Online bagging for recommendation with incremental matrix factorization

João Vinagre[1,2], Alípio Mário Jorge[1,2], and João Gama[1,3]

[1] LIAAD - INESC TEC, Porto, Portugal
[2] Faculty of Sciences, University of Porto
[3] Faculty of Economics, University of Porto
jnsilva@inesctec.pt, amjorge@fc.up.pt, jgama@fep.up.pt

Abstract. Online recommender systems often deal with continuous, potentially fast and unbounded flows of data. Ensemble methods for recommender systems have been used in the past in batch algorithms, however they have never been studied with incremental algorithms, that are capable of processing those data streams on the fly. We propose online *bagging*, using an incremental matrix factorization algorithm for positive-only data streams. Using prequential evaluation, we show that bagging is able to improve accuracy more than 20% over the baseline with small computational overhead.

Keywords: Recommender systems, bagging, matrix factorization, data streams

1 Introduction

In many real world recommender systems, user feedback is continuously generated at unpredictable rates and order, and is potentially unbounded. In large scale systems, the rate at which user feedback is generated can be very fast. Building predictive models from these continuous flows of data is a problem actively studied in the field of data stream mining. Ideally, algorithms that learn from data streams should be able to process data at least as fast as it arrives, in a single pass, while maintaining an always-available model [3]. Most incremental algorithms naturally have these properties, and are thus a viable solution, including in recommendation problems [12]. Incremental algorithms for recommendation treat user feedback data as a data stream, immediately incorporating new data in the recommendation model. In most recommendation tasks this is a desirable feature, since the task of a recommender system is to find the most relevant items – such as books, music tracks, restaurants or movies – to each user, individually. Naturally, users are human beings, whose preferences change over time. Moreover, in large scale systems, new users and items are permanently entering the system. A model that is immediately updated with fresh data has the capability of adjusting faster to such changes.

Ensemble methods in machine learning are convenient techniques to improve the accuracy of algorithms. Typically, this is achieved by combining results from

a number of weaker sub-models. Bagging [1], Boosting [4] and Stacking [13] are three well-known ensemble methods used with recommendation algorithms. Boosting is experimented in [2,9,7,10], bagging is studied also in [7,10], and stacking in [11]. In all of these contributions, ensemble methods work with batch learning algorithms only.

In this paper we propose online bagging for incremental recommendation algorithms designed to deal with streams of positive user feedback. To our best knowledge this is the first ensemble method proposed for incremental recommender systems in the literature.

2 Online bagging

Bagging [1] is an ensemble technique that takes a number of bootstrap samples of a dataset and trains a model on each one of the samples. Predictions from the various sub-models are then aggregated in a final prediction. This is known to improve the performance of algorithms by reducing variance, which is especially useful with unstable algorithms that are very sensitive to small changes in the data. The diversity offered by training several models with slightly different bootstrap samples of the data helps in giving more importance to the main concepts being learned – since they must be present in most bootstrap samples of the data – , and less importance to noise or irrelevant phenomena that may mislead the learning algorithm.

To obtain a bootstrap sample of a dataset with size N, we perform N trials, sampling a random example with replacement from the dataset. Each example has probability of $1/N$ to be sampled at each trial. The resulting dataset will have the same size of the original dataset, however some examples will not be present whereas some others will occur multiple times. To obtain M samples, we simply repeat the process M times.

In its original proposal [1], bagging is a batch procedure requiring $N \times M$ passes through the dataset. However, it has been shown in [8] that this can be done incrementally in a single pass, if the number of examples is very large – a natural assumption when learning from data streams. Looking at the batch method above, we observe that each bootstrap sample contains K occurrences of each example, with $K \in \{0, 1, 2, \ldots\}$, and:

$$P(K = k) = \binom{N}{k} \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{N-k} \tag{1}$$

In an incremental setting, one could just initialize M sub-models – or bootstrap nodes – and then use (1) to train new examples K times, redrawing K for each node. The problem is that this would still require knowing N beforehand. However, if we assume that $N \to \infty$, then the distribution of K tends to a $Poisson(1)$ distribution, and therefore

$$P(K = k) = \frac{e^{-1}}{k!} \tag{2}$$

eliminating the need of any prior knowledge about the data.

2.1 Online recommendation with bagging

We use incremental bagging with ISGD [12], an online matrix factorization algorithm for positive-only data. ISGD (Algorithm 1) uses Stochastic Gradient Descent in one pass through the data, which is convenient for data stream processing. It is designed for positive-only streams of user-item pairs (u, i). Each pair indicates a positive interaction between user u and item i.

Algorithm 1: ISGD - Incremental SGD for positive-only ratings [12]

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : no. of latent features k, no. of iterations $iter$, regularization factor λ, learn rate η
output: user and item factor matrices A and B

for $(u, i) \in D$ **do**
  if $u \notin \texttt{Rows}(A)$ **then**
    $A_u \leftarrow \texttt{Vector}(\texttt{size} : k)$
    $A_u \sim \mathcal{N}(0, 0.1)$
  if $i \notin \texttt{Rows}(B)$ **then**
    $B_i \leftarrow \texttt{Vector}(\texttt{size} : k)$
    $B_i \sim \mathcal{N}(0, 0.1)$
  for $n \leftarrow 1$ **to** $iter$ **do**
    $err_{ui} \leftarrow 1 - A_u \cdot B_i$
    $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
    $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$

By applying the online bagging approach described in Section 2, we obtain Algorithm 2 – BaggedISGD.

BaggedISGD learns a model based on M bootstrap nodes. To perform the actual list of recommendations for a user u, items i are sorted by a function $f = |1 - \hat{R}_{ui}|$ as with ISGD. The scores $\hat{R}_{ui}$ are the average score of all nodes:

$$\hat{R}_{ui} = \frac{\sum_{m=1}^{M} A_u^m \cdot B_i^m}{M} \tag{3}$$

At training time, this algorithm requires at least m times the computational resources needed for ISGD, with m bootstrap nodes. Recommendation also has the overhead of aggregating m predictions from the submodels.

3 Evaluation

To simulate a streaming environment we need datasets that maintain the natural order of the data points, as they were generated. Additionally, we need positive-only data, since the tested algorithm is not designed to deal with ratings. We use

Algorithm 2: BaggedISGD - Bagging version of ISGD (training algorithm)

Data: a finite set or a data stream of user-item pairs $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : no. of latent features k, no. of iterations $iter$, regularization factor λ, learn rate η, no. of bootstrap nodes M
output: M user and item factor matrices A^m and B^m

for $(u,i) \in D$ **do**
  for $m \leftarrow 1$ **to** M **do**
    $k \sim \text{Poisson}(1)$ `// eq. (2)`
    if $k > 0$ **then**
      for $l \leftarrow 1$ **to** k **do**
        if $u \notin \texttt{Rows}(A^m)$ **then**
          $A_u^m \leftarrow \texttt{Vector}(\texttt{size} : k)$
          $A_u^m \sim \mathcal{N}(0, 0.1)$
        if $i \notin \texttt{Rows}(B^m)$ **then**
          $B_i^m \leftarrow \texttt{Vector}(\texttt{size} : k)$
          $B_i^m \sim \mathcal{N}(0, 0.1)$
        for $n \leftarrow 1$ **to** $iter$ **do**
          $err_{ui} \leftarrow 1 - A_u^m \cdot B_i^m$
          $A_u^m \leftarrow A_u^m + \eta(err_{ui} B_i^m - \lambda A_u^m)$
          $B_i^m \leftarrow B_i^m + \eta(err_{ui} A_u^m - \lambda B_i^m)$

4 datasets that conciliate these two requirements – positive-only and naturally ordered –, described in Table 1. ML1M is based on the Movielens-1M movie rating dataset[4]. To obtain the YHM-6KU, we sample 6000 users randomly from the Yahoo! Music dataset[5]. LFM-50U is a subset consisting of a random sample of of 50 users taken from the Last.fm[6] dataset[7]. PLC-STR consists of the music streaming history taken from Palco Principal[8], a portuguese social network for non-mainstream artists and fans. All of the 4 datasets consist of a chronologically ordered sequence of positive user-item interactions. However, ML1M and YHM-50U are obtained from ratings datasets. To use them as positive-only data, we retain the user-item pairs for which the rating is in the top 20% of the rating scale. This means retaining only the rating 5 in ML1M and rating of 80 or more in the YHM-6KU dataset. Naturally, only single accurrences of user-item pairs are available in these datasets, since users do not rate the same item more than once. PLC-STR and LFM-50 have multiple occurrences of the same user-item pairs.

We run a set of experiments using the prequential approach [5] as described in [12]. Each observation in the dataset consists of a simple user-item pair (u, i)

[4] http://www.grouplens.org/data [Jan 2013]
[5] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r [Jan 2013]
[6] http://last.fm/
[7] http://ocelma.net/MusicRecommendationDataset [Jan 2013]
[8] http://www.palcoprincipal.com/

Dataset	Events	Users	Items	Sparsity
PLC-STR	588 851	7 580	30 092	99.74%
LFM-50U	1 121 520	50	159 208	85.91%
YHM-6KU	476 886	6 000	127 448	99.94%
ML1M	226 310	6 014	3 232	98.84%

Table 1. Dataset description

that indicates a positive interaction between user u and item i. The following steps are performed in the prequential evaluation process:

1. If u is a known user, use the current model to recommend a list of items to u, otherwise go to step 3;
2. Score the recommended list given the observed item i;
3. Update the model with (u, i) (optionally);
4. Proceed to – or wait for – the next observation

This process is entirely applicable to algorithms that learn either incrementally or in batch mode. This is the reason why step 3. is annotated as optional. For example, instead of performing this step, the system can store the data to perform batch retraining periodically.

Dataset	M	Rec@1	Rec@5	Rec@10	Rec@20	Upd. (ms)	Rec. (ms)
PLC-STR	ISGD	**0.127**	**0.241**	0.277	0.302	0.237	21.736
	8	0.076	0.194	0.257	0.316	2.563	64.793
	16	0.081	0.215	0.284	0.349	4.732	132.812
	32	0.088	0.229	0.302	0.370	9.508	264.846
	64	0.092	0.237	**0.313**	**0.384**	18.012	517.479
LFM-50U	ISGD	**0.034**	0.049	0.052	0.055	2.625	94.177
	8	0.023	0.044	0.052	0.058	21.449	241.452
	16	0.026	0.050	0.059	0.066	43.094	491.689
	32	0.028	0.055	0.064	0.071	84.536	984.060
	64	0.030	**0.057**	**0.067**	**0.075**	168.781	1.958 s
YHM-6KU	ISGD	**0.030**	**0.063**	0.082	0.103	4.462	89.321
	8	0.011	0.033	0.051	0.076	28.529	347.422
	16	0.012	0.037	0.058	0.086	54.723	667.898
	32	0.019	0.055	0.082	0.117	158.744	990.551
	64	0.021	0.059	**0.087**	**0.123**	328.924	1.934 s
ML1M	ISGD	0.005	0.021	0.034	0.055	0.069	2.557
	8	0.005	0.019	0.033	0.056	0.517	7.208
	16	0.006	0.022	0.038	0.063	1.390	21.816
	32	0.006	0.025	0.042	0.071	1.866	33.496
	64	**0.007**	**0.026**	**0.045**	**0.074**	3.999	41.090

Table 2. Average performance of ISGD with and without bagging. M is the number of bootstrap nodes. The last two columns contain the average update times and the average recommendation times.

To kickstart the evaluation process we use 10% the available data to train a base model in batch, and use the remaining 90% to perform incremental training and evaluation. We do this initial batch training to avoid *cold-start* problems, which are not the subject of our research.

In our setting, the items that users have already co-occurred with – i.e. items that users know – are not recommended. This has one important implication in the prequential evaluation process, specifically on datasets that have multiple occurrences of the same user-item pair. Evaluation at these points is necessarily penalized, since the observed item will be not be within the recommendations. In such cases, we bypass the scoring step, but still use the observation to update the model.

We measure two dimensions on the evaluation process: accuracy and time. In the prequential process described above, we need to make a prediction and evaluate it at every new user-item pair (u, i) that arrives in the data stream. To do this, we use the current model to recommend a list of items to user u. We then score this recommendation list, by matching it to the actually observed item i. We use a recommendation list with at most 20 items, and then score this list as 1 if i is within the recommended items, and 0 otherwise, using Recall@C with cutoffs $C \in \{1, 5, 10, 20\}$. Because only one item is tested against the list, Recall@C can only take the values $\{0, 1\}$. We can calculate the overall Recall@C by averaging the scores at every step. Additionally, we can also depict it using a moving average. Time is measured in milliseconds at every step and we depict it using the same techniques we use with accuracy.

All experiments were run in Intel Haswell 4-core machines, with CentOS Linux 7 64 bit. The algorithms and prequential evaluation code is implemented on top of MyMediaLite [6]. The recommendation step is implemented with multi-core code – predictions from nodes are computed in parallel.

3.1 Results

To evaluate bagging, we experiment with four levels of bootstrapping $M \in \{8, 16, 32, 64\}$. Table 2 summarizes the results of our experiments. Values in Table 2 are obtained by averaging Recall and time obtained at all prequential evaluation steps. With all datasets except YHM-6KU, bagging improves the Recall, especially with $M \geq 32$. One interesting observation is that bagging has a bigger influence on higher Recall cutoffs, which suggests that improvements of the predictive ability are typically not obtained in the top 5 recommended items.

The model update times increase approximately in proportion to the number of bootstrap nodes M, which is not surprising, since the algorithm performs the update operations one time (in average) in each one of the M bootstrap nodes. However, since the baseline update time is small, this overhead is also small. The last column of Table 2 contains the recommendation time, specifically the average time required to produce a recommendation list. This is important, because the bagging algorithm needs to aggregate predictions coming from all M nodes, which is obviously an overhead. Results show that both the update times

and recommendation times increase proportionally to M. However, the recommendation step is a far more costly operation, even when computed in parallel. For example, using $M = 64$ with LFM-50U and YHM-6KU, recommendations are computed in nearly two seconds in average, in 4-core machines.

A useful feature of prequential evaluation is that it allows us also to depict the evolution of Recall@20[9] in Figure 1. This visualization reveals how the predictive ability of the algorithm performs over time, as the incremental learning process occurs.

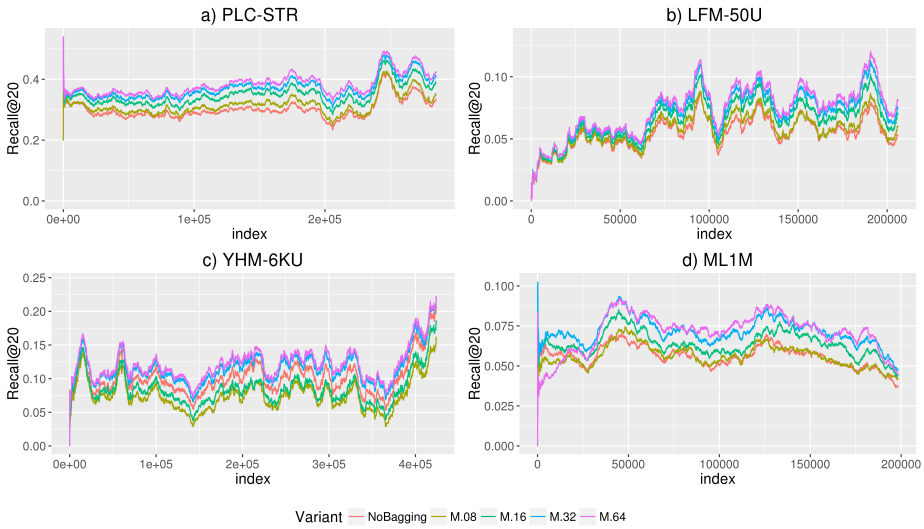

Fig. 1. Prequential evaluation of Recall@20 with ISGD with and without bagging. Lines are drawn using a moving average of Recall@20 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

3.2 Discussion

Results in Table 2 and Figure 1 show that bagging clearly improves the accuracy of ISGD, with accuracy improvements of 20% over the baseline (see Table 2 LFM-50U and ML1M). This improvement is mainly observable with cutoffs $C \geq 5$ of Recall. Given that bagging reduces variance [1], this suggests that the variance of ISGD is lower in the top few recommendations. Another observation is that improvements are not consistent with all datasets. With LFM-50U, for example, bagging only slightly outperforms the baseline ISGD – and only with $M \geq 32$ –, while with PLC-STR, the improvement is much higher in proportion, even with lower M.

[9] For the sake of space, we omit other cutoffs.

It is also clear that the time overheads grow linearly with the number of bootstrap models. However, the overhead in model update times is not very relevant in practice, given that the baseline update times are very low in ISGD – with $M = 64$ the highest update time falls below 400ms. The overhead at recommendation time is more evident, when aggregating results from the M bootstrap nodes. Fortunately, as with most ensemble techniques, parallel processing can be trivially used to alleviate this overhead. Additionally, there may be room for code optimization or approximate methods that require less and/or more efficient computations.

4 Conclusions

Bagging is a an ensemble technique successfully used with many machine learning algorithms, however it has not been thoroughly studied in recommendation problems, and particularly with incremental algorithms. In this paper, we experiment online bagging with an incremental matrix factorization algorithm that learns from unbounded streams of positive-only data. Our results suggest that with manageable overheads, accuracy clearly improves – more than 20% in some cases –, especially as the number of recommended items increases. In the near future, we intend to experiment this and other online ensemble methods in a larger number of stream-based recommendation algorithms.

5 Acknowledgments

Project "TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020" is financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF). This work is partially funded by the European Commission through project MAESTRA (Grant no. ICT-2013-612944). We thank Ubbin Labs, Lda. for kindly providing data from Palco Principal.

References

1. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996), http://dx.doi.org/10.1007/BF00058655
2. Chowdhury, N., Cai, X., Luo, C.: Boostmf: Boosted matrix factorisation for collaborative ranking. In: Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2015, Part II. LNCS, vol. 9285, pp. 3–18. Springer (2015), http://dx.doi.org/10.1007/978-3-319-23525-7_1
3. Domingos, P., Hulten, G.: Catching up with the data: Research issues in mining data streams. In: DMKD (2001), http://www.cs.cornell.edu/johannes/papers/dmkd2001-papers/p8_domingos.pdf
4. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Proc. of the 13th Intl. Conference on Machine Learning ICML '96. pp. 148–156. Morgan Kaufmann (1996)

5. Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Machine Learning 90(3), 317–346 (2013), http://dx.doi.org/10.1007/s10994-012-5320-9
6. Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Mymedialite: a free recommender system library. In: Proc. of the 2011 ACM Conference on Recommender Systems, RecSys 2011. pp. 305–308. ACM (2011)
7. Jahrer, M., Töscher, A., Legenstein, R.A.: Combining predictions for accurate recommender systems. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2010. pp. 693–702. ACM (2010), http://doi.acm.org/10.1145/1835804.1835893
8. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: Proc. of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD 2001. pp. 359–364. ACM (2001), http://portal.acm.org/citation.cfm?id=502512.502565
9. Schclar, A., Tsikinovsky, A., Rokach, L., Meisels, A., Antwarg, L.: Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In: Proc. of the 2009 ACM Conference on Recommender Systems, RecSys 2009. pp. 261–264. ACM (2009), http://doi.acm.org/10.1145/1639714.1639763
10. Segrera, S., Moreno, M.N.: An experimental comparative study of web mining methods for recommender systems. In: Proc. of the 6th WSEAS Intl. Conf. on Distance Learning and Web Engineering. pp. 56–61. WSEAS (2006)
11. Sill, J., Takács, G., Mackey, L.W., Lin, D.: Feature-weighted linear stacking. CoRR abs/0911.0460 (2009), http://arxiv.org/abs/0911.0460
12. Vinagre, J., Jorge, A.M., Gama, J.: Fast incremental matrix factorization for recommendation with positive-only feedback. In: Proc. of the 22nd Intl. Conference on User Modeling, Adaptation, and Personalization, UMAP 2014. LNCS, vol. 8538, pp. 459–470. Springer (2014), http://dx.doi.org/10.1007/978-3-319-08786-3_41
13. Wolpert, D.H.: Stacked generalization. Neural Networks 5(2), 241–259 (1992), http://dx.doi.org/10.1016/S0893-6080(05)80023-1