University of Zurich[UZH]

Utilizing Eccentric User Preferences and Negative Feedback to Improve Recommendation Quality

Thesis February 1, 2018

Sandro Luck
of Zürich ZH, Switzerland

Student-ID: 13-927-769
sandro.luck@gmx.ch

Advisor: **Bibek Paudel**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisors Bibek Paudel for his support, help, expertise and time he dedicated to me. I thank Prof. Dr. Abraham Bernstein, the head of the Dynamic and Distributed Information Systems Group at the University of Zurich, for making this project possible.

Zusammenfassung

Die Benutzerzufriedenheit in Recommender-Systemen hängt von vielen anderen Faktoren als dem Recall ab. Benutzer schätzen auch Qualitäten wie die Auswahlmöglichkeiten, Neuheit und Vielfalt.
In dieser Arbeit untersuchen wir zwei verschiedene Bereiche, um die Qualität und Vielfalt von Empfehlungen mithilfe von Collaborative Filtering zu verbessern. Im ersten Problem konzentrieren wir uns auf die Two-Class Collaborative Filterung, bei der das Ziel darin besteht, mehr positive Items zu empfehlen und gleichzeitig die Anzahl der negativen Items an der Spitze der Empfehlungsliste zu reduzieren. Die Modellierung des Nutzerverhaltens unter Berücksichtigung ihrer negativen Präferenzen hat gezeigt, dass sie vielfältigere und genauere Empfehlungen liefert. In dieser Arbeit erweitern wir das kürzlich entwickelte Collaborative Metric Learning, indem wir negative Entscheidungen modellieren. Wir konnten mit experimentellen Resultaten zeigen, dass unsere Methode in der Lage ist, die Qualität zu verbessern und die Anzahl der negativen Empfehlungen zu reduzieren. Im zweiten Problem betrachten wir das Problem der Verbesserung der Empfehlungsvielfalt. Nicht alle Benutzer bevorzugen Nischen-Empfehlungen in gleichem Masse, und es ist wichtig, die Empfehlungen entsprechend zu variieren. Wir erforschen das Konzept der Item Controvery und entwickeln eine neue Methode zur Generierung von Empfehlungen. Unsere Experimente zeigen, dass unsere Methode in der Lage ist, die Empfehlungen zu diversifizieren und in den meisten Fällen eine ähnliche oder bessere Genauigkeit zu erzielen.

Abstract

User satisfaction in Recommender Systems is dependent on many factors other than prediction accuracy. People also value qualities like variety, novelty and diversity. In this work, we explore two different areas to increase the quality and diversity of recommendations using well known Collaborative Filtering techniques.
In the first problem, we focus on Two-Class Collaborative Filtering, where the goal is to recommend more positive items, while reducing the number of negative items at the top of recommendation lists. Modeling user behavior by accounting for their negative preference has shown to produce more diverse and accurate recommendations. In this work, we extend the recently developed Collaborative Metric Learning by modeling negative choices. We show with experimental results on openly available datasets that our method is able to improve recommendation quality and reduce the number of negative recommendations at the top. In the second problem, we look at the problem of improving recommendation diversity. Not all users prefer niche items to the same extent, and it is important to diversify recommendations accordingly. We explore the concept of item controversy and eccentricity and develop a new method to recommend nice items to users based on their inclination to such items. Our experiments show that our method is able to diversify the recommendations while achieving competitive or better accuracy in most cases.

Table of Contents

1

Introduction

Machine learning has been used in a variety of fields and improving on these system adds great value to companies and individuals. Research and development in these areas are therefore useful and desirable.
Machine Learning can be used to improve recommendations generated by Recommender Systems. Recommender Systems are used in a variety of fields from online-shopping to music and movies. They have a big impact on how items are perceived and displayed to users. These offer their users a wide variety of advantages such as decreasing the search time and improving the user satisfaction.

To further improve these systems we recommend looking closer at the negative feedback and items that are not popular. This can further improve the value the system adds to the ecosystem it is used in. Using negative feedback and so-called Eccentric items, we created systems which aim at improving user satisfaction.

1.1 Recommender Systems

Recommender Systems are software tools and frameworks which provide suggestions items to users [Ricci et al., 2011]. The suggestions for the items might be in different fields. These suggestions are often related to a decision-making process, e.g. what item to buy, listen or watch. The term, "item" is the general term used to describe what object is recommended to the user e.g. product, movie or book. The approaches and techniques that are used for different classes of items might vary depending on the userbase and items that are recommended.

1.1.1 Collaborative Filtering

Collaborative filtering (CF) is an approach used by Recommender Systems [Ricci et al., 2011]. The general concept is to collect user interactions on items which can be classified as either positive, negative or a scale in between. This is then used to predict user interactions on different items in the future (e.g. recommend movies the user might like).
Collaborative Filtering systems usually need big amounts of user-item relations to perform well. Several methods for applying collaborative filtering exist and many people are

actively working on creating more. We will discuss several of these methods in the following sections and various combinations of them. In a general sense, collaborative filtering is a process where users collaborate to filter information [Sammut and Webb, 2011]. Typically such a process needs big collections of user-item relationships, often called datasets, which can be from various areas such as movies, books, music, financial data and environmental data.

1.1.2 Cold Start Problem

A problem that often arises when designing Recommender Systems is the so-called cold start problem. The cold start problem describes the situation in which a small amount or no information about users or items are present. Thus predictions can only be made with a high uncertainty. Often very specific algorithms and techniques can be used, which are good under these constraints. Paying attention to this can improve user satisfaction significantly.

1.1.3 Data sparsity

In commercial usage, Recommender Systems are based on large datasets and as a result, the user-item matrix that is used can become very big. While generally there are algorithms that operate on sparse matrices to speed the process up, some of these algorithms are only available for normal matrices. This aspect slows down learning and increases the need for memory on the system. Also, the cold start problem is related to this aspect since a new user will need to rate a sufficient amount of items to teach he Recommender Systems how to capture his preferences. Often new items have a similar problem, namely getting rated by enough users before being recommendable to other users. These problems have to be addressed and minimized to ensure a Recommender System's recommendations are beneficial for a user.

1.2 Metric Learning

One recent class of methods used for Recommender systems is based on metric learning and have shown their usage in multiple applications. Metric learning (ML) algorithms learn distance metrics that capture relations between data.
The goal is to learn a metric or embedding that assigns a low distance value to similar items and a big distance value to dissimilar items. These embeddings are n-dimensional and usually contain an embedding for the users and an embedding for the items. Generally, with more dimensions, the embeddings can capture the user item relations better. To capture this relation a metric or pseudo metric is learned. To achieve this, the program tries to minimize the distances between items and users which belong together e.g. which the user liked.
In the following document, we will also describe how the same approach can be used to increase the distance between items and user, which do not belong together e.g. the user

disliked or is likely to be disliked. The general goal of such an approach is to group items and users in n-dimensional space. The grouping can then be used to recommend items, which a user might not have consumed yet. Such an approach can consider multiple things such as likes, dislikes, features or also data related to the person.

1.2.1 Metrics and Distances

The understanding of distance in combination with similarity between users and items can add great value to machine learning algorithms including K-nearest neighbor, K-means, Collaborative Metric Learning and SVMs [Hsieh et al., 2017]. In this work, we will look into how the distance can be used to capture a relationship between users and items. Especially how to use negative feedback generated by the user to improve the value of the system. Given pairs of negative and positive item-user pairs, the goal of the system is to learn a distance that respects and predicts these relations.

1.3 Tensorflow

Tensorflow is an open source software usable among others in combination with python [Abadi et al., 2015]. It is developed and maintained by Google. Tensorflow is optimized for numerical computations mainly useful for the purpose of machine learning using data flow graphs. While the nodes in the graph represent operations the edges represent the data arrays. The architecture allows computation on CPUs and GPUs. We especially used the GPU approach to speed up computations.

1.4 Recommendation Diversity

Various datasets are concentrated in the sense that they have significantly more interactions on the most interacted items than on the rest. In other words, the rating distribution follows a long-tailed distribution and acknowledging the tail can be valuable to gain additional information [Park and Kim, 2017]. Also, often the tail items are ignored or not considered enough to provide the users with a satisfying experience. Since the tail items, also called Eccentric items, are far less interacted with than the popular items, they might show more user-specific preferences. This additional information might help to distinguish the user better from other users than with the more popular interactions.
However, not every unpopular item is valuable to every type of user. Some items are loved by the majority of users, some are only adored by a specific type of user with a common interest. The work of C. Park and S. Kim [Park and Kim, 2017] has shown that there is a big percentage of items in the long tail, which could be used to make both more diverse and more interesting recommendations to users. This is important since only increasing accuracy not necessarily increases user satisfaction [McNee et al., 2006]. The users do not only want to see the most popular items they also want a suitable

amount of diverse items. To capture this, the definition of Eccentric items has been introduced [Park and Kim, 2017]. By using the Eccentricity as a benchmark value one can show that recommendations are more diverse.

1.5 Our Contribution

The contribution of this work is improving the original Collaborative Metric Learning algorithm and creating an Eccentricity based Recommender System.
The Collaborative Metric Learning algorithm is improved in our work by including the negative feedback the user gave. We represent users and items in a higher dimensional space, and consider their distance to reflect their preference relationships, i.e, a user prefers an item that is near in this space. Previous approaches only focus on lowering the distance between a user and her/his preferred items, while ignoring the distance with disliked items. We specifically model the distance with disliked items in this work. We include the disliked items in the process and increase the distance between the user and her/his disliked items.
Also, this work contributes a Recommender System based on Eccentricity. Eccentricity has been overlooked in many other Recommender Systems. To improve user satisfaction and recommendation diversity we aimed at including it into a Recommender System. We will show that by including it into the recommendation process we can increase user satisfaction in general by increasing the diversity of the recommendations. We will also show that for the Eccentric users this improves the Recommender System significantly.

1.6 Thesis Outline

In Chapter 2 Collaborative Metric Learning in general is introduced. Our contribution Collaborative Metric Learning with explicit negative feedback is described and various aspects of the implementation of this application are discussed such as, the rating conversion, the mathematical formulas that have been used, the datasets, the results that have been created and certain restrictions that have influenced this application. In Chapter 3 various aspects that affect the Eccentricity are discussed. This includes formulas that have been used, to capture the Eccentricity. Also, a way how to construct a Recommender Systems using these formulas is introduced and tested. In Chapter 4 the limitations that have influenced this work, mainly hardware, are discussed. Also, the Conclusions of this work are given and additional work that is expected to improve the value of this work will be discussed. In Appendix A the installation instructions and contents of the CD are listed.

2

Collaborative Metric Learning with Negative Feedback

Collaborative Metric Learning(CML) has been shown to generate better recommendations than other state-of-the-art systems [Hsieh et al., 2017]. CML aims at combining CF with ML. The idea of CML is to group users and items, which are suitable for the user by learning a user-item joint metric to encode this relationship [Hsieh et al., 2017]. This algorithm generally, "pulls" (decreases the distance) similar pairs (item, user) closer together in the joint user-item space. During the learning process, the users who co-liked the same items will become closer neighbors in the learned vector space. The items which are co-liked by the same users will become closer neighbors in the learned n-dimensional space as well through this procedure. After execution the closest items of each user are the items he liked previously and those liked by a user who liked similar items. Also, features are taken into consideration to bring items with similar tags or features closer together. The result of this algorithm is a user-item embedding, this embedding can predict items, which a user might like by looking at the neighborhood of the user. The neighborhood can be understood as the items which are distance wise the closest to the user.

2.1 CML

In this section, we describe CML and the high-level concepts that are used to create it. CML's basic idea is that a set of user-item pairs S that are known to have positive relationships are used to learn a user-item joint metric to encode these relationships [Hsieh et al., 2017]. Generally one can see this process as pulling similar pairs closer together and pushing the other pairs relatively further apart. This process will also group users who co-liked the same items and group items which are co-liked by similar users closer together. The closest item-neighbors of the users will become the items the user liked and the items, which other users who share common likes, liked.

2.1.1 Model Formulation

For convenience, we state the model formulation stated in the original paper here again [Hsieh et al., 2017]. Let r_{ij} denote user i's rating to item j, one learns the user vector

$u_i \in \mathbb{R}^r$ and item vector $v_j \in \mathbb{R}^r$ such that the dot product $u_i^T v_j$ approximates r_{ij} [Koren et al., 2009] . Each user and each items is represented with a user vector $u_i \in R^r$ and an item vector $v_j \in R^r$. Then these vectors are learned in a way that their Euclidean distance,

$$d(i, j) = u_i - v_j$$

obeys user i's relative preferences. This will result in items being liked by the users becoming closer neighbors than items which the user did not like. The following loss functions is used to approach this [Hsieh et al., 2017],

$$L_{m0}(d) = \sum_{(i,j)} \sum_{(i,k)\notin S} w_{ij}[m + d(i, j)^2 - d(i, k)^2]_+$$

where j is an item user i liked, k is an item he did not like and $[z]+ = max(z, 0)$ denotes the standard hinge loss, w_{ij} is a ranking loss weight.

2.2 CML with Explicitly Negative Feedback

We propose our new method Collaborative Filtering with Explicitly Negative Feedback (CML-EN) which is a new model build upon the CML. CML with Explicitly Negative Feedback (CML-EN) is similar to the CML algorithm but aims at using all feedback given by users. While the CML algorithm only considers positive feedback e.g. a like, our approach also considers the explicitly negative feedback given by the user e.g. a dislike.

CML is a way to represent relationships between users in a distance by giving each a position in n-dimensional space. Also, the items are represented by a position in space such that their distance to the users represents if they are liked by him/her. The general idea is to push and pull items and users. This pushing and pulling is such that they eventually end up at positions which describes their relationships. By describing the positions of all items and users in the embedding, we gain new possible recommendations by looking at the closest item user pairs. This work specifically aims at pushing items further away from users which disliked these items. This process also pushes these items further away from users which have not yet consumed them. This decreases the probability a user is recommended an item he dislikes.

While the original only pulls the positive items closer to the user, this approach also pushes (increases the distance) the explicitly negative items further away from the user. We measure the squared difference between the user's position in the embedding and the explicitly negative item. This difference will then be added to the loss term as described in the section 2.2.1. This loss term will then be minimized. In other words, given a user u_i and an item he disliked i_n we look up the position of both in the n-dimensional embedding space. The function $EPos(u_i)$ will return an n-dimensional vector which describes the position of the item or user in the respective embedding space. While the look-up functions $EPos(u_i)$ and $EPos(i_n)$ are implementation wise differently, we will treat it here such that it returns the value of the position in the embedding space for both items and users. This can be achieved by using the function $tf.nn.embedding\_lookup$

[Abadi et al., 2015]. We then us a function to measure the distance between the two positions using

$$dist(u_i, i_n) = sqDiff(EPos(u_i), EPos(i_n))$$

we are using the Squared Difference. This method of distance measuring has been chosen to stay consistent with the CML, which also uses the squared difference to measure distances in the embedding. To capture the relations between a disliked item and a user. We need to define a function which decreases as the distance increases. To accomplish this it is suitable to look at the inverse of the distance e.g. $\frac{value}{distance}$. We try to define it such that the minimization process decreases this inverse until a certain threshold or safety margin is reached. We want to push the item far enough to not be in the recommendation area anymore. However, we also want this inverse not to be decreased too much since this would push all items very far away from all users. We, therefore, define a function,

$$UserNegativeDist(u_i, i_n) = max(\frac{\alpha}{dist(u_i, i_n)}, 1.0)$$

Where α is a distance tuner and describes the maximum distance the negative items should be pushed away from the User that rated it negatively. The goal is to push the user as far away from the item, as it is necessary for the user, to not be recommended this item again. The value of $\alpha > 1$ is the distance the item will eventually be away from the user. It is important to notice, that the distance value if to close (which is when we want to push it further away) will be smaller than one, to understand why this increases the distance. This effectively means that the $UserNegativeDist(u_i, i_n)$ is big when the $dist(u_i, i_n)$ is small. The intuition behind this is that the bigger the distance between u_i and i_n is, the smaller the value returned by $UserNegativeDist(u_i, i_n)$ becomes until it finally reaches the minimum of 1, where increasing the distance stops and minimizing this further is not possible. We finally add the term to our loss function which will be described in more detail in section 2.2.1. Given a set of users U and for each user, a set of items he did liked I_p and a set of items he disliked I_n,

$$\sum_{u \in I_p} \sum_{i_n \in I_n} UserNegativeDist(u, i)$$

which gives us the term we need to add to the loss function. The term we just described was indented to increase the distance between users and their disliked items. This on its own already brings improvements to the system and successfully decreases the negative items in the users top recommendations.
To improve the system further also pushing negative items and positive items further away from each other would be desirable. The approach to also change the distance between items makes little sense when only working with positive items. The reasons behind this is that decreasing the distance between items would let them lose their "direction". With direction, we mean the dimensions which differentiate different positive items, which were liked by the same user. However, when working both with negative and positive feedback increasing the distance between two items which are different, differentiates them better. We know that two items are different by looking at items the

same user disliked and liked. We then proceed to increase the distance between these two. This improves the system and further decreases the disliked items in the users top recommendations.
Additionally to the Explicitly Negative Feedback, we described before we now also generate a new pair constellation. This positive-negative pair is generated from the users explicitly negatively and positively interacted items. This means a pair of items (i_p, i_n) where i_p is a positively rated item and i_n is a negatively rated item, both items have been rated by the same user.
More formally we define a new set Dissimilar Pairs(DP) where every element is a pair $(i_p, i_n) \in DP$ where $i_p \in I_p$ and $i_n \in I_n$. DP can be calculated as the Cartesian Product of I_p and I_n

$$DP = I_p \times I_n$$

Now we sample some random element $(i_p, i_n) \in DP$ every iteration and increase the distance between i_p and i_n slightly. The impact of this distance change should be much smaller then the previous changes since the pairs (i_p, i_n) is only based on the feedback from one user but affects two items. The approach to defining a distance suitable for this is very similar to the approach for $UserNegativeDist(u_i, i_n)$. Given two items i_p and i_n we define,

$$ItemNegativeDist(i_p, i_n) = max(\tfrac{\alpha}{dist(i_p, i_n)}, 1.0)$$

Where α is a distance tuner and describes the maximum distance the negative items should be pushed away from the positive item. The intuition behind this is very similar to the intuition described for the $UserNegativeDist(u_i, i_n)$. The goal is to push the item i_p and i_n further away from each other. The value of $\alpha > 1$ is the distance the item will eventually be away from each other. It is important to notice, that the distance value if to close (which is when we want to push it further away) will be smaller than one, to understand why this increases the distance. This effectively means that the $ItemNegativeDist(i_p, i_n)$ is big when the $dist(i_p, i_n)$ is small. The intuition behind this is that the bigger the distance between i_p and i_n is, the smaller the value returned by $ItemNegativeDist(i_p, i_n)$ becomes until it finally reaches the minimum of 1, where increasing the distance stops and minimizing this further is not possible. When stopping at the minimum this means the item is now further away from the user. We push these two items further away from each other since we know a user thinks one is different than the other. In other words, we use the information the user gave us about this item to ensure the items will be more distinct. We finally add the term to our loss function described in 2.2.1, given the set DP,

$$\textstyle\sum_{(i,j) \in DP} ItemNegativeDist(i, j)$$

which gives us the term we need to add to the loss function. As explained in the graphical explanation 2.1 this method adds values by decreasing the amount negative items in the user's neighborhood. Where the neighborhood are the items which will eventually be recommended to him/her. Before any training has been done, some negative items are,

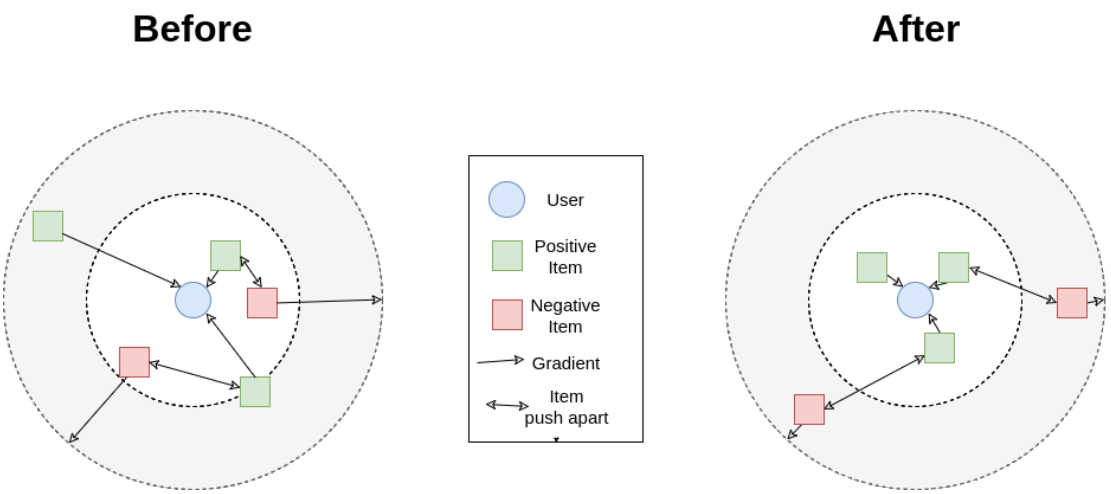


Figure 2.1: Graphical Explanation.

shown in red in the graphical explanation 2.1, close to the user. Our goal is to increase the distance between these negative items and the user (blue in the figure). This is achieved by pushing the items into the directions of the gradients, away from the user until a threshold value is reached. This threshold is indicated in gray in the graphical explanation. The positive items are shown in green in the graphical explanation 2.1 will be pulled closer to the user. As described also the distance between the negative items and the positive items will be increased. The arrows show the direction of the gradients e.g. in which directions the items will be pushed/pulled. All of this happens in an n-dimensional space.

2.2.1 Model Formulation

Additional to the model of the original CML. We add two loss terms described in the previous section of the model. Given a user set U and an item set the user disliked In,

$$L_{m1}(d) = \sum_{u \in U} \sum_{i \in I_n} UserNegativeDist(u, i)$$

And also given the set DP described earlier we define a second Loss function as,

$$L_{m2}(d) = \sum_{(i,j) \in DP} ItemNegativeDist(i, j)$$

2.2.2 Training Procedure

The complete objective function of the proposed model is,

$$\min_{u_*, v_*} L_{m0} + \lambda_f L_f + \lambda_c L_c + \lambda_0(\lambda_1 L_{m1} + \lambda_2 L_{m2})$$

Where L_f and L_c are loss terms defined in the original paper which can but do not have to be used in combination with this method [Hsieh et al., 2017]. The L_f can be used if

features are present. This loss improves the model by adding feature dimensions. The L_c describes the covariance loss. Which can improve the model by constraining the objective but was not used in our testing process. The reason for this is that it was not recommended in the original CML code. This can also be left out of the minimization process if no features are available. Where $\lambda_f, \lambda_c, \lambda_0, \lambda_1$ and λ_2 are hyperparameters which control the weight of the loss terms. We minimize this functions us with Mini-Batch Stochastic Gradient Descent (SGD) using *AdamOptimizer* [Kingma and Ba, 2014]. We used *AdamOptimizer* instead of *AdaGrad*, as proposed in the original paper since the creator of CML suggested on the projects GitHub-repo that *AdamOptimizer* is the better choice. The training procedure can be summarized as,

1. Sample N positive pairs from S
2. Sample N explicitly negative pairs from D
3. Sample N pairs from DP
4. For each positive pair, sample U negative items
5. For each positive pair, keep the negative item k that maximizes the hinge loss and form a mini-batch of size N
6. Compute gradients and update parameters with a learning rate controlled by AdamOptimizer.
7. Repeat this procedure until convergence.

2.3 Negative Items in Top K

The Top K recommendations of a user are the $k \in \mathbb{N}$ items the recommendation system recommends to the user. A Negative Item (NI) is defined as an item which has been recommended to the user but he does not like. Since user satisfaction with recommendations is affected by the number of negative items in his top k recommendation, improving on the Negative Items in Top K(NITK) is desirable. A low, preferably 0, value in NITK is desirable since this would mean there are not many items in the users top-recommendations he dislikes. We define the NITK as,

$$NITK = max(\frac{|NI \cap TK|}{k}, \frac{|NI \cap TK|}{|NI|})$$

Where set NI is the set of items the user interacted negatively with. The set TK is the set of items the recommendation system recommends to the user, the k refers to the size of the recommendation e.g. $|TK| = k$. We will use the notations NIT10 and NIT5, which indicates that the size of k is 10 for NIT10 and 5 for NIT5 respectively. The value NITK indicates the probability that a negative item is in the TK recommendations. A value of 0 for NITK is desirable since this would mean that there are no negative items in the top k recommendations which also affects precision. If not indicated the NITK

values given are the mean of the NITK for all users. The values in NITK are generally low given a suitable recommendation system since most items the user dislikes should not be in the top-recommendations. However, often users do not know about the items they hate the most since current recommendation systems are already quite good and will recommend the user suitable movies such that the user dislikes often movies which are closer than the movies which are the furthest away from him.

2.4 Ratings

In most modern websites, programs, and apps the possibility to rate certain items are present. Different rating systems exist which can mainly be divided into 5 categories. These 5 categories are [Aggarwal, 2016]

- Continuous Ratings
- Interval-based Ratings
- Ordinal Ratings
- Binary Ratings
- Unary Ratings

While the methods we propose are mainly useful for Binary Ratings, which are very popular and widely used. We used Interval-based Ratings for our development and converted them into Binary Ratings to generate methods which work for Binary, Interval-based and Continuous Ratings.

2.4.1 Convert Interval-based & Continuous Ratings to Binary Ratings

While the easiest way to convert Continuous and Interval-based ratings to Binary Ratings is to take the middle of the rating interval and classify everything above this as a like and everything below this as a disliked. This approach is possible and works well but can be improved by calculating,

$$MeanUserRating(R) = \frac{\sum_{r \in R} r}{|R|}$$

where R is the set of Ratings and r is a rating in the range of the interval of the rating. We classify everything below this Mean-User-Rating as a disliked and everything above it or equal to it as a like. The reason for this is to get a more even distribution of likes and dislikes which benefits both training and evaluation. The main problem of taking the mean of the rating system instead of the mean of the users ratings is that users usually vote strongly above the mean of the rating system e.g in the Movielense datasets the rating system ranges from 1-5 which would indicate a mean of 3 but the

global average is roughly around 3.5. An overview of the ratings in the datasets is shown for Movielens and Goodbooks in table 2.1.

Table 2.1: Overview Datasets Ratings

	Dataset Good books 10k	Dataset Movielens 20m	Dataset Movielens 1m
Average Rating	3.92	3.52	3.58
Amount 0.5 Ratings		239'125	
Amount 1 Ratings	124'195	680'732	56'174
Amount 1.5 Ratings		279'252	
Amount 2 Ratings	359'257	1'430'997	107'557
Amount 2.5 Ratings		883'398	
Amount 3 Ratings	1'370'916	4'291'193	261'197
Amount 3.5 Ratings		2'200'156	
Amount 4 Ratings	2'139'018	5'561'926	348'971
Amount 4.5 Ratings		1'534'824	
Amount 5 Ratings	1'983'093	2'898'660	226'310

2.5 Datasets

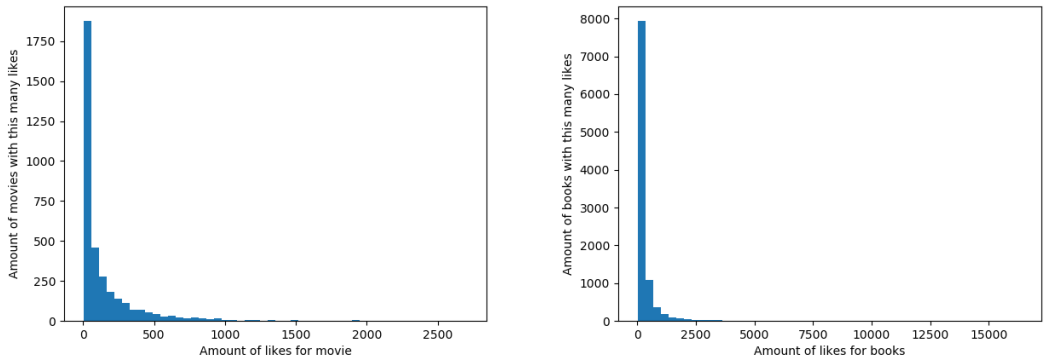
We have evaluated our result on two datasets. One being the Movielense 1 million [Harper and Konstan, 2015] and the other one being the Goodbooks 10k dataset [Zajac, 2017]. MovieLens is a dataset provided by GroupLens Research at the University of Minnesota. Goodbooks is a popular dataset for books and features ratings and tags from many different sources. For both datasets, there are ratings from 1-5 which are connected to one user and one item. We calculated the average user rating using $MeanUserRating(R)$ and classified everything that is bigger or equal to this $MeanUserRating(R)$ as a like and everything below this average as a dislike.
For Movielense 1 million we used the plot-tags from *www.themoviedb.org* and for Goodbooks 10k tags are already available but we only used the tags for each item which have been mentioned at least 5 times. Since some of the aspects we tested show clearer in bigger datasets we evaluate some aspects also on Movielens 20 million. We did not filter the datasets initially. However, we used thresholds on how many interactions users need to have at least. If this is the case it is mentioned in the related section. The Amount of Users, Amount of Items and the number of Ratings for the data sets Movielens 1 million, Movielens 20 million and Goodbooks are given in the table 2.2,

Table 2.2: Datasets

	Amount of Items	Amount of Users	Amount of Ratings
Dataset Goodbooks 10k	10'000	53'424	5'976'479
Dataset Movielens 20m	27'000	138'000	20'000'000
Dataset Movielens 1m	3'952	6'040	1'000'000

2.5.1 Concentration

The concentration seems to have an effect on the performance of the algorithm. Here concentration means the number of likes which are concentrated on the top most liked items. Tail-liked items are the items which have been consumed by only a few users but still are liked. Movielens and Goodbooks are both very concentrated but Goodbooks has a lot more tail-liked items than Movielens as seen in the table 2.3 below.



(a) Movielens 1 million

(b) Goodbooks 10k

Figure 2.2: Histogram items by amount of likes.

The Movielens datasets have a lot of items which are not liked a lot. The roughly 30% least liked items in Movielens only account for around 1% of all likes. The 20% most liked items concentrate roughly 72% of the likes on them.
The Goodbooks dataset has also a lot of items which are not liked a lot but has more tail-liked items. The roughly 30% least liked items in Movielens only account for around 1% of all likes, while the dataset Goodbooks has 5.4% in the 30% least liked items. This is 5 times more and has an impact on the results. The 20% most liked items concentrate roughly 69% of the likes on them, which is also quite high but common also in other datasets.

Table 2.3: Table describing the distribution of likes in the datasets

Concentration	Movielens 1 million	Goodbooks 10k
0-10% most liked	0.1%	1.2%
10-20% most liked	0.3%	1.9%
20-30% most liked	0.8%	2.3%
30-40% most liked	1.4%	2.9%
40-50% most liked	2.4%	3.6%
50-60% most liked	4.1%	4.6%
60-70% most liked	6.8%	5.8%
70-80% most liked	11.4%	8.3%
80-90% most liked	20.4%	14.04%
90-100% most liked	52.1%	55.13%

However as we seen the 50% least likes items in Movielens account for only 5% of all likes. The Goodbooks 50% least like items account for 11.9% which is more useful when analyzing for the Eccentricity. If not enough likes are present on the least liked items the chance that an item is recommended which is not very popular decreases dramatically as we will show in Chapter 3.

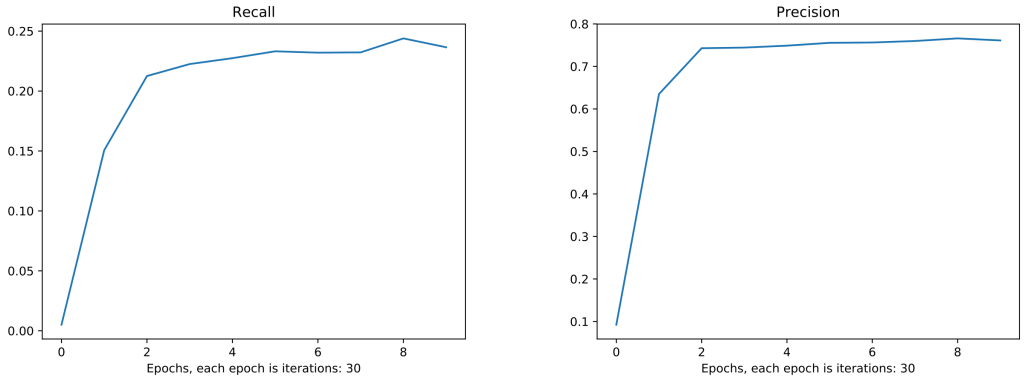
2.6 Results

A good recommendations system not only needs to have optimal values in the area of Recall and Precision. Also, the top-recommendations that a user receives or is shown should include as little Negative items as possible (Items the user won't like). The reasons behind this are that our assumption is, that showing/recommending a user an item he does not like decreases the customer satisfaction. Since customer satisfaction eventually is more important than most other aspects of recommendation systems we should also optimize for this.
We tested both the CML and the CML-EN algorithm on NITK. The parameters, exclusive to the CML-EN, have been set to show a clear trend on the validation set. A clear trend in this context means that we optimized such that all measured values improve and not just one improves drastically and the other ones decline.
Both Systems have been tested with the same hyperparameters, with the exception of those that are exclusive to the CML-EN. An overview of the hyperparameter setting can be found in table 2.4. Namely the parameters Explicitly Negative loss scale, Explicitly Negative-Positive-Pair loss scale, Distance tuner Explicitly Negative-Positive-Pair and Distance tuner Explicitly Negative are exclusive to the CML-EN. The reason for this is that they exist only in the CML-EN. Generally, if we change the hyperparameters affecting both the CML and the CML-EN both change relatively similar. With the notable exception of the number of iterations, in this regard when not trained enough the CML-EN will perform worse.
We used for this series of test a split of 33.3% train, 33.3% test and 33.3% validation for

the dataset. We used such a split to have more data for the evaluation which in case of Negative items in the top recommendations needs to have a bigger test/validation set to be clearly visible, this of course also has an influence on all other metrics. Using this setting only users with at least 3 likes are used for evaluation but all users are used for training. Users generally have roughly the same amount of likes and dislikes, due to the definition of a like and a dislike we used as described earlier.
We included only tags which have at least been mentioned for 2 items in the training process. For the Movielens dataset in general, we included tags from the dataset which have been given to the movies by at least 5 users. The reason behind this was to exclude some spelling errors and accidental tags which make little sense to be included. We also added tags for the movies from the *www.themoviedb.org* API to increase the number of tags available for training. In the case of goodbooks we used the tags included in the datasets which are already of a high quality. Also, in this case, we only included the tags which have been named at least 5 times by different users to exclude accidental spelling errors and other problems that might decrease the quality of the tags. We run each test 3 times with random seeds and present the average in the tables. The plots have been generated by an example run and might not represent the average exactly which are shown in the tables.

2.6.1 Results CML-Original

The following Result was created by using the Collaborative Metric Learning [Hsieh et al., 2017] in its original form. Our results are not as good as the author claimed in his paper for the Movielens 20m dataset, we contacted him about it but were not answered. We assume that the reason behind this is that the data filtering process is either different or the computational resources we are able to use are smaller. Also, tags and processing of them could have varied. The initial testing phase on the Movielens 100k set suggested though that splitting the ratings on the users-average instead of simply picking all rating bigger than 4 had a beneficial impact on the testing, we therefore assume that even though this is a difference to the author's original setup it was not the main problem. The number of features that were used has also varied compared to the author's original paper, which we can't explain since we used the API suggested by imdb. The author in his paper described he had 10'399 features, with more than 5 occurrences, for testing with the Movielens 20 million dataset [Hsieh et al., 2017]. Using our method for accumulating features only 3'942 features are created, with more than 5 occurrences. In this sections, we will show plots of the test set the final results on the validation set and comparison are shown in the last section.

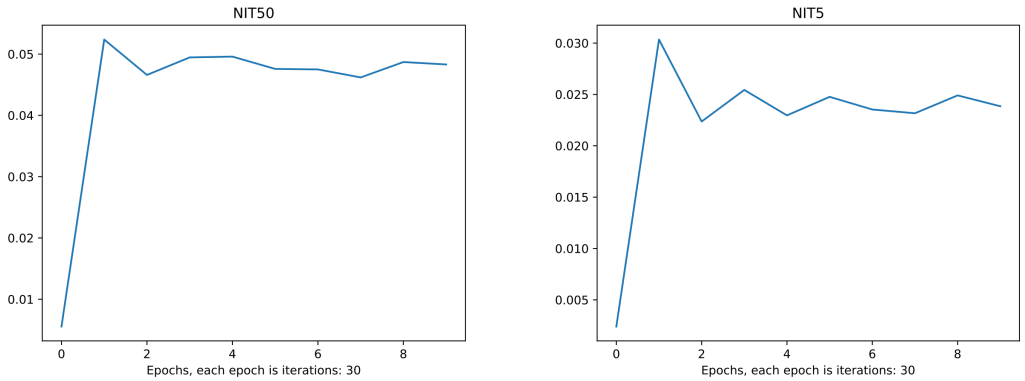


(a) Recall at 50 of CML over time for goodbooks10k

(b) Precision of CML over time for Goodbooks10k

Figure 2.3: Result for Goodbooks10 using CML.

The left figure 2.3a shows the development of the recall at 50 over the training process. The Recall at 50 of Goodbooks10k increases strongly in the beginning of the training processes and after the first few hundred iterations stays almost the same. Additional iterations bring little to no additional benefits in terms of Recall. Too few iterations, can result in worse results as we see. The second plot shows the development of the precision at 50 over the training process.
The Precision of Goodbooks10k at 50, as shown in figure 2.3b increases strongly in the beginning of the training processes as the Recall but continues increasing slightly during the following iterations. Additional iterations bring little benefits in terms of Precision but keep to improve slightly. The Precision is already quite high in comparison to the Recall in the beginning though.



(a) NIT50 of CML over time for Goodbooks10k

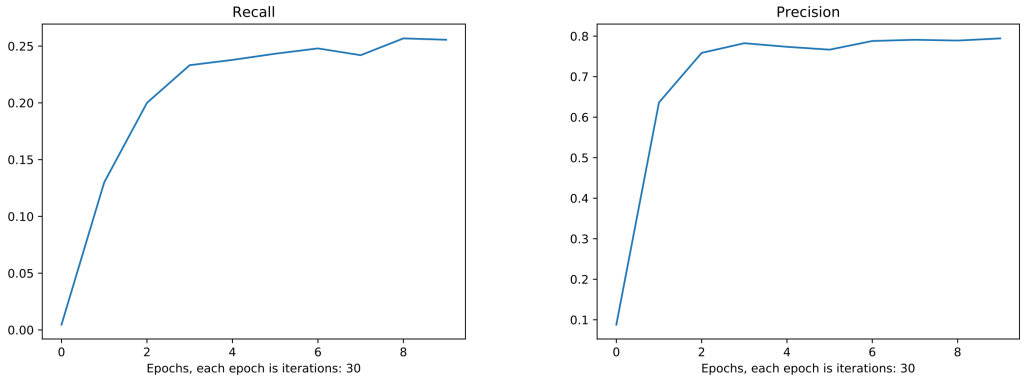
(b) NIT5 of CML over time for Goodbooks10k

Figure 2.4: Result for Goodbooks10 using CML.

The figure 2.4a shows the development of the NIT50 over the training process. The NIT50 described how many recommendations users got on their top-50 recommendation which they explicitly disliked e.g. downvoted. The value increases rapidly in the beginning since if the recommendations are random, like they are in the beginning, the chance of recommending an item the user disliked is not very high. Once the recommendation system begins to recommend sensible choices the NITK increases. Additional iterations bring little benefits in terms of NITK but keep to improve slightly. As mentioned earlier a low value preferably 0 is better for NITK. The values, can change relatively significantly in comparison between epochs.
The figure 2.4b shows the development of the NIT5 over time. Similar to the NIT50 the NIT5 describes how many negative items were in the users top-5 recommendations. A high value in NIT5 suggests that the recommendation system is not very beneficial for customer satisfaction since the top-5 items are almost certainly guaranteed to be seen by the user. In the Results Comparison also NITK at other values are given. The values, can change relatively significantly in a comparison between epochs and this difference changes even stronger with the NIT5 than with the NIT50. The reasons behind this is that analyzing only the top-5 is more volatile than analyzing the top-50.

2.6.2 Results CML-Explicit Negative Feedback

The following Results were created by using the Collaborative Metric Learning [Hsieh et al., 2017] with the addition of explicitly negative items-user pairs training (CML-EN). The results mainly differ from the original CML by having a worse Recall but improving in NITK and Precision if tuned to perform optimally on the NITK. However, to show that the CML-Explicit Negative(CML-EN) can also outperform the original in this matter the hyperparameters, which are exclusive to the CML-EN, have been set to show an even increase overall measurements. The hyperparameters exclusive to the CML-EN could also be set, such that only NITK improves heavily but Recall decreases.
The reasons for this improvement is that by including all-feedback a user has given and not just the likes the users have given we can gain additional information. This system could be further improved by adding more tags. By including the user's negative feedback, in particular, the embedding can be trained to increase the distance between a user and negatively rated items to improve the recommendations.

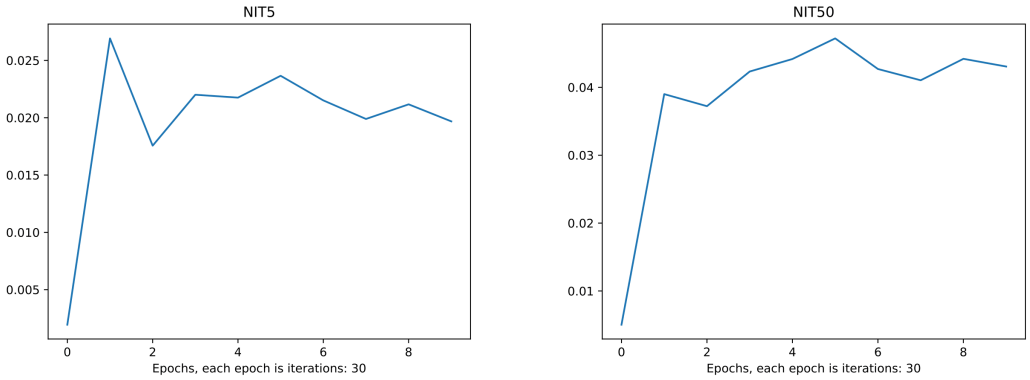


(a) Recall at 50 of CML-EN over time for Goodbooks10k

(b) Precision of CML-EN over time for Goodbooks10k

Figure 2.5: Result for Goodbooks10 using CML-EN.

The figure 2.5a shows the development of the Recall at 50 over the training process of the CML-EN. The Recall of Goodbooks10k increases strongly in the beginning of the training processes and after the first few hundredth iterations stays almost the same. Additional iterations bring little to no additional benefits in terms of Recall on this dataset. However, in comparison to the CML the Recall improves slower and over a longer time period.
The figure 2.5b shows the development of the Trecision at 50 over the training process of the CML-EN. The Precision of goodbooks10k increases strongly in the beginning of the training processes as the Recall but continues increasing slightly during the following iterations. Additional iterations bring little benefits in terms of Precision but keep to improve slightly. However, compared to the CML here additional training time is desirable to reach better results.



(a) NIT5 of CML-EN over time for goodbooks10k

(b) NIT50 of CML-EN over time for goodbooks10k

Figure 2.6: Result for Goodbooks10 using CML-EN.

The figure 2.6a shows the development of the NIT50 over the training process. Once the recommendation system begins to recommend sensible choices the NITK increases. This is since it is the lowest when random choices are recommended since the change of selecting a negative item at random is fairly low. In comparison to the CML-original, the NIT50 decreases after more training time as the minimization process decrease this value. Additional iterations bring benefits in terms of NIT50. As mentioned earlier a low value preferably 0 is desirable and one of the main goals of this algorithm. Therefore training for a longer period of time for a commercial system would be desirable
The figure 2.6b shows the development of the NIT5 over time. Similar to the NIT50 the NIT5 describes how many negative items where in the users top-5 recommendations, which are the items the user is the most likely to watch and is one of the measurements we aimed to improve. In the Result comparison, more NITK values are given for a better comparison. As already mentioned about the same plot for the CML this measurement tends to change relatively strongly between epochs. This could be decreased by changing the learning rate or increasing the strength of the negative loss values in general.

2.7 Result Comparison

This sections features the results of both the CML-Orginal and CML-Explicit negative(CML-EN). The hyperparameters can be tuned to punish NITK stronger but this comes at the cost of Recall. For this comparison the hyperparameters, exclusive to the CML-EN, have been tuned to show that CML-EN can be better in all measurement and show a clear trend. As mentioned earlier the hyperparameters, exclusive to the CML-EN, could also be set to improve NITK stronger but hurt the Recall. We run the tests with the hyperparameters,

Table 2.4: Hyperparameter for tests

Hyperparameters	Movielens 1 million	Goodbooks 10k
Batch-size	80'000	100'000
Embedding dimension	70	100
Nr Negatives	30	30
Hidden layer dimensions	256	512
Clip norm	1.1	1.1
feature projection scale	1.0	1.0
Iterations	1500	1500
drop-out rate	0.4	0.3
Explicitly Negative loss scale	1.25	1.5
Explicitly Negative-Positve-Pair loss scale	0.005	0.005
Distance tuner Explicitly Negative-Positve-Pair	3.5	3.0
Distance tuner Explicitly Negative	4.0	4.0

Note that the parameters Explicitly Negative loss scale, Explicitly Negative-Positve-Pair loss scale, Distance tuner Explicitly Negative-Positve-Pair and Distance tuner Explicitly Negative only apply to the CML-EN. Explicitly Negative loss scale affects all the effects the explicitly negative parameters have and affects the strength at which the values are added to the loss function. Generally the effect of the Negative loss functions, on the loss, should be smaller than the effect of the other loss functions. The reasons behind this is that pushing negative items too far away from the user is not desirable. It is not desirable since often ratings are given for relatively similar movies. Relatively similar in this context means that a user often will view movies which are from relatively similar genres. These movies might be bad or good for the user but a user often never sees the items that would, distance wise, be the furthest away from him.
As an example we can think of a user which watches a lot of action movies, in his rating list most likes and dislikes will be from the action genre, the items which should be the furthest away are the items which have absolutely no relationship with him. However, these items are not very likely to appear in his rating list at all e.g. a Japanese romance, which has absolutely no relation to his preference. But since he never consumes these items, we have no information on how much he dislikes it. Therefore we should not push an action movie he dislikes as far away from him, as an item he has absolutely no relationship with is, away from him e.g. a Japanese romance movie.
The Explicitly Negative-Positive-Pairs loss describes how strong the effect the Negative-Positive-Pair effect we described earlier, on the loss should be. This value has to be set

very low since the information has been generated by only one user and affects two items. The Distance tuner Explicitly Negative describes how far an item should be pushed away from a user and should be tuned such, that the item does not appear anymore in the user's top-k recommendations but is not pushed too far away. The Distance tuner for Explicitly Negative-Positive-Pair describes how far the items should be pushed away from each other, this distance should be smaller than the other distance tuner. The reason for this is, that it affects two items but the rating has been captured by only one user and different users might contradict each other in this decision. Therefor finding the right value for these parameters has to be approached for each data set differently. We found the parameters we use by performing a grid search. However the datasets Goodbooks10k and Movielens 1 million are relatively similar and we used similar hyperparameters for the tests.

2.7.1 Goodbooks 10k

The results for Goodbooks have been tested for both CML-original and CML-EN and the hyperparameters, exclusive to the CML-EN, have been set to show a clear increase in all measured values. Generally they could also be set to perform stronger in the NITK but this would hurt the Recall. The values for Recall, Precision at 10, 20, 50, 75 and 100 are all higher with the modified CML-EN compared to the original CML which is what was desired. The values for NITK at 1, 5, 10, 20, 30, 40, 50, 60, 70 and 80 are all lower using the CML-EN compared to the CML in case of the NITK lower is better. However the CML-original is faster and needs no downvotes and therefore less training time. Also the memory usage of CML-EN is higher compared to the CML which can be handled by using different batching and sampling.

Table 2.5: Recall Results Goodbooks 10k

Dataset Goodbooks 10k	CML-Original	CML-EN
Recall at 10	8.260%	9.187%
Recall at 20	13.615%	14.844%
Recall at 50	23.728%	25.682%
Recall at 75	29.372%	31.585%
Recall at 100	33.653%	36.078%

As we see the values for 10, 20, 50, 75 and 100 Recall are higher with the CML-EN compared to the CML-original. The value for Recall at 100 for example increases by roughly 2.425% which is a relative improvement of 7.2%. The value for Recall at 10 improved by 0.927% which is a relative improvement of 11.2%.

Table 2.6: Precision Results Goodbooks 10k

Dataset Goodbooks 10k	CML-Original	CML-EN
Precision at 10	56.075%	60.948%
Precision at 20	66.878%	71.285%
Precision at 50	75.815%	79.182%
Precision at 75	77.829%	80.087%
Precision at 100	78.667%	81.534%

As we see the values for 10, 20, 50, 75 and 100 Precision are higher with the CML-EN compared to the CML-original. The value for Precision at 100 for example increases by roughly 2.867% which is a relative improvement of 3.6%. The value for Precision at 10 improved by 4.873% which is a relative improvement of 8.6%.

Table 2.7: NITK Results Goodbooks 10k

Dataset Goodbooks 10k	CML-Original	CML-EN
NIT1	3.550%	3.267%
NIT5	2.199%	1.904%
NIT10	2.055%	1.792%
NIT20	2.405%	2.123%
NIT30	3.220%	2.850%
NIT40	4.096%	3.635%
NIT50	4.932%	4.377%
NIT60	5.705%	5.087%
NIT70	6.413%	5.750%
NIT80	7.084%	6.364%

As we see the values for NIT1, NIT5, NIT10, NIT20, NIT30, NIT40, NIT50, NIT60, NIT70 and NIT80 are lower with the CML-EN compared to the CML-original which was the main goal of this work. The value for NIT1 for example decreased by roughly 0.283% which is a relative improvement of 8.0%. The value for NIT80 decreased by 0.72% which is a relative improvement of 10.1%.

2.7.2 Movielens 1 million

he results for Movielens have been tested for both CML-original and CML-EN and the hyperparameters, exclusive to the CML-EN, have been set to show a clear increase in all measured values. Generally, they could also be set to perform stronger in the NITK but this would hurt the Recall. The values for Recall at 10, 20, 50 and 75 are all higher with the modified CML compared to the original CML which is what was desired. However, the value of Recall at 100 is lower which might have something to do with the distance tuner set to low since the values for the lower Recalls are better. The values for NITK at 1, 5, 10, 20, 30, 40, 50, 60, 70 and 80 are all lower using the CML-EN compared

to the CML. However, the CML-original is faster and needs no downvotes. Compared to Goodbooks 10k the difference in results is less significant which probably is due to the different concentrations of interactions in the datasets. In Movielens the amount of interactions concentrated on the top 50% of the items is significantly higher.

Table 2.8: Recall Results Movielens 1 million

Movielens 1 million	CML-Original	CML-EN
Recall at 10	11.642%	12.634%
Recall at 20	18.668%	19.691%
Recall at 50	32.151%	32.991%
Recall at 75	39.730%	40.261%
Recall at 100	45.764%	45.672%

As we see the values for Recall at 10, 20, 50, 75 are higher with the CML-EN compared to the CML-original. However the result for Recall at 100 is lower. The value for Recall at 100 decreased by roughly 0.092% which is the only value which decreased. The value for Recall at 10 improved by 0.927% which is a relative improvement of 8.5%.

Table 2.9: Precision Results Movielens 1 million

Movielens 1 million	CML-Original	CML-EN
Precision at 10	62.723%	66.068%
Precision at 20	71.169%	72.412%
Precision at 50	73.737%	74.357%
Precision at 75	72.738%	73.334%
Precision at 100	71.611%	71.995%

As we see the values for 10, 20, 50, 75 and 100 Precision are higher with the CML-EN compared to the CML-original. The value for Precision at 100 for example increases by roughly 0.384% which is very little but increased relatively more in the other areas. The value for Precision at 10 for example improved by 3.345% which is a relative improvement of 5.06%.

Table 2.10: NITK Results in Movielens 1 million

Movielens 1 million	CML-Original	CML-EN
NIT1	2.569%	1.840%
NIT5	2.202%	1.896%
NIT10	3.171%	2.815%
NIT20	5.493%	5.267%
NIT30	7.919%	7.550%
NIT40	10.012%	9.793%
NIT50	12.244%	11.907%
NIT60	14.380%	13.890%
NIT70	16.357%	15.706%
NIT80	18.407%	17.640%

As we see the values for NIT1, NIT5, NIT10, NIT20, NIT30, NIT40, NIT50, NIT60, NIT70 and NIT80 are higher with the CML-EN compared to the CML-original which was the main goal of this work. The value for NIT1 for example decreased by roughly 0.729% which is a relative decrement of 28.0%. The value for NIT80 decreased by 0.767% which is a relative decrement of 4.1%.

3

Diverse Recommendations

Often Recommender Systems aim so much at getting the Recall and Precision to a sensible level that user satisfaction is not the main priority anymore. Users do not only want accurate recommendations they also want diverse, surprising and interesting recommendations [McNee et al., 2006].
Therefore also looking at recommendations, which are not very likely to appeal to the users, since they are not very often rated, has value. One of the main reasons that affect that recommending uncommon items to users hurt Recall and Precision, is that most Recommender Systems never show these items to the users. With various evaluations regarding Eccentricity, we want to show the differences among the uncommon items. This work also aims at highlighting some of the ways that might improve the diversity. We will also propose a method which improves recommendations for user groups, which like more uncommon items.

3.1 Popularity

Most items have been consumed by users different amounts of times. While certain items might have interacted with a lot, certain items may only have been interacted with, a few times. We will classify the items which have been interacted with a lot as mainstream and those who have been interacted with only a few times as uncommon or niche items. Recommender Systems often recommend items which are very popular since they are very likely to be liked by the general user. For certain users though, which are very interested in niche items, popular recommendations are not satisfying. Therefore considering the popularity of items, is important and can add value to the system and increase user satisfaction.
We define popularity as the likelihood an interaction is made with a given item. We define the set Items(I) and for every item $i \in I$ we define the value $i_{interactions}$ which is the number of interactions made with this item. The popularity of an item $i \in I$ is defined as,

$$popularity(i) = \frac{i_{interactions}}{\sum_{j \in I} j_{interactions}}$$

Which is the amount of interactions made with the item divided by the amount of interactions made in the entire data set. This is the probability an item is interacted with.

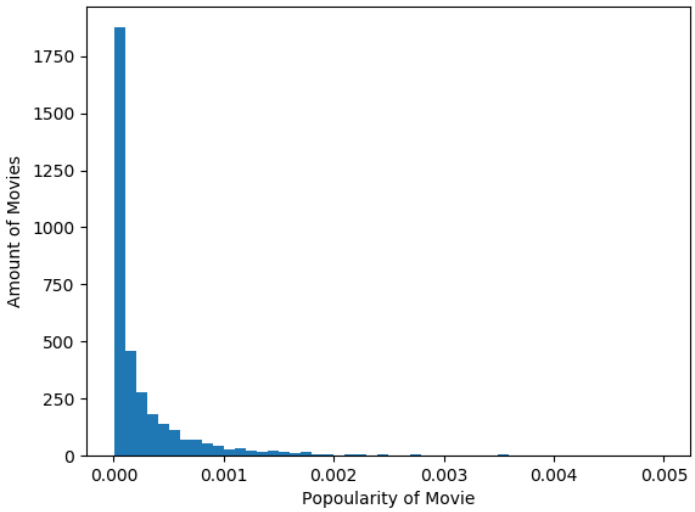


Figure 3.1: Popularity of Movielens 1 million

As we see in figure 3.1 the popularity of a few items is very high but the popularity of most items is very low. This has a lot to do with, the concentration of the data set.

3.1.1 Controversy

Controversy is related to the ratio between likes for an item and dislikes. It can be assumed that certain users like items which are heavily disliked by other users and vice versa. An example might be a controversial war-documentary which certain people adore and others dislike even though both parties like documentaries in general. Therefore looking at the controversy of a movie is important.
An interaction we define as a rating a user made on an item, the value of this rating does not matter. A dislike we define as a rating below the $MeanUserRating(R)$ for an interval based rating. A like is defined as a rating above or equal to the $MeanUserRating(R)$. In a binary rating system, the likes and dislikes would be given by the user directly. We define the controversy of an item $i \in I$

$$controversy(i) = \frac{i_{likes} - i_{dislikes}}{i_{interactions}}$$

The $controversy(i) \in [-1, 1]$, since all items that are interacted with are either liked or disliked. A value of $controversy(i) > 0$ indicates that i is a good movie and a value of $controversy(i) < 0$ indicates that the movie is a bad movie as perceived by the majority of the general users. There seems to be a certain group of people that really enjoy and

mainly consume items in the $controversy(i) \in [-0.1, 0.1]$, which we qualified as the controversial items. The intuition behind this definition is, that it will now only qualify all movies which have a difference of around 20% between likes and dislikes e.g. 45 likes and 55 dislikes. Defining it like this qualifies roughly 15% of the movies as controversial, which we thought was suitable. These items give us the set CI which is the set of the controversial items.

3.1.2 User-Controversy

As discussed earlier there might exist a group of users, in the respective data set, which consumes and likes an increased number of controversial items. Each user has a set UI which are items the user interacted with e.g. either liked or disliked. The user's positively interacted with items define a set UI_p and the users explicitly negatively interacted with items define a set UI_n. We define now the Positive-User-Controversy of a user u

$$PositiveUserControversy(u) = \frac{|CI \cap UI_p|}{|UI_p|}$$

And the Negative-User-Controversy as

$$NegativeUserControversy(u) = \frac{|CI \cap UI_n|}{|UI_n|}$$

The Positive-User-Controversy indicates how many of the items the user liked are controversial. A high value indicates that a user mainly enjoys controversial items. A low value indicates that user mainly enjoys very popular items e.g. only watches blockbusters.
The Negative-User-Controversy indicates how many of the items the user disliked are controversial. A high value indicates that the user mainly dislikes controversial items. A low value indicates that the user either watches very little controversial movies or likes them more often. These numbers should be compared to the user's User-Controversy which is the ratio of controversial movies a user consumes

$$UserControversy(u) = \frac{|CI \cap UI|}{|UI|}$$

A high value in User Controversy indicates the user watches mainly controversial items. A low value in User Controversy indicates that the user barely interacts with controversial items. In the figure 3.2, we see the Positive-User-Controversy (green) and the Negative-User-Controversy (red). The users are ordered by the User-Controversy and on the x-axis we see the users position in the ordering where each x is a user. The figure 3.2 tells us that there are generally more downvotes on Controversial movies than likes, this is visible since the Negative-User-Controversy is generally higher than the Positive-User-Controversy.

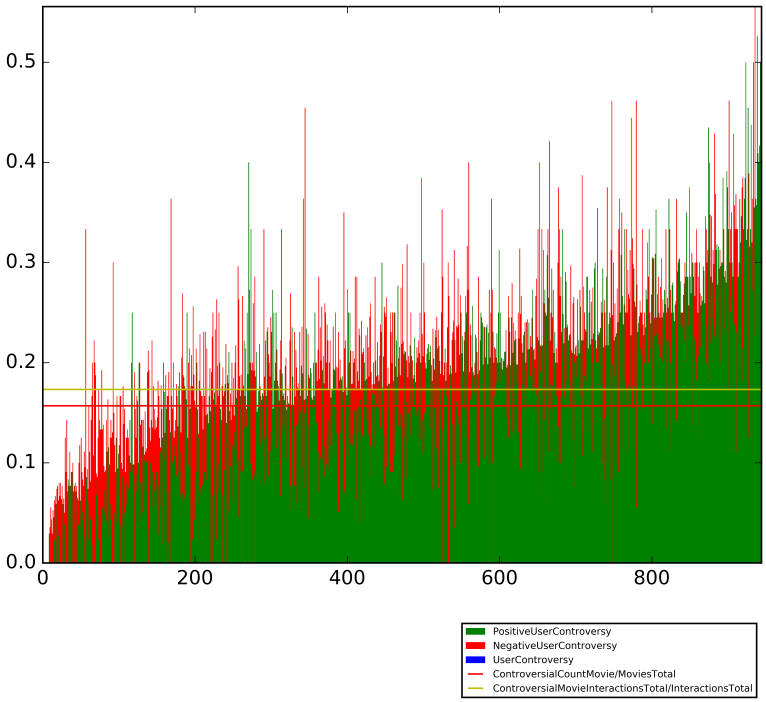


Figure 3.2: Positive and negative User Controversy sorted by User Controversy, for Movielens

3.2 Item-Eccentricity

The definition for Item-Eccentricity has been given by Park and Kim [Park and Kim, 2017]. The general idea of item Eccentricity is that there is a group of items which is loved by a small group of users but ignored by the majority of users e.g. a french-art movie. These items are niche items and mainly consumed by a small fan base. Park and Kim aimed at finding these items by defining the Item-Eccentricity [Park and Kim, 2017]. This measurement helps in identifying items which are not very often consumed but can be positively perceived by users which enjoy more niche items. We will later show how to use this measurement to recommend Eccentric items to Eccentric users and generally improve the diversity of the recommendations.

3.2.1 Item-Rarity

Eccentric Items are items which are only consumed by a small amount of people. Therefore it is important to know, how often an item is consumed, compared to other items. The definition of Item-Rarity helps to capture this relation between items and users. The Item-Rarity is defined as [Park and Kim, 2017],

$$ItemRarity(i) = IR(i) = -log(|F_i|)$$

Where $|F_i|$ is the number of interactions (feedback's) on the Item i. After applying $Item - Rarity(i)$ to all items in the Item set the resulting values are standardized to a z-score. Where the z-score is defined as,

$$zscore(IR(i)) = \frac{IR(i)-\mu}{\sigma}$$

Where μ is the population mean and σ is the standard deviation of all Item Rarities. When using the Item-Rarity in the following definitions always the z-score of the Item-Rarity is used.

3.2.2 User-Eccentricity

The User-Eccentricity is a value that describes how many rare items the user consumes. It can be seen as a measurement for users, which identifies how often the user likes items which are not often consumed by the vast majority of users. The original definition of the User Eccentricity is [Park and Kim, 2017],

$$UserEccentricity_u(UE_u) = \frac{\sum_{i \in I_u, t \in T} f_{u,i,t} \dot{IR}_{i,t}}{\sum_{i \in I_u, t \in T} f_{u,i,t}}$$

where i_u indicates the set of items that user u has consumed and the IR_i, t is the item rarity of item i in time window t. The value $f_{u,i,t}$ is the rating of the user u for the item i at time t. Our definition of the User-Eccentricity is a simplified form of the User-Eccentricity used in the Eccentric of Items paper [Park and Kim, 2017]. In our definition, we only work with the items above or equal to the $MeanUserRating(R)$, since we are using only likes and dislikes and do not use the timestamps. The User-Eccentricity of a User u is defined as,

$$UserEccentricity(u) = UE(u) = \frac{\sum_{i \in U_p} IR(i)}{|U_p|}$$

Where U_p are the items liked by the user u. As with the Item-Rarity, these values are standardized to the z-score and when using the User-Eccentricity in the following definitions we always mean the z-score of it. The figure 3.3 shows the distribution of the Eccentricity by users for the dataset Movielens 1 million.

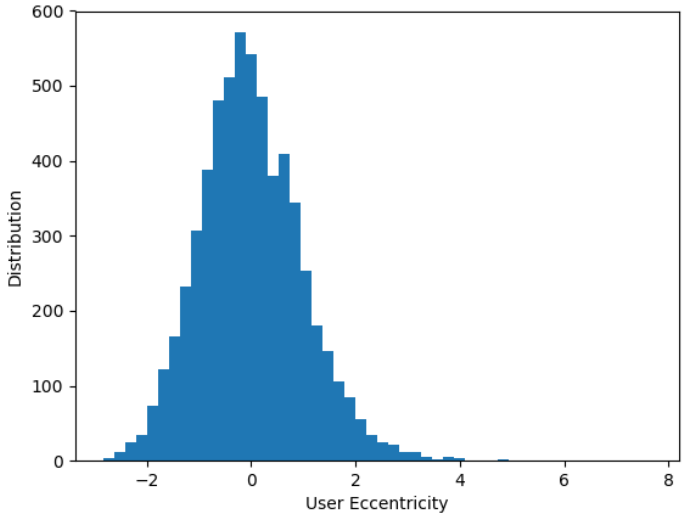


Figure 3.3: Distribution of Users by Eccentricity

3.2.3 Item Eccentricity

The Item Eccentricity is a measure of how often the item is liked by Eccentric users. It can be seen as a measurement of how Eccentric this item is and consequentially its value for niche items is higher. Our definition is a simplified form of the Item-Eccentricity used by Park et al. [Park and Kim, 2017].
We simplified it to only use likes and not Interval based ratings. Also, as defined earlier the $UE(u)$ definition is different, apart from these two changes they are the same. The Item Eccentricity for an item i is defined as,

$$ItemEccentricity(i) = IE(i) = \frac{\sum_{u \in IU_p} UE(u)}{|IU_p|}$$

Where IU_p are the users who liked the item i. As with the Item-Rarity, these values are standardized to the z-score and when using the Item-Eccentricity in the following definitions we always mean the z-score of it. The figure 3.4 shows the distribution of the Eccentricity by items for the dataset Movielens 1 million.

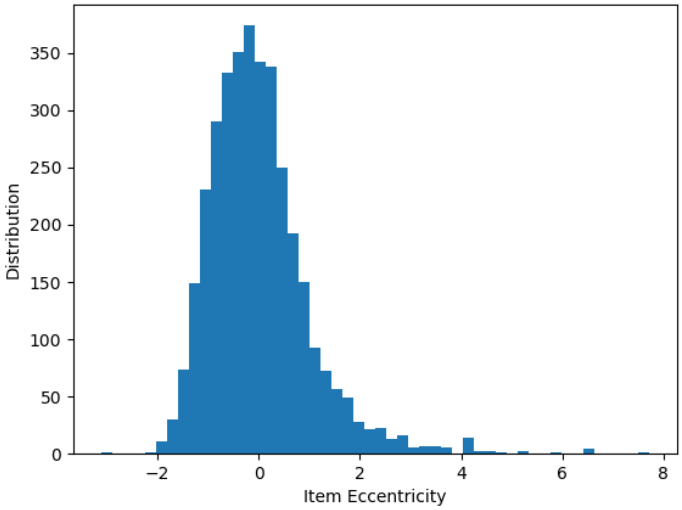


Figure 3.4: Distribution of Items by Eccentricity

3.2.4 User Similarity for Eccentric Items

Current Recommender systems often recommend the most popular items to a user, since statistically, this is far more likely to appear to the user than recommending niche item. However, most users have a strong desire for items which do not appear to everyone but to a small niche of users and are relatively specific e.g. samurai-movies. Therefore it is desirable to find small subsets of niche items which appear to Eccentric users.
To find such a small subset of very specific movies, we tried to find user pairs which are similar in the group of Eccentric items, with the goal to find another item which appears to this group. In order to capture this Eccentric similarity of users to users we defined the Eccentric-user-similarity ($EUS(u_1, u_2)$) as,

$$EUS(u_1, u_2) = \frac{\sum_{i \in (U_{1p} \cap U_{2p})} IE(i)}{|U_{1p}|}$$

Which is the sum of the Item Eccentricity of the items both liked u_1 and u_2 divided by the number of likes given by u_1. It is worth noting that this relation is not symmetric e.g. $EUS(u_1, u_2) \neq EUS(u_2, u_1)$. The reason for this is that if a user u_2 liked 100 item and u_1 liked 10 items and all items liked by u_1 are liked by u_2. The user u_1 in this example is similar to u_2 but u_2 is not similar to u_1. This can be used to find movies u_1 might like in the set of movies u_2 liked.
We started generating these numbers for the dataset Movielens 20 million and found some niches by looking at the User pairs with the highest EUS. We were able to identify niches such as Japanese movies from the 1940's and Sports documentaries. However we never fully developed an algorithm which takes this into account for recommendations,

since it would be very time-consuming. An example of the highest scoring User Similarity by Eccentricity is given in the table 3.1,

Table 3.1: Example: highest EUS, movies from the 1940's

Gaslight (1944) Eccentricity val: -0.974012354897
Maltese Falcon, The (1941) Eccentricity val: -1.38968808205
Laura (1944) Eccentricity val: -0.934747342516
Sword in the Stone, The (1963) Eccentricity val: -1.40328736383
Enchanted April (1992) Eccentricity val: -1.18999765866
Room with a View, A (1986) Eccentricity val: -1.3025864896
First Strike (1996) Eccentricity val: -1.57568266953
Lady Vanishes, The (1938) Eccentricity val: -0.841740797953
Jewel of the Nile, The (1985) Eccentricity val: -1.38617275624
Ideal Husband, An (1999) Eccentricity val: -1.22176600613
Pelican Brief, The (1993) Eccentricity val: -1.35108680292
Yellow Submarine (1968) Eccentricity val: -1.25332845171
Muppet Movie, The (1979) Eccentricity val: -1.38667880852
I Was a Male War Bride (1949) Eccentricity val: -0.279255796273
Another Thin Man (1939) Eccentricity val: -0.418494856136
Thin Man Goes Home, The (1945) Eccentricity val: -0.321803390378
Shadow of the Thin Man (1941) Eccentricity val: -0.503684514005
After the Thin Man (1936) Eccentricity val: -0.420642989892
How to Steal a Million (1966) Eccentricity val: -0.721571595527
Adventures of Sherlock Holmes, The (1939) Eccentricity val: -0.56716496043

While it might be surprising that the Eccentricity values for the movies shown in the table 3.1 are all below 0, which might indicate that they are not very Eccentric. However, the reason behind this is mainly that the vast majority of likes is concentrated on the least Eccentric items e.g. the popular items.
Only recommending a movie with an Eccentricity above 0 would mean, only recommending movies with very few likes e.g. usually between 0 and 20. Also, movies which have an Eccentricity below zero can be Eccentric. A value above zero simply indicates the item is very Eccentric. Recommending these movies would be very diverse but not beneficial for other measurements like the Recall, since the likelihood of recommendations being in the user's likes would drastically decline, when only recommending the least liked items. This is the reason why recommending movies which are Eccentric but not extremely Eccentric is a sensible choice.

3.2.5 Item Similarity based on Eccentricity

An Item Similarity based on Eccentricity can be useful for identifying items which are underrated in current Recommender Systems. With underrated we mean items which

are very popular with a small group of users but are for various reasons not shown to users which might like such niche items. By defining an Item Similarity based on Eccentricity we can recommend niche items to users, which suit their respective preferences. This could be used to recommend not very well known items to users which have already consumed the most likely and popular options. In order to capture this Eccentric Similarity of items we defined the Eccentric-Item-Similarity($EIS(i_1, i_2)$) as,

$$Eccentric - Item - Similarity(i_1, i_2) = EIS(i_1, i_2) = \frac{\sum_{u \in (IU_1 \cap IU_2)} UE(u)}{|(IU_1 \cap IU_2)|}$$

Where IU_i is the set of the users who liked the item i. Which is the mean of the User Eccentricity of the users which liked both $Item_1$ and $Item_2$. It is worth noting that in contrast to the Eccentric User Similarity, the Eccentric Item Similarity is symmetric. The main reason is computing this for large datasets of items is expensive. Therefore computing it as a symmetric relationship, as we realized, is significantly faster. Trying to define this without the symmetric usage might be an additional option and could yield better results especially in the relationships between very popular and very niche items.

3.2.6 Combining Item Similarity based on Eccentricity

The Item Similarity based on Eccentricity itself can mainly be used to recommend niche items. This can be used either for recommending an Eccentric item to Eccentric users or recommending items most users have not heard of but are similar to their respective taste. To combine the item similarity based on Eccentricity several approaches are feasible. Mainly these approaches will focus on combining the Eccentric approach with other more common approaches. The reason behind this is non-Eccentric users usually do not like Eccentric items as much as Eccentric users and therefore should be recommended less Eccentric items.

3.2.7 Eccentric Item Similarity based on Eccentricity

We use the definition for Eccentric-Item-Similarity (EIS) and combine it into a new Matrix I. This matrix is the similarity between items i_1, i_2 in regards of Eccentricity. We define each entry i,j in I as,

$$I[i, j] = EIS(i_1, i_2)$$

Where in $i_1 i_2$ the numbers 1,2 are the ids of the items in the dataset. We define a matrix R which contains the ratings for the users. Each entry R_{ij} means that the user u_i liked the item m_j. To get Eccentric predictions for a user u_i we multiply the matrix for the Ratings R with the matrix for combined item similarity S to get the matrix E as,

$$E = R \cdot I$$

Where $\cdot$ is the dot product between two matrices. We then for each user u_i look at the i-th row of the matrix E. By sorting the values in E we get the Top-k recommendations for

a user by Eccentric item similarity. Where k describes the number of recommendations we look at. This matrix E represents the Eccentric Recommendations. Where each index i,j is the Eccentric recommendation strength of j for the user i.
Similarly, we combine the matrix R with an Matrix M into a new matrix P. The Matrix M is an item-item similarity matrix. This matrix M can be created in various ways. We used the algorithm pairwise cosine similarity to create the matrix we will use. This matrix M is not related to Eccentricity. The matrix P is the generated as,

$$P = R \cdot M$$

Where $\cdot$ is the dot product between two matrices. By sorting the values in P we get the Top-k recommendations for a user by pairwise cosine similarity. Where k describes the number of recommendations we look at. This matrix P represents the non-Eccentric Recommendations. Where each index i,j is the non-Eccentric recommendation strength of j for the user i.

3.2.8 Weighted combination

Using the matrix Eccentric Recommendations (E) in which each entry (i,j) indicates the Eccentric recommendations strength for a user i for the item j. One can use this matrix E to combine it with the other matrix P. Where P is the usual recommendation matrix based on ratings and is not related to the Eccentric approach. Each entry (i,j) in P indicates the non-Eccentric recommendation strength for a user i for the item j (such a matrix can be generated in various ways e.g. pair-wise-cosine-similarity). The matrices P and E can now be combined into a new matrix L using,

$$L[i,j] = \alpha * E[i,j] + (1 - \alpha) * P[i,j]$$

Where α is a hyperparameter which indicates how strong the Eccentric influence should be on the recommendations. The symbol $*$ is the normal multiplication of two numbers. This method can be used to add various more Eccentric recommendations to the users top-recommendations. This can be tuned to add as much new Eccentric items to the recommendations as wanted by the Recommender System, using the parameter α. A higher value increases, therefore the average Eccentricity of the recommendations, while a lower value decreases this amount. We ran tests with this methods but discovered that it is not as suitable as the following method. We therefore will also not present results related to this method. However, the Weighted Combination is important to understand why the Sigmoid Weighted Combination is more suitable.

3.2.9 Sigmoid Weighted Combination

Since by definition the more Eccentric users will consume more Eccentric-items than less Eccentric users and also less Eccentric users want more mainstream e.g. popular items in their recommendation list (as their previously defined preferences have shown). It would be useful and desirable to combine the Eccentric recommendations with normal

recommendations in such a way that less Eccentric users get less Eccentric recommendations and the more Eccentric users get more Eccentric recommendations. This aspect is very similar to the Weighted Combination shown previously. How much this difference should be set to, has to be evaluated from dataset to dataset and userbase to userbase and desired serendipity. If for example, we are talking about a movie-dataset and global user base, mainly blockbuster will be watched and over 70% of the total interaction will be concentrated in the top 5% of the most watched movies (example from Movielens 20 million). But if the dataset and userbase are centered around a less concentrated itemset more Eccentricity is desirable, for example in the case of Images where only 13% of the interaction are concentrated on the top 5% most voted items (example Flicker dataset [Hsieh et al., 2017]).
Therefore using the User-Eccentricity, already described in the previous section, to influence the strength of the Eccentric-recommendations, is desirable. We are using the User-Eccentricity as the strength of the slope of the sigmoid function, indicated as $UE(u)$. Given a user u for which the recommendation should be given. We define the formula,

$$sigmoid(u) = \frac{\gamma}{1+e^{-UE(u)*\beta}}$$

where the hyperparameters β and γ can be used to define the strength of the slope and the ratio of Eccentric to less Eccentric recommendations. The parameter β can be used to define the length of the transition, from non-Eccentric to the Eccentric definition, and should be adjusted depending on the difference between the most and the least Eccentric users. The parameter γ can be used to define the general ratio between Eccentric and non-Eccentric recommendations e.g. how Eccentric the recommendations should be on average. This formula has been designed in such a way that, the higher the User Eccentricity is the higher the influence of the Eccentric recommendations become e.g. $sigmoid(u)$.
Using the matrix Eccentric Recommendations (E) defined before in which each entry (i,j) indicates the Eccentric recommendation strength for a user i for the item j. One can use this matrix E to combine it with another matrix P which is a normal recommendation matrix and is not related to Eccentricity. Where each entry (i,j) in P indicates the recommendations strength for a user i for an item j (such a matrix can be generated in various ways e.g. pair-wise-cosine-similarity). The matrices P and E can now be combined into a new matrix S.

$$S[i,j] = f(i,j,u) = sigmoid(u) * E[i,j] + (1 - sigmoid(u)) * P[i,j]$$

where i is a user and j is an item. The position i,j is the combined similarity between the user i for the item j. The variable u is the user for which this recommendation is created. The symbol $*$ is a normal multiplication of two numbers. The variable u and i are actually the same in this formula since it is the recommendation for the i-th row which are the recommendations for the user u. We named them differently to avoid confusion and clearly highlight that one is the index of the row and the other is a user.

3.3 Results Eccentric Algorithm

We ran the tests for the Eccentric Algorithm combined with cosine pairwise similarity. We combined as described in the previous sections the Eccentric Item Similarity with a normal Item Similarity to generate recommendations named S. Also for the Eccentric recommendations non-combined we ran the tests to show how it would behave if used isolated named E. We also ran the test for the non-Eccentric recommendations named P to have a comparison. We will now show the results of these algorithms separately and compare them in the end.

3.3.1 Results Pairwise Cosine Similarity

We created the recommendations for users using the pairwise cosine similarity. By sorting the values in P we get the Top-k recommendations for a user. In the following figure 3.5, we show the average Eccentricity for the top-20 recommendations.

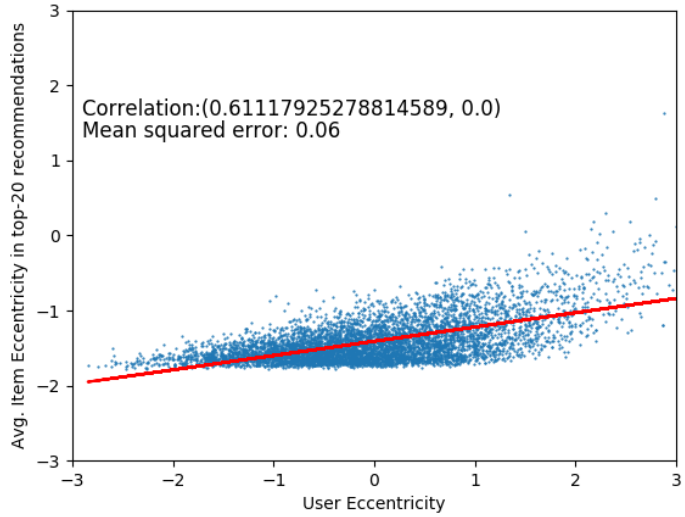


Figure 3.5: Distribution of Users by Eccentricity, Results using pairwise cosine similarity only

As we see in figure 3.5 the normal pairwise cosine similarity algorithm already captures the relations between more Eccentric users and more Eccentric items. As well as the relationship between less Eccentric users and less Eccentric items. However, as we see items with an Eccentricity over 0 are almost completely ignored which shows us that roughly 50% of the movies are very rarely recommended. In the following approaches, we aim at changing this.

3.3.2 Results Eccentric Similarity

To see the result also if only the Eccentric recommendations are taken into considerations we give here the results for this pure method, e.g. only using the Matrix called E (Eccentric Recommendations) before without combination. This method is not suitable to generate recommendations. Since it will recommend only very Eccentric items to the users, which is good for Eccentric users but not optimal for others.

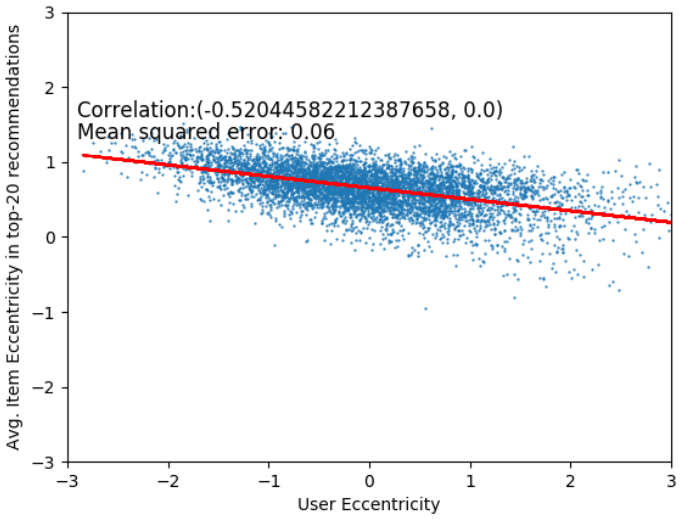


Figure 3.6: Distribution of Users by Eccentricity, Results for Eccentric Similarity only

As we see in figure 3.6 the average item Eccentricity in the top 20 recommendations is almost entirely above 0. This means again that roughly 50% of the items are ignored and very rarely recommended, this is wanted here. However, the most popular movies are never recommended. This approach used isolated, therefore is not recommendable. However, it could be used, to give users the option, to get very diverse recommendations.

3.3.3 Results Sigmoid Weighted combination

To use the Eccentric Similarity one needs to combine it with another approach. To make use of the Eccentric recommendations, which are more diverse but hurt the Recall of the algorithm for non-Eccentric users. This tells us that this algorithm is not entirely suitable for non-Eccentric users. However, by combining the two methods the Recall for the Eccentric user's increases and the overall diversity of the recommendations become higher than with the normal pairwise cosine similarity algorithm. We show now 3 promising versions of the same algorithm with different hyperparameters. All of these results have been created using the matrix S which are the recommendations for the users using Sigmoid Weighted combination,

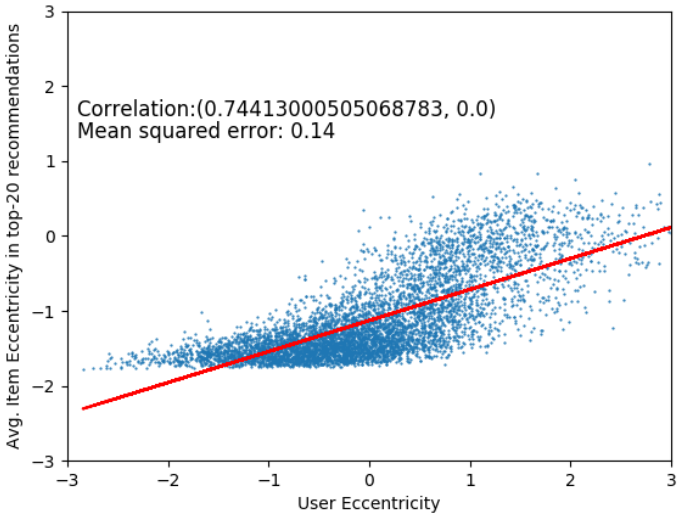


Figure 3.7: Distribution of Users by Eccentricity Combined #1

In this version called Combined #1, shown in figure 3.7, we used a Sigmoid-strength of 0.5 and an Eccentricity-strength of 0.2. As we see the average Eccentricity of the recommendations increases drastically for more Eccentric users and the recommendations become more diverse. This is the most diverse setting we are presenting. However, the results become only more diverse for the Eccentric users, which is desirable for this group.

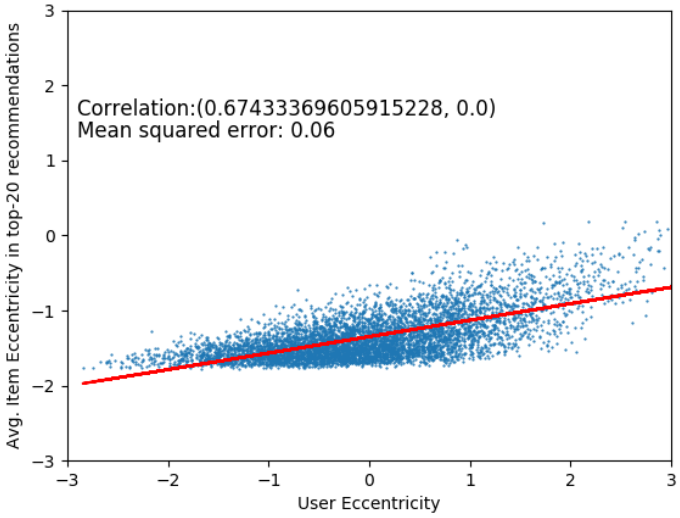


Figure 3.8: Distribution of Users by Eccentricity Combined #2

In this version called Combined #2, shown in figure 3.8, we used a Sigmoid-strength of 0.25 and an Eccentricity-strength of 0.1. As we see the average Eccentricity of the recommendations increases slightly for more Eccentric users and the recommendations become more diverse. This is the least Eccentric version we present in this section. However again the results become only more diverse for the Eccentric users, which is desirable for this group. The reason this version is better in comparison to the previous version is that the Recall improves both in comparison to the pairwise and in comparison to the Combined #1.

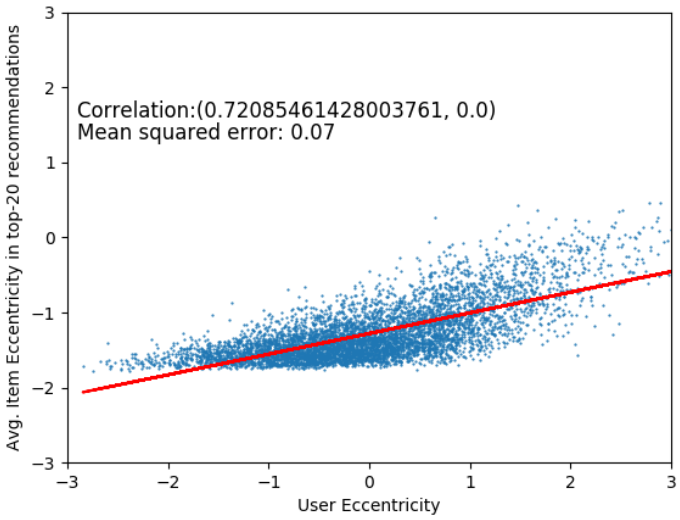


Figure 3.9: Distribution of Users by Eccentricity Combined #3

In this version called Combined #3, shown in figure 3.9, we used a Sigmoid-strength of 0.3 and an Eccentricity-strength of 0.15, which is in between version combined #1 and #2. As we see the average Eccentricity of the recommendations increases slightly for more Eccentric users and the recommendations become more diverse. However, again the results become only more diverse for the Eccentric users, which is desirable for this group. The reason this version is better, in comparison to the previous versions, is that Recall wise this is the best version we are presenting. Therefore we would recommend this version for usage since it improves various aspects but as mentioned it depends heavily on the used dataset and desired result.

3.4 Result Comparison

We show how the comparison of the methods for the different version of the algorithm. While the pairwise cosine similarity is used as our base value. We run all tests 3 times and took an average. We used a train test split of 50% to 50% which is slightly high but

shows the distribution of Eccentricity of the recommendations more clearly.
The Recall for the cosine pairwise similarity algorithm isolated, is at 31.158% and can be improved by using Combined version #3 by up to 0.243% while also increasing the diversity. We will use the Pearson Correlations, to show that the User Eccentricity and item Eccentricity now reflect each other better.

Table 3.2: Result comparison

Movielens 1 million	Pairwise	Eccentric only	Combined #1	Combined #2	Combined #3
Recall	31.158%	1.641%	29.331%	31.396%	31.401%
Mean Squared Error	2.63	1.75	1.61	2.42	2.16
Variance Score	0.37	0.27	0.55	0.45	0.52
Pearson Correlation	0.611	-0.520	0.744	0.674	0.721
Hyperparameters					
Sigmoid-strength			0.5	0.25	0.3
Eccentric-strength			0.2	0.1	0.15

As we see the Combined Eccentricity approach shows an increase in multiple measurements and has it's value. However, the results suggest there is a more suitable combination method. This could be used to improve the results further and improve on both Recall and Diversity. The Eccentricity as shown in the plots can be increase. However, the increase in Diversity over the entire spectrum of users comes at a cost of Recall. Therefore an approach which only improves the Diversity for Eccentric users is desirable.

Table 3.3: Recall by Eccentric Users

Recall at 50 for Eccentric Users	Pairwise	Combined #1	Combined #2	Combined #3
0.0-0.5 Eccentricity	24.94%	21.05%	24.96%	24.02%
0.5-1.0 Eccentricity	22.52%	17.68%	22.63%	22.03%
1.0-1.5 Eccentricity	19.91%	15.82%	20.64%	20.16%
1.5-2.0 Eccentricity	18.26%	17.56%	21.50%	22.17%
2.0-2.5 Eccentricity	22.31%	27.37%	28.74%	29.70%
2.5 >Eccentricity	20.41%	34.41%	29.69%	34.14%

As we see the Recall if only measured for the Eccentric users improves more significantly in comparison with the pairwise cosine similarity version. While the pairwise cosine similarity algorithm is still better when looking at the Users with an Eccentricity value between 0-0.5, the improvement for more Eccentric user from Eccentricity 1.0 is increasing. Specially when looking only at the users with an Eccentricity of 2.5 and

higher the improvement becomes very strong. The relative improvement for users with an Eccentricity of 2.5 > is an improvement of 13.72% which is a relative improvement of 67.3%. Also the Recall for user with an Eccentricity of 2.0-2.5 improves by 7.39% which is a relative improvement of 33.1%. However it has to be kept in mind that this improvement is only for a subset of the users and the Recall for Users with 0.0-0.5 decreases by 0.92% which might not be as much but since they are more users this has an impact on the mean Recall shown in the previous table. This indicates however that there might be a more suitable combination method that combines the advantages of both methods. We conclude that using the Eccentric Item Similarity can increase the user satisfaction for Eccentric users. Especially for the very Eccentric users, using such a system comes with a significant gain in Recall and other measurements.

4

Limitations, Future Work and Conclusions

4.1 Limitations

Limitations in this work were mainly dominated by the hardware. While at first, we worked with the Movielense 100k dataset the results were even more promising. However, the Movielens 100k dataset has not many Eccentric items and most Eccentric items are from the drama genre, which resulted in various problems in analyzing the results.
After realizing the Movielens 100k dataset is not suitable we switched to the Movielense 20 million, which was used in both papers, Measuring the Eccentricity of Items and Collaborative Metric Learning [Park and Kim, 2017, Hsieh et al., 2017]. This dataset is generally very interesting since it contains various different Eccentric items from Japanese 1940's movies to French art movies [Harper and Konstan, 2015]. However, while running and analyzing the algorithm on this dataset the size of the matrix was too big for the hardware I had at my disposal.
While generally, it would be possible to run the algorithm on this dataset and has been done, some of the algorithms we used were not able to perform on sparse Matrices which resulted in an out of memory problem. Generally working with it has been very time-consuming and slowed down the development process.
Similar problems persisted with Movielens 20 million in the CML algorithm since we were not able to reproduce the result the author mentioned in his paper. While generally similar results as reported in the paper by the CML author were achieved in a subset of the Movielense 20 million. We contacted the author of the CML-paper about this issue but were not answered we assume it could be because of the way the dataset was filtered but this was not explained detailed in the paper. Also, the number of features accumulated for the Movielens 20 million was not achieved by using the themoviedb.org API and did not match the number of tags the author mentioned in his paper.
While we were able to create a prototype for User based Eccentricity algorithm, it had various flaws and was not able to handle the Movielens 20 million dataset. Since the time it took to create the recommendations was too long, to ensure a productive workflow. Also, the User Controversy seemed to be promising. However, various problems in the execution and formal definition made it hard to include them into a Recommender

System.
The current algorithm for Eccentric Item Similarity can handle around 10'000 users and 5000 items with 16 GB of RAM. It is limited to this size and will produce no result if tested on bigger datasets. However, if the the RAM available increases also this limit increases. This could be improved by rewriting some aspects of the algorithm, which are not optimized for sparse matrices.
Generally, it would have been nice to run the Item Similarity based on Eccentricity algorithm on more datasets. However, due to the time, it takes to preprocess the datasets it was too time-consuming. Also, running the CML-EN algorithm on more datasets would have been good. However, finding datasets with a suitable format and enough features of a good quality can be harder than expected.
Last but not least, combining the two algorithms would have been one of the goals of this work. The problem with combining these two was that they are designed very differently, CML-EN is mainly based on tensorflow and integrating the Item Similarity based on Eccentricity into this system would have needed more time and various new design decisions.

4.2 Future Work

This work showed both the usage of negative feedback in Collaborative Metric Learning and the usage of less common e.g. Eccentric items in recommendation system in general. Future ideas for additional research that could be done would mainly be interesting in the area of Eccentric items since there seems to be still quite some aspects of this area which remain unexplored. The Eccentric User Similarity we defined showed promising and interesting results which could be used to find niche items and other interesting subsets of items in the dataset. While we were not able to explore these results further in this work, we think that there could be value gained by exploring this. Also, the combination method we used and tested sigmoid weighted combination is promising. While being a more sensible choice than the weighted combinations these results could, as the result of the Recall by Eccentric Users showed, be improved. Combining this Eccentric approach with other more sophisticated Recommender Systems might show interesting results too.

4.3 Conclusions

The implementation and design of the CML-EN and the Sigmoid Weighted Combination of Eccentric items has been an educational and challenging task.
The projects most demanding aspect was the usage of the relatively new tensorflow library and the combination of various different aspects of Computer Science, such as statistics, machine learning and Recommender System in general.
Creating and designing a system from scratch showed the author the importance of modularity in programs and various early design choices. Since there was no code base

to build upon in the Eccentricity area the author had the possibility to design and implement important aspects of the application himself.
Various aspects of the tensorflow technology that had to be understood to implement this solution showed the author many possibilities and limitations associated with this technology. While the fundamental usage and concepts of machine learning and python was known prior, the author had little practical experience in developing such a system. While the languages python and the technologies tensorflow where used by the author before he surely gained a lot of new experience and knowledge, most notably with tensorflow, sk-learn, and numpy.
During the time of this project, developing and experimenting with these different technologies was always interesting and educational. The author thoroughly enjoyed developing and writing this bachelor thesis and had a lot of fun doing so.

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Aggarwal, 2016] Aggarwal, C. C. (2016). *Recommender systems*. Springer.

[Harper and Konstan, 2015] Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19.

[Hsieh et al., 2017] Hsieh, C.-K., Yang, L., Cui, Y., Lin, T.-Y., Belongie, S., and Estrin, D. (2017). Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 193–201, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Koren et al., 2009] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8).

[McNee et al., 2006] McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM.

[Park and Kim, 2017] Park, C. and Kim, S. (2017). Measuring the Eccentricity of Items. *ArXiv e-prints*.

[Ricci et al., 2011] Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer.

[Sammut and Webb, 2011] Sammut, C. and Webb, G. I. (2011). *Encyclopedia of machine learning*. Springer Science & Business Media.

[Zajac, 2017] Zajac, Z. (2017). Goodbooks-10k: a new dataset for book recommendations. *FastML*.

A

Appendix

A.1 CD-content

The CD contains the source code for both the Eccentricity work and the CML-EN.

1. The algorithm CML-EN is based on the original CML.
2. Eccentricity pair-wise combination algorithm.
3. The latex files to generate this report.
4. The German summary.
5. The .pdf file of this document

List of Figures

List of Tables