An Initial Investigation of Multi-Cyclic Training Regimen for Collaborative Filtering Models in GraphChi

James W. Curnalia and Alina Lazar
Department of Computer Science and Information Systems
Youngstown State University
Youngstown, Ohio 44555
jwcurnalia@student.usu.edu
alazar@ysu.edu

Abstract

More and more outlets are utilizing collaborative filtering techniques to make sense of the sea of data generated by our hyper-connected world. How a collaborative filtering model is generated can be the difference between accurate or flawed predictions. This study is to determine the impact of a cyclical training regimen on the algorithms presented in the Collaborative Filtering Toolkit for GraphChi.

Introduction to Collaborative Filtering

With the growth of on-line services and e-commerce the amount of data that users need to make sense of has increased exponentially. The difficult task facing these service providers is to sift through a sea of options and find what a user wants before their competitors. One of the tools they have at their disposal is collaborative filtering.

One of the central assumptions about collaborative filtering is that people with similar tastes, or shopping histories, are better predictors of a user's future behavior than a random person. Collaborative filtering techniques sift through large datasets to identify patterns and similarities between these users, and then make recommendations. These processes are not limited to the retail sphere, collaborative filtering has also been applied to financial, geological, and other endeavors. An examination of a small example of collaborative filtering, to produce song recommendations for customers, will help illustrate some of these ideas.

Table 1: Example User Song History and Ratings

User	song1	song2	song3	song4	song5	song6	song7
A	3	X	X	4	1	5	2
B	1	2	X	2	4	X	4
C	5	X	4	X	1	4	X
D	2	X	1	X	5	X	5

The example data covers four different customers (A-D) and their listening history over a catalog of seven songs. In addition to the fact that a user has listened to a particular song, there is an explicit rating (1-5, 5 being the highest). If a user has not listened to a song this fact is highlighted with an 'X'. A predicted rating needs to be generated for these unknown songs, so that they can be suggested to the appropriate users.

The first step in the process is to identify which users have similar tastes. User A and User C have three points of similarity. They both responded very positively to song6 (ratings of A:5 and C:4) and very negatively to song5 (both had ratings of 1). The third point of similarity, song1, is less significant. User C responded very favorably to the song, while User A had a neutral reaction. A similar process can generate a taste profile for Users B and D (song1 B:1/D:2, song5 B:4/D:5, and song7 B:4/D:5). Using these pairings we can begin to identify likely reactions to some of the users' unknown songs.

Table 2: Suggestions Based on Similarities

User	song1	song2	song3	song4	song5	song6	song7
A	3	X	+	4	1	5	2
B	1	2	-	2	4	X	4
C	5	X	4	+	1	4	-
D	2	-	1	-	5	X	5

Six of the unknown songs have been replaced with a suggested reaction (+ for positive or – for negative). An exact score would require further analysis, and most likely a great deal more information, but a more simplistic measure of attitude can be inferred.

This still leaves four unknown songs in the table. A quick look over the table shows that the two pairs of users have completely opposite tastes in music. It would be reasonable to assume that what one pair likes the other will dislike, and vice versa. In this way we can utilize not only the similarity between users to determine attitudes, but also the reactions of diametrically opposed Users.

Now all that remains is to scale the entire process to handle millions of users and products, while maintaining

accuracy and minimizing costs (both financial and temporal).

Related Work

Collaborative Filtering has traditionally included methods such as Bayesian Networks and clustering. A Bayesian Network uses probability and causal relationships to classify new observations (Pearl 1994). Clustering algorithms attempt to represent observations as "points" in a multi-dimensional space. The closer together that two points are, the more similar the underlying observations (Witten, Frank and Hall 2011). These and other traditional methods are, and will continue to be, powerful and valid collaborative filtering methods.

However, in the wake of events like the Netflix Prize a new series of algorithms were developed to deal with a new phenomenon "Big Data". These new algorithms sought to incorporate the concepts of the traditional methods into new frameworks capable of dealing with data that was increasingly large, complex, and often extremely sparse.

This study will concern itself with thirteen of these modern Collaborative Filtering algorithms:

- Alternating Least Squares (ALS), (Zhou et al. 2008)
- Stochastic Gradient Descent (SGD), (Koren 2009)
- Bias Stochastic Gradient Descent (BSGD), (Koren 2008)
- Koren's Singular Value Decomposition (SVD++), (Koren 2008)
- Weighted ALS (WALS), (Hu, Koren, and Volinsky 2008)
- Non-Negative Matrix Factorization (NMF), (Lee and Seung 2001)
- Singular Value Decomposition (SVD), (Hernandez, Roman, and Tomas 2007)
- One-Sided SVD, (Hernandez, Roman, and Tomas 2007)
- Tensor ALS (TALS), (Comon, Luciani, and de Almeida 2009)
- Restricted Bolzman Machines (RBM), (Hinton 2010)
- Time-SVD++(TSVD++), (Koren 2009)
- libFM, (Rendle 2010)
- Probabilistic Matrix Factorization (PMF), (Salakhutdinov and Mnih 2008)

Each of the modern Collaborative Filtering algorithms seeks to either reduce or capitalize on the complexity of large datasets. Algorithms utilizing decomposition, factorization, and SGD seek to reduce the dimensionality of data in order to expose underlying relationships. Least squares methods treat recommendations as linear equations, and attempt to find the best estimation of the parameters necessary to calculate an accurate rating. TALS and TSVD++ try to leverage additional information, in this case time, in order to more accurately model behaviors.

This study will focus on graph-based implementations of these algorithms given their recent popularity and the ability to be executed on smaller machines. Recently graph-based algorithms have been adopted by many large, commercial websites including Amazon and YouTube (Walia 2008). While it is important to note the adoption of techniques like this by powerful and influential corporations, it is admittedly the latter that was the driving force in adopting a graph-based approach.

The GraphLab Project was developed in order to facilitate distributed, parallel, graph-based algorithms in an efficient and reliable manner (Low et al. 2010). GraphChi is an offshoot of the GraphLab Project that seeks to leverage the power graph-based algorithms on a single machine, while maintaining high performance standards (Kyrola, Blelloch, and Guestrin 2012). Bickson, one of the original developers of GraphLab, has ported a number of collaborative filtering algorithms from GraphLab to GraphChi in the form of the Collaborative Filtering Toolkit (CFT). Thirteen algorithms were supported at the time this study was run, December 2012, but two additional algorithms had already been added as this paper was being written, January 2013.

In addition to developing the toolkit, Bickson has written a blog entry which serves as an introduction to the CFT and its underlying algorithms (Bickson 2012). In the tutorial, Bickson identifies a number of algorithms which have an element of fault tolerance. These algorithms allow the user to save the model to disk and then resume training from that exact state.

Experimentation with the fault-tolerant algorithms seemed to yield an additional benefit, in that the accuracy of the model would often jump between executions of the training epochs (this paper will use the terms "epoch" and "cycle" interchangeably to refer to a group of training iterations). This would seem to suggest that there is an advantage to using multiple cycles beyond simple fault tolerance. The cumulative value of this inter-cycle bump in accuracy could be quite significant.

As these algorithms train a model, they attempt to reduce the sample space in an attempt to converge on an optimal answer. Given that these algorithms become more restrictive and focused the longer that they run, it is reasonable to assume that restarting an algorithm would have a significant positive impact on the final model. By loosening the bounds placed on the algorithm it is possible to identify the possibility that the current parameters have focused on a local rather than global minimum.

Proposal

The observed increase in inter-epoch accuracy of models being trained with fault tolerant algorithms suggests that there is significant value to be gained from such a training regimen. This study will first seek to establish whether there is in fact a boost in model accuracy between training cycles. If this is successful, we will attempt to identify a cycle size that results in the best final model.

It is our belief that a fault-tolerant algorithm, utilizing a epoch size of no more than 20, should result in a model

that is significantly more accurate than those trained using a single-epoch algorithm.

Methodology

Since this study is seeking to understand the impact of epoch size on the accuracy of the resulting model, there will be no attempt to tune any of the algorithms. The default settings were used in order to limit the impact of user proficiency on the resulting models (Bickson 2012).

The CFT reports the accuracy of the model generated using root mean squared error (RMSE). This statistic is generated for every complete pass over the training set, and is reported in terms of both training and validation. The training RMSE will not be used in this study as it is only a reflection of how well the model performs on the training data. Instead the validation RMSE will be the only measure reported since it demonstrates how well the model handled the test data. Additionally, the validation RMSE will identify problems with the model such as overfitting which are ignored in the training algorithm.

Fault Tolerance Algorithms (ALS, WALS, TALS, NMF, SGD, BSGD and SVD++)

The initial epoch size will be set to the maximum iterations specified in the tutorial, usually 6 iterations. After the first training cycle, the argument --load_factors_from_file=1 will be added to the algorithm to resume training with the current state. Training cycles will continue until one of the following three conditions are met:

1. The validation RMSE no longer improves with subsequent training cycles (a minima is reached).
2. The validation RMSE increases with additional training cycles (overfitting).
3. Multiple training cycles result in an improvement of the validation RMSE of less than .00005 / 10 iterations (diminishing returns).

Upon reaching one of the above criteria, the current state of the model will be saved and the validation RMSE and total number of training cycles recorded.

Next the epoch size will be increased and the entire process will repeat for the new model. The training epoch will be increased on the following schedule: 6, 20, 40, 80, 100, 120, 140, 180, 200. After 200 iterations the size of the epoch will be incremented by 50 for every subsequent increase. The training of new models on this schedule will continue until the resulting model has a higher validation RMSE than the previously generated model.

After the complete training of an algorithm is completed, an average starting RMSE is selected and recorded as well. The starting RMSE is defined as the validation RMSE after a single iteration of the algorithm. Since there is a degree of variation inherent in all of these algorithms it is necessary to choose a representative initial state. The starting RMSE will allow for a model's training regimen to be judged both by its final accuracy and the degree of accuracy that is a result of training.

Remaining Algorithms (SVD, One-Sided SVD, RBM, TSVD++, libFM, and PMF)

Even though these algorithms do not support a multi-cyclic training regimen, they will be trained on the same epoch schedule to provide additional context.

Additional Note on Alternating Least Squares (ALS) Algorithms

Overfitting was a significant problem with the ALS algorithms, and as such some modifications to the methods were made for those algorithms. Instead of stopping the training schedule with the first model resulting in a higher RMSE, all schedules were run to at least a training epoch of 80 iterations in size. This was an attempt to see if overfitting could be overcome with additional training. Overfitting also led to the inclusion of two smaller epoch sizes on the training schedule, two and ten, in order to identify if the optimal size was located at this smaller scale.

Control Groups

One final piece of information is necessary to determine the effectiveness of a multi-cyclic training regimen, a control group. The control group was trained using a single-epoch consisting of a large number of iterations. The control groups were generated after the experimental group in an attempt to limit the number of unnecessary training cycles, since large epochs require 8+ hours to run.

Waiting until after the experiment had the additional benefit of identifying a number of algorithms which did not need to included in the control group. All of the algorithms utilizing ALS already had runs which demonstrated that even moderately sized single iterations were outperformed by multi-cyclic regimens. This meant that it was only necessary to run control groups on SGD, BSGD, NMF, and SVD++.

Each of the control groups is trained using the same parameters utilized in the experimental phase, only the progression of epoch size is altered. A starting size of 200 iterations was selected since it is within the bounds of each of the selected algorithms total number of iterations run from the experiment. Each succeeding epoch will be doubled in size up to 1,600 iterations. The next epoch will be increased to 2,000 and then incremented by 1,000 every epoch after that. Training will continue until the RMSE no longer decreases between runs, and actually begins to increase.

Table 2: Netflix Control Group Results

Algorithm	# of Iterations	RMSE
SGD	200	1.123820
BSGD	400	1.117690
SVD++	400	0.982024
NMF	2000	2.370640

The Data

This study uses the same dataset featured in Bickson's blog, which is a synthetic Netflix dataset created using an anonymized sample of the original. Although the dataset from the Netflix challenge is unavailable because of copyright, the general characteristics of the data are well established by the competition's creators (Bennett and Lanning 2007). The GraphLab Netflix sample was done to preserve these characteristics (i.e. sparsity of data, user to movie ratio, user to rating ratio, etc) while ensuring the anonymity of the users.

The Netflix subset has the following general characteristics: 95,526 unique users, 3,561 movies, and 3,298,163 ratings (non-zeroes). This is a very sparse dataset with less than 0.97% of the resulting matrix having ratings.

Historic Benchmarks

In order to provide some context in which to view the results of this study the following results from the Netflix Prize (Netflix 2009) were retrieved:

- Cinematch (2006) : RMSE 0.9525
- 2007 Progress Prize : "KorBell" : RMSE 0.8723
- 2008 Progress Prize : "BellKor in BigChaos" : RMSE 0.8627
- Winners : "BellKor's Pragmatic Chaos" (2009) : RMSE 0.8567

Additionally, KorBell reported that the best result they could get from a single method was an increase of 6.57% over Cinematch, RMSE ~0.8882 (Bell and Koren 2007).

Results

The initial impressions of the CFT suggested that the apparent boost in accuracy between training epochs would favor a training regimen consisting of a large number of very small cycles (no more than 20 iterations per epoch). This type of training would lead to results superior, to those generated without it.

The first thing of note about Table 3 is that only eleven algorithms are included. Both SVD and One-Sided SVD have been left off of the table intentionally. No modification to the number of training iterations yielded any variation in how these algorithms performed. Every run of these methods resulted in both an identical process and model. Additionally, these algorithms utilize a different error metric than the rest of the CFT by reporting an error estimate for each of the features generated. For these reasons these algorithms were left out of the rest of the discussion of this study's results, but an example of the output of each has been included at the end of this paper as Appendix 1.

*Table 3: Algorithms Ordered by Final RMSE (Multi-Cyclic Algorithms **Bolded**)*

Algorithm	Initial RMSE	Final RMSE
PMF	2.498400	0.914566
RBM	0.979169	0.926279
SVD++	**1.124420**	**0.931921**
BSGD	**1.363540**	**0.952970**
SGD	**1.240700**	**0.959890**
TSVD++	1.041220	0.995435
LibFM	1.090030	1.025770
TALS	**1.244250**	**1.147030**
ALS	**1.251550**	**1.159920**
NMF	**1.580120**	**2.375430**
WALS	**5.522080**	**5.325280**

Looking at the results purely in terms of accuracy suggest that the fault-tolerant algorithms are generally of inferior quality. However, this view of the results is misleading. There was no effort made to tune these algorithms, or to even check if there current settings were conducive to producing good models, before these results were generated. So while it is interesting which algorithms handled the data best, it doesn't really show how well these models developed over the course of training.

By ordering the results by how much a model's RM*Figure 1: Trendlines of Algorithm Performance*SE was improved over the course of training reveals a far different picture of the fault-tolerant algorithms. The improvement in final RMSE would suggest that these algorithms are more effective at training, but it is unclear whether this is a product of the algorithm itself or the training regimen. A closer look at all of the results, as well as the effect of the restart boost, should provide a clearer picture of the factors at work.

Table 4: Results Ordered by Percent Improved Over Initial RMSE

Algorithm	% Improvement
PMF	63.39%
BSGD	**30.11%**
SGD	**22.63%**
SVD++	**17.12%**
TALS	**7.81%**
ALS	**7.32%**
LibFM	5.90%
RBM	5.40%
TSVD++	4.40%
WALS	**3.56%**
NMF	**-50.33%**

Figure 1: Algorithmic Spark Lines

Figure 1 shows the percentage difference between all of the training regimens for a given algorithm and its starting RMSE. The spark lines illustrate that while the effectiveness of the algorithms may vary, their general training behaviors are very similar. From this limited sample it would seem that there is no evidence that the restart boost creates a more effective training regimen. But this is not evidence that it has no effect.

While the cyclical training regimen fails to outperform the single epoch algorithms, it is clearly not without its benefits. Most of the cyclically generated models have significantly higher accuracy than models learned over the course of a lone epoch with the same algorithm. All of the algorithms in the CFT suffer from the same design flaw, they fail to take into account validation RMSE. This results in either overfitting or the algorithm becoming trapped in a local minima, as the parameters of the algorithms become more and more restrictive. Restarting the algorithm loosens the bounds on the program allowing it to move beyond erroneous assumptions about the data. So while the hypothesized accuracy failed to materialize, there are definitely significant advantages to this style of training with SGD, BSGD, and SVD++.

Table 9: Final RMSE of Control and Experimental Models

Algorithm	Control RMSE	Test RMSE	Improvement
SGD	1.123820	0.959890	14.59%
BSGD	1.117690	0.952970	14.74%
SVD++	0.982024	0.931921	5.10%
NMF	2.370640	2.375430	-0.20%

*Table 5: Epoch Size and Number of Cycles Trained for Each Algorithm (Multi-Cyclic Algorithms **Bolded**)*

Algorithm	Epoch Size (Iterations)	Training Cycles	Total Running Time (sec)
PMF	180	-	2,501.9900
RBM	80	-	692.3140
SVD++	**40**	**7**	**691.6511**
BSGD	**40**	**92**	**6,860.8908**
SGD	**40**	**59**	**3127.5310**
TSVD++	80	-	251.9750
LibFM	200	-	908.0320
TALS	**10**	**1**	**74.6055**
ALS	**2**	**4**	**47.4264**
NMF	**40**	**56**	**9,045.0080**
WALS	**10**	**1**	**66.1071**

The idea that a cyclical training regimen is superior to a single epoch system has been thoroughly disproved, but what about the ideal size for these epochs? Table 5 shows, rather conclusively, that the ideal number of iterations is larger than the 20 iteration ceiling that had been hypothesized. Each cycle needs to be large enough to take advantage of as many positive iterations (those that reduce the validation RMSE), while minimizing the number of overfitted iterations (a common occurrence in later training cycles).

Conclusion and Further Study

This study is only meant to serve as an initial foray into the algorithms represented by the CFT, and a great

deal remains to be researched. However, there are questions that relate directly to the results of this study which would be logical next steps. The first is to determine if the results here are a product of the algorithms or the data. This study needs to be repeated on a number of other datasets to see if similar results are generated. Similar training patterns would suggest that the findings of this study are a result of the algorithms used and not some feature of the dataset. Secondly, a study should determine if tuning the algorithms have an effect on training patterns.

The CFT represents a powerful tool for bringing compact, easily executed data analysis to a variety of ventures. Further experimentation with the package will illuminate how it, and the algorithms it represents, can be put to the best use.

Appendix 1: SVD and One-Sided SVD Output

```
./toolkits/collaborative_filtering/svd
--training=smallnetflix_mm --nsv=3 --nv=10 --max_iter=5
--quiet=1 --tol=1e-1
WARNING:  common.hpp(print_copyright:104):
GraphChi Collaborative filtering library is written by
Danny Bickson (c). Send any  comments or bug reports to
danny.bickson@gmail.com
[training] => [smallnetflix_mm]
[nsv] => [3]
[nv] => [10]
[max_iter] => [5]
[quiet] => [1]
[tol] => [1e-1]
Load matrix smallnetflix_mm
Starting iteration: 1 at time: 2.71863
Starting step: 1 at time: 3.42417
Starting step: 2 at time: 4.6307
Starting step: 3 at time: 5.88049
Starting step: 4 at time: 7.11613
Starting step: 5 at time: 8.36737
Starting step: 6 at time: 9.63118
Starting step: 7 at time: 10.9138
Starting step: 8 at time: 12.2115
Starting step: 9 at time: 13.5747
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
Number of computed signular values 5
Singular value 0     3276.69 Error estimate:   0.000305186
Singular value 1     1064.07 Error estimate:   1.18507e-13
Singular value 2     956.541 Error estimate:   0.00162432
Singular value 3     889.028 Error estimate:   0.00841469
Singular value 4      710.42 Error estimate:    0.0551811
Going to save output vectors U and V
Lanczos finished 22.5142
```

```
./toolkits/collaborative_filtering/svd_onesided
--training=smallnetflix_mm --nsv=3 --nv=10 --max_iter=5
--quiet=1 --tol=1e-1
WARNING:  common.hpp(print_copyright:104):
GraphChi Collaborative filtering library is written by
Danny Bickson (c). Send any  comments or bug reports to
danny.bickson@gmail.com
[training] => [smallnetflix_mm]
[nsv] => [3]
[nv] => [10]
[max_iter] => [5]
[quiet] => [1]
[tol] => [1e-1]
Load matrix smallnetflix_mm
Starting iteration: 1 at time: 0.560262
Starting step: 1 at time: 1.22546
Starting step: 2 at time: 4.76345
Starting step: 3 at time: 8.32286
Starting step: 4 at time: 11.8127
Starting step: 5 at time: 15.4576
Starting step: 6 at time: 19.1342
Starting step: 7 at time: 22.8375
Starting step: 8 at time: 26.5084
Starting step: 9 at time: 30.1965
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
Number of computed signular values 5
Singular value 0     3276.69 Error estimate:   3.8991e-14
Singular value 1     1064.07 Error estimate:   0.00146017
Singular value 2     956.541 Error estimate:   0.00782078
Singular value 3     889.028 Error estimate:    0.0440951
Singular value 4      710.42 Error estimate:    0.0864268
Lanczos finished in 47.1228
```

References

Bell, R. M.; Koren, Y. (2007). “Lessons from the Netflix Prize Challenge.” SIGKDD Explorations Vol. 9 Issue 2, 75-79. ACM New York, NY.

Bennett, J.; Lanning, S. (2007). “The Netflix Prize”. In Proceedings of KDD Cup and Workshop 2007, 3-6. ACM New York, NY.

Bickson, D. (2012). “Collaborative filtering with GraphChi”. Large Scale Machine Learning and Other Animals. Accessed on December 5, 2012. http://bickson.blogspot.com/2012/08/collaborative-filtering-with-demographical

Comon, P.; Luciani, X.; de Almeida, A. L. F. (2009). “Tensor Decompositions, Alternating Least Squares and other Tales.” Special issue, Journal of Chemometrics. In memory of R. Harshman.

Hernandez, V.; Roman, J. E.; Tomas, A. (2007). “Restarted Lanczos Bidiagonalization for the SVD in SLEPc.” SLEPc Technical Report STR-8.

Hinton, G. (2010).” A Practical Guide to Training Restricted Boltzmann Machines.” University of Toronto Tech Report UTML TR 2010-003.

Hu, Y.; Koren, Y.; Volinsky, C (2008). “Collaborative Filtering for Implicit Feedback Datasets”. IEEE International Conference on Data Mining (ICDM 2008), IEEE Washington, DC.

Koren, Y. (2009). “Collaborative filtering with temporal dynamics.” In Proceedings of the 15th ACM SIGKDD, 447-456. ACM, New York, NY.

Koren, Y (2008). “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”. ACM SIGKDD. ACM, New York, NY.

Koren, Y.; Bell, R.; Volinsky, C (2009). “Matrix Factorization Techniques for Recommender Systems.” In IEEE Computer, Vol. 42, No. 8. (07 August 2009), pp. 30-37. IEEE Washington, DC.

Kyrola, A.; Blelloch, G.; Guestrin, C. (2012). “GraphChi: Large-Scale Graph Computation on Just a PC.” In Proceedings of the Tenth USENIX Symposium on Operating Systems Design and Implementation, 31-46. OSDI Press Hollywood, CA.

Lee, D..D.; Seung, H.S. (2001). “Algorithms for Non-negative Matrix Factorization”, Advances in Neural Information Processing Systems 13, 556-562.

Low, Y.; Gonzalez, J.; Kyrola, A.; Bickson, D.; Guestrin, C.; Hellerstein,J. M. (2010). "GraphLab: A New Parallel Framework for Machine Learning." In Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence. AAAI Press Palo Alto, CA.

Netflix (2009). “Leaderboard”. NetflixPrize.com. Accessed on January 5, 2013. http://www.netflixprize.com//leaderboard

Pearl, J. (1994). "A Probabilistic Calculus of Actions". In UAI'94 Proceedings of the Tenth international conference on Uncertainty in artificial intelligence. Morgan Kaufman San Mateo CA. pp. 454–462.

Rendle, S. (2010). “Factorization Machines.” in Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010), Sydney, Australia. IEEE Washington, DC.

Salakhutdinov, R.; Mnih, A. (2008). “Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo.” In the Proceedings of the International Conference on Machine Learning.

Walia, R.R. (2008). Collaborative Filtering: A comparison of graph-based semi-supervised learning methods and memory-based methods. Strengthening the Role of ICT in Development, 70-82.

Witten, I.H.; Frank, E.; Hall, M.A. (2011). Data Mining: Practical Machine Learning Tools and Techniques 3rd ed. Morgan Kaufman San Mateo CA.

Zhou, Y.; Wilkinson, D.; Schreiber, R.; Pan, R. (2008). “Large-Scale Parallel Collaborative Filtering for the Netflix Prize.” In the Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management, 337-348. Shanghai, China.