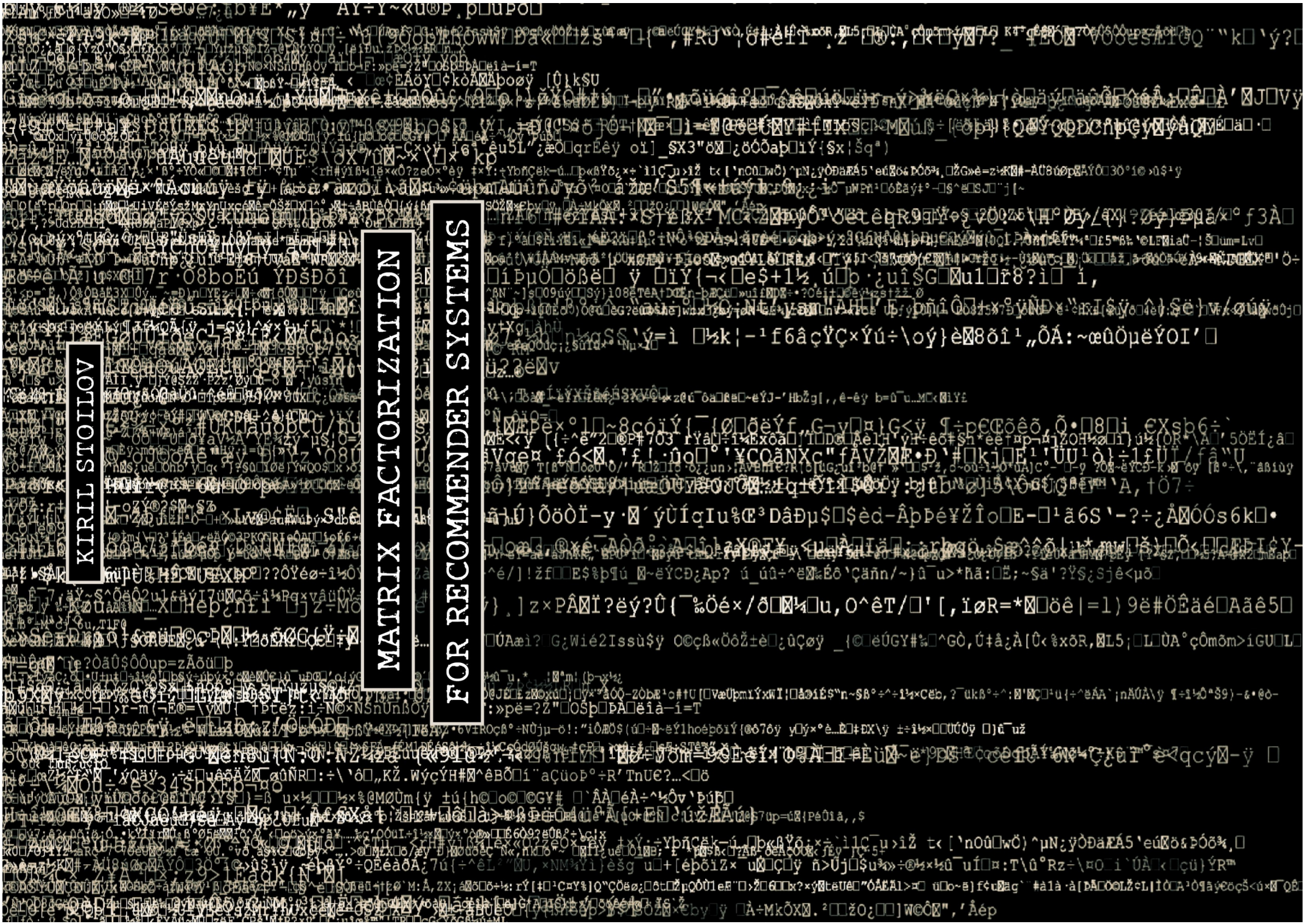
KIRIL STOILOV
MATRIX FACTORIZATION
FOR RECOMMENDER SYSTEMS

Kiril Stoilov

Matrix Factorization for Recommender Systems

MASTER'S THESIS

Graz University of Technology

Institute of Interactive Systems and Data Science

Advisor: Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Graz, May 2019

Kiril Stoilov

Faktorisierung von Matrizen für Empfehlungssysteme

MAGISTERARBEIT

Technische Universität Graz

Institute of Interactive Systems and Data Science

Betreuer: Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Graz, Mai 2019

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am _____ _____

(Datum) (Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources and that I have explicitly marked all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Graz, _____ _____

(Date) (Signature)

Abstract

Recommender Systems play an important role for successful e-commerce platforms ranging from retailers like Amazon.com to online DVD rental platforms like Netflix. With ever growing user and product bases, Recommender Systems are seen as a way of improving the user experience by matching products to customers.

Acknowledging the need for better Recommender Systems, Netflix set up a tournament in 2006 with a large cash prize for the team that improves the accuracy of their own recommendation algorithm – Cinematch - by 10%. The Netflix challenge served as a catalyst for the development of a plethora of Collaborative Filtering techniques for generating recommendations and led to large amount of fresh research in the field. With Matrix Factorization models topping the leader-boards, these types of methods proved to be the most popular and successful Latent Semantic Indexing approaches for Collaborative Filtering in this real-world inspired challenge.

In this thesis, two popular inference methods, namely Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS), were selected and implemented as interchangeable components of a generalized factorization model. Analysis was done in pragmatic fashion with the help of a practical implementation and tested on three large, real-world, sparse datasets. The factorization model represents an SVD-like decomposition, which maps the large matrix-like dataset to an approximation of smaller dimensions via the inference methods. The experimental results have shown that, although its relative simplicity, SGD can deliver satisfactory results when properly tuned. ALS on the other hand delivers excellent accuracy with great potential for further improvement, at the cost of being somewhat difficult to optimize and implement. Both return good results on the Netflix dataset with scores better than Netflix' own recommender system, and congruent with results by similar implementations from teams that have taken part in the Netflix challenge. The other two datasets - MovieLens and IMDB show even better results by using the same framework, partly because of their smaller size, but also because of the universality of the aforementioned algorithms.

Kurzfassung

Empfehlungssysteme spielen eine wichtige Rolle bei erfolgreichen e-commerce Platformen, vom Einzelhändler wie Amazon bis hin zum Videoverleih wie Netflix. Bei der ständig wachsenden Anzahl an Benutzern und Produkten sind Empfehlungssysteme ein Weg zur Verbesserung der Benutzererfahrung durch die Auswahl der Produkte die zu den Kunden passen. Nachdem Netflix die Notwendigkeit von besseren Empfehlungssystemen erkannte, wurde 2006 ein Wettbewerb mit großem Geldpreis für das Team ausgerufen, das die Genauigkeit des eigenen Empfehlungsalgorithmus - Cinematch - um 10% verbessert. Die Netflix Herausforderung diente als Katalysator für die Entwicklung einer Vielzahl kollaborativer Filtertechniken um Empfehlungen zu generieren und führte zu einer großen Anzahl an neuer Forschung in diesem Bereich. Mit Matrix Factorization-Modellen an der Spitze, erwiesen sich diese Methoden als die beliebtesten und erfolgreichsten Latent Semantic Indexing-Ansätze für Collaborative Filtering in dieser von der realen Welt inspirierten Herausforderung. In dieser Arbeit wurden zwei populäre Inferenzmethoden, nämlich Stochastic Gradient Descent (SGD) und Alternating Least Squares (ALS), ausgewählt und als austauschbare Komponenten eines Generalized Factorization Model implementiert. Die Analyse wurde mit einer praktischen Implementation durchgeführt und an drei großen, realen und spärlichen Datensätzen getestet. Das Faktorisierungsmodell repräsentiert eine SVD-artige Zerlegung, welche den großen Matrix-ähnlichen Datensatz auf eine Annäherung kleinerer Dimensionen mittels Inferenzmethoden abbildet. Die experimentellen Ergebnisse zeigen, dass SGD trotz seiner relativen Einfachheit mit den passenden Parametern zufriedenstellende Ergebnisse liefern kann. ALS hingegen liefert eine hervorragende Genauigkeit mit großem Verbesserungspotenzial, ist aber schwieriger zu optimieren und umzusetzen. Beide liefern gute Ergebnisse für den Datensatz von Netflix und übertreffen dabei den Algorithmus von Netflix und andere Implementationen von Teams die am Netflix Wettbewerb teilgenommen haben. Bei den beiden anderen Datensätzen - MovieLens und IMDB werden mit dem selben Framework sogar noch bessere Ergebnisse erzielt. Dies wird teilweise aufgrund der kleineren Größe, aber auch wegen der Universalität der oben genannten Algorithmen erreicht.

Acknowledgments

First and foremost my thanks go to the country of Austria and its great people for giving me the opportunity to study at this fine institution, what the TU Graz truly is. Second my thanks go to my parents, who supported me by all possible means in this enlightening endeavor. My further thanks go Stefan for his mentorship, thousands of shared laughs even in difficult moments and most importantly for his friendship. Last but not least I want to thank my supervisor prof. Denis Helic for the interesting theme of this thesis and also for the freedom he has granted me.

To C. F. Gauss the true enabler of all of this.

Table of Contents

List of Figures

List of Tables

Appendices

List of Abbreviations and Symbols

MF	Matrix Factorization
CF	Collaborative Filtering
LSI	Latent Semantic Indexing
SGD	Stochastic Gradient Descent
ALS	Alternating Least Squares
RBM	Restricted Boltzmann Machine(s)
NN	Nearest-Neighbor
MCMC	Markov Chain Monte Carlo
SVD	Singular Value Decomposition
ARP	Average Relative Position (ARP)
RMSE	Root-Mean-Square Error
PMF	Probabilistic Matrix Factorization
tf–idf	Term frequency–inverse document frequency
NFO	The original Netflix dataset (See: Chapt. 3.1.1)
NFU	The uniform Netflix dataset (See: Chapt. 3.1.1)
ML	Short for MovieLens
IMDB	Internet Movie Database

COPYRIGHTS

1 Introduction

The purpose of this chapter is to shed light on Recommender Systems as defined: *in own words, historically, loosely* and from *technical perspective*. Then an outline of the different types of Recommender Systems is listed. After that the motivation behind this work is presented and supplemented by research questions and contributions. Finally a quick outline of the rest of the work completes the chapter.

1.1 Definition

Recommender Systems offer customers or users a personalized selection of products or items based on different strategies. This personalized item selection is generated in order to aid the user with selection in an ever growing list of products or items. Previous research has shown that generally users are not eager to scroll down large lists of items on online platforms, this negatively affects the user satisfaction and impairs the commercial or practical success of these platforms. This gave rise to the development and adoption of recommender systems by different internet leaders like: Netflix, IMDB, Amazon.com and Google [10][11][5]. The following definition cascade follows the pattern presented by A. Felfernig et al. In [10], beginning with a pristine definition from 1997 [54]:

> "In a typical recommender system people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the system's value lies in its ability to make good matches between the recommenders and those seeking recommendations." (Resnick & Varian, 1997)

With *aggregation* in mind this definition prophetically defined *collaboration* between users as a means to generate *recommendations*. This was then further expanded to include *any* method for generating recommendations [13]:

> "Any system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options." (Burke, 2002)

This generalized and more colloquial definition can be complemented by the more technically oriented [11]:

> "More formally, the recommendation problem can be formulated as follows: Let C be the set of all users and let S be the set of all possible items that can be recommended...Let u be a utility function that measures the usefulness of item s to user c, i.e., $I: C \times S \Rightarrow R$, where R is a totally ordered set (e.g., non-negative integers or real numbers within a certain range). Then, for each user $c \in C$, we want to choose such item $s' \in S$ that maximizes the user's utility." (Adomavicius & Tuzhilin, 2005)

This definition, albeit not including group recommendations, is sufficient for the scope of this work, as it also does not consider group scenarios.

1.2 Types of Recommender Systems

As shown in Fugure 1, from both historical and approach perspective Recommender Systems can be roughly split into these strategies:

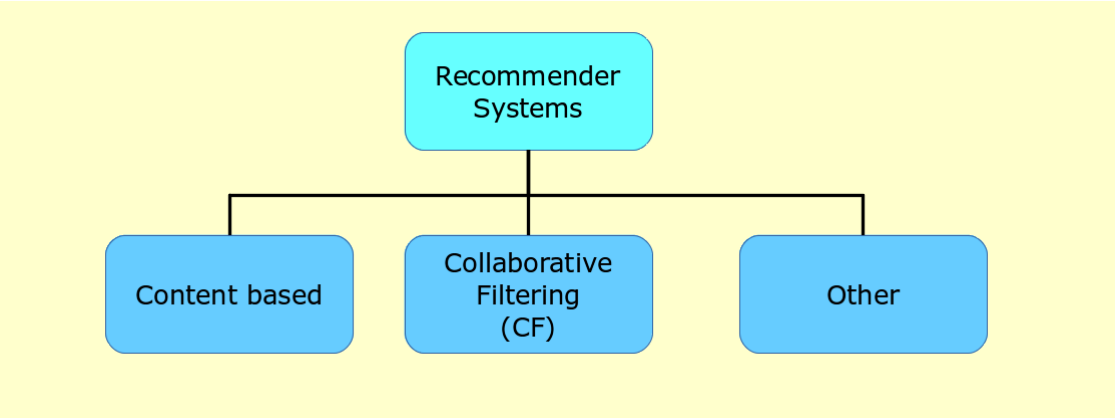

Fig. 1: Types of Recommender Systems.

1.2.1 Content based

Basically, profiling a user or an item, either implicitly or explicitly, in order to generate a personalized (profiled) recommendation. For items like movies such profiles, as an example, can take the form of: genre, director, actors involved, year of release etc. In the implicit case trained analyst assigns the item to a category; in the explicit case the users themselves categorize and score the items. User profiles on the other hand can contain various information like gender, age, preferred genre etc. This type of information is typically filled in provided forms by the users themselves. A prime example of this type of Recommender System is the Music Genome Project utilized by the Internet radio pandora.com[1]. Content based methods suffer from the fact, that they need to collect external information in advance, which is not always readily available or easy to collect [1].

[1] Pandora Internet Radio, URL: http://www.pandora.com (currently only available in: U.S., Australia and New Zealand).

1.2.2 Collaborative Filtering

Collaborative Filtering (shortly CF), which is the main focus of this work and its accompanying framework, is the most commonly used type of Recommender System and as a concept was named by the team that developed one of the first Recommender Systems - Tapestry [14][1]. Generally it is sub-divided into two main strategies or a combination of them:

1. *Nearest Neighbor* (methods), relies on finding similar users or items in order to predict future associations, hence the name.
2. *Model-* or *Factorization*-based Collaborative Filtering, relies on past user behavior (ratings) in order to generate a recommendation. Typically, this type of Collaborative Filtering works by utilizing various [1] Latent Semantic Indexing/Analysis (LSI/LSA for short) techniques in order to discover hidden relationships and interdependencies between the users and the items. This, then enables Model/Factorization-based CF to designate new user-item associations [6]. These associations or factors as technically termed, serve as a miniature (low rank) model of the original data (user-item matrix), that captures the most important aspects of the user-item interactions and is then used to generate recommendations with high accuracy [2].

Both methods do not need extra information like implicit data like: timestamps, viewing, renting, purchasing etc., but such information can and often is incorporated to boost prediction quality [7][9]. Unlike Content based Recommender Systems, CF's lack of need of gathering extensive profiling information makes it more universally applicable. This strategy also offers the potential to discover various domain-free patterns, which would be otherwise impossible to formulate in advance [2]. These traits made the technique a hot topic in the research community [18] with various projects [19] and real-life commercial applications [20]. Albeit its wide practical success, like in the Netflix challenge[2] [47], Collaborative Filtering is not without drawbacks. As a prime example that plagues both previously presented CF strategies, one can identify the so called Cold Start problem [30], to sum it

[2] http://www.netflixprize.com/

up: when a new user or item arrives, it lacks the necessary information (ratings) needed to assign it a group or to extract latent factors. For this problem various workarounds have been proposed bearing different degrees of success [31][32]. Collaborative Filtering will be thoroughly examined in the following chapters.

1.2.3 Other Recommender Systems

Systems such as - *Knowledge* based recommender systems and various hybrid approaches also exist mainly within the research world. Knowledge based recommender systems [21] employ various forms of deep knowledge, such as as semantic knowledge [22], instead of typical (con-)textual description or numerical ratings. This deep knowledge captures more properties about the items and can lead to richer recommender experience. Hybrid approaches, as the name suggests, try to combine the previously mentioned recommender strategies in different ways in order to achieve higher recommendation quality [13].

1.3 Motivation and Goals

This work takes inspiration from the extensive research done during the Netflix challenge [47]. During its course, various algorithms were developed [2][3][5][6][7][9][29][33][36][45][49][50][51][23] (and more) and further improved in order to tackle *already existing* and *newly arisen problems*. Among these algorithms one could distinguish: *Stochastic Gradient Descent*, *Alternating Least Squares, Restricted Boltzmann Machines, Markov Chain Monte Carlo* and various *Nearest-Neighbor* techniques that were used as machine learning approaches in order to tackle the Recommender System problem. Netflix further released a large dataset containing over 100 million ratings from over 480 thousand users on 17770 items which was used to *train, test, qualify* and *rate* and the models using the *RMSE* metric (more on *RMSE* in 4.2).

The motivating research objectives can be summarized in the following list:

- **Can a generalized implementation be drawn from the customized algorithms made explicitly for the Netflix dataset** – most of the algorithms are fine-tuned for the Netflix challenge and depend on large number of additional parameters that have to be explicitly trained.

- **A fresh evaluation of the accuracy of the selection of factorization algorithms on new and old datasets** – (1) how does the practical implementation, which is a part of this thesis, rate on the Netflix dataset compared to the ones used in the challenge. (2) How suitable are these algorithms for factorizing new datasets like MovieLens an IMDB. (3) Comparison of the results on old and new datasets.

- **Qualitative and Quantitative evaluation of performance metrics** – different algorithms have different demands on the hardware. With that in mind ones goal is to measure important performance figures like CPU time and memory usage and to draw conclusion of possible bottlenecks and improvement points in the selected algorithms.

- **Standalone algorithmic perspective** – since most of the winning approaches included ensembles of algorithms working in a feedback loop in order to achieve the desired goal of 10% improvement – from a research point of view a more atomized analysis of the different algorithms makes more sense, with fusion of algorithms being its own research field.

Derived from the objectives listed above, Table 1 shows the research questions and contributions of this work:

Questions	Contributions
Q1: *Is general* (works on all datasets) *implementation derivable?*	Yes, with remarks. The algorithms developed for the Netflix challenge work also well with other datasets, but require algorithmic parameters tuned for the task.
Q2: *How does the provided practical implementation stack against previous work?*	The accuracy of the practical implementation is consistent with the findings of from other works. This is true for older as well for newer datasets, confirming the universal applicability of the implemented algorithms.
Q3: *How does the selection of factorization algorithms perform on normal desktop hardware, and how do they fare against each other?*	The experimental results have shown that the selected algorithms can run in reasonable time with sufficient accuracy on a normal desktop PC. If proper optimization techniques are implemented both algorithms can run in similar times with comparable accuracy.
Q4: *Is an ensemble of factorization algorithms a necessity for achieving high prediction accuracy?*	No. Properly implemented and tuned algorithms can achieve high prediction accuracy on their own.

Tab. 1: Research Questions and Contributions

1.4 Outline

The next Chapter is dedicated to the major concepts of Recommender Systems and a selection of Algorithms from these concepts. The algorithms themselves are introduced and discussed. Chapter 3 presents the datasets used for obtaining the experimental results via a framework (which accompanies this work) of two matrix factorization algorithms. Later on in Chapter 4 these experimental results are exhibited in a graphical fashion in or-

der ease the analysis of the implications of the results. The key findings of this work are discussed in chapter 5. Finally Chapter 6 concludes this work with a general discussion about the implemented and tested algorithms and proposes means for improvement of their performance.

2 Theoretical Overview

This chapter provides theoretical overview of the core concepts of Collaborative Filtering with emphasis on Matrix Factorization. The beginning deals with the introduction of some of the most common Nearest Neighbor approaches and then the chapter continues with a general Matrix Factorization (short MF) model. This is then extended by various techniques for improving accuracy and dealing with cold start problems, these include implicit and explicit feedback. Finally a selection of two inference methods is introduced. These methods are also a part of the practical implementation and framework and are more thoroughly and pragmatically covered in the sub-chapters.

2.1 Collaborative Filtering

This thesis' main focus of work is an algorithmic perspective of matrix factorization for recommender systems. This algorithmic approach, is itself a part of a broader collection of techniques utilizing similarities between users or items. This collection is commonly referred to as *"Collaborative Filtering"*. Figure 2, as a complementation to Fig. 1 from Chapter 1, aims to draw a map of the different Recommender Systems and the relevant for this work algorithmic approaches.

Collaborative Filtering, as a paradigm, aims to generate predictions (used as recommendations, thus filtering irrelevant items/products) that are relevant to the interests of a particular user by rallying *similar* users (collaboration) and using their shared *preferences*. The general premise of Collaborative Filtering is that if user A has the same preference for products/items rated by user B (and also rated by A), then user A is going to have the same preferences as user B for other (non-rated by user A, but

rated by B) items/products. For illustrative purposes, in a typical collaborative filtering recommender system for movies or series like Netflix or IMDB, the system can make predictions about which movies or series user A could find relevant by using only user A's preferences (e.g. ratings or even binary information - like/dislike) and interpolating from that information by finding similar users with similar preferences and using their already seen movies or series as a base for recommendations. In other words, although these predictions are specifically bound to a particular user, the predictions themselves depend on other, *similar* users in order to be generated. This approach is more complex and precise, than for an example a typical linear or polynomial regression approach, which gives an average (not specifically bound to a particular user) prediction for each potential item/product. Or differently formulated – taking the global (of all items) or local mean score (for this particular item) does not enough to capture a specific user's preferences.

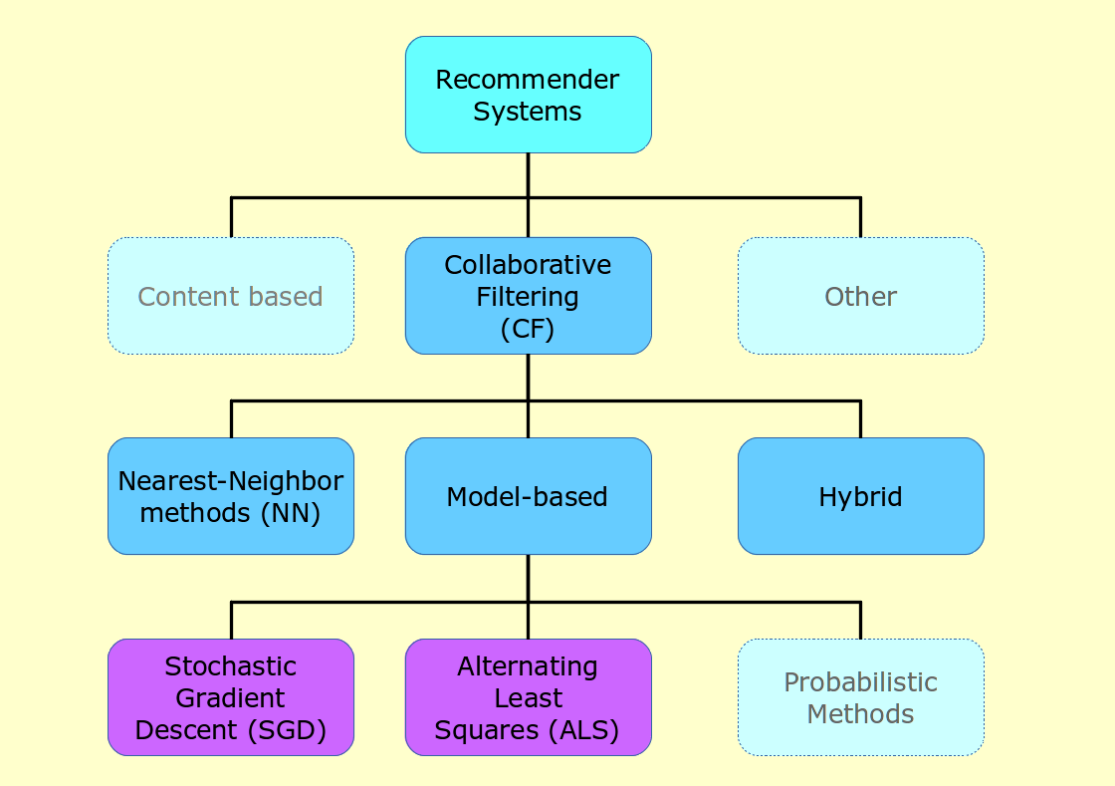

Fig. 2: Types of Recommender Systems and relevant subtypes. Recommender Systems (blue), relevant sub-types (blue), algorithmic approaches popular in the Netflix challenge and taken as basis for the practi-

cal implementation of this thesis (violet), faded: other types of algorithmic approaches.

As an alternative definition, the CF task can be viewed as a *missing value estimation* [6]. What is known: a dominantly sparse user-item matrix of ratings. What needs to be done: an estimation of the missing values in the sparse matrix. How it shall be done: based on the given ratings - predict the missing ones. The given user-item ratings are means to measure the *degree of interest* between users and items. These ratings are typically explicitly given by the users themselves by utilizing the rating capability of the commercial platform and serve as the input of the Collaborative Filtering algorithms. In the relevant literature one can also find the term *"score"* in place of *"rating"* with the same semantic meaning. Figure 3 illustrates the process.

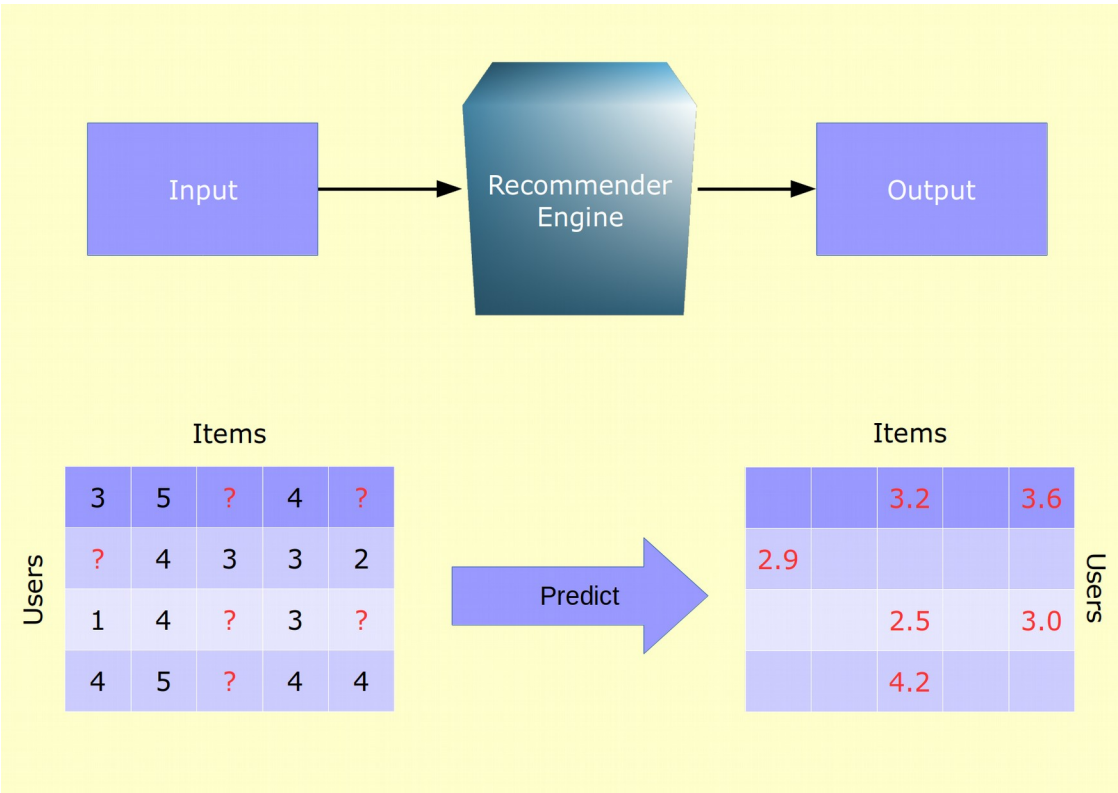

Fig. 3: The missing value estimation process.

Alternatively, in order to tackle the cold-start[3] problem or users with insufficient number of rated items, one can use (often) implicit data from previous interactions like historical purchases (but left unrated) or browsing of items.

2.2 The Nearest-Neighbor Approach

At the beginning the most common approach to Collaborative Filtering were various Nearest Neighbor methods (also known as k Nearest Neighbors, shortly kNN) [6][11]. These methods use various similarity measures [10][11][22] in order to identify groups of items that tend to get rated similarly. This approach can also go both directions [53] and thus can be used to find groups of users that share similar preferences in items. These groups of similar items or similar users are then used to discover new *interconnections* between the users and the items. The whole concept lies on the assumption that previous preferences of users will hold with time in a consistent and stable matter. [53] Figure 4 illustrates the process.

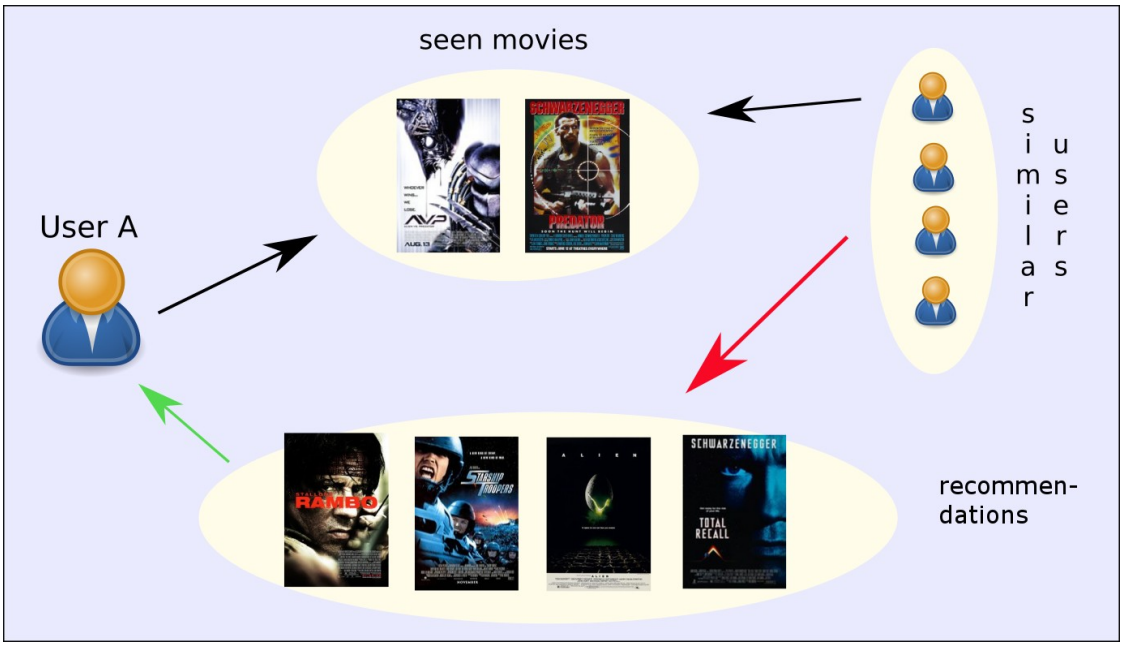

Fig. 4: The Nearest-Neighbor approach.
In order to generate predictions for User A, this approach aims to find a group of users that have already seen the movies seen by User A (black arrows), hence indicating similarity, and then generate a list from this similar

[3] cold-start - no or insufficient data at the beginning

user group (red arrow) and present it to User A (green arrow). (All images courtesy of their respective owners)

As positive feature of this approach one can distinguish:

1. **Ease of implementation** - compact and uncomplicated algorithms
2. **Computational performance** - fast computation times with low memory footprint
3. **Intuitive recommendations** - ease of explanation to the user of the rationale behind a recommendation

For a more balanced overview of the kNN approach, one should also consider its drawbacks. The main one can be considered the fact, that kNN epitomizes the so called *"lazy learning"* approach. This means, that the algorithm does not learn any useful props (like in feature extraction) from the training data, but rather uses the data itself for the classification task (side-note here: of the nearest neighbors or similar users/items) and the subsequent prediction task. Because generating a prediction is done via finding the k closest neighbors, the prediction itself will be the most common score among the k nearest neighbors, thus skipping any global effects until late in the learning phase. This makes drawing generalizations about the data difficult, which further makes the algorithm prone to have low resiliency for noisy data. Another serious drawback of the kNN algorithm is that for each training example, it must calculate the distance of its nearest neighbors and then sort them. This can be computationally expensive for large datasets (although using indexing like L-D trees may reduce the overhead). Further one must also determine a proper k value for the number of associated neighbors. Also in all distance-based learning techniques, there is no clarity which attribute one should use, or which distance measure.

In the core of the kNN algorithm lays the similarity measure function used to determine the nearest neighbors. Popular choices are the *Pearson* correlation coefficient and the closely related *Cosine* correlation. The former is typically used for user-based recommendations, while the latter mostly in item-based scenarios. One advantage of the Pearson correlation coefficient over the Cosine is the fact that its range is between -1 and 1,

where -1 indicates negative correlation and +1 indicates positive correlation [53]. The cosine correlation, which in its basic form covers the range of [0,1], however it can be modified to the -1 and +1 range. The main advantage of this modified cosine measure is that it takes into account the average rating behavior of the users, by subtracting the average value from the ratings. [53]

2.2.1 The Pearson Correlation Coefficient

Equation 1 depicts the Pearson Correlation coefficient for a typical user-based recommendation:

$$sim(a,b)=\frac{\sum_{i\in I}(r_{a,i}-\overline{r_a})(r_{b,i}-\overline{r_b})}{\sqrt{\sum_{i\in I}(r_{a,i}-\overline{r_a})^2}\sqrt{\sum_{i\in I}(r_{b,i}-\overline{r_b})^2}}$$

Equation 2.1: The Pearson Correlation Coefficient

In order to fully understand the mechanics behind correlation-based recommendations, one should consider the following example:

	User 1	User 2	User 3	User 4	User 5
Item 1	2	3	-	5	-
Item 2	5	-	-	-	-
Item 3	-	3	4	3	3
Item 4	-	5	5	-	4
Item 5	-	-	3	-	-
Item 6	-	4	4	-	4
Item 7	1	-	-	-	-
Item 8	-	2	-	-	-
Item 9	-	-	-	2	-
Item 10	-	-	-	1	-
$\bar{r}$	2.67	3.4	4	2.75	3.67

Tab. 2: Example Collaborative Filtering Dataset

Example rating matrix: items and ratings by users (ratings in range of 1–5). Empty cells indicate no rating for that item by this user. $\bar{r}$ - denotes the mean rating value for this item.

Using Equation 2.1 and the example dataset above, one can calculate the following correlation coefficients or similarities for users 2 and 3 in respect to user 5, shown in Table 3:

	User 1	User 2	User 3	User 4
User 5	-	0.866	0.5	-

Tab. 3: Example similarities

Example similarity values for User 5, since users 1 and 4 did not rate enough or any items that User 5 rated, no similarity measure for these users can be obtained.

Building upon the correlation coefficients calculated above one can proceed and generate predictions using the following Equation [53]:

$$pred(a,i) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,i} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

Equation 2.2: Prediction for Pearson Correlation Coefficients

Applying the Equation above one obtains following predictions:

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	Item 10
User 2	3	-	3	5	-	4	-	2	-	-
User 5	-	-	3	4	-	4	-	-	-	-
Prediction User 5	**3.27**	-	-	-	-	-	-	**2.27**	-	-

Tab. 4: Example user-based predictions

Predictions generated by user-based collaborative filtering for items not rated by User 5, but rated by User 2 (most similar user) - see Fig. 4. These predictions can later be sorted and used as recommendations.

2.2.2 The Cosine Correlation Coefficient

As an alternative to the user-based approach, the item-based approach emerged from the need of a faster calculation method in the context of a very large user/item database [4]. This approach utilizes the fact that there are typically far less items than users, so determining their neighborhoods is computationally less expensive [4]. The basic idea is to examine User A's rated items and determine his/her similar items. The algorithm then calculates a similarity measure and feeds it to a prediction function. Typically the cosine and adjusted cosine similarity measures are used [53] for determining similarity between the items. Both approaches c*alculate the similarity between two n-dimensional vectors* by using the *angle between them* [53]. They are also commonly used in information retrieval and *tf-idf* problems[4]. The cosine similarity measure is defined as follows [53]:

$$sim\left(\vec{a}\,,\vec{b}\right)=\frac{\vec{a}\,.\,\vec{b}}{|\vec{a}|*|\vec{b}|}$$

Equation 2.3: The cosine similarity measure

By applying the Equation above on the example dataset defined in Table 2, one obtains following similarities:

	User 1	User 2	User 3	User 4	User 5
Item 3	-	3	4	3	3
Item 6	-	4	4	-	4
Similarity:	**0.99**				

Tab. 5: Example cosine similarity

Example similarity values for Item 3 and Item 6, only overlapping values are taken into account. The possible range lies between 0 and 1, where values approaching 1 indicate strong similarity.

[4] http://www.ir-facility.org/scoring-and-ranking-techniques-tf-idf-term-weighting-and-cosine-similarity)

2.2.3 The Adjusted Cosine Correlation Coefficient

The basic cosine coefficient, however, does not take into account the average rating pattern of the current user. This is remedied by taking the adjusted cosine similarity coefficient [53]. This approach works by first subtracting each user's average value from the ratings. By doing this, the range now lies between -1 and 1 as in the the Pearson approach. The similarity is calculated with the following formula:

$$sim(a,b)=\frac{\sum_{u\in U}(r_{u,a}-\overline{r_u})(r_{u,b}-\overline{r_u})}{\sqrt{\sum_{u\in U}(r_{u,a}-\overline{r_u})^2}\sqrt{\sum_{u\in U}(r_{u,b}-\overline{r_u})^2}}$$

Equation 2.4: The adjusted cosine similarity measure

In order to continue, first the example dataset from Table 2 must be adjusted by subtracting the user's average values.

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	Item 10
User 1	-0.67	2.33	-	-	-	-	-1.67	-	-	-
User 2	-0.4	-	-0.4	1.6	-	0.6	-	-1.4	-	-
User 3	-	-	0	1	-1	0	-	-	-	-
User 4	2.25	-	0.25	-	-	-	-	-	-0.75	-1.75
User 5	-	-	-0.67	0.33	-	0.33	-	-	-	-

Tab. 6: Average-adjusted ratings

Example rating matrix: the values depicted are after subtracting the average user rating value and transposed for convenience

With the average ratings subtracted, one can choose for an example items 4 and 6, and then proceed calculating the similarity values by using Equation 2.4. Table 7 depicts the results:

	User 1	User 2	User 3	User 4	User 5
Item 4	-	1.6	1	-	0.33
Item 6	-	0.6	0	-	0.33
Similarity:	**0.815**				

Tab. 7: Example adjusted cosine similarity

Example similarity values for Item 4 and Item 6, only overlapping values are taken into account. The possible range lies between -1 and 1, where values approaching 1 indicate strong similarity and values approaching -1 indicate negative similarity.

After the calculation of the adjusted cosine similarity values has been taken care of, the next step is to generate predictions using the similarity coefficients. This can be done by using Equation 2.5 [53]:

$$pred(u,i) = \frac{\sum_{j \in ratedItems(u)} sim(j,i) * r_{u,j}}{\sum_{j \in ratedItems(u)} sim(j,i)}$$

Equation 2.5: Prediction by adjusted cosine similarity measure

Utilizing Equation 2.5 on the data from Table 7, one obtains: *prediction(User 4, Item 4) = 3.0 ,* i.e. based on similar items, User 4 would rate Item 4 with a rating of 3.0 .

2.2.4 Related work

During the course of the Netflix challenge many teams first approached the problem from the NN perspective with the main innovations coming from the team of Y. Koren and R. Bell (team “BellKor”) which won the challenge afterwards. In [6] they introduce an improved neighborhood technique, which trains all features simultaneously with minimal performance penalty. This approach also takes global biases into account and is tested on the Netflix dataset. In another paper [2] from Koren et al., the researchers improve the kNN method in two directions. First as in the previ-

ous paper - by training all of the interpolation factors simultaneously, by solving a global interpolation problem. And second, by applying the method on the slower user-oriented approach via a low-dimensional embedding of the users. In another paper [34] again inspired by the Netflix challenge an Austrian team consisting of A. Toescher, M. Jahrer and R. Legenstein which later joined “BellKor” in winning the prize, proposed a regularized matrix factorization algorithm which includes neighborhood information.

2.3 The Matrix Factorization Approach

Matrix Factorization is a popular method for performing Latent Semantic Indexing on various datasets, be it commercial or scientific [49][50][51][2][3][5][6][7][9][36][45][23]. In the case of application in a commercial recommender system, the method aims to infer (extract) *n* number of *factors* (also called *features*) which generalize an item's properties and a user's preferences (for an example: and online DVD rental service and its user-base). These features, normally in the range from 8 to 50 [2][3][5][6][7][9][1], are *inferred* from existing, explicit data (usually ratings) with no previous knowledge of their structure, hence they are *latent* in the data. The ratings are placed in a *matrix* where one dimension represents users and the other items. Inference is done in a row-wise fashion, hence *indexing*. The matrix itself is largely *sparse* due to the fact that the majority of the users have rated only a small proportion of the items. And with the goal being *inferring features* from this sparse matrix, which capture the essence of the items' properties and users' preferences, one comes to the term *semantic*. The terminology of *matrix factorization* comes from the analogy of this technique to Singular Value Decomposition which it closely mimics. The inferred features are placed in vectors and represent sometimes easy to understand dimensions like comedy, drama or thriller (for a movie oriented example). But more often than not, the features represent opaque dimensions like *geekiness* (movies with traits preferable by "geek" culture), *tarantinoesque*[5], or completely uninterpretable dimensions. A simplified example for such a latent factor space can be seen in Fig 5.

[5] https://en.wiktionary.org/wiki/Tarantinoesque

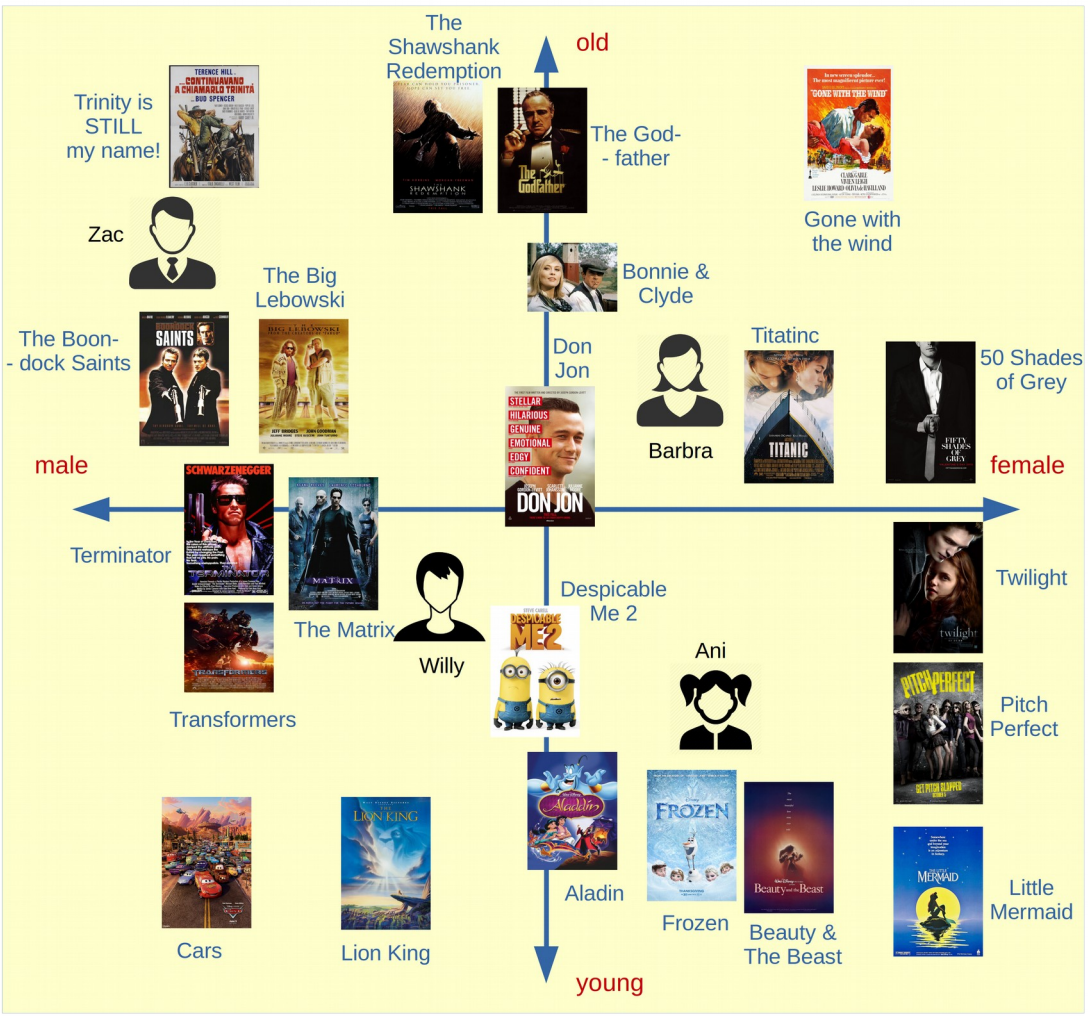

Fig. 5: The Matrix Factorization Approach.

This low-dimensional example splits the traits of users and movies into two dimensions. The first one is male / female orientedness and the second one is old / young viewership. Aligned on those dimensions there are several well-known movies and phew fictional users which aim to cover the whole spectrum of the aforementioned dimensions. For this type of recommender system a prediction is generated by taking the dot product of the user's and movie's points on the plane. As an example one can consider the user Willy who likes Transformers and The Matrix to not like Titanic. He would also give an average rating to The Shawshank Redemption and be thrilled by The Terminator. (All images courtesy of their respective owners)

Pair matching or recommendation generation is done when a high correlation between both user's and item's *feature vectors* is present. This mostly automated *praxeology*[6] has the positive traits of both good scalability and good recommendation accuracy. This made matrix factorization very attractive and popular as well for theoreticians as also for practitioners.

Another positive trait of Matrix Factorization is its flexibility in modeling real-life applications by adding *implicit* feedback to additionally enhance the quality of the extracted features. But in order for one to understand the importance of implicit data, let us first recap on explicit data. Types of explicit data are: *ratings* (collected by Netflix, IMDB, MovieLens) and *like/dislike* rudimentary indication of user preferences (collected by TiVo). Ratings typically have a range e.g 1-5 “stars” for Netflix or 0-10 score for IMDB, with higher values indicating higher interest. All these examples refer to companies specialized in movie/TV/Series types of entertainment products and their datasets comprise of highly sparse user/item matrices. As already mentioned this is so, because each user typically rates only a small portion of all the movies. This of course makes drawing generalizations on new users with few or no ratings challenging (the so called “*cold start problem”*). In contrast, implicit data is normally comprised of dense matrices, examples of types of data they hold include: viewing history, browsing history, search patterns and even mouse movements. As an example: a user that has seen many episodes of a series probably likes the series. The data is often stored in binary fashion and indicates the presence or absence of an event (e.g. user *u* browsed item *i*). Albeit implicit data is not scarce, the indirect nature of how it is collected, limits its capabilities to accurately draw generalizations by it alone, hence it is used to boost prediction accuracy and tackle *cold starts* [1].

From technical perspective Matrix Factorization operates by mapping users and items to a *joint latent factor space* with dimensionality *f*, in a way that allows user-item interactions to be modeled from the inner products of the feature vectors in this space [1][5]. From this follows, that each

[6] https://wiki.mises.org/wiki/Praxeology

item *i* and each user *u* is *described* by a vector $p,q \in \mathbb{R}^f$, where *p* is often reserved for users and *q* for items [1][5]. The elements of vector *q* describe the *features* of item *i*, i.e. how well an item scores on a certain factor - positive or negative. From the other side the elements of *p* describe the *interests* of a user *u* in items scoring high on the respective factors, again, they can indicate positive or negative correlation. Generating recommendations is done by taking the *dot product* between $q_i^T p_u$, the result captures the extent of the correlation between user *u's* interests in item *i's* properties. This approximation or prediction tries to mimic a rating for the particular item, as if it has been rated to by the user him/herself. The predicted rating takes the form of:

$$\hat{r_{ui}} = q_i^T p_u$$

Equation 2.6: Prediction by feature vectors

and is often used for *generating* recommendations (by picking the ones with the highest score) and *measuring* the accuracy of the used algorithm(s) by comparing the predicted ratings against the real ones (more on that follows). The predictions can also be fed to other Recommender Systems algorithms in order to boost the overall performance of the combined system [49][50][51]. This is generally done by chaining an *ensemble* of factorization algorithms [49][50][51] or in some cases in a synchronous matter. As it can be noted (seen in Equation 2.6) this is not a costly operation. The *difficulties* arise from inferring the feature vectors themselves. Only after the costly inference of the vectors is done, one can proceed and estimate a rating via eq. 2.6.

Feature factor inference is done in a similar fashion related to *Singular Value Decomposition* or SVD for short. SVD is a commonly used technique for feature inference in information retrieval [1][5], but in its pure form it can not operate in highly sparse datasets, which is the case in the conditions of explicit feedback systems. In other words since each user has rated only a small portion of the overall items, conventional SVD can not factorize the user-item matrix, as its behavior is undefined in the presence

of missing values. To further complicate the matter, selecting naive approach for the feature vector inference often leads to *overfitting*.

Overcoming the sparseness problem in early systems was done by imputation [1][5]. As Koren already noted in [1][5] using imputation is impractical as it drastically increases the size of dataset, this penalizes the performance significantly [1]. Further imprecise imputation can warp the data and lead to misinterpretations [1][5].

With imputation rendered impractical, additional research has been directed into the development of customized algorithms that factor the sparse matrix *directly* [1][5]. To deal with the omnipresent problem of overfitting, L_2 regularization technique was incorporated into the factorization algorithms. With that in mind a typical [1][5] factorization approach tries to minimize the *regularized squared error* on the dataset (the existing ratings), thus inferring the features of the user and item vectors p and q:

$$\min_{q^*,\,p^*} = \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda\left(\|q_i\|^2 + \|p_u\|^2\right)$$

Equation 2.7: Optimization problem of a Regularized Factorization Model

The equation follows Koren's notation as used in [1]. Here, κ represents the set of the user/item *(u,i)* pairs for which the rating $r_{u,i}$ is known. Inferring is done by modeling the vectors on the existing ratings, while combating overfitting by regularizing the model parameters in order to attain best possible generalizations, suitable for high quality predictions for the unobserved ratings. In short: this is done by penalizing high-scoring features so that low-scoring ones do not get overshadowed and their (vital) information lost. The regularization parameter λ is arbitrarily chosen or can be determined by cross-validation.

2.3.1 Global Factors

Before we begin with the learning algorithms themselves, perhaps this is a good spot to shed some light on the fact that the majority of the ratings are influenced by effects either on the user or on the item side. These *uni-*

lateral effects take the form of *biases*, which are systematic tendencies of certain *items* to be rated *higher* or *lower* than others, or certain *users* giving *higher* or *lower* (biased) ratings than other users [1][5]. As an example one can take the global average of all movies in the Netflix dataset with value ~3.6 (in the range of 0-5), then select a movie with a higher than the average rating - like *"The Godfather"*, which is rated with 4.4 (0.8 higher than the median) and assume a generally positive user that rates movies on average with 0.4 point higher than the other users. This leaves us with the following baseline estimation: 3.6 + 0.8 + 0.4 = 4.8. This calculation can be generalized into:

$$b_{u,i} = \mu + b_u + b_i$$

Equation 2.8: Baseline predictors - this Equation follows Koren's notation [1] with μ being the global average rating and b_u and b_i being the user and item unilateral biases which leaves b_{ui} as the desired baseline predictor for this user/item pair.

By using baseline predictors, one tries to remove the aforementioned biases from the final model, thus eliminating the unnecessary *unilateral* noise and leaving the factorization algorithms to capture the vital *multilateral* user-item interactions. This makes the baseline predictors of great importance for the *proper initialization* of the *factorization algorithms* and is typically done at the beginning of the learning process. Or to sum it up: one tries to capture how much *actually* does a user like or dislike a particular item *and* is not influenced by his/her own rating inclinations or the item's tendencies to be over/under-rated, which leaves purified data to be fed to the next step of the learning process - factorization, this in result shall give better prediction accuracy. By integrating the global biases to the factorization model one infers the following Equation [1]

$$\hat{r_{ui}} = \mu + b_i + b_u + q_i^T p_u$$

Equation 2.9: The components of the integrated prediction formula, where both unilateral and bilateral biases are taken into account.

After the biases are integrated, the inference process can be accomplished by minimizing the squared error function for all types of parameters:

$$\min_{q^*, p^*, b^*} = \sum_{(u,i)\in K} (r_{ui} - \mu - b_i - b_u - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Equation 2.10: The objective function of the model, integrating both unilateral and bilateral interactions (also called global biases or factors). It serves as a basis for evaluation of the inferred parameters, which are learned via either SGD or the ALS algorithms. Note the regularization parameter λ used to counter overfitting, which is selected apriori.

As finishing thoughts on biases one can note, that Koren et. al. [1][9][5] further recognized other types of so called *first* order (unilateral) interactions. For an example in his paper - "Collaborative Filtering with Temporal Dynamics" [36] he proposes various ways of dealing with *temporal* phenomena, which capture the alteration of the rating tendencies of the users and the fluctuations of popularity of the items over time. To illustrate, let us take movies with Christmas theme are rated higher and are more relevant around Christmas. Also users tend change their tastes slowly over time, or accounts get handed over.

Temporal interactions are not supported by the framework accompanying this work, this is in order to stay in line with the idea of keeping the experimental setup more general, because not all datasets may provide this kind of information, therefore the theoretical basis for calculating time related coefficients has been left outside of the scope of this work. For more information on temporal interactions please refer to the aforementioned work - [36]. There is, however, an additional source of information that is present in *all* datasets and which is is described in the next chapter.

2.3.2 Implicit input sources

The infamous *"cold start"* problem resulting from the fact that a large proportion of the users have rated only a small subset of the items leads to inaccurate conclusions about their preferences [1]. These types of conditions are called *sparse* and prevail in Recommender system datasets where the number of items is large enough so that the majority of the users are not able to rate them all or to a higher degree. This inherent property of these *voluntary input* datasets is most proficient at the beginning – hence its name – the "*cold start"* problem. To alleviate this, additional (to the gen-

eral factorization) measures have to be taken. One intuitive way of tackling missing information is by introducing new, *implicit* input (information) sources. As the name applies this is done in a fashion that circumvents the user's eagerness to provide additional data and can incorporate various metrics like browsing/view history and existing purchases (with and without rating!). This work discusses implicit feedback in the fashion of Koren et. al. in [1] and Paterek in [7]. Technically the additional sources represent normalized[7] (Koren) or weighted[8] (Paterek), *dense, binary linear models* [7] which are directly incorporated into the larger model. These include data about past "views" (ratings) in binary format i.e. 1 for viewed/rated (implying interest) and 0 otherwise (implying no interest). The addition of the binary models has also been shown to have the same positive effect of enhancing general accuracy also for other, than the discussed in this work, types of factorization algorithms [5][33]. This boosting of prediction accuracy is also true when there is *no* implicit feedback and one has *only* the explicit ratings, as shown in [nhood] Koren has found that, making the addition of a new factor set for *"shown explicit and implicit interest"* is beneficial for both cases – when there *is* implicit feedback and when there *is not*. To illustrate how this works, let us denote *N(u)* as a set which contains all items with implicit input for user *u*, these are the "1"s of the dense binary model discussed above, the "0"s are all the other items. We then store the implicitly gathered preferences in a new set of item factors, following Koren [1] and Paterek's [7] ideas. In them an item is associated with $x_i \in \mathbb{R}^f$, and for a user that showed interest for the items in *N(u)*, one gets the following characterization vector in non-normalized form [4][5]:

$$\sum_{i \in N_{(u)}} x_i$$

Finally, by integrating the implicit feedback into the prediction formula, one gets:

---

[7] Normalized binary additional input model [koren]: $|N(u)|^{-0.5} \sum_{i \in N_{(u)}} x_i$

[8] Weighted binary additional input model [paterek]: $m_j + e_i * \sum_{j_2 \in J_{(i)}} w_{j2}$ here everything is the same as in the Normalized model except the addition of the constant m_j which is the mean rating of the movie *j*.

$$\hat{r_{ui}} = \mu - b_i - b_u - q_i^T \left[p_u + |N(u)|^{-\frac{1}{2}} \sum_{i \in N_{(u)}} x_i \right]$$

Equation 2.11: The integrated prediction formula, where additional implicit input sources are taken into account by the summation of x_i.

2.3.3 Additional explicit input sources

Although explicit feedback is not used in the practical implementation in the accompanying framework, for the sake of completeness it will be introduced here as it takes a part in various other approaches for matrix factorization [1]. Before we begin with the introduction of the explicit input sources, one has to note that at this spot it is fairly easy to add various kinds of additional input sources in the same fashion as the implicit input sources from the previous chapter. Examples for explicit feedback can be comprised of various distinguishing characteristics such as: temporal data (mentioned in chapter 2.3.1), which can be used as an example to measure the up-to-dateness of the data a.k.a. older ratings have lesser influence; or arbitrary user attributes such as: age, gender, geographic location or other demographic statistics. Again caution should be taken as such information may be sensitive and its usage may be unwanted for specific regions, platforms or even individual users. This creates additional requirements for the factorization algorithms in the sense that:

1. **flexibility** - they shall be able to incorporate such input sources (if they are available and usage is permitted) in a fashion that does not require heavy modification of the existing parts of the factorization models
2. **robustness** - they shall be able to work in a stable fashion with or without additional explicit input sources or in other words - these sources shall only add to the accuracy of the predictions, not comprise the bulk of it

Fortunately the approach introduced in the previous chapters complies to both of these requirements, by adding additional explicit input to the Equation 2.11 one gets:

$$\hat{r_{ui}} = \mu - b_i - b_u - q_i^T \left[p_u + |N(u)|^{-\frac{1}{2}} \sum_{i \in N_{(u)}} x_i + \sum_{a \in A_{(u)}} y_a \right]$$

Equation 2.12: The integrated prediction formula, where additional explicit input sources are taken into account by the summation of y_a, here y_a represents an individual (per user) attribute vector that stores the various user attributes.

Note, that the data format must be adjusted accordingly, as the binary format from the previous chapter can prove to be insufficient or incompatible for capturing the depth of the new data encoded in y_a. Finally we can derive the full objective function including both explicit and implicit input data:

$$\min_{q^*, p^*, b^*} = \sum_{(u,i) \in K} \left(r_{ui} - \mu - b_i - b_u - q_i^T \left[p_u + |N(u)|^{-\frac{1}{2}} \sum_{i \in N_{(u)}} x_i + \sum_{a \in A_{(u)}} y_a \right] \right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 + \sum_{i \in N_{(u)}} \|x_i\|^2 + \sum_{a \in A_{(u)}} \|y_a\|^2 + b_u^2 + b_i^2 \right)$$

Equation 2.13: The integrated objective formula, where additional explicit input sources are taken into account. Note that the terms x_i and y_a are also regularized.

2.3.4 Inference via Stochastic Gradient Descent

One of the two algorithms implemented in the course of this thesis and a part of the accompanying framework is the widely known, basic, learning *(inferring)* algorithm – *Stochastic Gradient Descent*. It was selected following the steps of the various teams that participated in the Netflix[9] challenge. This method was firstly introduced by Simon Funk[10] and later adopted and extended by the other participants [1][7]. Its main advantages include: *ease of implementation, customizability (making extended versions like: e.g. incorporating additional input sources), fast running times* and *low memory footprint*. As disadvantages one can identify: the

[9] https://www.netflixprize.com/

[10] S. Funk, “Netflix Update: Try This at Home,” Dec. 2006; http://sifter.org/~simon/journal/20061211.html.

need of *tuning* the different *learning parameters* and the need of a coping *mechanism against overfitting* (typically this is done via *regularization*). Table 8 summarizes the advantages and disadvantages of Stochastic Gradient Descent:

Advantages	Disadvantages
Ease of implementation: this algorithm is among the easiest to understand and implement, it is also widely used and studied at different learning institutions, hence it is comprehensively covered in literature and has a broad base of examples online.	**Fine tuning of learning parameters:** SGD needs properly tuned: number of factors, iteration count and step size. While the former two parameters are common in various learning algorithms, the latter is key for the quality of the results. If one chooses a large value for the step, the algorithm will converge faster, but will be inaccurate, choosing a small value may lead to poor running to times or no convergence at all.
Customizability: one can modify and adapt the inference algorithm to suit one's needs and current problem definition, thus inferring various factors which may be a part of a custom learning algorithm.	
Extendability: one can make extended versions of existing learning algorithms by inferring parameters from additional input sources like: time of rating, browse/purchase history and other forms of *implicit feedback.*	**Coping mechanism for overfitting:** a mechanism has to be chosen and adapted. Typically [1] L_2 *regularization* [48] is chosen. Various contenders [1][7] of the Netflix challenge have found L_2 regularization to be working best for the Netflix dataset, implying that the sample size grows linearly in the number of irrelevant features [48]. Further the regularization parameters must be tuned, this can be problematic for users with large variance (i.e. rating either very high or very low) since they will need more regularization than users that rate with smaller variance (where conclusions are more easily drawn).
Fast running times low memory footprint: run time grows in linear fashion with the number of parameters and learning iterations/epochs. Memory footprint also grows linearly with the number of parameters since the algorithm does not need to store additional data in order to run.	

Tab. 8: Advantages and disadvantages of SGD

At its core Stochastic Gradient Descent tries to minimize the error between the prediction r_{ui} and the actual rating, this is for all the factors in the factor vectors:

$$e_{ui} \stackrel{\text{def}}{=} \hat{r_{ui}} - q_i^T p_u$$

Equation 2.14: Basic calculation of the error (difference) between the predicted value and the actual rating value from the training set.

But is most often used to infer the whole model from Equation 2.10, including global biases and also for inferring the implicit/explicit data from Equation 2.13.

Having the error in hand, the algorithm adjusts the parameter values with a degree γ (learning factor) in the opposite direction of the gradient of the error [1]:

$$p_u \leftarrow p_u + \gamma . (e_{ui} . q_i - \lambda . p_u)$$
$$q_i \leftarrow q_i + \gamma . (e_{ui} . p_u - \lambda . q_i)$$

Equation 2.15: Adjusting the values of the feature vectors by a factor of γ in direction opposite to the gradient of the error ("gradient descent"). Further prevention for overfitting is being governed by the regularization parameter λ.

For the global factors the process is similar and it is beneficial if they are regularized as well:

$$b_u \leftarrow b_u + \gamma . (e_{ui} - \lambda . b_u)$$
$$b_i \leftarrow b_i + \gamma . (e_{ui} - \lambda . b_i)$$

Equation 2.16: Adjusting the values of the global factors in the same fashion as for the feature vectors in Equation 2.15. It is to be noted that the regularization parameter λ can differ from the one in Equation 2.15 if needed, here for simplicity it is the same.

For a full-blown listing of all of the important variables from a complete run of 10 iterations on a very simple toy dataset and without any additional "weight" in the form of global factors or implicit/explicit data, just pure inference via Stochastic Gradient Descent, see: Appendix I. For discussion of

performance, see: Chapters 5 and 6. For potential improvements and future work, see Chapter 6.

2.3.5 Inference via Alternating Least Squares

At the beginning both q_i and p_u are unknown in Eq. 2.7, which represents an non-convex problem [1]. By initializing them in the fashion of SGD one can convert this problem to a quadratic one which is optimally solvable [1] [2]. A common way of doing this is to: oscillate between fixing one side (e.g. q) and computing each value of p via the solution of a least squares problem, and then doing the same for q. This switching and calculating is repeated until convergence [1][2]. Since the basic implementation of an Alternating Least Squares method requires f^3 of running time [3] (where f is the dimension of the feature vectors p and q), various techniques have been proposed for overcoming this problem. Some of them include representing the items as "bags of users" [50][51] which reduces the complexity to f^2 which still won't be efficient if there are a lot of items. Thus I have stopped myself at an implementation that utilizes a form of *dynamic programming*[11] technique [3] (basically sacrificing memory for CPU time) that also incorporates various mathematical optimizations in order to reduce the run time to a linear time [12]. Table 9 summarizes the advantages and disadvantages of the ALS algorithm:

Advantages	Disadvantages
No need of learning factor hyperparameter: ALS fixes one entry in a feature vector at a time, then optimizes until convergence, so no need of explicit learning factor is necessary.	**Basic version of the algorithm runs in f^3:** *an inherent property of* the algorithm is that in its basic form it is very slow. So a version that avoids this is needed.
Parallelization: ALS works by computing each q_i independently of the other item factors and then computes each p_u independently of the other user factors. This opens	**Coping mechanism for overfitting:** a mechanism has to be chosen and adapted. Typically L_2 *regularization* [48] is chosen. Various contenders [1][7] of the Netflix challenge have found L_2 regularization to

[11] https://www.geeksforgeeks.org/dynamic-programming/

an opportunity to massively parallelize the algorithm [1][44].	be working best for the Netflix dataset, implying that the sample size grows linearly in the number of irrelevant features [48]. Further the regularization parameters must be tuned, this can be problematic for users with large variance (i.e. rating either very high or very low) since they will need more regularization than users that rate with smaller variance (where conclusions are more easily drawn).
Extendability: one can make extended versions of existing learning algorithms by inferring parameters from additional input sources like: time of rating, browse/purchase history and other forms of *implicit/explicit feedback.*	
Handling of implicit data: since this type of data is not sparse the means of operation (iterating over all training cases and calculating gradients) of the other algorithm - SGD - is inefficient (where there are a lot of training cases). ALS is proficient at handling this type of cases [1][9]	**Complexity of implementation:** (this one is somewhat subjective) the implementation of the faster versions of ALS are more costly in terms of manpower.

Tab. 9: Advantages and disadvantages of ALS

To combat the exponential runtime cost Hu et al. in [12] describes a solution which utilizes a ridge regression and an uniform weight to all missing ratings in the dataset. This however is dependant on matrix inversion, which on itself is a costly operation relying again on optimizing p or q as a whole [12]. To combat this the team used the derivative of the objective function, which optimizes p and q in parallel but is dependent on continuous recalculation of the weights mentioned above. This last problem was solved via dynamic programming by storing the last state of the weights and reformulating the equations for computing p and q (see: Eq. 2.18 and 2.19). The whole algorithm is described in Eq. 2.17 and has a *linear* [12] runtime, traded for slightly worse accuracy and some memory cost:

Algorithm 1: ALS Learning algorithm.

Input: $\mathbf{R}$, K, λ, $\mathbf{W}$ and item confidence vector $\mathbf{c}$;
Output: Latent feature matrix $\mathbf{P}$ and $\mathbf{Q}$;

Randomly initialize $\mathbf{P}$ and $\mathbf{Q}$;
for $(u, i) \in \mathcal{R}$ **do** $\hat{r}_{ui} \leftarrow$ Eq. 2.6 ;
while *Stopping criteria is not met* **do**
    `// Update user factors`
    $\mathbf{S}^q = \sum_{i=1}^{N} c_i \mathbf{q}_i \mathbf{q}_i^T$;
    for $u \leftarrow 1$ **to** M **do**
        for $f \leftarrow 1$ **to** K **do**
            for $i \in \mathcal{R}_u$ **do** $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf} q_{if}$;
            $p_{uf} \leftarrow$ Eq. 2.18;
            for $i \in \mathcal{R}_u$ **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf} q_{if}$;
        end
    end
    `// Update item factors`
    $\mathbf{S}^p \leftarrow \mathbf{P}^T \mathbf{P}$;
    for $i \leftarrow 1$ **to** N **do**
        for $f \leftarrow 1$ **to** K **do**
            for $u \in \mathcal{R}_i$ **do** $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf} q_{if}$;
            $q_{if} \leftarrow$ Eq. 2.19;
            for $u \in \mathcal{R}_i$ **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf} q_{if}$;
        end
    end
end
return $\boldsymbol{P}$ *and* $\boldsymbol{Q}$

Equation 2.17: The linear time ALS algorithm

User estimation in Eq. 2.17 :

$$p_{uf} = \frac{\sum_{i \in \mathcal{R}_u} [w_{ui} r_{ui} - (w_{ui} - c_i) \hat{r}_{ui}^f] q_{if} - \sum_{k \neq f} p_{uk} s_{kf}^q}{\sum_{i \in \mathcal{R}_u} (w_{ui} - c_i) q_{if}^2 + s_{ff}^q + \lambda}.$$

Equation 2.18: Estimation of the user factors.

Item estimation in Eq. 2.17 :

$$q_{if} = \frac{\sum_{u \in \mathcal{R}_i} [w_{ui} r_{ui} - (w_{ui} - c_i) \hat{r}_{ui}^f] p_{uf} - c_i \sum_{k \neq f} q_{ik} s_{kf}^p}{\sum_{u \in \mathcal{R}_i} (w_{ui} - c_i) p_{uf}^2 + c_i s_{ff}^p + \lambda}$$

Equation 2.19: Estimation of the item factors.

Where c is:

$$c_i = c_0 \frac{f_i^\alpha}{\sum_{j=1}^N f_j^\alpha}$$

Equation 2.20: "popularity" estimation.

3 Datasets and Framework

This chapter introduces the datasets used in the experimental analysis, this includes basic statistics and description. Then a brief introduction of the accompanying practical implementation (framework) finishes the chapter.

3.1 Datasets

3.1.1 The Netflix Dataset x 2

The Netflix (NF) dataset from the mentioned in the previous chapters competition contains 100 480 507 ratings in the form of $r = \{id_{user}, id_{item}, r_{ui}\}$ and also timestamps when the rating was issued, these are not used in the experimental analysis and are thus not a part of the companion framework. These are however relevant in other works [36]. The ratings stem from 480 189 users and 17 770 movies and fill a total of roughly 1% of the rating matrix. The ratings are given on a 1 to 5 whole-number scale. On the Netflix platform they are called stars and higher number means that the customer has liked the movie, while the opposite means distaste. The data is provided as a standard Train-Test pair of sub-datasets. The test set contains 1 408 395 ratings and is called the Probe set, but this work utilizes the term “Test set” and it shall be called like this further on. The Test set contains newer [38] and fewer [49][23] ratings for each user than the Train and their selection process is undisclosed so that a potential model can use this as an advantage in the competition. This causes two major *issues:*

1. Ratings explicitly selected to be “hard” may benefit the commercial application (luring and keeping new customers) of a potential model that can beat the “hard” challenge, *but* for a general overview and evaluation of the performance of a particular model it may prove

misleading since it does not show the more average or general performance which can than be compared to similar datasets.

2. The unknown selection criteria make mimicking this Test dataset in other datasets challenging and may impair the objectiveness of a comparison between datasets

With these issues in mind a second, more “generic” Test set was selected from the NF dataset. This set is based on a method which selects indices (in the dataset) from a Gaussian uniform distribution and is also used for selecting the Test sets from the *other* datasets in this framework. To distinguish between them the old, original Test set will be referred as “NF-Original” and the new one “NF-Uniform”. Both of these sets are used in the experimental part of this work and since they share the same proto-set, they have the same statistics:

Datasets: NF-Original and NF-Uniform	
Movies with >= 1000 ratings: 7127	Total ratings: 100480507
Movies with >= 500 ratings: 2163	Train ratings: 98931315
Movies with >= 250 ratings: 2749	Test ratings: 1408789
Movies with >= 100 ratings: 4756	
Movies with >= 50 ratings: 918	
Movies with >= 10 ratings: 55	
Movies with no ratings: 2	

Tab. 10: Statistics of the Netflix dataset

3.1.2 The MovieLens Dataset

Scraped in 2015 this dataset is far newer than the one from the Netflix challenge which is from 2006, but is also significantly smaller at 21 622 187 ratings. These are split between 30 106 movies and were made by 234934. It is to be noted as the internet becomes more and more ubiquitous, more content is being delivered trough it, this results in a greater number of movies at 30 106 compared to NF’s 17770 albeit the dataset having roughly 20% of the total size. As with the NF dataset, MovieLens or ML for short utilizes the same 5 “star” scale, but this time with half-star increments, range: 0.5 – 5.0. Again it comes with additional data such as

timestamps and tag applications which are not utilized in the experimental part of this work. Some facts about the dataset:

Dataset: MovieLens	
Movies with >= 1000 ratings: 1974	Total ratings: 21622187
Movies with >= 500 ratings: 5738	Train ratings: 21315151
Movies with >= 250 ratings: 13223	Test ratings: 307036
Movies with >= 100 ratings: 32948	
Movies with >= 50 ratings: 33573	
Movies with >= 10 ratings: 117063	
Movies with >= 1 ratings: 30415	

Tab. 11: Statistics of the MovieLens dataset

3.1.3 The IMDB Dataset

The IMDB dataset is the second largest of the three by being slightly larger than the ML dataset with 25 947 627 ratings compared to the ~21m for the latter. The interesting here is the rating scale, which is different from the previous two and it takes the range from 1 – 10 in whole number increments. This leads to a RMSE value which is twice as large as on the other two dataset, but this is ok since by just dividing the RMSE by 2 one obtains results on the same scale as the other datasets. This is also the sparsest dataset since it tries to be the “main” movie database on the internet (and thus the world) and contains a lot of “indie” or unknown entries. Again some statistics:

Dataset: IMDB	
Movies with >= 1000 ratings: 4585	Total ratings: 25947627
Movies with >= 500 ratings: 2760	Train ratings: 25583828
Movies with >= 250 ratings: 4346	Test ratings: 363799
Movies with >= 100 ratings: 11779	
Movies with >= 50 ratings: 16121	
Movies with >= 10 ratings: 80379	

Tab. 12: Statistics of the IMDB dataset

3.2 Framework

As a part of a practical implementation for this work, the two factorization algorithms described in Chapters 2.3.4 and 2.3.5 were implemented as a means to solve the problem described in Equation 2.10 as part of the prediction model from Chapter 2.3. The inference part, however, is only a part of the functionality that the framework must be capable of in order to achieve its goal. The framework must also:

1. convert data from the different formats of the datasets to the one used internally: $r = \{id_{user}, id_{item}, r_{ui}\}$
2. generate a Test subset based on a common method for all datasets
3. be reasonably fast on loading and calculating data
4. have a small memory footprint so that event the largest datasets can fit
5. be numerically stable, since some parameters of the algorithms can get really small

With these criteria in mind, I have stopped at C++11 as a technology for developing my own factorization framework. The reasoning behind is, that it covers all of the above requirements and not at last place since I am familiar with the language. No external libraries were needed since the newer versions of the language support all the required functionalities. The used compiler was GCC[12] 7.4.0 under a 64 bit Linux environment.

4 Experimental Setup

This chapter first briefly introduces the test system (hardware part) used for factorizing the datasets. Then it follows with the definition of the Root Mean Squared Error used for measuring the performance of the predictions of the factorization model. After that the main part of this chapter begins with fine tuning the parameters of the factorization algorithms and then the results from all of the datasets. Along the results various other data

[12] https://gcc.gnu.org/

deemed worth the attention by the author is also presented. All the results from the experiments are in plot or table format with a short description and discussion.

4.1 Hardware Setup

The experimental system involved is my own personal computer which has a 4.2 GHz Octa-core processor and 16 GB RAM (8GB after a module failed during factorizing), solid state disks and improved cooling capabilities in order to handle the sometimes week long marathons of calculating results and experimenting. At its full performance the system could handle running the ALS algorithm (more resource intensive) with the NF dataset (largest) with 100 factorization parameters (dimensions) *two* times in parallel. Of course smaller datasets could be fitted more than twice simultaneously into the main memory and then the system profited from the higher core count of the processor, enabling to run up to 6 factorizations at once. Although measuring running time isn't the main objective of this chapter, they are sometimes given for the sake of clarity.

4.2 Measuring Accuracy via Root Mean Squared Error

The accuracy of the model is measured via a method called *Root Mean Squared Error* or RMSE for short:

$$\mathrm{RMSE}_{Test} = \sqrt{\sum_{(u,i)\in \mathrm{TestSet}} \left(r_{ui} - \hat{r_{ui}}\right)^2 / |TestSet|}$$

Equation 4.1: Calculation of RMSE for a Test dataset, for training it is equivalent.

This method was chosen by the organizers of the Netflix challenge and has been the de facto standard way of measuring the performance of recommender systems [large list refs]. As the name suggests the first step is to square the difference (error) between the prediction and the actual data, this is done in order to:

1. Always get a positive value for the difference and avoiding errors with opposite signs canceling each other out (we are interested if there is a difference and how big it is, not its sign).
2. Add greater emphasis to values further from the target in comparison to let's say taking the absolute value. Thus predictions with poor performance are emphasized more

The "root" at the end (not in the name) is to remove the effects of the squaring. "Mean" means average over all elements of a data set (training or test). The RMSE for a Train set is typically always (much) lower than the one for a Test set, since the Train set that is used to "train" the model is larger and more complete than the Test set which is used to test the model by calculating the RMSE of the difference between the predicted and actual values. Lower RMSE is better.

4.3 Fine-tuning of Factorization Hyperparameters

Each factorization algorithm in this work depends on one or more "global" (for the algorithm, hence the name "hyper") parameters to combat some deficiencies of the factorization method or in the data itself, see Chapters 2.3.4 and 2.3.5. Thus proper selection of the parameters is crucial for the end result. Selection can be done via a parameter sweep on the parameter space, which is simple but can be very costly with lots of parameters, since the computation time grows exponentially with each parameter or by more complex means like cross-validation.

4.3.1 Fine-tuning SGD

Stochastic Gradient Descent as the more simple of the two algorithms depends on two parameters γ and λ, without which it can not work. The first one - γ or *gamma* is the *"learning"* parameter and it represents the rate of factor adjustment opposite to the gradient of the error (hence the name of the algorithm). In other words this factor specifies the size of the step of error reduction for each factor per iteration, which can be described as "reverse" learning.

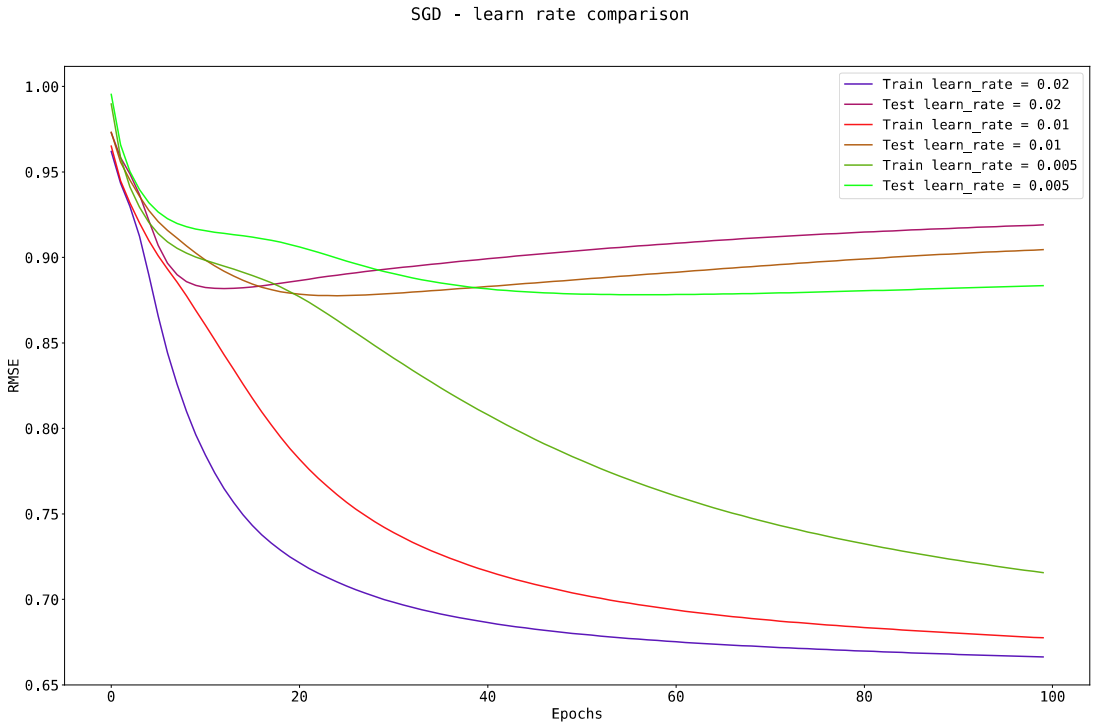

Fig. 6: Comparison of the results for various values for the learning step – ɣ. These are aggregates of all of the datasets.

Discussion: choosing a too large learning parameter (*γ)* leads to a very fast overfitting (over-learning) and possible falling into the local minimum on the Train set, accompanied first by a fast and short-lived reduction of the Test RMSE, followed by a steep increase. On the other hand the very small learning parameter seems to deliver goods results after some time, but this is misleading. The best result is achieved by the brown line (representing the Train RMSE of the "medium" *γ*) at epoch 21-22, at this point it has the same Test RMSE as the Train RMSE of the small *γ,* which indicates no convergence. One way of spotting bad values for parameters is to look at the distribution of their results:

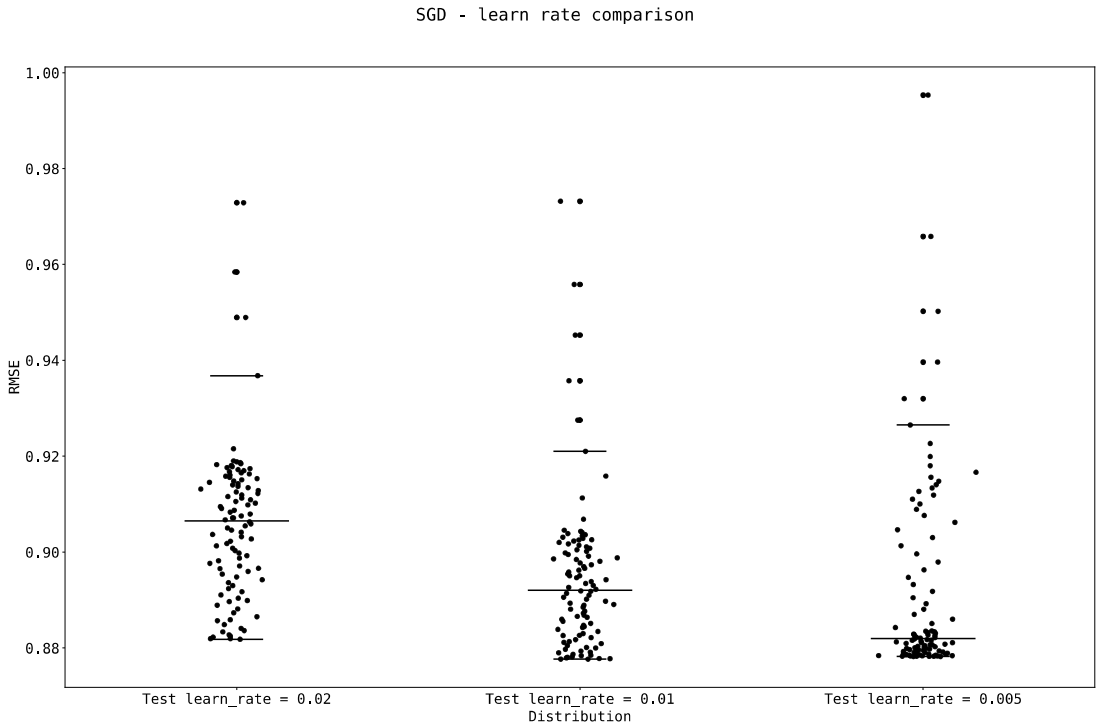

Fig. 7: Distributions of the RMSE values for the Test sets with different values for ɣ.

In this dotplot one can see the "footprint" of the RMSE errors that the algorithm leaves with different values for γ. The first one (large $\gamma = 0.02$) experiences a "hole" in the upper half (worse RMSE) of its distribution, this means that with that γ the algorithm jumped in one step from the lower end of the distribution straight into the middle. This "jumping" alone can be taken as a sign for a too high γ. After that the values are evenly distributed and spread out with slight edge for the upper half which contains the higher (worse) RMSE values, hence its bad performance. This can mean, that after "falling" into the local minimum, the algorithm stayed there since its γ was too strong. The right one ($\gamma = 0.005$) shows also a bizzare skewing of the distribution with very long "bad" RMSE tail (indicating slow learning) in the upper half. And after finally the median is reached, the algorithm continues to suffer from its slow learning pace and one can find the majority of the "good" (low RMSE - lower end) values tightly packed in a "dead-end street" achieving not much. The one in the middle is the one that I recommend to look out for, its distribution is slightly skewed towards

the lower (better) RMSE end and the majority of its values are around the median with no major holes or extreme skewing (smooth line).

The second parameter that SGD needs is the *regularization* parameter – λ. This parameter controls the intensity of regularization that is applied. Since we are working in very sparse conditions – let's say with a "higher" factor dimension of 50 per user we will have more factors that describe the ratings of single user, than the average user has ratings, thus overfitting is an inherent property of these type of datasets and one must select λ accordingly:

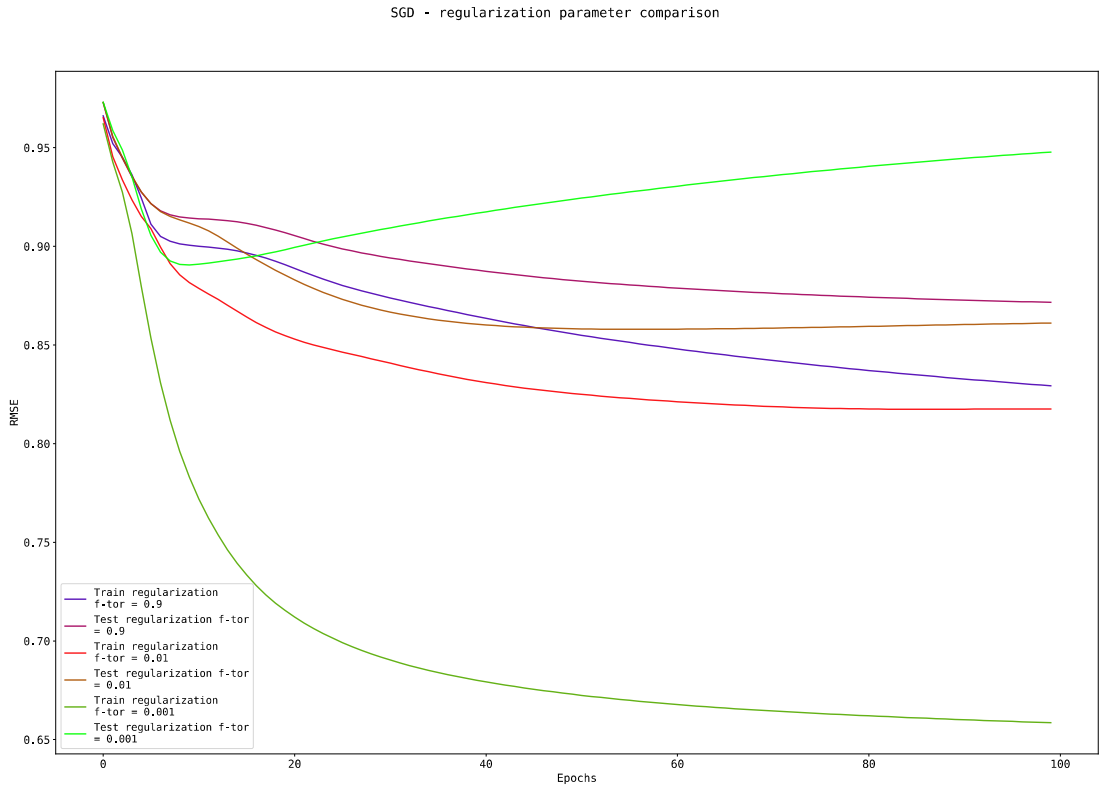

Fig. 8: Comparison of the results of various values for the regularization parameter – λ for the SGD algorithm . These are aggregates of all of the datasets.

This result is similar to the one for γ but with a larger/smaller spread between the Train and Test RMSE for the lower/higher λ values. The lines are not that edgy as in Fig. 6, which leaves one to speculate that a bad γ can be somewhat restrained by a good λ. For the low value λ case the huge discrepancy between Train and Test RMSE combined with the ultra-low score for Train RMSE one can see a classical example of overfitting. On the

other end with a high λ one can see that both Train and Test are very close together which indicates slow learning on Train, but at least this doesn't send Test in a wrong direction, also the spike at around epoch 18-20 can indicate a missed local minimum.

4.3.2 Fine-tuning ALS

ALS as the more sophisticated algorithm requires only one hyperparameter - the *regularization* parameter – λ – which plays the same role as in SGD. Also the motivation is the same since both algorithms are working in highly sparse conditions. Actually since ALS has better convergence than SGD it benefits from regularization even higher than SGD:

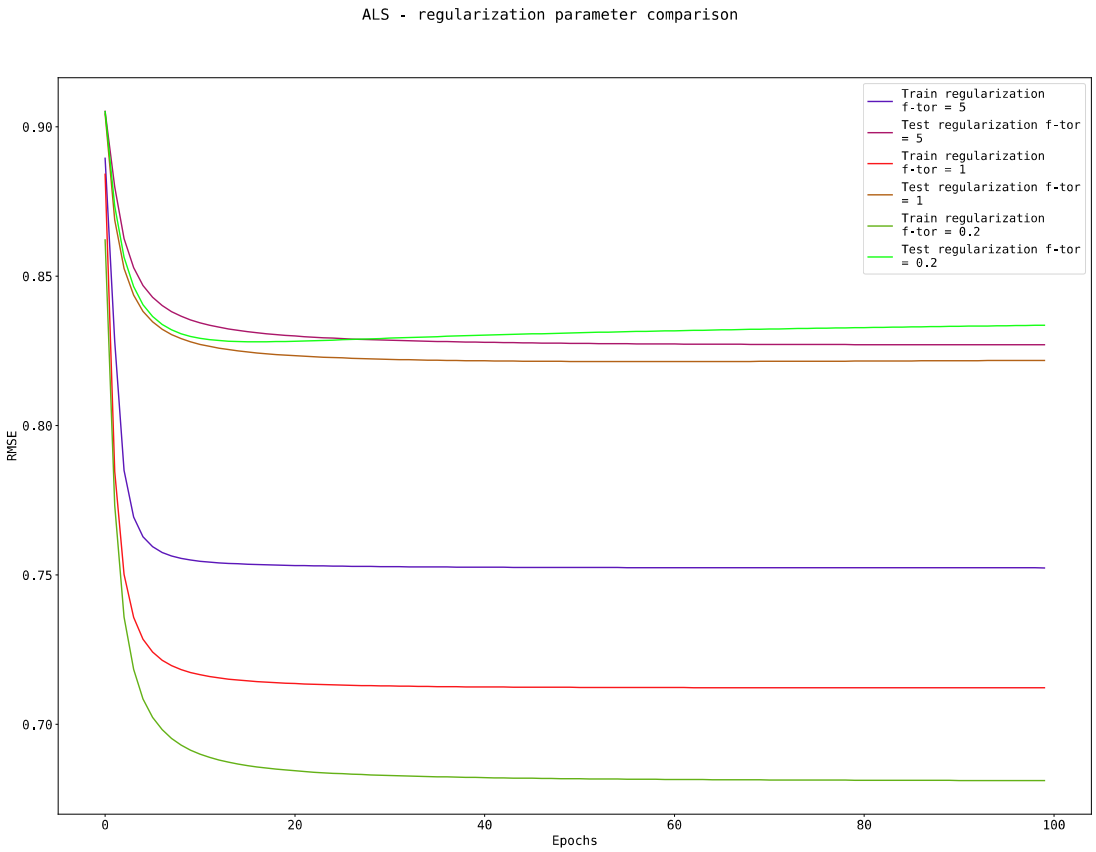

Fig. 9: Comparison of the results of various values for the regularization parameter – λ for the ALS algorithm. These are aggregates of all of the datasets.

Discussion of Fig. 9: this result is to be expected – low regularization means a high spread between Train and Test RMSE (the two green lines) – classical case for overfitting. Interesting is the good performance of the

high λ value, which can be explained with sparseness and serves as a reminder of how important regularization parameters are.

With the proper parameters selected one can proceed with the actual experimental runs where each of the datasets is factorized individually.

4.4 Experimental Approach

All datasets: Netflix-original, Netflix-uniform, MovieLens and IMDB (for more info, see Chapter 3.1) are tested under the *same* conditions with the goal to ensure a more realistic comparison of the results, these conditions include:

	SGD	ALS	Epochs	Datasets	Hyperparameters
small factorization size $f = 10$	yes	yes	100	all	fixed
medium factorization size $f = 20$	yes	yes	100	all	fixed
large factorization size $f = 50$	yes	yes	100	all	fixed

Tab. 13: Experimental modes

Epochs are fixed at 100 which is enough for convergence even for a model with a large factor space on the bigger Netflix datasets. All the algorithms hyperparameters are the *same* for all datasets and are selected by the means of *"best average"*. This is in order to ensure *equal conditions* for the combined *evaluation* of the different models and *not* to measure the best possible performance of each model. How this is achieved is shown in the previous chapter (4.3.1 and 4.3.2). Thus we get: for each of the 4 datasets we run the 2 algorithms 3 times for a total of (4*2*3) = 24 experiments. The results are represented on 8 diagrams containing the results for the 3 different factorization sizes for each algorithm and dataset.

4.5 Experimental Results

4.5.1 IMDB – SGD

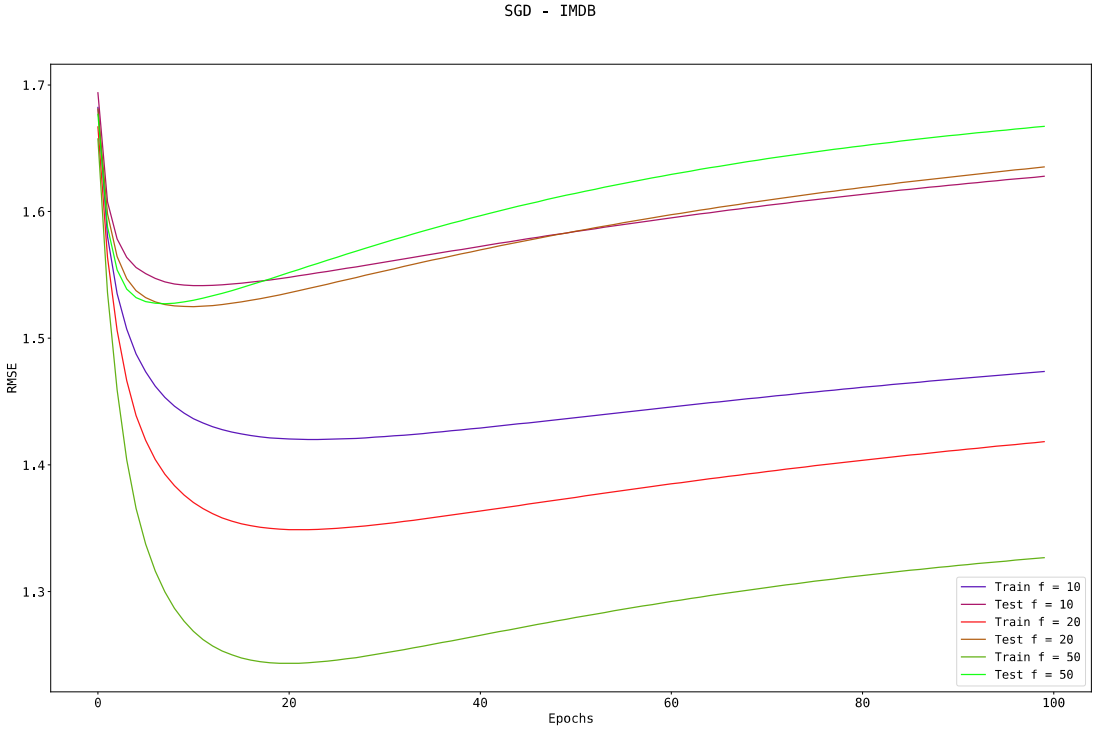

Fig. 10: SGD running on the IMDB dataset for 100 epochs.

Analysis: Due to its extreme sparsity the IMDB dataset is even more prone to overfitting, which is a possible explanation for why the model with the most parameters produced slightly worse results on the Test set, despite the good Train RMSE. A possible remedy is to increase the regularization for the model with 50 factors. Note that the RMSE here is twice as big as in the other datasets, this is so because the range of the input data is twice as large: 1-10 compared to 1-5 for the other datasets. By dividing the RMSE by 2 one can obtain a result that can be compared directly to the other datasets.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	1.42014	1.5416
medium factorization size $f = 20$	1.34883	1.52498
large factorization size $f = 50$	1.2434	1.52725

Tab. 14: Best RMSE: SGD - IMDB

4.5.2 IMDB – ALS

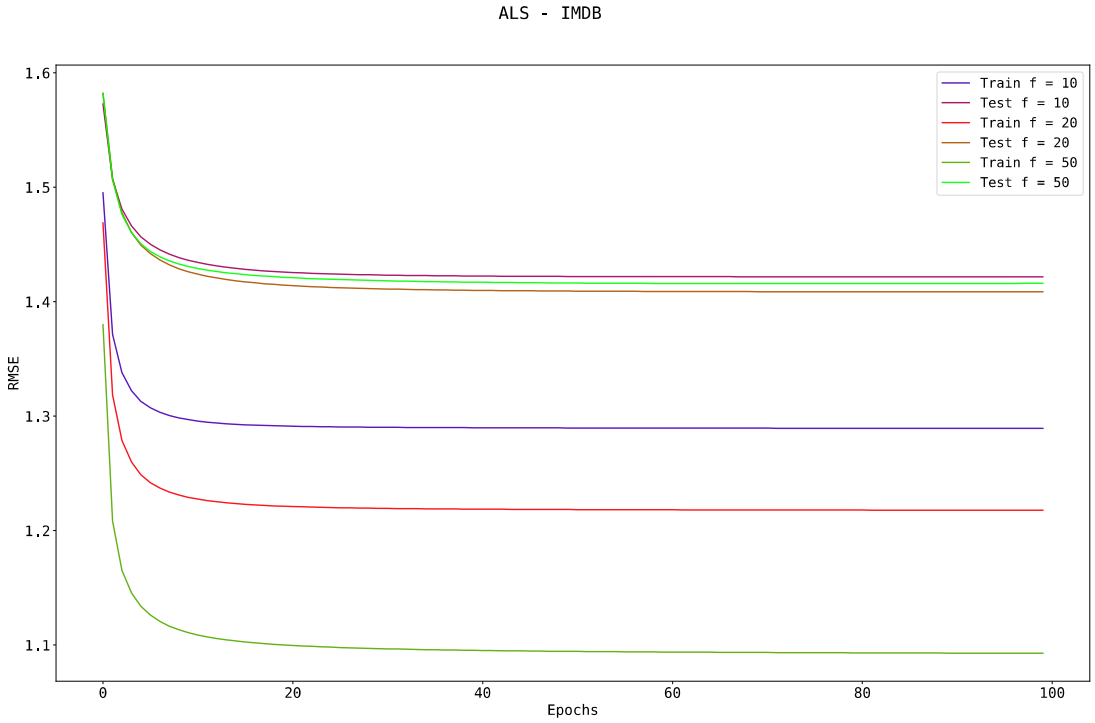

Fig. 11: ALS running on the IMDB dataset for 100 epochs.

Analysis: this time a somewhat more predictable results by the more accurate ALS algorithm. Again the best result is delivered by the medium sized factor model, but with the large model slightly behind. As with the SGD case for this dataset, a possible remedy for the under-performing large factor model is to increase the regularization for this model.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	1.28927	1.42175
medium factorization size $f = 20$	1.21773	1.40865
large factorization size $f = 50$	1.09274	1.4158

Tab. 15: Best RMSE: ALS – IMDB

4.5.3 MovieLens – SGD

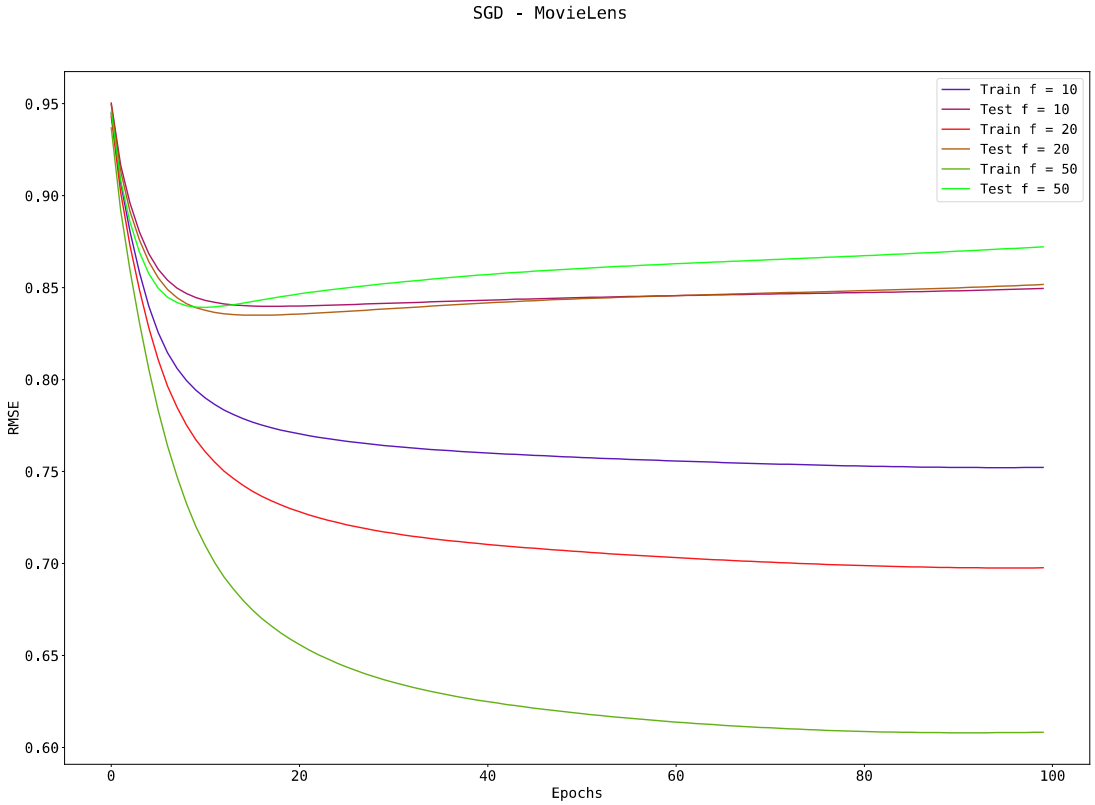

Fig. 12: SGD running on the MovieLens dataset for 100 epochs.

Analysis: here we see a continuation of the trend from the IMDB dataset, where the large model quickly reaches the local minimum, only to overfit afterwards. Also interesting is how the smallest model is being able to overtake the other two at the later stages, but this is still after its best result has been reached.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	0.752152	0.839865
medium factorization size $f = 20$	0.697598	0.835004
large factorization size $f = 50$	0.607969	0.839264

Tab. 16: Best RMSE: SGD – MovieLens

4.5.4 MovieLens – ALS

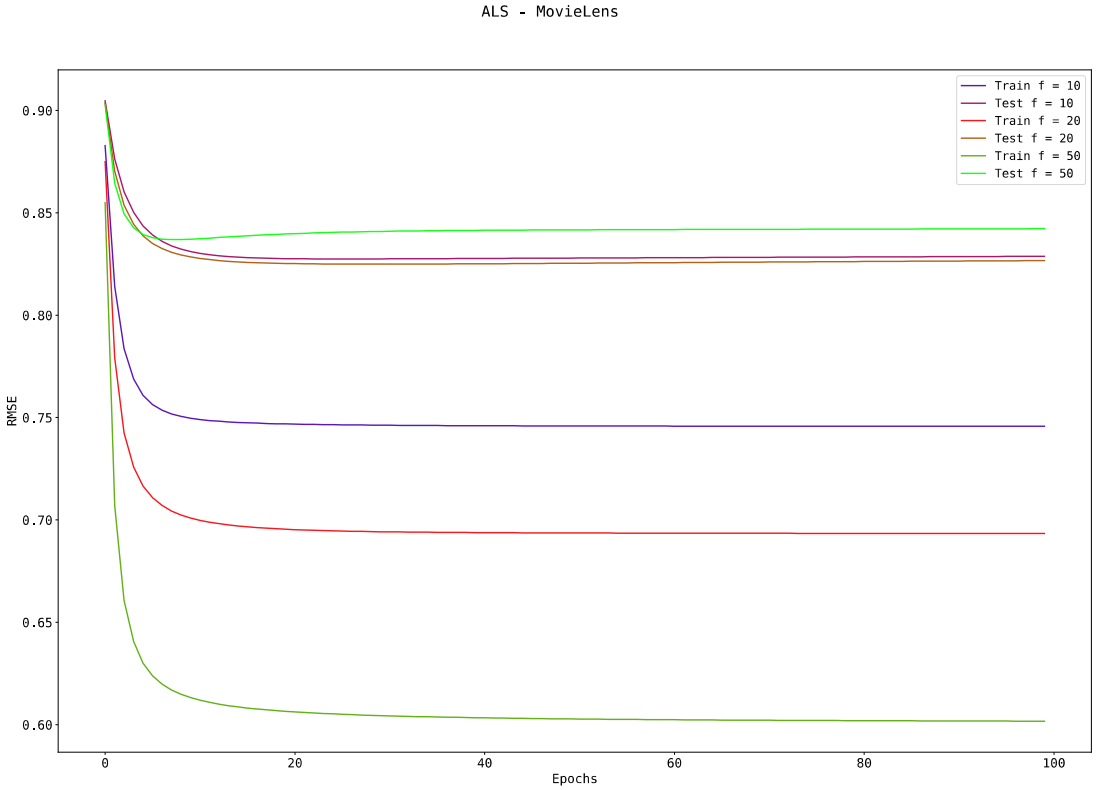

Fig. 13: ALS running on the MovieLens dataset for 100 epochs.

Analysis: Here a surprisingly poor result by the bigger model is visible, confirming the emerging trend that a bigger model is not necessarily better in sparse conditions.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	0.74576	0.827475
medium factorization size $f = 20$	0.693363	0.824946
large factorization size $f = 50$	0.601685	0.836909

Tab. 17: Best RMSE: ALS – MovieLens

4.5.5 Netflix original – SGD

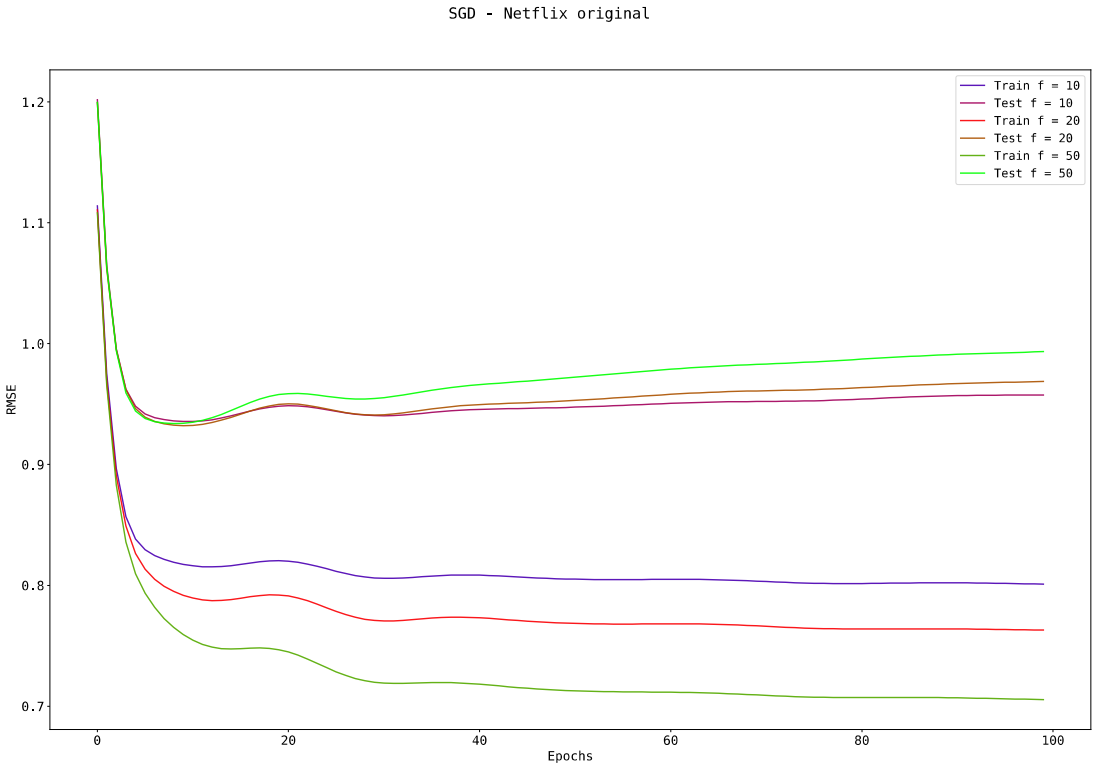

Fig. 14: SGD running on the original Netflix dataset for 100 epochs.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	0.801034	0.93562
medium factorization size $f = 20$	0.763003	0.932218
large factorization size $f = 50$	0.705507	0.933851

Tab. 18: Best RMSE: ALS – Netflix original

Analysis: The poor results on the original Netflix Test set serve as a proof of the difficult selection of Test examples in the Netflix challenge and should *not* be compared to the other results in this work. Without the additional techniques [49][50][51][2][3][5][6][7][9][36][45][23] (and many more) developed during the Netflix challenge a "generic" factorization algorithm offers poor performance against this challenging Test set (which is inspired by real-world problems that Recommender Systems face e.g. new users, users sharing accounts etc). As finishing thoughts on the analysis of the

curves and RMSE values, one can note the initial “good” looking results on the Training set only to be followed by very poor ones on the Test set, this suggests that the small amount of training examples for these (usually) users is quickly overfitted, only to return poor performance in the Test set later on. This type of missing information problem can not be just “regularized” away.

4.5.6 Netflix original – ALS

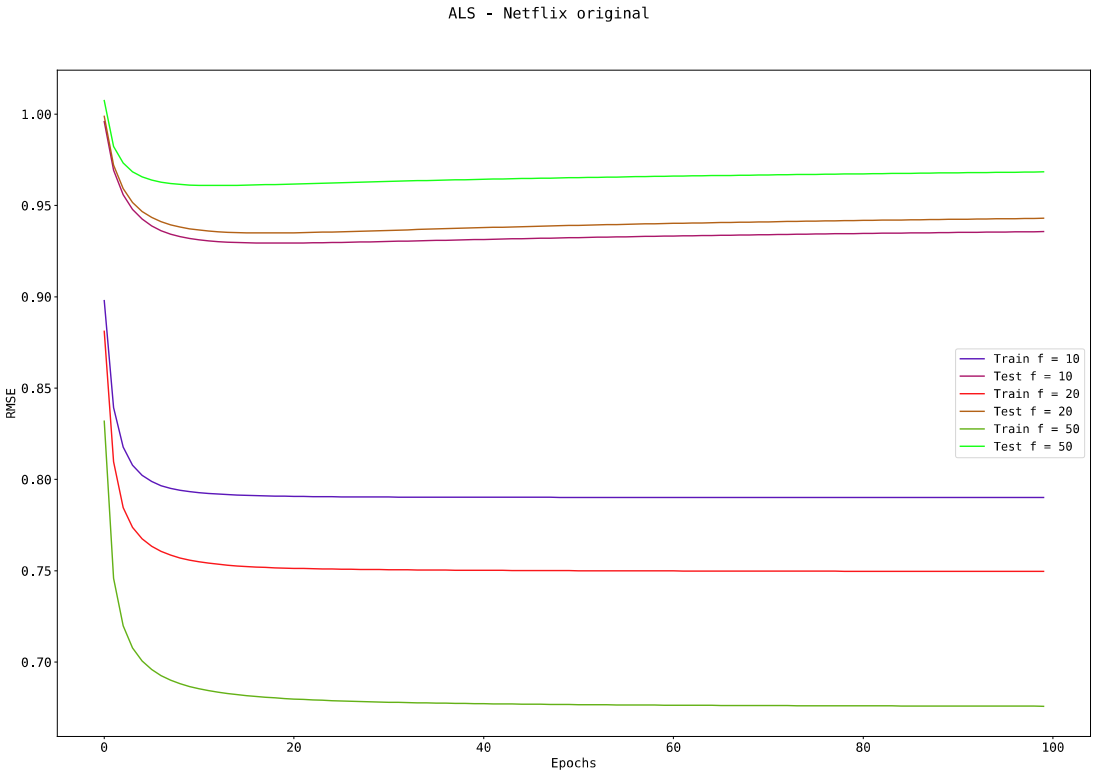

Fig. 15: ALS running on the original Netflix dataset for 100 epochs.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	0.790118	0.929385
medium factorization size $f = 20$	0.749652	0.934915
large factorization size $f = 50$	0.675827	0.960924

Tab. 19: Best RMSE: ALS – Netflix original

See: Analysis for: SGD – Netflix original in 4.5.5

4.5.7 Netflix uniform – SGD

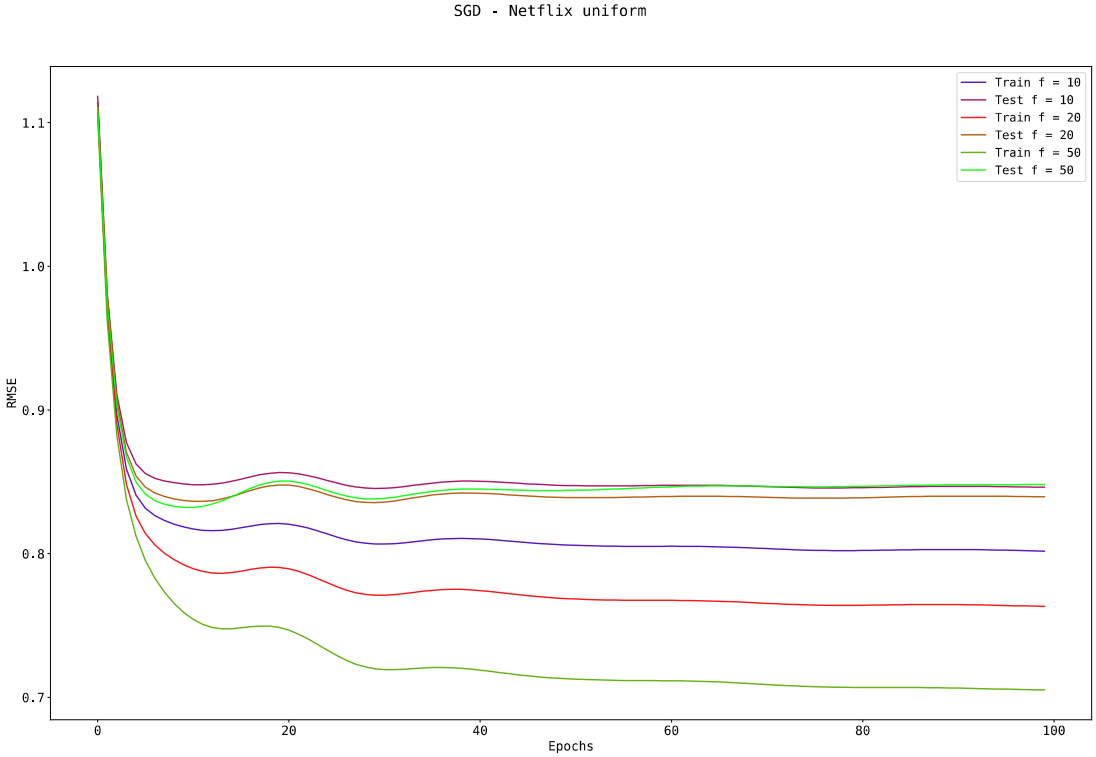

Fig. 16: SGD running on the uniform Netflix dataset for 100 epochs.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	0.801812	0.845378
medium factorization size $f = 20$	0.7633	0.835507
large factorization size $f = 50$	0.705125	0.832141

Tab. 20: Best RMSE: SGD – Netflix uniform

Analysis: a different means of Test set selection (random index generated by a uniform distribution) shows a completely different result than the one in 4.5.5. Here the evolution of the RMSE looks similar to the other uniformly selected Test sets (IMDB and MovieLens). There is however one notable difference – for the first time the large factorization model takes the lead. This is not a surprise since the Netflix dataset is much larger and instead of overfitting via the extra factors, the factorization model utilizes them accordingly.

4.5.8 Netflix uniform – ALS

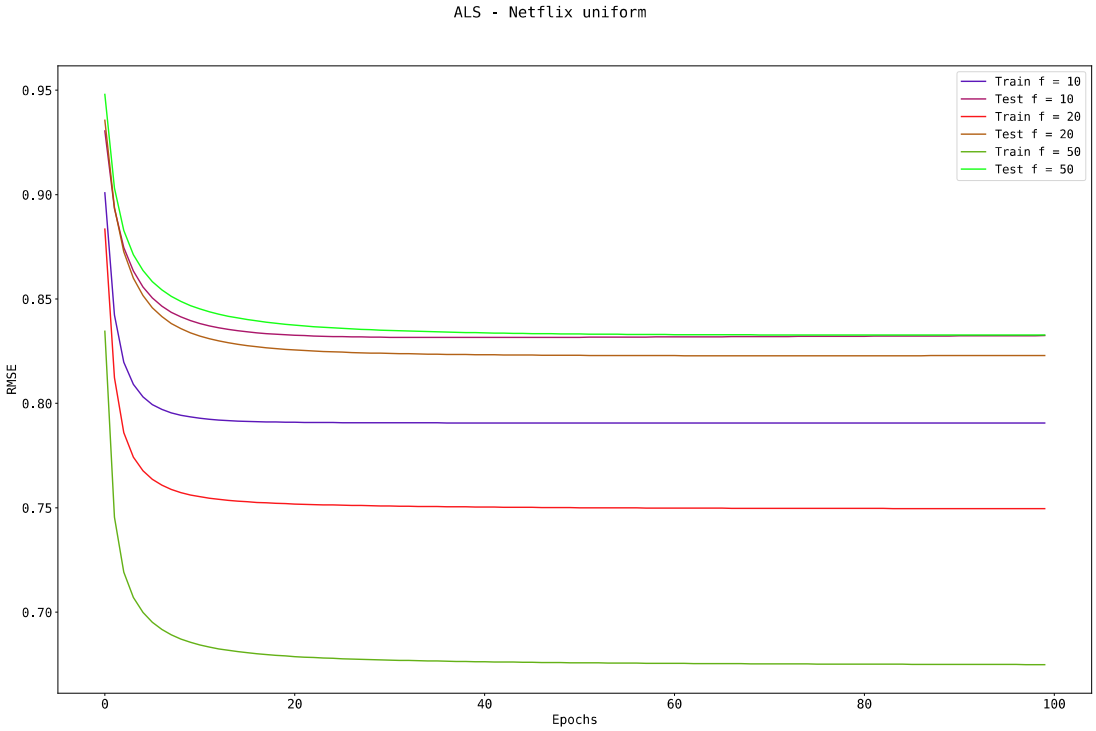

Fig. 17: ALS running on the uniform Netflix dataset for 100 epochs.

	Train best RMSE	Test best RMSE
small factorization size $f = 10$	0.790601	0.831562
medium factorization size $f = 20$	0.749601	0.822821
large factorization size $f = 50$	0.674888	0.832769

Tab. 21: Best RMSE: ALS – Netflix uniform

Analysis: Here the medium factorization model regains the first place, while the larger one struggles with overfitting. The cause for this may be the more “fine-grained” learning of ALS which is able to squeeze the most from the medium sized model by improving SGD’s 0.835507 to 0.822821. The early overfitting problem of the large model suggests that heavier regularization may be beneficial for this model only and potentially outperform the two smaller ones like in 4.5.7.

5 Discussion

In this chapter a summarization of the key findings is presented in a structured fashion. Each of these findings are then discussed and have their overall implications distilled. It is to be noted however, that the validity of the implications is localized to the datasets and algorithms (and their specific implementations) investigated in this thesis.

5.1 Summary of Key Findings

After running the algorithms on the 3 datasets used for evaluation, the most prominent findings or *insights* gained from the results of the experiments in Chapter 4.5 are:

1. Both factorization algorithms seem to favor a medium sized factor space ($f = 20$) (See: Table 22)
2. The low prediction accuracy of the large ($f = 50$) factorization model due to omnipresent overfitting issues (See: Table 22)
3. The poor performance of both algorithms on the original Netflix Test set compared to generic (uniform) Test set selected from the Netflix dataset (See: Table 22)
4. ALS slightly outperforms SGD (See: Table 22)

Table 22 presents the **best** results from all model dimensions on all datasets:

		Train f = 10	Test f = 10	Train f = 20	Test f = 20	Train f = 50	Test f = 50
SGD	IMDB	1.42014	1.5416	1.34883	1.52498	1.2434	1.52725
ALS	IMDB	1.28927	1.42175	1.21773	1.40865	1.09274	1.4158
SGD	ML	0.752152	0.839865	0.697598	0.835004	0.607969	0.839264
ALS	ML	0.74576	0.827475	0.693363	0.824946	0.601685	0.836909
SGD	NFO	0.801034	0.93562	0.763003	0.932218	0.705507	0.933851
ALS	NFO	0.790118	0.929385	0.749652	0.934915	0.675827	0.960924
SGD	NFU	0.801812	0.845378	0.7633	0.835507	0.705125	0.832141
ALS	NFU	0.790601	0.831562	0.749601	0.822821	0.674888	0.832769

Tab. 22: Best RMSE of all runs

5.2 Discussion of the Key Findings

5.2.1 Factor Dimensionality and Regularization

Findings (1) and (2) seem to be connected. For (2) the fault is clearly a too week regularization parameter, this can be most prominently seen on the IMDB dataset due to its extreme sparsity. Since the large model has way more factors than the other two models (2.5 and 5 times respectively) it has more “space” to “memorize” the learned data. So selecting a “best average” regularization parameter for all 3 models has proven discriminatory for the large model since it requires heavier regularization, especially on smaller datasets. By selecting a “best average” parameter the two smaller ones are “pulling” (2 against 1) towards a lower factor in the parameter estimation calculation. This fits them well since they are not that far apart from each other (in terms of factor dimensionality) compared to the big model. Remedy for this problem is to take the best *individual* parameters for each model and remove the hyperparameters from the “equal conditions” for comparison.

5.2.2 Difficulty of the Netflix Test dataset

Finding (3) stems from the fact that this is the dataset that caused (not taking the monetary prize of the Netflix challenge into account) the development of a large number (see: Bibliography) of all kinds of prediction models and factorization algorithms. Some of these specialize at finding additional sources of information [36][9][2][5][26]. Selection of such sources is described in chapters 2.3.2 “Implicit input sources” and 2.3.3 “Additional explicit input sources”. Others use ensembles of models and algorithms to achieve a lower RMSE [49][50][51]. Neither of these strategies are however a part of the framework and thus the algorithms can not take advantage of their benefits. The general performance, however is comparable to other similar implementations before the addition of such improvements [2][5][7]. The root cause for (3) is the nature of the Netflix Test/Validation sets which include “tough” examples of users/items with very few ratings (new users/items) or contradictory patterns (user account shared

by two or more persons with different taste). More on the motivation about this in Chapter 6. Hence the implication of (3): the results from the Netflix original – SGD and Netflix original – ALS datasets are not to be compared to the other results of this work, they serve to compare how the practical implementation of this work stacks against other implementations (since there are lots of results for the original Netflix dataset from various algorithms). As a finishing thought about (3) I want to add a quote from Yehuda Koren, co-winner of the Netflix prize from his “The BellKor Solution to the Netflix Grand Prize” paper [49]:

> "Another helpful factor was some touch of luck. The most prominent one is the choice of the 10% improvement goal. Any small deviation from this number, would have made the competition either too easy or impossibly difficult.."

5.2.3 ALS slightly outperforms SGD

Finding (4): As Table 22 shows, Alternating Least Squares (ALS) emerges as the slightly better performing algorithm in terms of prediction accuracy. This comes at the cost of slightly increased memory footprint and slower run times. Note, that the implemented version runs in linear time, as opposed to the much more time consuming basic version of the of the ALS algorithm, which runs in exponential time (f^3).

5.3 Best Result

As to conclude the discussion, the dataset with lowest RMSE achieved on its Test set is: **IMDB**. With a model factorization dimension of: $f = 20$ and inferred via the **ALS** algorithm. It achieves a RMSE of 1.40865 which scaled down to the range of the other datasets (1-5) (from native 1-10) results to: **0,704325.**

6 Conclusion

Two algorithms for inferring the factors of a general factorization model for a Recommender System were introduced, implemented and analyzed and had their results *compared* with the goal of answering a set of Research Questions which were set in Chapter 1.3 and are listed and answered bellow.

The core concept for the comparison was to find a common ground of *"equal conditions"* for measuring the overall *competence* of a factorization model instead of concentrating on boosting single performance criteria, like: small improvements of the prediction accuracy which more often than not comes at a great cost, like: huge factor dimension or hyper-parameterization (fine-tuning) of the algorithms for a specific task or dataset. To achieve this, both factorization algorithms were hyper-parameterized to their *"best average"* (Chapter 4.4) values and were let to always run the same amount of iterations (epochs) - 100. The only parameter tested in multiple configurations was the dimension of the factorization, or in other words the number of factors used for describing a particular user or item. Since this parameter directly affects both the run time and memory consumption, which on their hand dictate the real world feasibility and quality (responsiveness/accuracy) of a factorization model for a Recommender System. Thus in this work a model that achieves a second best prediction accuracy (slightly behind the best) is deemed the *better* one, if in doing so it needs considerably *less* factors than the one with the lowest prediction error. The values chosen for the factorization dimension or *"f"* were 10, 20 and 50 representing a *small*, *medium* and *large* models respectively. As it was found out and discussed in Chapter 5.2.1 choosing a *"one-size-fits-all"* parameters can hinder the performance of the algorithms, but it does *not* negate their universal applicability for similar tasks if they are initialized with parameters best fitting for the task or particular dataset. This directly answers Research Question 1 from Tab. 1: Research Questions and Contributions:

- Is general (works on all datasets) implementation derivable?

Even with proper parameterization a “generic” factorization model (pure algorithm only, without additional input sources or ensembling [35] of algorithms) can underperform on real-world inspired Test datasets. This is true even if it performs well on generic selections from the same parent dataset, see Chapter 5.2.2. Netflix’ motivators for choosing such a dataset come from a commercial perspective:

1. offer good recommendations to new users which are in the “critical” phase – having interest in the service, but not yet loyal customers
2. better handling of “difficult” users (“contradictory rating patterns”)
3. general improvement of prediction quality under sparse conditions

Commercial success is one of the primary quality measures (and driver for research in the field) for a Recommender System and the motivators listed above are not only domain to Netflix. New users for example represent the *cold-start* problem which is a problem in almost all Latent Semantic Indexing approaches. Unfortunately however Netflix did not provide the *exact* selection criteria for their Test dataset, since it could have been exploited in the Netflix challenge. This led to the “generic” selection method for generating Test datasets utilized in this work. With that in mind the practical implementation of this work proves to have *similar* performance compared to the “generic” parts of other implementations i.e. *before* their improvement by various means like ensembling [35] or additional input sources [1]. This leads to the answer of Research Question 2 from Tab. 1: Research Questions and Contributions:

- How does the provided practical implementation stack against previous work?

As finishing thoughts one can note, that both algorithms achieve *good* results on the tested datasets and they further achieve this without the need of large number of iterations (epochs) or parameters (since even the small dimensional factorization model has a reasonably good performance in all of the results). To sum up: for a good recommendation from a

typical dataset one doesn't need an overly complex model, thousands of iterations or specialized hardware (all experiment runs were done on a desktop PC). This answers Research Question 3 and 4 from Tab. 1: Research Questions and Contributions:

- Is an ensemble of factorization algorithms a necessity for achieving high prediction accuracy?
- How does the selection of factorization algorithms perform on normal desktop hardware, and how do they fare against each other?

A summary of research questions and their corresponding contributions can be found in Table 1.

6.1 Future Work

Multiple promising aspects were left out of scope by this work, but the most immediate ones include:

- Adding additional implicit and explicit input sources to the existing model and performing analysis of the potential benefits (Chapters 2.3.2 and 2.3.3)
- Adding adaptive (hyper-)parameters for real-time adjustments of learning rates and regularization rates like: adaptive learning rate which has a fixed "penalty" coefficient of e.g. 0.9 after each step.
- Removing the need of hyper-parameters altogether by using advanced methods like Markov chain Monte Carlo [29]

7 Appendices

Appendix I: Example run of Stochastic Gradient Descent

What follows is a 10 iteration run of the algorithm defined in chapter 2.3.4 with no global factors or implicit/explicit data. Learn rate is 0.01, regularization parameter is 0.1. Here the dataset, the last value of each entry is the rating that the user gave for this item:

Items = {The Notebook (NB), Don Jon (DJ), Rush Hour (RH), Predator (PR)};
Users = {Alice (A), Bob (B), Carl (C)};
Ratings = {(A, NB, 5), (A, DJ, 3), (A, RH, 1), (B, RH, 4), (B, PR, 5), (C, NB, 1), (C, DJ, 3), (C, RH, 5)} = *Training set;*
Test set = {(A, DJ, 3), (B, RH, 4), (C, DJ, 3)};

Bellow is a listing of all relevant variables, per iteration, with floating point precision – 5 digits behind the decimal mark. Since for simplicity both the user's and the item's feature vector is both mapped to the same factor matrix F, here are the indices of the users and the items:

0	1	2	3	4	5	6
Alice	The Notebook	Don Jon	Rush Hour	Bob	Predator	Carl

Factors:	Gaussian, initialization: sigma = 0.1							
F[0]:	-0.02157	0.06538	-0.05692	0.13727	0.03333	0.04631	0.0063	
F[1]:	0.05864	0.1296	-0.10846	0.11342	-0.05712	0.00654	0.16433	
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.02157	0.06538	0.05864	0.1296	0.00619	1	4
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u, previous}$	$F_{u, new}$	
$F_u[0][0]$	0.01	0.1	-0.06538	0.26153	0.00216	-0.02157	-0.01893	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i, previous}$	$F_{i, new}$	
$F_i[0][1]$	0.01	0.1	0.02157	-0.08626	-0.00654	0.06538	0.06445	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u, previous}$	$F_{u, new}$	
$F_u[1][0]$	0.01	0.1	-0.1296	0.5184	-0.00586	0.05864	0.06377	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i, previous}$	$F_{i, new}$	
$F_i[1][1]$	0.01	0.1	-0.05864	0.23457	-0.01296	0.1296	0.13182	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01893	-0.05692	0.06377	-0.10846	0.07416	1	2
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u, previous}$	$F_{u, new}$	
$F_u[0][0]$	0.01	0.1	0.05692	-0.11383	0.00189	-0.01893	-0.02005	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i, previous}$	$F_{i, new}$	

$F_i[0][2]$	0.01	0.1	0.01893	-0.03786	0.00569	-0.05692	-0.05724	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.10846	-0.21693	-0.00638	0.06377	0.06153	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.06377	0.12753	0.01085	-0.10846	-0.10708	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.02005	0.13727	0.06153	0.11342	0.12419	1	0
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.13727	0	0.002	-0.02005	-0.02003	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.02005	0	-0.01373	0.13727	0.13714	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.11342	0	-0.00615	0.06153	0.06147	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.06153	0	-0.01134	0.11342	0.11331	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.03333	0.13714	-0.05712	0.11331	0.0581	1	3
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.13714	0.41141	-0.00333	0.03333	0.03741	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.03333	0.1	-0.01371	0.13714	0.138	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.11331	0.33993	0.00571	-0.05712	-0.05366	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.05712	-0.17135	-0.01133	0.11331	0.11148	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.03741	0.04631	-0.05366	0.00654	0.12138	1	4
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.04631	0.18525	-0.00374	0.03741	0.03923	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.03741	0.14966	-0.00463	0.04631	0.04776	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.00654	0.02614	0.00537	-0.05366	-0.05335	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.05366	-0.21465	-0.00065	0.00654	0.00438	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.0063	0.06445	0.16433	0.13182	0.19207	1	0
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.06445	0	-0.00063	0.0063	0.0063	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.0063	0	-0.00645	0.06445	0.06439	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.13182	0	-0.01643	0.16433	0.16416	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.16433	0	-0.01318	0.13182	0.13168	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.0063	-0.05724	0.16416	-0.10708	0.13206	1	2
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05724	-0.11448	-0.00063	0.0063	0.00515	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.0063	0.01259	0.00572	-0.05724	-0.05706	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.10708	-0.21416	-0.01642	0.16416	0.16186	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[1][2]$	0.01	0.1	-0.16416	0.32833	0.01071	-0.10708	-0.10369	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.00515	0.138	0.16186	0.11148	0.21875	1	4
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.138	0.552	-0.00051	0.00515	0.01066	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.00515	0.02058	-0.0138	0.138	0.13807	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.11148	0.44593	-0.01619	0.16186	0.16615	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.16186	0.64743	-0.01115	0.11148	0.11785	
Factors:								
F[0]:	-0.02003	0.06439	-0.05706	0.13807	0.03923	0.04776	0.01066	
F[1]:	0.06147	0.13168	-0.10369	0.11785	-0.05335	0.00438	0.16615	
Epoch= 0	Train RMSE=	2.85044	Test RMSE=	2.38048				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.02003	0.06439	0.06147	0.13168	0.29667	1	4
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.06439	0.25756	0.002	-0.02003	-0.01743	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.02003	-0.08011	-0.00644	0.06439	0.06352	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.13168	0.52673	-0.00615	0.06147	0.06668	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.06147	0.24589	-0.01317	0.13168	0.13401	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01743	-0.05706	0.06668	-0.10369	0.3639	1	2
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05706	-0.11411	0.00174	-0.01743	-0.01856	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	0.01743	-0.03487	0.00571	-0.05706	-0.05735	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.10369	-0.20738	-0.00667	0.06668	0.06454	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.06668	0.13336	0.01037	-0.10369	-0.10225	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.01856	0.13807	0.06454	0.11785	0.44475	1	0
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.13807	0	0.00186	-0.01856	-0.01854	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.01856	0	-0.01381	0.13807	0.13793	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.11785	0	-0.00645	0.06454	0.06447	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.06454	0	-0.01178	0.11785	0.11773	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.03923	0.13793	-0.05335	0.11773	0.389	1	3
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.13793	0.41379	-0.00392	0.03923	0.04333	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.03923	0.11769	-0.01379	0.13793	0.13897	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.11773	0.35318	0.00533	-0.05335	-0.04976	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[1][3]$	0.01	0.1	0.05335	-0.16004	-0.01177	0.11773	0.11601	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.04333	0.04776	-0.04976	0.00438	0.42175	1	4
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.04776	0.19105	-0.00433	0.04333	0.04519	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.04333	0.17331	-0.00478	0.04776	0.04945	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.00438	0.01753	0.00498	-0.04976	-0.04954	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.04976	-0.19905	-0.00044	0.00438	0.00239	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.01066	0.06352	0.16615	0.13401	0.48284	1	0
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.06352	0	-0.00107	0.01066	0.01065	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.01066	0	-0.00635	0.06352	0.06346	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.13401	0	-0.01662	0.16615	0.16599	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.16615	0	-0.0134	0.13401	0.13388	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.01065	-0.05735	0.16599	-0.10225	0.42228	1	2
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05735	-0.11469	-0.00106	0.01065	0.00949	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.01065	0.0213	0.00573	-0.05735	-0.05708	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.10225	-0.2045	-0.0166	0.16599	0.16378	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.16599	0.33198	0.01023	-0.10225	-0.09883	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.00949	0.13897	0.16378	0.11601	0.54001	1	4
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.13897	0.55587	-0.00095	0.00949	0.01504	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.00949	0.03797	-0.0139	0.13897	0.13921	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.11601	0.46404	-0.01638	0.16378	0.16825	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.16378	0.65511	-0.0116	0.11601	0.12244	
Factors:								
F[0]:	-0.01854	0.06346	-0.05708	0.13921	0.04519	0.04945	0.01504	
F[1]:	0.06447	0.13388	-0.09883	0.12244	-0.04954	0.00239	0.16825	
Epoch= 1	Train RMSE=	2.85044	Test RMSE=	2.38048				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.01854	0.06346	0.06447	0.13388	0.58692	1	4
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.06346	0.25384	0.00185	-0.01854	-0.01598	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.01854	-0.07415	-0.00635	0.06346	0.06266	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.13388	0.53551	-0.00645	0.06447	0.06976	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[1][1]$	0.01	0.1	-0.06447	0.25789	-0.01339	0.13388	0.13632	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01598	-0.05708	0.06976	-0.09883	0.6534	1	2
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05708	-0.11415	0.0016	-0.01598	-0.01711	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	0.01598	-0.03196	0.00571	-0.05708	-0.05734	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.09883	-0.19766	-0.00698	0.06976	0.06772	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.06976	0.13953	0.00988	-0.09883	-0.09734	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.01711	0.13921	0.06772	0.12244	0.76498	1	0
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.13921	0	0.00171	-0.01711	-0.01709	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.01711	0	-0.01392	0.13921	0.13907	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.12244	0	-0.00677	0.06772	0.06765	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.06772	0	-0.01224	0.12244	0.12232	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.04519	0.13907	-0.04954	0.12232	0.71962	1	3
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.13907	0.41721	-0.00452	0.04519	0.04932	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.04519	0.13558	-0.01391	0.13907	0.14029	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.12232	0.36697	0.00495	-0.04954	-0.04582	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.04954	-0.14861	-0.01223	0.12232	0.12071	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.04932	0.04945	-0.04582	0.00239	0.72195	1	4
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.04945	0.19779	-0.00493	0.04932	0.05125	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.04932	0.19729	-0.00494	0.04945	0.05137	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.00239	0.00955	0.00458	-0.04582	-0.04568	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.04582	-0.18327	-0.00024	0.00239	0.00055	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.01504	0.06266	0.16825	0.13632	0.77342	1	0
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.06266	0	-0.0015	0.01504	0.01503	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.01504	0	-0.00627	0.06266	0.06259	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.13632	0	-0.01683	0.16825	0.16809	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.16825	0	-0.01363	0.13632	0.13619	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.01503	-0.05734	0.16809	-0.09734	0.71224	1	2
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	

$F_u[0][6]$	0.01	0.1	0.05734	-0.11468	-0.0015	0.01503	0.01386	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.01503	0.03005	0.00573	-0.05734	-0.05698	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.09734	-0.19467	-0.01681	0.16809	0.16597	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.16809	0.33617	0.00973	-0.09734	-0.09388	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.01386	0.14029	0.16597	0.12071	0.86097	1	4
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.14029	0.56115	-0.00139	0.01386	0.01946	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.01386	0.05546	-0.01403	0.14029	0.1407	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.12071	0.48285	-0.0166	0.16597	0.17063	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.16597	0.66388	-0.01207	0.12071	0.12723	
Factors:								
F[0]:	-0.01709	0.06259	-0.05698	0.1407	0.05125	0.05137	0.01946	
F[1]:	0.06765	0.13619	-0.09388	0.12723	-0.04568	0.00055	0.17063	
Epoch= 2	Train RMSE=	2.85044	Test RMSE=	2.38048				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.01709	0.06259	0.06765	0.13619	0.87695	1	4
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.06259	0.25037	0.00171	-0.01709	-0.01457	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.01709	-0.06836	-0.00626	0.06259	0.06185	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.13619	0.54474	-0.00677	0.06765	0.07303	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.06765	0.2706	-0.01362	0.13619	0.13876	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01457	-0.05698	0.07303	-0.09388	0.94266	1	2
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05698	-0.11396	0.00146	-0.01457	-0.01569	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	0.01457	-0.02914	0.0057	-0.05698	-0.05722	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.09388	-0.18775	-0.0073	0.07303	0.07108	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.07303	0.14606	0.00939	-0.09388	-0.09232	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.01569	0.1407	0.07108	0.12723	1.08488	1.08488	-0.08488
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.1407	-0.01194	0.00157	-0.01569	-0.0158	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.01569	0.00133	-0.01407	0.1407	0.14057	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.12723	-0.0108	-0.00711	0.07108	0.0709	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.07108	-0.00603	-0.01272	0.12723	0.12704	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.05125	0.14057	-0.04568	0.12704	1.04827	1.04827	2.95173
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	

$F_u[0][4]$	0.01	0.1	-0.14057	0.41494	-0.00513	0.05125	0.05535	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.05125	0.15128	-0.01406	0.14057	0.14195	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.12704	0.375	0.00457	-0.04568	-0.04188	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.04568	-0.13482	-0.0127	0.12704	0.12557	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.05535	0.05137	-0.04188	0.00055	1.02017	1.02017	3.97983
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.05137	0.20445	-0.00553	0.05535	0.05734	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.05535	0.22028	-0.00514	0.05137	0.05352	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.00055	0.0022	0.00419	-0.04188	-0.04182	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.04188	-0.16668	-6E-05	0.00055	-0.00111	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.01946	0.06185	0.17063	0.13876	1.06227	1.06227	-0.06227
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.06185	-0.00385	-0.00195	0.01946	0.0194	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.01946	-0.00121	-0.00618	0.06185	0.06177	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.13876	-0.00864	-0.01706	0.17063	0.17038	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.17063	-0.01063	-0.01388	0.13876	0.13851	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.0194	-0.05722	0.17038	-0.09232	0.99918	1	2
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05722	-0.11443	-0.00194	0.0194	0.01824	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.0194	0.03881	0.00572	-0.05722	-0.05677	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.09232	-0.18465	-0.01704	0.17038	0.16836	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.17038	0.34075	0.00923	-0.09232	-0.08882	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.01824	0.14195	0.16836	0.12557	1.17753	1.17753	3.82247
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.14195	0.54258	-0.00182	0.01824	0.02365	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.01824	0.06972	-0.01419	0.14195	0.1425	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.12557	0.47998	-0.01684	0.16836	0.17299	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.16836	0.64355	-0.01256	0.12557	0.13188	
Factors:								
F[0]:	-0.0158	0.06177	-0.05677	0.1425	0.05734	0.05352	0.02365	
F[1]:	0.0709	0.13851	-0.08882	0.13188	-0.04182	-0.00111	0.17299	
Epoch= 3	Train RMSE=	2.6732	Test RMSE=	2.17026				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.0158	0.06177	0.0709	0.13851	1.16133	1.16133	3.83867
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	

$F_u[0][0]$	0.01	0.1	-0.06177	0.23712	0.00158	-0.0158	-0.01341	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.0158	-0.06064	-0.00618	0.06177	0.0611	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.13851	0.5317	-0.00709	0.0709	0.07615	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.0709	0.27216	-0.01385	0.13851	0.14109	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01341	-0.05677	0.07615	-0.08882	1.22372	1.22372	1.77628
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05677	-0.10084	0.00134	-0.01341	-0.01441	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	0.01341	-0.02382	0.00568	-0.05677	-0.05695	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.08882	-0.15777	-0.00761	0.07615	0.07449	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.07615	0.13526	0.00888	-0.08882	-0.08738	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.01441	0.1425	0.07449	0.13188	1.38882	1.38882	-0.38882
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.1425	-0.05541	0.00144	-0.01441	-0.01495	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.01441	0.0056	-0.01425	0.1425	0.14241	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.13188	-0.05128	-0.00745	0.07449	0.07391	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.07449	-0.02896	-0.01319	0.13188	0.13146	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.05734	0.14241	-0.04182	0.13146	1.36071	1.36071	2.63929
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.14241	0.37587	-0.00573	0.05734	0.06104	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.05734	0.15133	-0.01424	0.14241	0.14379	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.13146	0.34695	0.00418	-0.04182	-0.0383	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.04182	-0.11037	-0.01315	0.13146	0.13022	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.06104	0.05352	-0.0383	-0.00111	1.30206	1.30206	3.69794
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.05352	0.19792	-0.0061	0.06104	0.06296	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.06104	0.22572	-0.00535	0.05352	0.05573	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	0.00111	-0.00412	0.00383	-0.0383	-0.03831	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.0383	-0.14165	0.00011	-0.00111	-0.00253	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.02365	0.0611	0.17299	0.14109	1.33097	1.33097	-0.33097
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.0611	-0.02022	-0.00236	0.02365	0.02342	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.02365	-0.00783	-0.00611	0.0611	0.06096	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	

$F_u[1][6]$	0.01	0.1	-0.14109	-0.0467	-0.0173	0.17299	0.17235	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.17299	-0.05725	-0.01411	0.14109	0.14038	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.02342	-0.05695	0.17235	-0.08738	1.26194	1.26194	1.73806
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05695	-0.09899	-0.00234	0.02342	0.02241	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.02342	0.04071	0.0057	-0.05695	-0.05649	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.08738	-0.15187	-0.01724	0.17235	0.17066	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.17235	0.29956	0.00874	-0.08738	-0.0843	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.02241	0.14379	0.17066	0.13022	1.45874	1.45874	3.54126
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.14379	0.50918	-0.00224	0.02241	0.02748	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.02241	0.07935	-0.01438	0.14379	0.14444	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.13022	0.46115	-0.01707	0.17066	0.1751	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.17066	0.60435	-0.01302	0.13022	0.13614	
Factors:								
F[0]:	-0.01495	0.06096	-0.05649	0.14444	0.06296	0.05573	0.02748	
F[1]:	0.07391	0.14038	-0.0843	0.13614	-0.03831	-0.00253	0.1751	
Epoch= 4	Train RMSE=	2.46451	Test RMSE=	1.90952				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.01495	0.06096	0.07391	0.14038	1.41324	1.41324	3.58676
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.06096	0.21867	0.00149	-0.01495	-0.01274	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.01495	-0.05361	-0.0061	0.06096	0.06037	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.14038	0.50351	-0.00739	0.07391	0.07887	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.07391	0.26508	-0.01404	0.14038	0.14289	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01274	-0.05649	0.07887	-0.0843	1.47008	1.47008	1.52992
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05649	-0.08642	0.00127	-0.01274	-0.0136	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	0.01274	-0.0195	0.00565	-0.05649	-0.05663	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.0843	-0.12897	-0.00789	0.07887	0.0775	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.07887	0.12066	0.00843	-0.0843	-0.08301	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.0136	0.14444	0.0775	0.13614	1.65322	1.65322	-0.65322
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.14444	-0.09435	0.00136	-0.0136	-0.01453	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.0136	0.00888	-0.01444	0.14444	0.14438	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	

$F_u[1][0]$	0.01	0.1	-0.13614	-0.08893	-0.00775	0.0775	0.07653	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.0775	-0.05062	-0.01361	0.13614	0.13549	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.06296	0.14438	-0.03831	0.13549	1.6366	1.6366	2.3634
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.14438	0.34123	-0.0063	0.06296	0.06631	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.06296	0.14879	-0.01444	0.14438	0.14572	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.13549	0.32022	0.00383	-0.03831	-0.03507	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.03831	-0.09054	-0.01355	0.13549	0.13445	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.06631	0.05573	-0.03507	-0.00253	1.55406	1.55406	3.44594
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.05573	0.19203	-0.00663	0.06631	0.06816	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.06631	0.22849	-0.00557	0.05573	0.05795	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	0.00253	-0.00872	0.00351	-0.03507	-0.03512	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.03507	-0.12084	0.00025	-0.00253	-0.00374	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.02748	0.06037	0.1751	0.14289	1.56494	1.56494	-0.56494
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.06037	-0.0341	-0.00275	0.02748	0.02711	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.02748	-0.01552	-0.00604	0.06037	0.06015	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.14289	-0.08072	-0.01751	0.1751	0.17412	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.1751	-0.09892	-0.01429	0.14289	0.14176	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.02711	-0.05663	0.17412	-0.08301	1.49091	1.49091	1.50909
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05663	-0.08545	-0.00271	0.02711	0.02623	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.02711	0.04091	0.00566	-0.05663	-0.05616	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.08301	-0.12527	-0.01741	0.17412	0.17269	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.17412	0.26276	0.0083	-0.08301	-0.0803	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.02623	0.14572	0.17269	0.13445	1.70352	1.70352	3.29648
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.14572	0.48037	-0.00262	0.02623	0.03101	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.02623	0.08646	-0.01457	0.14572	0.14644	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.13445	0.44322	-0.01727	0.17269	0.17695	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.17269	0.56928	-0.01345	0.13445	0.14001	
Factors:								

F[0]:	-0.01453	0.06015	-0.05616	0.14644	0.06816	0.05795	0.03101	
F[1]:	0.07653	0.14176	-0.0803	0.14001	-0.03512	-0.00374	0.17695	
Epoch= 5	Train RMSE=	2.29038	Test RMSE=	1.68144				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.01453	0.06015	0.07653	0.14176	1.63236	1.63236	3.36764
u= 0	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[0][0]$	0.01	0.1	-0.06015	0.20257	0.00145	-0.01453	-0.01248	
i = 1	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[0][1]$	0.01	0.1	0.01453	-0.04892	-0.00602	0.06015	0.0596	
u= 0	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[1][0]$	0.01	0.1	-0.14176	0.47739	-0.00765	0.07653	0.08123	
i = 1	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[1][1]$	0.01	0.1	-0.07653	0.25773	-0.01418	0.14176	0.14419	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01248	-0.05616	0.08123	-0.0803	1.68457	1.68457	1.31543
u= 0	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[0][0]$	0.01	0.1	0.05616	-0.07388	0.00125	-0.01248	-0.01321	
i = 2	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[0][2]$	0.01	0.1	0.01248	-0.01642	0.00562	-0.05616	-0.05627	
u= 0	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[1][0]$	0.01	0.1	0.0803	-0.10563	-0.00812	0.08123	0.08009	
i = 2	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[1][2]$	0.01	0.1	-0.08123	0.10685	0.00803	-0.0803	-0.07915	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.01321	0.14644	0.08009	0.14001	1.88303	1.88303	-0.88303
u= 0	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[0][0]$	0.01	0.1	-0.14644	-0.12931	0.00132	-0.01321	-0.01449	
i = 3	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[0][3]$	0.01	0.1	0.01321	0.01167	-0.01464	0.14644	0.14641	
u= 0	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[1][0]$	0.01	0.1	-0.14001	-0.12363	-0.00801	0.08009	0.07877	
i = 3	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[1][3]$	0.01	0.1	-0.08009	-0.07072	-0.014	0.14001	0.13916	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.06816	0.14641	-0.03512	0.13916	1.88039	1.88039	2.11961
u= 4	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[0][4]$	0.01	0.1	-0.14641	0.31034	-0.00682	0.06816	0.0712	
i = 3	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[0][3]$	0.01	0.1	-0.06816	0.14447	-0.01464	0.14641	0.14771	
u= 4	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[1][4]$	0.01	0.1	-0.13916	0.29497	0.00351	-0.03512	-0.03213	
i = 3	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[1][3]$	0.01	0.1	0.03512	-0.07444	-0.01392	0.13916	0.13828	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.0712	0.05795	-0.03213	-0.00374	1.77974	1.77974	3.22026
u= 4	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[0][4]$	0.01	0.1	-0.05795	0.18663	-0.00712	0.0712	0.07299	
i = 5	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[0][5]$	0.01	0.1	-0.0712	0.22927	-0.0058	0.05795	0.06019	
u= 4	γ	λ	∇u	∇u * err	$\lambda * F_u$	$F_{u,\,previous}$	$F_{u,\,new}$	
$F_u[1][4]$	0.01	0.1	0.00374	-0.01203	0.00321	-0.03213	-0.03222	
i = 5	γ	λ	∇i	∇i * err	$\lambda * F_i$	$F_{i,\,previous}$	$F_{i,\,new}$	
$F_i[1][5]$	0.01	0.1	0.03213	-0.10348	0.00037	-0.00374	-0.00477	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err

6	1	0.03101	0.0596	0.17695	0.14419	1.76855	1.76855	-0.76855
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][6]	0.01	0.1	-0.0596	-0.04581	-0.0031	0.03101	0.03052	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][1]	0.01	0.1	-0.03101	-0.02383	-0.00596	0.0596	0.0593	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][6]	0.01	0.1	-0.14419	-0.11082	-0.0177	0.17695	0.17567	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][1]	0.01	0.1	-0.17695	-0.136	-0.01442	0.14419	0.14269	
u	i	F_u[0][6]	F_i[0][2]	F_u[1][6]	F_i[1][2]	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.03052	-0.05627	0.17567	-0.07915	1.69032	1.69032	1.30968
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][6]	0.01	0.1	0.05627	-0.07369	-0.00305	0.03052	0.02975	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][2]	0.01	0.1	-0.03052	0.03997	0.00563	-0.05627	-0.05581	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][6]	0.01	0.1	0.07915	-0.10366	-0.01757	0.17567	0.17445	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][2]	0.01	0.1	-0.17567	0.23007	0.00791	-0.07915	-0.07677	
u	i	F_u[0][6]	F_i[0][3]	F_u[1][6]	F_i[1][3]	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.02975	0.14771	0.17445	0.13828	1.91644	1.91644	3.08356
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][6]	0.01	0.1	-0.14771	0.45548	-0.00297	0.02975	0.03427	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][3]	0.01	0.1	-0.02975	0.09173	-0.01477	0.14771	0.14848	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][6]	0.01	0.1	-0.13828	0.42639	-0.01745	0.17445	0.17854	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][3]	0.01	0.1	-0.17445	0.53794	-0.01383	0.13828	0.14352	
Factors:								
F[0]:	-0.01449	0.0593	-0.05581	0.14848	0.07299	0.06019	0.03427	
F[1]:	0.07877	0.14269	-0.07677	0.14352	-0.03222	-0.00477	0.17854	
Epoch= 6	Train RMSE=	2.14594	Test RMSE=	1.48194				
u	i	F_u[0][0]	F_i[0][1]	F_u[1][0]	F_i[1][1]	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.01449	0.0593	0.07877	0.14269	1.82285	1.82285	3.17715
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][0]	0.01	0.1	-0.0593	0.18842	0.00145	-0.01449	-0.01259	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][1]	0.01	0.1	0.01449	-0.04604	-0.00593	0.0593	0.05879	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][0]	0.01	0.1	-0.14269	0.45335	-0.00788	0.07877	0.08323	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][1]	0.01	0.1	-0.07877	0.25028	-0.01427	0.14269	0.14505	
u	i	F_u[0][0]	F_i[0][2]	F_u[1][0]	F_i[1][2]	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01259	-0.05581	0.08323	-0.07677	1.87121	1.87121	1.12879
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][0]	0.01	0.1	0.05581	-0.063	0.00126	-0.01259	-0.01321	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][2]	0.01	0.1	0.01259	-0.01421	0.00558	-0.05581	-0.0559	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][0]	0.01	0.1	0.07677	-0.08666	-0.00832	0.08323	0.08228	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][2]	0.01	0.1	-0.08323	0.09395	0.00768	-0.07677	-0.07575	
u	i	F_u[0][0]	F_i[0][3]	F_u[1][0]	F_i[1][3]	$\hat{r}$	$\hat{r}_{[1:5]}$	err

0	3	-0.01321	0.14848	0.08228	0.14352	2.0826	2.0826	-1.0826
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][0]	0.01	0.1	-0.14848	-0.16075	0.00132	-0.01321	-0.0148	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][3]	0.01	0.1	0.01321	0.0143	-0.01485	0.14848	0.14847	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][0]	0.01	0.1	-0.14352	-0.15538	-0.00823	0.08228	0.08064	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][3]	0.01	0.1	-0.08228	-0.08908	-0.01435	0.14352	0.14249	
u	i	F_u[0][4]	F_i[0][3]	F_u[1][4]	F_i[1][3]	r̂	$\hat{r}_{[1:5]}$	err
4	3	0.07299	0.14847	-0.03222	0.14249	2.09602	2.09602	1.90398
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][4]	0.01	0.1	-0.14847	0.28269	-0.0073	0.07299	0.07574	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][3]	0.01	0.1	-0.07299	0.13897	-0.01485	0.14847	0.14972	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][4]	0.01	0.1	-0.14249	0.27129	0.00322	-0.03222	-0.02948	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][3]	0.01	0.1	0.03222	-0.06135	-0.01425	0.14249	0.14173	
u	i	F_u[0][4]	F_i[0][5]	F_u[1][4]	F_i[1][5]	r̂	$\hat{r}_{[1:5]}$	err
4	5	0.07574	0.06019	-0.02948	-0.00477	1.98226	1.98226	3.01774
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][4]	0.01	0.1	-0.06019	0.18164	-0.00757	0.07574	0.07749	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][5]	0.01	0.1	-0.07574	0.22858	-0.00602	0.06019	0.06242	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][4]	0.01	0.1	0.00477	-0.01439	0.00295	-0.02948	-0.02959	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][5]	0.01	0.1	0.02948	-0.08896	0.00048	-0.00477	-0.00565	
u	i	F_u[0][6]	F_i[0][1]	F_u[1][6]	F_i[1][1]	r̂	$\hat{r}_{[1:5]}$	err
6	1	0.03427	0.05879	0.17854	0.14505	1.94565	1.94565	-0.94565
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][6]	0.01	0.1	-0.05879	-0.05559	-0.00343	0.03427	0.03368	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][1]	0.01	0.1	-0.03427	-0.03241	-0.00588	0.05879	0.0584	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][6]	0.01	0.1	-0.14505	-0.13717	-0.01785	0.17854	0.17699	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][1]	0.01	0.1	-0.17854	-0.16884	-0.0145	0.14505	0.14322	
u	i	F_u[0][6]	F_i[0][2]	F_u[1][6]	F_i[1][2]	r̂	$\hat{r}_{[1:5]}$	err
6	2	0.03368	-0.0559	0.17699	-0.07575	1.86394	1.86394	1.13606
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][6]	0.01	0.1	0.0559	-0.0635	-0.00337	0.03368	0.03301	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[0][2]	0.01	0.1	-0.03368	0.03827	0.00559	-0.0559	-0.05546	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[1][6]	0.01	0.1	0.07575	-0.08606	-0.0177	0.17699	0.17596	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
F_i[1][2]	0.01	0.1	-0.17699	0.20107	0.00758	-0.07575	-0.07367	
u	i	F_u[0][6]	F_i[0][3]	F_u[1][6]	F_i[1][3]	r̂	$\hat{r}_{[1:5]}$	err
6	3	0.03301	0.14972	0.17596	0.14173	2.1015	2.1015	2.8985
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
F_u[0][6]	0.01	0.1	-0.14972	0.43395	-0.0033	0.03301	0.03732	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[0][3]$	0.01	0.1	-0.03301	0.09569	-0.01497	0.14972	0.15052	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.14173	0.4108	-0.0176	0.17596	0.17989	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.17596	0.51001	-0.01417	0.14173	0.14669	
Factors:								
F[0]:	-0.0148	0.0584	-0.05546	0.15052	0.07749	0.06242	0.03732	
F[1]:	0.08064	0.14322	-0.07367	0.14669	-0.02959	-0.00565	0.17989	
Epoch= 7	Train RMSE=	2.02681	Test RMSE=	1.30743				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	r^	$\hat{r}_{[1:5]}$	err
0	1	-0.0148	0.0584	0.08064	0.14322	1.98833	1.98833	3.01167
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.0584	0.17589	0.00148	-0.0148	-0.01303	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.0148	-0.04458	-0.00584	0.0584	0.0579	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.14322	0.43132	-0.00806	0.08064	0.08488	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.08064	0.24287	-0.01432	0.14322	0.1455	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	r^	$\hat{r}_{[1:5]}$	err
0	2	-0.01303	-0.05546	0.08488	-0.07367	2.03356	2.03356	0.96644
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05546	-0.0536	0.0013	-0.01303	-0.01355	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	0.01303	-0.01259	0.00555	-0.05546	-0.05553	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.07367	-0.07119	-0.00849	0.08488	0.08408	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.08488	0.08203	0.00737	-0.07367	-0.07277	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	r^	$\hat{r}_{[1:5]}$	err
0	3	-0.01355	0.15052	0.08408	0.14669	2.25575	2.25575	-1.25575
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.15052	-0.18902	0.00136	-0.01355	-0.01543	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.01355	0.01702	-0.01505	0.15052	0.15054	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.14669	-0.1842	-0.00841	0.08408	0.08215	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.08408	-0.10558	-0.01467	0.14669	0.14549	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	r^	$\hat{r}_{[1:5]}$	err
4	3	0.07749	0.15054	-0.02959	0.14549	2.28692	2.28692	1.71308
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.15054	0.25789	-0.00775	0.07749	0.07999	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.07749	0.13274	-0.01505	0.15054	0.15172	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.14549	0.24923	0.00296	-0.02959	-0.02707	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.02959	-0.05069	-0.01455	0.14549	0.14483	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	r^	$\hat{r}_{[1:5]}$	err
4	5	0.07999	0.06242	-0.02707	-0.00565	2.16438	2.16438	2.83562
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.06242	0.17699	-0.008	0.07999	0.08168	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[0][5]$	0.01	0.1	-0.07999	0.22681	-0.00624	0.06242	0.06462	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	0.00565	-0.01603	0.00271	-0.02707	-0.0272	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.02707	-0.07676	0.00057	-0.00565	-0.00641	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.03732	0.0579	0.17989	0.1455	2.09959	2.09959	-1.09959
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.0579	-0.06366	-0.00373	0.03732	0.03665	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.03732	-0.04104	-0.00579	0.0579	0.05743	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.1455	-0.15999	-0.01799	0.17989	0.17811	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.17989	-0.1978	-0.01455	0.1455	0.14338	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.03665	-0.05553	0.17811	-0.07277	2.01503	2.01503	0.98497
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05553	-0.0547	-0.00366	0.03665	0.03606	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.03665	0.0361	0.00555	-0.05553	-0.05511	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.07277	-0.07168	-0.01781	0.17811	0.17721	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.17811	0.17543	0.00728	-0.07277	-0.07094	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.03606	0.15172	0.17721	0.14483	2.26223	2.26223	2.73777
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.15172	0.41537	-0.00361	0.03606	0.04018	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.03606	0.09873	-0.01517	0.15172	0.15256	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.14483	0.39652	-0.01772	0.17721	0.181	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.17721	0.48517	-0.01448	0.14483	0.14954	
Factors:								
F[0]:	-0.01543	0.05743	-0.05511	0.15256	0.08168	0.06462	0.04018	
F[1]:	0.08215	0.14338	-0.07094	0.14954	-0.0272	-0.00641	0.181	
Epoch= 8	Train RMSE=	1.92904	Test RMSE=	1.15476				
u	i	$F_u[0][0]$	$F_i[0][1]$	$F_u[1][0]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	1	-0.01543	0.05743	0.08215	0.14338	2.13198	2.13198	2.86802
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.05743	0.16471	0.00154	-0.01543	-0.01377	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	0.01543	-0.04425	-0.00574	0.05743	0.05693	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.14338	0.41121	-0.00822	0.08215	0.08618	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.08215	0.23562	-0.01434	0.14338	0.14559	
u	i	$F_u[0][0]$	$F_i[0][2]$	$F_u[1][0]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	2	-0.01377	-0.05511	0.08618	-0.07094	2.17469	2.17469	0.82531
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	0.05511	-0.04549	0.00138	-0.01377	-0.01421	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[0][2]$	0.01	0.1	0.01377	-0.01136	0.00551	-0.05511	-0.05517	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	0.07094	-0.05855	-0.00862	0.08618	0.08551	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][2]$	0.01	0.1	-0.08618	0.07113	0.00709	-0.07094	-0.07016	
u	i	$F_u[0][0]$	$F_i[0][3]$	$F_u[1][0]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
0	3	-0.01421	0.15256	0.08551	0.14954	2.40581	2.40581	-1.40581
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][0]$	0.01	0.1	-0.15256	-0.21446	0.00142	-0.01421	-0.01634	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	0.01421	0.01997	-0.01526	0.15256	0.1526	
u= 0	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][0]$	0.01	0.1	-0.14954	-0.21023	-0.00855	0.08551	0.08332	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.08551	-0.12021	-0.01495	0.14954	0.14819	
u	i	$F_u[0][4]$	$F_i[0][3]$	$F_u[1][4]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	3	0.08168	0.1526	-0.0272	0.14819	2.45611	2.45611	1.54389
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.1526	0.2356	-0.00817	0.08168	0.08395	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.08168	0.1261	-0.01526	0.1526	0.15371	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	-0.14819	0.22879	0.00272	-0.0272	-0.02489	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	0.0272	-0.042	-0.01482	0.14819	0.14762	
u	i	$F_u[0][4]$	$F_i[0][5]$	$F_u[1][4]$	$F_i[1][5]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
4	5	0.08395	0.06462	-0.02489	-0.00641	2.32852	2.32852	2.67148
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][4]$	0.01	0.1	-0.06462	0.17263	-0.0084	0.08395	0.08559	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][5]$	0.01	0.1	-0.08395	0.22427	-0.00646	0.06462	0.0668	
u= 4	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][4]$	0.01	0.1	0.00641	-0.01713	0.00249	-0.02489	-0.02503	
i = 5	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][5]$	0.01	0.1	0.02489	-0.06649	0.00064	-0.00641	-0.00707	
u	i	$F_u[0][6]$	$F_i[0][1]$	$F_u[1][6]$	$F_i[1][1]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	1	0.04018	0.05693	0.181	0.14559	2.23332	2.23332	-1.23332
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.05693	-0.07021	-0.00402	0.04018	0.03944	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][1]$	0.01	0.1	-0.04018	-0.04956	-0.00569	0.05693	0.05638	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.14559	-0.17956	-0.0181	0.181	0.17902	
i = 1	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][1]$	0.01	0.1	-0.181	-0.22323	-0.01456	0.14559	0.14321	
u	i	$F_u[0][6]$	$F_i[0][2]$	$F_u[1][6]$	$F_i[1][2]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	2	0.03944	-0.05517	0.17902	-0.07016	2.14645	2.14645	0.85355
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	0.05517	-0.04709	-0.00394	0.03944	0.03893	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][2]$	0.01	0.1	-0.03944	0.03366	0.00552	-0.05517	-0.05478	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	0.07016	-0.05989	-0.0179	0.17902	0.17825	
i = 2	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	

$F_i[1][2]$	0.01	0.1	-0.17902	0.15281	0.00702	-0.07016	-0.06856	
u	i	$F_u[0][6]$	$F_i[0][3]$	$F_u[1][6]$	$F_i[1][3]$	$\hat{r}$	$\hat{r}_{[1:5]}$	err
6	3	0.03893	0.15371	0.17825	0.14762	2.4017	2.4017	2.5983
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[0][6]$	0.01	0.1	-0.15371	0.39939	-0.00389	0.03893	0.04288	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[0][3]$	0.01	0.1	-0.03893	0.10115	-0.01537	0.15371	0.15457	
u= 6	γ	λ	∇u	∇u * err	λ * F_u	$F_{u,\ previous}$	$F_{u,\ new}$	
$F_u[1][6]$	0.01	0.1	-0.14762	0.38356	-0.01782	0.17825	0.1819	
i = 3	γ	λ	∇i	∇i * err	λ * F_i	$F_{i,\ previous}$	$F_{i,\ new}$	
$F_i[1][3]$	0.01	0.1	-0.17825	0.46314	-0.01476	0.14762	0.1521	
Factors:								
F[0]:	-0.01634	0.05638	-0.05478	0.15457	0.08559	0.0668	0.04288	
F[1]:	0.08332	0.14321	-0.06856	0.1521	-0.02503	-0.00707	0.1819	
Epoch= 9	Train RMSE=	1.84911	Test RMSE=	1.02119				

Tab. 23: Example run of SGD for 10 Epochs on a toy dataset defined at the beginning of this Appendix. The algorithm runs for 10 (0-9) epochs/iterations on 8 training samples and infers 2 factors for each of the 7 attributes (3 users and 4 items), these are all mapped into the same factor matrix - F. Testing is done on 2 samples, which are included in the training set. Note: no global factors are any other implicit or explicit input data is taken into account – only pure SGD factorization.

8 Bibliography

[1] R. M. Bell and Y. Koren and C. Volinsky, 2009, Matrix Factorization Techniques for Recommender Systems. IEEE Computer 42, 8, 42–49.

[2] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights.", In Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA, pages 43–52. IEEE Computer Society, 2007.

[3] Pilászy, I., Zibriczky, D., Tikk, D., "Fast ALS-based matrix factorization for explicit and implicit feedback datasets", In: Recsys'10: ACM Conf. on Recommender Systems, Barcelona, Spain (2010) 71–78

[4] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, John T. Riedl, "Application of Dimensionality Reduction in Recommender System -- A Case Study", Journal Foundations and Trends in Human-Computer Interaction, Volume 4 Issue 2, February 2011, Pages 81-173

[5] Yehuda Koren, "Factorization Meets the Neighborhood - a Multifaceted Collaborative Filtering Model", KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA

[6] R. Bell and Y. Koren, "Improved neighborhood-based collaborative filtering", in Proceedings of KDD Cup and Workshop, 2007

[7] Arkadiusz Paterek, "Improving regularized singular value decomposition for collaborative filtering", KDDCup.07 August 12, 2007, San Jose, California, USA

[8] Gabor Takacs, Istvan Pilaszy, Bottyan Nemeth, Domonkos Tikk, 2009, "Scalable collaborative filtering approaches for large recommender systems", Journal ACM Transactions on Internet Technology (TOIT) TOIT Homepage, Volume 17 Issue 3, June 2017 Issue-in-Progress, Article No. 31

[9] Y.F. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets", Proc. IEEE Int'l Conf. Data Mining (ICDM 08), IEEE CS Press, 2008, pp. 263-272.

[10] R. Burke, A. Felfernig, and M. Goeker. Recommender Systems: An Overview, AI Magazine, AAAI, 32(3):13-18, 2011

[11] G. Adomavicius and A. Tuzhilin, "Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions", IEEE Transactions on Knowledge and Data Engineering 17 (2005), 634–749.

[12] Xiangnan He Hanwang Zhang Min-Yen Kan Tat-Seng Chua. "Fast Matrix Factorization for Online Recommendation with Implicit Feedback". SIGIR '16, July 17-21, 2016, Pisa, Italy.

[13] R. Burke. 2002 Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction. 12, 4, 331-370

[14] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry", Communications of the ACM 35 (1992), 61–70.

[15] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, "An Algorithmic Framework for Performing Collaborative Filtering", Proc. 22nd ACM SIGIR Conference on Information Retrieval, pp. 230–237, 1999.

[16] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl, "GroupLens: Applying Collaborative Filtering to Usenet News", Communications of the ACM 40 (1997), 77–87, www.grouplens.org.

[17] G. Linden, B. Smith and J. York, "Amazon.com Recommendations: Item-to-item Collaborative Filtering", IEEE Internet Computing 7 (2003), 76–80.

[18] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, "An Algorithmic Framework for Performing Collaborative Filtering", Proc. 22nd ACM SIGIR Conference on Information Retrieval, pp. 230–237, 1999.

[19] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl, "GroupLens: Applying Collaborative Filtering to Usenet News", Communications of the ACM 40 (1997), 77–87, www.grouplens.org.

[20] G. Linden, B. Smith and J. York, "Amazon.com Recommendations: Item-to-item Collaborative Filtering", IEEE Internet Computing 7 (2003), 76–80.

[21] Burke, R., 2000, Knowledge-based recommender systems. Encyclopedia of Library and Information Systems 69, 32, 180–200.

[22] Felfernig, A., Friedrich, G., Jannach, D. and Zanker, M, 2006. An Integrated Environment for the Development of Knowledge-based Recommender Applications. Intl. Journal of Electronic Commerce (IJEC) 11, 2, 11–34.

[23] R. Bell, Y. Koren and C. Volinsky. The BellKor 2008 Solution to the Netflix Prize. 2008.

[24] Markovsky I. (2012) Low-Rank Approximation: Algorithms, Implementation, Applications, Springer

[25] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk. Matrix factorization and neighbor based algorithms for the Netflix Prize problem. Proc. 2008 ACM Conference on Recommender Systems (RECSYS'08), pp. 267–274, 2008.

[26] Robert M. Bell, Yehuda Koren and Chris Volinsky, "Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems", KDD'07, August 12–15, 2007, San Jose, California, USA

[27] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization", In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, Advances in Neural Information Processing Systems 20, Cambridge, MA, 2008. MIT Press.

[28] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin, "A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems", RecSys'13, October 12–16, 2013, Hong Kong, China

[29] R. Salakhutdinov, A. Mnih, "Bayesian probabilistic matrix factorization using Markov chain Monte Carlo", In Proceedings of the 25th International Conference on Machine Learning, volume 25, 2008.

[30] Andrew I. Schein; Alexandrin Popescul; Lyle H. Ungar; David M. Pennock (2002). Methods and Metrics for Cold-Start Recommendations. Proceedings of the 25th Annual International ACM SIGIR Conference

on Research and Development in Information Retrieval (SIGIR 2002). New York City, New York: ACM. pp. 253–260. ISBN 1-58113-561-0.

[31] Elahi, Mehdi; Ricci, Francesco; Rubens, Neil. Active Learning in Collaborative Filtering Recommender Systems. Springer International Publishing. pp. 113–124. ISBN 978-3-319-10491-1.

[32] Elahi, Mehdi; Ricci, Francesco; Rubens, Neil (2016). A survey of active learning in collaborative filtering recommender systems. Computer Science Review

[33] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering", In Proc. of ICML'07, the 24 th Int. Conf. on Machine Learning, pages 791–798, Corvallis, OR, USA, 2007.

[34] Andreas Töscher, Michael Jahrer, Robert Legenstein, "Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems", 2nd Netflix-KDD Workshop, August 24, 2008, Las Vegas, NV, USA

[35] Mingrui Wu, "Collaborative Filtering via Ensembles of Matrix Factorizations", KDDCup. 07, August 12, 2007, San Jose, California, USA

[36] Yehuda Koren, "Collaborative Filtering with Temporal Dynamics", KDD'09, June 28–July 1, 2009, Paris, France.

[37] Gábor Takács, István Pilászy, Domonkos Tikk , "Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering", RecSys'11, October 23–27, 2011, Chicago, Illinois, USA

[38] Balázs Hidasi, Domonkos Tikk, "Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback", Proceedings of the 2012 European conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II

[39] Xiangnan He, Hanwang Zhang, Min-Yen Kan, Tat-Seng Chua, "Fast Matrix Factorization for Online Recommendation with Implicit Feedback", SIGIR '16, July 17-21, 2016, Pisa, Italy

[40] Gábor Takács, Domonkos Tikk, "Alternating Least Squares for Personalized Ranking", RecSys'12, September 9–13, 2012, Dublin, Ireland.

[41] Mohsen Jamali, Martin Ester, "A Matrix Factorization Technique with Trust Propagation for Recommendation in Social Networks", RecSys2010, September 26–30, 2010, Barcelona, Spain.

[42] Daniel Lamprecht, Markus Strohmaier, Denis Helic, "Improving Reachability and Navigability in Recommender Systems", i-KNOW '15 Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business, Article No. 21

[43] A. CICHOCKI and R. ZDUNEK, "Regularized Alternating Least Squares Algorithms for Non-negative Matrix/Tensor Factorization", ISNN '07 Proceedings of the 4th international symposium on Neural Networks: Advances in Neural Networks, Part III, Pages 793 - 802

[44] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan , "Large-scale Parallel Collaborative Filtering for the Netflix Prize", AAIM '08 Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management

[45] Robert M. Bell and Yehuda Koren, "Lessons from the Netflix Prize Challenge", ACM SIGKDD Explorations Newsletter - Special issue on visual analytics, Volume 9 Issue 2, December 2007, Pages 75-79

[46] Gábor Takács, István Pilászy, Bottyán Németh, Domonkos Tikk, "Major components of the Gravity Recommendation System", ACM SIGKDD Explorations Newsletter - Special issue on visual analytics, Volume 9 Issue 2, December 2007, Pages 80-83

[47] James Bennett, Stan Lanning, "The Netflix Prize", KDDCup’07, August 12, 2007, San Jose, California, USA.

[48] Ng, Andrew Y, "Feature selection, L1 vs. L2 regularization, and rotational invariance", Proceedings of the 21 st International Conference on Machine Learning, Banff, Canada, 2004.

[49] Yehuda Koren. The BellKor Solution to the Netflix Grand Prize. 2009

[50] A. Töscher and M. Jahrer. The BigChaos Solution to the Netflix Prize 2008, 2008.

[51] A. Töscher and M. Jahrer. The BigChaos Solution to the Netflix Grand Prize, 2009.

[52] R. Bell, Y. Koren and C. Volinsky. The BellKor Solution to the Netflix Prize. 2007.

[53] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich 2011. Recommender Systems An Introduction

[54] P. Resnick and H. R. Varian. 1997. Recommender systems. Commun. ACM 40, 3, 56-58. 1997