On the Difficulty of Evaluating Baselines

A Study on Recommender Systems

Steffen Rendle[*]
srendle@google.com

Li Zhang[*]
liqzhang@google.com

Yehuda Koren[†]
yehuda@google.com

Abstract

Numerical evaluations with comparisons to baselines play a central role when judging research in recommender systems. In this paper, we show that running baselines properly is difficult. We demonstrate this issue on two extensively studied datasets. First, we show that results for baselines that have been used in numerous publications over the past five years for the Movielens 10M benchmark are suboptimal. With a careful setup of a vanilla matrix factorization baseline, we are not only able to improve upon the reported results for this baseline but even outperform the reported results of any newly proposed method. Secondly, we recap the tremendous effort that was required by the community to obtain high quality results for simple methods on the Netflix Prize. Our results indicate that empirical findings in research papers are questionable unless they were obtained on standardized benchmarks where baselines have been tuned extensively by the research community.

1 Introduction

In the field of recommendation systems, numerical evaluations play a central role for judging research. Newly published methods are expected to be compared to baselines, i.e., well-known approaches, in order to measure the improvements over prior work. The best practices require reproducible experiments on several datasets with a clearly described evaluation protocol, baselines tuned by hyperparameter search, and testing for statistical significance of the result. Findings from such experiments are considered reliable. In this work, we question this practice and show that running baselines properly is difficult.

We highlight this issue on the extensively studied Movielens 10M (ML10M) benchmark [11]. Over the past five years, numerous new recommendation algorithms have been published in prestigious conferences such as ICML [17, 21, 36],

[*]Google Research, Mountain View, USA
[†]Google, Haifa, Israel

NeurIPS [18], WWW [31, 19], SIGIR [5], or AAAI [20, 4] reporting large improvements over baseline methods. However, we show that with a careful setup of a vanilla matrix factorization baseline, we are not only able to outperform the reported results for this baseline but even the reported results of any newly proposed method. Our findings question the empirical conclusions drawn from five years of work on this benchmark. This is worrisome because the ML10M benchmark follows the best practices of reliable experiments. Even more, if results on a well studied benchmark such as ML10M are misleading, typical one-off evaluations are more prone to producing unreliable results. Our explanation for the failure of providing reliable results is that the difficulty of running baselines is widely ignored in our community. This difficulty of properly running machine learning methods could be observed on the Netflix Prize [2] as well. Section 2.2 recaps the tremendous community effort that was required for achieving high quality results for vanilla matrix factorization. Reported results for this simple method varied substantially but eventually the community arrived at well-calibrated numbers. The Netflix Prize also highlights the benefits of rigorous experiments: the findings stand the test of time and, as this paper shows, the best performing methods of the Netflix Prize also work best on ML10M.

Recognizing the difficulty of running baselines has implications both for conducting experiments and for drawing conclusions from them. The common practice of one-off evaluations where authors run several baselines on a few datasets is prone to suboptimal results and conclusions should be taken with care. Instead, high confidence experiments require standardized benchmarks where baselines have been tuned extensively by the community. Finally, while our work focuses exclusively on evaluations, we want to emphasize that empirical comparisons using fixed metrics are not the only way to judge work. For example, despite comparing to sub-optimal baselines on the ML10M benchmark, recent research has produced many useful techniques, such as local low rank structures [17], mixtures of matrix approximation [18], and autoencoders [31].

2 Observations

In this section, we first examine the commonly used Movielens 10M benchmark for rating prediction algorithms [17, 31, 5, 36, 32, 3, 21, 4, 18, 19]. We show that by carefully setting up well known methods, we can largely outperform previously reported results. The surprising results include (1) Bayesian MF [29, 8] which was reported by previous authors to be one of the poorest performing methods, can outperform any result reported so far on this benchmark including all of the recently proposed algorithms. (2) Well known enhancements, proposed a decade ago for the Netflix Prize, such as SVD++ [12] or timeSVD++ [14], can further improve the quality substantially.

Secondly, we compare these findings to the well documented evolution of results on the Netflix Prize. Also on the Netflix Prize, we observe that setting up methods is challenging. For example, results reported for matrix factorization

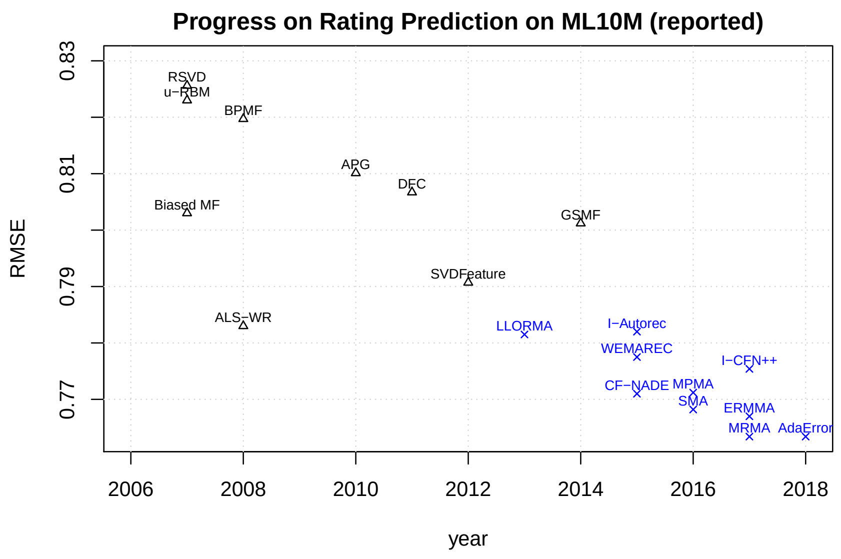

Figure 1: Progress on rating prediction measured on the Movielens 10M benchmark. Results marked as blue crosses were reported by the corresponding inventors. Results marked as black triangles were run as baselines by authors of newly invented methods. Results are from [17, 31, 5, 36, 32, 3, 21, 4, 18, 19]. See Table 1 for details.

vary considerably over different publications [9, 16, 22, 10, 24, 29, 25, 1, 37]. However, the Netflix Prize encouraged tweaking and reporting better runs of the same method, so over the long run, the results were well calibrated.

2.1 Movielens

Measuring Root Mean Square Error (RMSE) on a global random 90:10 split of Movielens 10M is a common benchmark for evaluating rating prediction methods [17, 31, 5, 36, 32, 3, 21, 4, 18]. Figure 1 shows the progress reported over the past 5 years on this benchmark. All newly proposed methods clearly outperform the earlier baselines such as matrix factorization or Boltzman machines [30] (RBM). Both SGD (RSVD, Biased MF) and Bayesian versions (BPMF) of matrix factorization have been found to perform poorly. The figure indicates a steady progress, by improving the state-of-the art in rating prediction considerably. Many results include also standard deviations to show that the results are statistically significant, e.g., [32, 18].

2.1.1 A Closer Look at Baselines

The reported results for Biased MF, RSVD, ALS-WR, and BPMF indicate some issues.

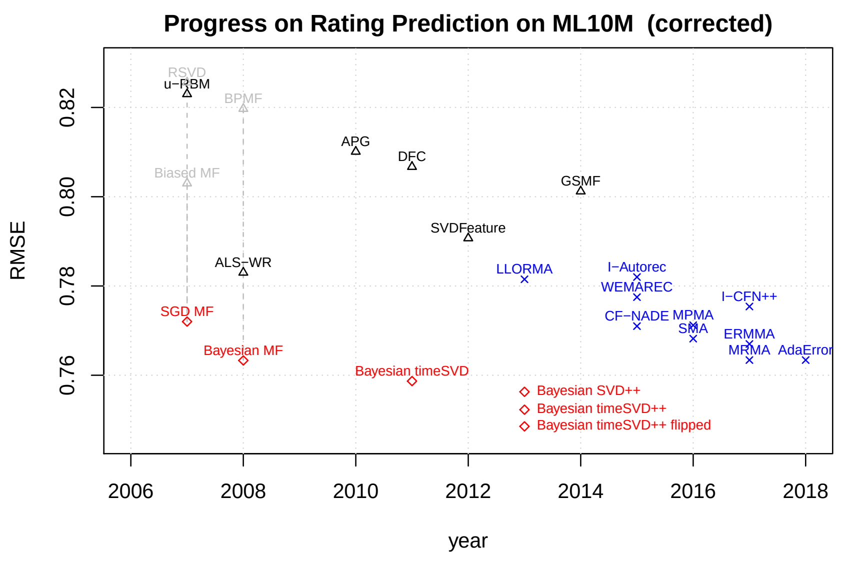

Figure 2: Rerunning baselines for Bayesian MF, and adding popular methods such as Bayesian versions of SVD++, timeSVD++, timeSVD[1]. Bayesian MF (=BPMF) and SGD MF (=RSVD, Biased MF) can achieve much better results than previously reported. With a proper setup, well known methods can outperform any recently proposed method on this benchmark.

1. Biased MF [15] and RSVD [24] are essentially the same method: L2 regularized matrix factorization learned with SGD. Qualitative differences should only stem from different setups such as hyperparameters, training data ordering, or implementations.

2. ALS-WR [37] and Biased MF/ RSVD are identical models learned by different algorithms. When both are tuned well, they have shown very similar results on the Netflix Prize (see Section 2.2).

3. BPMF [29] shares the model with RSVD/ALS-WR but is learned with a Gibbs sampler. On the Netflix Prize it has shown the best performance for learning matrix factorization models [29, 28] compared to other learning methods (SGD, ALS, VB). It is surprising that it shows much worse quality than Biased MF and ALS-WR on Movielens 10M.

2.1.2 Rerunning Baselines

We reran the baselines and a different picture emerged (see Figure 2 and Table 1). More details about the experiments can be found in the Appendix.

[1]SVD++ was introduced in 2008 [12], timeSVD++ in 2009 [14]. We use here a Bayesian FM solver which was introduced in 2011 [8]. The Bayesian solver to handle the implicit case efficiently (i.e., the "++") was published in 2013 [28].

Matrix Factorization First, we ran a matrix factorization model learned by an SGD algorithm (similar to RSVD and Biased MF). SGD-MF achieved an RMSE of 0.7720 for a 512-dimensional embedding and an RMSE of 0.7756 for 64 dimensions. This is considerably better than the reported values for RSVD (0.8256) and Biased MF (0.803), and outperforms even several of the newer methods such as LLORMA (0.7815), Autorec (0.782), Wemarec (0.7769), or I-CFN++ (0.7754).

Next, we trained a Bayesian matrix factorization model using Gibbs sampling (similar to BPMF). Bayesian MF achieved an RMSE score of 0.7633 for a 512 dimensional embedding. This is not only much better than the previously reported [21, 32] number for BPMF (0.8197) but it outperforms even the best method (MRMA 0.7634) ever reported on ML10M.

Stronger Baselines One of the lessons of the Netflix Prize was that modeling implicit activity was highly predictive and outperformed vanilla matrix factorization. SVD++[12], the asymmetric model (NSVD1) [24] and to some extent RBMs [30] are examples of models harnessing implicit feedback. Another important aspect was capturing temporal effects [14].

First, we added a time variable to the Bayesian matrix factorization model, and achieved an RMSE of 0.7587. Second, we trained an implicit model by adding a bag-of-words predictor variable that includes all the videos a user watched. This model is equivalent to SVD++ [12, 27]. It further improved over Bayesian MF, and achieves an RMSE of 0.7563. Next, we trained a joint model with time and the implicit usage information, similar to the timeSVD++ model [14]. This model achieved an RMSE of 0.7523. Finally, we added a flipped version of the implicit usage in timeSVD++: a bag of word variable indicating the other users who have watched a video. This model dropped the RMSE to 0.7485.

Summary By carefully setting up baselines, we could outperform any result even with a simple Bayesian matrix factorization – a method that was previously reported to perform poorly on ML10M. Applying modeling techniques known for almost a decade, we were able to achieve substantial improvements – in absolute terms, we improved over the previously best reported result, MRMA, from 2017 by 0.0144, a similar margin as several years of improvements reported on this dataset[2]. Our results question conclusions drawn from previous experimental results on ML10M. Instead of improving over the baselines by a large margin, all recently proposed methods *underperform* well-known baselines substantially.

[2]The difference between LLORMA in 2013 to MRMA in 2017 is 0.0186.

Method	RMSE	Comment
RSVD [24]	0.8256	result from [21]
U-RBM [30]	0.823	result from [31]
BPMF [29]	0.8197	result from [21]
APG [7]	0.8101	result from [21]
DFC [23]	0.8067	result from [21]
Biased MF [15]	0.803	result from [31]
GSMF [35]	0.8012	result from [21]
SVDFeature [6]	0.7907	result from [32]
ALS-WR [37]	0.7830	result from [32]
I-AUTOREC [31]	0.782	result from [31]
LLORMA [17]	0.7815	result from [17]
WEMAREC [5]	0.7769	result from [5]
I-CFN++ [32]	0.7754	result from [32]
MPMA [3]	0.7712	result from [3]
CF-NADE 2 layers [36]	0.771	result from [36]
SMA [21]	0.7682	result from [21]
GLOMA [4]	0.7672	result from [4]
ERMMA [20]	0.7670	result from [20]
AdaError [19]	0.7644	result from [19]
MRMA [18]	0.7634	result from [18]
SGD MF [24, 15]	0.7720	same method as RSVD, Biased MF
Bayesian MF [29, 8]	0.7633	same method as BPMF
Bayesian timeSVD [14, 8, 28]	0.7587	MF with a time variable
Bayesian SVD++ [12, 28]	0.7563	similar to [12] learned with MCMC
Bayesian timeSVD++ [14, 28]	0.7523	similar to [14] learned with MCMC
Bayesian timeSVD++ flipped [28]	0.7485	adding implicit item information

Table 1: Movielens 10M results: first group are baselines. Second group are newly proposed methods. Third group are baseline results that we reran. See Appendix for details of our results.

2.2 Netflix Prize

The Netflix Prize [2] also indicates that running methods properly is hard. We are highlighting this issue by revisiting the large community effort it took to get well calibrated results for the vanilla matrix factorization model.

2.2.1 Background

The Netflix Prize was awarded to the first team that decreases by 10% the RMSE of Netflix's own recommender system, Cinematch, with an RMSE score of 0.9514 [2]. It took the community about three years and hundreds of ensembled models to beat this benchmark. Given that the overall relative improvement for winning the prize was 0.095, a difference of 0.01 in RMSE scores is considered large – e.g., it took one year to lower the RMSE from 0.8712 (*progress prize 2007*) to 0.8616 (*progress prize 2008*) and the *Grand prize* was awarded in 2009 for an RMSE of 0.8554.

The Netflix Prize dataset is split into three sets: a training set, the *probe* set for validation and the *qualifying* set for testing. The ratings of the qualifying set were withheld during the competition. Participants of the Netflix Prize could submit their predictions on the qualifying set to the organizer and the RMSE score on half of this set, the *quiz* set, was reported on the *public* leaderboard. The RMSE on the remaining half, the *test* set, was private to the organizer and used to determine the winner. Scientific papers usually report either the probe RMSE or the quiz RMSE[3].

The data was split based on user and time between the training, probe and qualifying set. In particular, for every user, the last six ratings were withheld from training, three of them were put into the probe set and three into the qualifying set. This split technique is more challenging than global random splitting[4] for two reasons. (1) Users with few ratings have the same number of evaluation data points as frequent users. Compared to a global random split, ratings by users with little training data, i.e., harder test cases, are overrepresented in the test dataset. (2) Withholding by time makes this a forecasting problem where test ratings have to be extrapolated whereas a global random split allows simpler interpolation.

2.2.2 Matrix Factorization

From the beginning of the competition, matrix factorization algorithms were identified as promising methods. Very early results used traditional SVD solvers with sophisticated imputation and reported results close to a probe RMSE of

[3]The quiz set is easier to predict than the probe set (typically, RMSE is about 0.007 lower), because for training the *quiz model*, the probe set can be added to enrich the training set.

[4]Unfortunately, in recent work on rating prediction it is common to evaluate on a global random 90:10 split of the Netflix Prize data. Results of global splitting are not comparable to numbers reported on the Netflix Prize split. This is unfortunate because the original Netflix split has been extensively studied and has very well calibrated results.

Team / Publication	Probe RMSE	Quiz RMSE
Kurucz et al. [16]	0.94	-
Simon Funk [9]	0.93	-
Lim and Teh [22]	0.9227	-
Gravity [10]	0.9190	-
Paterek [24]	-	0.9094
Pragmatic Theory [26]	0.9156	0.9088
Big Chaos [33]	-	0.9028
Pilaszy et al. [25]	-	0.9018
BellKor [1]		0.8998
Zhou et al. [37]	-	0.8985

Table 2: Netflix Prize: Results for vanilla matrix factorization models using ALS and SGD optimization methods.

0.94 [16]. A breakthrough was FunkSVD [9], a sparse matrix factorization algorithm that ignored the missing values. It was learned by iterative SGD with L2 regularization and achieved a probe RMSE of 0.93. This encouraged more researchers to experiment with matrix factorization models and in the KDDCup 2007 workshop, participants reported improved results of 0.9227 (probe) [22], 0.9190 (probe) [24], and 0.9094 (quiz) [24]. Participants continued to improve the results for the basic matrix factorization models and reported scores as low as 0.8985 [37] for ALS and 0.8998 [13] for SGD methods.

Table 2 summarizes some of the key results for vanilla matrix factorization including the results reported by top competitors and the winners. These results highlight that achieving good results even for a presumably simple method like matrix factorization is non trivial and takes large effort.

2.2.3 Refinements and Winning Algorithms

Our previous discussion was restricted to vanilla matrix factorization models. After the community converged to well calibrated results for matrix factorization, the focus shifted to more complex models taking into account additional information such as implicit feedback (e.g., SVD++ [12]) and time (e.g., timeSVD++ [14]). The most sophisticated timeSVD++ models achieved RMSEs as low as 0.8762 [13].

All the top performing teams also relied heavily on ensembling as many diverse models as possible including sophisticated nearest neighbor models [15], or Restricted Boltzmann Machines [30]. The final models that won the Netflix Prize were ensembles of several teams each with dozens of models [13].

2.3 Discussion

Compared to recent evaluations on ML10M, the Netflix Prize encouraged rerunning methods and reporting improvements on identical methods (see Table 2). This ensured that the community converged towards understanding which methods work well. In contrast to this, for ML10M there is no encouragement to rerun results for simple baselines, or even to outperform complex new methods. One explanation is that for the Netflix Prize, participants get rewarded by getting a low RMSE – no matter how it was achieved. In terms of publications, which motivates current work on rating prediction, achieving good results with old approaches is usually not seen as a scientific contribution worth publishing.

However, the ultimate goal of empirical comparisons is to better understand the trade-offs between alternative methods and to draw insights about which patterns lead to successful methods. Our experiments have shown that previous empirical results on ML10M fail to deliver these insights. Methods that were reported to perform poorly actually perform very well. In contrast to this, our experiments on ML10M show that all the patterns learned on the Netflix Prize hold also on ML10M, and the best Netflix Prize methods perform also best on ML10M. In this sense, the Netflix Prize was successful and ML10M benchmark was not (so far).

Like previous baselines were not properly tuned on ML10M, it is possible that the recently proposed methods could also improve their results with a more careful tuning. This would not be a contradiction to our observation but be further evidence that running experiments is hard and needs large effort of experimentation and tuning to achieve reliable results.

Finally, we want to stress that this is not a unique issue of ML10M. Quite the opposite, most work in recommendation systems is not even evaluated on a standardized benchmark such as ML10M. Results obtained on one-off evaluations are more prone to questionable experimental findings than ML10M.

3 Insufficient Indicators for Experimental Reliability

We shortly discuss commonly used indicators that have been used to judge the reliability of experimental results, such as statistical significance, reproducibility or hyperparameter search. While all of them are necessary, we argue that they are not sufficient to ensure reliable results. Our results in Section 2 suggest that they are less important than proper set ups.

3.1 Statistical Significance

Most of the results for ML10M are reported with a standard deviation (e.g, [32, 18]). The reported standard deviation is usually low and the difference of the reported results are *statistically significant.* Even for the reported BPMF results in [18], the standard deviation is low (0.0004). Based on our study

(Section 2.1), statistically significant results should not be misinterpreted as a "proof" that method A is better than B. While this sounds like a contradiction, statistical significance does not measure how well a method is set up. It measures the variance within one setup.

Statistical significance and standard deviations should only be considered after we have evidence that the method is used well. We argue that setting the method up properly is a much larger source for errors. In this sense, statistical significance is of little help and often provides a false confidence in experimental results.

3.2 Reproducibility

The ability to rerun experiments and to achieve the same numbers as reported in previous work is commonly referred to as *reproducibility*. Often, implementations and hyperparameter setups are shared by authors to allow reproducing results. While reproducibility is important, it does not solve the issue we point out in this work. Rerunning the code of authors can reproduce the results but it is not evidence of a proper setup. In the example of the ML10M dataset, the dataset is public, the experimental protocol is documented and simple, and there exist plenty implementations of the baselines – even authors commonly make their new methods public. The same holds for the Netflix prize, or most machine learning competitions. Despite the easy reproducibility, the conclusions from experimental results can be questionable (see Section 2.1).

3.3 Tuned Hyperparameters

One of our central arguments is that it is not easy to run a machine learning method properly. In most research papers, it is common practice to search over the hyperparameter space (e.g., learning rates, embedding dimension, regularization) and report the results for the "best" setting. However, Section 2 indicates that this still does not solve the problem, and reported results can vary substantially from a proper setup. We speculate that hyperparameter search spaces are often incomplete and do not replace experience with a method. For example, interpreting and acting on the results of different hyperparameter settings is non-trivial, e.g., should the boundary be extended, or refined? What is the right search grid? Can we search hyperparameters on a small model and transfer the results to a larger one? All these questions make it hard for setting up an unknown "black-box".

A second source are knobs that are not even considered during hyperparameter search. For example, a method might require to recenter the data before running it, or that the training data is shuffled, or to stop training early, or to use a certain initialization. Such knobs might be trivial and not even worth reporting for someone with experience in a method, but will make it almost impossible for others to set comparisons up properly. This becomes a problem when the method is run by a non-expert on a different dataset or experimental setup.

4 Improving Experimental Quality

Based on our findings, reliable experiments are hard to achieve by authors of a single paper but require a community effort. We see two key requirements for this: (1) standardized benchmarks; and (2) incentives to run and improve baseline results.

4.1 Standardized Benchmarks

While today's best practices encourage authors of a paper to run as many baselines as possible, our findings indicate that this should be discouraged because it is prone to producing unreliable results. If running baselines from scratch is discouraged, the only way to get points of comparisons to other methods are standardized benchmarks, i.e., datasets with well-defined train–test splits and evaluation protocols. ML10M with 10 fold CV or the Netflix Prize split, both measured on RMSE, are examples of well defined benchmarks for comparing rating prediction algorithms. However, recommender tasks are diverse, e.g. item recommendation vs. rating prediction, cold-start vs. active users, forecasting, explanation, etc. and most of them miss standardized benchmarks. While it is important to explore new tasks, over time it is crucial for the community to converge to standardized benchmarks for reoccurring problems. As we have argued in this paper, empirical findings on non-standardized benchmarks are likely less reliable.

A common concern about benchmarks is that methods ”overfit” to a particular dataset, leading to false discoveries. However, this is less of a problem for the scale of the data typically used in machine learning tasks. For example, one of the most heavily researched datasets, the Netflix Prize, has shown very little signs of overfitting after more than 10 years of study. Both the public leaderboard[5] and the private (hidden) leaderboard[6] show only minor differences in ordering and the same relative improvements. Also, our results on the ML10M dataset (see Section 2.1) emphasize that the lessons learned on the Netflix Prize still hold after a decade and the patterns and methods that worked the best for Netflix Prize are also those best performing on ML10M. While signs of overfitting might show up in the long run, the benefits from well calibrated results outweight issues that improper baselines might cause.

4.2 Incentives for Running Baselines

ML10M and the Netflix Prize are both standardized benchmarks. However, one of them produced well calibrated results, while the other one had misleading baselines for many years (see Table 1). Our explanation of this phenomena is that there is no encouragement to keep on improving baselines for ML10M. *Novelty* is a key criterion to judge research contributions. Achieving good results with a well known method gets little reward, so researchers do not spend much

[5]`https://www.netflixprize.com/leaderboard_quiz.html`
[6]`https://www.netflixprize.com/leaderboard.html`

effort on it – and even if good results would be achieved, it is hard to publish them. In contrast to this, the Netflix Prize encouraged spending time on experimenting with existing methods. This was the most certain way of getting good results, and a chance to improve on the contest leaderboard. Real life systems often also incentivize bettering known, well established methods rather than inventing new ones. We think it is crucial to find incentives for tuning well-known methods on benchmarks. As we have shown, this is a non-trivial task which needs expertise and time. Without well calibrated results, conclusions drawn from experiments are questionable.

Besides evaluations motivated by scientific publications, machine learning competitions, e.g., on platforms such as Kaggle[7] or organized by conferences such as the annual KDDCup, can serve as standardized benchmarks with well-calibrated results.

5 Conclusion

In this paper, we have shown that results for baselines that have been used in numerous publications over the past five years for the ML10M benchmark are suboptimal. With a careful setup of a vanilla matrix factorization baseline, we were not only able to outperform the reported results for the baselines but even the reported results of any newly proposed method. Other well-known models such as SVD++ provide an even higher gain. These results are surprising as the papers follow the best practices in our community to ensure reliable results: they conduct a reasonable hyperparameter search, report statistical significance and allow reproducibility. This indicates that running baseline methods properly is difficult. As recommender systems evaluation relies heavily on empirical results, the shortcomings discussed in this work highlight a major issue in our ability to judge work. Our findings question the common practice in recommender systems research papers of running baseline models and experimenting on multiple datasets. Even when following the best practices as outlined above, results can be unreliable. Our work indicates that trustworthy baselines require standardized benchmarks and considerable tuning effort by the community.

References

[1] Bell, R. M., Koren, Y., and Volinsky, C. The bellkor solution to the netflix prize, 2007.

[2] Bennett, J., and Lanning, S. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD* (2007).

[3] Chen, C., Li, D., Lv, Q., Yan, J., Chu, S. M., and Shang, L. Mpma: Mixture probabilistic matrix approximation for collaborative filtering. In

[7] `http://www.kaggle.com/`

Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (2016), IJCAI'16, AAAI Press, pp. 1382–1388.

[4] Chen, C., Li, D., Lv, Q., Yan, J., Shang, L., and Chu, S. Gloma: Embedding global information in local matrix approximation models for collaborative filtering. In *AAAI Conference on Artificial Intelligence* (2017).

[5] Chen, C., Li, D., Zhao, Y., Lv, Q., and Shang, L. Wemarec: Accurate and scalable recommendation through weighted and ensemble matrix approximation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2015), SIGIR '15, ACM, pp. 303–312.

[6] Chen, T., Zhang, W., Lu, Q., Chen, K., Zheng, Z., and Yu, Y. Svdfeature: A toolkit for feature-based collaborative filtering. *J. Mach. Learn. Res. 13*, 1 (Dec. 2012), 3619–3622.

[7] chuan Toh, K., and Yun, S. An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems. *Pacific J. Optim 6* (2010), 615–640.

[8] Freudenthaler, C., Schmidt-Thieme, L., and Rendle, S. Bayesian factorization machines. In *Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation* (2011).

[9] Funk, S. Netflix update: Try this at home. `http://sifter.org/~simon/journal/20061211.html`, 2006.

[10] Gabor Takacs, Istvan Pilaszy, B. N., and Tikk, D. On the gravity recommendation system. In *KDDCup* (2007).

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

[12] Koren, Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2008), KDD '08, ACM, pp. 426–434.

[13] Koren, Y. The bellkor solution to the netflix grand prize, 2009.

[14] Koren, Y. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2009), KDD '09, ACM, pp. 447–456.

[15] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (Aug. 2009), 30–37.

[16] Kurucz, M., Benczúr, A. A., and Csalogány, K. Methods for large scale svd with missing values. In *KDDCup* (2007).

[17] Lee, J., Kim, S., Lebanon, G., and Singer, Y. Local low-rank matrix approximation. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (2013), ICML'13, JMLR.org, pp. II–82–II–90.

[18] Li, D., Chen, C., Liu, W., Lu, T., Gu, N., and Chu, S. Mixture-rank matrix approximation for collaborative filtering. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 477–485.

[19] Li, D., Chen, C., Lv, Q., Gu, H., Lu, T., Shang, L., Gu, N., and Chu, S. M. Adaerror: An adaptive learning rate method for matrix approximation-based collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference* (Republic and Canton of Geneva, Switzerland, 2018), WWW '18, International World Wide Web Conferences Steering Committee, pp. 741–751.

[20] Li, D., Chen, C., Lv, Q., Shang, L., Chu, S., and Zha, H. Ermma: Expected risk minimization for matrix approximation-based recommender systems. In *AAAI Conference on Artificial Intelligence* (2017).

[21] Li, D., Chen, C., Lv, Q., Yan, J., Shang, L., and Chu, S. M. Low-rank matrix approximation with stability. In *Proceedings of the 33rd International Conference on Machine Learning - Volume 48* (2016), ICML'16, JMLR.org, pp. 295–303.

[22] Lim, Y. J., and Te, Y. W. Variational bayesian approach to movie rating prediction. In *KDDCup* (2007).

[23] Mackey, L., Talwalkar, A., and Jordan, M. I. Divide-and-conquer matrix factorization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems* (USA, 2011), NIPS'11, Curran Associates Inc., pp. 1134–1142.

[24] Paterek, A. Improving regularized singular value decomposition for collaborative filtering. In *KDDCup* (2007).

[25] Pilászy, I., Zibriczky, D., and Tikk, D. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (New York, NY, USA, 2010), RecSys '10, ACM, pp. 71–78.

[26] Piotte, M., and Chabbert, M. The pragmatic theory solution to the netflix grand prize, 2009.

[27] Rendle, S. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol. 3*, 3 (may 2012), 57:1–57:22.

[28] Rendle, S. Scaling factorization machines to relational data. In *Proceedings of the 39th international conference on Very Large Data Bases* (2013), PVLDB'13, VLDB Endowment, pp. 337–348.

[29] Salakhutdinov, R., and Mnih, A. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning* (New York, NY, USA, 2008), ICML '08, ACM, pp. 880–887.

[30] Salakhutdinov, R., Mnih, A., and Hinton, G. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, ACM, pp. 791–798.

[31] Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web* (New York, NY, USA, 2015), WWW '15 Companion, ACM, pp. 111–112.

[32] Strub, F., Mary, J., and Gaudel, R. Hybrid recommender system based on autoencoders. *CoRR abs/1606.07659* (2016).

[33] Töscher, A., and Jahrer, M. The bigchaos solution to the netflix grand prize, 2009.

[34] Yamada, M., Lian, W., Goyal, A., Chen, J., Wimalawarne, K., Khan, S. A., Kaski, S., Mamitsuka, H., and Chang, Y. Convex factorization machine for toxicogenomics prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2017), KDD '17, ACM, pp. 1215–1224.

[35] Yuan, T., Cheng, J., Zhang, X., Qiu, S., and Lu, H. Recommendation by mining multiple user behaviors with group sparsity. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (2014), AAAI'14, AAAI Press, pp. 222–228.

[36] Zheng, Y., Tang, B., Ding, W., and Zhou, H. A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33rd International Conference on Machine Learning - Volume 48* (2016), ICML'16, JMLR.org, pp. 764–773.

[37] Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management* (Berlin, Heidelberg, 2008), AAIM '08, Springer-Verlag, pp. 337–348.

Name	Features	Comment
Matrix Factorization	u, i	Equivalent to biased matrix factorization [15] and RSVD [24]
SVD++	u, i, iu	Similar to [12]
timeSVD	u, i, t	Similar to [14]
timeSVD++	u, i, t, iu	Similar to [14]
timeSVD++ flipped	u, i, t, iu, ii	

Table 3: Models from our ML10M experiment and the corresponding features that we used in a Factorization Machine. See Section 5.1 for an explanation of the features.

Appendix

In this section, we describe our experiments and setup in more detail. We experiment on the Movielens 10M dataset[8] with a 10 fold cross validation protocol. That is, 10 random splits each with 90% training data and 10% test data, where the 10 test splits do not overlap. Our test protocol allows to compare our results to previous publications on the Movielens 10M dataset with a random 90:10 split [17, 31, 21, 32, 3, 36, 4, 18]. Our experiments focus mainly on Bayesian models learned by Gibbs sampling, a Markov Chain Monte Carlo method, because they have fewer critical hyperparameters than SGD. However, we also run SGD matrix factorization to be able to compare to existing numbers [31, 21] reported for this method.

5.1 Factorization Models

We used the factorization machine library, libFM[9] [27], for all experiments. We consider five features which have been used successfully on the Netflix Prize:

1. User id (u): a categorical variable indicating the user
2. Item id (i): a categorical variable indicating the movie
3. Time (t): the day of the rating, which is treated as a categorical variable with one category per day
4. Implicit user information (iu): a bag of words variable that is the set of all movie ids a user has ever watched[10]

[8]`http://grouplens.org/datasets/movielens/10m/`

[9]`https://github.com/srendle/libfm`

[10]We used the same protocol as in the Netflix prize and included *which* movies the user rated from both the training and test set – for sure, we did not include the rating of the test set. We also experimented with a stricter definition of implicit information and removed information about which movies a user rated in the test set. As expected, this resulted in slightly worse RMSE numbers. However, implicit information is still very useful in the stricter setting.

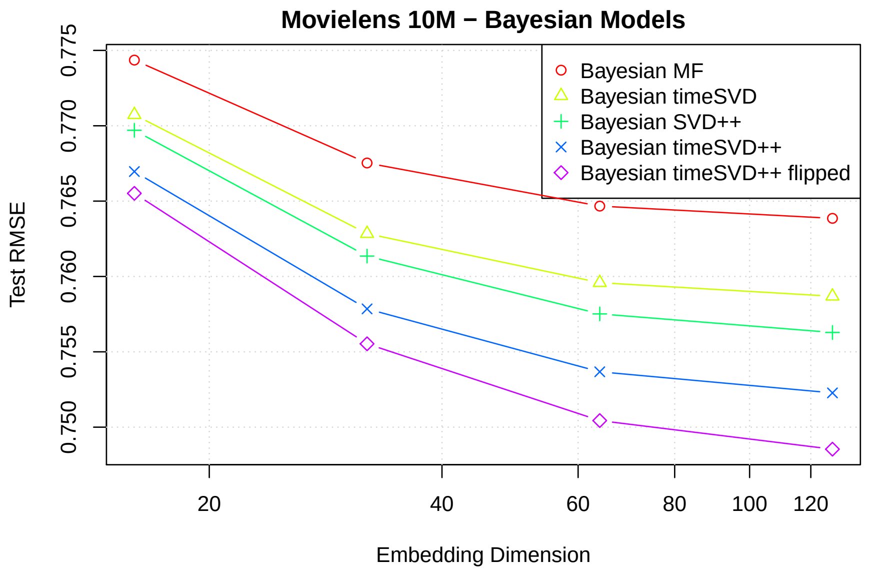

Figure 3: Quality of the Bayesian models with an increasing embedding dimension (with 512 sampling steps). Larger dimensions show better quality.

5. Implicit item information (ii): a bag of words variable that is the set of all user ids that have ever watched a movie

Table 3 lists the combination of features that we used.

5.2 Bayesian Learning

Setting up a Bayesian model is very simple. There are three critical settings: (a) number of sampling steps, (b) dimension of the embedding, and (c) initialization of model parameters. In our experience, the more sampling steps and the higher the dimension the better the quality. We report results for up to 512 steps (iterations), and embedding dimensions of 16, 32, 64, and 128 – for matrix factorization, we also ran with embedding dimensions of 256 and 512. For the initialization, we choose a random initialization from a Gaussian distribution with standard deviation of 0.1. The value 0.1 is the default in libFM and has worked well in the Netflix prize [28] and also for other Movielens splits [34]. We use the relational data representation and solver of libFM [28].

Figure 3 shows the final test RMSE vs. the embedding dimension for these models. The plot confirms that increasing the embedding dimension helps. It also shows that features that worked well in the Netflix prize, achieve high quality on Movielens 10M. The convergence graph, Figure 4, confirms that the prediction quality improves with more sampling steps.

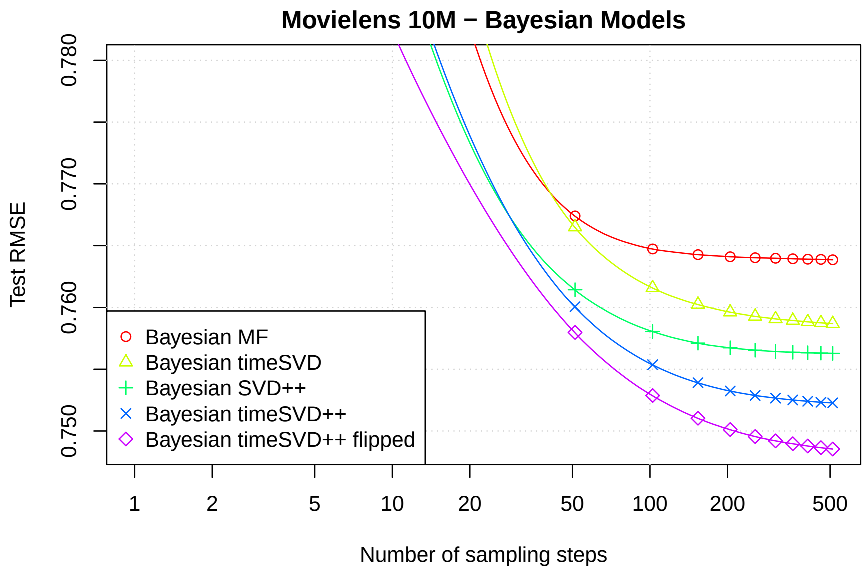

Figure 4: Quality of the Bayesian models with an increasing number of sampling steps (with 128 dimensional embeddings). More sampling steps show better quality.

5.3 Stochastic Gradient Descent

For SGD, we experimented only with matrix factorization. Compared to the Bayesian models, our SGD implementation has two additional hyperparameters: the learning rate, and regularization. In our experience, the smaller the learning rate the better the quality, however, the number of iterations will also grow. That means learning rate and number of iterations are not independent settings but form a runtime trade-off. We fix the number of iterations to 128 epochs and search for the best learning rate within this computational budget. Also for the embedding dimension, the larger the dimension the better the quality, provided that the regularization value and number of iterations is set properly. Larger dimensions are more costly, so we report numbers for 16, 32, 64, 128, 256, and 512 dimensions. For Normal initialization, we pick 0.1 as the standard deviation, which is the same value as for the Bayesian methods.

That leaves us with two hyperparameters to tune: (a) learning rate, and (b) regularization value. We perform hyperparameter selection on the training set. We use 5% of the training set for validating hyperparameter choices and the remaining 95% for training. We set up a grid over four regularization values $\{0.02, 0.03, 0.04, 0.05\}$ and two learning rates $\{0.001, 0.003\}$. This range of values was motivated by successful SGD hyperparameter combinations for matrix factorization on the Netflix prize [24, 12]. We picked 64 dimensions for tuning the hyperparameters because it is sufficiently large to show the impor-

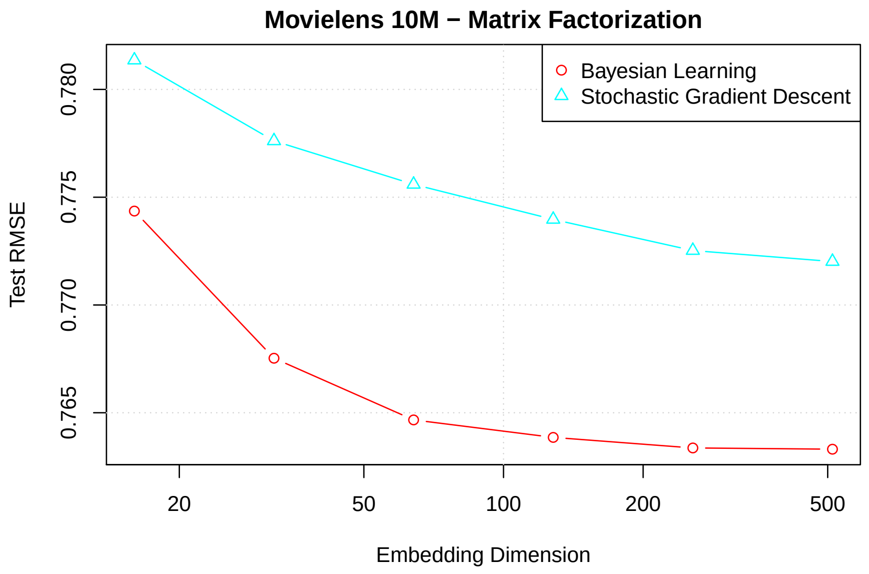

Figure 5: Comparison of Matrix Factorization learned by Gibbs Sampling (Bayesian Learning) and stochastic gradient descent (SGD) for an embedding dimension from 16 to 512.

tance of the regularization value but small enough that the computational cost of hyperparameter search is reasonable.

The selected hyperparameters were stable among folds and for all 10 folds, the best regularization value was 0.04 and the best learning rate 0.003. Finally, using the previously selected hyperparameters, we trained models for 16, 32, 64, 128, 256, and 512 dimensions on the full training data and evaluated on the 10% test split. Figure 5 shows the final quality of matrix factorization models learned by SGD and MCMC from 16 to 512 dimensions. The picture confirms that the larger the dimension, the better the quality.

It is likely that more sophisticated hyperparameter selection might lead to better SGD performance. For example, for the Netflix prize, we observed that using individual learning rates and regularization values for biases and embeddings as well as users and items can further improve results. In addition, a decay schedule of learning rates, which decreases them at later iterations, is also known to improve accuracy. Nevertheless, even with the above described hyperparameter search, we were able to outperform previously reported results for SGD substantially.