Recommender: An Analysis of Collaborative Filtering Techniques

Christopher R. Aberger
caberger@stanford.edu

ABSTRACT

Collaborative filtering is one of the most widely researched and implemented recommendation algorithms. Collaborative filtering is simply a mechanism to filter massive amounts of data based upon a previous interactions of a large number of users. In this project I analyze and benchmark several collaborative filtering implementations in PowerGraph, an advanced machine learning framework, across a variety of different sparse datasets. My goal is to empirically discover the performance and algorithmic tradeoffs of state-of-the-art techniques for collaborative filtering on modern commodity multi-socket, multi-core, non-uniform memory access (NUMA) hardware.

1. INTRODUCTION

Recommendation systems are composed of filtering algorithms that aim to predict a rating or preference a user would assign to a given item. Recommender systems have become increasingly important across a variety of commercial domains including movies (Netflix), restaurants (Yelp), friends (Facebook and Twitter), and music (Pandora). These systems generally produce recommendations via one of two methods: 1) content based filtering or 2) collaborative based filtering. Content based filtering techniques use attributes of an item in order to recommend future items with similar attributes. Collaborative filtering builds a model from a user's past behavior, activities, or preferences and makes recommendations to the user based upon similarities to other users [15]. Other work has aimed at creating hybrid systems which use a mix of content and collaborative filtering based approaches [13].

While both content and collaborative filtering approaches have their respective strengths and weaknesses collaborative filtering's main advantage is that it is capable of recommending complex items without a prori knowledge about the item [15]. Therefore collaborative filtering is domain free while generally providing more accuracy than content-based techniques [9]. Because of this collaborative filtering systems have been widely used in machine learning research and practice.

Collaborative filtering algorithms typically suffer from three main issues:

Cold Start A large amount of existing data is necessary to make accurate recommendations for a given user.

Scalability In the era of big data many systems need to make recommendations on datasets with millions of users and products. Because of this a large amount of computational power is necessary to compute timely recommendations.

Sparsity The number of items typically far exceeds the number of users making our relations extremely sparse as most active users will have only rated a small subset of the total items.

In this project I analyze and benchmark the performance of several state-of-the-art collaborative filtering algorithms in the PowerGraph framework in order to identify the different implementations of collaborative filtering that best mitigate the aforementioned scalability and sparsity problems [5]. Here I run a variety of tuned collaborative filtering algorithms *at scale* on modern commodity multi-socket, multi-core, non-uniform memory access (NUMA) hardware. I benchmark and analyze parallel matrix factorization collaborative filtering implementations in the PowerGraph framework on the datasets in Table 1. The matrix factorization algorithms benchmarked are the following:

- Stochastic Gradient Descent (SGD)
- Bias Stochastic Gradient Descent (B-SGD)
- Alternating Least Squares (ALS)
- Weighted Alternating Least Squares (W-ALS)

2. COLLABORATIVE FILTERING PROBLEM

Collaborative filtering identifies patterns of user intersets to make targeted recommendations. Collaborative filtering breaks down into two primary approaches memory-based and model-based approaches.[1] Each approach is described briefly in the following sections with a more in-depth explanation of the model-based approaches benchmarked in this paper following in Section 3.

2.1 Memory-Based

The memory-based approach takes user rating data to compute similarities between users and items in order to make a recommendation. The most famous memory-based approach are neighbohood-based algorithms. Neighborhod-based algorithms focus on computing the relationship between either items or users. Here a recommendation for a user is predicted based upon ratings of similar (or neighboring) items by the same user. The neighborhood-based algorithm calculates the similarity between two users or items then producing a prediction for the user by taking the weighted average of all ratings. A popular way to compute the similarity between two users x and y is to use the cosine similarity where I_{xy} is the set of items rated by both users x and y:

[1]Hybrid approaches between memory-based and model-based collaborative filtering algorithms are often used in practice. Due to the complexity of developing hybrid solutions they are outside the scope of this project.

Dataset	Number of Ratings	Number of Users	Number of Items	Description
Amazon [1]	5,838,041	2,146,057	1,230,915	Product ratings from Amazon.
BookCrossing [2]	433,652	77,802	185,955	Book ratings.
Epinions [3]	13,668,320	120,492	755,760	Epinion product ratings.
MovieLens [4]	10,000,054	69,878	10,677	Movie ratings.

Table 1: Rating datasets used in the experiments.

$$similarity(x,y) = cos(\vec{x},\vec{y}) = \frac{\sum_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_x} r_{x,i}^2} \sqrt{\sum_{i \in I_y} r_{y,i}^2}}$$

After identifying the top-K most similar users to an active user the corresponding user-item matrices are aggregated to choose the set of items to be recommended.[2] This approach is commonly called the K-nearest neighbor (KNN) algorithm. While it is relatively simple to compose and explain this algorithms performance degrades as the data becomes increasingly sparse and does not typically work well on large datasets, which occur frequently in practice [15].

2.2 Model-Based

Model-based (or matrix factorization based) methods build models based on modern machine learning algorithms discovering patterns in the training data. The models are then used to make predictions on real data. Model-based approaches uncover latent factors which can be used to construct the training data ratings. Model-based methods have become widely popular recently as they handle sparsity better than their memory-based counterparts while improving prediction accuracy [15].

Model-based methods are often classified as latent factor models. Here the ratings are explained by characterizing both items and users on a number of factors inferred from the ratings patterns [9]. Most algorithms take a rating matrix, which is extremely sparse and build a linear model finding two low dimensional matrices. Formally let $R = \{r_{ij}\}_{n_u \times n_v}$ be a user-item matrix where each item R_{ij} represents the rating score of item j by user i with the value being either a real number or missing [17]. Here n_u designates the number of users and n_v designates the number of items. In this setup, collaborative filtering is designed to estimate missing values in R based upon the known values.

The collaborative filtering problem typically starts with a low-rank approximation of the user-item matrix R and then models both users and movies by giving them coordinates in a low dimensional feature space. Both the users and items have individual feature vectors where the rating of an item by a user is modeled as the inner product of the desired user and movie feature vectors. Let U represent the user feature matrix and V represent the item feature matrix composed of both user and item feature vectors respectively. A visualization of this is given in Figure 1 where the items are movies and d is the number of features. The dimension of the feature space, or d, is a system parameter that is determined by a hold-out dataset or cross-validation.

[2]Another popular way to compute similarity is Pearsons equality which is not discussed in this paper but is a slight variant of the computation above.

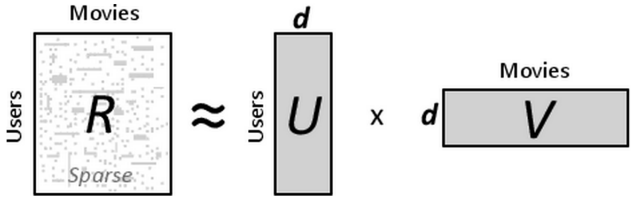


Figure 1: Visualization of Model-Based Collaborative Filtering Computation. Image taken from GraphLab website.

Ideally $r_{i,j} =< u_i, v_j > \forall i,j$, but in practice we need to minimize loss functions of U and V to obtain these matrices. The loss function minimized for the learning aglorithms run in this paper is the root mean square error (RSME). Therefore,

$$RMSE = \sqrt{\frac{1}{n} \sum_{u,v} |(p_{u,v} - r_{u,v})^2|}$$

where $p_{u,v}$ and $r_{u,v}$ are predicted and observed ratings for user u and item v respectively [11]. Here the predicted value is computed via the following equation:

$$p_{i,j} =< u_i, v_j >$$

The low rank approximation problem is thus formulated as follows to learn the factor vectors (u_i, v_j):

$$(u_i, v_j) = \min_{u,v} \sum_{(u,i) \in K} (p_{u,v} - r_{u,v})^2$$

This problem has $(n_u + n_v) * d$ free parameters that need to be determined. The set of known ratings K is a sparse matrix and thus is much smaller than the size of it's dense counterpart $n_u n_v$ elements. Therefore solving the low rank approximation problem as formulated above usually overfits the data [17]. In order to avoid overfitting a common technique is to use Tikhonov regularization to transform the low rank approximation problem into the following:

$$(u_i, v_j) = \min_{u,v} \sum_{(i,j) \in K} (p_{i,j} - r_{i,j})^2 + \lambda(||u_i||^2 + ||v_j||^2)$$

3. MATRIX FACTORIZATION ALGORITHMS

In this section the matrix factorization algorithms (model-based methods) run for the experiments are described.

3.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent computes a single prediction p_{ij} and it's error:

$$e_{ij} = p_{ij} - r_{ij}$$

The parameters of this algorithm are updated by a magnitude propotional to the learning rate α in the opposite direction [9] of the gradient resulting in the following update rules:

$$u_i := u_i + \alpha(e_{ij}\dot{v}_j - \alpha\dot{u}_i)$$

$$v_j := v_j + \alpha(e_{ij}\dot{u}_i - \alpha\dot{v}_j)$$

The implementation of SGD benchmarked for this paper uses an adapative learning rate that has a constant multiplicative step decrease of 0.9.

3.2 Bias Stochastic Gradient Descent (B-SGD)

Bias stochastic gradient descent is similar to the classic Section 3.1 with a different objective function. B-SGD tries to take advantage of biases in users and items. For example a item might be consistently rated higher by all users because it is a good item [9]. B-SGD incorporates this knowledge into its model by introducing a bias of rating $r_{i,j}$ derived as follows:

$$b_{i,j} = \mu + b_i + b_j$$

In this equation μ is the overal average rating, b_i is the observed deviations of user i, and b_j is the observed deviations of item j. New ratings are now predicted as:

$$p_{i,j} =< u_i, v_j > +b_{i,j}$$

Finally the objective function to be minimized now becomes:

$$(u_i, v_j) = \min_{u,v} \sum_{(i,j)\in K} (p_{i,j} - b_{i,j} - r_{i,j})^2 + \lambda(||u_i||^2 + ||v_j||^2 + b_i^2 + b_j^2)$$

3.3 Alternating Least Square (ALS)

Alternating Least Squares rotates between fixing one of the unknowns u_i or v_j. When one is fixed the other can be computed by solving the least-squares problem. This approach is useful because it turns the previous non-convex problem into a quadratic that can be solved optimally [9]. A general desctiption of the algorithm for ALS algorithm for collaborative filtering taken from Zhou et. al [17] is as follows:

Step 1 Initialize matrix V by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries.

Step 2 Fix V, solve U by minimizing the RMSE function.

Step 3 Fix U, solve V by minimizing the RMSE function similarly.

Step 4 Repeat Steps 2 and 3 until convergence.

3.4 Weighted Alternating Least Square (W-ALS)

Weighted alternating least squares is an algorithm very similar in spirit to the one described in Section 3.3 but with a different objective function. W-ALS is aimed at trying to optimize collaborative filtering algorithms for datasets that are derived off of implicit ratings. An example of implicit ratings is a cable company assigning user ratings of channels based upon the amount of time the user watches that channel. Here it is not explicity clear that the user likes the channel just because they watch it for a large time period. W-ALS introduces different confidence levels for which items are preferred by the user changing the objective function to be minimized to:

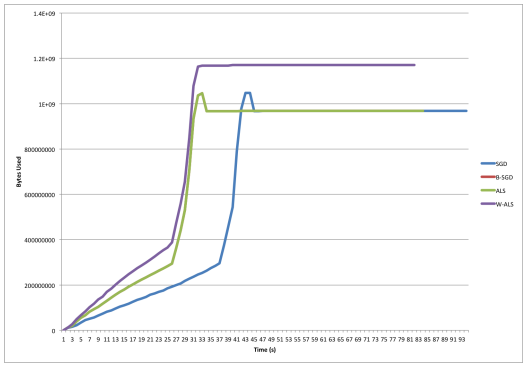


Figure 7: Memory Usage of Algorithms on MovieLens Dataset with 48 threads.

$$(u_i, v_j) = \min_{u,v} \sum_{(i,j)\in K} c_{i,j}(p_{i,j} - r_{i,j})^2 + \lambda(||u_i||^2 + ||v_j||^2)$$

Here $c_{i,j}$ measures the confidence of the predicted preference $p_{i,j}$. There are many ways to compute the confidence level [6] but the algorithm benchmarked in this paper uses the following:

$$c_{i,j} = 1 + \alpha log(1 + r_{i,j}/\epsilon)$$

4. EXPERIMENTS

In this section I present and analyze several model based implementations run on a number of datasets at scale on commodity NUMA hardware.

4.1 Experimental Setup

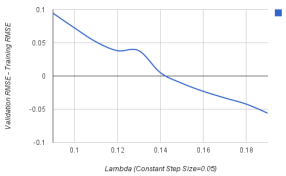
I ran all experiments on a single machine with four NUMA nodes, each equipped with an 12-core Intel Xeon E5-4657L v2 CPU and 256 GiB of RAM. The machine was running Ubuntu 12.04 LTS and we used g++ 4.6.4 for compilation. I measured the wall-clock time for each algorithm to complete on all datasets listed and do not count the time used for data loading for the system. In addition, I monitor the memory usage of each system by recording the amount allocated for process by the operating system.

For each dataset I ran each algorithm on a fixed training and validation sets. I held out 30% of the total samples for the validation set while training on the remaining 70% of the original data. I report both training and validation errors. I run with the default PowerGraph engine and scheduler arguments.

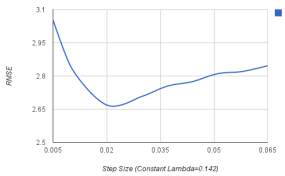
Because the purpose of this project is to benchmark which algorithm performs the best at scale on modern hardware I attempt to fix as many variables as possible. I used a fixed number of features $D = 25$ for all datasets and algorithms. In addition I ran all algorithms for a 5 iterations measuring RMSE, memory usage, and updates per second for comparison. I sweep the parameter space for λ and step size to select the parameters giving the best validation error on each dataset.

4.2 Parameter Tuning

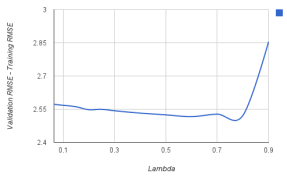
In order to select valid λ and SGD gradient step sizes I run a parameter sweep to find which values gave me the lowest RMSE on each dataset. The parameter sweeps are shown in Figures 2,3,4, and 5. Table 2 shows the final error of all algorithms across all datasets while using the best observed parameters.



(a) SGD Amazon λ Selection

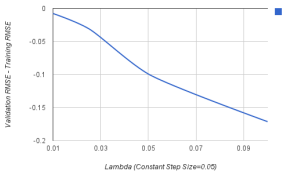


(b) SGD Amazon Step Size Selection

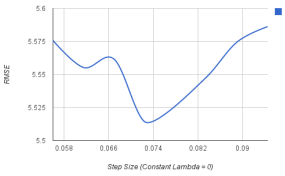


(c) ALS Amazon λ Selection

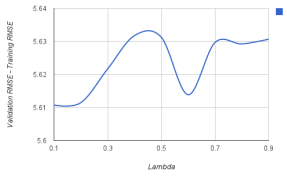
Figure 2: Validation of Parameter Choices for Amazon Dataset



(a) SGD BookCrossing λ Selection

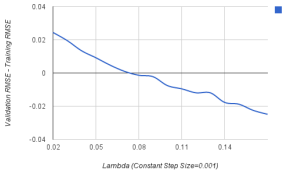


(b) SGD BookCrossing Step Size Selection

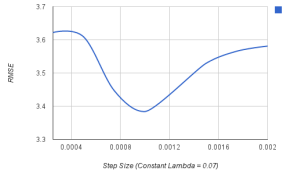


(c) ALS BookCrossing λ Selection

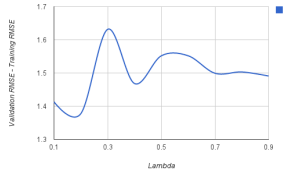
Figure 3: Validation of Parameter Choices for BookCrossing Dataset



(a) SGD Epinion λ Selection

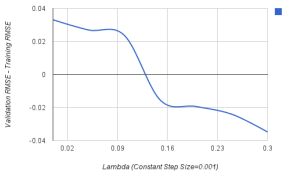


(b) SGD Epinion Step Size Selection

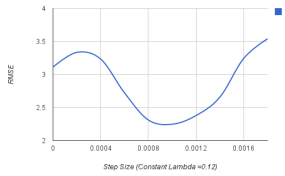


(c) ALS Epinion λ Selection

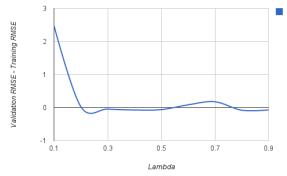
Figure 4: Validation of Parameter Choices for Epinion Dataset



(a) SGD MovieLens λ Selection



(b) SGD MovieLens Step Size Selection

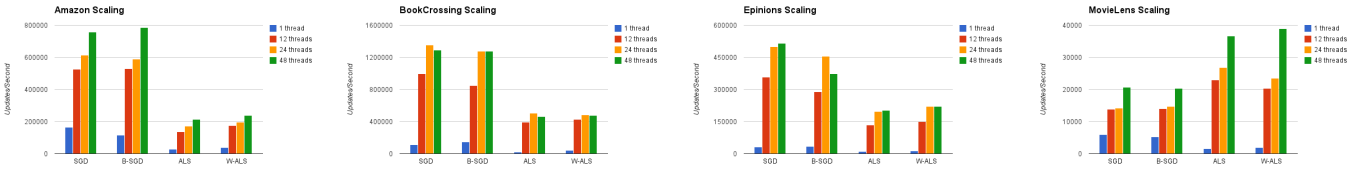


(c) ALS MovieLens λ Selection

Figure 5: Validation of Parameter Choices for MovieLens Dataset

	SGD		B-SGD		ALS		W-ALS	
Dataset	Training	Validation	Training	Validation	Training	Validation	Training	Validation
Amazon	2.44	2.91	0.60	1.43	0.66	2.85	0.34	3.02
BookCrossing	5.63	5.52	1.52	2.33	0.40	6.03	0.28	6.04
Epinions	3.38	3.38	0.81	0.84	0.63	2.13	0.40	1.46
MovieLens	2.20	2.23	1.01	1.02	0.66	0.92	0.66	0.95

Table 2: Training and Validation RMSE on Algorithms with 25 features.



(a) Amazon Dataset (b) BookCrossing Dataset (c) Epinions Dataset (d) MovieLens Dataset

Figure 6: Scaling Comparison of Different Algorithms Across Datasets

4.3 Benchmarking Results

The performance of each algorithm on all datasets run using various numbers of threads can be seen in Figure 6 which when used in comparison to Table 2 provides the performance and accuracy tradefoff of each dataset and algorithm. Finally, Figure 7 shows the memory usage of each algorithm on the MovieLens dataset when running with 48 threads.

For a scaling comparison I ran each dataset on 1, 12, 24, and 48 threads reporting the updates per second in the Figure 6. Scaling often ceases or degrades at 48 threads which at first might seem unexpected, but when one considers the underlying NUMA architecture of the hardware memory access patterns can explain this trend. Without replication of the data across sockets the memory bandwidth of communication between sockets exceeds the benefit of extra computational power.

In addition, one will notice that gradient descent is consistently faster than alternating least squares in almost all cases. Alternating least squares performs better on the MovieLens dataset which is extremely sparse. Further, alternating least squares often scales bettern than gradient descent, but typically does not overcome the performance loss that the algorithm starts off with.

5. FUTURE WORK

In the future I plan to use the knowledge discovered here while integrating collaborative filtering into my own framework for shared memory sparse matrix computation. Here I have tight control over the algorithms scaling and memory usage rather than relying on the PowerGraph runtime, which is optimized for distributed computation. Further, I want to benchmark collaborative filtering with using Hogwild!, a lock free method of parallelizing stochastic gradient descent [14]. Finally, I want to futher experiment with varying numbers of features rather than using a fixed value.

6. CONCLUSION

In conclusion Bias Stochastic Gradient Descent appears to be the most promising algorithm for all datasets except MovieLens on a NUMA machine when optimizing for performance and accuracy. The MovieLens dataset, which is extremely sparse, works much better on alternating least squares providing higher accuracy and better scaling on this extremely sparse dataset. The data here did not run extremely well on weighted alternating least squares which is not extremely suprising as this is explicit rating data and weighted alternating least squares is optimized for implicit rating data sources.

Finally, I learned that perhaps the most challenging part of machine learning in practice is picking both the proper number of features and the proper algorithmic parameter values. Without the theroy learned in this class, one cannot make these decsisons in a reasonable fashion.

7. REFERENCES

[1] Amazon ratings network dataset – KONECT, Nov. 2014.
[2] Bookcrossing (ratings) network dataset – KONECT, Nov. 2014.
[3] Epinions product ratings network dataset – KONECT, Nov. 2014.
[4] Movielens 10m network dataset – KONECT, Nov. 2014.
[5] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.
[6] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.
[7] Z. Huang, D. Zeng, and H. Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22(5):68–78, Sept. 2007.
[8] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.
[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.
[10] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 31–46, Berkeley, CA, USA, 2012. USENIX Association.
[11] J. Lee, M. Sun, and G. Lebanon. A comparative study of collaborative filtering algorithms. *CoRR*, abs/1205.3193, 2012.
[12] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. *CoRR*, abs/1006.4990, 2010.
[13] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 187–192, Edmonton, Alberta, 2002.
[14] F. Niu, B. Recht, C. Re, and S. J. Wright. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. *ArXiv e-prints*, June 2011.
[15] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
[16] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, Jan. 2009.
[17] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.