will be used to train the meta models. The validation interactions are used to fine tune the hyperparameters, and the test data is used to evaluate the performance of the model. Since the framework creates a well defined dataset we can choose the evaluation method more suitable to the task we aim to solve.

3.2.3 Inference

In real world systems, this framework has to be trained before its deployment. The training will be used to select the best meta learners for the task. During the training process the representation learning technique is chosen. The evaluations of the multiple base learners are also obtained in this stage. The training stage will create the metadataset and select the best meta model.

After training the framework, it can be deployed and will automatically update itself. The framework will retrieve the user embeddings periodically, using the newly collected user-item interactions. These embeddings will be retrieved using the best RL technique found on the training step. The meta learner will then extract knowledge which will provide the ability to make personalized algorithm recommendations. For the new users or existing users, for which the system is not able to compute embeddings, the framework will use a naive strategy by recommending the Most Popular algorithm.

3.3 Summary

In summary this framework tries to make personalized algorithm recommendations. It is improbable that one single Collaborative Filtering algorithm has the best performance for all the users. μ-cf2vec proposes using Meta Learning to select the best algorithm for each user. It leverages the usage of Representation Learning techniques to retrieve metafeatures, removing the need of manually choosing metafeatures while capturing characteristics of the data. The framework generates the metadataset using the metafeatures extracted with RL techniques and the metatarget based on the performance of the base learners. The meta learner uses the metadataset to extract knowledge and predict the best CF algorithm for each user. This framework can be used in real world systems by re-running the RL techniques for the users together with the meta learner.

Chapter 4

Empirical Study

This chapter gives an overview of a implementation of the proposed meta-learning framework.

4.1 Base-level Experimental Setup

4.1.1 Dataset

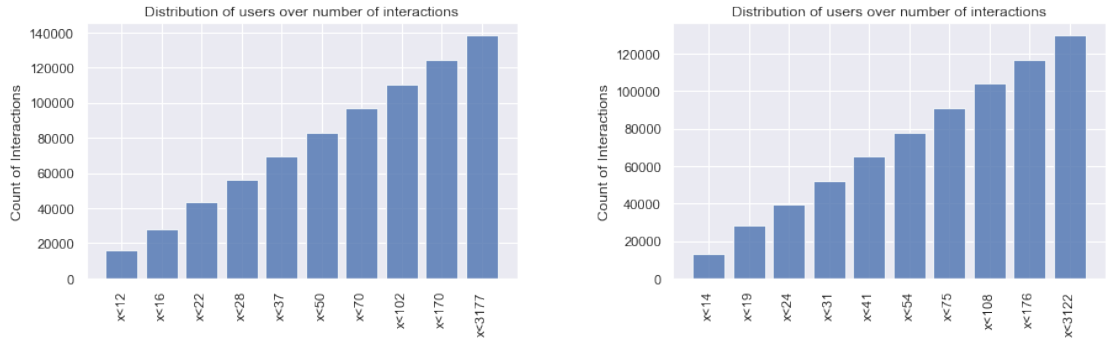
Collaborative Filtering needs a big amount of data. We had many different possibilities to choose from the set D, however, we decided to use the MovieLens 20M Dataset [66] due to its up-to-dateness and the fact that it contains more than 20 millions user-item interactions. The user-item interactions of this dataset are ratings between 1-5, 1 being the worst and 5 the best. We transformed it into an implicit dataset by setting a threshold for ratings greater than 3.5. There is a positive interaction when the rating given by a user to an item is greater that 3.5.

With our dataset chosen, we need to clean our data. For this study we are not going to consider the Cold Start Problem. For this reason we will remove all users and items with less than 10 interactions. This is one of the limitations of this implementation but there are multiple techniques to address the Cold Start Problem [67]. We could try to use them in future work.

Before cleaning the users and items with less than 10 interactions, we had 9 995 410 interactions from 138287 users to 20720 items. Figure 4.1 presents a cumulative distribution of the number of clicks per user. We can observe that 10% of the users have between 1 and 12 interactions, and 90% of the users have less than 170 interactions. The maximum number of interactions that a user has is 3177 interactions.

After cleaning the users and items with less than 10 interactions, we have 9 911 968 interactions from 129 757 users to 11 518 items. We can observe in figure 4.1 that 10% of the users are now between 10 and 14 interactions and 90% of the users have less than 176 interactions. The maximum interactions one user has is 3122.

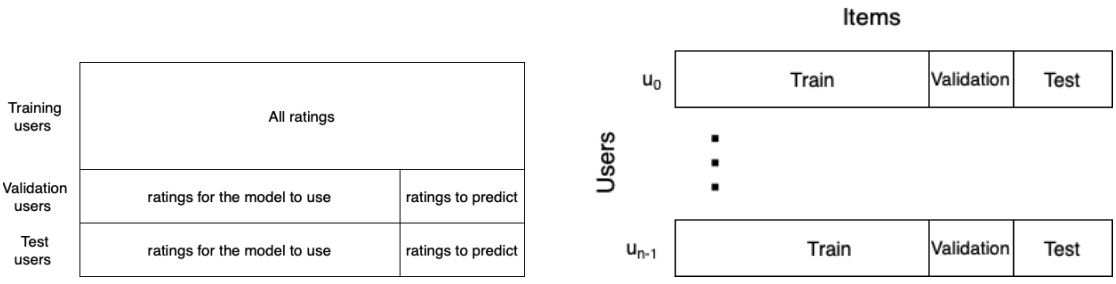
After cleaning our data we have to select the best splitting strategy to tackle this problem. There are multiple different strategies to split the data. This is a metalearning task, hence we want the meta model to be trained on all users. Figure 4.2a presents the common partitions to the Collaborative Filtering task. Figure 4.2b presents the splitting strategy we are going to use. When



(a) Raw Data (b) Removed users and items with less 10 interactions

Figure 4.1: Cumulative Distribution Clicks over Interactions

comparing both figures we can identify some aspects. In CF tasks (figure 4.2a) we split the users into three groups: training, validation and test. The model is trained with the first group. The validation group is used to tune the hyperparameters. The final group is used to do an unbiased evaluation of the performance of the model. The meta learning tasks (figure 4.2b) require all the users to train the model. Each users' ratings are divided in the same three groups. In meta learning tasks it is relevant to define the different percentages of each group. These are selected according to the problem we want to solve. For that reason, in this work we are going to explore the different percentages in order to select the best proportions. In the meta level we only use the training ratings. The other two groups are only used in the base level.



(a) CF task (b) MtL task

Figure 4.2: Splitting strategies in different tasks

4.1.2 Base Learners

Collaborative Filtering has many different algorithms, each one with its own biases. To benefit from different algorithms' biases we have to use a wide variety of algorithms. To do this, we are going to use the Fast Python Collaborative Filtering Framework [68] from which we are going to use the following algorithms:

- Alternating Least Squares (ALS)

- Bayesian Personalized Ranking (BPR)
- Logistic Matrix Factorization (LMF)
- Item-Item Nearest Neighbour models using Cosine, TFIDF or BM25 as a distance metric
- Most popular (to be used as baseline)

Using these algorithms we can capture different biases. We have model-based and memory-based Collaborative Filtering techniques. The first three base learners are matrix factorization techniques. They have a parameter that is the number of latent factors to be computed by each model. Once we decided on the splitting strategy and the base learners we would use, we then needed to understand the performance of the base learners to decide upon the split data percentages.

4.1.3 Data Partitioning

Before choosing the splitting proportions, we have to analyze the base learners' performance. We have chosen implicit datasets, therefore we have to choose an appropriate evaluation method. We have to select a metric that was designed for item ranking and not rating prediction. There are many different choices; our choice was to use the Normalized Discounted Cumulative Gain (NDCG). This metric measures the relevancy for the top K items. Using this evaluation metric, we will analyze the best proportions to divide the users' ratings in our dataset in train, validation and test. The NDCG evaluation will allow us to understand the proportions of each class. The proportions are related to the number of users belonging to each class. Each class has a percentage of occurrences. The Train dataset will be used to train the base learners and to retrieve the metafeatures. The validation set will be used to tune the base learners' hyperparameters. The test set will be used to evaluate the performance of the base learners after being tuned.

We are going to use NDCG@K to analyze the performance of our base learners. However, we have to be careful with the proportions of our split. We have 7 different algorithms and we still may face the problem that in some users none of the algorithms will have a good performance. There may exist a group of users, for which all of the base learners have a NDCG of 0. We are going to call this class of users *zeroes*. To test different data splits we are going to run the base learners and retrieve the NDCG of all models for each user. In the table 4.1 we can compare different algorithms percentages when using different splitting proportions and different values of K.

As stated previously we aim to solve a MtL task. The interactions of each user are going to be splitted in three groups: training, validation and test. This partition is required since we need to to have interactions for all users. We tested three splitting proportions: 80-10-10, 70-15-15 and 70-10-20. Together with three values of K: 10, 20, 100. From table 4.1 we can already tell that the class *zeroes* will have an impact in our framework, since the results for lower values of K have a high percentage of *zeroes*.This issue may impact in our meta model performance. In order to

	80-10-10			70-15-15			70-10-20		
	K=10	K=20	K=100	K=10	K=20	K=100	K=10	K=20	K=100
ALS	10,40	14,10	18,60	11,10	20,30	18,40	19,30	22,90	19,10
BPR	9	11,70	16,50	9,40	15,70	15,70	14,80	17,30	15,90
KNN_BM25	11	13,40	17,80	11,80	5,90	17,30	6,60	7,10	17,20
KNN_COS	4,40	5,30	8,30	5,60	2,50	9,50	2,90	3,10	10,50
KNN_TFIDF	8,90	9,50	8,60	9,80	3,50	8,70	3,50	3,90	9,20
LMF	7,40	9,90	12,40	8,70	12,50	12,20	12,20	13,60	11,30
MOST POP	8,80	10,40	12,50	11,10	17,30	14,70	17,90	19,50	14,90
ZEROES	40,11	25,80	5,50	32,60	22,30	3,40	22,80	12,70	1,70

Table 4.1: Different Algorithm percentages for different Splitting proportions. Values in %.

choose the percentages of our data, we have to find a balance between the value K and the class distribution.

After analyzing the distribution of the classes, we can observe that the matrix factorization performs better than the more simple nearest neighbours algorithms. This happens specially in the proportions of 70-10-20 which is the one with a balance between the number of *zeroes* and the value of K. We were expecting this behaviour, since matrix factorization became the standard approach to solve CF problems. For this reason we decided to remove the KNN algorithms. Table 4.2 presents the new proportions of each algorithm for the different splitting proportions.

	80-10-10			70-15-15			70-10-20		
	K=10	K=20	K=100	K=10	K=20	K=100	K=10	K=20	K=100
ALS	16,40%	22%	31,50%	18,10%	23,50%	31,60%	21,50%	26,40%	33,10%
BPR	12,9%	16,90%	23,70%	13,80%	18,00%	23,30%	16,5%	20,00%	24,10%
LMF	9,60%	12,90%	17,00%	11,50%	13,70%	16,90%	12,90%	14,80%	16,00%
Most Pop	13,40%	16,60%	20,80%	16,90%	20,00%	23,70%	19,80%	22,40%	24,40%
ZEROES	47,70%	31,60%	7,00%	39,70%	24,80%	4,40%	29,20%	16,40%	2,30%

Table 4.2: Different Algorithm proportions for different Splitting proportion without KNN algorithms

In real world applications it is unrealistic to choose a K as high as 100. It is very unlikely that a user chooses an item from the top 100 recommendations. However, higher K makes the *zeroes* class less abundant. We have to choose a K that has a balance between the amount of recommendations the system makes and the percentage of the *zeroes* class. For the previously stated reasons, we decided to choose the $K = 20$ and split 70% for train, 10% for validation, and 20% for test. As we can observe in table 4.2, 16,4% belongs to the class *zeroes*. This choice of splitting proportions allows for a balance between the number of users that are labeled as *zeroes* and the number of top K recommendations.

Using these splitting proportions and $K = 20$ we tuned the hyper parameters for each base learner with the validation set. Table 4.3 presents the factors for each algorithm. The most popular algorithm (baseline) does not have any hyper parameters.

Algorithms	Factors
ALS	50
BPR	50
LMF	20

Table 4.3: Hyper parameters used in base learners

We have our dataset ready to start solving the problem. Table 4.4 presents the percentages of each class in the first 4 stands. This analysis was made to understand how the algorithms were distributed in different standings.

Algorithms	First	Second	Third	Fourth
ALS	26,4%	21,3%	10,1%	2,4%
BPR	20%	16,0%	9,2%	31,1%
LMF	14,8%	11,9%	31,8%	26,4%
Most Popular	22,4%	29%	23,2%	23,7%
Zeroes	16,4%	21,9%	24,6%	23,7%

Table 4.4: Algorithm percentage in the first 4 stands

4.1.4 Obtaining User-level metafeatures based on embeddings

In this work we are going to use state of the art techniques in Representation Learning: Denoising Auto-Encoders for Top-N Recommender Systems [44] and Variational Autoencoders for Collaborative Filtering [45]. Both of these architectures are the state of the art in Representation Learning in Collaborative Filtering tasks as stated in 2. Each architecture has its own biases.

For the Variational Auto-encoders (VAE) we are going to use the code from [69]. Since this model is designed for Collaborative Filtering it splits all the users in Train (80%), Validation (10%) and Test (10%). In our approach, as it is a Meta Learning task, we use all the users in every step and in each user we split the user ratings in Train (70%), Validation (10%) and Test (20%). The splitting strategy of the [69] is integrated with the model, however, we need to obtain the metafeatures for all users. To retrieve the embeddings we fed the model [69] with all of our training data. We let the model train and split the data in its own partitions. After the model was trained we obtained the embeddings for all the users. The model creates a representation of size 200 for each user. Figure 4.3 presents this procedure. All the users training interaction enter the VAE. These users are automatically splitted in 80% for training, 10% for validation and 10% for test. The training users are used to train the models and the parameters are tuned with the validation users. After the model is trained we retrieve the embeddings for all the users.

For the Collaborative Denoising Auto-Encoders (CDAE) we are going to use the DRECPY [70] library. In the original paper of this model the author facilitated the retrieval of the factors for each user. The attribute V has the user embeddings. This attribute has a dimension of UxK, U being the number of users and K the hidden factors. We are going to use two different hidden factors: 50 because it is the number of hidden factors used by the Author, and 200 hidden factors.

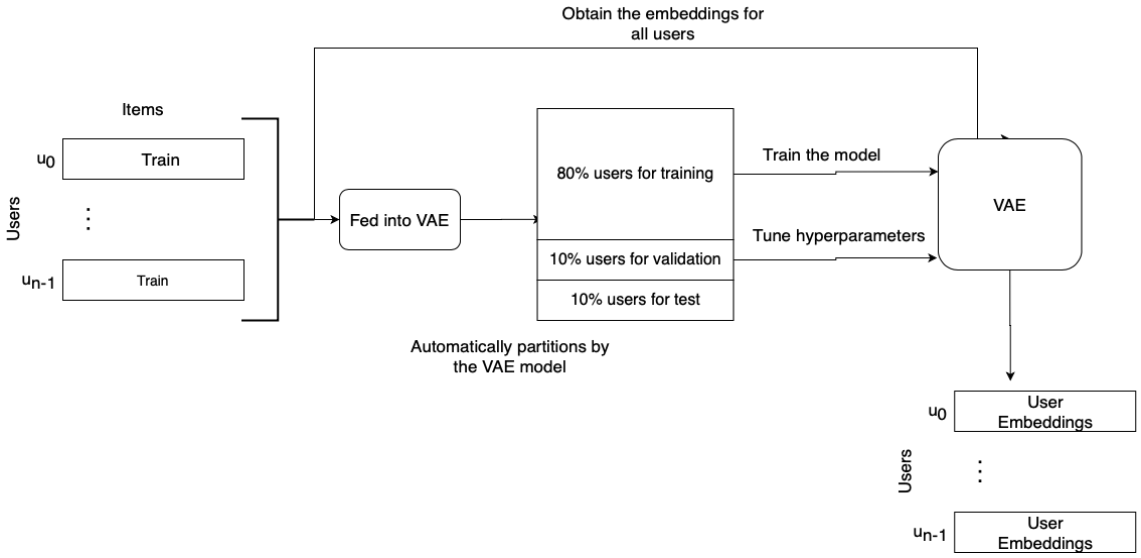
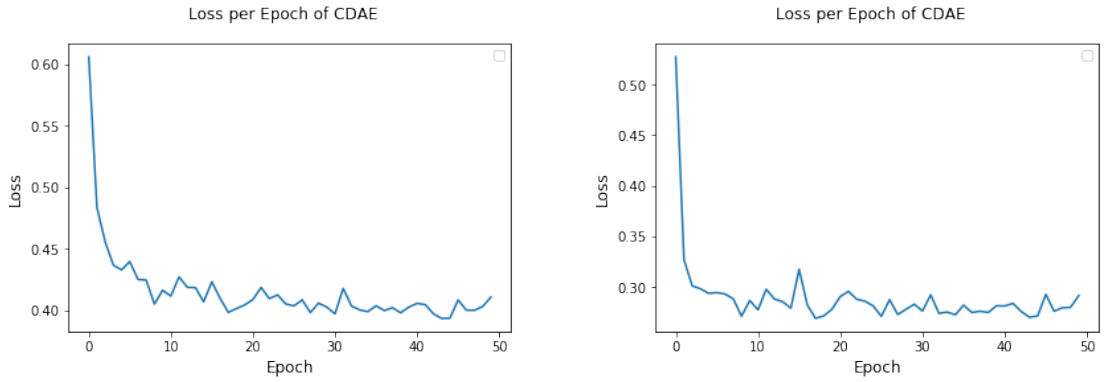


Figure 4.3: VAE model procedure

We retrieve the embeddings of size 200 to make a fair comparison with the Variational Auto-encoders which have size 200. On the figure 4.4 we can see the loss evolution over epoch.



(a) CDAE50 loss per epoch

(b) CDAE200 loss per epoch

Figure 4.4: Loss per epoch

We are going to have three metadatasets: VAE, CDAE50, and CDAE200. Each one of them is built using the metafeatures obtained by RL techniques with the same name, e.g., the VAE dataset is built using metafeatures retrieved with the Variational Auto-Encoder and with the metatarget that is the name of the base learner with the best performance.

4.2 Meta-level Experimental Setup

These metadatasets are going to be used to solve the meta-level problem. We are going to build meta models which will leverage these metadatasets to select the best algorithm for each user. To

achieve this, we are going to implement and evaluate a wide variety of models. We will then select the most suitable for our problem. The meta learners we are going to use are the following:

- Logistic Regression [71]
- Multi Layer Perceptron[72]
- Support Vector Machine [73]
- Random Forest [74]
- Light GBM [75]

These algorithms have different characteristics and different biases. Logistic Regression is an algorithm that tries to fit a logistic regression into the data with the goal of creating a decision boundary. Since this problem is a multiclass problem, we are going to use the one vs all approach. Multilayer perceptron (MLP) is a feed forward artificial neural network. It is composed of a sequence of neurons connected that have an activation function. The activation function maps the weighted inputs to the output of each neuron. This architecture will learn information that will be able to distinguish non-linearly separable data. Support Vector Machine (SVM) tries to find an hyperplane in an N-dimensional space that classifies the data points. The random forest is a supervised learning algorithm that randomly creates and merges multiple decision trees into one forest [76]. Each tree makes a classification and the class with more votes is the prediction. Light GBM is a gradient boosting framework that makes the use of tree based algorithms. Light GBM grows trees vertically while other algorithm grows trees horizontally. Light GBM has a leaf-wise growth, meaning that new trees will grow from a leaf. Other Boosting algorithms have a level-wise growth, the tree will grow in currrent level. The metadatasets are going to be normalized using a z-score normalization. We are going to perform a grid search of every different algorithm in every metadataset. Table 4.5 presents the parameters used by all algorihtms in the different datasets, both normalized and non normalized.

We use Cross Validation with 5 Folds, since this technique allows a deeper look into the model performance. Comparing with the traditional train-test split, this approach reduces the variance.

4.3 Exploratory Metadata Analysis

Our metadataset is composed by the metafeatures and the label of the best algorithm for each user. We are going to use techniques to visualize the metadataset. PCA and T-SNE [77] are well-suited dimensionality reduction techniques for vizualizing high-dimensional data in a low-dimensional space. We are going to use both of these techniques to visualize our three metadatasets normalized and non normalized. In figure 4.5 we can observe the different metadatasets visualizations. In both CDAE datasets the low dimension visualization is identical. In comparison with the VAE dataset the CDAE metafeatures are more concentrated in one place. If this problem was trivial to solve we could group the points with the same label. This would mean that it would be relatively easy

Metadataset	Algorithms	Logistic Regression	MLP	Light GBM	SVM	Random Forest
VAE	Norm	C': 316.23, 'penalty': 'l2' 'solver': 'lbfgs'	{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}	{'colsample_bytree': 0.4, 'learning_rate': 0.05, 'num_leaves': 200, 'reg_alpha': 0.25, 'reg_lambda': 0.25}	{'C': 1, 'loss': 'squared_hinge'}	{'max_features': 'auto', 'n_estimators': 1000, 'bootstrap': True, 'max_depth': 150}
	No Norm	C': 100000.0 'penalty': 'l2' 'solver': 'liblinear'	{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}	{'colsample_bytree': 0.4, 'learning_rate': 0.05, 'num_leaves': 200, 'reg_alpha': 0.15, 'reg_lambda': 0.15}	{'C': 1, 'loss': 'squared_hinge'}	{max_features=auto, n_estimators=1000, bootstrap=True, max_depth=75,
CDAE 200	Norm	C': 1.0 'penalty': 'l2' 'solver': 'newton-cg'	{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}	{'colsample_bytree': 0.25, 'learning_rate': 0.05, 'num_leaves': 5, 'reg_alpha': 0.25, 'reg_lambda': 0.25}	{'C': 1, 'loss': 'squared_hinge'}	{'max_features': 'auto', 'n_estimators': 100, 'bootstrap': True, 'max_depth': 75}
	No Norm	C': 1e-05 'penalty': 'l2' 'solver': 'newton-cg'	{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}	{'colsample_bytree': 0.4, 'learning_rate': 0.05, 'num_leaves': 5, 'reg_alpha': 0.25, 'reg_lambda': 0.25}	{'C': 0.1, 'loss': 'hinge'}	{'max_features': 'log2', 'n_estimators': 250, 'bootstrap': True, 'max_depth': 10}
CDAE 50	Norm	C': 1.0 'penalty': 'l2' 'solver': 'liblinear'	{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}	{'colsample_bytree': 0.15, 'learning_rate': 0.001, 'num_leaves': 5, 'reg_alpha': 0.15, 'reg_lambda': 0.15}	{'C': 10, 'loss': 'squared_hinge'}	{'bootstrap': True, 'max_depth': 10, 'max_features': 'auto', 'n_estimators': 250}
	No Norm	C': 316.22776601683796 'penalty': 'l2' 'solver': 'newton-cg'	{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}	{'colsample_bytree': 0.25, 'learning_rate': 0.01, 'num_leaves': 100, 'reg_alpha': 0.25, 'reg_lambda': 0.25}	{'C': 0.1, 'loss': 'squared_hinge'}	{'max_features': 'auto', 'n_estimators': 100, 'bootstrap': True, 'max_depth': 150}

Table 4.5: Meta Models Hyper parameters

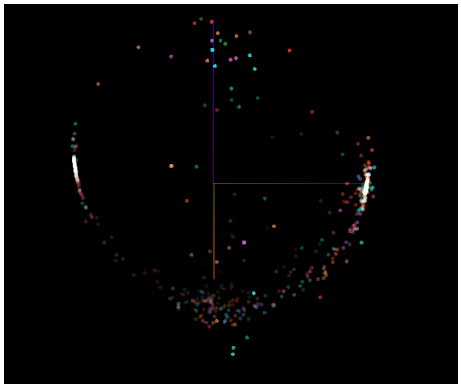
to identify the users with the same label. After analyzing figure 4.5 we understood hat this task was not going to be an easy one to solve.

4.3.1 Meta-level evaluation

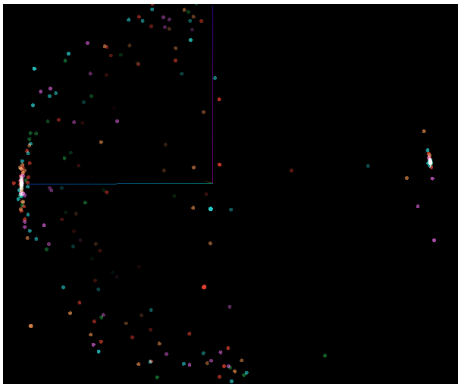
We are going to evaluate the tuned meta models using cross validation with 5 folds and a different variety of metrics: accuracy, precision, recall and all their averages. These metrics are acquired in the meta level and provide analysis for the performance of the meta models in each class. To evaluate the performance of the meta models in the base level, we are going to understand the impact of the meta models' predictions in the base level performance. The impact in base level performance will be in terms of NDCG and the intuition behind it is the following: assume for user u_i ALS has a NDCG of 0.4, BPR a NDCG of 0.35, and Most popular a NDCG 0.2. The best algorithm might be the ALS but if our meta model selects BPR, the overall performance for that user would be similar to the correct label ALS. Nevertheless, if the meta model selects Most Popular, the performance on that user will be affected.

The upper bound, the meta-level oracle, of the model is called perfect NDCG. This hypothetical meta model would always select the best algorithm for each user. This measure is an upper bound for our model, we have to be as close as possible to the oracle. Table 4.6 presents the different proportions of each algorithm, and the NDCG if the meta model only selected that algorithm. Using table 4.7 we can observe the impact of the perfect model in the base level. If the meta model has a perfect score, we would improve the performance by 71% in comparison with the single best algorithm (ALS).

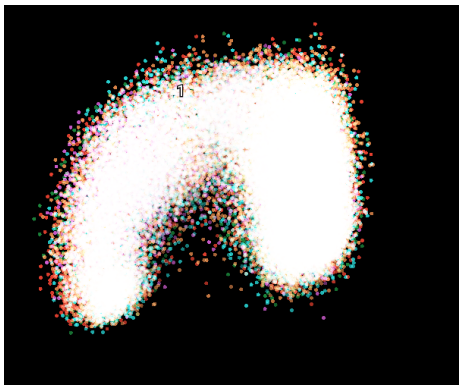
Figure 4.6 shows the meta level accuracy results, i.e. proportion of users for whom the best algorithm is recommended by the meta learners, using all the algorithms in all normalized datasets.



(a) CDAE 50



(b) CDAE 200



(c) VAE

Figure 4.5: Vizualization embeddings using PCA. Points with same label have the same color

Algorithms	# users	Percentage
ALS	34215	26,37
Most Popular	29088	22,42
BPR	25925	19,98
Zeroes	21277	16,40
LMF	19252	14,84
Total	129757	100

Table 4.6: Proportions of each class

Algorithms	NDCG Evaluation
Perfect NDCG	0,46
choosing only ALS	0,27
choosing only BPR	0,21
choosing only LMF	0,17
choosing only Most Popular	0,21

Table 4.7: Base level performance in terms of NDCG.

We can observe in figure 4.6 that all algorithms using the VAE dataset have the best overall performance. Table 4.8 presents a classification report of the best two overall models on the normalized VAE dataset. Both CDAE metadatasets have similar results in every algorithm. Table 4.11 presents the confusion matrix for the logistic regression meta model when using normalized CDAE50. From this table we can understand that the model predicts that every user should use the ALS algorithm. The results in the other algorithms are similar.

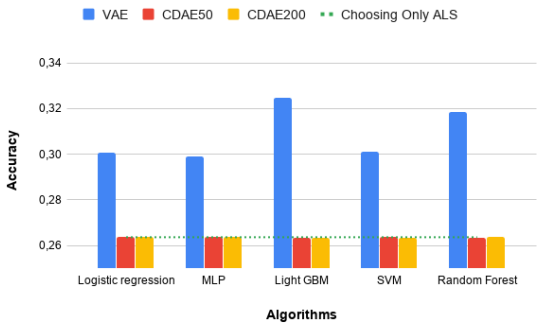


Figure 4.6: Meta Level Accuracy score on normalized datasets

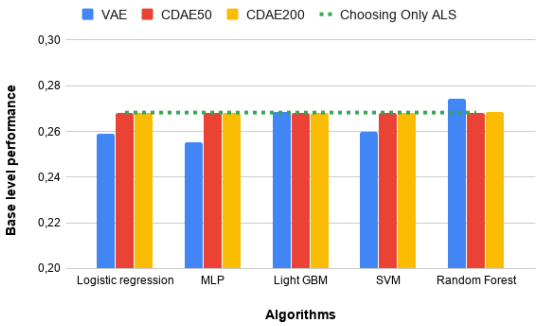


Figure 4.7: Base Level Performance on normalized datasets

Figure 4.7 presents the impact in base level of the different algorithms on normalized data. In this figure we can only detect marginal gains in the Random Forest and in the Light GBM models. For both CDAE metadasets we can observe that the impact in the base level performance is the same as if we only have the ALS algorithm. To evaluate the impact of the *zeroes* class we are going to remove all these users and analyze the performance of the meta models. By removing these users the distributions of classes will change when comparing to table 4.6. Table 4.12 presents the updated distribution of the the classes. Table 4.13 present the new base level

Table 4.8: Different meta models classification report on normalized VAE

	Precision	Recall	F1
ALS	0.31	0.55	0.40
BPR	0.31	0.21	0.25
LMF	0.23	0.02	0.04
MostPopular	0.32	0.31	0.31
Zeroes	0.40	0.40	0.40
–	–	–	–
Macro avg	0.31	0.30	0.28
Weighted avg	0.32	0.32	0.30

Table 4.9: Light GBM

	Precision	Recall	F1
ALS	0.30	0.75	0.42
BPR	0.36	0.13	0.19
LMF	0.40	0.0014	0.0028
MostPopular	0.34	0.21	0.26
Zeroes	0.40	0.30	0.34
–	–	–	–
Macro avg	0.36	0.28	0.24
Weighted avg	0.35	0.32	0.26

Table 4.10: Random Forest

	ALS	BPR	LMF	MostPopular	Zeroes
ALS	6837	0	0	4	2
BPR	5182	0	0	1	1
LMF	3847	0	0	2	1
MostPopular	5812	0	0	3	3
Zeroes	4249	0	0	3	4

Table 4.11: Logistic Regression meta model confusion matrix for the normalized CDAE50 dataset.

performances after removing the zeroes users.

Algorithms	# users	Percentage
ALS	34215	31,54
Most Popular	29088	26,81
BPR	25925	23,90
LMF	19252	17,75
Total	108480	100

Table 4.12: Proportions of each class

Algorithms	NDCG Evaluation
Perfect NDCG	0,55
choosing only ALS	0,32
choosing only BPR	0,25
choosing only LMF	0,20
choosing only Most Popular	0,26

Table 4.13: Base level performance in terms of NDCG.

Table 4.14: Classes percentages and NDCG evaluation after removing class zeroes

Figures 4.8 and 4.9 present the results for the accuracy and the base level performance for all algorithms in all embeddings techniques when removing the users with class *zeroes*. The results in these figures povide similar conclusions to the ones with the users *zeroes*. In the meta level accuracy all meta models using the VAE dataset achieved the best results. In the base level we only have marginal gains from the same two meta models: Light GBM and Random Forest.

After analyzing the previous results we decided to use a technique to balance the labels. We used the SMOTE technique [78], that uses interpolation to create synthetic data. SMOTE tries to address imbalanced datasets by generating synthetic data. Figures 4.10 and 4.11 present the results of the the different meta models when using all normalized datasets with syntethic data created by SMOTE. At the meta level, the VAE metadataset still has the best accuracy for all algorithms. However, both CDAE datasets have decreased their performance when comparing to

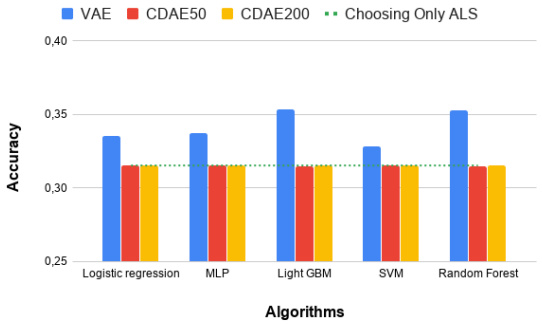


Figure 4.8: Accuracy score on normalized datasets, without zeroes

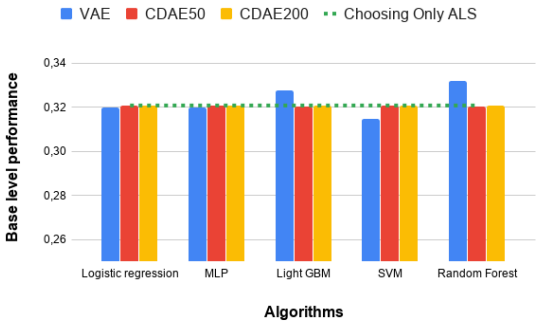


Figure 4.9: Base Level Performance on normalized datasets, without zeroes

the previous results. On the base level results presented in figure 4.11 we achieved a significant decrease in performance. When using SMOTE the results are, once again, consistent over all metadatasets, normalized or non normalized, and with or without zeroes. We are still running the SVM evaluation, although we expect the results to follow the other algorithms' results.

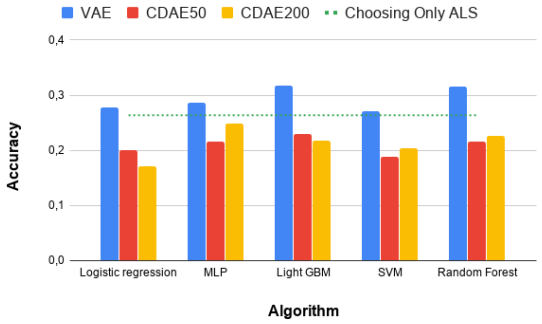


Figure 4.10: Accuracy score on normalized datasets,with SMOTE

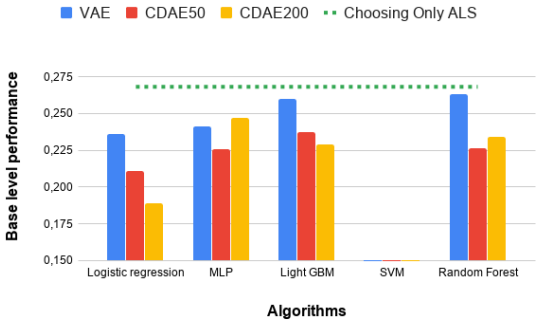


Figure 4.11: Base Level Performance on normalized datasets,with SMOTE

4.4 RQ1: Can we use Representation Learning to obtain useful metafeatures on the User Level?

This project addresses several new challenges. It is the first, to the best of our knowledge, that uses meta learning together with Representation Learning in order to solve the algorithm selection problem on user level. Using Representation Learning techniques to automatically retrieve metafeatures for each user, was firstly introduced in this work. This RQ will answer the question whether it is possible to retrieve expressive metafeatures about the users.

Figure 4.5 was obtained by using a dimension reduction technique which allowed the plotting of the dataset in another dimension space. These techniques reduced the metafeatures into three dimensions. If this task was trivial to solve, we could group the datapoints with the same label (same color). None of the figures present the points grouped by the label. When using the T-SNE

technique, the results are similar for all the datasets. However, for the PCA technique, the images are very different. Figure 4.5 presents a distinction between the metafeatures obtained from the three datasets. The visualization of both CDAE datasets are similar. The points are more focused on two points. In the VAE visualization the points are more disperse. This may indicate that the VAE metafeatures are more expressive (have more information about the data) than the CDAE.

We know that we have five classes, so we can assume that a completely random model would have an accuracy of 20%. Table 4.6 presents the distribution of the classes, where ALS is the biggest with a value of 26,37%. For our model to have an impact on solving this problem, we have to be better than a random model (that has an accuracy of 20%) and have to improve the result of only recommending one class (better than 26,37%). μ-cf2vec proposes to remove the biases presented in CF algorithms, hence the meta models have to predict more than one class. Figure 4.6, which presents the accuracy for the models with normalization, can help us answer this question.

In the CDAE datasets the models achieved an accuracy lower or equal to 26,37%. Table 4.11 presents the confusion matrix for the Logistic Regression model for the normalized CDAE50 dataset. The confusion matrix indicates that the model only recommends one class, *ALS*. The recall of this class is 1, while in the other classes it is around 0. The accuracy of the models using both CDAE metadatasets is the same as the percentage that the ALS class presents in table 4.6. The results of the other meta models are similar.

When using the models on the VAE metadataset, every meta model improves the accuracy score of 26,37%. The best model is the Light GBM with normalized data, achieving an accuracy of 32,48%. When analyzing the results in table 4.9 we can identify some aspects. The LMF class is the one with the worst overall score, which can be due to the low proportion of the class as stated in the table 4.6. This model has a best overall score in the class *zeroes*, even though it is the fourth in amount of appearances. The precision score and the F1 score for this class are the best in comparison with the other classes. In the metric recall it is in second place, just behind the most common class *ALS*. These results indicate that the *zeroes* class affect the performance of the model. Since ideally we would want to recommend different algorithms, i.e. it would be better if the the meta models have better performance in other classes instead of the *zeroes* class. There is another model in this same dataset that has almost the same accuracy but its recommendations spectrum is different.

The Random Forest meta model has an accuracy of 31,85%. While analyzing the results in the table 4.10 we have a new scenario in comparison with the previous meta model. Random Forest model has a better score when predicting the class *ALS*. The metrics F1 and Recall on the class *ALS* are the overall best. The overall performance of this model has improved in comparison with Light GBM.

In table 4.6 we can observe that the classes are not evenly distributed. To overcome this problem, we used the SMOTE technique to create synthetic data. Although we expected to achieve better results with the use of this technique, the results presented in figure 4.10 do not indicate that. In both figures 4.10 and 4.11 the performance is consistently worst when compared to the figure

4.6 and 4.7, both of them without the synthetic created data.

Based on these results we can deduce that on the meta level we can achieve an improvement in the performance. It is possible to use Representation Learning techniques to make personalized algorithm selection in recommender systems. From these results we can infer some points: the best overall meta model is the Light GBM (it has the best accuracy and the best performance for the *zeroes* class), although the Random Forest has its own advantages (it has a good performance for the *ALS* class). Through Light GBM, we can infer that, although class *zeroes* is the fourth most common class, it has the best performance for this model. That shows how this class can mislead the results.

4.5 RQ2: Which Representation Learning Technique is the best?

As stated previously this project introduces a completely new approach to the algorithm selection problem on the user level. We proposed to use meta learning combined with Representation Learning, in order to retrieve meta features. Later, these meta features will be used by the meta model to try to predict the best algorithm for each user. There are many different Representation Learning techniques as we stated in chapter 2. For this project we have chosen two techniques: Variational Auto-encoders and Denoising Auto-encoders. Even though both share the same foundation, the auto-encoders, these two techniques are different as we discussed in chapter 2.

From these two techniques we have three experimental setups as stated in chapter 3. The VAE metadaset used the user embeddings retrieved with the Variational Auto-encoder and has a size of 200. The CDAE dataset is created with users representations acquired by the Denoising Auto-encoders. We have two different datasets from these techniques: CDAE50 and CDAE200. The original paper of this implementation found out that the optimal number of factors is 50, therefore we started by obtaining the results with 50 factors. However, the VAE dataset is composed by representations with size 200. To have an head-to-head comparison between the two techniques we retrieved the CDAE representations with size 200.

To identify which of the techniques is more suitable to solve this problem, we have to understand what their performance is. We have different techniques as stated previously: VAE and CDAE. From the figures presented before (4.6 and 4.7) the VAE metadataset achieves better results in every single meta model. These results are consistent in all metadatasets: normalized or non normalized, with or without zeroes and with or without the SMOTE technique. The results presented indicate that the Variational Auto-encoder is the better implementation in this situation.

4.6 RQ3: What is the impact on the base level performance achieved by μ-cf2vec?

μ-cf2vec is a framework which aims to improve the recommendations' performance of every user. As stated previously, the meta is able to predict the best algorithm for each user with an accuracy

higher than the baselines. However, we need to look deeper into the performance affected by the users, in order to understand whether the model can improve the performance for the users or not.

In table 4.7 we have the upper and lower bounds for our model. The *Perfect NDCG* is the upper bound of our model. This value is the average of NDCG of the best algorithm (with higher NDCG) for each user. The other rows represent the value of the NDCG of a meta model that has only selected one single algorithm (ALS, BPR, etc). One thing we have to revisit is that the class *zeroes* has zero NDCG in every algorithm. As stated in chapter 3 μ-cf2vec will recommend the Most Popular algorithm to users identified as *zeroes*. This is the policy of the framework to address users that we don't have enough information to make good recommendations. Thus, the goal of our meta model is to improve the value of 0,27 presented in table 4.7. If we improve this score, we can infer that the performance is better than if we only used the ALS.

From figure 4.7 we can observe some interesting aspects. At the meta level the VAE dataset was always the best. However, at the base level, only Light GBM and Random Forest achieved better results with VAE. For the other algorithms the CDAE datasets achieved a better perfromance when comparing to the VAE performance.

When using the VAE dataset, which can be more expressive as stated previously, the meta models have a wide spectrum of recommendations. While in the meta level we could have improvements by using the VAE and recommending different algorithms, in the base level it can be worse. For instance, the MLP meta model has an accuracy of around 30% using the VAE dataset compared to an accuracy of around 26% for both CDAE datasets. From this result we could expect that in the base level the performance would also be better when using the VAE dataset. However, the figure 4.7 indicates the opposite: the base level performance in MLP using the VAE is worse than both CDAE. While using the VAE dataset the MLP is recommending different algorithms, in the CDAE datasets it is only recommending *ALS*. In this case it achieves a better performance.

When inspecting figure 4.7, we can observe two models where the base level performance is better than if we only used *ALS*. The Random Forest and the Light GBM models both achieve marginal gains over recommending the best single class.

From these results we only achieved marginal gains in the base level performance. Even though the VAE dataset achieved the best meta level performance, this was not true on the base level. In fact, in the base level (in some cases) it is better to use the CDAE datasets. Sometimes it is better to select one single algorithm to achieve the best performance, than selecting multiple algorithms.

4.7 RQ4: What is the impact of the zeroes class on μ-cf2vec?

As expected, there are some users for which none of the algorithms considered provide any useful recommendation. These users belong to the class *zeroes*, since the maximum NDCG that all base learners algorithms' could achieve was zero.The users with this class are a group of users that we do not have enough information on to solve their problem. This can be related to the Cold Start Problem.

In research question 4.4 we presented that even though the *zeroes'* class is in fourth place, in terms of occurrences, most of the models could detect this class more easily than the BPR and Most Popular (although both are more frequent). These results might indicate that these users are misleading the models performance. μ-cf2vec aims to recommend the best algorithm for each user. Since the *zeroes* users do not have good performance in any algorithm, we decided to remove them. This will allow us to understand their impact on the meta models performance. Table 4.12 presents the proportions of data after we removed all the users that belong to the class of *zeroes*. This analysis was made to understand the impact of the class *zeroes*.

Figures 4.8 and 4.9 present the accuracy and base level performance for each algorithm in all metadatasets without users belonging to class *zeroes*. Since we removed the users labeled as *zeroes*, the distribution of the classes has changed, as we can see in table 4.12. When comparing to the distribution with all users (table 4.6) we can highlight some points. The perfect meta model would increase the NDCG evaluation by 20%. The base level lower bound is the most common class and it still is *ALS* with a percentage of 31,54%. The overall improvement in the base level performance was around 20%. These values by themselves show the impact of the users in the overall metrics. These are the reference results we have to use when analyzing the ones without the users labeled as *zeroes*.

The meta level accuracy scores are similar to the results with all users. The VAE dataset still has the best overall performance when comparing to the CDAE. The best models are the Light GBM and the Random Forest achieving an accuracy of around 35%. This accuracy showed some improvements, however, we have to recall that the new percentage of the most common algorithm *ALS* is 31,54%. From these results the improvement is not significant. For the CDAE datasets the results are similar, and the meta models only recommend the most common class, achieving an accuracy of around 31,54%.

The base level performance results in figure 4.9 are similar to the results with all users (shown by figure 4.7). The only two models that improved the base level performance are the Random Forest and the Light GBM, when comparing to the ones that only used the most common class. These marginal gains where achieved by using VAE dataset, which reinforces the hypothesis that the VAE technique is able to achieve better results. One aspect where the results in figure 4.9 differ from the ones in figure 4.7 is the fact that the Logistic Regression and the MLP are closer to the lower boundary. This is explained by the fact that False *zeroes* users identified before are now being identified as *ALS*. The CDAE dataset has similar results when using all users. All the models are recommending the *ALS* class, hence the base level performance will be around the same as the *ALS* class.

We can conclude, from these results and the ones in the research question 4.4, that this organically created class can bring problems to this framework. There are different ways to try to minimize the effects of this class. They will be discussed in the future work.

4.8 Summary

This chapter presented an implementation of the proposed framework μ-cf2vec. This implementation used a set of different base learners and meta learners. We used two RL techniques to automatically retrieve the users' metafeatures: Variational Auto-encoders and the Denoising Auto-encoders. We answered four research questions in order to understand the performance of this implementation. As stated previously, we can achieve consistent gains in the meta level by using the metafeatures acquired by RL techniques. From the two RL techniques, the VAE was the one with better performance. The gains in the base level are marginal. The class *zeroes* may have a negative impact on the performance of the meta models.

Chapter 5

Final Remarks

This dissertation presents a new framework to tackle the meta learning Algorithm Selection on user-level and an empirical study of it. The proposed framework is the first of its kind. μ-cf2vec proposes to use Representation Learning techniques to automatically generate user representation that will be used as metafeatures. This approach removes the need of manually choosing the metafeatures. This work also presents an implementation of the framework previously introduced.

μ-cf2vec proposes to solve the algorithm selection on the user level. To do so, it uses a Meta Learning approach by learning from prior experience. The meta learners leveraged from a dataset which is built with the users metafeatures and the user label. The metafeatures are automatically retrieved by using Representation Learning techniques. Using these techniques we removed the need of manually selecting the metafeatures, a process that can be slow and inefficient. The meta target is related to the algorithms we want to recommend. The meta models use the metafeatures to recommend the best algorithm for each user.

This implementation of the framework generated metafeatures from two Representation Learning techniques: Variational Auto-encoders and Denoising Auto-encoders. In this implementation the metatarget is obtained by selecting the name of the algorithm that achieved the best performance for each user. We used a set of 5 meta models to try to solve the algorithm selection problem on the user level. In every meta model the dataset with metafeatures obtained using Variational Auto-encoders achieved the best meta level results. From this study we can conclude that we are able to achieve meta level improvements by using Representation Learning (using VAE). The meta models Light GBM and Random Forest are the ones with the best overall performance. In the base level we only achieved marginal gains. The organically created *zeroes* class has a negative impact on the meta models performance. We can relate this class to the cold start problem. This implementation is the first of its kind when addressing Personalized Algorithm Selection in Recommender Systems.

5.1 Limitations and Future Work

μ-cf2vec proposes an innovative approach to solve Personalized Algorithm Selection in Recommender Systems. This implementation of the framework has some limitations that can lead to future work:

- In the implementation of the μ-cf2vec framework we only used a single dataset. To validate this framework, we have to test it in more datasets.
- We only used two Representation Learning techniques. However, as stated in chapter 2 there are many different techniques. The choice of these techniques and its results may have an impact on the performance. The techniques we used achieve state of the art results in Collaborative Filtering tasks. However, they are not designed for meta learning tasks. This limitation is reinforced by the results presented in the low-space visualization and when using the SMOTE technique. Using these results we can understand that the metafeatures cannot extract a meaningful information about the task. In future work we should use different RL techniques.
- We did a grid search to find the best parameters, however there is a chance that the hyperparameters we have chosen aren't the best for the task we aim to solve. Not having the optimal parameters for a meta model can have a negative impact on the performance.
- The base learners selected are a limitation of this work. The framework μ-cf2vec wants to remove the bias of the CF algorithms. We have to choose a variety of base learners to leverage different biases. In this implementation we use four different base learners, however, there are some users that are affected by the biases in the algorithms. To address this limitation we should have a more diverse group of base learners.
- The metatarget selection is another limitation. The framework μ-cf2vec wants to remove the bias of the CF algorithms. In this implementation we decided to define the metatarget as the name of the base learner with the best performance for each user. However, this decision might not have been the optimal one. We can try different metatargets definitions.
- The evaluation technique is another aspect on which we should focus. We have used the common 5-fold cross validation technique, however, it might not be the one more suitable to use for the characteristics of the data.

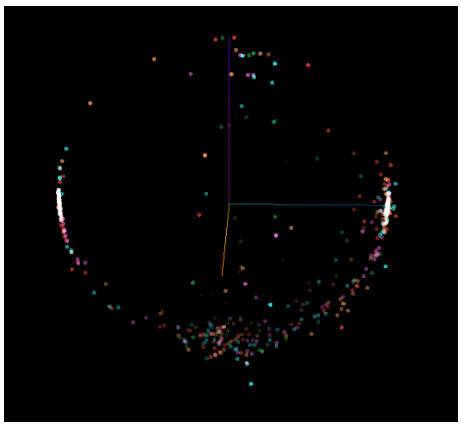
On the general framework we will focus on addressing the *zeroes* problem in more depth. We can address the class *zeroes* in a multi level problem. Instead of our meta model selecting the best algorithm, in the first level the meta model can be used to distinguish if a user belongs to the class *zeroes* or not. On the second level the meta model will predict the best algorithm for the users that the system did not identify as *zeroes*.

Appendix A

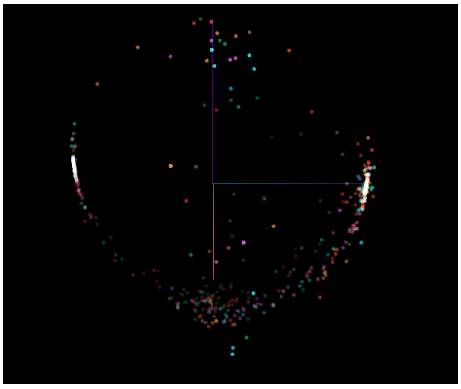
Results

A.1 Embeddings Visualizations

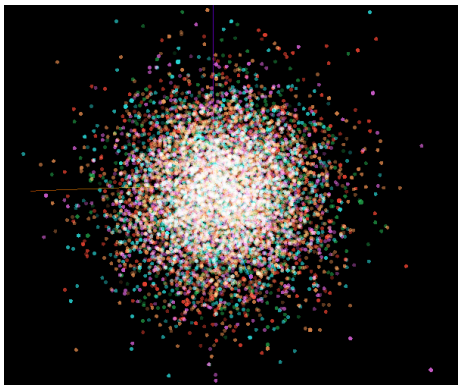
In figure A.1 we can observe different visualizations for the data obtained with CDAE with 50 factors. The visualization is similar for both normalization and no normalization. If the task was trivial to solve we would see the points grouped by its label (same color).



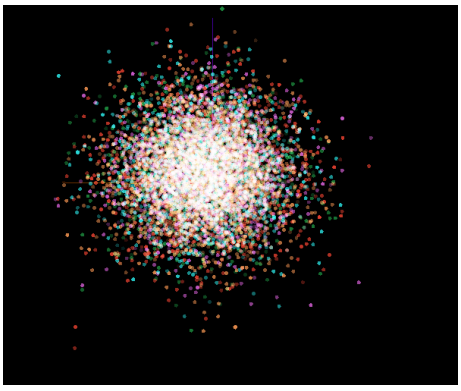
(a) PCA with normalization



(b) PCA without normalization



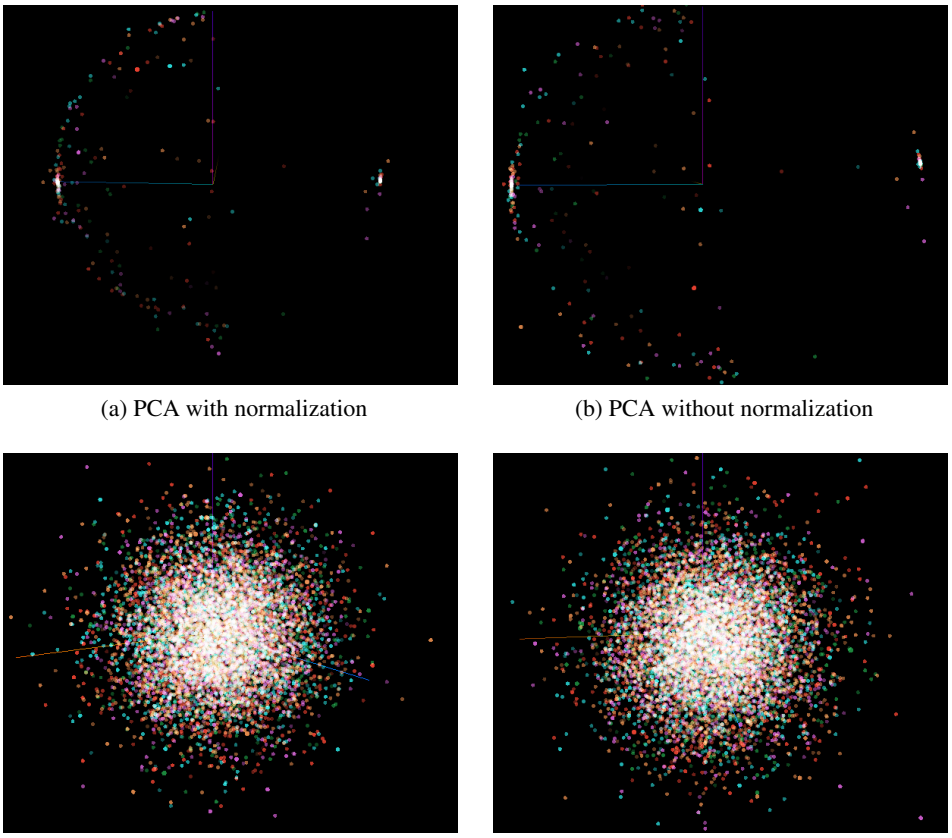
(c) T-SNE with normalization. Iteration 5.



(d) T-SNE without normalization

Figure A.1: Vizualization of the meta dataset built with CDAE 50 factors using PCA and T-SNE.

Figure A.2 presents a visualization for the metadataset with 200 factors obtained with the CDAE. These results are similar to the ones in figure A.1. We cannot see any clustering of the points by its label.



(a) PCA with normalization

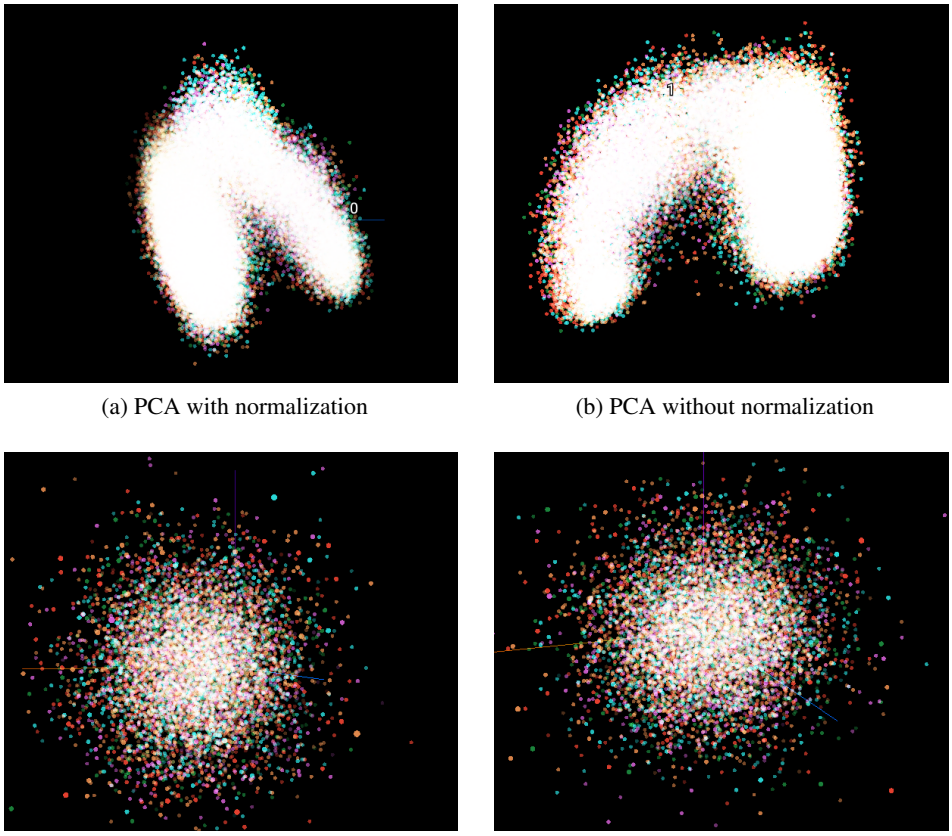
(b) PCA without normalization

(c) T-SNE with normalization. Iteration 5.

(d) T-SNE without normalization

Figure A.2: Vizualization of the meta dataset built with CDAE 200 factors using PCA and T-SNE.

Figure A.3 presents the visualization of the VAE metadataset, the last dataset obtained in this implementation. Once again the results are consistent with the ones obtained previously. Using the the T-SNE technique the results are similar in all the three datasets. The PCA visualization is diferent on the VAE dataset. In this dataset the PCA visualization represents the points more dispersed, while in both CDAE dataset the points are more focused in two points. After analysing these figures we might infer that the VAE dataset can retrieve better metafeatures when comparing with both CDAE metafeatures.



(a) PCA with normalization

(b) PCA without normalization

(c) T-SNE with normalization. Iteration 5.

(d) T-SNE without normalization

Figure A.3: Vizualization of the meta dataset built with VAE with 200 factors using PCA and T-SNE.

A.2 Meta Models evaluations

Figure A.4 presents the results for all non normalized datasets. These results are similar with the ones in chaper 4. The VAE metadaset achives the best results. These results shows that we can achieve gains in the meta level.

The impact on the base level performance present in figure A.4 are similar to the ones in the previous chapters. We can only achieve marginal gains. The best meta model is once again the random forest.

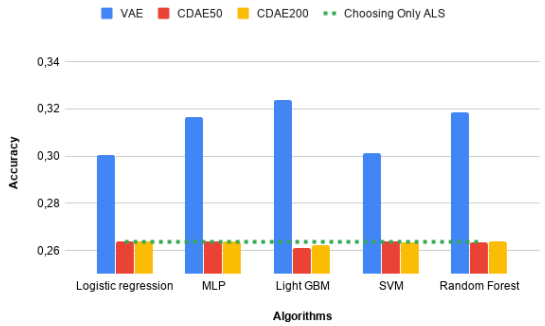


Figure A.4: Accuracy score on non normalized datasets

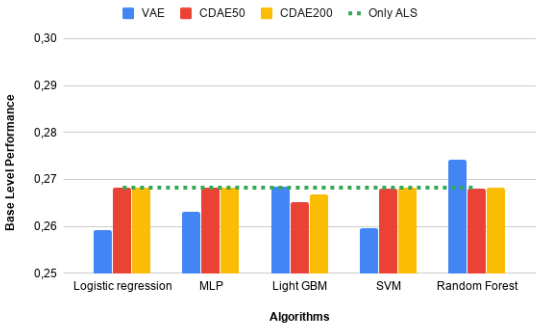


Figure A.5: Base Level Performance on non normalized datasets

After analysing the previous results we decided to the SMOTE technique, that can create synthetic data. When analysing figure A.6 we can once again understand that the VAE dataset is the one with better performance. For all meta learners the VAE dataset can achieve better accuracy than if only choosing the ALS algorithm. However, for the others datasets the accuracy has decreased significantly. This may indicicate that the metafeatures cannot create a clear representation about the user. If we could easily relate the metafeatures, then the SMOTE technique would generate an improvement on the results.

Figure A.7 presents the results for the impact on the base level performance. Using this technique to generate synthetic data, decreased the performance in almost every meta learner. Even the CDAE datasets which previously only predicted the ALS class, in this new scenario they tried to select more than one class. With these results we can conclude that sometimes may be better to just select one algorithm instead of trying to select different ones.

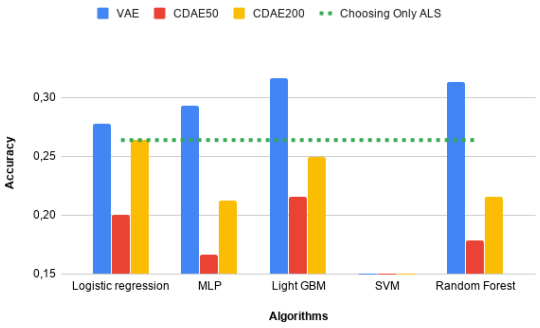


Figure A.6: Accuracy score on non normalized datasets, with smote

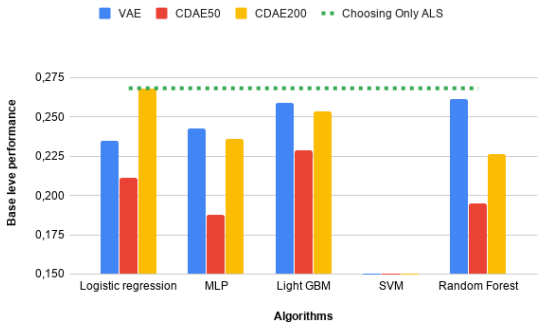


Figure A.7: Base Level Performance on non normalized datasets, with smote

References

[1] Kate Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41, 12 2008. doi:10.1145/1456650.1456656.

[2] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 2016. URL: https://doi.org/10.1145/2843948.

[3] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, 2010. URL: https://doi.org/10.1145/1864708.1864770.

[4] Pedram Bashiri. Recommender systems: Survey and possible extensions, 08 2018. doi:10.13140/RG.2.2.15714.63685.

[5] Joeran Beel and Siddharth Dinesh. Real-world recommender systems for academia: The pain and gain in building, operating, and researching them [long version], 2017. arXiv:1704.00156.

[6] Panagiotis Adamopoulos and Alexander Tuzhilin. On unexpectedness in recommender systems: Or how to better expect the unexpected. *ACM Trans. Intell. Syst. Technol.*, 5(4), December 2014. URL: https://doi.org/10.1145/2559952, doi:10.1145/2559952.

[7] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329 – 354, 1979. URL: http://www.sciencedirect.com/science/article/pii/S0364021379800129, doi:https://doi.org/10.1016/S0364-0213(79)80012-9.

[8] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., USA, 1989.

[9] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, page 47–51, New York, NY, USA, 2010. Association for Computing Machinery. URL: https://doi.org/10.1145/1869446.1869453, doi:10.1145/1869446.1869453.

[10] 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*, 17(6):734–749, 2005.

[11] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques, 2009.

[12] Michael D. Ekstrand, Mucun Tian, Ion Madrazo Azpiazu, Jennifer D. Ekstrand, Oghenemaro Anuyah, David McNeill, and Maria Soledad Pera. All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness. In *FAT*, 2018.

[13] Masoud Mansoury, Bamshad Mobasher, Robin Burke, and Mykola Pechenizkiy. Bias disparity in collaborative recommendation: Algorithmic evaluation and comparison, 2019. arXiv:1908.00831.

[14] Joaquin Vanschoren. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018. URL: http://arxiv.org/abs/1810.03548, arXiv:1810.03548.

[15] Ricardo Vilalta, Christophe Giraud-Carrier, and Pavel Brazdil. *Meta-Learning - Concepts and Techniques*, pages 717–731. 07 2010. doi:10.1007/978-0-387-09823-4_36.

[16] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012. URL: http://arxiv.org/abs/1206.5538, arXiv:1206.5538.

[17] Liwei Wu. Advances in collaborative filtering and ranking, 2020. arXiv:2002.12312.

[18] Yibo Chen, Chanle Wu, Ming Xie, and Xiaojun Guo. Solving the sparsity problem in recommender systems using association retrieval. *JCP*, 6:1896–1902, 08 2011. doi:10.4304/jcp.6.9.1896-1902.

[19] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, June 2009.

[20] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery. URL: https://doi.org/10.1145/371920.372071, doi:10.1145/371920.372071.

[21] Charu C. Aggarwal. *Neighborhood-Based Collaborative Filtering*. Springer International Publishing, Cham, 2016. URL: https://doi.org/10.1007/978-3-319-29659-3_2, doi:10.1007/978-3-319-29659-3_2.

[22] Al Mamunur Rashid, Shyong K. Lam, George Karypis, and John Riedl. Clustknn: A highly scalable hybrid model & memory-based cf algorithm. In *KDD 2006*, 2006.

[23] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004. URL: https://doi.org/10.1145/963770.963776, doi:10.1145/963770.963776.

[24] Michal Kompan and Maria Bielikova. Content-based news recommendation. volume 61, pages 61–72, 09 2010. doi:10.1007/978-3-642-15208-5_6.

[25] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation - J DOC*, 60:503–520, 10 2004. doi:10.1108/00220410410560582.

[26] Shaivya Kaushik and Pradeep Tomar. Evaluation of similarity functions by using user based collaborative filtering approach in recommendation systems. *International Journal of Engineering Trends and Technology*, 21:194–200, 03 2015. doi:10.14445/22315381/IJETT-V21P234.

[27] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[28] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science*, 49:136 – 146, 2015.

[29] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. pages 263–272, 12 2008. doi:10.1109/ICDM.2008.22.

[30] Gábor Takács, István Pilászy, and Domonkos Tikk. Applications of the conjugate gradient method for implicit feedback collaborative filtering. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, page 297–300, New York, NY, USA, 2011. Association for Computing Machinery. URL: https://doi.org/10.1145/2043932.2043987, doi:10.1145/2043932.2043987.

[31] Michael Lublin Haoming Li, Bangzheng He. Cme 323: Distributed algorithms and optimization, 2015. URL: http://stanford.edu/~rezab/classes/cme323/S15/notes/lec14.pdf.

[32] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. arXiv:1609.04747.

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

[34] Ethen. URL: http://ethen8181.github.io/machine-learning/recsys/4_bpr.html.

[35] Christopher C. Johnson. Logistic matrix factorization for implicit feedback data. 2014.

[36] Jon Herlocker, Joseph Konstan, Loren Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 01 2004. doi:10.1145/963770.963772.

[37] Harald Steck. Evaluation of recommendations: Rating-prediction and ranking. pages 213–220, 10 2013. doi:10.1145/2507157.2507160.

[38] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. A theoretical analysis of ndcg type ranking measures, 2013. arXiv:1304.6480.

[39] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. *CoRR*, abs/1708.05031, 2017. URL: http://arxiv.org/abs/1708.05031, arXiv:1708.05031.

[40] Florian Strub, Jérémie Mary, and Romaric Gaudel. Hybrid collaborative filtering with neural networks. *CoRR*, abs/1603.00806, 2016. URL: http://arxiv.org/abs/1603.00806, arXiv:1603.00806.

[41] ThaiBinh Nguyen and Atsuhiro Takasu. NPE: neural personalized embedding for collaborative filtering. *CoRR*, abs/1805.06563, 2018. URL: http://arxiv.org/abs/1805.06563, arXiv:1805.06563.

[42] Suvash Sedhain, Aditya Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering, 05 2015. doi:10.1145/2740908.2742726.

[43] Oleksii Kuchaiev and Boris Ginsburg. Training deep autoencoders for collaborative filtering, 2017. arXiv:1708.01715.

[44] Yao Wu, Christopher DuBois, Alice Zheng, and Martin Ester. Collaborative denoising autoencoders for top-n recommender systems, 02 2016. doi:10.1145/2835776.2835837.

[45] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering, 2018. arXiv:1802.05814.

[46] Daeryong Kim and Bongwon Suh. Enhancing vaes for collaborative filtering. *Proceedings of the 13th ACM Conference on Recommender Systems*, Sep 2019. URL: http://dx.doi.org/10.1145/3298689.3347015, doi:10.1145/3298689.3347015.

[47] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374, 2016.

[48] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. pages 14–36, 01 2012. doi:10.1007/978-3-642-33275-3_2.

[49] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, page 37–50. JMLR.org, 2011.

[50] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[51] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery. URL: https://doi.org/10.1145/1390156.1390294, doi:10.1145/1390156.1390294.

[52] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

[53] Siddharth Misra and Hao Li. Chapter 7 - deep neural network architectures to approximate the fluid-filled pore size distributions of subsurface geological formations. In Siddharth Misra, Hao Li, and Jiabo He, editors, *Machine Learning for Subsurface Characterization*, pages 183 – 217. Gulf Professional Publishing, 2020. URL: http://www.sciencedirect.com/science/article/pii/B9780128177365000077, doi:https://doi.org/10.1016/B978-0-12-817736-5.00007-7.

[54] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. URL: http://dx.doi.org/10.1561/2200000056, doi:10.1561/2200000056.

[55] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, Cambridge, MA, USA, 1986.

[56] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. arXiv:1301.3781.

[57] Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine Learning*, 81, Issue 1:21, 2010. URL: http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s10994-010-5198-3&sa_campaign=Email/ACE/Paginated.

[58] Tiago Cunha, Carlos Soares, and André C. P. L. F. de Carvalho. cf2vec: Collaborative filtering algorithm selection using graph distributed representations. *CoRR*, abs/1809.06120, 2018. URL: http://arxiv.org/abs/1809.06120, arXiv:1809.06120.

[59] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey, 2020. arXiv:2004.05439.

[60] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019. arXiv:1911.02685.

[61] John R. Rice. The algorithm selection problem**this work was partially supported by the national science foundation through grant gp-32940x. this chapter was presented as the george e. forsythe memorial lecture at the computer science conference, february 19, 1975, washington, d. c. volume 15 of *Advances in Computers*, pages 65 – 118. Elsevier, 1976. URL: http://www.sciencedirect.com/science/article/pii/S0065245808605203, doi:https://doi.org/10.1016/S0065-2458(08)60520-3.

[62] Tiago Cunha, Carlos Soares, and Andre de Carvalho. Selecting collaborative filtering algorithms using metalearning. pages 393–409, 09 2016.

[63] Hadi S. Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2vec: Learning dataset meta-features, 2019. arXiv:1905.11063.

[64] Andrew Collins, Jöran Beel, and Dominika Tkaczyk. One-at-a-time: A meta-learning recommender-system for recommendation-algorithm selection on micro level. *CoRR*, abs/1805.12118, 2018. URL: http://arxiv.org/abs/1805.12118, arXiv:1805.12118.

[65] Mi Luo, Fei Chen, Pengxiang Cheng, Zhenhua Dong, Xiuqiang He, Jiashi Feng, and Zhenguo Li. Metaselector: Meta-learning for recommendation with user-level adaptive model selection, 2020. arXiv:2001.10378.

[66] Group lens. Movielens 20m dataset. URL: http://www.sciencedirect.com/science/article/pii/S0950705111001882, doi:http://dx.doi.org/10.17632/n6sjkpy87f.5#file-8b2203cf-ad2a-4b48-8de4-909387df278b.

[67] Mi Zhang, Jie Tang, Xuchen Zhang, and Xiangyang Xue. Addressing cold start in recommender systems: A semi-supervised co-training algorithm. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR 14, page 73–82, New York, NY, USA, 2014. Association for Computing Machinery. URL: https://doi.org/10.1145/2600428.2609599, doi:10.1145/2600428.2609599.

[68] Ben Frederickson. Fast python collaborative filtering for implicit datasets, 2017. URL: http://dx.doi.org/10.17632/n6sjkpy87f.5#file-8b2203cf-ad2a-4b48-8de4-909387df278b.

[69] NVIDIA. URL: https://github.com/NVIDIA/DeepLearningExamples/tree/master/TensorFlow/Recommendation/VAE-CF.

[70] F. Iuri. URL: https://github.com/fabioiuri/DRecPy.

[71] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives, 2014. arXiv:1407.0202.

[72] Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1):185 – 234, 1989. URL: http://www.sciencedirect.com/science/article/pii/0004370289900490, doi:https://doi.org/10.1016/0004-3702(89)90049-0.

[73] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[74] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. URL: https://doi.org/10.1023/A:1010933404324, doi:10.1023/A:1010933404324.

[75] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017. URL: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[76] Deep AI. URL: https://deepai.org/machine-learning-glossary-and-terms/random-forest.

[77] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. URL: http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[78] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL: http://jmlr.org/papers/v18/16-365.html.