Cold-start recommendations for the user- and item-based recommender system algorithm k-Nearest Neighbors

Cold start-rekommendationer för den användar- och objektbaserade rekommendationsalgoritmen k-Nearest Neighbors

Robert Lorentz,
Oskar Ek

2017-05-12

Abstract

Recommender systems apply machine learning methods to solve the task of providing appropriate suggestions to users in both static and dynamic environments. An example of this is a movie service like Netflix that recommends movies to its users. Although many algorithms have been proposed, making predictions for users with few ratings remains a challenge in recommender systems.

In this study the performance of the algorithm k-NN, both user- and item-based, was empirically evaluated. This was done using the MovieLens 1M and 100K datasets in scenarios where the users have between 1 and 9 ratings, simulating cold-start scenarios of various degree. The results were then compared with the accuracy of the algorithm in a simulated normal case, to see how the cold-start affected the two algorithms, and which one of them that handled it best.

In summary, this report shows that user-based k-NN performs better in relation to item-based k-NN for new users having few rated items. Overall the accuracy improved as the number of ratings increased for the new users for both user- and item-based k-NN.

Sammanfattning

Rekommendationssystem tillämpar maskininlärningsmetoder för att lösa uppgiften att tillhandahålla lämpliga förslag till användare i både statiska och dynamiska miljöer. Ett exempel på detta är en filmtjänst som Netflix som rekommenderar filmer till sina användare. Även om många algoritmer har utvecklats för ändamålet, är det fortfarande en utmaning för rekommendationssystem att göra förutsägelser för användare med få betyg.

I denna studie utvärderades algoritmen K-NN, både användar- och objektbaserad, empiriskt. Detta gjordes med hjälp av MovieLens 1M och 100K dataset i scenarier där användarna har mellan 1 och 9 betyg och simulerar cold start-scenarier av olika grad. Resultaten jämfördes sedan med ett simulerat normalfall, för att se hur cold start påverkat de båda algoritmerna och vilken av dem som hanterade det bäst.

Sammanfattningsvis visar denna rapport att användarbaserad k-NN fungerar bättre än objektbaserad k-NN för nya användare med få betygsatta objekt. Sammantaget förbättrades noggrannheten eftersom antalet betyg ökade för de nya användarna för både användar- och objektbaserad k-NN.

Contents

1. Introduction

In today's society large amounts of data is available for the common person. An example of this is Netflix, where there is a large diversity of movies to be watched.
An ongoing problem for the companies such as Netflix is to suggest the best suitable movies for the user, so that the user wants to keep paying for their services. For this task one can use a *Recommender system* [1].

A newly introduced user in a system may experience the issue of *cold start,* where the user needs to leave some amount of ratings in order for the recommender system to leave accurate suggestions. Cold-start is a famous problem in many *neighborhood based collaborative* recommender systems, where the recommendations are based on the ratings of other *similar* users. The problem exists for new users when the similarity to other users are hard to calculate due to little or no known ratings [3].

The user may decide to leave the system if the predictions are not satisfying, which is a big problem for recommender systems. A way of approaching the cold-start problem, is to have the user *explicitly* rate an initial amount of items in order for the recommender system to leave good predictions. In this study two version of the algorithm k-Nearest Neighbors will be investigated, user-based and item-based, and how cold-start affects them [1]. This study will also investigate how the number of explicit ratings a new user has left affects the predictions in a simulated cold-start scenario.

1.1 Purpose

The purpose of this report is to examine the collaborative filtering algorithm user- and item-based k-NN (k-Nearest Neighbors) and how well they perform with the cold-start problem. Cold-start is a known issue for k-NN algorithms due to the lack of availability of similar users when a user has few ratings [3].

Some applications collect information about users from social media and other sources, forming a warm-start, which has been shown to increase accuracy [2].
Warm-start can also include interviewing the user and receiving explicit ratings to prevent the

cold-start scenario. This report investigates the evaluation performance of the user based algorithm k-NN when the amount of ratings left by the user is increased in a cold-start scenario.

1.2 Problem statement

- How does the recommender system algorithm user- based k-NN get affected compared to the item- based k-NN in cold-start conditions?
- How does the number of explicit ratings made by new users influence the accuracy of the algorithms in cold-start situations?

1.3 Limitations

This study will focus on the cold start scenario for the user- and item-based recommender system algorithm k-NN (k-Nearest Neighbors) and how the number of explicit ratings users make affects the accuracy of the recommender algorithm, this compared with the normal case where the dataset is unchanged.
We will in this report only investigate explicit data out of ratings, in an offline static context, and not a dynamic database where users can get inserted into the database at different times.

The datasets used for this study is MovieLens 1M, MovieLens 100K [8] and made available from MovieLens free for education and studies.
The recommender system implementer Surprise [10] is used to operate the algorithms mentioned above, with limitations to the algorithms and applications of the system.

2. Background

2.1 Recommender system

We begin clarifying what a recommender system is and the general definition. Recommender systems in software engineering (RSSEs) aim to help users in their selections in systems where large amounts of products exists.

RSSEs work by recommending items either *implicitly* or *explicitly* depending on the system. The implicit approach infers to monitor user behaviour without the knowledge of the user, and basing recommendations out of the reactions of incoming data. The ambition is to learn from the data item presented to the user. With the explicit approach users are required to actively specify preferences of items to the system, often by entering a value on 5-point or 7-point *Thurston scales.* Lower values commonly represents less appreciation of the product while a higher values indicates the user's liking of the product. The RSSE then examines this information and suggests items [1].

Item recommendations can be categorized into two main groups, *Content-based recommendations* and *Collaborative filtering*. Content-based recommendations are based on finding items that are similar to the ones that have already been liked *by the individual user*. The collaborative filtering system takes a more social approach, where other users are considered.

2.2 Collaborative Filtering

Collaborative filtering involves suggesting items based on similarity calculations between users and/or items [4].

The assumption for collaborative filtering is that similar users will like the same items (same items will get liked if one user likes one of these). Collaborative filtering algorithms measuring the distance between users and providing their predictions based on that are called *user-based*, while collaborative filtering algorithms measuring the distance between items are called *item-based.*

2.2.1 Memory-based collaborative filtering

Memory-based algorithms applies the method of finding a set of users with statistical techniques that trends to appreciate the same products, also known as *neighbours.* Once a set of *neighbours* is calculated the RSSEs use different types of algorithms to associate preferences of neighbours and predict the top-N recommendations for the user. This technique is also known as *nearest-neighbour.*

2.3 Cold start

New users or inactive users not contributing enough ratings can suffer from the cold start issue, where the collaborative filtering method has insufficient data to provide good recommendations. This way, the accuracy error increases for cold-start scenarios for collaborative filtering algorithms [3].

2.5 Algorithms

The algorithm focused on in this experiment is k-NN, which measures the distance between nodes in a graph in respect to user or item similarities. The similarities for the neighborhood-based implementations can be calculated with different metrics, this study will use the *Pearson correlation coefficient*. This is a widely and commonly used metric for similarity measurements, used in earlier studies of k-NN benchmark analysis for movie datasets [14][15].
This similarity formula calculates the similarity between user *u* and user *v:*

$$\text{pearson\_sim}(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

where:

- I_{uv} - The items rated by both user *u* and user *v*.
- r_{ui} - The rating given by user *u* to item *i.*

- r_{vi} - The rating given by user *v* to item *i.*
- μ_u - The mean rating given by user *u.*
- μ_v - The mean rating given by user *v.*

This similarity formula calculates the similarity between the item *i* and the item *j*:

$$\text{pearson\_sim}(i,j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i) \cdot (r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}$$

where:

- U_{ij} - The users which have rated both item *i* and *j*.
- r_{ui} - The rating given by user *u* to item *i.*
- μ_i - The mean rating for item *i.*
- μ_j - The mean rating for item *j.*

2.5.1 User-based k-Nearest Neighbors

The memory-based algorithm k-Nearest Neighbour is one of the most popular CF approaches to RSSEs. The basic strategy used by this algorithm is to measure a weight for the user's score by looking at votes by other k similar users. In this study, this algorithm will be referred to as UserKNN.

The prediction $\widehat{r_{ui}}$ for an item rating for a user is calculated with the following formula [9]:

$$\widehat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

where:

- $N_i^k(u)$ - The k-neighborhood of users with respect to the user u.
- $sim(u, v)$ - The similarity function explained above between user u and user v.
- r_{vi} - The rating for item i given by user v.

2.5.2 Item k-Nearest Neighbors

Another memory based neighbor algorithm, calculating similarity between items instead of users in the graph. In this study, this algorithm will be referred to as ItemKNN. The rating prediction for a user of an item is calculated by the following formula [9]:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} sim(i,j)}$$

where:

- $N_u^k(i)$ - The *k*-neighborhood of items with respect to the item *i.*
- $sim(i,j)$ - The similarity function explained above between item *i* and item *j.*
- r_{uj} - The rating for item *j* given by user *u.*

2.6 Accuracy metric

To measure the accuracy of a recommender system, there are two metrics that are commonly used; the *root-mean-squared-error* (RMSE) and the *mean absolute error* (MAE). Accuracy in this context means how well the predicted ratings matches the actual ratings of the model.

The RMSE is calculated using the following formula:

$$RMSE(T) = \sqrt{\frac{\sum_{(u,i) \in T} (\hat{r}_{ui} - r_{ui})^2}{N}}$$

The MAE is calculated using the formula:

$$MAE(T) = \frac{\sum_{(u,i) \in T} |\hat{r}_{ui} - r_{ui}|}{N}$$

In the formulas above, T is the set of (user, item)-tuples that constitute the training set. r_{ui} is the actual rating of the item i given by user u, and $\hat{r}_{ui}$ is the corresponding prediction (guess). The constant N is the amount of ratings in the set [12].

A difference between these two metrics is that the RMSE value will be more affected by

large errors in the prediction. This is because the errors for each item are squared in the formula, giving large error a bigger impact on the final value. This makes RMSE more suitable as a metric if large errors are of great importance.

2.7 Related work

The majority of previous research uses RMSE and MAE and measures the accuracy between several RSSE algorithms in their respective dataset environment. This way the suggestions made by the algorithms can be compared and evaluated, to determine which one is the best performing [6][2][5].

Different approaches have been made to improve the performance for movie recommendations in RSSE systems in cold-start scenarios. One of those is to use decision trees with interview questions all depending on the answers of the previous questions. This way more appropriate suggestions can be made with fewer questions [5]. Other approaches include converting a cold-start user into a warm-start user by extracting information about the user in social media [2].

Other related work benchmarks k-NN for the netflix dataset, receiving good predictions in relation to other popular RSSEs for the movie dataset. The k-value used in their research was equal to 100. [16]

2.8 Explanation of terms

In this section some terms and concepts that will be used in the report are explained.

2.8.1 Training set

A training set is the part of the dataset that the recommender system algorithm will use as training data when being evaluated. This is the data that the rating predictions will be based on.

2.8.2 Test set

This is the part of the dataset that will be used as measurement for the accuracy of the algorithms. The accuracy will be determined by looking at how closely the predictions resemble this actual set of data.

2.8.3 k-fold cross-validation

K-fold cross-validations is the act of dividing a dataset into *k* equally sized subsets and use each of these subsets as the test set one time each, with the other subsets combined acting as the training set. These *k* different setups will each generate an accuracy measurement, which will be combined into a single result, usually by taking the mean [13].

2.9 Hypothesis

For the user-based algorithm k-NN the neighborhood with users, $N_i^k(u)$, will be of users which have rated similarly for the given movies the newly added user have explicitly rated. The training set will include lots of users and ratings, thus the more explicit ratings given by a new user, the better the performance accuracy one would assume. This is because more ratings equals a better representation of taste for a user, which leads to the neighborhood $N_i^k(u)$ having users with a more probable similar taste, than one neighborhood based out of the taste of few movies rated by a new user. Therefore the performance accuracy should increase as the amount of ratings increases.

For the item-based algorithm k-NN the neighborhood $N_u^k(i)$ will be of items rated similarly by other users. These items in the neighborhood is rated by the new user, denoted by r_{uj}.
For new users with few ratings, the neighborhood will consist of the few movies rated, thus the algorithm has little data to predict from.
For the user-based k-NN, the algorithm can base the predictions for the other similar users' product ratings, having more collaborative data to predict from.

Consequently the assumption is that the user-based k-NN will perform better in situations where the new user has explicitly input few ratings. Both algorithms is assumed to perform better in situations where the new user has more ratings since the neighborhoods calculated ($N_i^k(u)$ and $N_i^k(u)$) would have more data to base the predictions from.

3. Method

3.1 Structure of the study

This study tested the cold-start scenario, with a varying amount of initial items rated by the *cold start* users; that is, the users that are assumed to be new to the system. This was done in two steps.

Step one - building the cold-start simulation data:
The users of the dataset are partitioned into four equally sized sets, so each contains 25% of the full set of users. Then the following procedure is performed for each of these sets, one time each:

Let the users in the set represent the cold-start users, and the users of the other three sets represent the non-cold-start users. For all the cold-start users, place *n* of their ratings in the training set, whereas the rest are instead appended to the test set. For the non-cold-start users, all of their ratings are added to the training set. This simulates the situation in which the cold-start users has only provided *n* ratings, and the rest of their ratings should be predicted by the algorithm.

Repeat this for different values of *n*, ranging from 1 up to 9. The value 9 is chosen as a maximum, due to earlier research partitioning similarly.
The reason why 0 is not included is because the implementation of the algorithms in use would in that case only return a mean value of all the item's ratings as a prediction. This is not interesting since it does not demonstrate the cold start problem.

This first step is the responsibility of the program ColdStartGenerator, whose code is included under Attachments, section 8.3.1.

Step two - testing the algorithms with the data:
Next, the two algorithms; user-based kNN and item-based kNN, are tested using the data generated in the previous step. For each algorithm, the test is conducted as follows:

For each *n*-value, the algorithm is trained and then tested using *4-fold cross validation*. This

method has been used in previous research for similar purposes in similar datasets [5].

The algorithms are also tested in the *normal case* scenario, which means that 75% of the ratings are used as training data, and 25% are used as test data. This is also conducted using 4-fold cross validation. The *mean rating* scenario is also tested since Surprise uses the mean rating when predicting for a user with 0 ratings, and compared with the algorithms.

This second step is the responsibility of the two Python programs included under Attachments, section 8.3.2 and 8.3.3.

Users having equal to or under 3 ratings will be assumed *extremely cold-start users*, users in the interval [4,9] will be assumed *cold-start users.* Same partitioning can be seen in other research [7]. The difference in our case is that we do not consider the case of users with 0 ratings, so these are not included in our extremely cold-start users.

The neighborhood size, *k*, is chosen to be 100 in this study. This value was used in other similar research using k-NN algorithms in movie datasets and thus seemed most fitting for our research [16]. The accuracy metrics RMSE and MAE were chosen to be evaluated due to earlier research often using both of these accuracy metrics [6][2][5].

3.2 Dataset

3.2.1 MovieLens

For our dataset we are using MovieLens, a set of movie ratings made available by GroupLens, a research lab at the university of Minnesota. We use two two sizes of it, MovieLens 100K containing 100.000 ratings, and MovieLens 1M containing 1.000.000 ratings. The ratings are explicit containing user id, movie id, rating [1,5] and a time tag. The different datasets on the website are free for research and education.

3.3 Software

3.2.1 Python Surprise

Python Surprise is developed and maintained by Nicolas Hug. Surprise is an easy to use python library kit used to evaluate, analyze and compare the performance of RSSE algorithms. It provides the user with ready to use RSSE algorithms, with support to develop new algorithms [10].

For impossible rating predictions in Surprise, for example when a user does not have any previous ratings, the predicted rating will be the mean of all the ratings for the specific item. This is the case for both the UserKNN and ItemKNN implementations.

4. Results

This section includes the experimental results for our study beginning with graphs where y-value stands for the accuracy and x-value for the number of ratings per user. Then, the difference for the various intervals (*extremely cold-start* and *cold-start*) will be compared with the normal case represented in percentage. A positive percentage in the results means that the accuracy metric is less than the corresponding accuracy metric for the normal case. A negative, in the same way, represents a higher value in accuracy.

4.1 Normal case

These results below are for the algorithms run on the dataset in the normal case scenario.

100k MovieLens	UserKNN	ItemKNN	Mean rating
RMSE	1,0157	1,0401	1,1248
MAE	0,8059	0,8297	0,94439

Table 1: MovieLens 100K Normal case accuracy.

1M MovieLens	UserKNN	ItemKNN	Mean rating
RMSE	0,9677	0,9884	1,11692
MAE	0,7707	0,7924	0,93374

Table 2: MovieLens 1M Normal case accuracy.

4.2 Explicit rating and accuracy

These results represent the results in accuracy measuring for the algorithms in relation to amount of explicit ratings by the users.

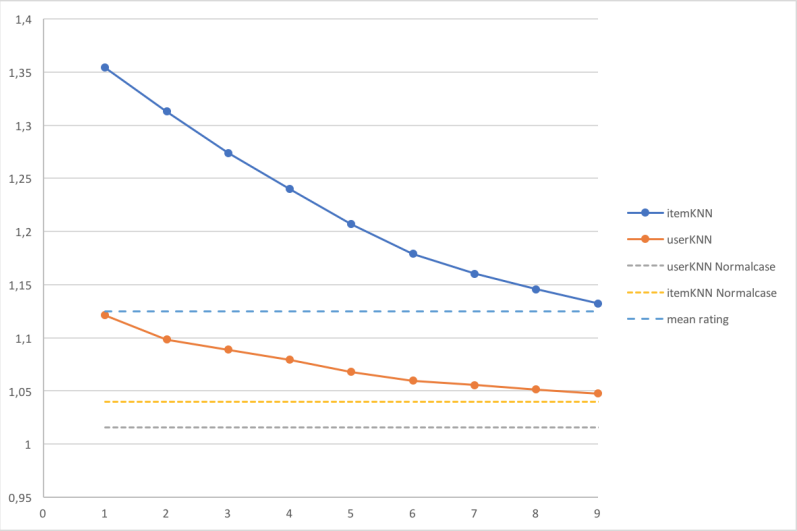

Figure 1: MovieLens 100K, RMSE.

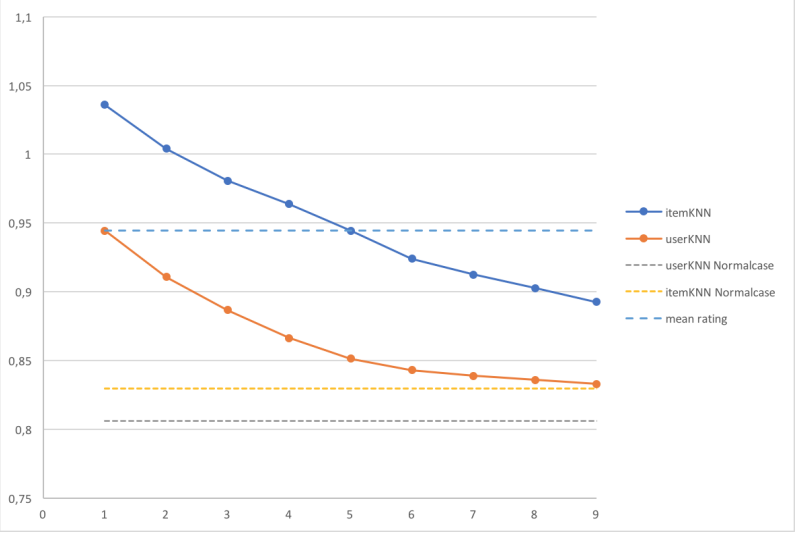

Figure 2: MovieLens 100K, MAE.

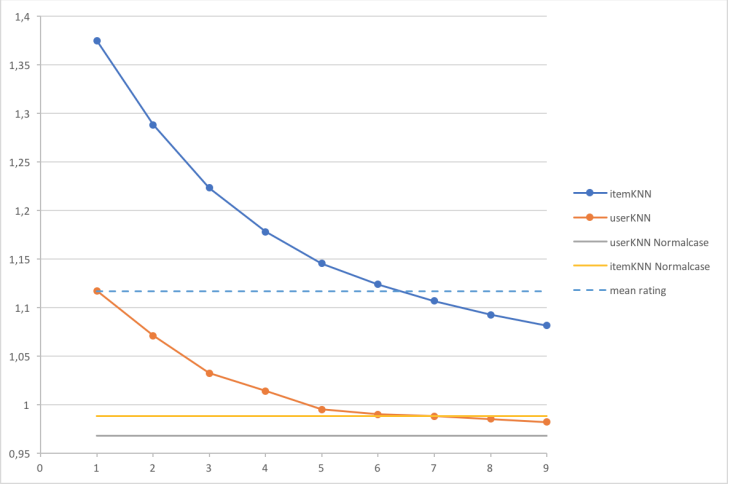

Figure 3: MovieLens 1M, RMSE.

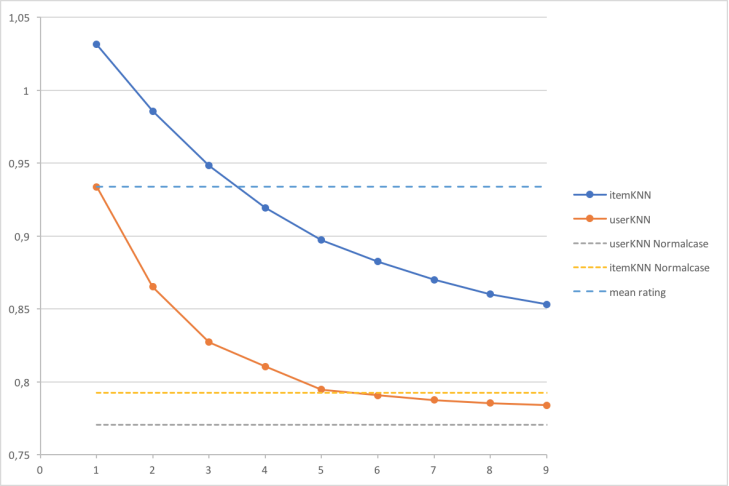

Figure 4: MovieLens 1M, MAE.

As can be seen in the graph, as the amount of ratings for the algorithms increases, the accuracy in relation to RMSE and MAE steadily improves. For ItemKNN the amount of ratings 1-3 for RMSE and 1-6 for MAE gives higher RMSE for a user than using the mean rating with respect to MovieLens 1M. Considering MovieLens 100K and the RMSE results, the mean rating is a better way of predicting for all the ratings 1-9. By looking at the MAE results, using the mean resulted in lower values for the first 4 ratings. The bigger dataset received overall better predictions than the smaller dataset as can be seen in the graph. Another observation is that for all the scenarios UserKNN performs better with higher accuracy measurements both for RMSE and MAE, while simultaneously having lower MAE and RMSE than using the mean rating for an item.

4.3 Extremely cold-start (1-3 ratings)

These results represents the accuracy in RMSE and MAE for users rating ≤ 3 movies in relation to the normal case.

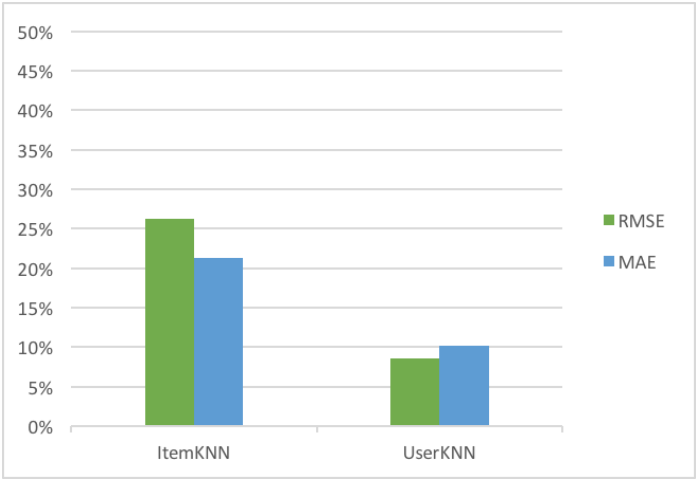

Figure 5: MovieLens 100k.

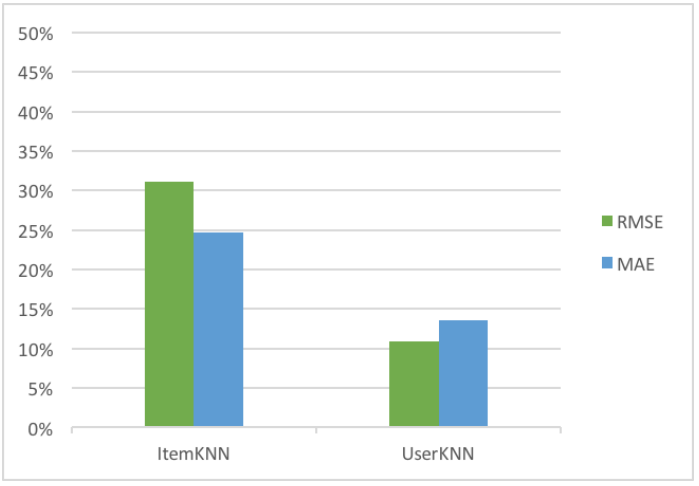

Figure 6: MovieLens 1M.

The increase in percentage for ItemKNN is very high compared to UserKNN, 8% and 11% versus 25% and 30,5% with respect to RMSE. Between the datasets ItemKNN receives about 5% worse predictions for MovieLens 1M compared to MovieLens 100K considering RMSE. UserKNN also performed worse with the larger dataset with 3% higher RMSE.

4.4 Cold-start (4-9 ratings)

These results are for users rating $4 \leq n \leq 9$ movies, where n is the number of explicit ratings.

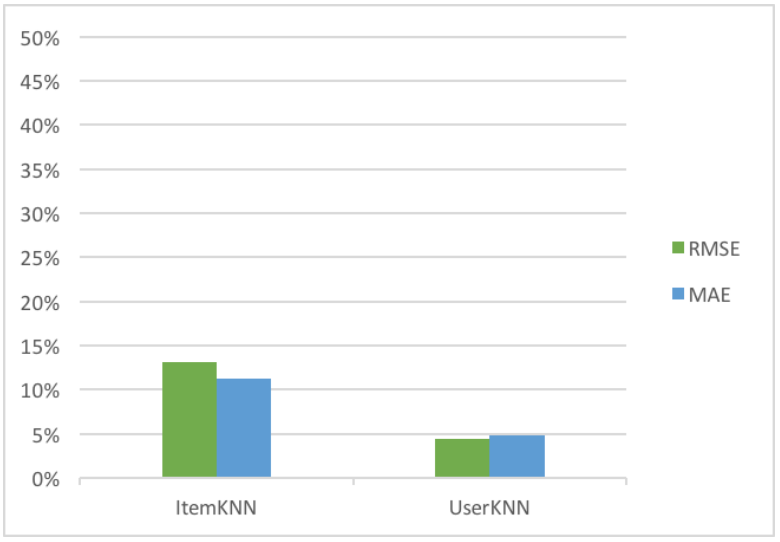

Figure 7: MovieLens 100k.

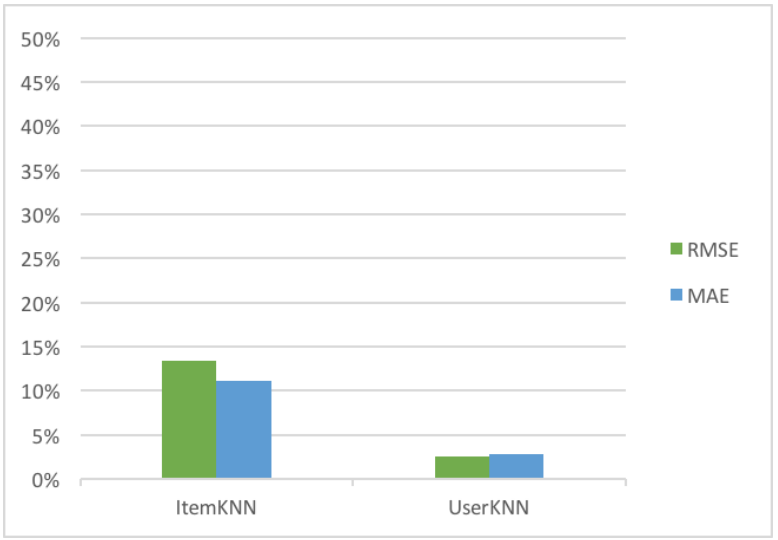

Figure 8: MovieLens 1M.

The observations here are that the increase in percentage is lower in relation to the extremely cold-start situation, however the increase is still higher for ItemKNN than UserKNN. UserKNN has a low increase here, 2,5% and 5% for the dataset with respect to RMSE, which indicates that increasing the amount of ratings does not increase the accuracy significantly for this algorithm in this study. The dataset MovieLens 100K received higher RMSE values in

this interval opposed to the extremely-cold start scenario where the dataset received lower values.

5. Discussion

5.1 Analysis of the results

In this section the results will be discussed including method analysis.

Both ItemKNN and UserKNN are affected by the various simulated cold-start scenarios by an increase in RMSE and MAE. ItemKNN is affected noticeably more than UserKNN in the simulated scenarios, which confirms our hypothesis. UserKNN showed a more stable increase in RMSE as the amount of ratings increased than ItemKNN relative to the normal case. This indicates that UserKNN is more stable for extremely cold-start environments when only considering the prediction quality.

Considering the difference of performance in accuracy between RMSE and MAE, you can see that the RMSE values for UserKNN is lower than MAE while the relation is reversed for ItemKNN with RMSE values higher than MAE.
As described in the background RMSE is affected more than MAE when large errors occurs. The fact that RMSE gives higher values than MAE for ItemKNN while the reverse relation is true for userKNN seems to suggest that ItemKNN suffers more from large errors. This is probably due to the fact that the amount of similar items that the user has rated is generally smaller than the amount of similar users which have rated the item. So the risk of not having any neighbours that are particularly similar, is bigger for the ItemKNN algorithm. This could naturally lead to a larger amount of very big errors.

One could assume that the algorithms would receive better quality in predictions in the larger dataset due to the larger amount of ratings. However, when looking at the results and comparing the algorithms to the normal case in the extremely-cold-start simulations (figures 5 and 6) it might look a little odd that the algorithms appear to be performing better on the smaller 100K dataset than the 1M dataset.

We believe that the reason for this is that the accuracy in the normal case is significantly

better for the larger dataset than the smaller one, for both algorithms. So in the extremely-cold-start scenario, where the accuracy is quite poor in both the small and large dataset, the difference to the normal case is relatively higher for the larger dataset. Keep in mind that this *does not* mean that the accuracy of the algorithms are worse on the larger dataset, it just means that they have a relatively higher difference to the normal case.

By using the mean for 1-6 ratings in MovieLens 1M, and 1-9 for MovieLens 100K both considering RMSE, the prediction accuracy was better. In the Surprise implementation this illustrates that the mean rating for an item is a superior way of predicting a movie than letting the algorithm ItemKNN predict the rating for a new user giving between these explicit ratings. UserKNN showed having a stable increase, predicting better than both ItemKNN and by using the mean rating.

5.2 Method criticism

In this research Surprise was used to perform the various algorithms. This limits all the results to their algorithm implementation, which could be different in other recommender system programs.

Another constraint is the dataset used. This study only focused on the dataset provided by MovieLens. A further approach would be to look at other datasets to see if the accuracy varies further than in this research.

6. Conclusions and future research

Recommender systems are a leading measure of predicting user behaviour in systems with extensive amount of information. The systems provide the applications in which they are used with valuable suggestions towards customers, increasing profit by the business. It helps the customer by contributing relevant products while simultaneously generating proceeds for the respective company. The system also needs to provide the newly added users with sufficient suggestions such that the user feel satisfied with the suggestions. One way of increasing the prediction accuracy is to urge the user to rate a number of movies, increasing the accuracy of the suggestions.

This study shows that UserKNN is a better performing algorithm than ItemKNN under cold-start conditions. It is also shown that both versions of the k-NN algorithm has a significant decrease in accuracy when cold-start is introduced. This is consistent with previous research as well as our assumptions based on the theory.

Increasing the number of initial ratings by cold-start users had a positive effect on the accuracy of the algorithm. This was shown to be the case both for UserKNN and ItemKNN.

Future research could consist of investigating the effect in cold-start scenarios for other item- and user-based algorithms and compare them. Improvements for ItemKNN could consist of using the mean rating for users with few ratings, which could be looked into. Future research could also include experimenting with other k values and investigate whether the result differs.

One thing to look into is whether various parameters such as gender, age or profession have an impact on the accuracy of the algorithms in a cold-start scenario. If, for example, children needs to explicitly rate fewer movies than older people to achieve good predictions.

7. References

[1] Lampropoulos AS, Tsihrintzis GA. Machine Learning Paradigms: Applications in Recommender Systems [Internet]. New York, USA: Springer International Publishing; 2015. [quoted 23 April 2017].

[2] CHAMSI ABU QUBA Rana, HASSAS Salima, FAYYAD Usama, CHAMSI Hammam, From a “Cold” to a “Warm” Start in Recommender systems
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6927067
[quoted 1 May 2017].

[3] Mi Zhang, Jie Tang, Xuchen Zhang, Xiangyang Xue
Addressing Cold Start in Recommender Systems: A Semi-supervised Co-training Algorithm, 2014. Gold Coast, Queensland, Australia.
[quoted 27 April 2017].

[4] Rajaraman A, Ullman JD. Mining of massive datasets. New York, USA: Cambridge University Press; 2012
[quoted 12 May 2017].

[5] Hongyuan Zha, Shuang-Hong Yang, Ke Zhou, 2011, Beijing, China.
Functional Matrix Factorizations for Cold-Start Recommendation.
[quoted 3 March 2017].

[6] [Conference] Combining Memory-Based and Model-Based Collaborative Filtering in Recommender System, Circuits, Communications and Systems, 2009. PACCS '09. Pacific-Asia Conference. IEEE. Collected from:
http://ieeexplore.ieee.org/document/5232419/?part=1
[quoted 12 May 2017].

[7] Hong Cheng, Xiao Wen, Yu Rong. 2014, Seoul, Korea
A Monte Carlo Algorithm for Cold Start Recommendation
http://wwwconference.org/proceedings/www2014/proceedings/p327.pdf
[quoted 15 Aril 2017].

[8] MovieLens [Internet]. Minnesota: GroupLens, University of Minnesota; 2003. [quoted 16 februari 2016]. Collected from: http://grouplens.org/datasets/movielens/

[9] [Internet] Surprise, 2015, basic k-NN.
http://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithms.knns.KNNBasic
[quoted 12 Mars 2017].

[10] Surprise. Surprise: A python library for Recommender Systems, ver. 1.0.2 [Program]. Publisher:Nicolas Hug. Licensed with BSD 3-clause license. [quoted 23 april 2017]. Collected from: https://github.com/NicolasHug/Surprise

[11] [Internet] Surprise, 2015, http://surprise.readthedocs.io/en/stable/similarities.html
[quoted 13 Mars 2017].

[12] Martin P. Robillard, Walid Maalej, Robert J Walker, Thomas Zimmermann. Recommendation Systems in Software Engineering. 2014
[quoted 9 May 2017].

[13] Yoshua Bengio, Yves Grandvalet. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. 2004
[quoted 1 May 2017].

[14] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001, HongKong Item-Based Collaborative Filtering Recommendation Algorithms.
[quoted 12 May 2017].

[15] Yehuda Koren. 2008. Las Vegas, Nevada. USA.
Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model.
[quoted 12 May 2017].

[16] Ted Hong, Dimitris Tsamis Stanford University. Unknown year.
Use of KNN for the Netflix Prize.
[Collected from:] http://cs229.stanford.edu/proj2006/HongTsamis-KNNForNetflix.pdf
[quoted 12 May 2017].

8. Attachments

8.1 100K MovieLens accuracy results

8.1.1 RMSE

#Ratings	itemKNN	userKNN
1	1,3542	1,1212
2	1,3129	1,0982
3	1,2740	1,0887
4	1,2400	1,0793
5	1,2068	1,0680
6	1,1789	1,0597
7	1,1603	1,0555
8	1,1457	1,0514
9	1,1322	1,0475

8.1.2 MAE

#Ratings	itemKNN	userKNN
1	1,0360	0,9444
2	1,0040	0,9107
3	0,9807	0,8867
4	0,9638	0,8663
5	0,9443	0,8514
6	0,9241	0,8431
7	0,9125	0,8389
8	0,9028	0,8359
9	0,8927	0,8330

8.2 1M MovieLens accuracy results

8.2.1 RMSE

#Ratings	ItemKNN	UserKNN
1	1,3747	1,1172
2	1,2882	1,0709
3	1,2231	1,0324
4	1,1780	1,0141
5	1,1455	0,9952
6	1,1239	0,9900
7	1,1067	0,9880
8	1,0924	0,9851
9	1,0814	0,9820

8.2.2 MAE

#Ratings	itemKNN	userKNN
1	1,0315	0,9339
2	0,9856	0,8652
3	0,9484	0,8272
4	0,9194	0,8107
5	0,8973	0,7947
6	0,8826	0,7907
7	0,8702	0,7877
8	0,8602	0,7855
9	0,8531	0,7840

8.3 Program code

8.3.1 Program to generate cold start training and test files (Java)

```
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

class RatingEntry {
  int userId;
  int itemId;
  double rating;
  String rowString;

  RatingEntry(String row) {
    this.rowString = row;
    String[] parts = row.split("::");
    this.userId = Integer.parseInt(parts[0]);
    this.itemId = Integer.parseInt(parts[1]);
    this.rating = Double.parseDouble(parts[2]);
  }
}

class User {
  int id;
  List<RatingEntry> ratingEntries = new ArrayList<>();

  User(int id, List<RatingEntry> ratingEntries) {
    this.id = id;
    this.ratingEntries = ratingEntries;
  }
}

class Tuple<T, U> {
  T first; U second;
  Tuple(T fst, U snd) { this.first = fst; this.second = snd; }
}

/**
 * Program to generate training and test files that simulates the cold start
problem.
 * Created by oskarek on 2017-05-02.
 */
public class ColdStartGenerator {
  public static void main(String[] args) {
    ArrayList<User> users = usersFromDataFile("/ml-1m/ratings.dat");
```

```
    createTrainingAndTestFiles(users, 4);
  }

  /** Get a list of users from a ratings file. */
  static ArrayList<User> usersFromDataFile(String path) {
    try (Stream<String> lines = Files.lines(Paths.get(path))) {
      ArrayList<User> users = lines
          .map(RatingEntry::new)
          .sorted((entry1, entry2) -> entry2.userId - entry1.userId)
          .collect(Collectors.groupingBy(entry -> entry.userId))
          .entrySet().stream()
          .map(entry -> new User(entry.getKey(), entry.getValue()))
          .collect(Collectors.toCollection(ArrayList::new));
      Collections.shuffle(users);
      return users;
    } catch (IOException e) {
      e.printStackTrace();
      return null;
    }
  }

  /** Create the training and test files given a list of users, and a given
number of folds to perform */
  static void createTrainingAndTestFiles(ArrayList<User> users, int nFolds) {
    int userCount = users.size(), testSize = userCount/nFolds;
    for (int fold = 0; fold < nFolds; fold++) {
      int testStart = fold * testSize, testEnd = testStart + testSize;
      for (int n = 0; n <= 9; n++) {
        File training_file = new File("fold" + (fold+1) + "/training_" + n +
"_ratings.txt");
        File test_file = new File("fold" + (fold+1) + "/test_" + n +
"_ratings.txt");
        training_file.getParentFile().mkdirs();
        test_file.getParentFile().mkdirs();
        try (PrintWriter training_pw = new PrintWriter(new
FileWriter(training_file));
             PrintWriter test_pw = new PrintWriter(new FileWriter(test_file))) {
          for (int i = 0; i < userCount; i++) {
            List<RatingEntry> entries = users.get(i).ratingEntries;
            if (i >= testStart && i < testEnd) {
              Tuple<List<RatingEntry>, List<RatingEntry>> splitEntries =
splitList(entries, n);
              splitEntries.first.forEach(entry ->
training_pw.write(entry.rowString+"\n"));
              splitEntries.second.forEach(entry ->
test_pw.write(entry.rowString+"\n"));
            } else {
              entries.forEach(entry -> training_pw.write(entry.rowString+"\n"));
            }
          }
```

```
        } catch (IOException e) {
          e.printStackTrace();
          return;
        }
      }
    }
  }

  /** Split a list in two at a given index */
  private static <T> Tuple<List<T>, List<T>> splitList(List<T> list, int
splitIndex) {
    List<T> first = list.subList(0, splitIndex);
    List<T> second = list.subList(splitIndex, list.size());
    return new Tuple<>(first, second);
  }
}
```

8.3.2 Program to test the algorithms under cold start (Python)

```
from surprise import KNNBasic
from surprise import Dataset, Reader
from surprise.accuracy import rmse, mae

# Open the files to print the results to
rmse_file = open('rmse.txt', 'w', 1)
mae_file = open('mae.txt', 'w', 1)

reader = Reader(line_format='user item rating timestamp', sep='::')

for n in range(0,10):
    folds_files = []
    for fold in range(1,5):
        train_file = 'fold'+str(fold)+'/training_'+str(n)+'_ratings.txt'
        test_file = 'fold'+str(fold)+'/test_'+str(n)+'_ratings.txt'
        folds_files.append((train_file, test_file))

    # Load the dataset
    data = Dataset.load_from_folds(folds_files, reader=reader)

    # Build an algorithm, and train it.
    algo = KNNBasic(sim_options={'name':'pearson', 'user_based':False})

    # Evaluate performances of the algorithm in terms of RMSE and MAE,
    # and print the results to the corresponding files.
    rmseValues = []
    maeValues = []
    for trainset, testset in data.folds():
        algo.train(trainset)
        predictions = algo.test(testset)
```

```
        rmseValues.append(rmse(predictions))
        maeValues.append(mae(predictions))

    # Print average values to file
    avgRMSE = sum(rmseValues)/len(rmseValues)
    rmse_file.write(str(avgRMSE) + '\n')
    avgMAE = sum(maeValues)/len(maeValues)
    mae_file.write(str(avgMAE) + '\n')
```

8.3.3 Program to test the algorithms under the normal case (Python)

```
from surprise import KNNBasic
from surprise import Dataset, Reader
from surprise import evaluate, print_perf

reader = Reader(line_format='user item rating timestamp', sep='::')

# Load the dataset and split it into 4 folds for cross-validation.
data = Dataset.load_from_file('/ml-1m/ratings.dat', reader=reader)
data.split(n_folds=4)

# Build an algorithm, and train it.
algo = KNNBasic(sim_options={'name':'pearson', 'user_based':False})

# Evaluate performances of our algorithm on the dataset; RMSE and MAE.
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])

print_perf(perf)
```